Informatics 3.
Lecture X: Bonus

Kristóf Kovács

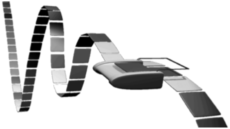
Based on Ferenc Wettl's presentations

Budapest University of Technology and Economics

2024-02-29

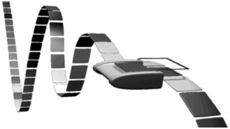
Turing machine

- A Turing machine can be defined by $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, where



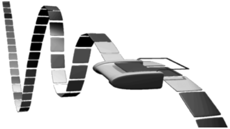
Turing machine

- A Turing machine can be defined by $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, where
- Q is the non-empty set of "states",



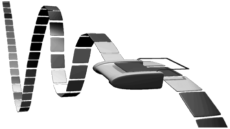
Turing machine

- A Turing machine can be defined by $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, where
- Q is the non-empty set of "states",
- Γ the finite, non-empty "tape alphabet",



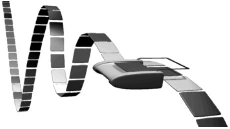
Turing machine

- A Turing machine can be defined by $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, where
- Q is the non-empty set of "states",
- Γ the finite, non-empty "tape alphabet",
- $b \in \Gamma$ the "blank symbol" (the only symbol allowed to occur on the tape infinitely often),



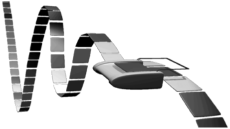
Turing machine

- A Turing machine can be defined by $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, where
- Q is the non-empty set of "states",
- Γ the finite, non-empty "tape alphabet",
- $b \in \Gamma$ the "blank symbol" (the only symbol allowed to occur on the tape infinitely often),
- $\Sigma \subseteq \Gamma \setminus \{b\}$ the set of "input symbols",



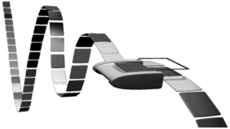
Turing machine

- A Turing machine can be defined by $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, where
- Q is the non-empty set of "states",
- Γ the finite, non-empty "tape alphabet",
- $b \in \Gamma$ the "blank symbol" (the only symbol allowed to occur on the tape infinitely often),
- $\Sigma \subseteq \Gamma \setminus \{b\}$ the set of "input symbols",
- $q_0 \in Q$ the "initial state"



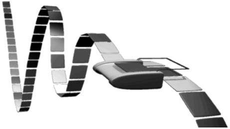
Turing machine

- A Turing machine can be defined by $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, where
- Q is the non-empty set of "states",
- Γ the finite, non-empty "tape alphabet",
- $b \in \Gamma$ the "blank symbol" (the only symbol allowed to occur on the tape infinitely often),
- $\Sigma \subseteq \Gamma \setminus \{b\}$ the set of "input symbols",
- $q_0 \in Q$ the "initial state"
- $F \subseteq Q$ the set of "final states" (this is when the machine stops),



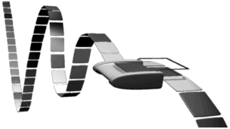
Turing machine

- A Turing machine can be defined by $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, where
- Q is the non-empty set of "states",
- Γ the finite, non-empty "tape alphabet",
- $b \in \Gamma$ the "blank symbol" (the only symbol allowed to occur on the tape infinitely often),
- $\Sigma \subseteq \Gamma \setminus \{b\}$ the set of "input symbols",
- $q_0 \in Q$ the "initial state"
- $F \subseteq Q$ the set of "final states" (this is when the machine stops),
- $\delta : (Q \setminus F) \times \Gamma \hookrightarrow Q \times \Gamma \times \{L, R\}$ is a partial function called the "transition function", where L is left shift, R is right shift (moves the tape)



Turing machine

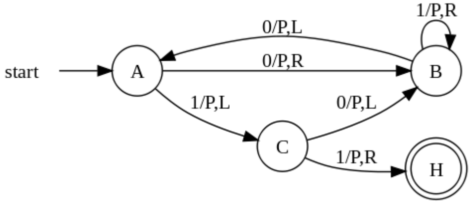
- A Turing machine can be defined by $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$, where
- Q is the non-empty set of "states",
- Γ the finite, non-empty "tape alphabet",
- $b \in \Gamma$ the "blank symbol" (the only symbol allowed to occur on the tape infinitely often),
- $\Sigma \subseteq \Gamma \setminus \{b\}$ the set of "input symbols",
- $q_0 \in Q$ the "initial state"
- $F \subseteq Q$ the set of "final states" (this is when the machine stops),
- $\delta : (Q \setminus F) \times \Gamma \hookrightarrow Q \times \Gamma \times \{L, R\}$ is a partial function called the "transition function", where L is left shift, R is right shift (moves the tape)



H *Church–Turing thesis*: Every formalizable problem, that can be solved with an algorithm can be solved with a Turing-machine.

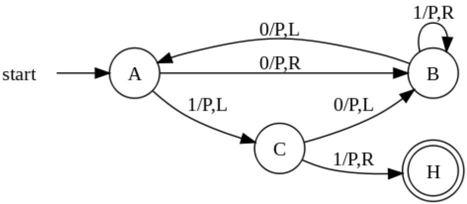
Turing machine

- Busy beaver (Tibor Radó, 1962) The Turing machine that writes the most non-empty symbols on an empty tape, and halts in finite steps.



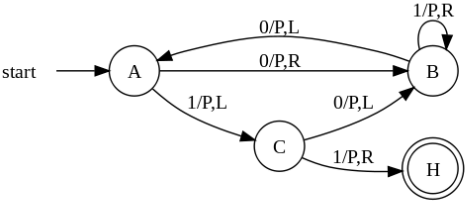
1	A	0 0 0 0 0 |0| 0 0 0 0 0 0
2	B	0 0 0 0 0 |0| 1 0 0 0 0 0
3	A	0 0 0 0 1 |1| 0 0 0 0 0 0
4	C	0 0 0 1 1 |0| 0 0 0 0 0 0
5	B	0 0 1 1 1 |0| 0 0 0 0 0 0
6	A	0 1 1 1 1 |0| 0 0 0 0 0 0
7	B	0 0 1 1 1 |1| 1 0 0 0 0 0
8	B	0 0 0 1 1 |1| 1 1 0 0 0 0
9	B	0 0 0 0 1 |1| 1 1 1 0 0 0
10	B	0 0 0 0 0 |1| 1 1 1 1 0 0
11	B	0 0 0 0 0 |0| 1 1 1 1 1 0
12	A	0 0 0 0 1 |1| 1 1 1 1 0 0
13	C	0 0 0 1 1 |1| 1 1 1 0 0 0
14	H	0 0 0 1 1 |1| 1 1 1 0 0 0

- Busy beaver (Tibor Radó, 1962) The Turing machine that writes the most non-empty symbols on an empty tape, and halts in finite steps.
- $Q = \{\text{A}, \text{B}, \text{C}, \text{HALT}\}$



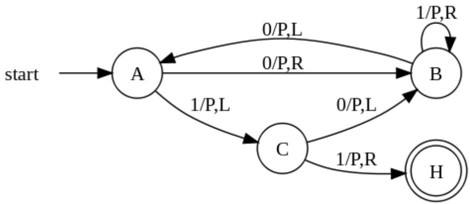
1	A	0 0 0 0 0	0	0 0 0 0 0 0
2	B	0 0 0 0 0	0	1 0 0 0 0 0
3	A	0 0 0 0 1	1	0 0 0 0 0 0
4	C	0 0 0 1 1	0	0 0 0 0 0 0
5	B	0 0 1 1 1	0	0 0 0 0 0 0
6	A	0 1 1 1 1	0	0 0 0 0 0 0
7	B	0 0 1 1 1	1	1 0 0 0 0 0
8	B	0 0 0 1 1	1	1 1 0 0 0 0
9	B	0 0 0 0 1	1	1 1 1 0 0 0
10	B	0 0 0 0 0	1	1 1 1 1 0 0
11	B	0 0 0 0 0	0	1 1 1 1 1 0
12	A	0 0 0 0 1	1	1 1 1 1 0 0
13	C	0 0 0 1 1	1	1 1 1 0 0 0
14	H	0 0 0 1 1	1	1 1 1 0 0 0

- Busy beaver (Tibor Radó, 1962) The Turing machine that writes the most non-empty symbols on an empty tape, and halts in finite steps.
- $Q = \{A, B, C, HALT\}$
- $\Gamma = \{0, 1\}$



1	A	0 0 0 0 0	0	0 0 0 0 0 0
2	B	0 0 0 0 0	0	1 0 0 0 0 0
3	A	0 0 0 0 1	1	0 0 0 0 0 0
4	C	0 0 0 1 1	0	0 0 0 0 0 0
5	B	0 0 1 1 1	0	0 0 0 0 0 0
6	A	0 1 1 1 1	0	0 0 0 0 0 0
7	B	0 0 1 1 1	1	1 0 0 0 0 0
8	B	0 0 0 1 1	1	1 1 0 0 0 0
9	B	0 0 0 0 1	1	1 1 1 0 0 0
10	B	0 0 0 0 0	1	1 1 1 1 0 0
11	B	0 0 0 0 0	0	1 1 1 1 1 0
12	A	0 0 0 0 1	1	1 1 1 1 0 0
13	C	0 0 0 1 1	1	1 1 1 0 0 0
14	H	0 0 0 1 1	1	1 1 1 0 0 0

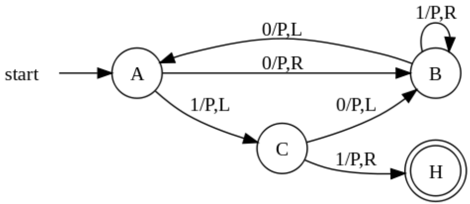
- Busy beaver (Tibor Radó, 1962) The Turing machine that writes the most non-empty symbols on an empty tape, and halts in finite steps.
- $Q = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{HALT}\}$
- $\Gamma = \{0, 1\}$
- $b = 0$ (empty symbol)



1	A	0 0 0 0 0	0	0 0 0 0 0 0
2	B	0 0 0 0 0	0	1 0 0 0 0 0
3	A	0 0 0 0 1	1	0 0 0 0 0 0
4	C	0 0 0 1 1	0	0 0 0 0 0 0
5	B	0 0 1 1 1	0	0 0 0 0 0 0
6	A	0 1 1 1 1	0	0 0 0 0 0 0
7	B	0 0 1 1 1	1	1 0 0 0 0 0
8	B	0 0 0 1 1	1	1 1 0 0 0 0
9	B	0 0 0 0 1	1	1 1 1 0 0 0
10	B	0 0 0 0 0	1	1 1 1 1 0 0
11	B	0 0 0 0 0	0	1 1 1 1 1 0
12	A	0 0 0 0 1	1	1 1 1 1 0 0
13	C	0 0 0 1 1	1	1 1 1 0 0 0
14	H	0 0 0 1 1	1	1 1 1 0 0 0

Turing machine

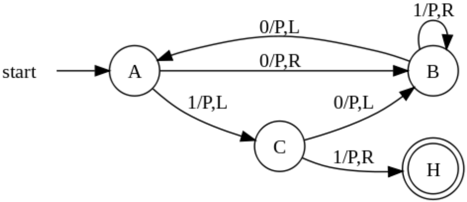
- Busy beaver (Tibor Radó, 1962) The Turing machine that writes the most non-empty symbols on an empty tape, and halts in finite steps.
- $Q = \{A, B, C, HALT\}$
- $\Gamma = \{0, 1\}$
- $b = 0$ (empty symbol)
- $\Sigma = \{1\}$



1	A	0 0 0 0 0 |0| 0 0 0 0 0 0
2	B	0 0 0 0 0 |0| 1 0 0 0 0 0
3	A	0 0 0 0 1 |1| 0 0 0 0 0 0
4	C	0 0 0 1 1 |0| 0 0 0 0 0 0
5	B	0 0 1 1 1 |0| 0 0 0 0 0 0
6	A	0 1 1 1 1 |0| 0 0 0 0 0 0
7	B	0 0 1 1 1 |1| 1 0 0 0 0 0
8	B	0 0 0 1 1 |1| 1 1 0 0 0 0
9	B	0 0 0 0 1 |1| 1 1 1 0 0 0
10	B	0 0 0 0 0 |1| 1 1 1 1 0 0
11	B	0 0 0 0 0 |0| 1 1 1 1 1 0
12	A	0 0 0 0 1 |1| 1 1 1 1 0 0
13	C	0 0 0 1 1 |1| 1 1 1 0 0 0
14	H	0 0 0 1 1 |1| 1 1 1 0 0 0

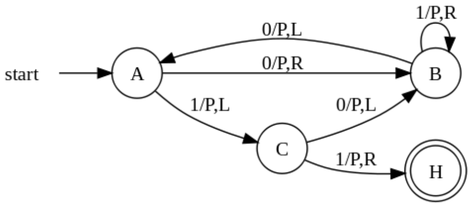
Turing machine

- Busy beaver (Tibor Radó, 1962) The Turing machine that writes the most non-empty symbols on an empty tape, and halts in finite steps.
- $Q = \{A, B, C, HALT\}$
- $\Gamma = \{0, 1\}$
- $b = 0$ (empty symbol)
- $\Sigma = \{1\}$
- $q_0 = A$ (initial state)



1	A	0 0 0 0 0	0	0 0 0 0 0 0
2	B	0 0 0 0 0	0	1 0 0 0 0 0
3	A	0 0 0 0 1	1	0 0 0 0 0 0
4	C	0 0 0 1 1	0	0 0 0 0 0 0
5	B	0 0 1 1 1	0	0 0 0 0 0 0
6	A	0 1 1 1 1	0	0 0 0 0 0 0
7	B	0 0 1 1 1	1	1 0 0 0 0 0
8	B	0 0 0 1 1	1	1 1 0 0 0 0
9	B	0 0 0 0 1	1	1 1 1 0 0 0
10	B	0 0 0 0 0	1	1 1 1 1 0 0
11	B	0 0 0 0 0	0	1 1 1 1 1 0
12	A	0 0 0 0 1	1	1 1 1 1 0 0
13	C	0 0 0 1 1	1	1 1 1 0 0 0
14	H	0 0 0 1 1	1	1 1 1 0 0 0

- Busy beaver (Tibor Radó, 1962) The Turing machine that writes the most non-empty symbols on an empty tape, and halts in finite steps.
- $Q = \{A, B, C, HALT\}$
- $\Gamma = \{0, 1\}$
- $b = 0$ (empty symbol)
- $\Sigma = \{1\}$
- $q_0 = A$ (initial state)
- $F = \{HALT\}$

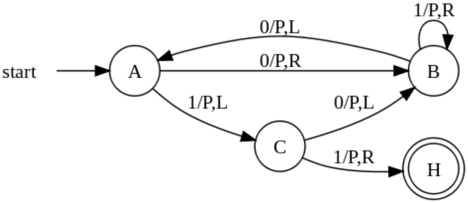


1	A	0 0 0 0 0	0	0 0 0 0 0 0
2	B	0 0 0 0 0	0	1 0 0 0 0 0
3	A	0 0 0 0 1	1	0 0 0 0 0 0
4	C	0 0 0 1 1	0	0 0 0 0 0 0
5	B	0 0 1 1 1	0	0 0 0 0 0 0
6	A	0 1 1 1 1	0	0 0 0 0 0 0
7	B	0 0 1 1 1	1	1 0 0 0 0 0
8	B	0 0 0 1 1	1	1 1 0 0 0 0
9	B	0 0 0 0 1	1	1 1 1 0 0 0
10	B	0 0 0 0 0	1	1 1 1 1 0 0
11	B	0 0 0 0 0	0	1 1 1 1 1 0
12	A	0 0 0 0 1	1	1 1 1 1 0 0
13	C	0 0 0 1 1	1	1 1 1 0 0 0
14	H	0 0 0 1 1	1	1 1 1 0 0 0

Turing machine

- Busy beaver (Tibor Radó, 1962) The Turing machine that writes the most non-empty symbols on an empty tape, and halts in finite steps.
- $Q = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{HALT}\}$
- $\Gamma = \{0, 1\}$
- $b = 0$ (empty symbol)
- $\Sigma = \{1\}$
- $q_0 = \mathrm{A}$ (initial state)
- $F = \{\mathrm{HALT}\}$
- δ table:

	A	B	C
0	1RB	1LA	1LB
1	1LC	1RB	1RH



1	A	0 0 0 0 0	0	0 0 0 0 0 0
2	B	0 0 0 0 0	0	1 0 0 0 0 0
3	A	0 0 0 0 1	1	0 0 0 0 0 0
4	C	0 0 0 1 1	0	0 0 0 0 0 0
5	B	0 0 1 1 1	0	0 0 0 0 0 0
6	A	0 1 1 1 1	0	0 0 0 0 0 0
7	B	0 0 1 1 1	1	1 0 0 0 0 0
8	B	0 0 0 1 1	1	1 1 0 0 0 0
9	B	0 0 0 0 1	1	1 1 1 0 0 0
10	B	0 0 0 0 0	1	1 1 1 1 0 0
11	B	0 0 0 0 0	0	1 1 1 1 1 0
12	A	0 0 0 0 1	1	1 1 1 1 0 0
13	C	0 0 0 1 1	1	1 1 1 0 0 0
14	H	0 0 0 1 1	1	1 1 1 0 0 0

BIOS (Basic Input/Output System)

- What comes before the operating system? How does a computer know from where to load the operating system? How does a computer without an operating system know how to use a monitor or a keyboard?

BIOS (Basic Input/Output System)

- What comes before the operating system? How does a computer know from where to load the operating system? How does a computer without an operating system know how to use a monitor or a keyboard?
- The first thing that comes online once a computer starts is the BIOS.

BIOS (Basic Input/Output System)

- What comes before the operating system? How does a computer know from where to load the operating system? How does a computer without an operating system know how to use a monitor or a keyboard?
- The first thing that comes online once a computer starts is the BIOS.
- This is a minimal system integrated into the motherboard, its main task is to initialize the computer.

BIOS (Basic Input/Output System)

- What comes before the operating system? How does a computer know from where to load the operating system? How does a computer without an operating system know how to use a monitor or a keyboard?
- The first thing that comes online once a computer starts is the BIOS.
- This is a minimal system integrated into the motherboard, its main task is to initialize the computer.
- There are drivers stored inside the BIOS for the use of basic input / output devices (drivers are software that describes to the machine how a component works).

BIOS (Basic Input/Output System)

- What comes before the operating system? How does a computer know from where to load the operating system? How does a computer without an operating system know how to use a monitor or a keyboard?
- The first thing that comes online once a computer starts is the BIOS.
- This is a minimal system integrated into the motherboard, its main task is to initialize the computer.
- There are drivers stored inside the BIOS for the use of basic input / output devices (drivers are software that describes to the machine how a component works).
- The BIOS finds the highest priority storage device and starts to load the operating system.

MBR (Master Boot Record)

- The first step in loading the operating system is when the BIOS reads the first 512 bytes of the chosen device, this is called the MBR.

MBR (Master Boot Record)

- The first step in loading the operating system is when the BIOS reads the first 512 bytes of the chosen device, this is called the MBR.
- The first part of the MBR is a short code (bootstrap code), which describes the steps of starting the operating system.

MBR (Master Boot Record)

- The first step in loading the operating system is when the BIOS reads the first 512 bytes of the chosen device, this is called the MBR.
- The first part of the MBR is a short code (bootstrap code), which describes the steps of starting the operating system.
- The next part is the partition table

MBR (Master Boot Record)

- The first step in loading the operating system is when the BIOS reads the first 512 bytes of the chosen device, this is called the MBR.
- The first part of the MBR is a short code (bootstrap code), which describes the steps of starting the operating system.
- The next part is the partition table
- The third and last part of the MBR is the magical number, which is the same for all computers (0xAA55 = 0b1010101001010101, this is a failsafe, a way to check if the MBR is valid.

MBR (Master Boot Record)

- The first step in loading the operating system is when the BIOS reads the first 512 bytes of the chosen device, this is called the MBR.
- The first part of the MBR is a short code (bootstrap code), which describes the steps of starting the operating system.
- The next part is the partition table
- The third and last part of the MBR is the magical number, which is the same for all computers (0xAA55 = 0b1010101001010101, this is a failsafe, a way to check if the MBR is valid.
- Until this point the starting procedure of the machine is independent of the operating system.

Storage

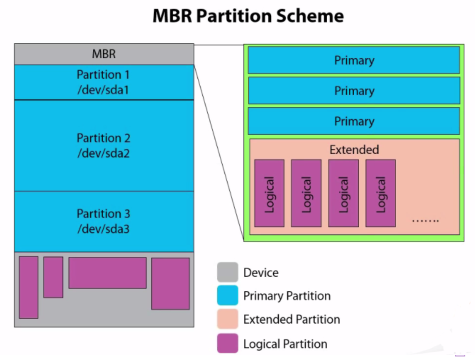
- After the MBR there can be one or more partitions

Storage

- After the MBR there can be one or more partitions
- There can be at most 4 primary partitions.

Storage

- After the MBR there can be one or more partitions
- There can be at most 4 primary partitions.
- It is recommended to install your operating system on a primary partition (Windows can only be installed there).



Extended partition

- The extended partition counts as a primary partition, so there can be at most 3 primary and 1 extended partition on a storage device.

Extended partition

- The extended partition counts as a primary partition, so there can be at most 3 primary and 1 extended partition on a storage device.
- It can contain however many logical partition this is a possible way to have more than 4 partitions.

Extended partition

- The extended partition counts as a primary partition, so there can be at most 3 primary and 1 extended partition on a storage device.
- It can contain however many logical partition this is a possible way to have more than 4 partitions.
- It can only be located at the end of the storage device, no primary partition can follow it.

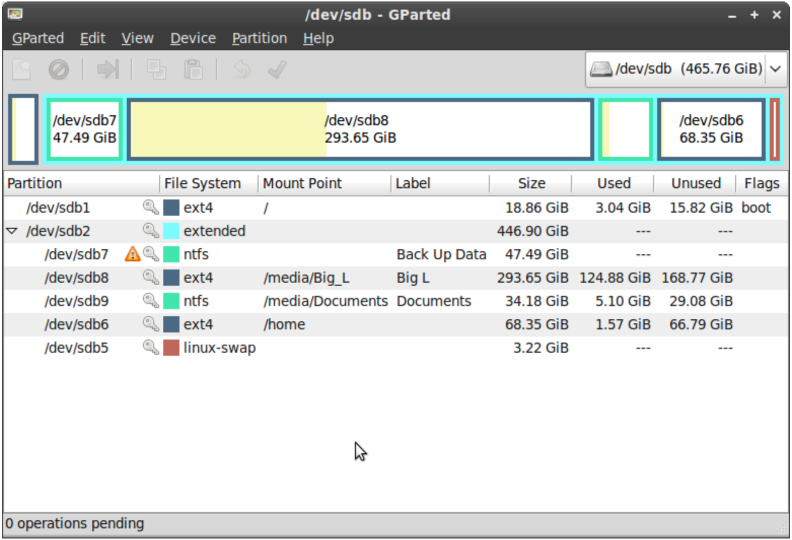
Extended partition

- The extended partition counts as a primary partition, so there can be at most 3 primary and 1 extended partition on a storage device.
- It can contain however many logical partition this is a possible way to have more than 4 partitions.
- It can only be located at the end of the storage device, no primary partition can follow it.
- Windows usually creates a recovery partition on install, which comes before the partition of the operating system, should the operating system fail, it will try to recover itself using this partition.

Extended partition

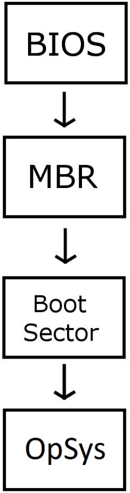
- The extended partition counts as a primary partition, so there can be at most 3 primary and 1 extended partition on a storage device.
- It can contain however many logical partition this is a possible way to have more than 4 partitions.
- It can only be located at the end of the storage device, no primary partition can follow it.
- Windows usually creates a recovery partition on install, which comes before the partition of the operating system, should the operating system fail, it will try to recover itself using this partition.
- Linux uses multiple partitions (usually 4), one of them is the previously mentioned virtual memory. This is where the unused part of the memory can be stored (swapping, paging).

Example for a graphical partition manager



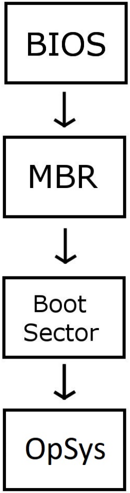
Boot Sector

- At the beginning of every primary partition is a Boot Sector, the MBR stores the location of this sector and this is what starts to load the operating system.



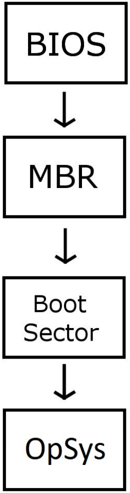
Boot Sector

- At the beginning of every primary partition is a Boot Sector, the MBR stores the location of this sector and this is what starts to load the operating system.
- Similarly to the MBR this is a 512 byte sector as well, which provides the necessary instructions to start the operating system, this stores a magical number as well.



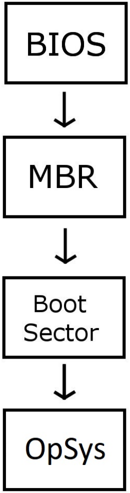
Boot Sector

- At the beginning of every primary partition is a Boot Sector, the MBR stores the location of this sector and this is what starts to load the operating system.
- Similarly to the MBR this is a 512 byte sector as well, which provides the necessary instructions to start the operating system, this stores a magical number as well.
- On linux systems the Boot Sector is actually empty and the operating system is loaded in another way, this is why it is possible to install linux onto a logical partition.



Boot Sector

- At the beginning of every primary partition is a Boot Sector, the MBR stores the location of this sector and this is what starts to load the operating system.
- Similarly to the MBR this is a 512 byte sector as well, which provides the necessary instructions to start the operating system, this stores a magical number as well.
- On linux systems the Boot Sector is actually empty and the operating system is loaded in another way, this is why it is possible to install linux onto a logical partition.
- If the machine's storage device contains more than one operating system and the MBR contains the necessary instructions, then it is possilbe to choose which one to load at every start.



File system

Operating system	WINDOWS	LINUX	MAC	Mobile storage
File system	NTFS	ext4	APFS	FAT32 or NTFS

Files of the operating system

- Operating system (OS): core program, which

Files of the operating system

- Operating system (OS): core program, which
  - directly controls the hardware (memory, peripheries,...),

Files of the operating system

- Operating system (OS): core program, which
  - directly controls the hardware (memory, peripheries,...),
  - provides a unified environment for applications,

Files of the operating system

- Operating system (OS): core program, which
  - directly controls the hardware (memory, peripheries,...),
  - provides a unified environment for applications,
  - organizes the execution of these applications,

Files of the operating system

- Operating system (OS): core program, which
  - directly controls the hardware (memory, peripheries,...),
  - provides a unified environment for applications,
  - organizes the execution of these applications,
  - handles possible program failures,

Files of the operating system

- Operating system (OS): core program, which
  - directly controls the hardware (memory, peripheries,...),
  - provides a unified environment for applications,
  - organizes the execution of these applications,
  - handles possible program failures,
  - handles files,

Files of the operating system

- Operating system (OS): core program, which
  - directly controls the hardware (memory, peripheries,...),
  - provides a unified environment for applications,
  - organizes the execution of these applications,
  - handles possible program failures,
  - handles files,
  - provides basic protection to the machine,

Files of the operating system

- Operating system (OS): core program, which
  - directly controls the hardware (memory, peripheries,...),
  - provides a unified environment for applications,
  - organizes the execution of these applications,
  - handles possible program failures,
  - handles files,
  - provides basic protection to the machine,
  - logs important operation events...

Files of the operating system

- Operating system (OS): core program, which
  - directly controls the hardware (memory, peripheries,...),
  - provides a unified environment for applications,
  - organizes the execution of these applications,
  - handles possible program failures,
  - handles files,
  - provides basic protection to the machine,
  - logs important operation events...
- The OS is part of the system programs

Files of the operating system

- Operating system (OS): core program, which
  - directly controls the hardware (memory, peripheries,...),
  - provides a unified environment for applications,
  - organizes the execution of these applications,
  - handles possible program failures,
  - handles files,
  - provides basic protection to the machine,
  - logs important operation events...
- The OS is part of the system programs
- Other system programs for example are anti-viruses, file compressors, file encrypters, file explorers, network programs, task manager...

Types of operating systems

- single-, multi-user

Types of operating systems

- single-, multi-user
- single-, multi-tasking

Types of operating systems

- single-, multi-user
- single-, multi-tasking
- distributed (cloud),

Types of operating systems

- single-, multi-user
- single-, multi-tasking
- distributed (cloud),
- embedded (for small machines, with limited resources)

Types of operating systems

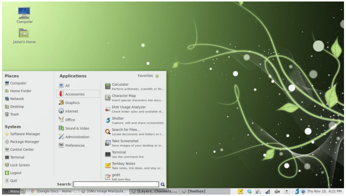
- single-, multi-user
- single-, multi-tasking
- distributed (cloud),
- embedded (for small machines, with limited resources)
- by its role: personal, server,. . .

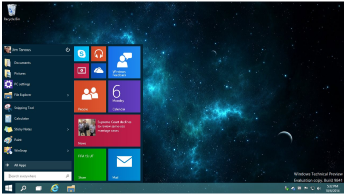
Types of operating systems

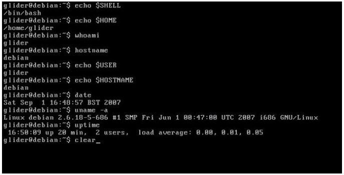
- single-, multi-user
- single-, multi-tasking
- distributed (cloud),
- embedded (for small machines, with limited resources)
- by its role: personal, server,. . .
- by the step of memory addressing 32- or 64 bits (processors themselves use 32 or 64 bits, in essence they either use numbers stored on 32 bits or 64 bits)

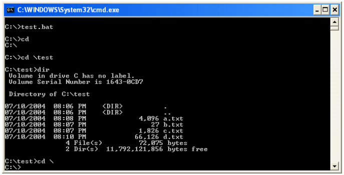
Two important part of operating systems

- Kernel: provides basic control over the hardware, organizes the resources required by the running programs.



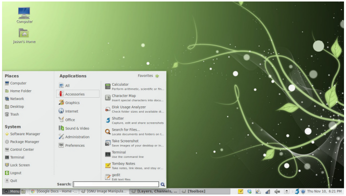


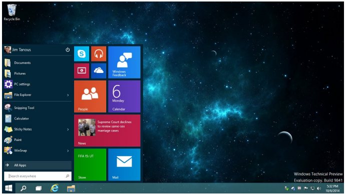


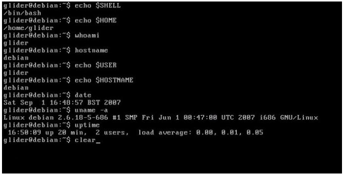


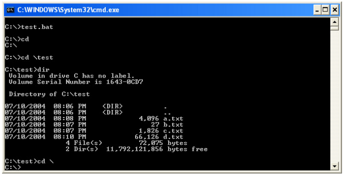
Two important part of operating systems

- Kernel: provides basic control over the hardware, organizes the resources required by the running programs.
- Shell: the user interface to the system. It can be graphical or command bases.

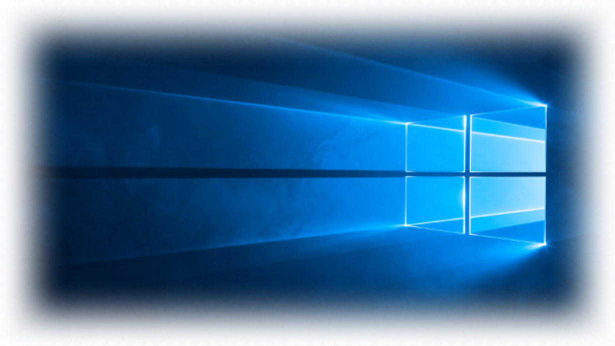






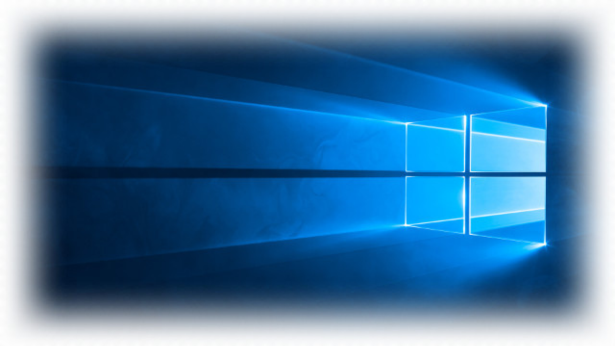


Windows summary



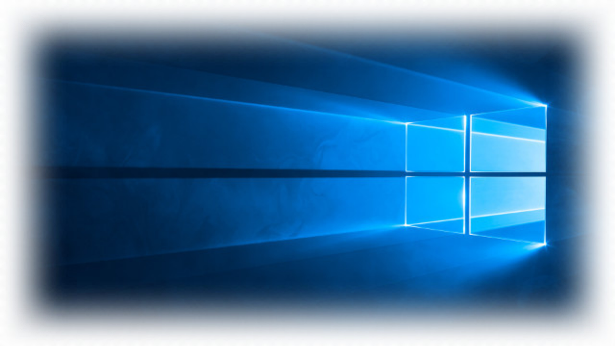
- File system: NTFS

Windows summary



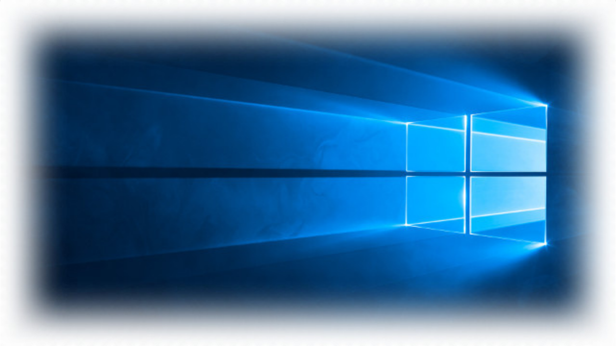
- File system: NTFS
- Source code: closed

Windows summary



- File system: NTFS
- Source code: closed
- Used on most public computers

Windows summary



- File system: NTFS
- Source code: closed
- Used on most public computers
- Developed in batches, there is always an actively developed branch (Windows 11), while the older verions only get smaller fixes and security updates (Windows 8.1, 10), or nothing at all (Windows XP)

Linux summary

- File system: ext4

Linux summary

- File system: ext4
- Source code: open

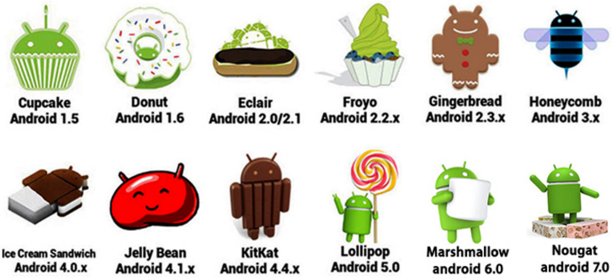
Linux summary

- File system: ext4
- Source code: open
- Most widespread on servers, but also used on personal computers

Linux summary

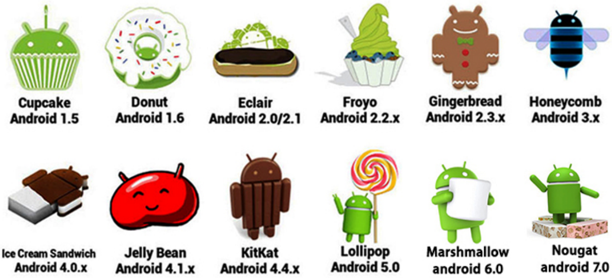
- File system: ext4
- Source code: open
- Most widespread on servers, but also used on personal computers
- Development is on multiple branches, there are a number of different distributions, there are branches specialized for research or programming (SUSE) and there are those for simple users (Linux Mint, Ubuntu).

Android summary



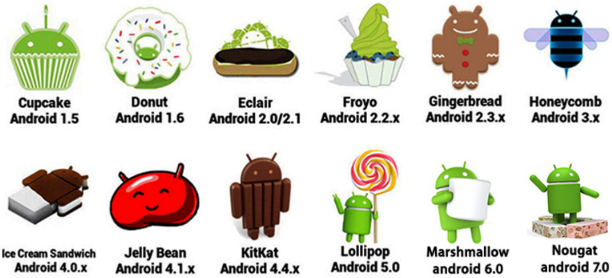
- File system: varies, optimized for flash memory: yaffs2, vfat (SD-card), (Samsung: Flash-Friendly File System f2fs),...

Android summary



- File system: varies, optimized for flash memory: yaffs2, vfat (SD-card), (Samsung: Flash-Friendly File System f2fs),...
- Source code: open

Android summary



- File system: varies, optimized for flash memory: yaffs2, vfat (SD-card), (Samsung: Flash-Friendly File System f2fs),. . .
- Source code: open
- Mostly used on mobile phones, tablets, smart watches, TVs, cars,. . .

Network – IP address

- Machines connected to the internet are addressed by a unique IP address

Network – IP address

- Machines connected to the internet are addressed by a unique IP address
  - IPv4 standard: format: `nnn.nnn.nnn.nnn` (32 bits, 4 number of 8-bit numbers in decimal format) – it already ran out

Network – IP address

- Machines connected to the internet are addressed by a unique IP address
  - IPv4 standard: format: `nnn.nnn.nnn.nnn` (32 bits, 4 number of 8-bit numbers in decimal format) – it already ran out
  - IPv6 standard: format: xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx (128 bits, 8 number of 16 bits in hexadecimal format)

Network – IP address

- Machines connected to the internet are addressed by a unique IP address
  - IPv4 standard: format: `nnn.nnn.nnn.nnn` (32 bits, 4 number of 8-bit numbers in decimal format) – it already ran out
  - IPv6 standard: format: xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx (128 bits, 8 number of 16 bits in hexadecimal format)

machine	IP address	how to find out?
local network	172.17.148.238	ifconfig (WIN ipconfig)
	192.168.xxx.xxx	Reserved IP addresses
outside IPv4:	152.66.83.241	https://www.whatismyip.com/
		http://www.howtofindmyipaddress.com/
IPv6:	2001:738:2001:2010:891b:efb:2b36:5447	
		http://whatismyipaddress.com/
server	152.66.83.17	ping leibniz.math.bme.hu

Ping

- ping is a system utility, it provides a means to check if a data package reaches its destination.

```
C:\Users\Tofi>ping bme.hu

Pinging bme.hu [152.66.115.203] with 32 bytes of data:
Reply from 152.66.115.203: bytes=32 time=66ms TTL=52
Reply from 152.66.115.203: bytes=32 time=69ms TTL=52
Reply from 152.66.115.203: bytes=32 time=73ms TTL=52
Reply from 152.66.115.203: bytes=32 time=62ms TTL=52

Ping statistics for 152.66.115.203:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 62ms, Maximum = 73ms, Average = 67ms

C:\Users\Tofi>_
```

Ping

- ping is a system utility, it provides a means to check if a data package reaches its destination.
- If the ping command is followed by something other than an IP address it will find the IP address paired with that host name using the DNS (Domain Name System)

```
C:\Users\Tofi>ping bme.hu

Pinging bme.hu [152.66.115.203] with 32 bytes of data:
Reply from 152.66.115.203: bytes=32 time=66ms TTL=52
Reply from 152.66.115.203: bytes=32 time=69ms TTL=52
Reply from 152.66.115.203: bytes=32 time=73ms TTL=52
Reply from 152.66.115.203: bytes=32 time=62ms TTL=52

Ping statistics for 152.66.115.203:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 62ms, Maximum = 73ms, Average = 67ms

C:\Users\Tofi>_
```

Ping

- ping is a system utility, it provides a means to check if a data package reaches its destination.
- If the ping command is followed by something other than an IP address it will find the IP address paired with that host name using the DNS (Domain Name System)
- PING means "Send a packet to a computer and wait for its return (Packet INternet Groper)"

```
C:\Users\Tofi>ping bme.hu

Pinging bme.hu [152.66.115.203] with 32 bytes of data:
Reply from 152.66.115.203: bytes=32 time=66ms TTL=52
Reply from 152.66.115.203: bytes=32 time=69ms TTL=52
Reply from 152.66.115.203: bytes=32 time=73ms TTL=52
Reply from 152.66.115.203: bytes=32 time=62ms TTL=52

Ping statistics for 152.66.115.203:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 62ms, Maximum = 73ms, Average = 67ms

C:\Users\Tofi>_
```

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \ldots b_1 b_0 . b_{-1} \ldots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 =$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \ldots b_1 b_0 . b_{-1} \ldots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$106 = 2 \cdot 53 + 0$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$106 = 2 \cdot 53 + 0 \rightarrow 0$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$106 = 2 \cdot 53 + 0 \to 0$

$53 = 2 \cdot 26 + 1$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$106 = 2 \cdot 53 + 0 \rightarrow 0$

$53 = 2 \cdot 26 + 1 \rightarrow 1$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$106 = 2 \cdot 53 + 0 \rightarrow 0$

$53 = 2 \cdot 26 + 1 \rightarrow 1$

$26 = 2 \cdot 13 + 0$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$106 = 2 \cdot 53 + 0 \to 0$

$53 = 2 \cdot 26 + 1 \to 1$

$26 = 2 \cdot 13 + 0 \to 0$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$106 = 2 \cdot 53 + 0 \to 0$

$53 = 2 \cdot 26 + 1 \to 1$

$26 = 2 \cdot 13 + 0 \to 0$

$13 = 2 \cdot \ \ 6 + 1$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned} 106 &= 2 \cdot 53 + 0 \rightarrow 0 \\ 53 &= 2 \cdot 26 + 1 \rightarrow 1 \\ 26 &= 2 \cdot 13 + 0 \rightarrow 0 \\ 13 &= 2 \cdot \ \ 6 + 1 \rightarrow 1 \end{aligned}$$

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$106 = 2 \cdot 53 + 0 \rightarrow 0$$
$$53 = 2 \cdot 26 + 1 \rightarrow 1$$
$$26 = 2 \cdot 13 + 0 \rightarrow 0$$
$$13 = 2 \cdot \ \ 6 + 1 \rightarrow 1$$
$$6 = 2 \cdot \ \ 3 + 0$$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \ldots b_1 b_0 . b_{-1} \ldots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned} 106 &= 2 \cdot 53 + 0 \to 0 \\ 53 &= 2 \cdot 26 + 1 \to 1 \\ 26 &= 2 \cdot 13 + 0 \to 0 \\ 13 &= 2 \cdot 6 + 1 \to 1 \\ 6 &= 2 \cdot 3 + 0 \to 0 \end{aligned}$$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned}
106 &= 2 \cdot 53 + 0 \to 0 \\
53 &= 2 \cdot 26 + 1 \to 1 \\
26 &= 2 \cdot 13 + 0 \to 0 \\
13 &= 2 \cdot 6 + 1 \to 1 \\
6 &= 2 \cdot 3 + 0 \to 0 \\
3 &= 2 \cdot 1 + 1
\end{aligned}$$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned}
106 &= 2 \cdot 53 + 0 \to 0 \\
53 &= 2 \cdot 26 + 1 \to 1 \\
26 &= 2 \cdot 13 + 0 \to 0 \\
13 &= 2 \cdot 6 + 1 \to 1 \\
6 &= 2 \cdot 3 + 0 \to 0 \\
3 &= 2 \cdot 1 + 1 \to 1
\end{aligned}$$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \ldots b_1 b_0 . b_{-1} \ldots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned}
106 &= 2 \cdot 53 + 0 \rightarrow 0 \\
53 &= 2 \cdot 26 + 1 \rightarrow 1 \\
26 &= 2 \cdot 13 + 0 \rightarrow 0 \\
13 &= 2 \cdot 6 + 1 \rightarrow 1 \\
6 &= 2 \cdot 3 + 0 \rightarrow 0 \\
3 &= 2 \cdot 1 + 1 \rightarrow 1 \\
1 &= 2 \cdot 0 + 1
\end{aligned}$$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned}
106 &= 2 \cdot 53 + 0 \to 0\\
53 &= 2 \cdot 26 + 1 \to 1\\
26 &= 2 \cdot 13 + 0 \to 0\\
13 &= 2 \cdot 6 + 1 \to 1\\
6 &= 2 \cdot 3 + 0 \to 0\\
3 &= 2 \cdot 1 + 1 \to 1\\
1 &= 2 \cdot 0 + 1 \to 1
\end{aligned}$$

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned}
106 &= 2 \cdot 53 + 0 \to 0\\
53 &= 2 \cdot 26 + 1 \to 1\\
26 &= 2 \cdot 13 + 0 \to 0\\
13 &= 2 \cdot \ \ 6 + 1 \to 1\\
6 &= 2 \cdot \ \ 3 + 0 \to 0\\
3 &= 2 \cdot \ \ 1 + 1 \to 1\\
1 &= 2 \cdot \ \ 0 + 1 \to 1
\end{aligned}$$

so the binary form is 1101010.

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$106 = 2 \cdot 53 + 0 \rightarrow 0$

$53 = 2 \cdot 26 + 1 \rightarrow 1$

$26 = 2 \cdot 13 + 0 \rightarrow 0$

$13 = 2 \cdot 6 + 1 \rightarrow 1$

$6 = 2 \cdot 3 + 0 \rightarrow 0$

$3 = 2 \cdot 1 + 1 \rightarrow 1$

$1 = 2 \cdot 0 + 1 \rightarrow 1$

106	2
53	0

so the binary form is 1101010.

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i.$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned}
106 &= 2 \cdot 53 + 0 \to 0\\
53 &= 2 \cdot 26 + 1 \to 1\\
26 &= 2 \cdot 13 + 0 \to 0\\
13 &= 2 \cdot 6 + 1 \to 1\\
6 &= 2 \cdot 3 + 0 \to 0\\
3 &= 2 \cdot 1 + 1 \to 1\\
1 &= 2 \cdot 0 + 1 \to 1
\end{aligned}$$

106	2
53	0
26	1

so the binary form is 1101010.

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned} 106 &= 2 \cdot 53 + 0 \to 0 \\ 53 &= 2 \cdot 26 + 1 \to 1 \\ 26 &= 2 \cdot 13 + 0 \to 0 \\ 13 &= 2 \cdot \ \ 6 + 1 \to 1 \\ 6 &= 2 \cdot \ \ 3 + 0 \to 0 \\ 3 &= 2 \cdot \ \ 1 + 1 \to 1 \\ 1 &= 2 \cdot \ \ 0 + 1 \to 1 \end{aligned}$$

106	2
53	0
26	1
13	0

so the binary form is 1101010.

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned} 106 &= 2 \cdot 53 + 0 \rightarrow 0 \\ 53 &= 2 \cdot 26 + 1 \rightarrow 1 \\ 26 &= 2 \cdot 13 + 0 \rightarrow 0 \\ 13 &= 2 \cdot \ \ 6 + 1 \rightarrow 1 \\ 6 &= 2 \cdot \ \ 3 + 0 \rightarrow 0 \\ 3 &= 2 \cdot \ \ 1 + 1 \rightarrow 1 \\ 1 &= 2 \cdot \ \ 0 + 1 \rightarrow 1 \end{aligned}$$

106	2
53	0
26	1
13	0
6	1

so the binary form is 1101010.

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned} 106 &= 2 \cdot 53 + 0 \to 0 \\ 53 &= 2 \cdot 26 + 1 \to 1 \\ 26 &= 2 \cdot 13 + 0 \to 0 \\ 13 &= 2 \cdot 6 + 1 \to 1 \\ 6 &= 2 \cdot 3 + 0 \to 0 \\ 3 &= 2 \cdot 1 + 1 \to 1 \\ 1 &= 2 \cdot 0 + 1 \to 1 \end{aligned}$$

106	2
53	0
26	1
13	0
6	1
3	0

so the binary form is 1101010.

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned} 106 &= 2 \cdot 53 + 0 \to 0 \\ 53 &= 2 \cdot 26 + 1 \to 1 \\ 26 &= 2 \cdot 13 + 0 \to 0 \\ 13 &= 2 \cdot \ \ 6 + 1 \to 1 \\ 6 &= 2 \cdot \ \ 3 + 0 \to 0 \\ 3 &= 2 \cdot \ \ 1 + 1 \to 1 \\ 1 &= 2 \cdot \ \ 0 + 1 \to 1 \end{aligned}$$

106	2
53	0
26	1
13	0
6	1
3	0
1	1

so the binary form is 1101010.

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned} 106 &= 2 \cdot 53 + 0 \to 0 \\ 53 &= 2 \cdot 26 + 1 \to 1 \\ 26 &= 2 \cdot 13 + 0 \to 0 \\ 13 &= 2 \cdot \ \ 6 + 1 \to 1 \\ 6 &= 2 \cdot \ \ 3 + 0 \to 0 \\ 3 &= 2 \cdot \ \ 1 + 1 \to 1 \\ 1 &= 2 \cdot \ \ 0 + 1 \to 1 \end{aligned}$$

so the binary form is 1101010.

106	2
53	0
26	1
13	0
6	1
3	0
1	1
0	1

Binary numbers

Conversion from base 2 to base 10:

$$b_n b_{n-1} \dots b_1 b_0 . b_{-1} \dots b_{-m} = \sum_{i=-m}^{n} b_i 2^i .$$

For example $110.101_2 = 6.625$

Conversion from base 10 to base 2

- for integers repeated division by 2,
- for the fractional parts repeated multiplication by 2.

For example 106 in base 2:

$$\begin{aligned} 106 &= 2 \cdot 53 + 0 \to 0 \\ 53 &= 2 \cdot 26 + 1 \to 1 \\ 26 &= 2 \cdot 13 + 0 \to 0 \\ 13 &= 2 \cdot \ \ 6 + 1 \to 1 \\ 6 &= 2 \cdot \ \ 3 + 0 \to 0 \\ 3 &= 2 \cdot \ \ 1 + 1 \to 1 \\ 1 &= 2 \cdot \ \ 0 + 1 \to 1 \end{aligned}$$

so the binary form is 1101010.

106	2
53	0
26	1
13	0
6	1
3	0
1	1
0	1

Example

How to convert a fractional number into binary?

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1.

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, 1/2, 1/4,. . . , $1/2^n$,. . . . For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:
$0.3 \cdot 2 = 0.6$

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \to 0$

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \to 0$

$0.6 \cdot 2 = 1.2$

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$

$0.6 \cdot 2 = 1.2 \rightarrow 1$

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$

$0.6 \cdot 2 = 1.2 \rightarrow 1$

$0.2 \cdot 2 = 0.4$

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \to 0$

$0.6 \cdot 2 = 1.2 \to 1$

$0.2 \cdot 2 = 0.4 \to 0$

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$

$0.6 \cdot 2 = 1.2 \rightarrow 1$

$0.2 \cdot 2 = 0.4 \rightarrow 0$

$0.4 \cdot 2 = 0.8$

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$

$0.6 \cdot 2 = 1.2 \rightarrow 1$

$0.2 \cdot 2 = 0.4 \rightarrow 0$

$0.4 \cdot 2 = 0.8 \rightarrow 0$

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$

$0.6 \cdot 2 = 1.2 \rightarrow 1$

$0.2 \cdot 2 = 0.4 \rightarrow 0$

$0.4 \cdot 2 = 0.8 \rightarrow 0$

$0.8 \cdot 2 = 1.6$

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \to 0$

$0.6 \cdot 2 = 1.2 \to 1$

$0.2 \cdot 2 = 0.4 \to 0$

$0.4 \cdot 2 = 0.8 \to 0$

$0.8 \cdot 2 = 1.6 \to 1$

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$

$0.6 \cdot 2 = 1.2 \rightarrow 1$

$0.2 \cdot 2 = 0.4 \rightarrow 0$

$0.4 \cdot 2 = 0.8 \rightarrow 0$

$0.8 \cdot 2 = 1.6 \rightarrow 1$

$0.6 \cdot 2 = 1.2$

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$

$0.6 \cdot 2 = 1.2 \rightarrow 1$

$0.2 \cdot 2 = 0.4 \rightarrow 0$

$0.4 \cdot 2 = 0.8 \rightarrow 0$

$0.8 \cdot 2 = 1.6 \rightarrow 1$

$0.6 \cdot 2 = 1.2 \rightarrow 1$

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$

$0.6 \cdot 2 = 1.2 \rightarrow 1$

$0.2 \cdot 2 = 0.4 \rightarrow 0$

$0.4 \cdot 2 = 0.8 \rightarrow 0$

$0.8 \cdot 2 = 1.6 \rightarrow 1$

$0.6 \cdot 2 = 1.2 \rightarrow 1$

So the binary form of 0.3 is 0.010011, we can even see that its infinite binary form is: $0.0\dot{1}00\dot{1}$.

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \dots, 1/2^n, \dots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \to 0$

$0.6 \cdot 2 = 1.2 \to 1$

$0.2 \cdot 2 = 0.4 \to 0$

$0.4 \cdot 2 = 0.8 \to 0$

$0.8 \cdot 2 = 1.6 \to 1$

$0.6 \cdot 2 = 1.2 \to 1$

0.3	2
0.6	0

So the binary form of 0.3 is 0.010011, we can even see that its infinite binary form is: $0.0\dot{1}00\dot{1}$.

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$
$0.6 \cdot 2 = 1.2 \rightarrow 1$
$0.2 \cdot 2 = 0.4 \rightarrow 0$
$0.4 \cdot 2 = 0.8 \rightarrow 0$
$0.8 \cdot 2 = 1.6 \rightarrow 1$
$0.6 \cdot 2 = 1.2 \rightarrow 1$

0.3	2
0.6	0
1.2	1

So the binary form of 0.3 is 0.010011, we can even see that its infinite binary form is: $0.0\dot{1}00\dot{1}$.

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$
$0.6 \cdot 2 = 1.2 \rightarrow 1$
$0.2 \cdot 2 = 0.4 \rightarrow 0$
$0.4 \cdot 2 = 0.8 \rightarrow 0$
$0.8 \cdot 2 = 1.6 \rightarrow 1$
$0.6 \cdot 2 = 1.2 \rightarrow 1$

0.3	2
0.6	0
1.2	1
0.4	0

So the binary form of 0.3 is 0.010011, we can even see that its infinite binary form is: $0.0\dot{1}00\dot{1}$.

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \to 0$

$0.6 \cdot 2 = 1.2 \to 1$

$0.2 \cdot 2 = 0.4 \to 0$

$0.4 \cdot 2 = 0.8 \to 0$

$0.8 \cdot 2 = 1.6 \to 1$

$0.6 \cdot 2 = 1.2 \to 1$

0.3	2
0.6	0
1.2	1
0.4	0
0.8	0

So the binary form of 0.3 is 0.010011, we can even see that its infinite binary form is: $0.0\dot{1}00\dot{1}$.

Binary numbers

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$
$0.6 \cdot 2 = 1.2 \rightarrow 1$
$0.2 \cdot 2 = 0.4 \rightarrow 0$
$0.4 \cdot 2 = 0.8 \rightarrow 0$
$0.8 \cdot 2 = 1.6 \rightarrow 1$
$0.6 \cdot 2 = 1.2 \rightarrow 1$

0.3	2
0.6	0
1.2	1
0.4	0
0.8	0
1.6	1

So the binary form of 0.3 is 0.010011, we can even see that its infinite binary form is: $0.0\dot{1}00\dot{1}$.

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, 1/2, 1/4,. . . , $1/2^n$,. . . . For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \rightarrow 0$
$0.6 \cdot 2 = 1.2 \rightarrow 1$
$0.2 \cdot 2 = 0.4 \rightarrow 0$
$0.4 \cdot 2 = 0.8 \rightarrow 0$
$0.8 \cdot 2 = 1.6 \rightarrow 1$
$0.6 \cdot 2 = 1.2 \rightarrow 1$

0.3	2
0.6	0
1.2	1
0.4	0
0.8	0
1.6	1
1.2	1

So the binary form of 0.3 is 0.010011, we can even see that its infinite binary form is: $0.0\dot{1}00\dot{1}$.

Example

How to convert a fractional number into binary? For example let us write the first 6 digits of 0.3 in binary!

Solution: The meaning of digits after the decimal point, $1/2$, $1/4, \ldots, 1/2^n, \ldots$. For example multiplying the binary number 0.1011001 by 2 the integer part of the result in order is 1, 0, 1, 1, 0, 0, 1. Using this method:

$0.3 \cdot 2 = 0.6 \to 0$
$0.6 \cdot 2 = 1.2 \to 1$
$0.2 \cdot 2 = 0.4 \to 0$
$0.4 \cdot 2 = 0.8 \to 0$
$0.8 \cdot 2 = 1.6 \to 1$
$0.6 \cdot 2 = 1.2 \to 1$

0.3	2
0.6	0
1.2	1
0.4	0
0.8	0
1.6	1
1.2	1

So the binary form of 0.3 is 0.010011, we can even see that its infinite binary form is: $0.0\dot{1}00\dot{1}$.

Hexadecimal numbers

Hexadecimal (base 16) numbers:

bin	hex	bin	hex
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Hexadecimal (base 16) numbers:

bin	hex	bin	hex
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

For example $0011\,1100\;1111\,1010 = 0x3CFA$.

1's complement representation

1's complement on n-bits: the first bit is the sign.

1's complement representation

1's complement on n-bits: the first bit is the sign. The range of representable numbers: $-2^{n-1}+1$ to $2^{n-1}-1$.

1's complement representation

1's complement on n-bits: the first bit is the sign. The range of representable numbers: $-2^{n-1}+1$ to $2^{n-1}-1$.

For example on 4 bits: -7 to 7.

$1001 \rightarrow -1$

$1100 \rightarrow -4$

$1111 \rightarrow -7$

$1000 \rightarrow -0$

$0000 \rightarrow +0$

1's complement representation

1's complement on n-bits: the first bit is the sign. The range of representable numbers: $-2^{n-1}+1$ to $2^{n-1}-1$.

For example on 4 bits: -7 to 7.

$1001 \rightarrow -1$

$1100 \rightarrow -4$

$1111 \rightarrow -7$

$1000 \rightarrow -0$

$0000 \rightarrow +0$

Disadvantage: There's $+0$ and -0.

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$.

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}\ + 1$.

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}$ $+1$.
the form of -1 is

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}$ $+1$.
the form of -1 is $11\ldots11_2$,

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}$ $+1$.
the form of -1 is $11\ldots11_2$, of -2 is

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}\ + 1$.
the form of -1 is $11\ldots11_2$, of -2 is $11\ldots10_2$,

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}$ $+1$.
the form of -1 is $11\ldots11_2$, of -2 is $11\ldots10_2$, of -3 is

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}\ + 1$.
the form of -1 is $11\ldots11_2$, of -2 is $11\ldots10_2$, of -3 is $11\ldots01_2$.

2's complement representation

2's complement representation on *n*-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\dots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}$ $+1$.
the form of -1 is $11\dots11_2$, of -2 is $11\dots10_2$, of -3 is $11\dots01_2$.

Example

let $n = 4$, $x = -5$: $-5 \rightarrow$

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}\ + 1$.
the form of -1 is $11\ldots11_2$, of -2 is $11\ldots10_2$, of -3 is $11\ldots01_2$.

Example

let $n = 4$, $x = -5$: $-5 \to \bar{x} = 16 - 5$

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}$ $+1$.
the form of -1 is $11\ldots11_2$, of -2 is $11\ldots10_2$, of -3 is $11\ldots01_2$.

Example

let $n = 4$, $x = -5$: $-5 \rightarrow \bar{x} = 16 - 5 = 11$

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}$ $+1$.
the form of -1 is $11\ldots11_2$, of -2 is $11\ldots10_2$, of -3 is $11\ldots01_2$.

Example

let $n = 4$, $x = -5$: $-5 \rightarrow \bar{x} = 16 - 5 = 11 = 1011_2$

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\dots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x|$ = complement of $\bar{x}$ $+1$.
the form of -1 is $11\dots11_2$, of -2 is $11\dots10_2$, of -3 is $11\dots01_2$.

Example

let $n = 4$, $x = -5$: $-5 \to \bar{x} = 16 - 5 = 11 = 1011_2$
with bit operations:
$x = -5 \to |x| = 5$

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\dots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}$ $+1$.
the form of -1 is $11\dots11_2$, of -2 is $11\dots10_2$, of -3 is $11\dots01_2$.

Example

let $n = 4$, $x = -5$: $-5 \rightarrow \bar{x} = 16 - 5 = 11 = 1011_2$
with bit operations:
$x = -5 \rightarrow |x| = 5 \rightarrow 0101_2$

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}\ + 1$.
the form of -1 is $11\ldots11_2$, of -2 is $11\ldots10_2$, of -3 is $11\ldots01_2$.

Example

let $n = 4$, $x = -5$: $-5 \to \bar{x} = 16 - 5 = 11 = 1011_2$
with bit operations:
$x = -5 \to |x| = 5 \to 0101_2 \to \bar{x} = 1010_2 + 1_2 = 1011_2$

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}$ $+1$.
the form of -1 is $11\ldots11_2$, of -2 is $11\ldots10_2$, of -3 is $11\ldots01_2$.

Example

let $n = 4$, $x = -5$: $-5 \to \bar{x} = 16 - 5 = 11 = 1011_2$
with bit operations:
$x = -5 \to |x| = 5 \to 0101_2 \to \bar{x} = 1010_2 + 1_2 = 1011_2$
the reverse: $\bar{x} = 1011_2$

2's complement representation

2's complement representation on n-bits: we want a signed representation of numbers where there aren't $+0$ and -0.

$$\bar{x} = \begin{cases} x & \text{if } x \text{ is non-negative,} \\ 2^n - |x| & \text{if } x \text{ is negative.} \end{cases}$$

To calculate $2^n - |x|$ you can take the complement of $|x|$ and add 1: $2^n - |x| = (2^n - 1) - |x| + 1 = 11\ldots1_2 - |x| + 1$. Since $|x| = 2^n - (2^n - |x|)$, calculating x from $\bar{x}$ can be done the same way, so if the first bit is 1, then $|x| =$ complement of $\bar{x}\ + 1$.
the form of -1 is $11\ldots11_2$, of -2 is $11\ldots10_2$, of -3 is $11\ldots01_2$.

Example

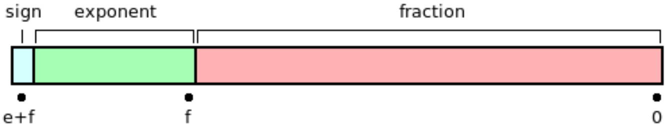
let $n = 4$, $x = -5$: $-5 \rightarrow \bar{x} = 16 - 5 = 11 = 1011_2$
with bit operations:
$x = -5 \rightarrow |x| = 5 \rightarrow 0101_2 \rightarrow \bar{x} = 1010_2 + 1_2 = 1011_2$
the reverse: $\bar{x} = 1011_2 \rightarrow x = 0100_2 + 1_2 = 0101_2 = 5$.

Sign, exponent, fraction

IEEE 754-2008, ISO/IEC/IEEE 60559:2011

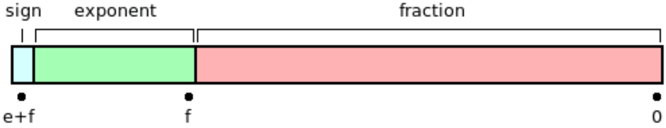
Sign, exponent, fraction

IEEE 754-2008, ISO/IEC/IEEE 60559:2011



Sign, exponent, fraction

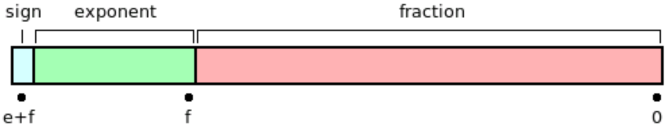
IEEE 754-2008, ISO/IEC/IEEE 60559:2011



	s=sign	e=exponent	fraction	all	bias
simple	1	8	23	32	127 (01111111)
double	1	11	52	64	1023 (01111111111)

Sign, exponent, fraction

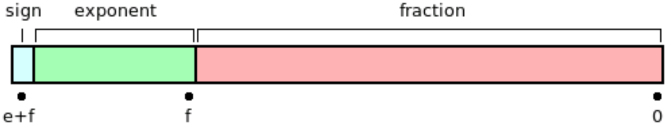
IEEE 754-2008, ISO/IEC/IEEE 60559:2011



	s=sign	e=exponent	fraction	all	bias
simple	1	8	23	32	127 (01111111)
double	1	11	52	64	1023 (01111111111)

simple: $(-1)^s(1.b_{22}b_{21}\dots b_0)_2 \cdot 2^{e-127} = \left(1 + \sum_{i=1}^{23} b_{23-i}2^{-i}\right) \cdot 2^{e-127}$

IEEE 754-2008, ISO/IEC/IEEE 60559:2011



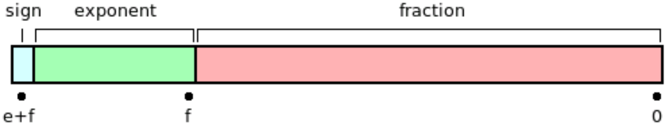
	s=sign	e=exponent	fraction	all	bias
simple	1	8	23	32	127 (01111111)
double	1	11	52	64	1023 (01111111111)

simple: $(-1)^s(1.b_{22}b_{21}\dots b_0)_2 \cdot 2^{e-127} = \left(1 + \sum_{i=1}^{23} b_{23-i}2^{-i}\right) \cdot 2^{e-127}$

double: $(-1)^s(1.b_{51}b_{50}\dots b_0)_2 \cdot 2^{e-1023} = \left(1 + \sum_{i=1}^{52} b_{52-i}2^{-i}\right) \cdot 2^{e-1023}$

Sign, exponent, fraction

IEEE 754-2008, ISO/IEC/IEEE 60559:2011



	s=sign	e=exponent	fraction	all	bias
simple	1	8	23	32	127 (01111111)
double	1	11	52	64	1023 (01111111111)

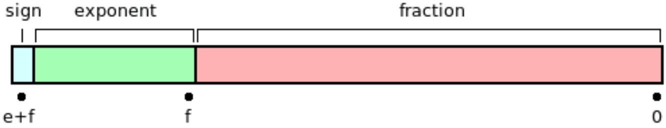
simple: $(-1)^s(1.b_{22}b_{21}\dots b_0)_2 \cdot 2^{e-127} = \left(1+\sum_{i=1}^{23} b_{23-i}2^{-i}\right) \cdot 2^{e-127}$

double: $(-1)^s(1.b_{51}b_{50}\dots b_0)_2 \cdot 2^{e-1023} = \left(1+\sum_{i=1}^{52} b_{52-i}2^{-i}\right) \cdot 2^{e-1023}$

For example using double precision, between $2^{52} = 4\,503\,599\,627\,370\,496$ and $2^{53} = 9\,007\,199\,254\,740\,992$ only integers are represented.

Sign, exponent, fraction

IEEE 754-2008, ISO/IEC/IEEE 60559:2011



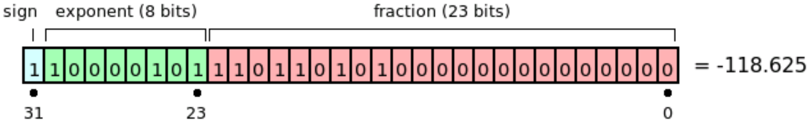
	s=sign	e=exponent	fraction	all	bias
simple	1	8	23	32	127 (01111111)
double	1	11	52	64	1023 (01111111111)

simple: $(-1)^s(1.b_{22}b_{21}\dots b_0)_2 \cdot 2^{e-127} = \left(1+\sum_{i=1}^{23} b_{23-i}2^{-i}\right)\cdot 2^{e-127}$

double: $(-1)^s(1.b_{51}b_{50}\dots b_0)_2 \cdot 2^{e-1023} = \left(1+\sum_{i=1}^{52} b_{52-i}2^{-i}\right)\cdot 2^{e-1023}$

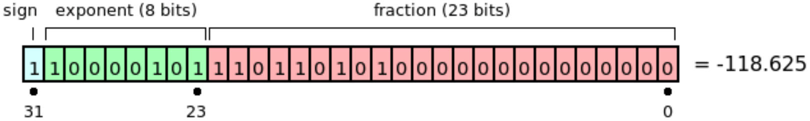
For example using double precision, between $2^{52} = 4\,503\,599\,627\,370\,496$ and $2^{53} = 9\,007\,199\,254\,740\,992$ only integers are represented. between 2^{53} and 2^{54} only even integers...

Sign, exponent, fraction



sign 1 $\rightarrow$ negative

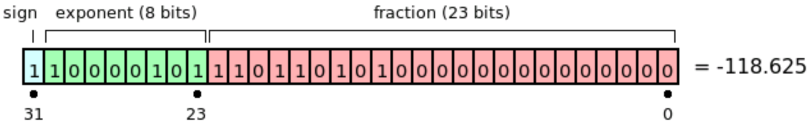
Sign, exponent, fraction



sign 1 $\rightarrow$ negative

exponent $10000101_2 - 01111111_2 = 00000110_2$, so 6

Sign, exponent, fraction

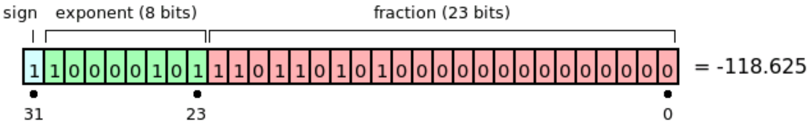


sign 1 $\rightarrow$ negative

exponent $10000101_2 - 01111111_2 = 00000110_2$, so 6

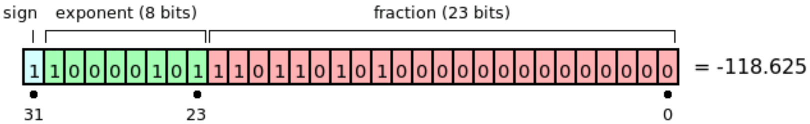
fraction (1.significand) 1.110110101_2,

Sign, exponent, fraction



sign 1 $\rightarrow$ negative
exponent $10000101_2 - 01111111_2 = 00000110_2$, so 6
fraction (1.significand) 1.110110101_2,
the number -1110110.101_2,

Sign, exponent, fraction



sign 1 $\rightarrow$ negative

exponent $10000101_2 - 01111111_2 = 00000110_2$, so 6

fraction (1.significand) 1.110110101_2,

the number -1110110.101_2, which is -118.625

These are nearly history

1. ISO-8859-1 Latin1 (West European)

These are nearly history

1. ISO-8859-1 Latin1 (West European)
2. ISO-8859-2 Latin2 (East European)

These are nearly history

1. ISO-8859-1 Latin1 (West European)
2. ISO-8859-2 Latin2 (East European)
3. ISO-8859-3 Latin3 (South European)

These are nearly history

1. ISO-8859-1 Latin1 (West European)
2. ISO-8859-2 Latin2 (East European)
3. ISO-8859-3 Latin3 (South European)
4. ISO-8859-4 Latin4 (North European)

These are nearly history

1. ISO-8859-1 Latin1 (West European)
2. ISO-8859-2 Latin2 (East European)
3. ISO-8859-3 Latin3 (South European)
4. ISO-8859-4 Latin4 (North European)
5. ISO-8859-5 Cyrillic

These are nearly history

1. ISO-8859-1 Latin1 (West European)
2. ISO-8859-2 Latin2 (East European)
3. ISO-8859-3 Latin3 (South European)
4. ISO-8859-4 Latin4 (North European)
5. ISO-8859-5 Cyrillic
6. ISO-8859-6 Arabic

These are nearly history

1. ISO-8859-1 Latin1 (West European)
2. ISO-8859-2 Latin2 (East European)
3. ISO-8859-3 Latin3 (South European)
4. ISO-8859-4 Latin4 (North European)
5. ISO-8859-5 Cyrillic
6. ISO-8859-6 Arabic
7. ISO-8859-7 Greek

These are nearly history

1. ISO-8859-1 Latin1 (West European)
2. ISO-8859-2 Latin2 (East European)
3. ISO-8859-3 Latin3 (South European)
4. ISO-8859-4 Latin4 (North European)
5. ISO-8859-5 Cyrillic
6. ISO-8859-6 Arabic
7. ISO-8859-7 Greek
8. ISO-8859-8 Hebrew

These are nearly history

1. ISO-8859-1 Latin1 (West European)
2. ISO-8859-2 Latin2 (East European)
3. ISO-8859-3 Latin3 (South European)
4. ISO-8859-4 Latin4 (North European)
5. ISO-8859-5 Cyrillic
6. ISO-8859-6 Arabic
7. ISO-8859-7 Greek
8. ISO-8859-8 Hebrew
9. ISO-8859-9 Latin5 (Turkish)

These are nearly history

1. ISO-8859-1 Latin1 (West European)
2. ISO-8859-2 Latin2 (East European)
3. ISO-8859-3 Latin3 (South European)
4. ISO-8859-4 Latin4 (North European)
5. ISO-8859-5 Cyrillic
6. ISO-8859-6 Arabic
7. ISO-8859-7 Greek
8. ISO-8859-8 Hebrew
9. ISO-8859-9 Latin5 (Turkish)
10. ISO-8859-10 Latin6 (Nordic)

These are nearly history

ISO-8859-2, Microsoft CP1250 (Windows Latin2), CP852 (DOSLatin2)

ISO-8859-1	C1	Á	U+00C1	LATIN CAPITAL LETTER A WITH ACUTE
ISO-8859-1	E1	á	U+00E1	LATIN SMALL LETTER A WITH ACUTE

These are nearly history

ISO-8859-2, Microsoft CP1250 (Windows Latin2), CP852 (DOSLatin2)

ISO-8859-1	C1	Á	U+00C1	LATIN CAPITAL LETTER A WITH ACUTE
ISO-8859-1	E1	á	U+00E1	LATIN SMALL LETTER A WITH ACUTE
ISO-8859-1	D5	Õ	U+00D5	LATIN CAPITAL LETTER O WITH TILDE
ISO-8859-1	DB	Û	U+00DB	LATIN CAPITAL LETTER U WITH CIRCUMFLEX
ISO-8859-1	F5	õ	U+00F5	LATIN SMALL LETTER O WITH TILDE
ISO-8859-1	FB	û	U+00FB	LATIN SMALL LETTER U WITH CIRCUMFLEX

These are nearly history

ISO-8859-2, Microsoft CP1250 (Windows Latin2), CP852 (DOSLatin2)

ISO-8859-1	C1	Á	U+00C1	LATIN CAPITAL LETTER A WITH ACUTE
ISO-8859-1	E1	á	U+00E1	LATIN SMALL LETTER A WITH ACUTE
ISO-8859-1	D5	Õ	U+00D5	LATIN CAPITAL LETTER O WITH TILDE
ISO-8859-1	DB	Û	U+00DB	LATIN CAPITAL LETTER U WITH CIRCUMFLEX
ISO-8859-1	F5	õ	U+00F5	LATIN SMALL LETTER O WITH TILDE
ISO-8859-1	FB	û	U+00FB	LATIN SMALL LETTER U WITH CIRCUMFLEX
ISO-8859-2	D5	Ő	U+0150	LATIN CAPITAL LETTER O WITH DOUBLE ACU
ISO-8859-2	DB	Ű	U+0170	LATIN CAPITAL LETTER U WITH DOUBLE ACU
ISO-8859-2	F5	ő	U+0151	LATIN SMALL LETTER O WITH DOUBLE ACUT
ISO-8859-2	FB	ű	U+0171	LATIN SMALL LETTER U WITH DOUBLE ACUT

These are nearly history

ISO-8859-2, Microsoft CP1250 (Windows Latin2), CP852 (DOSLatin2)

ISO-8859-1	C1	Á	U+00C1	LATIN CAPITAL LETTER A WITH ACUTE
ISO-8859-1	E1	á	U+00E1	LATIN SMALL LETTER A WITH ACUTE
ISO-8859-1	D5	Õ	U+00D5	LATIN CAPITAL LETTER O WITH TILDE
ISO-8859-1	DB	Û	U+00DB	LATIN CAPITAL LETTER U WITH CIRCUMFLEX
ISO-8859-1	F5	õ	U+00F5	LATIN SMALL LETTER O WITH TILDE
ISO-8859-1	FB	û	U+00FB	LATIN SMALL LETTER U WITH CIRCUMFLEX
ISO-8859-2	D5	Ő	U+0150	LATIN CAPITAL LETTER O WITH DOUBLE ACU
ISO-8859-2	DB	Ű	U+0170	LATIN CAPITAL LETTER U WITH DOUBLE ACU
ISO-8859-2	F5	ő	U+0151	LATIN SMALL LETTER O WITH DOUBLE ACUT
ISO-8859-2	FB	ű	U+0171	LATIN SMALL LETTER U WITH DOUBLE ACUTE
CP1250	82	‚	U+201A	SINGLE LOW-9 QUOTATION MARK
CP1250	84	„	U+201E	DOUBLE LOW-9 QUOTATION MARK
CP1250	85	...	U+2026	HORIZONTAL ELLIPSIS
CP1250	91	‘	U+2018	LEFT SINGLE QUOTATION MARK
CP1250	92	’	U+2019	RIGHT SINGLE QUOTATION MARK
CP1250	93	“	U+201C	LEFT DOUBLE QUOTATION MARK
CP1250	94	”	U+201D	RIGHT DOUBLE QUOTATION MARK
CP1250	96	–	U+2013	EN DASH
CP1250	97	—	U+2014	EM DASH

Latin encoding

- U+0000 - U+007F ASCII

Latin encoding

- U+0000 - U+007F ASCII
- U+0080 - U+00FF Latin-1

Latin encoding

- U+0000 - U+007F ASCII
- U+0080 - U+00FF Latin-1
- U+0100 - U+017F Latin Extended-A (latin1, hungarian ő, ű)

Latin encoding

- U+0000 - U+007F ASCII
- U+0080 - U+00FF Latin-1
- U+0100 - U+017F Latin Extended-A (latin1, hungarian ő, ű)
- U+0180 - U+024F Latin Extended-B

Latin encoding

- U+0000 - U+007F ASCII
- U+0080 - U+00FF Latin-1
- U+0100 - U+017F Latin Extended-A (latin1, hungarian ő, ű)
- U+0180 - U+024F Latin Extended-B
- U+1E00 - U+1EFF Latin Extended Additional

UTF – Unicode Transformation Format

- UTF-8 every character is represented on 8, 16, 24 or 32-bits.

UTF – Unicode Transformation Format

- UTF-8 every character is represented on 8, 16, 24 or 32-bits.
- UTF-16 every character is represented on 16 or 32-bits.

UTF – Unicode Transformation Format

- UTF-8 every character is represented on 8, 16, 24 or 32-bits.
- UTF-16 every character is represented on 16 or 32-bits.
- UTF-32 every character is represented on 32-bits.

UTF-8

Unicode		UTF-8	a official name of the character
U+0020		20	SPACE
U+0030	0	30	DIGIT ZERO
U+0040	@	40	COMMERCIAL AT
U+0041	A	41	LATIN CAPITAL LETTER A
U+0061	a	61	LATIN SMALL LETTER A

UTF-8

Unicode		UTF-8	a official name of the character
U+0020		20	SPACE
U+0030	0	30	DIGIT ZERO
U+0040	@	40	COMMERCIAL AT
U+0041	A	41	LATIN CAPITAL LETTER A
U+0061	a	61	LATIN SMALL LETTER A
U+00C1	Á	c3 81	LATIN CAPITAL LETTER A WITH ACUTE
U+00C9	É	c3 89	LATIN CAPITAL LETTER E WITH ACUTE
U+00CD	Í	c3 8d	LATIN CAPITAL LETTER I WITH ACUTE
U+00D3	Ó	c3 93	LATIN CAPITAL LETTER O WITH ACUTE
U+00D6	Ö	c3 96	LATIN CAPITAL LETTER O WITH DIAERESIS
U+00DA	Ú	c3 9a	LATIN CAPITAL LETTER U WITH ACUTE
U+00DC	Ü	c3 9c	LATIN CAPITAL LETTER U WITH DIAERESIS
U+00E1	á	c3 a1	LATIN SMALL LETTER A WITH ACUTE
U+00E9	é	c3 a9	LATIN SMALL LETTER E WITH ACUTE
U+00ED	í	c3 ad	LATIN SMALL LETTER I WITH ACUTE
U+00F3	ó	c3 b3	LATIN SMALL LETTER O WITH ACUTE
U+00F6	ö	c3 b6	LATIN SMALL LETTER O WITH DIAERESIS
U+00FA	ú	c3 ba	LATIN SMALL LETTER U WITH ACUTE
U+00FC	ü	c3 bc	LATIN SMALL LETTER U WITH DIAERESIS

UTF-8

Unicode		UTF-8	a official name of the character
U+0020		20	SPACE
U+0030	0	30	DIGIT ZERO
U+0040	@	40	COMMERCIAL AT
U+0041	A	41	LATIN CAPITAL LETTER A
U+0061	a	61	LATIN SMALL LETTER A
U+00C1	Á	c3 81	LATIN CAPITAL LETTER A WITH ACUTE
U+00C9	É	c3 89	LATIN CAPITAL LETTER E WITH ACUTE
U+00CD	Í	c3 8d	LATIN CAPITAL LETTER I WITH ACUTE
U+00D3	Ó	c3 93	LATIN CAPITAL LETTER O WITH ACUTE
U+00D6	Ö	c3 96	LATIN CAPITAL LETTER O WITH DIAERESIS
U+00DA	Ú	c3 9a	LATIN CAPITAL LETTER U WITH ACUTE
U+00DC	Ü	c3 9c	LATIN CAPITAL LETTER U WITH DIAERESIS
U+00E1	á	c3 a1	LATIN SMALL LETTER A WITH ACUTE
U+00E9	é	c3 a9	LATIN SMALL LETTER E WITH ACUTE
U+00ED	í	c3 ad	LATIN SMALL LETTER I WITH ACUTE
U+00F3	ó	c3 b3	LATIN SMALL LETTER O WITH ACUTE
U+00F6	ö	c3 b6	LATIN SMALL LETTER O WITH DIAERESIS
U+00FA	ú	c3 ba	LATIN SMALL LETTER U WITH ACUTE
U+00FC	ü	c3 bc	LATIN SMALL LETTER U WITH DIAERESIS
U+0150	Ő	c5 90	LATIN CAPITAL LETTER O WITH DOUBLE ACUTE
U+0151	ő	c5 91	LATIN SMALL LETTER O WITH DOUBLE ACUTE

UTF-8

Unicode		UTF-8	a official name of the character
U+0020		20	SPACE
U+0030	0	30	DIGIT ZERO
U+0040	@	40	COMMERCIAL AT
U+0041	A	41	LATIN CAPITAL LETTER A
U+0061	a	61	LATIN SMALL LETTER A
U+00C1	Á	c3 81	LATIN CAPITAL LETTER A WITH ACUTE
U+00C9	É	c3 89	LATIN CAPITAL LETTER E WITH ACUTE
U+00CD	Í	c3 8d	LATIN CAPITAL LETTER I WITH ACUTE
U+00D3	Ó	c3 93	LATIN CAPITAL LETTER O WITH ACUTE
U+00D6	Ö	c3 96	LATIN CAPITAL LETTER O WITH DIAERESIS
U+00DA	Ú	c3 9a	LATIN CAPITAL LETTER U WITH ACUTE
U+00DC	Ü	c3 9c	LATIN CAPITAL LETTER U WITH DIAERESIS
U+00E1	á	c3 a1	LATIN SMALL LETTER A WITH ACUTE
U+00E9	é	c3 a9	LATIN SMALL LETTER E WITH ACUTE
U+00ED	í	c3 ad	LATIN SMALL LETTER I WITH ACUTE
U+00F3	ó	c3 b3	LATIN SMALL LETTER O WITH ACUTE
U+00F6	ö	c3 b6	LATIN SMALL LETTER O WITH DIAERESIS
U+00FA	ú	c3 ba	LATIN SMALL LETTER U WITH ACUTE
U+00FC	ü	c3 bc	LATIN SMALL LETTER U WITH DIAERESIS
U+0150	Ő	c5 90	LATIN CAPITAL LETTER O WITH DOUBLE ACUTE
U+0151	ő	c5 91	LATIN SMALL LETTER O WITH DOUBLE ACUTE
U+0170	Ű	c5 b0	LATIN CAPITAL LETTER U WITH DOUBLE ACUTE
U+0171	ű	c5 b1	LATIN SMALL LETTER U WITH DOUBLE ACUTE

UTF-8

Range (number)	binary form	UTF-8

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1→1100 0001

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1→1100 0001→00011 000001

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1→1100 0001→00011 000001→11000011 10000001

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1→1100 0001→00011 000001→11000011 10000001→C3 81

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1→1100 0001→00011 000001→11000011 10000001→C3 81

Õ 00D5→1101 0101→00011 010101→11000011 10010101→C3 95

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1→1100 0001→00011 000001→11000011 10000001→C3 81

Õ 00D5→1101 0101→00011 010101→11000011 10010101→C3 95

Ő 0150→0001 0101 0000→00101 010000→11000101 10010000→C5 90

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1→1100 0001→00011 000001→11000011 10000001→C3 81
Õ 00D5→1101 0101→00011 010101→11000011 10010101→C3 95
Ő 0150→0001 0101 0000→00101 010000→11000101 10010000→C5 90
Byte Order Mark FEFF

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1→1100 0001→00011 000001→11000011 10000001→C3 81
Õ 00D5→1101 0101→00011 010101→11000011 10010101→C3 95
Ő 0150→0001 0101 0000→00101 010000→11000101
10010000→C5 90
Byte Order Mark FEFF→11111110 11111111→
11101111 10111011 10111111

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1→1100 0001→00011 000001→11000011 10000001→C3 81
Õ 00D5→1101 0101→00011 010101→11000011 10010101→C3 95
Ő 0150→0001 0101 0000→00101 010000→11000101 10010000→C5 90
Byte Order Mark FEFF→11111110 11111111→ 11101111 10111011 10111111→EF BB BF (ï»¿ When viewing files written in UTF-8 formats on windows and reading with a latin-1 encoder)

UTF-8

Range (number)	binary form	UTF-8
000000-00007F (128)	0zzzzzzz	0zzzzzzz
000080-0007FF (1920)	00000yyy yyzzzzzz	110yyyyy 10zzzzzz
000800-00FFFF (63488)	xxxxyyyy yyzzzzzz	1110xxxx 10yyyyyy 10zzzzzz
010000-10FFFF (1048576)	000wwwxx xxxxyyyy yyzzzzzz	11110www 10xxxxxx 10yyyyyy 10zzz

Á 00C1→1100 0001→00011 000001→11000011 10000001→C3 81
Õ 00D5→1101 0101→00011 010101→11000011 10010101→C3 95
Ő 0150→0001 0101 0000→00101 010000→11000101 10010000→C5 90
Byte Order Mark FEFF→11111110 11111111→ 11101111 10111011 10111111→EF BB BF (ï»¿ When viewing files written in UTF-8 formats on windows and reading with a latin-1 encoder)

RAM-machine (random access machine)

- The RAM-machine consists of a p program register and an r data register, both of them indexed by natural numbers, the data register contains zeros initially.

RAM-machine (random access machine)

- The RAM-machine consists of a p program register and an r data register, both of them indexed by natural numbers, the data register contains zeros initially.
- The execution of the program starts with executing the command in cell p_0 and ends with an empty command.

RAM-machine (random access machine)

- The RAM-machine consists of a p program register and an r data register, both of them indexed by natural numbers, the data register contains zeros initially.
- The execution of the program starts with executing the command in cell p_0 and ends with an empty command.
- The contents of the ith cell of the data register ($i \in \mathbb{N}_0$) is denoted by $r[i]$ or r_i, these can only contain integers.

RAM-machine (random access machine)

- The RAM-machine consists of a p program register and an r data register, both of them indexed by natural numbers, the data register contains zeros initially.
- The execution of the program starts with executing the command in cell p_0 and ends with an empty command.
- The contents of the ith cell of the data register ($i \in \mathbb{N}_0$) is denoted by $r[i]$ or r_i, these can only contain integers.
- These are the possible commands, where $z \in \mathbb{Z}$, $i, n \in \mathbb{N}_0$:

RAM-machine (random access machine)

- The RAM-machine consists of a p program register and an r data register, both of them indexed by natural numbers, the data register contains zeros initially.
- The execution of the program starts with executing the command in cell p_0 and ends with an empty command.
- The contents of the ith cell of the data register ($i \in \mathbb{N}_0$) is denoted by $r[i]$ or r_i, these can only contain integers.
- These are the possible commands, where $z \in \mathbb{Z}$, $i, n \in \mathbb{N}_0$:

$r_i \leftarrow z$

RAM-machine (random access machine)

- The RAM-machine consists of a p program register and an r data register, both of them indexed by natural numbers, the data register contains zeros initially.
- The execution of the program starts with executing the command in cell p_0 and ends with an empty command.
- The contents of the ith cell of the data register ($i \in \mathbb{N}_0$) is denoted by $r[i]$ or r_i, these can only contain integers.
- These are the possible commands, where $z \in \mathbb{Z}$, $i, n \in \mathbb{N}_0$:

  $r_i \leftarrow z$

  $r_i \leftarrow r_n$, $r_i \leftarrow r_{r_n}$ (same as $r_i \leftarrow r[r[n]]$),

RAM-machine (random access machine)

- The RAM-machine consists of a p program register and an r data register, both of them indexed by natural numbers, the data register contains zeros initially.
- The execution of the program starts with executing the command in cell p_0 and ends with an empty command.
- The contents of the ith cell of the data register ($i \in \mathbb{N}_0$) is denoted by $r[i]$ or r_i, these can only contain integers.
- These are the possible commands, where $z \in \mathbb{Z}$, $i, n \in \mathbb{N}_0$:

  $r_i \leftarrow z$

  $r_i \leftarrow r_n$, $r_i \leftarrow r_{r_n}$ (same as $r_i \leftarrow r[r[n]]$),

  $r_i \leftarrow r_i \pm r_n$, ($r_i \leftarrow r_i * r_n$, $r_i \leftarrow r_i / r_n$),

RAM-machine (random access machine)

- The RAM-machine consists of a p program register and an r data register, both of them indexed by natural numbers, the data register contains zeros initially.
- The execution of the program starts with executing the command in cell p_0 and ends with an empty command.
- The contents of the ith cell of the data register ($i \in \mathbb{N}_0$) is denoted by $r[i]$ or r_i, these can only contain integers.
- These are the possible commands, where $z \in \mathbb{Z}$, $i, n \in \mathbb{N}_0$:

  $r_i \leftarrow z$

  $r_i \leftarrow r_n$, $r_i \leftarrow r_{r_n}$ (same as $r_i \leftarrow r[r[n]]$),

  $r_i \leftarrow r_i \pm r_n$, ($r_i \leftarrow r_i * r_n$, $r_i \leftarrow r_i / r_n$),

  p_n: jump to the nth program line,

RAM-machine (random access machine)

- The RAM-machine consists of a p program register and an r data register, both of them indexed by natural numbers, the data register contains zeros initially.
- The execution of the program starts with executing the command in cell p_0 and ends with an empty command.
- The contents of the ith cell of the data register ($i \in \mathbb{N}_0$) is denoted by $r[i]$ or r_i, these can only contain integers.
- These are the possible commands, where $z \in \mathbb{Z}$, $i, n \in \mathbb{N}_0$:

  $r_i \leftarrow z$

  $r_i \leftarrow r_n$, $r_i \leftarrow r_{r_n}$ (same as $r_i \leftarrow r[r[n]]$),

  $r_i \leftarrow r_i \pm r_n$, ($r_i \leftarrow r_i * r_n$, $r_i \leftarrow r_i / r_n$),

  p_n: jump to the nth program line,

  if $r_i = 0$ p_n: jump to the nth program line if $r_i = 0$,

RAM-machine (random access machine)

- The RAM-machine consists of a p program register and an r data register, both of them indexed by natural numbers, the data register contains zeros initially.
- The execution of the program starts with executing the command in cell p_0 and ends with an empty command.
- The contents of the ith cell of the data register ($i \in \mathbb{N}_0$) is denoted by $r[i]$ or r_i, these can only contain integers.
- These are the possible commands, where $z \in \mathbb{Z}$, $i, n \in \mathbb{N}_0$:

  $r_i \leftarrow z$

  $r_i \leftarrow r_n$, $r_i \leftarrow r_{r_n}$ (same as $r_i \leftarrow r[r[n]]$),

  $r_i \leftarrow r_i \pm r_n$, ($r_i \leftarrow r_i * r_n$, $r_i \leftarrow r_i / r_n$),

  p_n: jump to the nth program line,

  if $r_i = 0$ p_n: jump to the nth program line if $r_i = 0$,

  if $r_i > 0$ p_n: jump to the nth program line if $r_i > 0$,

RAM-machine (random access machine)

For this lecture let us use this "computer like" RAM-machine:

- The program register and memory is finite,

RAM-machine (random access machine)

For this lecture let us use this "computer like" RAM-machine:

- The program register and memory is finite,
- every memory cell is 1 byte long, every program line is 2 bytes long, the first byte contains the command and the second byte contains the operand, i.e.

RAM-machine (random access machine)

For this lecture let us use this "computer like" RAM-machine:

- The program register and memory is finite,
- every memory cell is 1 byte long, every program line is 2 bytes long, the first byte contains the command and the second byte contains the operand, i.e.

  `ADD 12` means: $r_0 \leftarrow r_0 + r_{12}$

RAM-machine (random access machine)

For this lecture let us use this "computer like" RAM-machine:

- The program register and memory is finite,
- every memory cell is 1 byte long, every program line is 2 bytes long, the first byte contains the command and the second byte contains the operand, i.e.

  `ADD 12` means: $r_0 \leftarrow r_0 + r_{12}$
- every calculation is done with the 0th memory cell (and sometimes another one),

RAM-machine (random access machine)

For this lecture let us use this "computer like" RAM-machine:

- The program register and memory is finite,
- every memory cell is 1 byte long, every program line is 2 bytes long, the first byte contains the command and the second byte contains the operand, i.e.

  `ADD 12` means: $r_0 \leftarrow r_0 + r_{12}$
- every calculation is done with the 0th memory cell (and sometimes another one),
- we use mnemonics for the commands, there are three types:

RAM-machine (random access machine)

For this lecture let us use this "computer like" RAM-machine:

- The program register and memory is finite,
- every memory cell is 1 byte long, every program line is 2 bytes long, the first byte contains the command and the second byte contains the operand, i.e.

  `ADD 12` means: $r_0 \leftarrow r_0 + r_{12}$
- every calculation is done with the 0th memory cell (and sometimes another one),
- we use mnemonics for the commands, there are three types:
  - explicit: the operand n is a number (denoted by an $=$ at the end of the expression)

RAM-machine (random access machine)

For this lecture let us use this "computer like" RAM-machine:

- The program register and memory is finite,
- every memory cell is 1 byte long, every program line is 2 bytes long, the first byte contains the command and the second byte contains the operand, i.e.

  `ADD 12` means: $r_0 \leftarrow r_0 + r_{12}$
- every calculation is done with the 0th memory cell (and sometimes another one),
- we use mnemonics for the commands, there are three types:
  - explicit: the operand n is a number (denoted by an $=$ at the end of the expression)
  - direct: the operand n is a memory cell, the operation is done with the contents of $r[n]$,

RAM-machine (random access machine)

For this lecture let us use this "computer like" RAM-machine:

- The program register and memory is finite,
- every memory cell is 1 byte long, every program line is 2 bytes long, the first byte contains the command and the second byte contains the operand, i.e.

  `ADD 12` means: $r_0 \leftarrow r_0 + r_{12}$
- every calculation is done with the 0th memory cell (and sometimes another one),
- we use mnemonics for the commands, there are three types:
  - explicit: the operand n is a number (denoted by an $=$ at the end of the expression)
  - direct: the operand n is a memory cell, the operation is done with the contents of $r[n]$,
  - indirect: the operand n is the index of a memory cell, the operation is done with $r[r[n]]$ (denoted by a * at the end of the expression)

RAM-machine (random access machine)

Controller commands

JUMP	n	jump to the nth command
JZERO	n	jump to the nth command if $r_0 = 0$
JGTZ	n	jump to the nth command if $r_0 > 0$
HALT		stop

Arithmetic commands

direct			*indirect*			*explicit op*		
ADD	n	$r_0 \leftarrow r_0 + r_n$	ADD*	n	$r_0 \leftarrow r_0 + r_{r_n}$	ADD=	n	$r_0 \leftarrow r_0 + n$
SUB	n	$r_0 \leftarrow r_0 - r_n$	SUB*	n	$r_0 \leftarrow r_0 - r_{r_n}$	SUB=	n	$r_0 \leftarrow r_0 - n$
MULT	n	$r_0 \leftarrow r_0 * r_n$	MULT*	n	$r_0 \leftarrow r_0 * r_{r_n}$	MULT=	n	$r_0 \leftarrow r_0 * n$
DIV	n	$r_0 \leftarrow r_0 / r_n$	DIV*	n	$r_0 \leftarrow r_0 / r_{r_n}$	DIV=	n	$r_0 \leftarrow r_0 / n$

Data manipulation, IO

direct			*indirect*			*explicit op*		
LOAD	n	$r_0 \leftarrow r_n$	LOAD*	n	$r_0 \leftarrow r_{r_n}$	LOAD=	n	$r_0 \leftarrow n$
STORE	n	$r_n \leftarrow r_0$	STORE*	n	$r_{r_n} \leftarrow r_0$			

READ	n	reads n numbers from the input into $r_1, r_2, \ldots, r_n$
WRITE	n	writes n numbers to the output from $r_1, r_2, \ldots, r_n$

RAM-machine (random access machine)

Write a program to calculate (a, b) (greatest common divisor), where $a, b \in \mathbb{N}_0$!

p	command	operand	notes
0	LOAD =	12	
1	STORE	1	r[1] <- a
2	LOAD =	16	
3	STORE	2	r[2] <- b
4	JZERO	17	
5	LOAD	1	r[0] <- r[1]
6	DIV	2	r[0] <- ⌊a/b⌋
7	STORE	3	r[3] <- ⌊a/b⌋
8	MULT	2	
9	STORE	4	r[4] <- b*⌊a/b⌋
10	LOAD	1	
11	SUB	4	r[0] <- a - b*⌊a/b⌋ = a mod b
12	STORE	5	
13	LOAD	2	
14	STORE	1	r[1] <- b
15	LOAD	5	b <- a mod b
16	JUMP	3	
17	LOAD	1	
18	STORE	6	this is (a,b)
19	HALT	0	

RAM-machine (random access machine)

A program for the Collatz-problem: let $x \in \mathbb{N}^+$, if x is even, then $x \leftarrow x/2$, if x is odd, then $x \leftarrow 3x+1$. Is it true that starting from any number we eventually reach 1?

p	Assembly		op.	Machine code		3x + 1 (COLLATZ PROBLEM)
0	LOAD	=	33	10000011	00100001	load input value
1	STORE		2	10010000	00000010	store into cell 2
2	DIV	=	2	01110011	00000010	divide by 2
3	STORE		1	10010000	00000001	store into cell 1
4	MULT	=	2	01100011	00000010	multiply by 2
5	SUB		2	01010000	00000010	
6	JZERO		11	11100000	00001100	if it is even, jump
7	LOAD		2	10000000	00000010	
8	MULT	=	3	01100011	00000011	multiply by 3
9	ADD	=	1	01000011	00000001	plus 1
10	JUMP		1	11010000	00000010	jump to 1
11	LOAD		1	10000000	00000001	if it was even
12	STORE		2	10010000	00000010	
13	SUB	=	1	01010011	00000001	is it equal 1?
14	JZERO		17	11100000	00010010	if so, then stop
15	LOAD		1	10000000	00000001	if not, continue
16	JUMP		2	11010000	00000010	jump to 2
17	HALT			11000000	00000000	