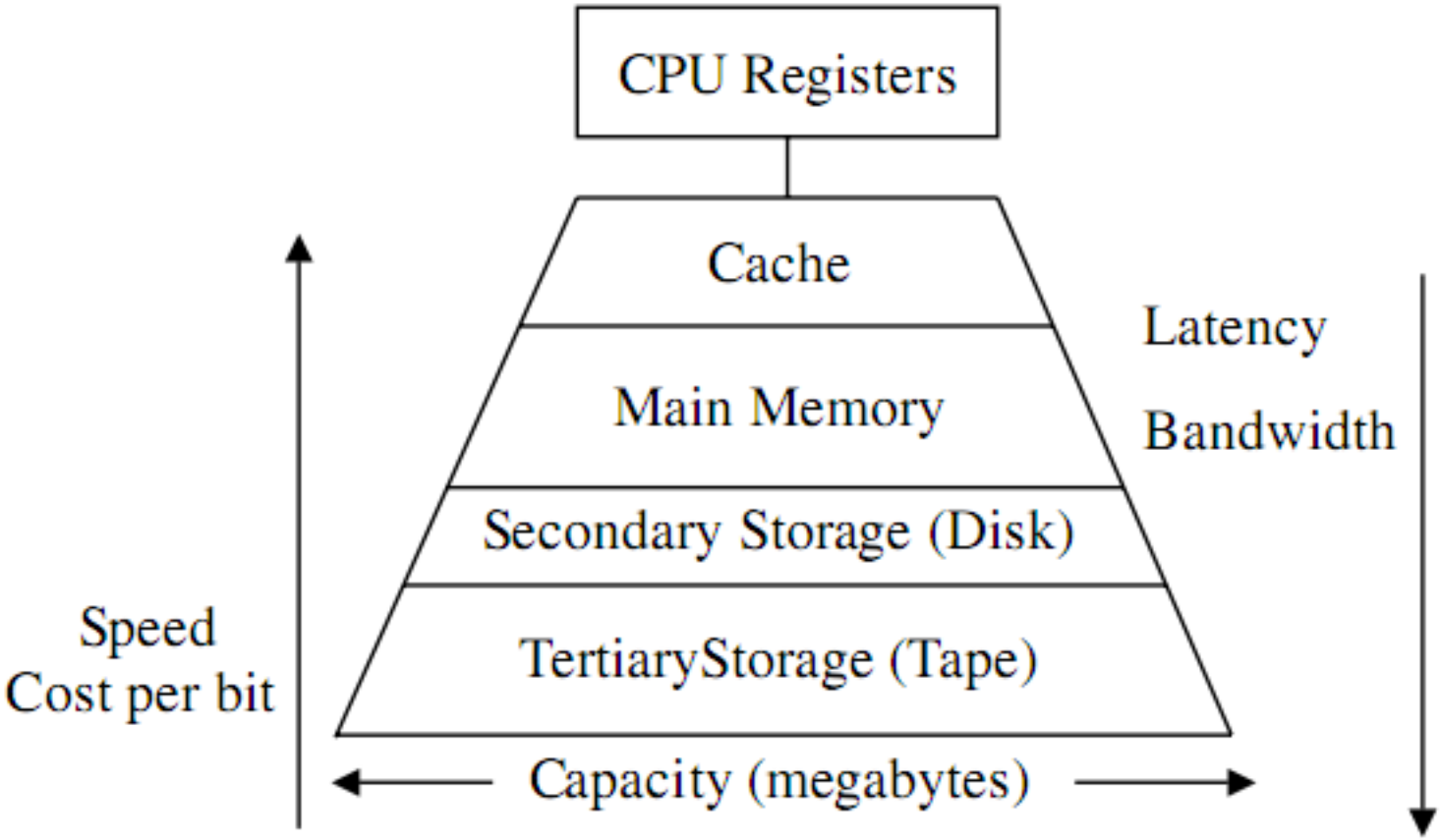
Computer Architecture
5. The Memory System

Lecturer: A.Prof.Dr. Hoàng Xuân Dậu
Email: dauhx@ptit.edu.vn
Faculty of Information Security
Posts & Telecommunications Institute of Technology

Main topics

- Introduction to memory system and the memory hierarchical model
- ROM
- RAM
- Cache memory
  - Introduction to cache
  - Cache architecture and organization
  - Cache performance and measures to improve cache performance.

The Memory System



Memory Hierarchy Parameters

	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU registers	Random	64–1024 bytes	1–10 ns	System clock rate	High
Cache memory	Random	8–512 KB	15–20 ns	10–20 MB/s	$500
Main memory	Random	16–512 MB	30–50 ns	1–2 MB/s	$20–50
Disk memory	Direct	1–20 GB	10–30 ms	1–2 MB/s	$0.25
Tape memory	Sequential	1–20 TB	30–10,000 ms	1–2 MB/s	$0.025

Memory Hierarchy Elements

- CPU registers:
  - Very small in size (few dozen bytes to few KB)
  - Very fast in speed (runs at CPU clock), access time is about 0.25 ns.
  - Very expensive
  - Store temporary input data and output for instructions
- Cache:
  - Small in size (64KB to 16MB)
  - Fast in speed; access time is about 1-5 ns
  - Expensive
  - Also called “smart memory”
  - Stores data and instructions for CPU

Memory Hierarchy Elements

- Main memory:
  - Big in size, capacity from 256MB to 4GB for 32 bit systems.
  - Slow in speed, access time is about 50-70 ns
  - Stores data and instructions for system and users
  - Pretty cheap
- Secondary memory:
  - Very big in size, capacity from 20GB to 1000GB
  - Very slow in speed, access time is about 5ms
  - Stores large amount of data in files in long time
  - Very cheap

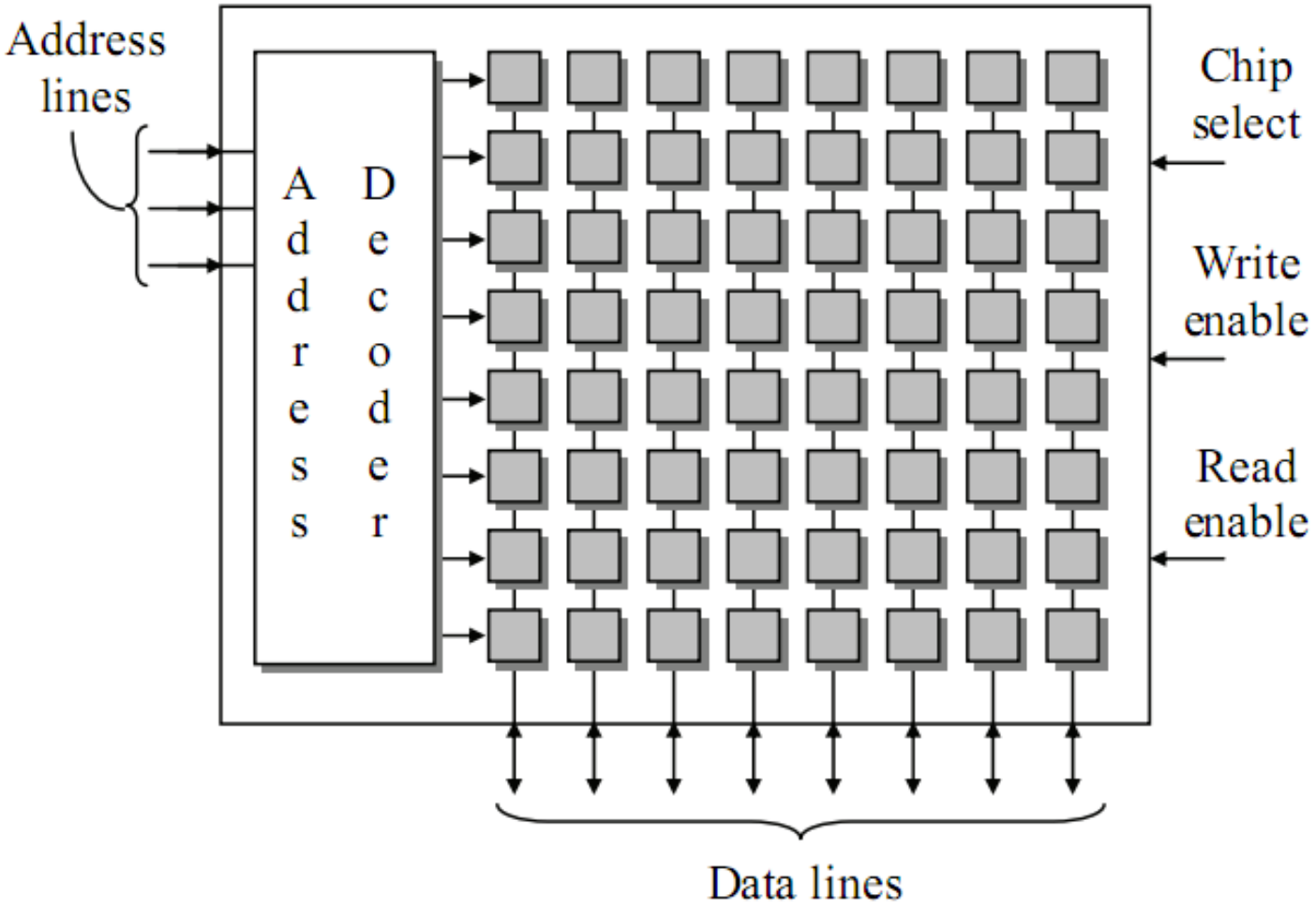
Memory Hierarchy Roles

- Improve the system performance
  - Balance the fast speed of CPU and slow speed of main memory and secondary memory
  - The average access time of the memory system is near cache's access time
- Reduce the manufacturing costs
  - More expensive elements are used in smaller capacity
  - Less expensive elements are used in larger capacity

Memory Classifications

- Based on access types:
  - Random Access Memory (RAM)
  - Serial Access Memory (SAM)
  - Read Only Memory (ROM)
- Based on information sustainability:
  - Volatile memory: stored information is lost when power is off.
  - Non-volatile memory: stored information is retained when power is off.
- Based on manufacturing technologies:
  - Semiconductor memory: ROM, RAM, SSD
  - Magnetic memory: HDD, FDD, tape
  - Optical memory: CD, DVD

Organization of memory device



Organization of memory device

- Address lines: connect to A bus; transmit address of memory line from CPU.
- Address decoder: Uses address to select the correct memory line: make it active.
- Data lines: connect to D bus; transmit data from/to CPU.
- Chip select (CS): the memory chip is activated when CS is 0. CPU works with only one memory chip at a time.
- Write enable (WE): write to memory line is enable when WE is 0.
- Read enable (RE): read from memory line is enable when RE is 0.

Introduction to ROM

- ROM is Read Only Memory, where information can only be read from. Write to ROM is only possible using special equipment or special methods.
- ROM is non-volatile memory: all stored information are maintained when power is off.
- ROM is semiconductor memory. Each memory cell is a semiconductor gate.
- ROM is usually used to stored system information.
  - System information: hardware and BIOS information

ROM - Examples

Types of ROM

- Ordinary ROM: oldest ROM which we have to use violet beam to write information.
- PROM: Programmable ROM. PROM can be written to or programmed via a special device called a PROM programmer.
- EPROM: Erasable programmable read-only memory. It can be erased by exposure to strong ultraviolet light.
- EEPROM: It is EPROM, but its content can be erased electrically.
- Flash memory:
  - A new type of EEPROM that can be erased and written faster than ordinary EEPROM.
  - Can be erased and written in large blocks.

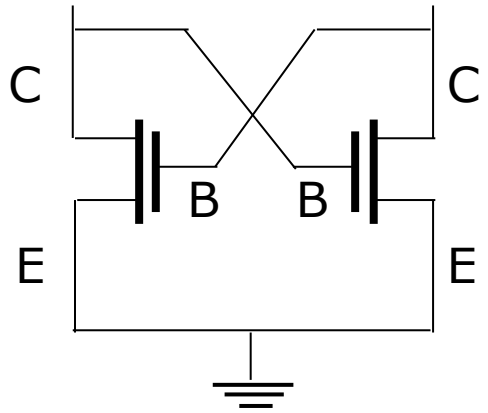
Introduction to RAM

- RAM is Random Access Memory, where each memory cell is accessed randomly or in any order.
- RAM is volatile memory: all stored information are lost when power is off.
- RAM is semiconductor memory. Each memory cell is a semiconductor gate.
- RAM is used to stored system and user information.
  - System information: hardware and OS information
  - User information: user application data and code.

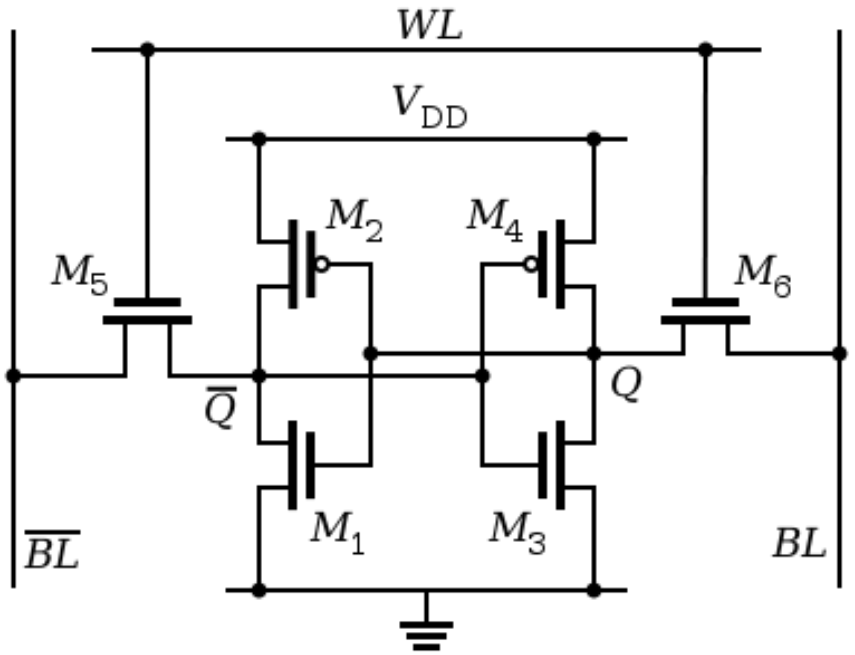
Types of RAMs

- There are two major types of RAMs:
  - Static RAM (SRAM)
    - Each bit if SRAM is based on a flip-flop
    - "Static" means information stored in SRAM bit is stable and it doesn't need to be refreshed periodically.
    - SRAM is fast but more expensive than DRAM
  - Dynamic RAM (DRAM)
    - Each bit if DRAM is based on a capacitor
    - "Dynamic" means information stored in DRAM bit is not stable and it needs to be refreshed periodically. Why information stored in DRAM bit is not stable?
    - DRAM is slower than SRAM but cheaper than SRAM

SRAM's Basic Elements



A simple flip-flop or cross-coupled transistors

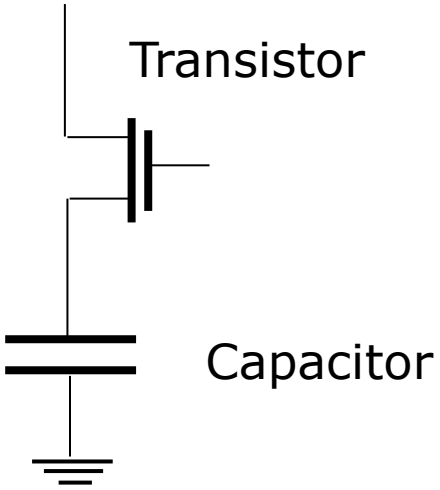


SRAM cell-6T

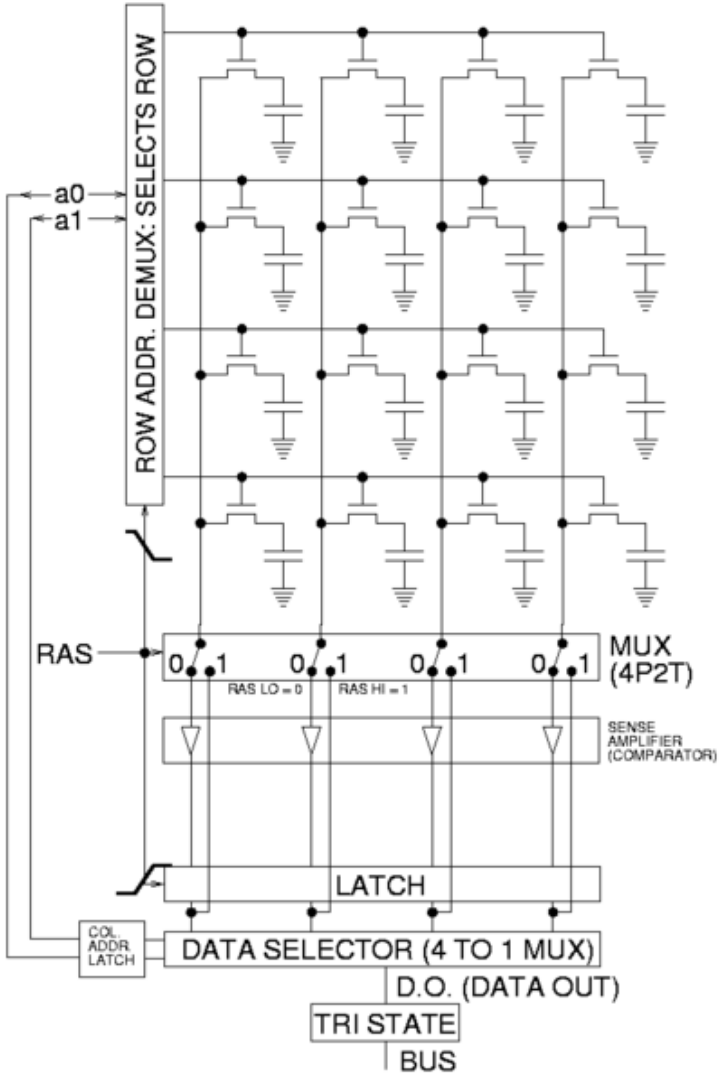
SRAM Characteristics

- SRAM uses *bistable latching circuitry* to store each bit
- A bit SRAM's flip-flop usually uses 6, 8, 10 transistors (also called 6T, 8T and 10T).
- SRAM is faster because:
  - Its bits have the symmetric structure
  - SRAM chips accept all address bits at a time
- SRAM is more expensive because:
  - It uses more transistors for a bit than DRAM
  - Due to the internal structure is more complex, SRAM's density is less than DRAM.

DRAM's Basic Elements



One DRAM bit



DRAM Characteristics

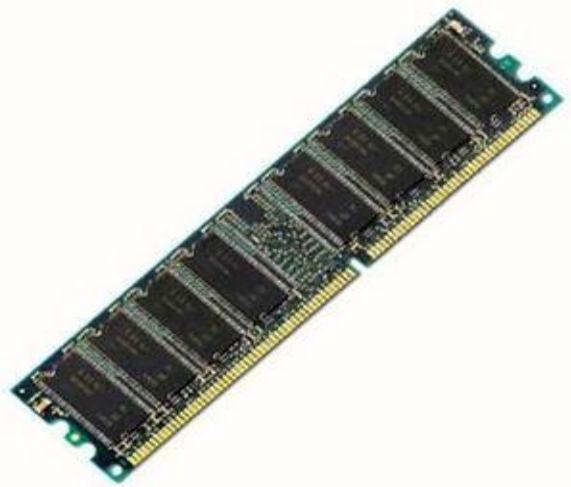
- DRAM's bit is based on a capacitor and a transistor. 2 levels of capacitor electric charge represent 2 logic values: 0-empty and 1-full.
- By nature, electric charge in capacitor is leaking, it must be re-charged periodically to maintain the information.
- DRAM is usually arranged in a square array of one capacitor and transistor per cell.
- DRAM is slower than SRAM as it needs periodical refresh and the capacitor charge takes time.
- DRAM is cheaper than SRAM as it uses less transistors per cell and hence its density is higher.

Types of DRAMs

- SDRAM: Synchronous DRAM
- SRD SDRAM: Single Data Rate SDRAM accepts one command and transfers one word of data per clock cycle; 100MHz, 133MHz
- DDR SDRAM: Double Data Rate SDRAM
  - DDR1 SDRAM: 266, 333, 400 MHz (2 words/clk cycle)
  - DDR2 SDRAM: 400, 533, 800 MHz (4 words/clk cycle)
  - DDR3 SDRAM: 800, 1066, 1333, 1600 MHz (8 words/clk cycle)
  - DDR4 SDRAM: 1600, 2400, 2666, 3200 MHz (16 words/clk cycle).

Types of DRAMs

SDRAM PC133



DDR3 1066 SDRAM

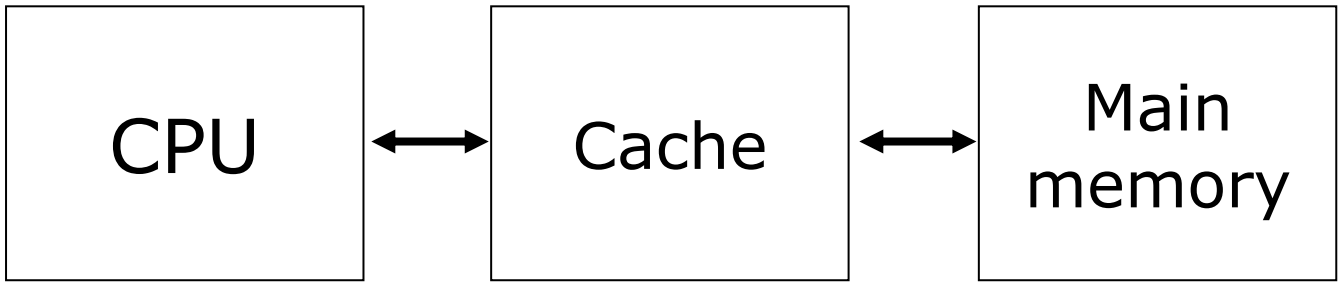
DDR4 2666 16GB

Cache Memory

- What is cache?
- Roles of cache
- Basic principles of cache operation
- Cache architectures
- Cache organization
- Read/write in cache
- Cache replacement policies
- Method to improve cache performance

What is cache?

- Cache is a memory element in the computer memory hierarchy.
  - It acts as the intermediary between CPU and the main memory;
- Cache may reside outside CPU (old systems) or reside inside CPU (modern systems);



What is cache?

- Cache usually has small size:
  - 16K, 32K, 128K, 256K, 512K for old systems, and
  - 1MB, 2MB, 3MB, 4MB,... for new systems;
- Cache is much faster than main memory;
- Cache cost per bit is much more expensive than that of main memory.
- In modern CPUs, cache is usually divided into levels:
  - 2 level caches (L1 and L2) in single or dual core CPUs;
  - 3 or more level caches for CPUs with more than 2 cores.

Roles of cache

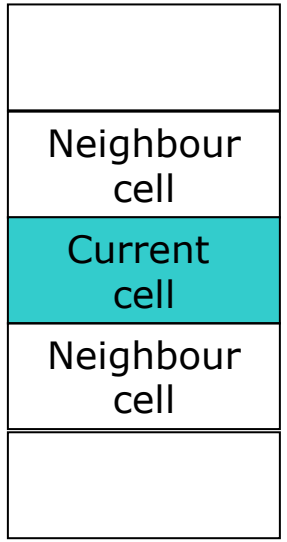
- Improve the system performance
  - Balance the fast speed of CPU and slow speed of main memory (reduce the number of CPU accesses directly to main memory).
  - The average access time of the memory system is near cache's access time
- Reduce the manufacturing costs
  - If two systems have the same performance, the system with cache is cheaper;
  - If two systems have the same cost, the system with cache is faster.

Cache Basic Principles

- Cache is considered a “smart” memory because:
  - Cache can predict the CPU’s need of instruction and data;
  - Necessary instruction and data are fetched from memory to cache in advance, so CPU only needs to get them from cache without accessing memory. This reduces the memory access time.
- Cache works based on 2 basic principles:
  - Spatial locality
  - Temporal locality.

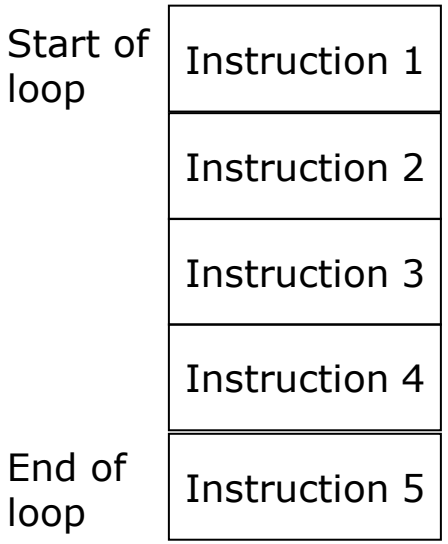
Cache Basic Principles (cont.)

- Spatial locality:
  - If a memory location is accessed, the probability that its closed neighbours are accessed in near future is high.
  - Applied to data items and instructions which have highly serial order in program space.
  - Most of instructions in a program are in serial order, so cache reads a block of data from main memory, which covers all neighbouring items of the currently accessed item.



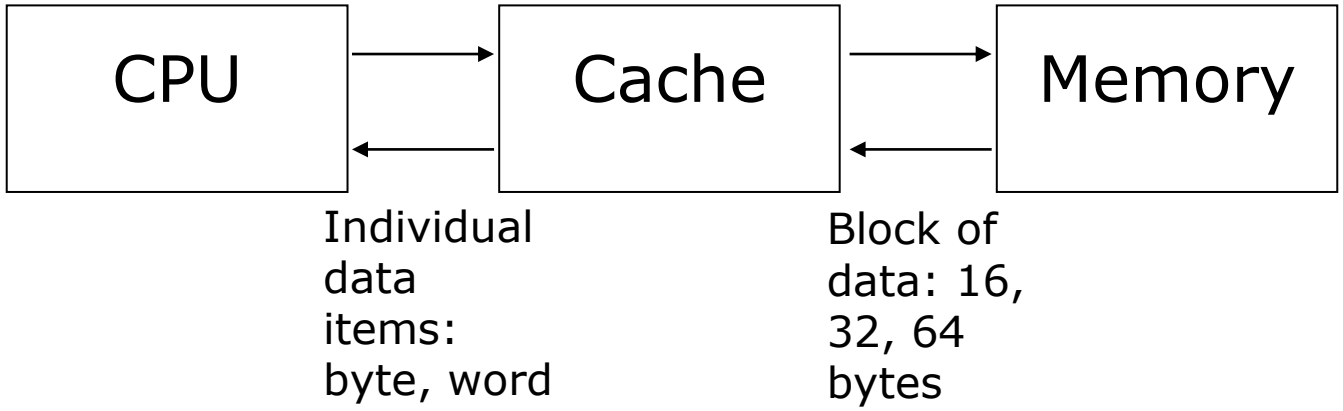
Cache Basic Principles (cont.)

- Temporal locality:
  - If a memory location is accessed, the probability that it is accessed again in near future is high.
  - Applied to data items and instructions in a loop.
  - Cache reads a block of data from main memory, which covers all memory items of the loop.



Exchanging Data

- CPU reads/writes individual data items from/to cache;
  - Why?
- Cache reads/writes large block of data from/to main memory;
  - Why?



Cache Hit and Miss Ratios

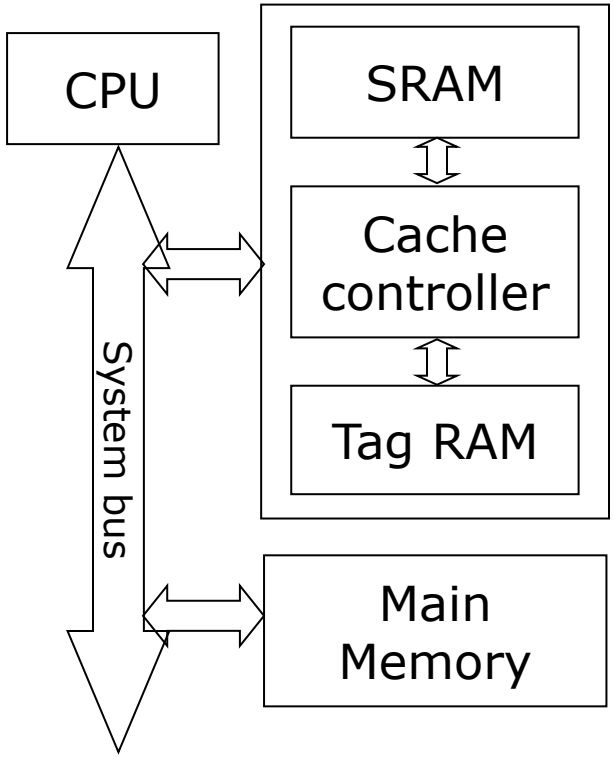
- *Hit* is an event in which CPU accesses a data item that is found in cache.
  - The probability of a hit is called hit ratio, or *H*.
  - 0 <= H <= 1
  - The higher H the better cache.
- *Miss* is an event in which CPU accesses a data item that is not found in cache.
  - The probability of a miss is called miss ratio, or 1- *H*.
  - 0 <= (1 – H) <= 1
  - It is desire to make miss ratio low.

Cache Architecture – Look Aside

- Cache and memory are connected to system bus
- Cache and memory "see" the CPU bus cycle at the same time
- Pros:
  - Simple design
  - Miss is fast (why?)
- Cons:
  - Hit is slow (why?)

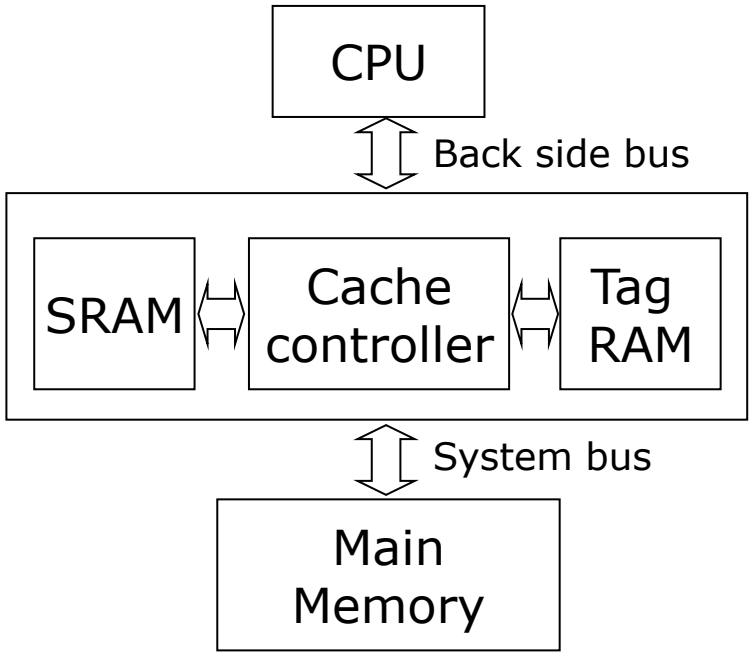
SRAM: RAM to store cache data

Tag RAM: RAM to store memory address



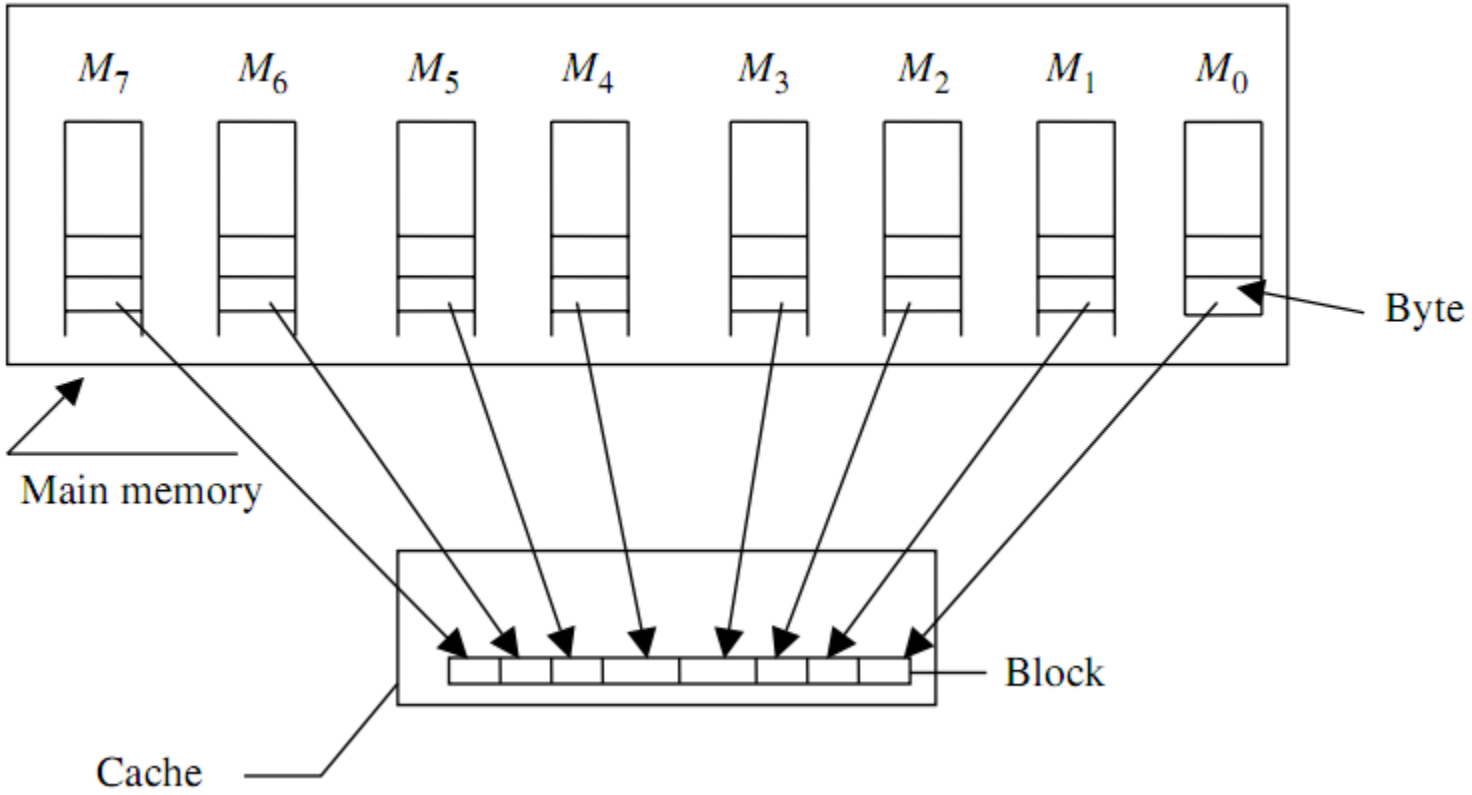
Cache Architecture – Look Through

- Cache is placed between CPU and memory
- Cache "sees" the CPU bus cycle first then it passes CPU bus cycle to memory
- Pros:
  - Hit is fast (why?)
- Cons:
  - Complicated design
  - Expensive
  - Miss is slow (why?)



Cache Organization

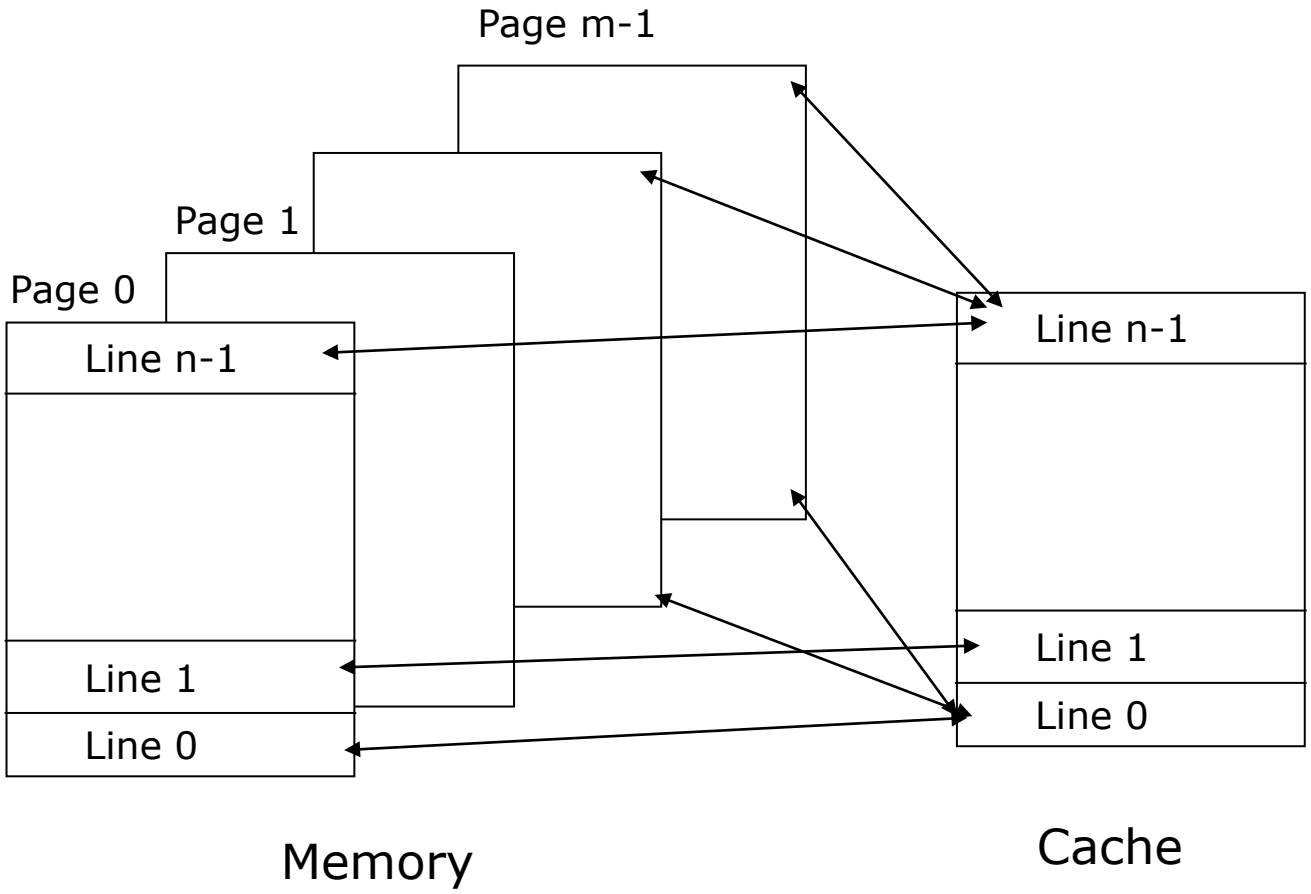
- Cache organization is that how cache and memory are working together?



Cache Organization Techniques

- Direct mapping
  - Simple, fast and fixed mapping
- Fully associative mapping
  - Complicated, slow and flexible mapping
- Set associative mapping
  - Complicated, fast and flexible mapping

Direct mapping



Direct mapping

- Cache:
  - Divided into *n* blocks or lines, from $Line_0$ to $Line_{n-1}$
- Memory:
  - Divided into *m* pages, from $page_0$ to $page_{m-1}$.
  - Page has same size as cache
  - Each page has *n* lines, from $Line_0$ to $Line_{n-1}$
- Mapping:
  - $Line_0$ of ($page_0$ to $page_{m-1}$) is mapped to $Line_0$ of cache;
  - $Line_1$ of ($page_0$ to $page_{m-1}$) is mapped to $Line_1$ of cache;
  - ....
  - $Line_{n-1}$ of ($page_0$ to $page_{m-1}$) is mapped to $Line_{n-1}$ of cache;

Direct mapping address

- *Tag* (bit) is the address of page in memory
- *Line* (bit) is the address of line in cache
- *Word* (bit) is the address of word in line

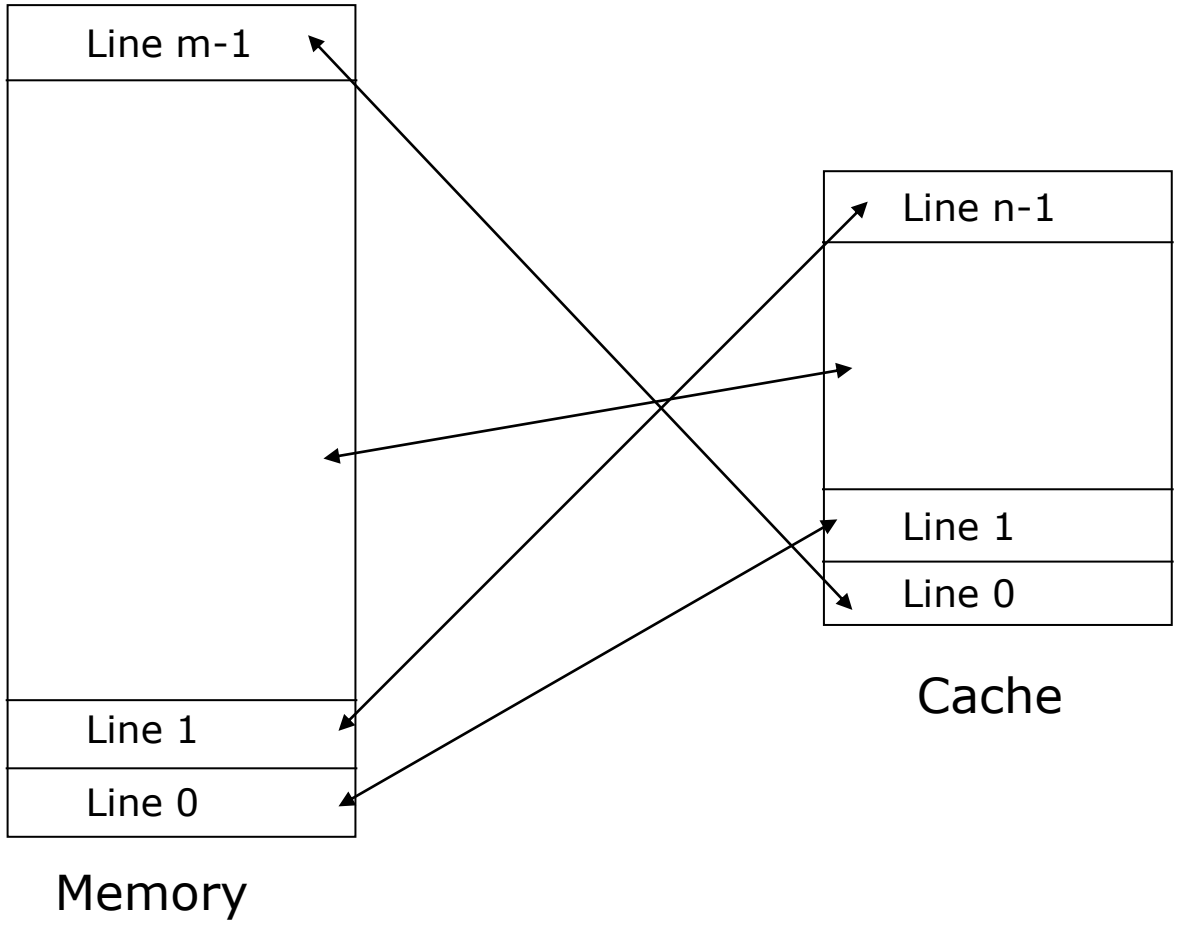
Direct mapping address

- Example:
  - Input:
    - Memory size = 4GB
    - Cache size = 1MB
    - Line size =32 byte
  - Output
    - Line size =32 byte = 2^5 → Word = 5 bit
    - Cache size = 1MB = 2^{10} → there are 2^{10} / 2^5 = 2^5 lines → Line = 5 bit
    - Tag = 32bit address – Line – Word = 32 – 5 – 5 = 22 bit.

Direct mapping

- Pros:
  - Simple design
  - Fast because the mapping is fixed: once we know the memory address we can find it in cache quickly.
- Cons:
  - Due to the mapping is fixed the probability for conflicts is high (why?)
  - Hit ratio is low.

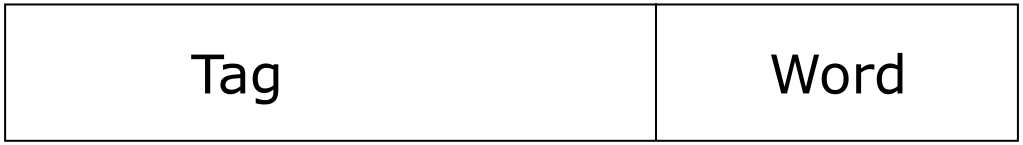
Fully associative mapping



Fully associative mapping

- Cache:
  - Divided into *n* blocks or lines, from $Line_0$ to $Line_{n-1}$
- Memory:
  - Divided into *m* blocks or lines, from $Line_0$ to $Line_{m-1}$.
  - Cache line size is equal to memory line size
  - Number of memory lines if much bigger than Number of cache lines (m >> n).
- Mapping:
  - A line of memory can be mapped to any line of cache;
  - $Line_i$ of memory can be mapped to $Line_j$ of cache;

Fully associative mapping address



- *Tag* (bit) is the address of line in memory (page =1)
- *Word* (bit) is the address of word in line

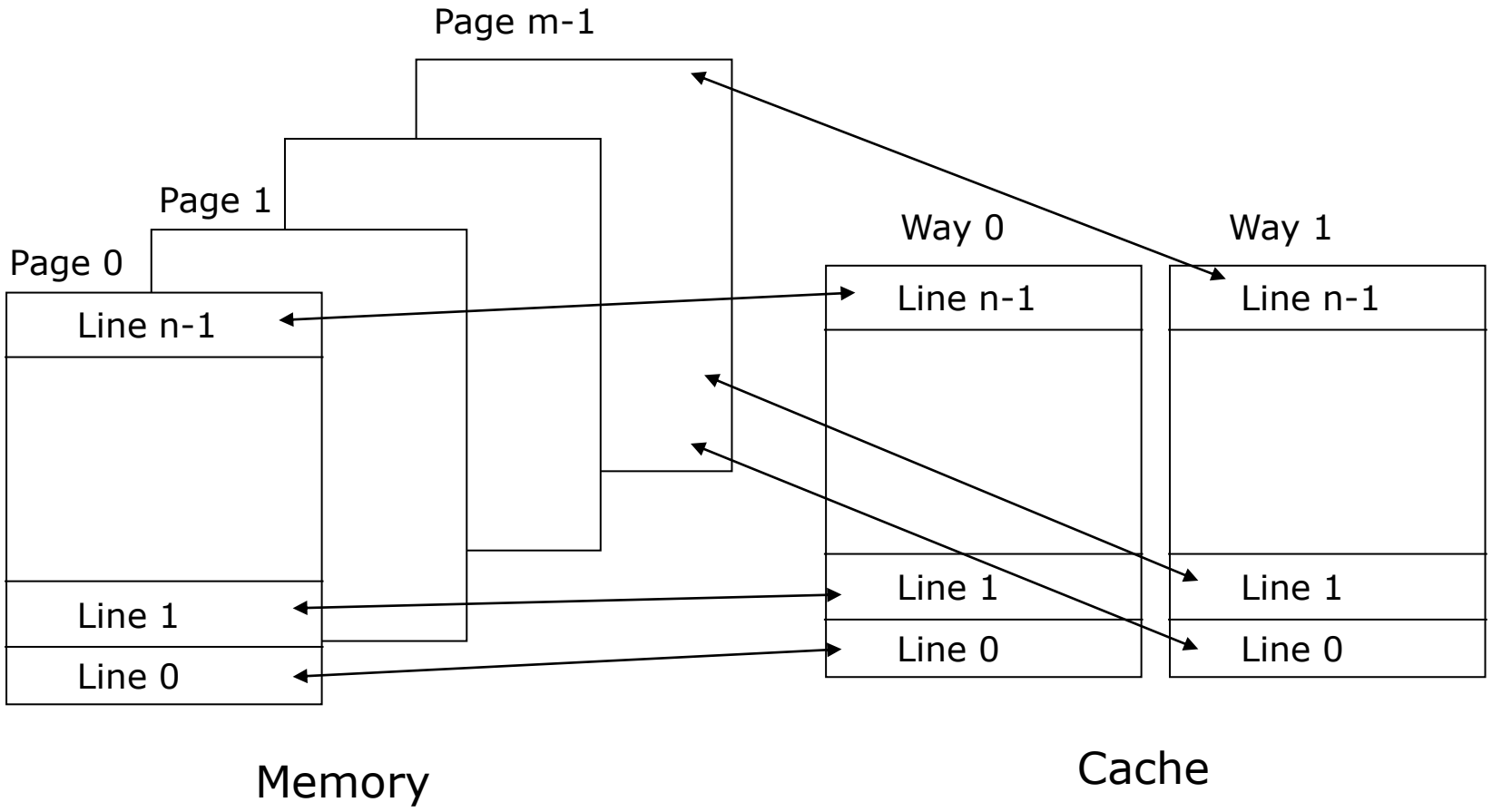
Fully associative mapping address

- Example:
  - Input:
    - Memory size = 4GB
    - Cache size = 1MB
    - Line size =32 byte
  - Output
    - Line size =32 byte = 2^5 → Word = 5 bit
    - Tag = 32bit address – Word = 32 – 5 = 27 bit.

Fully associative mapping

- Pros:
  - Less conflicts because the mapping is flexible
  - Hit ratio is higher.
- Cons:
  - Slow due to we have to search for memory address in cache.
  - Complicated due to there are n address comparators are added into cache.
- Usually uses for small-size caches.

Set associative mapping



Set associative mapping

- Cache:
  - Divided into k ways with same size;
  - Each way is divided into *n* blocks or lines, from $Line_0$ to $Line_{n-1}$
- Memory:
  - Divided into *m* pages, from $page_0$ to $page_{m-1}$.
  - Page has same size as way of cache
  - Each page has *n* lines, from $Line_0$ to $Line_{n-1}$
- Mapping:
  - Page to Way mapping (flexible mapping):
    - A memory page can be mapped to any way of cache.
  - Lines of Page to Lines of Way mapping (fixed mapping):
    - $Line_0$ of $page_i$ is mapped to $Line_0$ of way_j;
    - $Line_1$ of $page_i$ is mapped to $Line_1$ of way_j;
    - ....
    - $Line_{n-1}$ of $page_i$ is mapped to $Line_{n-1}$ of way_j;

Set associative mapping address

- *Tag* (bit) is the address of page in memory
- *Set* (bit) is the address of line in way of cache
- *Word* (bit) is the address of word in line

Set associative mapping address

- Example:
  - Input:
    - Memory size = 4GB
    - Cache size = 1MB, 2 Ways
    - Line size =32 byte
  - Output
    - Line size =32 byte = 2^5 → Word = 5 bit
    - Cache size = 1MB = 2^{20} → there are 2^{20} / 2^5 / 2 = 2^{14} lines in a way → Set = 14 bit
    - Tag = 32bit address – Set – Word = 32 – 14 – 5 = 13 bit.

Set associative mapping

- Pros:
  - Fast because direct mapping is used for line mapping (large number of mappings);
  - Less conflicts because the mapping from memory pages to cache ways is flexible
  - Hit ratio is higher.
- Cons:
  - Complicated in design and control due to cache is divided into a number of ways.

Cache Read/Write Operations

- Read Operations
  - Hit cases (required data item is found in cache)
    - Data item is fetched from cache to CPU
    - Main memory doesn’t participate into the operation
  - Miss cases (required data item is not found in cache)
    - Data item is fetched from main memory to cache and then
    - Data item is fetched from cache to CPU

    ----> miss penalty

Cache Read/Write Operations

- Write Operations
  - Hit cases
    - Write through: Data item is written to cache and to main memory at the same time
    - Write back: Data item is first written to cache and its block in cache is written to main memory when the block is replaced.
  - Miss cases
    - Write allocate (fetch on write): Data item is first written to main memory and then its block is fetched into cache.
    - Write non-allocate (not fetch on write): Data item is only written to main memory.

Cache Replacement Policies

- Why do we need to replace cache lines?
  - Mapping from memory lines to a cache line is usually a many-to-one mapping;
  - Since many memory lines share a cache line, memory lines are loaded into cache for use. After a period of time, these memory lines are replaced by new lines to meet CPU's data/instruction requests.

Cache Replacement Policies

- Replacement policies determine how cache blocks are selected to be replaced when there are new blocks coming in from main memory.
- There are 3 major policies:
  - Radom replacement
  - FIFO replacement
  - LRU replacement

Cache Replacement Policies

- Radom replacement:
  - Cache blocks are randomly selected to be replaced.
  - Pros:
    - Simple to implement
  - Cons:
    - Miss rate is high because the method doesn't take into account which blocks are being used.
      - If a block that is being used is replaced, miss event happens and it needs to be fetched into cache again.

Cache Replacement Policies

- FIFO replacement:
  - Based on FIFO (First In First Out) principle
  - Cache blocks that are fetched into cache first will be replaced first.
  - Pros:
    - Has lower miss rate than random method
  - Cons:
    - Miss rate is still high because the method doesn't take into account which blocks are really being used.
      - An "old" block may still be being used.
    - Complicated to implement because it needs a circuit to monitor the load order of cache blocks.

Cache Replacement Policies

- LRU replacement:
  - Cache blocks that are least recently used (LRU) are selected to be replaced.
  - Pros:
    - Take into account the being used blocks. Blocks with higher usage frequency recently have less chance to be replaced.
    - Has lowest miss rate than random and FIFO methods
  - Cons:
    - Complicated to implement because it needs a circuit to monitor the usage frequency of cache blocks.

Cache Performance and Affected Factors

- Average access time of a memory system with cache:

  t_{access} = (Hit cost) + (miss rate) * (miss penalty)

  $t_{access} = t_{cache} + (1 - H) * (t_{memory})$

  where H is hit rate.

  If t_{cache} = 5ns, t_{memory} = 60ns and H=80%, we have:

  $t_{access} = 5 + (1 - 0.8) * (60) = 5+12 = 17ns$

  If t_{cache} = 5ns, t_{memory} = 60ns and H=95%, we have:

  $t_{access} = 5 + (1 - 0.95) * (60) = 5+3 = 8ns$

Performance Affected Factors

- Factors that affect cache performance:
  - Cache size
    - Big cache or small cache?
  - Split cache: a cache for data and a cache for instructions
  - Create multi-level cache

Affected Factors – Cache Size

- Some statistical figures:
  - Miss rate of instruction cache is much smaller than data cache:

    8KB instruction cache has miss rate < 1%

    256KB instruction cache has miss rate < 0.002%

    ----> increase size of instruction cache is not efficient.

    8KB data cache has miss rate < 4%

    256KB data cache has miss rate < 3%

    ----> increase size of data cache to 32 times, the miss rate reduces 25% (from 4% to 3%).

Affected Factors – Cache Size

- Big cache:
  - Can store more memory blocks
  - Reduce number of times data blocks of different programs have to be swapped between cache and memory.
  - Big cache is slower than small cache
    - Search space for a memory location is larger.
  - Future trend: caches get bigger and bigger:
    - Better support for multi-tasking
    - Better support for parallel processing
    - Better support for multi-core CPU systems.

Affected Factors – Split Cache

- Cache can be divided into data cache and instruction cache for better performance because:
  - Data and instructions are different in localities
    - Data have stronger temporal locality than spatial locality while instructions have stronger spatial locality than temporal locality
  - Instruction cache only supports read operations while data cache supports both read & write.

  ---> easier to optimize caches.
  - Supports multiple read/write operations at the same time ---> reduce resource conflict.

Affected Factors – Split Cache

- Integrate other functions, such as instruction pre-decode into instruction cache ---> better instruction processing.

- In practice, most L1 cache is divided into 2 parts: data cache and instruction cache. Higher level caches are not divided. Why?
  - Splitting L1 cache gives most of benefits because it is nearest to the CPU. CPU directly read/write to L1 caches.
  - Splitting higher level caches doesn't give such benefits but the high level of complexity in cache control system.

Multi-Level Cache

- Multi-level cache:
  - Improve the system performance because multi-level cache can balance the speed of CPU and main memory better than one level cache.

CPU	L1	L2	L3	Main memory
1ns	5ns	15ns	30ns	60ns
1ns	5ns			60ns

  - In practice, cache usually has 2 levels: L1 and L2. Some CPUs' caches have 3 levels: L1, L2 and L3.
  - Reduce the costs.

Cache Miss Reduction Methods

- Good cache:
  - Hit rate is high
  - Miss rate is low
  - Miss penalty is not too slow
- Miss types:
  - Compulsory misses: misses usually happen at the program loading time when program code are loading into memory and code are not fetched into cache.
  - Capacity misses: misses due to cache limited capacity, especially in multi-tasking environment. As cache is small, code of programs are swapped frequently between cache and main memory.
  - Conflict misses: misses due to conflicts when there are many memory blocks are competing for one cache block.

Cache Miss Reduction Methods

- Increase cache block size:
  - Reduce compulsory misses:
    - Bigger blocks cover the locality better than small blocks ---> can reduce compulsory misses.
  - Increase conflict misses:
    - Bigger blocks lead to the reduction of number of cache blocks, hence there are more memory blocks are referencing to one cache block ---> increase conflict misses.
    - Bigger blocks may waste the cache capacity (some part of cache blocks may never be used).

Cache Miss Reduction Methods

- Increase cache associative level (increase number of cache ways):
  - Reduce conflict misses:
    - Increase number of cache ways ---> memory – cache mapping is more flexible, or more choice ---> reduce conflict misses.
  - Make cache slower:
    - Increase number of cache ways ---> increase the search space ---> make cache slower.