Computing architectures Part 1

TMA4280—Introduction to Supercomputing

NTNU, IMF

January 19. 2018

Outline

Data representation

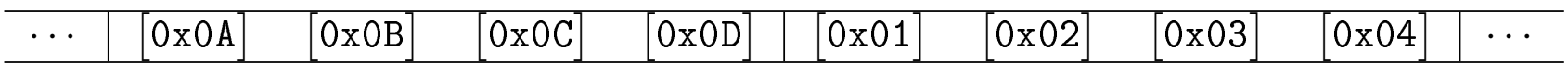
Single processor system

Performance

Data representation

Data is stored in binary in the memory:

- A *bit* or binary digit is a value 0 or 1.
- The basic unit is the *byte* consisting of 8 bits,
- Memory is as a contiguous array of bits
- These bits can be accessed by taking the address of the entry.
- Memory may be addressed with a multiple of *bytes*,
- Memory alignment denotes addressing the memory modulo *N* bytes.



Memory addressing: *byte-addressed*, the lowest unit accessible is usually the *byte*.

Data representation

A byte is a sequence of 8 binary digits.
`char` is encoded on a byte, it is the smallest integral type.

$$\begin{bmatrix} b_7 & b_6 & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \end{bmatrix}$$

$$\begin{aligned}
0 &: \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\
1 &: \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \\
2 &: \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \\
3 &: \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \\
4 &: \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \\
\ldots &: \quad \ldots \\
255 &: \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}
\end{aligned}$$

Conversion from binary to decimal representation of a number:

$$\begin{bmatrix} b_k & b_{k-1} & \ldots\ldots & b_1 & b_0 \end{bmatrix}_2 = \sum_{i=0}^{k} b_i \, 2^i$$

Types: integral types

ANSI/ISO C must support four signed and four unsigned data types:

1. `char`
2. `short`
3. `int`
4. `long`

ANSI requirements:

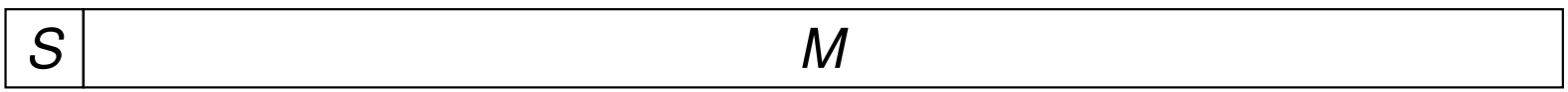
1. `short` and `int` are at least 16-bit long.
2. `long` is at least wider than `int` and not less than 32-bits.

Types: integer

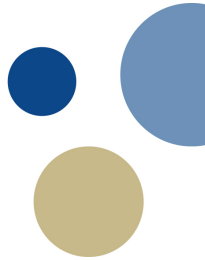
— Representation given number of binary digits: *fixed width*
— Signed integers, different strategies:
  1. explicit sign: *sign-magnitude*, *one's complement*, *two's complement*, ...
  2. implicit sign: *excess-k*, *base-2*, ...

In explicit N-bit wide signed representations:

— S: sign bit
— M: N-1 bits for the magnitude



Types: signed integers



Two pieces of information need to be represented: the sign and the absolute value (or magnitude).

Different representations for the magnitude of negative numbers:

— *sign-magnitude*: same as non-negative numbers
  - zero is encoded in two ways
  - representation is natural thus not machine friendly

— *one's complement*: bitwise not of non-negative magnitude
  - zero is encoded in two ways
  - *end-around carry*: if 1 is carried it is added back (implementation difficulty)

— *two's complement*: bitwise not of non-negative magnitude + 1
  - zero is encoded in only one way
  - can be seen as a circular encoding

The *two's complement* is the most widely used on modern architectures.

Types: Sign-and-magnitude

This representation is the intuitive way of expressing signedness: one signe bit and N-1 bit for the magnitude.

S	M

Range is $[-127, +127]$. Zero is encoded as -0 and $+0$:

$$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

The definition of two zero and the impossibility of representing the subtraction in an efficient way are strong limitations.

Types: One's complement

Negative values consists of applying a bitwise NOT operator to the unsigned representation.

Range is $[-127, +127]$.

Zero is encoded as -0 and $+0$:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Hard to implement in pratice: *end-around carry*: if 1 is carried then it is added back. This representation has been abandoned.

Types: Two's complement

Most used representation developed to workaround the double zero and the carry issue: if 1 is carried is is simply ignored.

Negative values consists of applying a bitwise `NOT` operator to the unsigned representation and adding 1.

Range is $[-128, +127]$: notice that it is not symmetric anymore.

Zero is encoded uniquely as:

$$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

Representation of the bounds:

$$+127: \quad [0\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$$

$$-127: \quad [1\ 0\ 0\ 0\ 0\ 0\ 0\ 1]$$

$$-128: \quad [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

Types: Two's complement

Motivation: the subtraction is consistent since:

$$[+x]_2 \; + \; [-x]_2 = [0]_2$$

Take the simple example of: $+127 - 127$

$$\begin{array}{ll} +127 & \left[0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\right] \\ -127 & \left[1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\right] \\ = 0 & 1\left[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\right] \end{array}$$

Real numbers

The difficulty to represent infinitely many real numbers with a finite number of bits: only some numbers can be represented exactly.

A number is *binary rational* if it can be written exactly in base 2.

Other number are then approximated: the error due to the representation is coined *round-off error*.

Real numbers

$$[527.2]_{10} = 1 \cdot 2^{+9} + 1 \cdot 2^{+3} + 1 \cdot 2^{+2} + 1 \cdot 2^{1} + 1 \cdot 2^{0} + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} + \cdots$$

$$[527.2]_{10} \approx (+1) \cdot 2^{+9} \left(1 \cdot 2^{+0} + 1 \cdot 2^{-6} + 1 \cdot 2^{-7} + 1 \cdot 2^{-8} + \cdot 2^{-9} + 0 \cdot 2^{-10} + 1 \cdot 2^{-11} + 1 \cdot 2^{-12}\right)$$

$$[527.2]_{10} \approx (+1) \cdot 2^{+9} \ [1.0000011110011]_{2}$$

$$(+1) \cdot 2^{+9} \ [1.0000011110011]_{2} = 527.1875$$

Fixed point numbers

— How to encode decimal numbers in the binary system?

— First alternative:

1. Use x bits to represent digits before the radix point (integer),
2. and y bits to represent digits after the radix points,

where $x + y = w$ for a w bit representation: fixed point representation.

— Problem: only a fixed range of numbers can be represented i.e. $2^x + 1 - 2^{-y}$ is the largest.

— Called fixed point since the point is fixed. The *absolute* accuracy is constant.

S	M	F

Types: floating-point

Infinitely many real numbers $\rightarrow$ finite number of bits of representation

Representation and behaviour defined by the IEEE-754 standard:

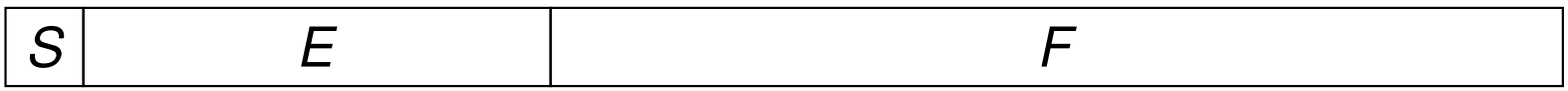
- Definition of *guard digits* to reduce error e.g when substracting nearby numbers.
- Definition of algorithms for:
  1. addition/substraction
  2. multiplication/division
  3. square root

- Constraints: implementations should produce the same results as algorithms
- Goal: ensure portability across platform

Bypassing IEEE-754 can be specified to the compiler: faster to the expense of accuracy.

Floating point numbers

— A better idea is to let the comma position “float”.
— Floating-point numbers have constant *relative* accuracy.
— Allows us to represent a much larger range of numbers.



S The sign bit (1 bit).
E The exponent
F The significand (formerly: mantissa)

Floating point numbers

Thus;

$$V = (-1)^S \cdot 2^{E-B} \cdot M$$

where the significand M is defined as

$$M = \underbrace{1}_{\times 2^0} . \overbrace{\underbrace{b_1}_{\times 2^{-1}} \underbrace{b_2}_{\times 2^{-2}} \ldots}^{F}$$

for *normalized* numbers (most common). Here B is the *bias*. Common precisions:

Precision	S	E	F	Total
Single	1	8	23	32
Double	1	11	52	64

The *fractional* part is stored as *sign-magnitude* and the *exponent* as *biased representation*.

Floating point numbers

— The *bias* allows us to give a bias to large or small numbers (large or small exponents).

— Since we use a finite representation, we have a finite precision.

— Smallest and largest numbers:

$$V_{\text{min}} = 1 \cdot 2^{1-127} \cdot 1 = 2^{-126} = 1.17 \ldots 10^{-38}$$
$$V_{\text{max}} = 1 \cdot 2^{254-127} \cdot 2 = 3.40 \ldots 10^{38}.$$

— More important: The smallest *relative* difference between numbers we can represent

$$2^{-23} = 1.19....10^{-7}.$$

i.e. under perfect circumstances we have about 7 digits of accuracy.

— The *denormalization* allows to represent smaller numbers.

Denormalization

The smallest numbers can be represented using the first bit.

Minimum: 2^{1-B-F}

Single precision: $2^{-149} \approx 1.6 \cdot 10^{-45}$ Double precision: $2^{-1074} \approx 5.0 \cdot 10^{-324}$

Floating point operations

— A very useful estimate for the size of a scientific code is the total number of floating point operations performed.

— Floating point operations: $+$, $-$, $\times$, $/$

— A floating point operation is called a FLOP.

— It is tempting to use FLOPS for the plural, but that is *not* the norm. Rather, FLOPS is used about Floating-Point OPeration per Second, which is a very useful performance metric.

— 3 FLOPS or 1 FLOPS

Floating point limitations

— Finite precision of the number representation leads to accuracy issues: the *condition* measure the error propagation for a given function.

— Subtraction: $1.2345 \cdot 10^4$ minus $1.2344 \cdot 10^4$.

$$
\begin{aligned}
& 1.2345 \cdot 10^4 - 1.2344 \cdot 10^4 \\
&= (1.2345 - 1.2344) \cdot 10^4 \\
&= 0.0001 \cdot 10^4 \\
&= 1.0000 \cdot 10^0
\end{aligned}
$$

We have only a single digit of accuracy! This is called *cancellation*, due to the *truncation* error.

— Example where this matters: Approximations of derivatives

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{1}{h}\left(f(x+h) - f(x)\right)$$

Floating point limitations

— Addition: $1.2345 \cdot 10^4$ plus $1.0000 \cdot 10^0$.

$$\begin{aligned} & 1.2345 \cdot 10^4 + 1.0000 \cdot 10^0 \\ &= (1.2345 + 0.0001) \cdot 10^4 \\ &= 1.2346 \cdot 10^4 \end{aligned}$$

OK!

— Addition: $1.2345 \cdot 10^4$ plus $1.0000 \cdot 10^{-1}$.

$$\begin{aligned} & 1.2345 \cdot 10^4 + 1.0000 \cdot 10^{-1} \\ &= (1.2345 + 0.0000) \cdot 10^4 \\ &= 1.2345 \cdot 10^4 \end{aligned}$$

Adding the small number has no effect: the number of *significant* digits is crucial.

Data models

Model	`char`	`short`	`int`	`long`	`long long`	`ptr`	Systems
ILP32	8	16	32	32	N/A (64)	32	UNIX/Linux
LP32	8	16	16	32	N/A (64)	32	Windows/Mac
LP64	8	16	32	64	64	64	UNIX/Linux/Mac
LLP64	8	16	32	64	64	64	Windows

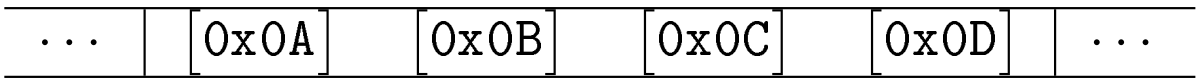
— Exotic platform may use alternate models (ILP64, SILP64).

— Important for portability: avoid invalid assumption regarding conversion.

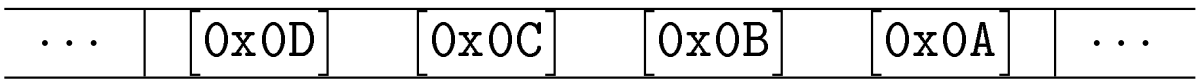
Endianness

Integral data types are stored packed by *bytes*, 2 options.
The binary word can be stored with bytes from *left-to-right* or from *right-to-left*.

Big-Endian: Most-Significant Byte is first:



Little-Endian: Least-Significant Byte is first:



This is especially important when dealing with IO formats.

Outline

Data representation

Single processor system

Performance

Ideal single processor model

In this part a processor is a conceptual unit able to load data, perform computations and store the result.

The Von Neumann architecture:

— Processor and Memory are connected by a Bus.
— Memory contains programs and data (i.e no Harvard which splits them).

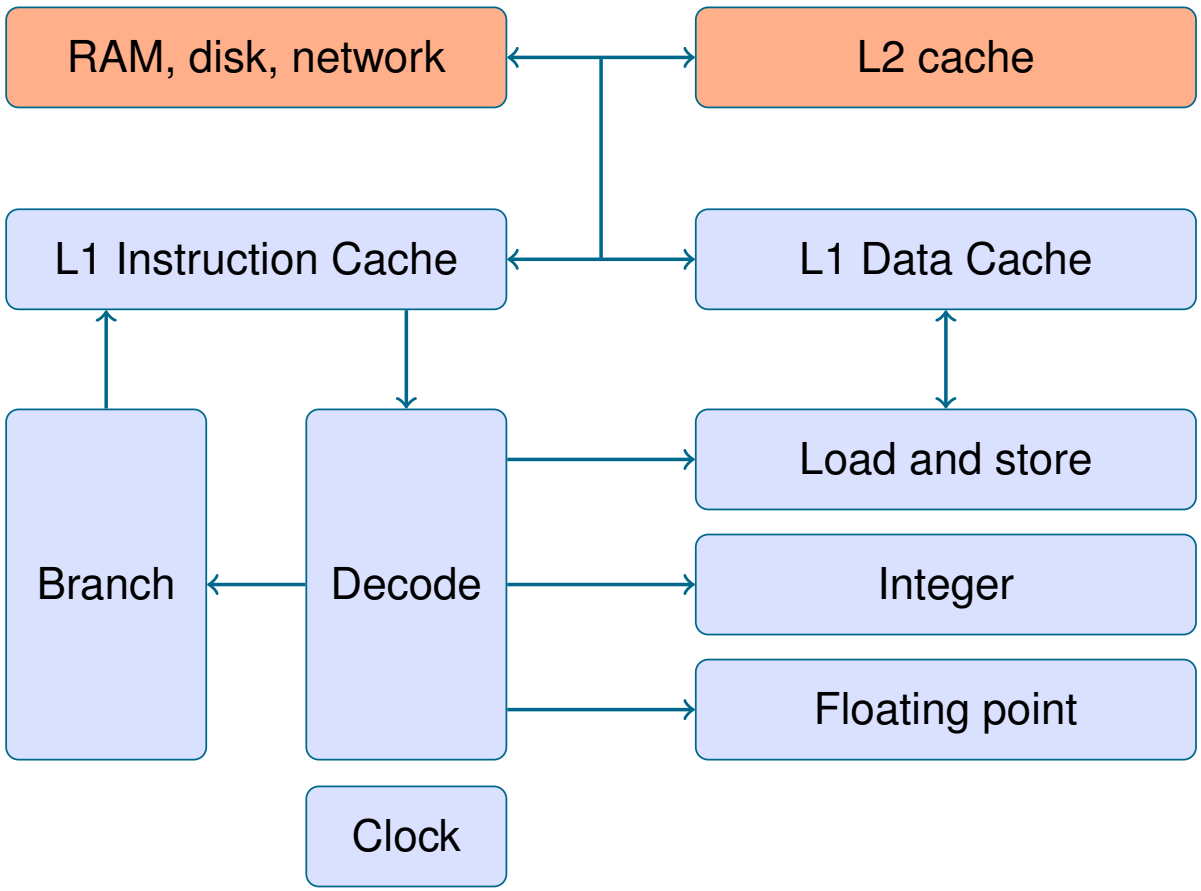
Real models deviate from this idealized model.

$\rightarrow$ Detailed knowledge of the hardware is crucial.

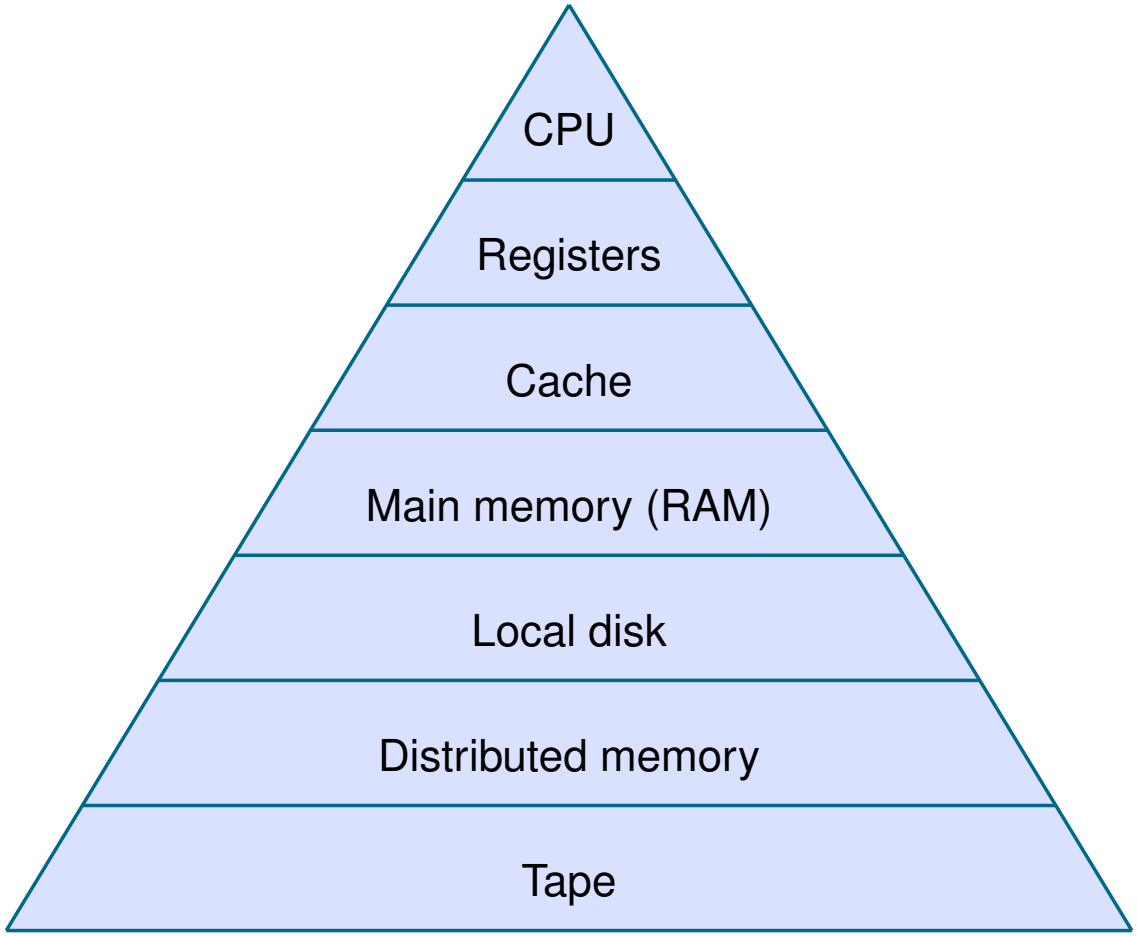
Three basics types: *processors*, *memory*, *connections*

$\rightarrow$ Each categories has performance improvement techniques: pipelining, block caching, patterns.

Single processor systems: RISC architecture



Single processor systems: memory hierarchy



Single processor systems: memory hierarchy

Typical memory access times for the MIPS R14000 processor. The numbers represent number of clock cycles.

Memory type	Clock cycles
Registers	1
L1 cache	2–3
L2 cache	10–12
Main memory	100–200
Message passing	$\mathcal{O}(10^3)$–$\mathcal{O}(10^4)$
Local disk	$\mathcal{O}(10^6)$

Single processor systems: memory hierarchy

Locality is crucial to achieve performance, it is usually based on heuristics.

1. Temporal locality: how likely will we need the data in the future?
2. Spatial locality: how likely will we need the data in the block of memory?

Memory stall cycles: number of cycles during which the processor is stalled, waiting for a memory access.

Caching is an improvement to the Von Neumann model: higher speed of smaller memory, keep some data in a fast memory if it is reused.

Pitfalls

The increasing gap between processor clock frequencies and the lower-level memory makes the processor *starve* for instructions.

Techniques were developed to circumvent this issue.

Outline

Data representation

Single processor system

Performance

Different types of parallelism

1. Memory-Level Parallelism (MLP): several pending memory operations.

2. Instruction-Level Parallelism (ILP): how many instruction can be scheduled simultaneously.

3. Data-Level Parallelism (DLP): execute one instruction on multiple data.

4. Thread-Level Parallelism (DLP): executing several threads of computation on the same processor.

The main notion to understand the bottlenecks is *data dependencies*.

Performance factors

At the single processor level, several techniques to takes advantage of *Instruction-Level Parallelism* (ILP):

- instruction pipelining,
- superscalar execution,
- branch prediction,
- speculative execution,
- vectorization (SIMD),
- prefetching.

Vectorization using additional *vector units* (SIMD) can exploit *Data parallelism*.

Pipelining

Multiple instructions are overlapped in execution: take advantage of parallelism existing between actions required to execute an instruction.

View of an assembly line: different task performed by different operators.

Prototypical Five-Stage RISC:

IF	Instruction-Fetch	
ID	Instruction-Decode	
EX	Execute	
MEM	Memory Access	Load/Store
WR	Write in Register	Register-Register, ALU, Load

The theoretical speed-up for a *N*–stage is *N*, but:

- Overhead for controlling the pipeline.
- Lowest common denominator: speed is limited to the slowest element.
- Hazards: data dependencies, branch misprediction.

Pipelining/Vectorization

Hockney formula: time to operate on vectors of size N

$$t_N = (s + l + (N - 1))\tau$$

1. s: startup phase, time to prepare the pip and fetch operands
2. l: number of cycles to fill the pipe
3. τ: time for one clock cycle

Operation rate: $r = n/t_n$ number of operations per unit time.

Pipelining/Vectorization

Consider the fused scalar multiplication and vector addition:

$$\boldsymbol{c} = \boldsymbol{a} + \gamma \boldsymbol{b}$$

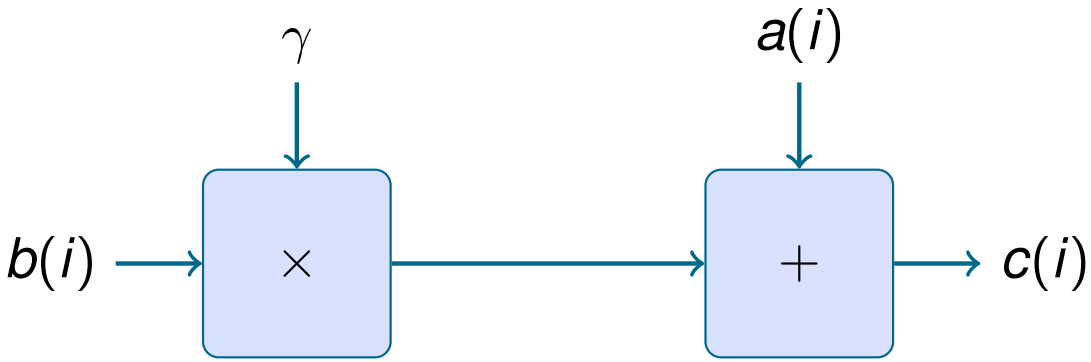
We can write this operation as the loop

```
for i=1,n
  c(i) = a(i) + gamma * b(i)
end
```

A single iteration is performed in a certain number of stages. These stages can be viewed as a pipeline.

Superscalar operations

$$\boldsymbol{c} = \boldsymbol{a} + \gamma \boldsymbol{b}$$



Fused multiply and add (FMADD)

Vectorization

Modern processors contain vector (SIMD, Single Instruction Multiple Data) units. This allows to apply the same operation to multiple data in parallel. In modern Intel (Sandy Bridge, Haswell) chips, the vector units are also superscalar (both AVX and SSE).
Three ways to enable SIMD usage in your programs:

- autovectorizing compilers (ICC),
- hand-written assembly,
- intrinsics (built-in functions)

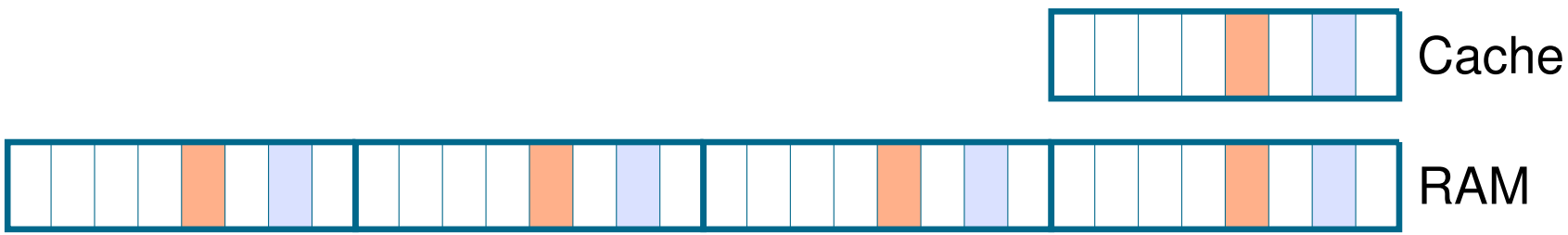
Caching techniques

Memory latency is a bottleneck: use of memory hierarchy to take advantage of *space locality* and *time locality*.

Scientific computing involving:

- long data streams,
- very few conditionals,
- regular memory access patterns

is suitable for performance improvement by caching.



$$\text{Memory address} = \underbrace{b_1 \ldots b_k}_{\text{tag bits}} \underbrace{b_{k+1} \ldots b_N}_{\text{cache address}}.$$

Caching techniques

Problem: the memory latency has increased relative to the CPU frequency, moving data from/to the memory is expensive relative to computational cost on the CPU.

1. Cache hit: the processor find the data in the cache
2. Cache miss: the processor does not find the data in the cache

Cache miss penalty depends on:

1. the memory latency: time to access the first block.
2. the memory bandwidth: time of transfer per unit block.

Using locality to buffer reuse of reoccurring items.

Caching techniques

Different cache strategies based on different implementations of:

1. block placement: where is the cached data stored?
2. block identification: how is the data found in the cache?
3. block replacement: how to decide which block should be replaced?
4. write strategy: if data is changed, how to write to the main memory?

Caching techniques

Block placement:

- *direct-mapped*: block can be placed only at one location
- *fully associative*: block can be placed anywhere
- *set associative*: block can be placed in a restricted set of places, “n-way associative” means that nw

Block identification: address tag to find the block.

Caching techniques

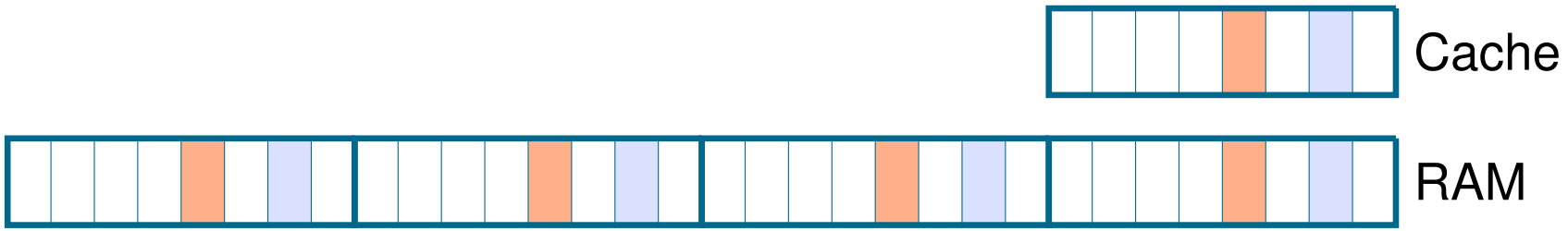
Block replacement:

- *Random*: block replaced is chosen randomly
- *FIFO*: First-In-First-Out
- *LRU*: Least-Recently-Used
- *pseudo-LRU*: Least-Recently-Used with heuristics

Write strategy:

- *Write-though*: write to main memory directly
- *Write-back*: write a next synchronization

Direct mapped cache

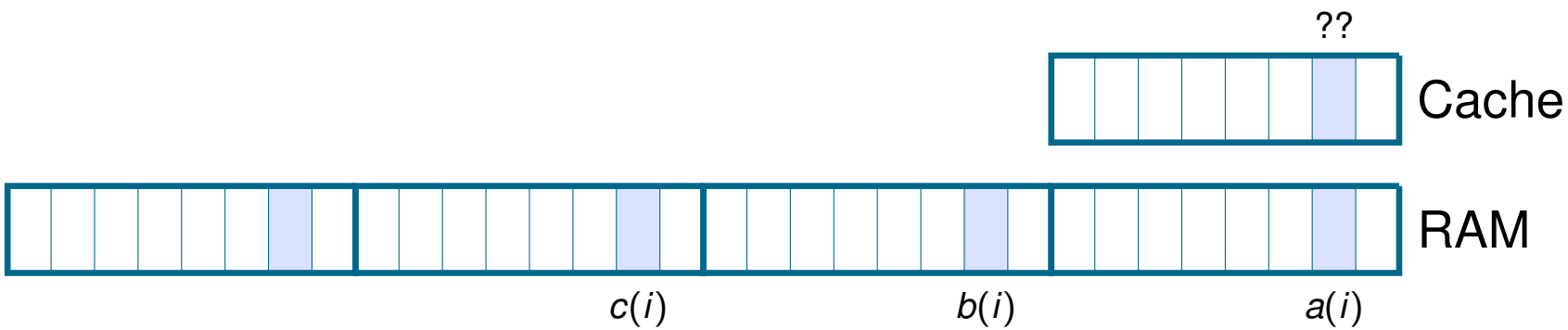


$$\text{Memory address} = \underbrace{b_1 \ldots b_k}_{\text{tag bits}} \underbrace{b_{k+1} \ldots b_N}_{\text{cache address}}.$$

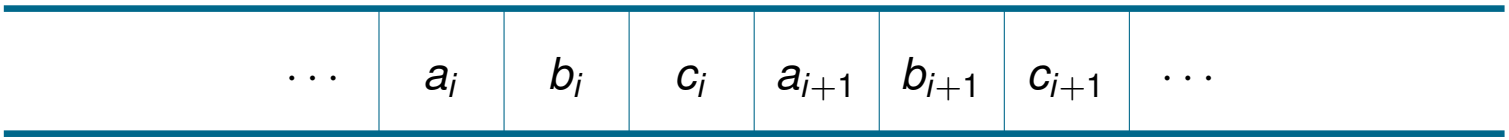
Direct mapped cache

```
for i=1,n
  c(i) = a(i) + b(i)
end
```

If the vectors are precisely the same length as a cache line, we get cache trashing.



Adjacent memory layout



Interleaving the vectors in memory avoids cache trashing.

Other cache strategies

— Fully associative cache: *all* bits in a memory address are used as tag bits, none of them as cache address. All cache addresses are available.

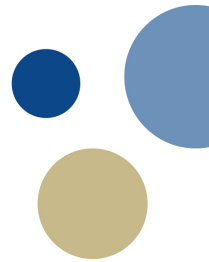
— n-way associative cache: uses m fewer bits for cache address than a direct mapped cache, thus freeing up a choice of $n = 2^m$ cache addresses for each chunk. This is a good compromise between direct mapped cache and fully associative cache

Speculative and Out-of-Order execution

During the execution there may exist dependencies between data such that the process needs to wait for data before proceeding further.

1. *in-order*: pause of execution until data is fetched
2. *out-of-order*: instruction waits but execution of other instructions with non-dependent data is scheduled.

Conclusion



- As the processor clock frequency increasing, memory latency became the bottleneck.
- Many techniques developed to increase concurrency of tasks: instruction parallelism and overlapping of data movement.
- Complexity of single-processors and technological reached a limit calling for other types of parallelism to achieve performance.

Timeframe:

- 1986–2000: the gap between memory and CPU performance increases, development of memory hierarchies.
- 2005: Intel cancels the 4GHz Pentium 4, limits of ILP are reached (hear/power/complexity).
- 2005–Now: Development of multichip–multicore platforms:
  1. NUMA: memory hierarchies improved to ensure coherence and consistency.
  2. TLP: explosion of the thread model.