

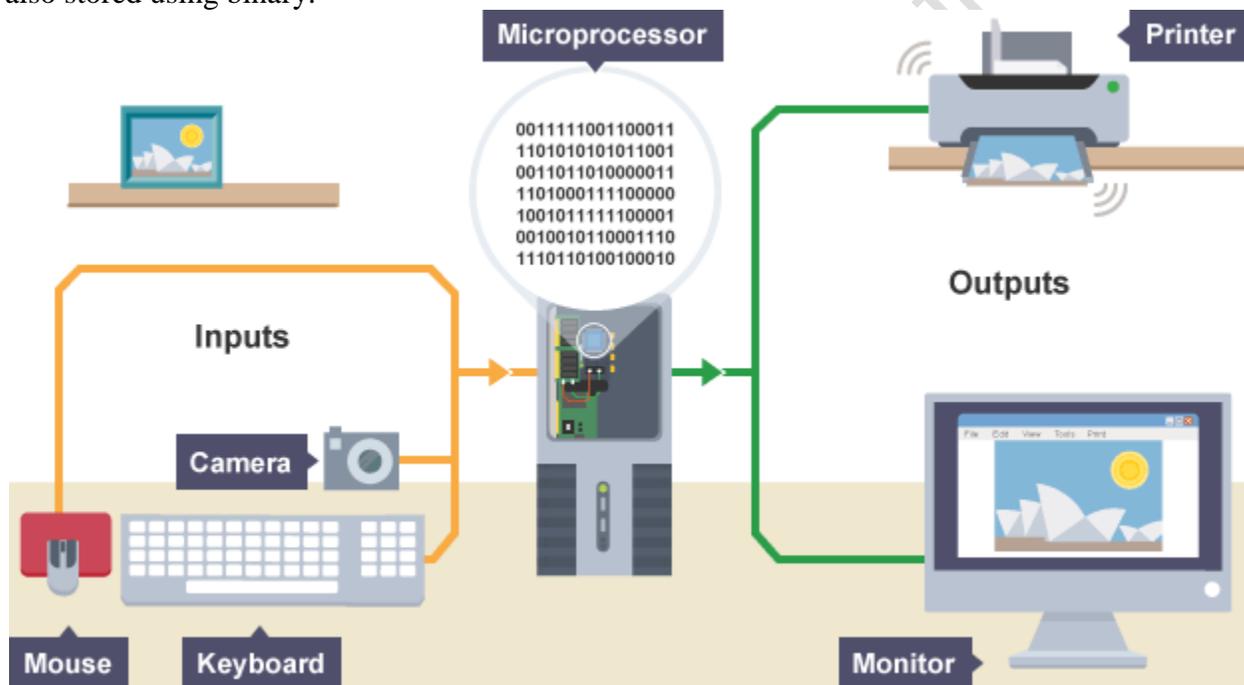
Bits and binary: Computers use **binary** - the digits 0 and 1 - to store data. A binary digit, or **bit**, is the smallest unit of data in computing. It is represented by a 0 or a 1. **Binary numbers are made up of binary digits (bits)**, e.g. the binary number **1001**.

The circuits in a computer's processor are made up of billions of **transistors**. A transistor is a tiny switch that is activated by the electronic signals it receives.

The digits 1 and 0 used in binary reflect the on and off states of a transistor.

Computer programs are sets of instructions. Each instruction is translated into **machine code** - simple binary codes that activate the **CPU**. Programmers write computer code and this is converted by a **translator** into binary instructions that the processor can **execute**.

All **software**, music, documents, and any other information that is processed by a computer, is also stored using binary.



Encoding: Everything on a computer is represented as streams of binary numbers. **Audio, images and characters all look like binary numbers in machine code.** These numbers are encoded in different data formats to give them meaning, e.g. the 8-bit pattern **01000001** could be the number **65**, the character 'A', or a color in an image.

Encoding formats have been standardized to help compatibility across different platforms. For example:

- audio is encoded as audio file formats, e.g. mp3, WAV, AAC
- video is encoded as video file formats, e.g. MPEG4, H264
- text is encoded in character sets, e.g. ASCII, Unicode

- images are encoded as file formats, e.g. BMP, JPEG, PNG

The more bits used in a pattern, the more combinations of values become available. This larger number of combinations can be used to represent many more things, eg a greater number of different symbols, or more colors in a picture.

Bits and bytes

Bits can be grouped together to make them easier to work with. **A group of 8 bits is called a byte.**

Other groupings include:

- Nibble** - 4 bits (half a byte)
- Byte** - 8 bits
- Kilobyte (KB)** - 1024 bytes (or 1024 x 8 bits) = 2^{10}
- Megabyte (MB)** - 1024 kilobytes (or 1048576 bytes) = 2^{20}
- Gigabyte (GB)** - 1024 megabytes = 2^{30}
- Terabyte (TB)** - 1024 gigabytes = 2^{40}
- Petabyte (PB)** - 1024 Terabytes = 2^{50}
- Exabyte (EB)** - 1024 Petabytes = 2^{60}
- Zettabyte (ZB)** - 1024 Exabytes = 2^{70}
- Yottabyte (YB)** 1024 Zettabytes = 2^{80}

Most computers can process millions of bits every second. A **hard drive's** storage capacity is measured in gigabytes or terabytes. **RAM** is often measured in megabytes or gigabytes.

Binary and denary

The **binary** system on computers uses combinations of 0s and 1s.

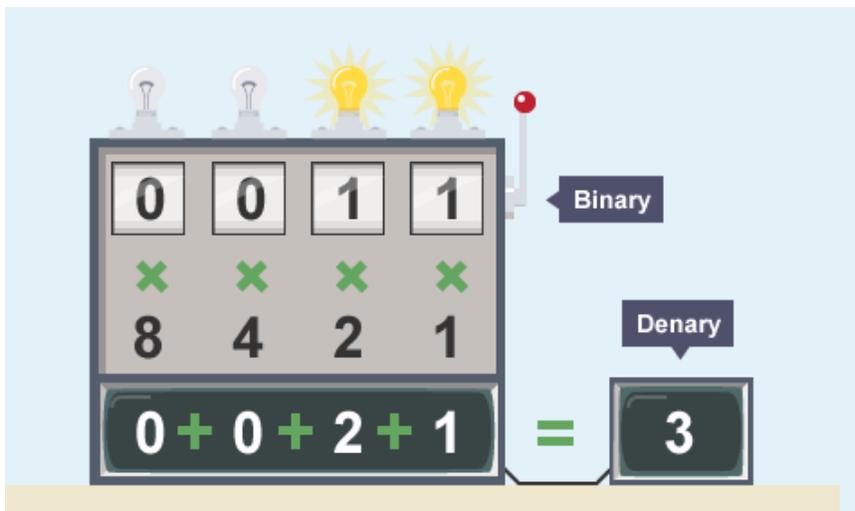
In everyday life, we use numbers based on combinations of the digits between 0 and 9.

This counting system is known as **decimal, denary** or **base 10**.

A number **base** indicates how many digits are available within a numerical system.

Denary is known as **base 10** because there are ten choices of digits between 0 and 9. For binary numbers there are only two possible digits available: 0 or 1. The binary system is also known as **base 2**.

All denary numbers have a binary equivalent and it is possible to **convert** between denary and binary.



Place values

Denary place values

Using the **denary** system, **6432** reads as six thousand, four hundred and thirty two. One way to break it down is as:

- **six** thousands
- **four** hundreds
- **three** tens
- **two** ones

Each number has a **place value** which could be put into columns. Each column is a power of ten in the base 10 system:

Thousands 1000s (10^3)	Hundreds 100s (10^2)	Tens 10s (10^1)	Ones 1s (10^0)
6	4	3	2

Or think of it as:

$$(6 \times 1000) + (4 \times 100) + (3 \times 10) + (2 \times 1) = 6432$$

Binary place values

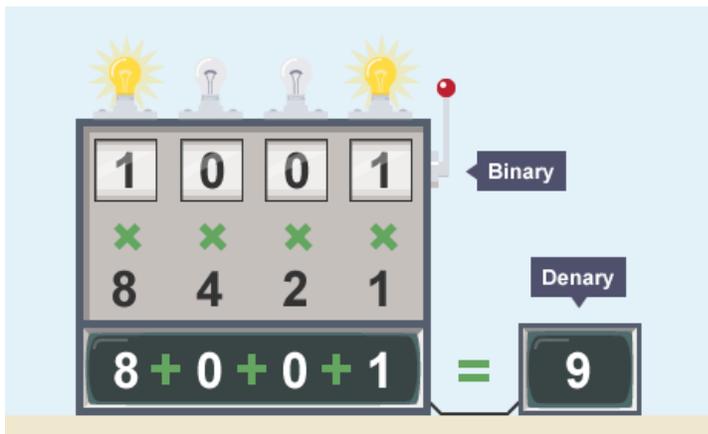
You can also break a **binary** number down into place-value columns, but each column is a power of two instead of a power of ten.

For example, take a binary number like **1001**. The columns are arranged in multiples of 2 with the binary number written below:

Eights 8s (2 ³)	Fours 4s (2 ²)	Twos 2s (2 ¹)	Ones 1s (2 ⁰)
1	0	0	1

By looking at the place values, we can calculate the equivalent denary number.

That is: $(1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 8+0+0+1$
 $(1 \times 8) + (0 \times 4) + (0 \times 2) + (1 \times 1) = 8 + 1 = 9$



Converting binary to denary

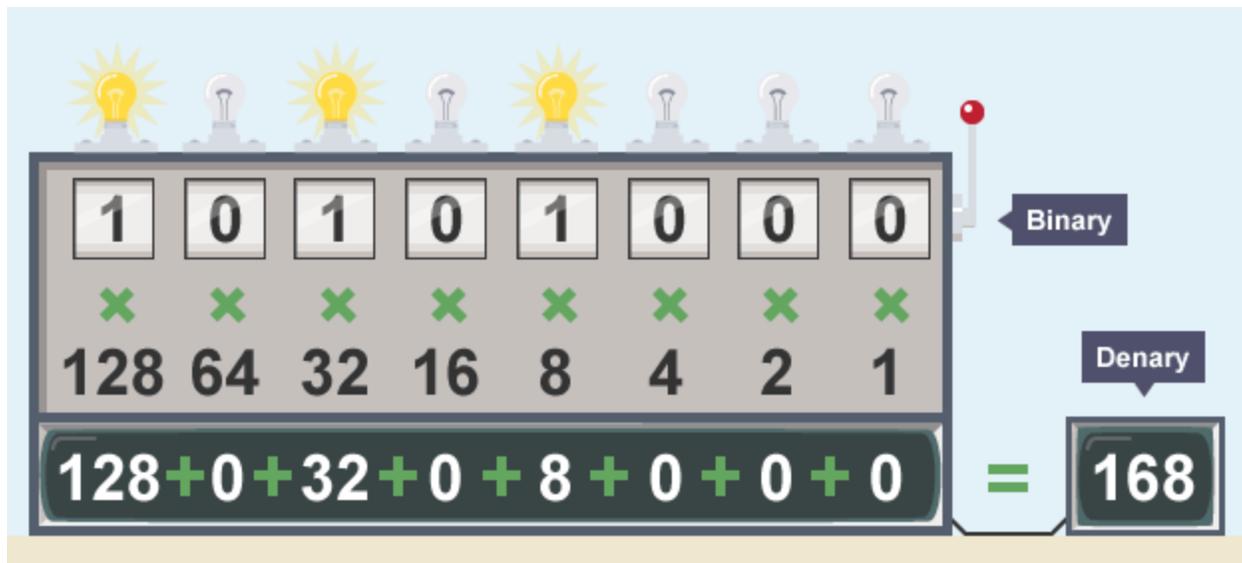
To calculate a large **binary** number like **10101000** we need more place values of multiples of 2.

- $2^7 = 128$
- $2^6 = 64$
- $2^5 = 32$
- $2^4 = 16$
- $2^3 = 8$
- $2^2 = 4$
- $2^1 = 2$
- $2^0 = 1$

In **denary** the sum is calculated as:

$$(1 \times 2^7) + (0 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 168$$

$$(1 \times 128) + (0 \times 64) + (1 \times 32) + (0 \times 16) + (1 \times 8) + (0 \times 4) + (0 \times 2) + (0 \times 1) = 128 + 32 + 8 = 168$$



The table below shows denary numbers down the left with their equivalent binary numbers marked out below the base 2 columns. Each individual column in the table represents a different **place value** equivalent to the base 2 powers.

	Binary pattern								
	Place value	Place value	Place value	Place value	Place value	Place value	Place value	Place value	Place value
	128	64	32	16	8	4	2	1	
0									0
1									1
2							1		0
3							1	1	
4						1	0	0	
5						1	0	1	
6						1	1	0	
7						1	1	1	
8					1	0	0	0	
9					1	0	0	1	
10					1	0	1	0	
....									
255	1	1	1	1	1	1	1	1	1

Converting denary to binary: Method 1

There are two methods for converting a denary (base 10) number to binary (base 2). This is method one.

Divide by two and use the remainder

Divide the starting number by 2. If it divides evenly, the binary digit is 0. If it does not - if there is a remainder - the binary digit is 1.

Play

A method of converting a denary number to binary

[Download Transcript](#)

Worked example: Denary number 83

-  $83 \div 2 = 41$ remainder **1**
-  $41 \div 2 = 20$ remainder **1**
-  $20 \div 2 = 10$ remainder **0**
-  $10 \div 2 = 5$ remainder **0**
-  $5 \div 2 = 2$ remainder **1**
-  $2 \div 2 = 1$ remainder **0**
-  $1 \div 2 = 0$ remainder **1**

Put the remainders in **reverse** order to get the final number: **1010011**.

64	32	16	8	4	2	1
1	0	1	0	0	1	1

To check that this is right, convert the binary back to denary:

$$(1 \times 64) + (0 \times 32) + (1 \times 16) + (0 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1) = 83$$

Worked example: Denary number 122

1. $122 \div 2 = 61$ remainder **0**
2. $61 \div 2 = 30$ remainder **1**
3. $30 \div 2 = 15$ remainder **0**
4. $15 \div 2 = 7$ remainder **1**
5. $7 \div 2 = 3$ remainder **1**
6. $3 \div 2 = 1$ remainder **1**
7. $1 \div 2 = 0$ remainder **1**

Put the remainders in **reverse** order to get the final number: **1111010**.

128	64	32	16	8	4	2	1
0	1	1	1	1	0	1	0



To check that this is right, convert the binary back to denary:

$$(1 \times 64) + (1 \times 32) + (1 \times 16) + (1 \times 8) + (0 \times 4) + (1 \times 2) + (0 \times 1) = 122$$

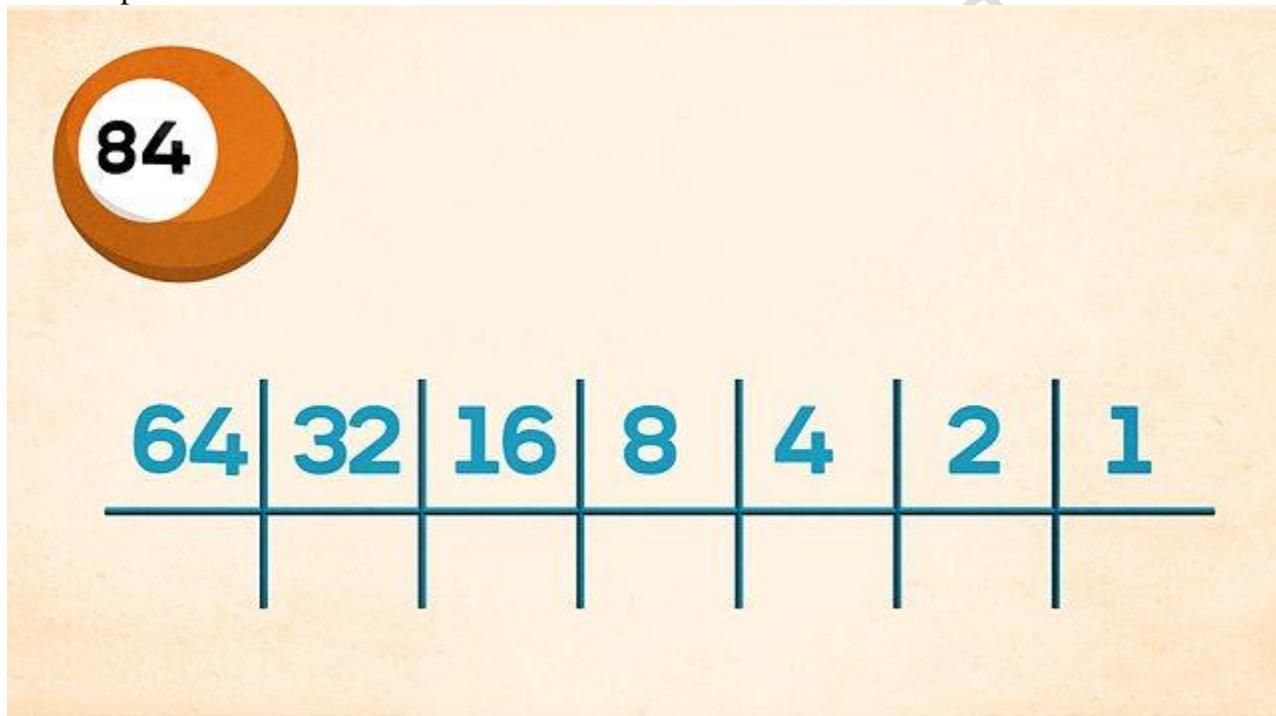
The binary representation of an even number always ends in 0 and an odd number in 1.

Converting denary to binary: Method 2

There are two methods for converting a **denary** (base 10) number to **binary** (base 2). This is method two.

Take off the biggest 2^n value you can

Remove the 2^n numbers from the main number and mark up the equivalent 2^n column with a 1. Work through the remainders until you reach zero. When you reach zero, stop and complete the final columns with 0s.



Play

A method of converting a denary number to binary

Worked example: Denary number 84

First set up the columns of base 2 numbers. Then look for the highest 2^n number that goes into 84.

1. Set up the columns of base 2 numbers
2. Find the highest 2^n number that goes into 84. The highest 2^n number is $2^6 = 64$
3. $84 - 64 = 20$. Find the highest 2^n number that goes into 20. The highest 2^n number is $2^4 = 16$
4. $20 - 16 = 4$. Find the highest 2^n number that goes into 4. The highest 2^n number is $2^2 = 4$

= 4

5. $4 - 4 = 0$

6. Mark up the columns of base 2 numbers with a 1 where the number has been the highest 2^n number, or with a 0:

64	32	16	8	4	2	1
1	0	1	0	1	0	0

Result: **84** in denary is equivalent to **1010100** in binary.

To check that this is right, convert the binary back to denary:

$$(1 \times 64) + (0 \times 32) + (1 \times 16) + (0 \times 8) + (1 \times 4) + (0 \times 2) + (0 \times 1) = 84$$

Bit number patterns

Computer systems and files have limits that are measured in **bits**. For example, image and audio files have **bit depth**.

The bit depth reflects the number of **binary** numbers available, similar to the number of combinations available on a padlock. The more wheels of numbers on a padlock, the more combinations of numbers are possible. The higher the **bit rate**, the more combinations of binary numbers are possible. Every time the bit depth increases by one, the number of binary combinations is doubled.

A 1-bit system uses combinations of numbers up to one place value (1). There are just two options: 0 or 1.

A 2-bit system uses combinations of numbers up to two place values (11). There are four options: 00, 01, 10 and 11.

Bit rate	Max (binary)	Max (denary)	Combinations available
1	1	1	2
2	11	3	4
3	111	7	8
4	1111	15	16
5	11111	31	32

A 1-bit image can have 2 colors, a 4-bit image can have 16, an 8-bit image can have 256, and a 16-bit image can have 65,536.



Binary combinations:

These tables show how many binary combinations are available for each bit size.

Binary combinations = 2^n (n = number of inputs)

One bit:

		Binary number	
		Place value 1	
Denary number	0	0	0
	1	1	1

Maximum binary number = 1

Maximum denary number = 1

Binary combinations = 2

Two bit: Maximum 4 binary combinations

		Binary pattern	
		Place value 2	Place value 1
Denary number	0	0	0
	1	0	1
	2	1	0
	3	1	1

Maximum binary number = 11

Maximum denary number = 3

Three bits = 8 binary combinations.

Maximum binary number = 111

Maximum denary number = 7

Binary combinations = 8

		Binary pattern		
		Place value 4	Place value 2	Place value 1
Denary number	0	0	0	0
	1	0	0	1
	2	0	1	0
	3	0	1	1
	4	1	0	0
	5	1	0	1
	6	1	1	0
	7	1	1	1

Just Remember the formula: Binary combinations = 2^n (n = number of inputs)

Binary in registers

A register is a group of bits, often represented in a grid. The following grid shows an 8-bit register.

1	1	0	1	1	0	1	1
---	---	---	---	---	---	---	---

Computers (or microprocessors) are used to control devices, such as robots, industrial or household environments. Registers are used as part of the control system. Each bit in the register will control a different part of the system. For example, suppose a microprocessor controls a household security system. The register could be setup as follows:

1	0	1	0	1	0	1	1
Kitchen light on	Kitchen light off	Kitchen camera on	Kitchen camera off	Sound alarm	Silence alarm	Send notification	Auto call Police

In this state, the kitchen light would be triggered ON, the camera would be triggered ON, the alarm would be triggered ON, the notification would be sent, and the police would be auto called. When another register is send to the control system, the state of each part can be changed by adjusting the bits in the register.

Computer Registers:

A register is a very small amount of very fast memory that is built into the CPU (central processing unit) in order to speed up its operations by providing quick access to commonly used values. Registers refers to semiconductor devices whose contents can be accessed (i.e., read and written to) at extremely high speeds but which are held there only temporarily (i.e., while in use or only as long as the power supply remains on).

Registers are the top of the memory hierarchy and are the fastest way for the system to manipulate data. Registers are normally measured by the number of bits they can hold, for example, an 8-bit register means it can store 8 bits of data or a 32-bit register means it can store 32 bit of data.

Registers are used to store data temporarily during the execution of a program. Some of the registers are accessible to the user through instructions. Data and instructions must be



put into the system. So we need registers for this.

The basic computer registers with their names, size and functions are listed below

Register Symbol	Register Name	Description
ACC	Accumulator	Processor Register
MDR	Memory Data Register	Hold memory data
IR	Index Register	Holds index data
MAR	Memory Address Register	Holds memory address
PC	Program Counter	Holds address of next instruction
CIR	Current Instruction Register	Holds current instruction

Binary in registers for Robotics

The ROBOT has no intelligence. It cannot think or plan its own activities. The ROBOT is capable only of following a program of instructions that it retrieves from its memory and then executes one instruction at a time.

The ROBOT has a **memory**. In that memory are stored **instructions** that make up a program. In addition the ROBOT is equipped with circuits, which enable it to **fetch** instructions from its memory, interpret or **decode** their meaning, and **execute** them. Let's define further all the terms printed in **boldfaced** type.

Memory - The ROBOT's built-in memory is quite tiny. It contains 32 memory locations, numbered consecutively from 0 to 31. Each memory location is capable of storing a single ROBOT instruction.

Instructions - The ROBOT does not understand human speech or any written language. It responds only to binary numbers, each eight bits (binary digits) in length. Each possible combination of the eight digits is a meaningful instruction to the ROBOT.

Fetch - The ROBOT's electronic circuits cause it to fetch (or retrieve) its instructions from memory, one at a time, and usually in the order in which they are stored. The only exception to this rule involves the GOTO command from the ROBOT's instruction set.

Decode - The ROBOT knows what action to carry out for a given binary number because the ROBOT's hardware contains an instruction decoder. Such a unit is present in one form or another in all digital computers. The instruction decoder is a set of circuits that causes the appropriate actions to be

taken based on the particular binary number instruction that is received as input

Execute - The ROBOT executes the instructions by carrying out the action or command that each individual instruction represents. Each instruction has its own particular purpose and characteristics. Some instructions cause the ROBOT to take an observable action.

Software - The Robot's Language

Each ROBOT instruction consists of an eight-bit binary number. Up to 32 ROBOT instructions may be stored in the ROBOT's memory, one in each memory location. Thus, the maximum number of **statements** in any one ROBOT program is 32.

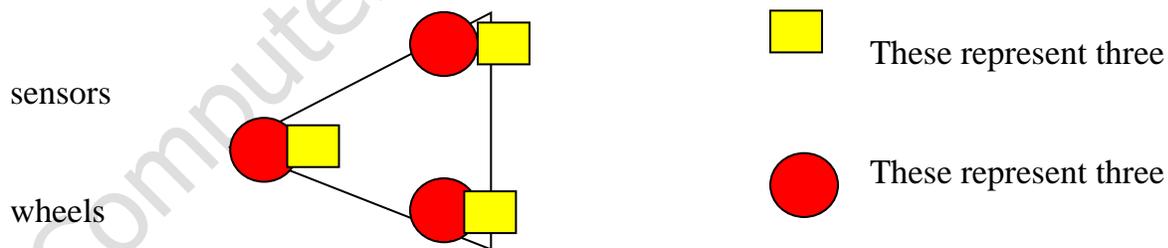
USE OF BINARY SYSTEMS FOR SPECIFIC APPLICATIONS

It is useful to understand that computer systems are capable to perform specific and user controlled and oriented applications like robotics, power systems, and digital instruments and so on. Controlled devices usually contain registers which are made up of binary digits (bits). The following example shows how these registers can be used to control a device.

ROBOTIC SYSTEMS:

Robotic systems rely on processors to access user provided values and data. It is essential to note here that all the data is sent via the processor in a binary format so conversions and storage has to be provided by the processor.

Example 1



Front wheel turns left (on/off)	Front wheel turns Right (on/off)	Rear wheels turn left (on/off)	Rear wheels turn right (on/off)	Forward direction (on/off)	Backward direction (on/off)	Motor (on/off)	Error (Object in the way=1, clear path=0)
---------------------------------	----------------------------------	--------------------------------	---------------------------------	----------------------------	-----------------------------	----------------	---

(1=on and 0=off)

Therefore if the input is for example: **1 0 1 0 1 0 1 0**

1	0	1	0	1	0	1	0
Front wheel turn left	-	Back wheel turn left	-	Direction is forward	-	The motors are on	-

Hence, the trolley is moving forward and turning left.

(i) What does this register mean?

“0 0 0 1 0 1 1 1”

(ii) How would the following be represented using the above register?

-  front wheel turning right
-  back wheels turning left
-  moving in a forward direction
-  motors on
-  no object in its path

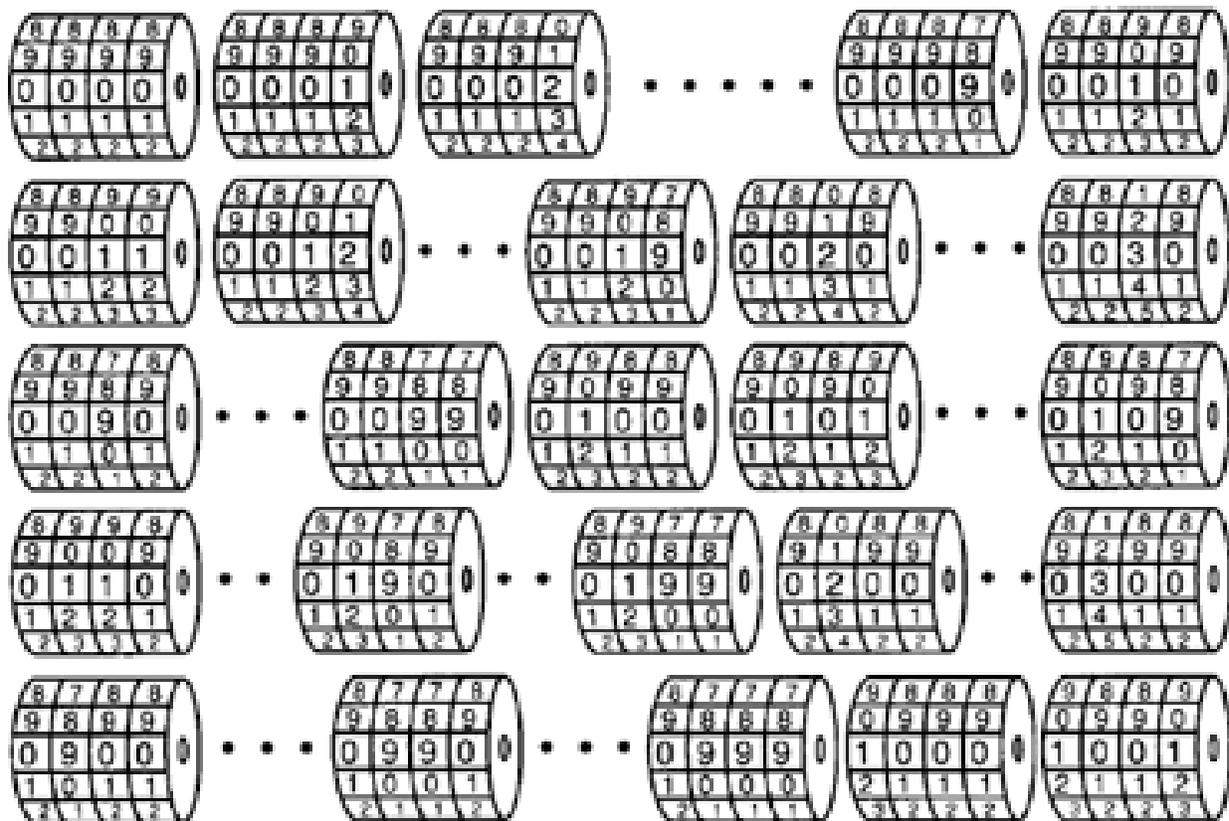
Answers

-  front wheel **not** turning left or right
-  Rear wheels turning right
-  going in backward direction
-  motors on
-  Error – object in path

So the vehicle is going nowhere

(ii) **0 1 1 0 1 0 1 0**

Denary in Counting Systems:



DECIMAL IN TENS

1
1 2 3 4 5 6 7 8 9 0

Systems of counting

Before electronic digital devices were invented, we used counters with little wheels that carried numbers. The numbers that showed through the front window were like those that electronic digital devices display. If you took the cover off the rest of the wheel, you could see how it work, which helped you understand number systems.

The right-most wheel counted from 0 to 9 on a decimal system. When it came to 9, it would move from 9 to 0, and move the next wheel from 0 to 1. Every time the first wheel passed from 9 to 0, the next wheel would advance 1 more, until it returned to 9. Then, two wheels would read 99. As the first wheel moved from 9 to

0 this time, the next one would also move from 9 to 0, and the third wheel would move from 0 to 1, making it read 100.

Binary counting:

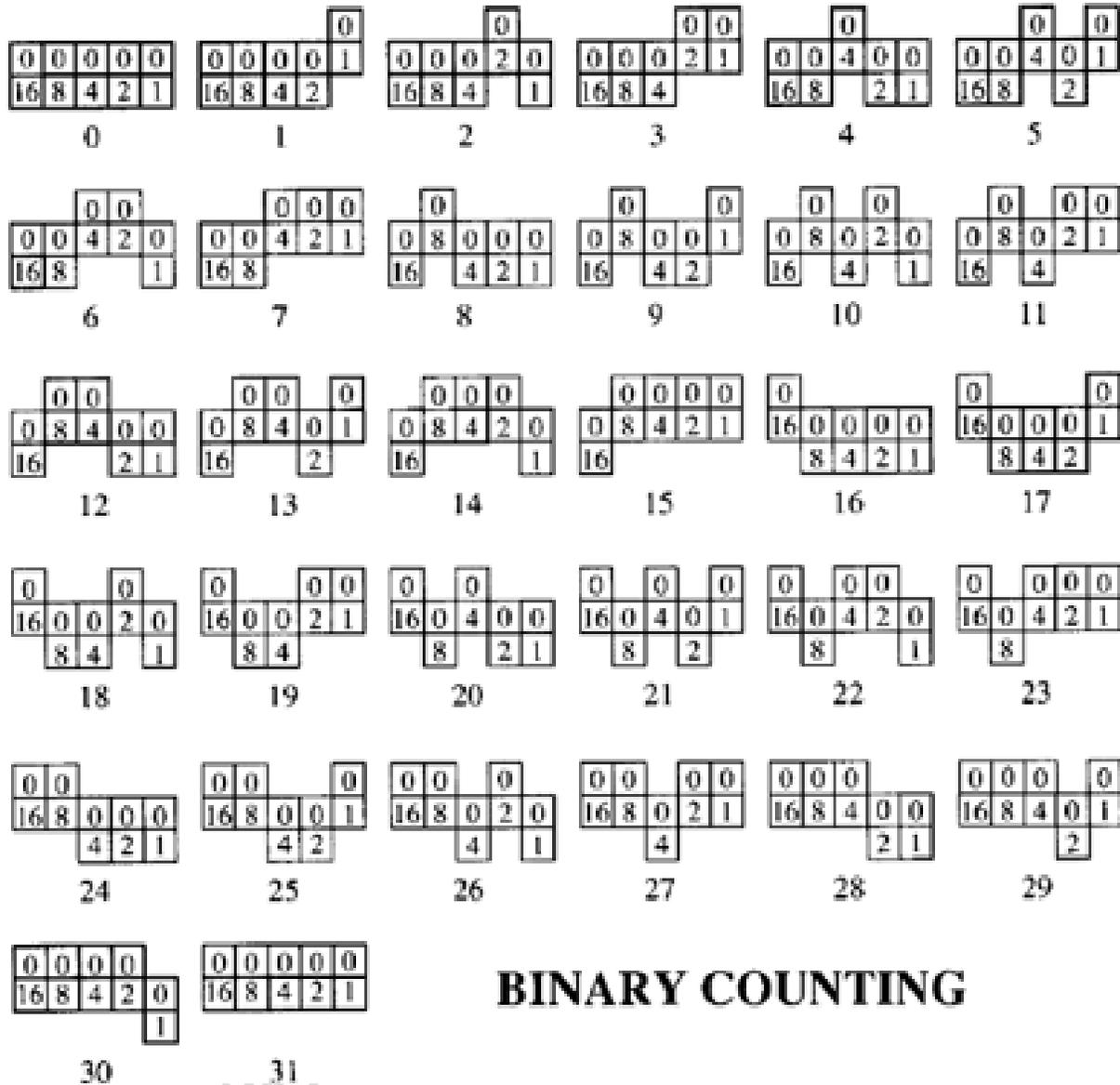
The difficulty about working in binary is that each place has only two "states," which are 0 and 1. You don't count "up to" something and then move to the next place. If you already have 1, the next 1 puts it back to 0 and passes a 1 to the next place. If you have a row of 1s, then adding another 1 shifts them all back to 0, and passes a 1 to the next place (from right to left).

In the window panel here, the decimal number equivalent replaces the binary numbers. In the binary system, every place would be either a 1 or a 0.

Considering Place Value see the figure below

Computers(2210) with Majid Tahir





SENSORS IN PRINTING DEVICES:

EXAMPLE#2:

Three sensors are attached to a printing device, with three alarms attached to the sensors.

- The first sensor, "A," detects if the device needs ink.
- The second sensor, "B," detects if the device needs repair.
- The third sensor, "C," detects if the device has jammed paper. If the device jams or needs repair, alarm 1 sound. If the device jams or is short on ink, alarm 2 sounds. If two or more problems occur at once, alarm 3 sounds.

It can now be implied that:



- A=1 refers to “low ink”
- B=1 refers to “Device needs repair”
- C=1 refers to “device should jam”

The outputs to the system are as follows:

- A1: alarm 1 sounds if B=1 or C=1 (either of B=1 or C=1 will result in A1)
- A2: Alarm 2 sounds if C=1 or A=1 (either of A or C being true will result in A2)
- A3= any two or more of A, B, and C being ‘1’ will result in A3.

Let us now look at the cases that will form:

A	B	C	A1	A2	A3
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Questions:

- 1) What binary values can the register hold to alert the user of a “low ink” situation?
- 2) The register is holding “010”. Which alarm will sound?
- 3) Is the system accurate in telling the user of exactly which problem is being occurred? Point out anyone situation where user is misguided?

Answers:

- 1) 010, 111, 111, 111
- 2) A2 will sound.
- 3) No. System has the same alarm for one problem or more than one problem. E.g. 111 could mean that all three problems have occurred but 111 can be received even if printer is “full on ink” or not having “jammed paper.”

References:

- <http://www.bfoit.org/itp/ComputerContinuum/RobotComputer.html>
https://en.wikibooks.org/wiki/GCSE_Computer_Science/Binary_representation
<http://bssbmi.com/olevel/computer-science-2210/class-9/binary-systems/>
<http://www.math10.com/en/algebra/systems-of-counting/binary-system.html>