Computer Organization

COMP2120

Qi Zhao

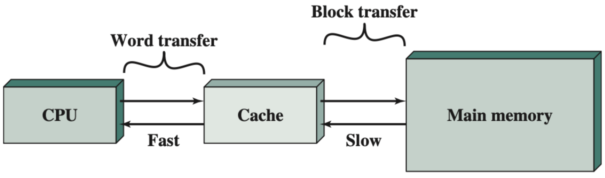
February 25, 2024

Cache Memory Part I

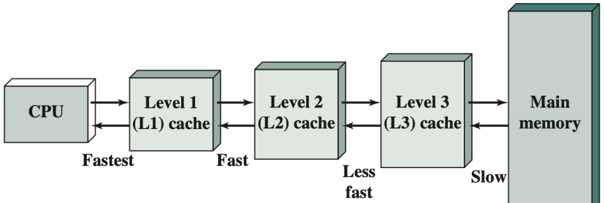
Cache Memory

- Cycle time of CPU: $0.25ns$ (4GHz), even clock on the motherboard $< 10ns$ ($> 100MHz$)
- Cycle time of Primary Memory: $50 - 60ns$ (memory access time)
- To bridge the gap, add extra layer(s) in between — cache Static RAM is used, which is faster than Dynamic RAM.
- Access time is about $10ns$ for cache on mother board, and can be $< 1ns$ if on-chip (inside the CPU chip)
- Cache Memory is transparent to (hidden from) the program (i.e. the user)

Cache and Main Memory



(a) Single cache



(b) Three-level cache organization

Figure 4.3 Cache and Main Memory

Cache Organization

- Divide the memory and cache into equal-sized blocks. A **block** in cache is also called a cache **line**.

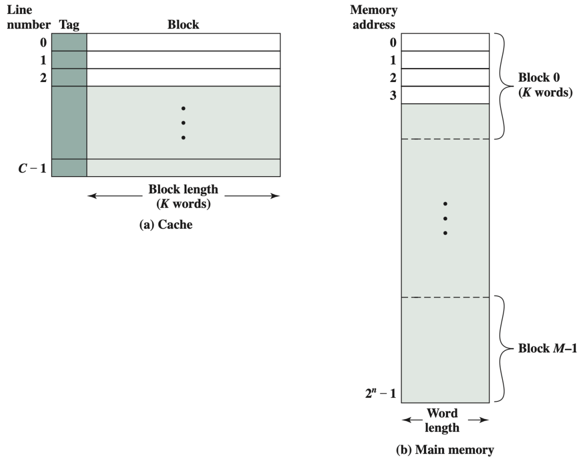
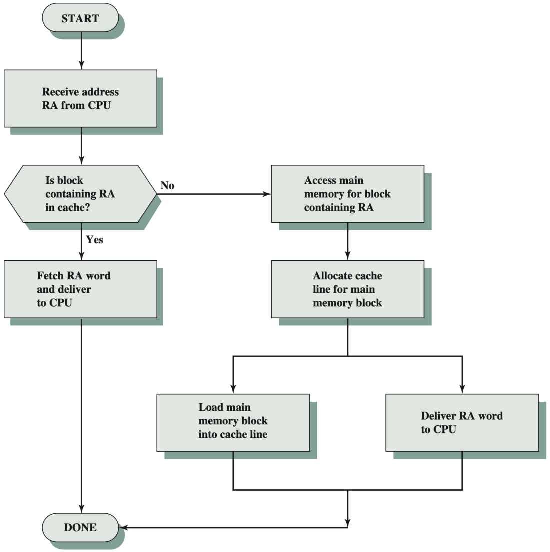


Figure 4.4 Cache/Main Memory Structure

Cache Organization

- When a memory block is referenced, the entire block is moved into the cache.
- The next time that block is referenced, it will be retrieved from the cache, instead of from the main memory again.
- The cache is addressed by the block number of the main memory, not by the address of the cache.
- When the data is not in the cache, the entire block will be brought into the cache. If the cache is full, then a block is selected to move out of the cache to provide space for the incoming block.
- The principle of Multi-level cache is the same as single-level cache.

Cache Read Operation


START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

- RA: read address
- Address: Which word?
- Address: Which block?

Elements of Cache Design- Cache size

- large cache tends to be slower (generally true for memory)
- large cache, more data cache, less cache misses.
- More expensive
- Actually, principle of diminishing return holds, i.e. The gain for more cache diminishes, after optimal size of cache reached.
- Typical values,

L1 cache	L2 cache	L3 cache
$64KB - 2MB$	$256KB - 8MB$	$4MB - 64MB$

Address Mapping

- How to desgin the address format?
- We need, block number, word/Byte number, cache line number

block no.	Offset(word/Byte id)

- The CPU gives out a 32-bit address `Addr`. Each byte has an address.
- The block size is b, one block b bytes.
- Then `Addr` is in Block `Addr`/b, and it is the (`Addr`%b)-th byte in the block.
- Modulo is a math operation that finds the remainder, **mod**, or symbol %.
- If the divisor (block size) is in the form of $b = 2^n$, then
  - A%b is given by the rightmost n-bits of A,
  - A/b is given by the remaining bits.

Address Mapping

- The CPU gives an address to the memory.
- The rightmost bits (offsets in block) represents the position of the Byte /word within the block.
- The leftmost bits (block id + cache line) represents the block number in main memory.
- Where to allocate one block (cache line) in the cache memory?

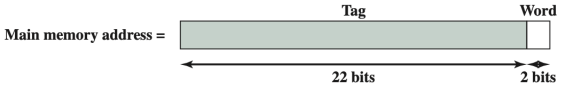
$$\text{Mapping} : \text{Block number} \longrightarrow \text{Tag}, \text{Line number}$$

- This can be a many-to-one mapping. (many memory block maps to the same cache line)

Fully associative

A straightforward way:

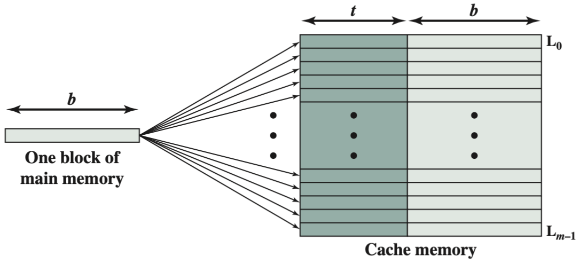
$$\text{Mapping} : \text{Block number} = \text{Tag}$$



- No cache line number; only have tag and offset parts
- Address length: 24
- Number of blocks in Main memory address: 2^{22}
- Number of addressable units 2^{24} bytes or words
- Block size: 2^{2} bytes or words

Fully associative

each main memory block can be loaded into any line of the cache.



(b) Associative mapping

Fully associative

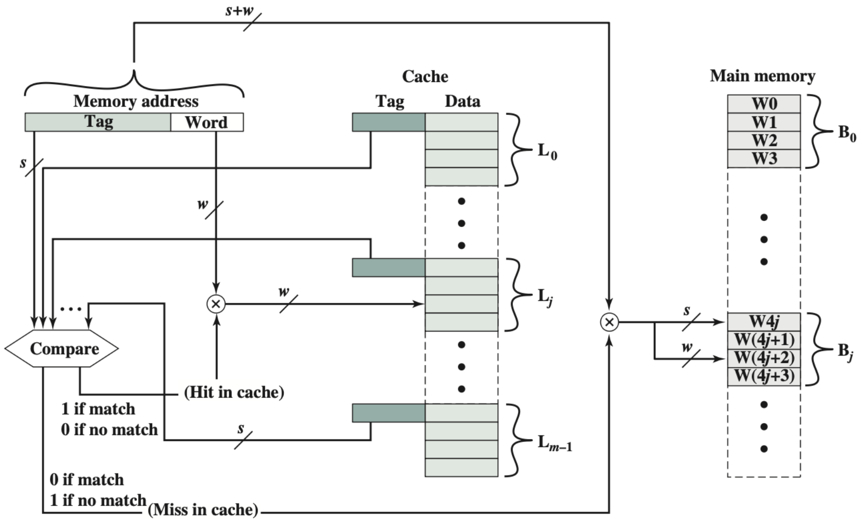


Figure 4.11 Fully Associative Cache Organization

Fully associative

- **Advantage**: it is flexible to choose which block to replace when a new block is read into the cache. Higher hit ratio
- **Disadvantage**: must simultaneously examine every line's tag for a match.

Direct-Map Organization

- Map each block of main memory into **only one** possible line
- Number of lines of in the cache: m, $m = 2^r$
- Main memory block number: j
- Cache line number:

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

Cache line	**Main memory blocks assigned**
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m+1, 2m+1, \ldots, 2^s - m + 1$
$\vdots$	$\vdots$
$m - 1$	$m-1, 2m-1, 3m-1, \ldots, 2^s - 1$

Direct-Map Organization

- Address

Tag	Line number	Offset
$s-r$ bits	cache line number: r bits	word/Byte id: w bits

- Example:

Tag	Line number	Offset (Byte)
5 bits	7 bits	4 bits

- 2^4 bytes in a block
- $2^7 = 128$ Cache lines
- $2^{7+5} = 4096$ main memory blocks

Direct-Map Organization

- CPU is looking for the address [A7B4], MAR $= 1010\ 0111\ 1011\ 0100$
- Example:

tag: 10100	Line number: 1111011	Offset (Byte): 0100
5 bits	7 bits	4 bits

- Go to cache line 1111011, see if the tag is 10100
- If YES, cache hit!
- Read the corresponding Byte
- Otherwise, get the targeted block into cache line 1111011

Direct-Map Organization

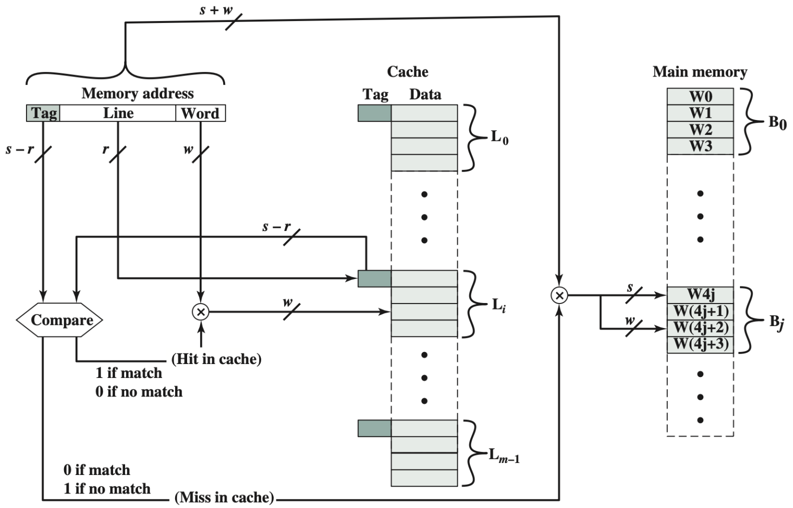
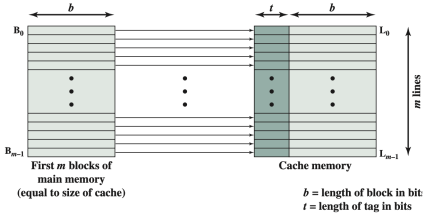


Figure 4.9 Direct-Mapping Cache Organization

Direct-Map Organization

- **Advantage**: only need to check one cache line, simple, faster, inexpensive
- so there is no need to perform selection (which require some logic to perform, and logic gates need time to perform its task).
- note that cache memory is fast, even a slight improvement is significant)
- **Disadvantage**:


b
t
b
B_0
B_{m-1}
L_0
L_{m-1}
m lines
First m blocks of main memory (equal to size of cache)
Cache memory
b = length of block in bit
t = length of tag in bits

(a) Direct mapping

- Multiple memory blocks are mapped to the same cache line, and these blocks may be needed by the running program at the same time, e.g. program code and data needed by the program,
- causing a lot of cache misses. The chance of this occurring is small, but it may occurs.

k-way set associative

- Direct-Map, fully associative, we need some compromised methods that exhibit the strength of direct-map and fully associative methods.
- Cache consists of a number of sets, each one a few lines.
- Each cache set contains k cache lines. The most common value is 2, and maximum value is 8.
- Direct-map: 1-way set associative, each set has one line.
- Store each block of main memory into one certain Cache set with **k** possible line.
- Cache set number i

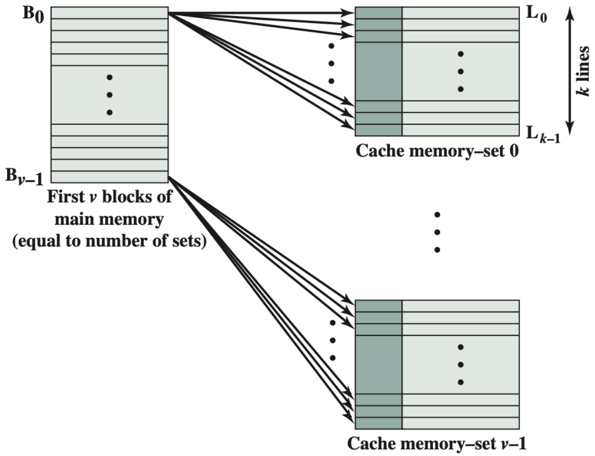
$$m = v \times k$$

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

where

- Number of lines of in the cache: m
- Number of lines of sets: v,
- Number of lines in each set: k
- Main memory block number: j

k-way set associative



(a) v associative–mapped caches

k-way set associative

- The block size is b.
- Then `Addr` is in Block `Addr`/b, and it is the (`Addr`%b)-th byte in the block.
- If these blocks are divided into v sets, then block B will be put into set $B\%v$
- If the divisor is in the form of $v = 2^n$, then
  - A%v is given by the rightmost n-bits of block no part, cache set number
  - A/v is given by the remaining bits.
- Address:

Tag	set number	Offset
	Cache set number	word/Byte id

$\underbrace{\qquad\qquad\qquad}_{\text{block no. in main memory}}$

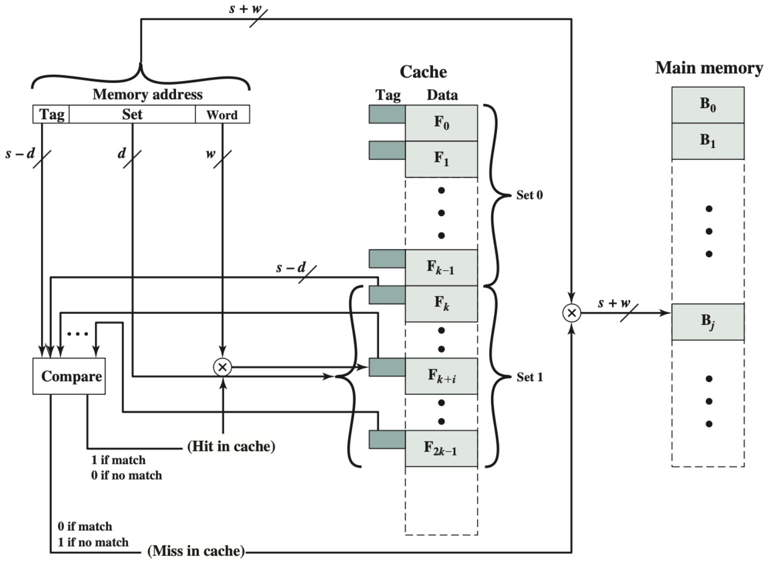
k-way set associative

- CPU is looking for [A7B4] MAR = 1010011110110100
- Example:

tag: 10100	set number: 1111011	Offset (Byte): 0100
5 bits	7 bits	4 bits

- 2-way set associative, 2 lines in each set
- Go to cache set 1111011, check if the tag is 10100
- If YES, cache hit!
- Read the corresponding Byte
- Otherwise, get the targeted block into one line in cache set 1111011

k-way set associative



k-way set associative

- Fewer misses, because we can avoid the situation mentioned above (in direct-map organization).
- Need selection logic to choose the matched one, so it is slightly slower that direct-map cache.
- Empirical studies shows that increase the number of lines in a set do not improve the performance very much. At most use 8-way, and 2-way is the most common organization.

Analogy on Cache Memory

- Assume a university with $100,000$ students (memory size). The id (address of memory) of the student is from 00000 to 99999.
- Their information is stored in the central storeroom (main memory) which takes a longer time to access.
- The records of the students are stored in boxes (blocks). Each box contains 10 records. Then
  - No. of Box = $100,000/10 = 10,000$
  - You want to find someone's record, with an ID, say 32767,
  - Box No. = int $(32767/10) = 3276$
  - it is $(32767\%10)$-th record within the box, ie. 7th record.
- Note that because of the box size = 10, we can take the first 4 digits as the box no., and the last digit as the order of the record within the box.

Analogy on Cache Memory

Assume that in the president's office (cache memory), which can contain $1,000$ records, ie. 100 boxes. When 3276-th boxes are moved from central storeroom (main memory) to the local office (cache memory), we need to:

- put the 10 records into one of the 100 boxes, and stick a label of 3276 in front to identify the record.
- When you want to find another student's record, with an id 32768, we need to
  - get the box number (block no.) by $\text{int}(32768/10) = 3276$.
  - check whether the 3276-th box is in the Head's office.
  - If yes, get the 32768-th record as the 8-th record in this box.
  - If not, get the 3276-th record from main memory and put it in Head's office.

Analogy on Cache Memory

- The above scheme (full associative), is inefficient, as for every access at the local office, we need to go through all 100 boxes to check whether the 3276-th box is there.
- To make it more efficient, we divide the students into different faculties.
- We have 10 faculties
- The n-th box (every ten students) will be assigned to (n%number of sets)-th faculty. For example, 3276 will be assigned to the 6th faculty. Note that each block will be assigned to exactly one set.
- In the Head's office, each house contains 10 boxes (10-way set associative). Then there will be 10 sets, each with 10 boxes.
- We need only to go through all boxes for 6th faculty (10 boxes), instead of checking all boxes in the office to see if the 3276-th boxes is already there.

Binary Version

- We have 2^{32} bytes, address A is 32-bit.
- if block size is $128 = 2^7$ bytes,

$$\mathrm{B\ (block\ no.)} = \mathrm{A}/128$$

  which is the left most 25-bit, and the $A\%128$=the rightmost 7-bits, R.
- Set number, which is the block address

$$\mathrm{S\ (set\ no.)} = \mathrm{B\%(number\ of\ set)}.$$

- If we have 64KB cache, and each set = 2 block (2-way set associative), then we have $64 \times 1024 = 2^{16}$ Bytes,

$$\mathrm{no.\ of\ set} = (2^{16}/128)/2 = 256 = 2^8$$

- S = rightmost 8-bit of the 25bits.

tag	S	R
17 bits	8 bits	7 bits

- We need to compare for only 2-blocks (because each set contain only 2 blocks If it is in one of the 2 blocks, then cache hit. Otherwise cache miss.