1 Model for computer architecture

2 Data representation

2.1 Memory layout

Data is stored in binary in the memory:

- the basic unit is the *byte* consisting of 8 bits,
- memory may be addressed with a multiple of *bytes*,
- names: *half-word* 16-bit, *word* 32-bit, *double-word* 32-bit, ...

Conversion from binary to decimal representation:

$$\left[b_k\ b_{k-1}\ \ldots\ldots b_1\ b_0\right]_2 = \sum_{i=0}^{k} b_i\ 2^i$$

A bit or binary digit is a value 0 or 1.

A sequence of N bits is called a binary word, and N is called the width or size.

The word size can be 8, 16, 32, 64.

A byte is a sequence of 8 binary digits.

`char`

$$\left[b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0\right]$$

$$\begin{array}{rc}
0: & \left[0\,0\,0\,0\,0\,0\,0\,0\right] \\
1: & \left[0\,0\,0\,0\,0\,0\,0\,1\right] \\
2: & \left[0\,0\,0\,0\,0\,0\,1\,0\right] \\
3: & \left[0\,0\,0\,0\,0\,0\,1\,1\right] \\
4: & \left[0\,0\,0\,0\,0\,1\,0\,0\right] \\
\ldots & \ldots \\
255: & \left[1\,1\,1\,1\,1\,1\,1\,1\right]
\end{array}$$

The width depends on the specification of the data model.

The width of the word will define the range of numbers that can be represented and will also determine the precision of floating-point numbers.

Data is stored in memory using binary words with a width equal to a powers of two of a byte.

The memory is stored as a contiguous array of bits and these bits can be accessed by taking the address of the entry.

···	0x0A	0x0B	0x0C	0x0D	0x01	0x02	0x03	0x04	···

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

	byte	8-bit
	half-word	16-bit
Common taxonomy:	*word*	32-bit
	double word	32-bit
	quad word	64-bit

Endianness: the binary word can be stored with bytes from *left-to-right* or from *right-to-left*.

Big-Endian: Most-Significant Byte is first:

···	0x0A	0x0B	0x0C	0x0D	···

Little-Endian: Least-Significant Byte is first:

···	0x0D	0x0C	0x0B	0x0A	···

2.2 Integral numbers

Integral numbers consist of a countable finite set, therefore the implementation in binary format can use a given finite number of digits.

Most common memory storage on 32 or 64 bits.

Integers can be signed or unsigned, the default is signed.

Characterized by a range $[A, B]$.

2.2.1 Unsigned values, $\mathbb{N}$

M

2.2.2 Signed values, $\mathbb{Z}$

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

Explicit sign:

S	M

Implicit sign:

V

Representation usually have an explicit sign:

- Sign-and-magnitude,
- One's complement,

- Two's complement,
- Excess-K,

but can also have an implicit sign like Base-2.

Two's complement is the most used.

Sign-and-magnitude

This representation is the intuitive way of expressing signedness.

S	M
1	N-1

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

$$\left[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\right]$$

$$\left[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\right]$$

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

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 $+0$:

$$\left[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\right]$$

$$\left[1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\right]$$

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

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:

$$\left[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\right]$$

Representation of the bounds:

$$\begin{aligned} +127: &\quad \left[0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\right] \\ -127: &\quad \left[1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\right] \\ -128: &\quad \left[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\right] \end{aligned}$$

Motivation: the subtraction can be implemented like an addition since:

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

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

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

2.3 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*.

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

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

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

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

IEEE754: standard defining the properties of floating-point operations.
Definition of:

- guard digits to reduce the error when subtracting two nearby digits,
- algorithms for:
  1. addition
  2. subtraction
  3. multiplication
  4. division
  5. square root

2.3.1 Fixed-point values

S	M	F

2.3.2 Floating-point values

S	E	F

2.3.3 Denormalization

Smallest numbers that can be represented using the first bit.

Minimum: 2^{1-b-f}

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

2.4 Data model

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

2.5 Von Neumann model

2.5.1 Description

Description

2.5.2 Improvements

Caching: higher speed of smaller memory, keep some data in a fast memory if it is reused.

2.5.3 Memory hierarchy

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

Temporal locality: how likely will we need the data in the future?

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.

3 Implementation of a processor

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

4 Different types of parallelism

4.1 Pipelining

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

View of an assembly line:

	IF	Instruction-Fetch	
	ID	Instruction-Decode	
Prototypical Five-Stage RISC:	**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.

The last element *Hazards* comprises:

-

Rule:

- Deeper pipe: increased throughput, but increased penalty.

4.2 Superscalar execution

4.3 Vectorization

4.4 Caching

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.

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?

4.4.1 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

4.4.2 Identification

The block is assigned an address tag to

4.4.3 Replacement

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

4.4.4 Write strategy

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

4.5 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.