B35APO: Computer Architectures

Lecture 04. Memory Hierarchy

Pavel Píša
pisa@fel.cvut.cz

Petr Štěpán
stepan@fel.cvut.cz

13. March, 2024

Outline

1 Memory – Introduction

2 Semiconductor Memories – HW Realization

3 Cache Memory to Speed Up Data Access

4 Virtual Memory and Paging

5 Cache Memory and Paging Together

Motivation

Algorithm A

```
int matrix[N][N];
int main() {
  long int i, j, sum1 = 0;
  for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      sum1 += matrix[i][j];
}
```

Algorithm B

```
int matrix[N][N];
int main() {
  long int i, j, sum1 = 0;
  for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      sum1 += matrix[j][i];
}
```

Both algorithms have the same result and use the same approach. Program A iterates matrix column by column, and program B iterates row by row.

Is there a rule how to iterate over matrix elements efficiently? Which one is faster?

Motivation – Speed Revealed

Algoritmus A

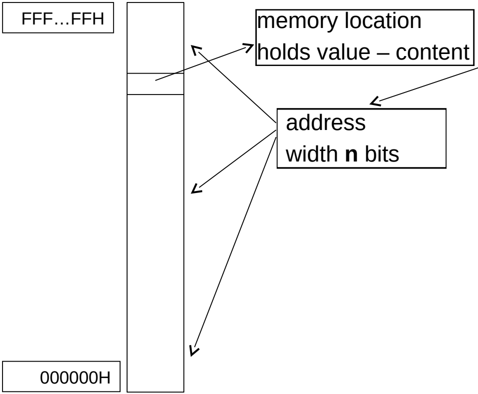
```c
int matrix[N][N];
int main() {
  long int i, j, sum1 = 0;
  for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      sum1 += matrix[i][j];
}
```

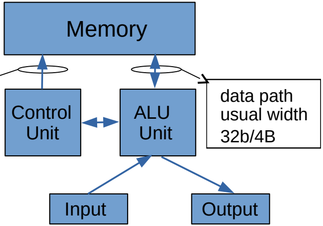
Algoritmus B

```c
int matrix[N][N];
int main() {
  long int i, j, sum1 = 0;
  for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      sum1 += matrix[j][i];
}
```

N	A	B
100000	12.791328s	138.047563s
10000	0.126945s	0.486535s
1000	0.001329s	0.001756s
100	0.000083s	0.000094s

What is the Memory





The most common size of addressable memory unit is 1B (8 bits)

n	2^n
8	256 distinct locations
16	64K (K=1024)
…	……
32	4G (4096M, $M=K^2$)

What is the Memory – Description

Memory:

- it is array of addressable memory cells
    - all cells (elements) have the same size – 8-bits (byte) is usual or multiple by two power
- physical address space capacity is limited by number of bits forming address signal generated by CPU (ISA addressing modes) which are sent to address bus. For memory organize by bytes:
    - 16-bit address allows to access cells in 64 KiB of memory
    - 32-bit address allows to access 4 GiB of memory capacity
    - 37-bit address (maximum for Intel Core i9-13900K when mainboard support is as well) allows to address in 128 GiB of capacity
- Access types, rules:
    - the internal memory of the computer allows access in random order (= random access – RAM)
    - some of external memory (for example magnetic tape used for backup – cheaper than HDD and SSD) allows only sequential access, that is reading and writing of bytes in the fixed order
    - in HDD, SSD and Flash cases, only read or write by whole block is possible; for SSD/Flash is write possible on block basis and it is possible only once after much larger holding erase blocks is reset into initial consistent state.

Semiconductor Memory – Basic Terms

Address input, it selects exactly one cell (entry) for next data read or write access (same as index for array in C language)

Value (data) the corresponding cell data or signals to read or write them

control signals set of signals choosing read or wrote operation, data validity, or used for redundancy which allows even bit-flip corrections

Main semiconductor memory parameters:

Access time (latency) the time interval from the request to data access to their availability to the requester

Read/write cycle time access time (with precharge for DRAM) + refresh after destructive read + required idle time to next access

Throughput/bandwidth main performance indicator. Rate of transferred data units per time.

parameters can appear in the datasheet as maximal, average and minimal required

Outline

1 Memory – Introduction

2 Semiconductor Memories – HW Realization

3 Cache Memory to Speed Up Data Access

4 Virtual Memory and Paging

5 Cache Memory and Paging Together

Memory Types and Required Maintenance

Division of memories according to write operation mehanism:

- **ROM** (Read-Only Memory) memories that can only be read from (content defined ta production), include EEPROM (Electrically Erasable Programmable Read-Only Memory) requiring high voltage for erase and then allow write new data, write is not normal operation.
- **RAM** (Random Access Memory) also RWM (Read-Write Memory) classic memories designed for reading and writing any cell in any order – Random Access.

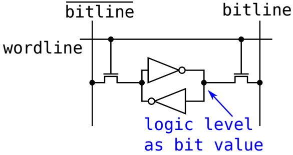
Division according data preservation when power is turned off and on:

- **Permanent** (Non-volatile) memory does not need power to keep information - e.g. Flash, EEPROM, EPROM, ROM, ferromagnetic memories, HDD, SSD, 3D-X Point – Intel Optane Memory
- **Volatile** (Volatile) for example DDR memory SDRAM, SRAM, cache in classical computer, continuous power (and data refresh for DRAM) is needed to maintain information.

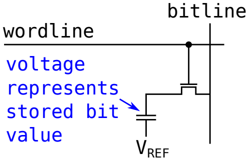
Memory Realization by Electrical Circuit

According to the realization, we divide RAM (RWM) into:

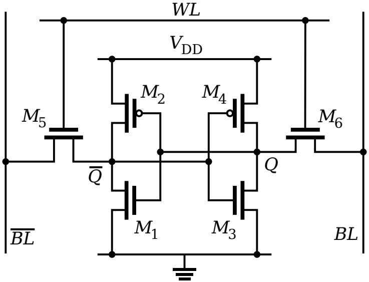
SRAM Static RAM – a physical quantity representing a value has a constant value over time. Typically two loop-connected inverters. Multiple components, more expensive, no periodic maintenance required, only needs power supply

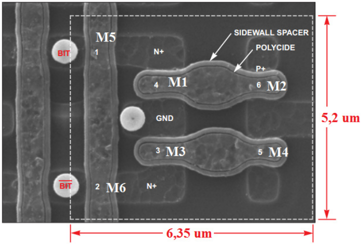


DRAM Dynamic RAM – a quantity (voltage) changes its value over time. Typically a capacitor that discharges spontaneously and therefore it is necessary to refresh the information at regular times. Only a capacitor and a transistor – cheap, takes less space.



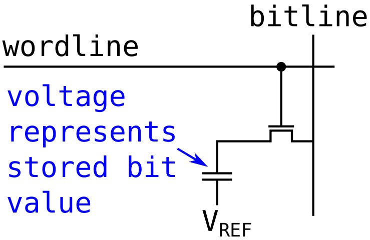
Static Memory (SRAM) – Cell Detail

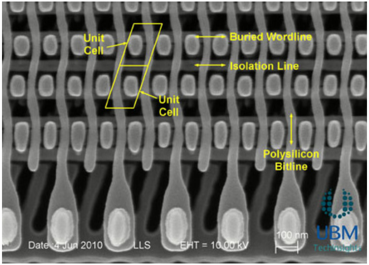




- Fully static circuit (positive feedback)
- Requires power supply (volatile memory) to maintain information
- Disadvantage, needs 6 transistors, large area
- Reading, after selecting a word (world line – WL) the data is transferred to the respective non-powered conductor (bit line – BL)
- Writing – connecting bit line and it inverted signal to log. 1 and log. 0 or vice versa will force the state with a greater current than on the inverter outputs in internal positive loop

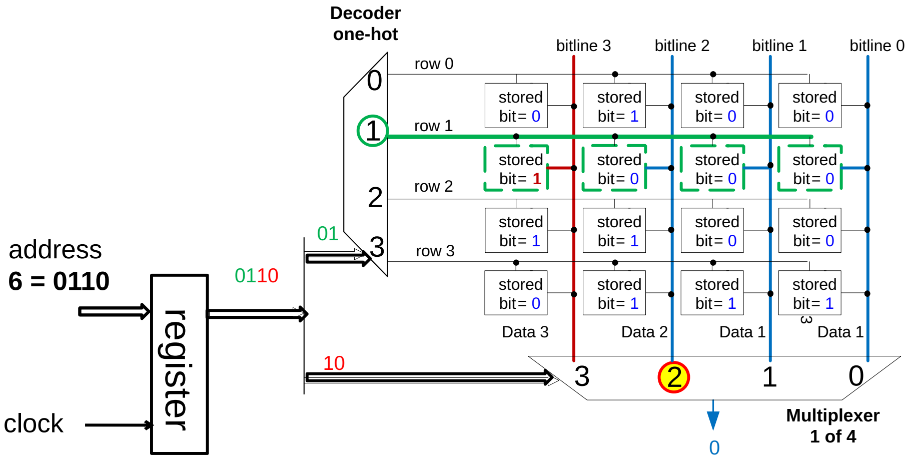
Dynamic memory (DRAM) – Cell Detail



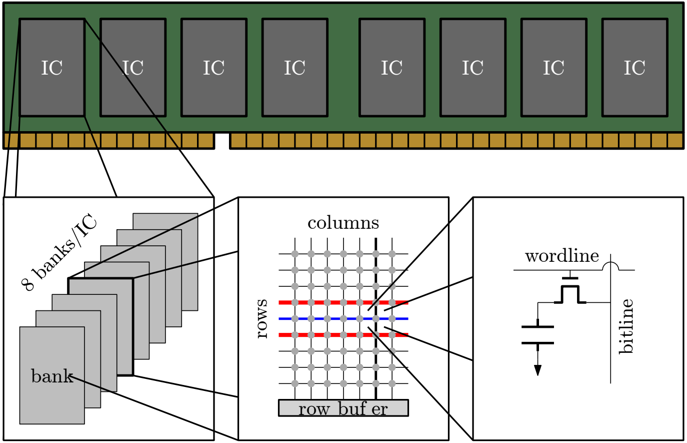


- nMOS transistor represents a switch that connects (or not) the capacitor to the ”bitline” conductor. The connection is controlled by the ”wordline” conductor.
- The reading process discharges the capacitor. Therefore, it must be restored afterwards.
- Refreshing of memories (refresh) – the charge is spontaneously lost from the capacitor. Necessary working phase of dynamic memory. Negatively affects (prolongs) the average access time.

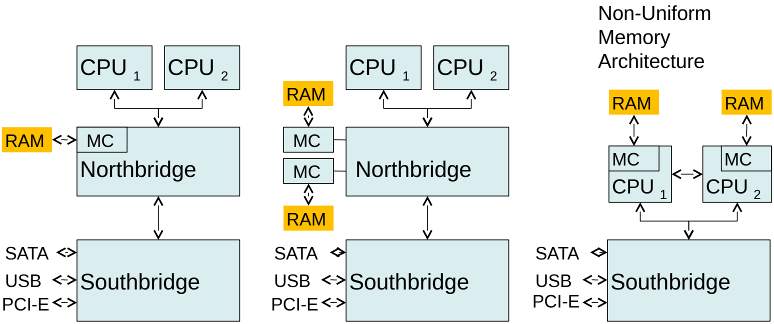
Memory Array Organization – the Principle



SDRAM Packaged as Dual In-Line Memory Module



Dual In-Line Memory Module (DIMM), 64-bit data path, Synchronous Dynamic Random Access Memory (SDRAM), operation and signals synchronized (S in abbreviation) by clock



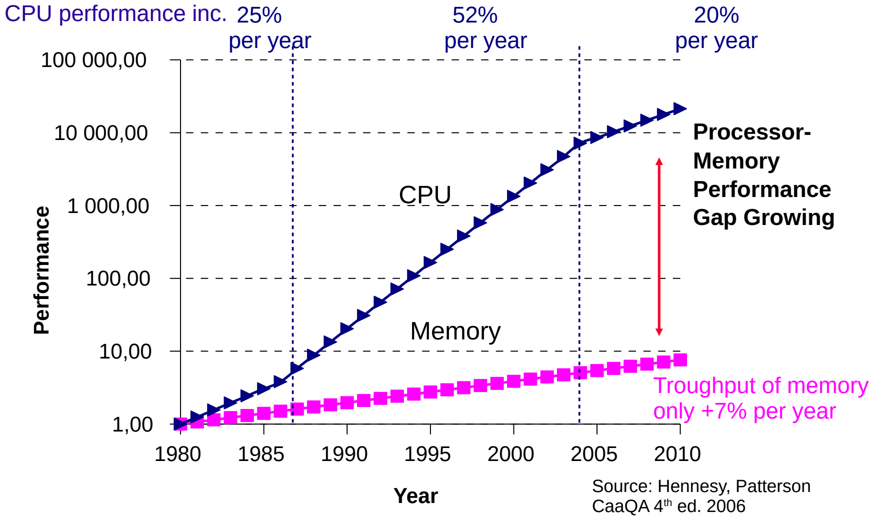
MC - Memory controller – provides read and write control to SDRAM. It also controls refresh of information of each memory cell once every 64 ms typically (iterates over whole memory row by row and cells in each row are refreshed simultaneously).

Most Frequently Used Dynamic Memory Types

- **SDRAM** – clock frequency up to 100 MHz, 2.5V, synchronous data transfer on the clock edge
- **DDR SDRAM** – data transfer on both clock signal edges, 2.5V, I/O clocks up to 100-200 MHz, 0.2-0.4 GT/s (billions of transmissions per skund)
- **DDR2 SDRAM** – lower power consumption, 1.8V, frequency up to 400 MHz, 0.8 GT/s
- **DDR3 SDRAM** – even lower power consumption at 1.5V, frequency up to 800 MHz, 1.6 GT/s
- **DDR4 SDRAM** – 1.05 – 1.2V, I/O bus clock 1.2 GHz, 2.4 GT/s
- **DDR5 SDRAM** – 1.1V, up to 6.4 GT/s

All of these types are predominantly optimized for throughput, not random access, latency 20 to 35 ns.

Memory and CPU Speed Evolution in History



Source: Hennesy, Patterson
CaaQA 4th ed. 2006

Outline

1 Memory – Introduction

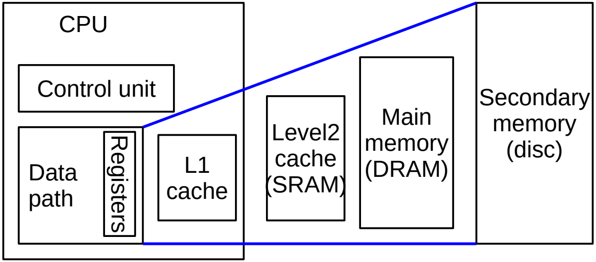
2 Semiconductor Memories – HW Realization

3 Cache Memory to Speed Up Data Access

4 Virtual Memory and Paging

5 Cache Memory and Paging Together

Memory Hierarchy from CPU Registers to SSD



Type	L1 SRAM	Sync SRAM	DDR3	HDD
Velikost	32kB	1 MB	16 GB	3TB
Cena	10 kč/kB	300 kč/MB	123 kč/Gb	1 kč/GB
Rychlost	0.2...2ns	0.5...8 ns/word	15 GB/sec	100 MB/sec

Some data may exist in multiple copies (levels, SMPs). Modifying data requires mechanisms to maintain the coherence (mostly by HW) of words and consistency of data structures (programmer care required).

Memory Hierarchy – Fundamental Principles

- Programs/processes access only a small portion of their address space at a time
- **Temporal locality**
  - Items accessed recently will be needed again soon
  - Example: program loop, function local and induction variables
- **Spatial locality**
  - Items near those accessed recently are likely to be accessed soon
  - Example: sequence code access (program memory), data arrays sequential access (data memory).

The principle considered on all levels, in algorithms (local variables), compilers (move to registers), between memory levels (automatically), operating system (move pages between disk and main memory) or again programmatically, reading and writing to files or caching web pages.

Memory Hierarchy Introduced Based on Locality

- The solution to resolve capacity and speed requirements is to build address space (data storage in general) as hierarchy of different technologies.
- Store input/output data, program code and its runtime data on large and cheaper secondary storage (hard disk)
- Copy recently accessed (and nearby) items from disk to smaller DRAM based main memory (usually under operating system control)
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory (cache) attached to CPU (hidden memory, transactions under HW control), optionally, tightly coupled memory under program's control
- Move currently processed variables to CPU registers (under machine program/compiler control)

Cache Memory to Overcome CPU Speed Growth Gap

- component that (transparently) stores data so that future requests for that data can be served faster
- transparent cache – hidden memory
- the purpose of hidden memory is to speed up access to frequently used data on ”slow” media by copying it to fast media.
- it mostly works automatically and adapts to the current needs of the program.
- however, it is necessary to know about its existence and often needs servicing at the level of the operating system and, in some cases, even programs.
- if you choose data structures, access paterns and algorithms unwisely, its effect is lost.

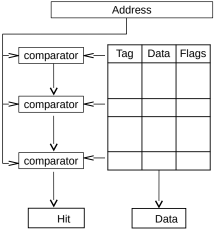
Cache Memory – Terms Definition

- **Cache hit** naming the situation when the data for requested address is found in the cache.
- **Cache miss**, opposite, **search failure**, the data is not yet in the cache.
- **Cache line** (line) or **Cache block** – the basic copieable unit between hierarchical levels.
- In practice, the size of the **Cache** line varies from 8B to 1KB, typically 64B.
- **Hit rate** – the number of memory accesses served by a given cache level divided by all accesses
- **Miss rate** – the ratio of accesses to be served from slower memory = 1 - Hit rate
- **Average Memory Access Time** (AMAT)

$$AMAT = HitTime + MissRate \times MissPenalty$$

- AMAT for multi-level cache can be calculated by recursively applying the aforementioned relationship

Cache Memory – Implementation



Cache line of fully associative cache

Tag	Data	Flags

- **Tag** is the index of the corresponding block in the operating memory (basically, it is the value of the pointer/address divided by the length of the block or cache way – see later).
- **Data** is an array containing actual values at the corresponding address(s).
- **Validity bit** – indicates whether the contents of the **Data** field are even valid.
- **Dirty bit** – indicates that there is a different value in the cache than in the main memory.
- more additional bits (metadata) attached to memory content can be required for SMP and maintenance

Processing of Cache Miss Situation

- Data must be loaded from main memory, but usually all cache entries are already filled with data retained from a previous run of the program.
- One of the blocks that can be used to store data from the given address needs to be released.
- Deciding an block to discard is very important, if the one that will be needed again is selected, performance will decrease.
- **Cache replacement policy** – rules for selecting an item to discard
  - **Random** – the selected block is at random position in the cache
  - **LRU** (Least Recently Used) – the selected block is the longest unused one, additional information must be added to the cache circuits to each block group to track the order of recent hits for each item.
  - **LFU** (Least Frequently Used) – tracks how often/how many times the blocks are accessed, requires adding forgetting.
  - **ARC** (Adaptive Replacement Cache) - combination of LRU and LFU

Processor Writes to Cached Main Memory

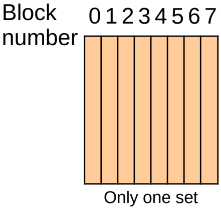
- Cache is on the way that may contain the block being written to.
- At least from the point of view of the given processor, it is necessary to ensure the data coherence for the given processor (often for multiple processors – threads) for accessing each individual address even if there are multiple access paths
- **Write through** cache – if the data is already in the cache, it is modified, in the variant with automatic allocation, block is loaded and then modified even if there is a miss. The data is sent to the main memory at the same time, either directly or via write buffer
- **Write back** – the data is written to the appropriate cache block, if it is not in the cache, the block is loaded first. The block is marked by **Dirty bit**. Writing to the next level is activated only if the cache entry needs to be released for replacement of other data or when sychronization by is requested by the processor, system (cache flush) is required.

Basic Cache Organization Configurations

We consider a cache of 8 blocks and situation where main memory block at address/block_size equal to 12 is acessed. Which cache block(s) can be used for this situation

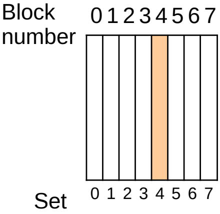
Fully Associative

Address 12 can be placed arbitrarily



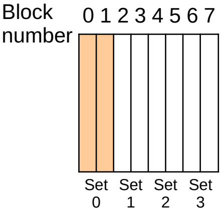
Direct Mapped

Address 12 can be placed only in block 4 (12 mod 8)

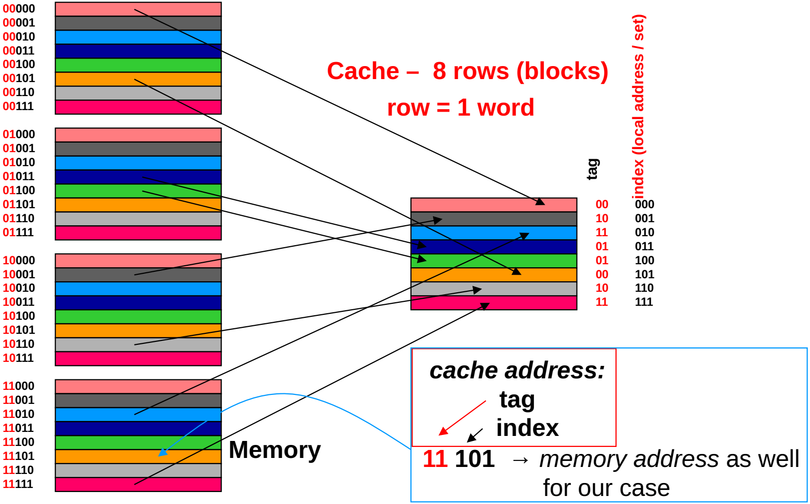


2-Way Associative

Address 12 can be placed in set 0 (12 mod 4)



Direct Mapped Cache Memory – Mapping of Addresses



Cache – 8 rows (blocks)

row = 1 word

cache address:

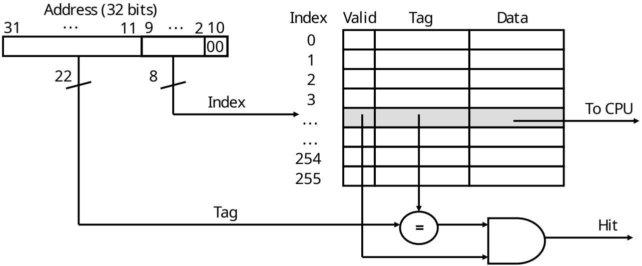
tag

index

11 101 → *memory address* as well for our case

Direct Mapped Cache Memory for Block Equal to Word

- **Capacity** – C … in figure 1024 bytes which is 256 words
- **Number of sets** – SN .. 256 sets (equal to $C/WS/BS$)
- **Word size** – WS .. 4 bytes for considered example
- **Block size** – BS .. 1 word (4 bytes) for shown example
- **Number of blocks** – BN .. number of blocks (256 there)
- **Degree of associativity** – N number of ways to which capacity or blocks are divided (1 there)

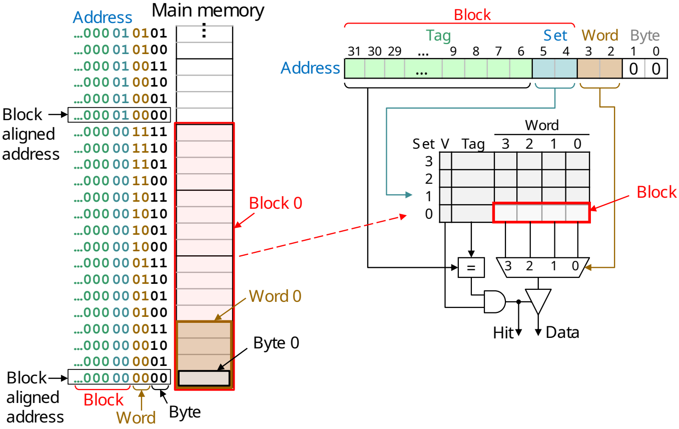


$\mathrm{C} = 1024$ bytes, $\mathrm{SN} = \mathrm{BN} = 256$, $\mathrm{BS} = 1$, $\mathrm{N} = 1$

Direct Mapped Cache Memory for Larger Block Size

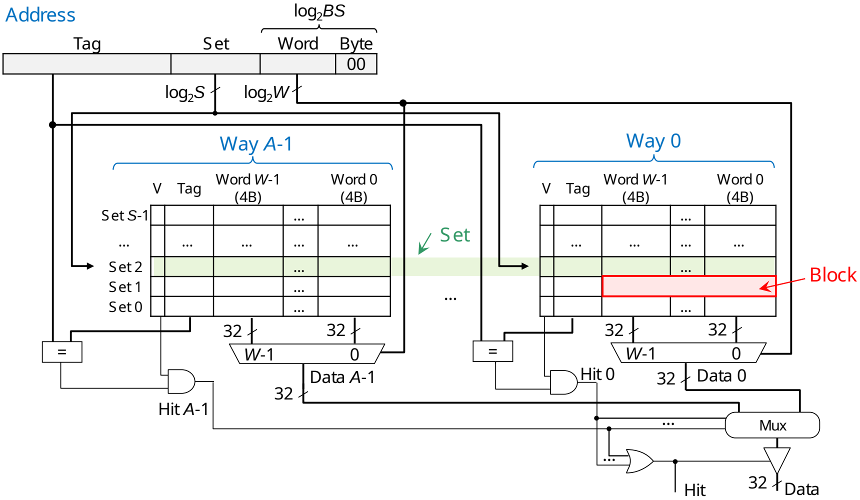
BS = 4 (16 byte, word WS = 4 byts), number of sets SN = 4

set = (adresa div BS) mod SN



Source: Michal Štepanovský

General Cache Memory Organization with N Ways



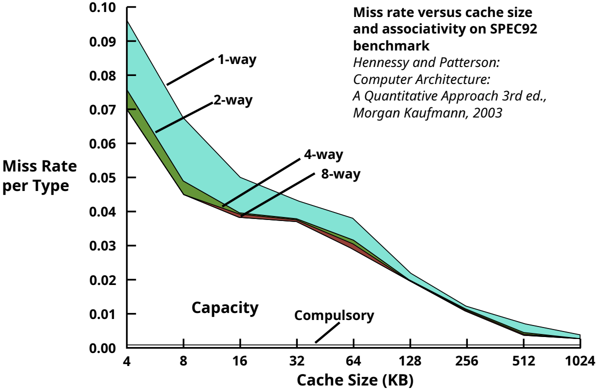
Source: Michal Štepanovský

Demonstration: QtRvSim vect-inc, vect-add2, vect-add

Start with direct mapped cache and `vect-inc`. If the block size is greater than one word and the vector size is greater than the capacity of one path, `vect-add2` will experience considerable performance degradation. After increasing the degree of associativity to two ways even while maintaining the overall capacity, aliasing does not occur anymore. However, after extending the algorithm to work with three vectors (`vect-add`), the cache trashing occur again for write-back or write- through and allocate setup. The worst situation will occur for the LRU policy. Increasing the degree to three, or rather to the more usual four, ways will again suppress the number of misses to one for each sequentially accessed block.

- `https://gitlab.fel.cvut.cz/b35apo/stud-support/-/tree/master/seminaries/qtrvsim/vect-inc`
- `https://gitlab.fel.cvut.cz/b35apo/stud-support/-/tree/master/seminaries/qtrvsim/vect-add2`
- `https://gitlab.fel.cvut.cz/b35apo/stud-support/-/tree/master/seminaries/qtrvsim/vect-add`

Miss Rate and Cache Organization and Size Relations



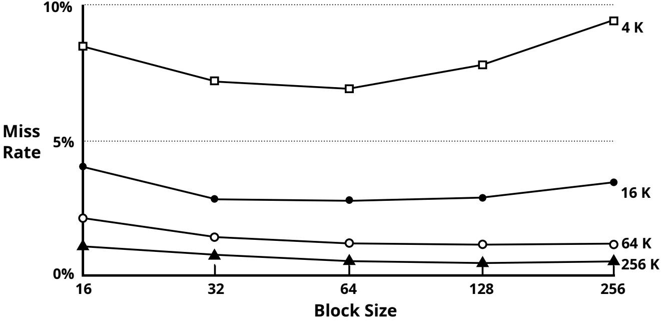
Miss rate versus cache size and associativity on SPEC92 benchmark
Hennessy and Patterson: Computer Architecture: A Quantitative Approach 3rd ed., Morgan Kaufmann, 2003

- miss rate is not cache parameter/feature
- miss rate is not parameter/feature of the program/algorithm

Miss rate depends on both the algorithms in the program and the cache parameters and often the data being processed.

Miss Rate Changes with Cache capacity and Block Size

Miss rate versus block size and cache size on SPEC92 benchmark



Source: Hennessy and Patterson: Computer Architecture: A Quantitative Approach 3rd ed., Morgan Kaufmann, 2003

Increasing block size helps to load neighboring data, which would often subsequently be needed. But if additional cached date are not needed, then the cache miss penalty increases and at the same time there are more collisions and cache capacity is wasted for unnecessary data.

Miss Rate Changes with Cache capacity and Block Size

1. Larger block size
   Reduces compulsory misses; increases other misses, miss penalty
2. Larger cache size
   Reduces capacity/conflict misses; increases hit time, power, cost
3. Greater associativity
   Reduces conflict misses; increases hit time, power
4. Multiple cache levels
   Reduces Miss Penalty, allows for optimizations at each level
5. Prioritize read misses over writes
6. Avoid address translation of cache index

Outline

1 Memory – Introduction

2 Semiconductor Memories – HW Realization

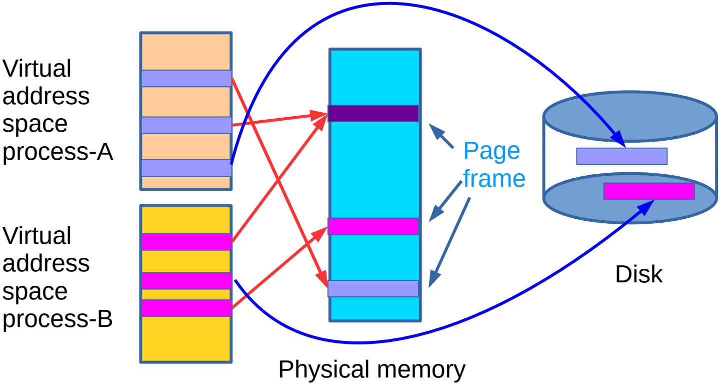
3 Cache Memory to Speed Up Data Access

4 Virtual Memory and Paging

5 Cache Memory and Paging Together

Process Address Space and Swapping Pages to Disk

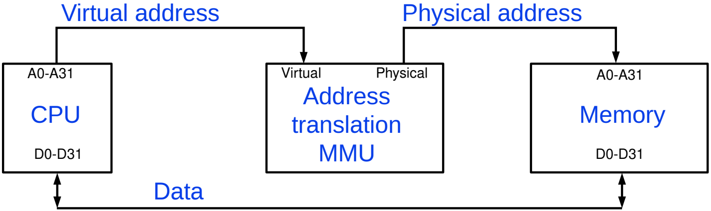
Multiple processes, each with its own memory address space. Mutual Protection and the possibility of expanding the main memory capacity by secondary storage – **swap**.


Virtual address space process-A
Virtual address space process-B
Page frame
Physical memory
Disk

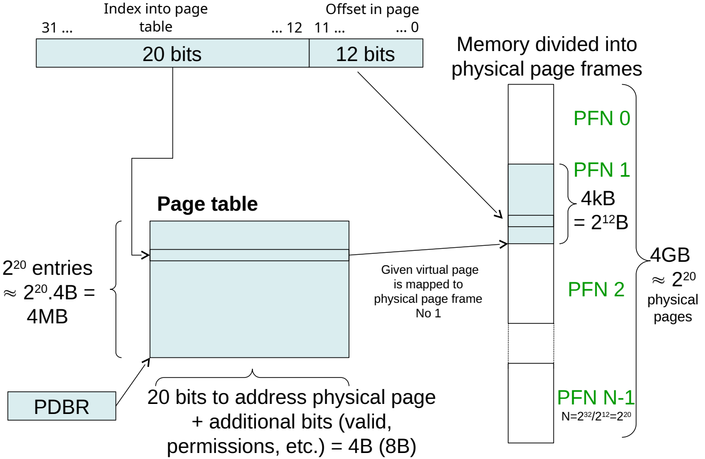
Translation by bytes would be expensive, space is divided into (aligned), typically 4 kB (sometimes larger, e.g. 64 kB), **pages**.

Virtual to Physical Address Translation

The translation is realized by the **Memory Management Unit** (MMU).
Translation to the pages present in memory is usually done automatically in hardware after the operating system fills page tables and sets up the **Page Directory Base Register** (PDBR) for given process.
On the other hand, page faults are resolved by the operating system code.
If there is access to a virtual memory area mapped into the processes and actual page is not presnet then the system reads data from the disk, network, swap partition.
As with the cache, it is necessary and challenging to find space for newly needed pages (a principle similar to LRU).

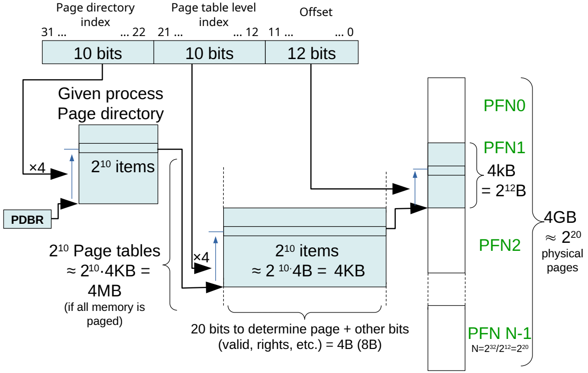


Single-level Page Table (MMU)



Drawback of the solution, for every, even small, running process on a 32-bit system and for 4 kB pages, it is necessary to allocate 4 MB (translation of 20-bit address, 4-byte entry for 32-bit physical address)

Two-levels Page Table – 32-bit Intel x86



Page Directory for each process, but the second level of table with page etries is allocated only when respective virtual space area is used.

When translated address is divided in 10-bit groups, the array of entries of both the directory and the table occupies to 4 kB, same as page size. Higher-order allocations (multiple consecutive pages) are not needed and memory management is not compicated by fragmentation.

Page Directory Entry (PDE) Bit Fields

31...							1	0
Can be used by the operating system								P=0

31... 12		6	5	4	3	2	1	0
The base address of the Page table		...	A	PCD	PWT	U/S	R/W	P=1

- bit 0: Present bit – the next level or page is presnet in physical memory (1). If it is not (0) then data are stored on disk or area is not mapped. The bit is sometimes denoted as V – valid bit.
- bit 1: Read/Write: 1 – R/W – write is permitted; 0 – read only (RO)
- bit 2: User/Supervisor: 1 – user program access permitted; 0 – only accessible by operating system
- bit 3: Write-through/Write-back – which method is used for writes
- bit 4: Cache disabled/enabled – important for memory mapped peripherals, which require immediate (uncached) registers read/write
- bit 5: Accessed – set by MMU when clean and the page is accessed, used by operating system

Final **Page Table Entry** (PTE) Bit Fields

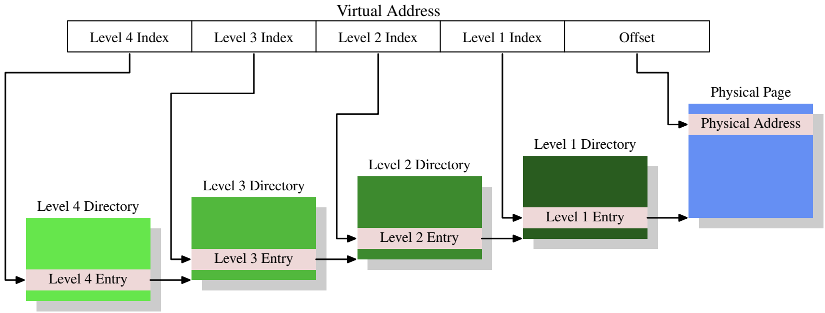
31...							1	0
Can be used by the operating system								P=0

31... 12	7	6	5	4	3	2	1	0
The base address of the Page	...	D	A	PCD	PWT	U/S	R/W	P=1

- bit 6: Dirty bit, also Modified sometimes – is set by MMU if page write into page range was executed from last operating system check and the flag clear
- The rest of flags and fields have same meaning as for page directory

The **Accessed** bit is crucial when the operating system searches for physical memory page frame to be released to allow it reuse for new virtual page frame. The OS usually counts more comprehensive statistics based on these bits and zeroes them as it passes through the page frames list. If **Dirty** bit is set, then when the physical page (PFN) is released, it is necessary to sync the data back to the file, swap partition, etc.

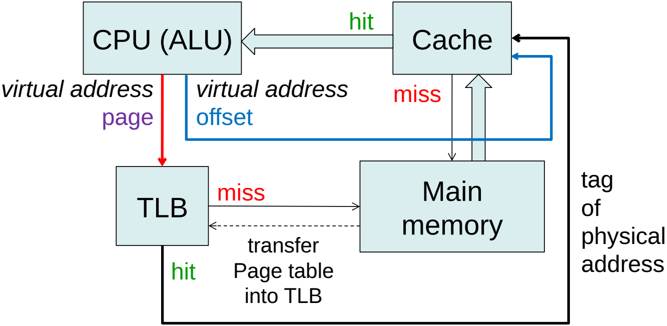
Virtual Address Translation for 64-bit CPU Architectures



It requires multiple levels. For 64-bits and 4 kB pages, it is theoretically necessary to translate 52-bits. If the individual parts of the tables are sized to fit exactly into one page and entries are 8 bytes (64 bits) each, then the translation processes 9 bits ($512 \times 8 = 4096$) per level and thus $\mathit{ceil}(52/9) = 6$. Often, however, one to two levels are omitted, see more later.

Speed Up (Caching) of Repeated Translation

Each translation represents several accesses to memory, and although it is accelerated by cache memory, it slows down operations. The **Translation Look-Aside Buffer** (TLB) is being utilized. It works on the same principle as the memory cache, but stores a virtual address with its translation to a physical one. Again, limited capacity, LRU, and the it needs to be taken into account when programming.



Outline

1 Memory – Introduction

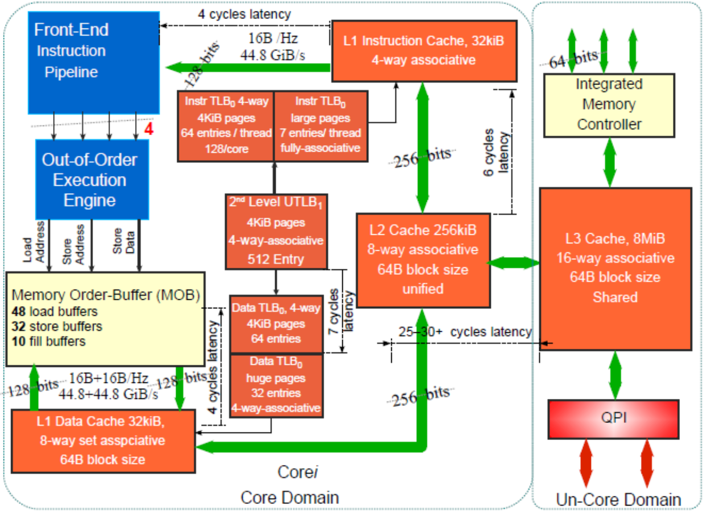
2 Semiconductor Memories – HW Realization

3 Cache Memory to Speed Up Data Access

4 Virtual Memory and Paging

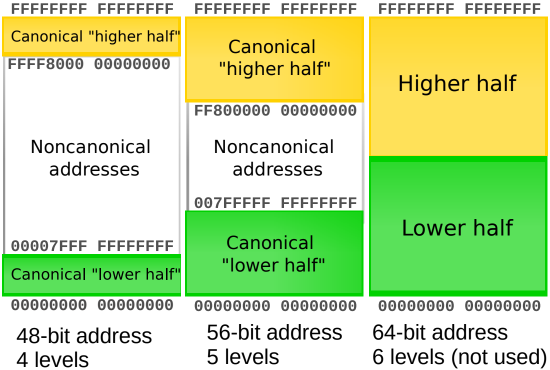
5 Cache Memory and Paging Together

Memory Subsystem on Real Chip - Intel Nehalem


Front-End Instruction Pipeline
4 cycles latency
16B /Hz
44.8 GiB/s
128 bits
L1 Instruction Cache, 32kiB 4-way associative
Instr TLB$_0$ 4-way 4KiB pages 64 entries / thread 128/core
Instr TLB$_0$ large pages 7 entries/ thread fully-associative
4
Out-of-Order Execution Engine
Load Address
Store Address
Store Data
2nd Level UTLB$_1$ 4KiB pages 4-way-associative 512 Entry
256 bits
6 cycles latency
L2 Cache 256kiB 8-way associative 64B block size unified
7 cycles latency
Memory Order-Buffer (MOB)
48 load buffers
32 store buffers
10 fill buffers
Data TLB$_0$, 4-way 4KiB pages 64 entries
Data TLB$_0$ huge pages 32 entries 4-way-associative
4 cycles latency
128 bits
16B+16B/Hz
44.8+44.8 GiB/s
128 bits
L1 Data Cache 32kiB, 8-way set asspciative 64B block size
25-30+ cycles latency
256 bits
Core$i$
Core Domain
64 bits
Integrated Memory Controller
L3 Cache, 8MiB 16-way associative 64B block size Shared
QPI
Un-Core Domain

Paging on 64-bit Processor Architectures

The full 64-bit length of the physical address is not (yet) in use. Neither is the 64-bit virtual address. Translation levels slow down the execution. The highest address bits are replaced by the sign extension. The top virtual memory range area is reserved for the operating system, the the bottom for applications. Further optimization of the optional larger page – huge pages.



Literature and Online Resources

- Ulrich Drepper: What every programmer should know about memory
- Agner Fog: Software optimization resources. C++ and assembly
- `https://www.7-cpu.com/cpu/Haswell.html`
- `https://www.7-cpu.com/cpu/Skylake.html`
- WikiChips: Zen - Microarchitectures - AMD

Left Empty for Notes