Lecture 4

DATA STORAGE

Data Storage

Bits and Their Storage

Representing Information as Bit Patterns

Storing Numbers

Bits and Bit Patterns

Bit: Binary Digit (0 or 1)

Bit Patterns are used to represent information.

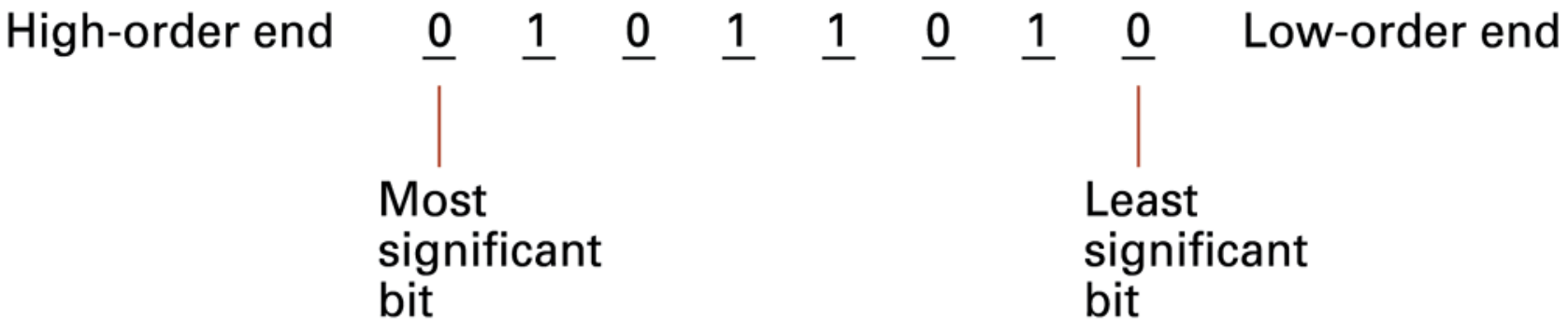
- Numbers
- Text characters
- Images
- Sound
- And others

Main Memory Cells

Cell: A unit of main memory (typically 8 bits which is one **byte**)

- **Most significant bit:** the bit at the left (high-order) end of the conceptual row of bits in a memory cell
- **Least significant bit:** the bit at the right (low-order) end of the conceptual row of bits in a memory cell

The organization of a byte-size memory cell

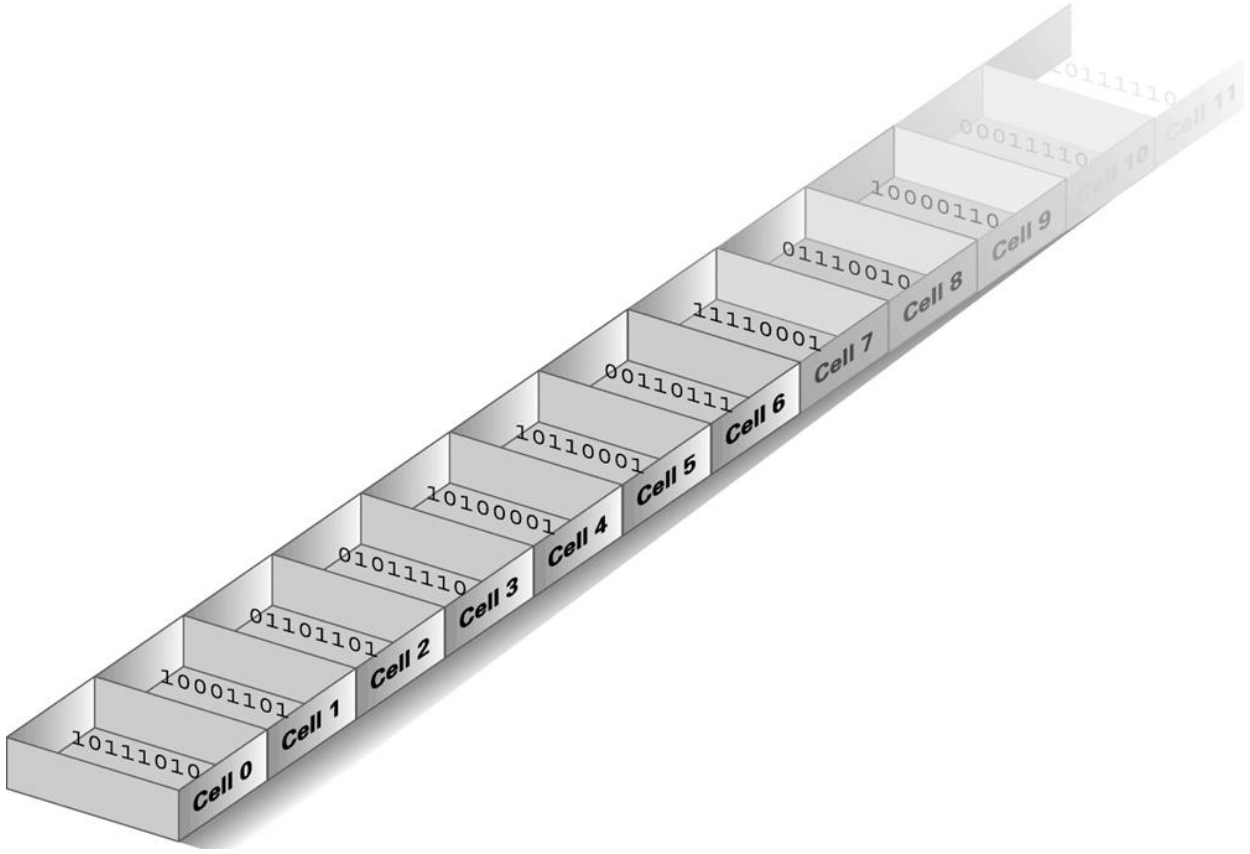


Main Memory Addresses

Address: A “name” that uniquely identifies one cell in the computer’s main memory

- The names are actually numbers.
- These numbers are assigned consecutively starting at zero.
- Numbering the cells in this manner associates an order with the memory cells.

Memory cells arranged by address



Mass Storage

Typically larger than main memory

Typically less volatile than main memory

Typically slower than main memory

Mass Storage Systems

Magnetic Systems

- Disk
- Tape

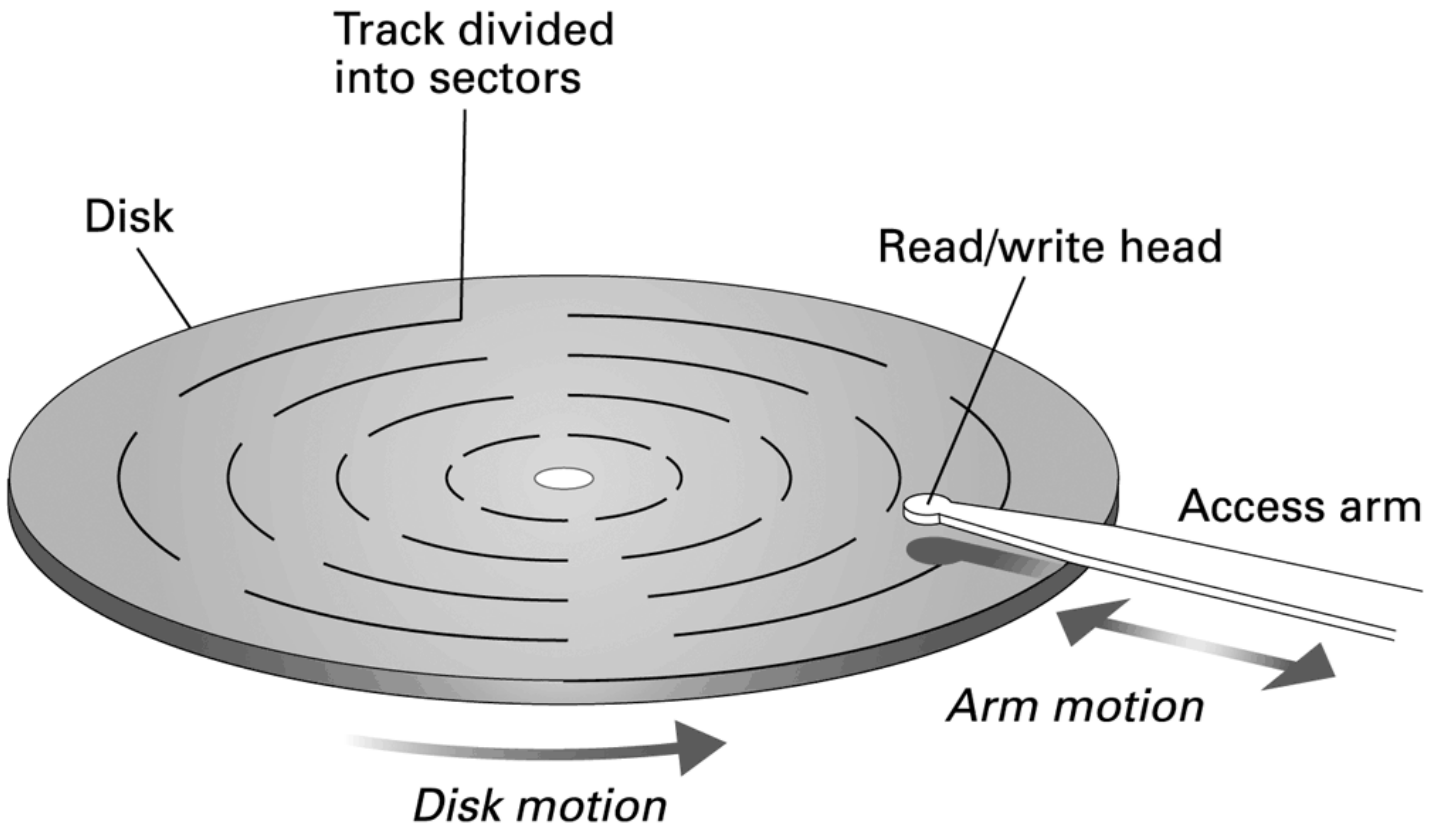
Optical Systems

- CD
- DVD

Flash Technology

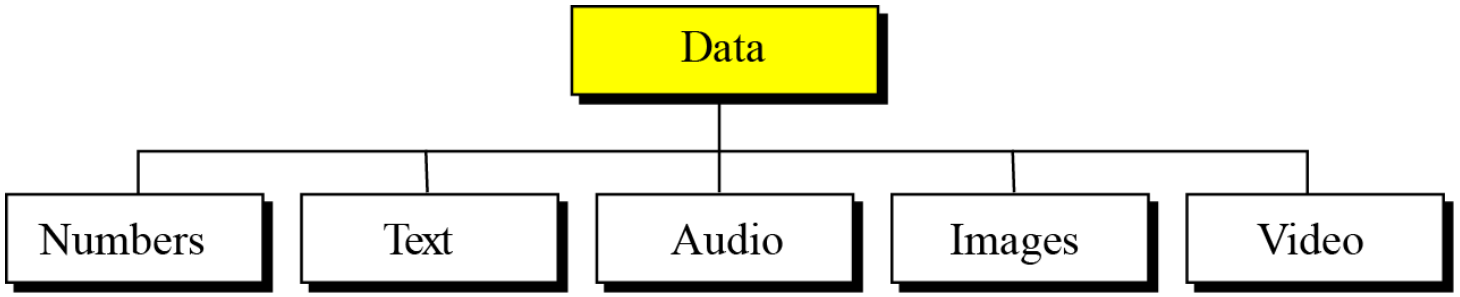
- Flash Drives
- Secure Digital (SD) Memory Card

A magnetic disk storage system



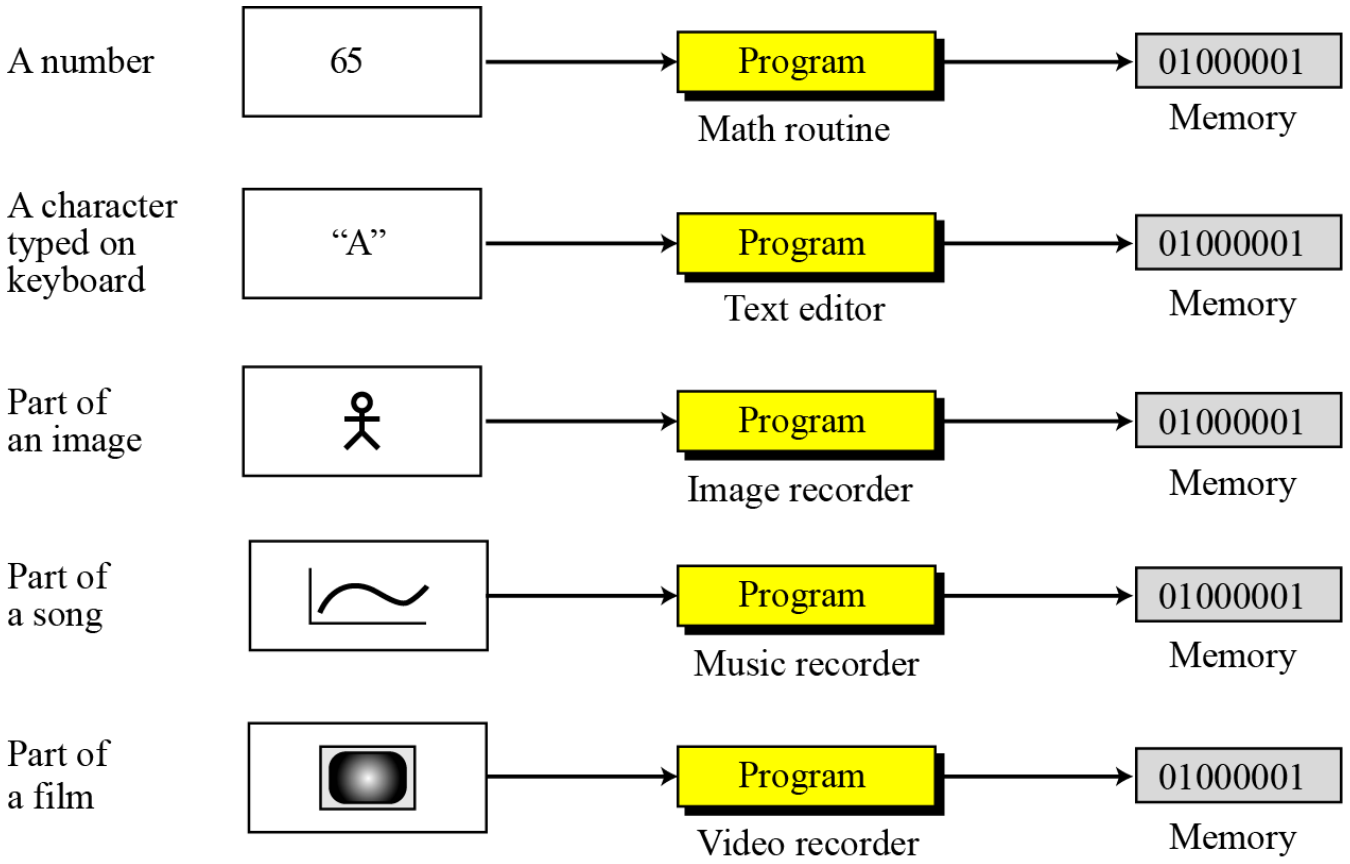
Types of data

- Data today comes in different forms including numbers, text, audio, image and video.



The computer industry uses the term “multimedia” to define information that contains numbers, text, images, audio and video.

Storage of different data types



Representing Numeric Values

Binary notation: Uses bits to represent a number in base two

Limitations of computer representations of numeric values

- Overflow: occurs when a value is too big to be represented
- Truncation: occurs when a value cannot be represented accurately

Storing Numbers

A number is changed to the binary system before being stored in the computer memory.

Unsigned representation

An unsigned integer is an integer that can never be negative and can take only 0 or positive values. Its range is between 0 and positive infinity.

An input integer is changed device stores an unsigned integer using the following steps:

1. Convert The number to binary.
2. If the number of bits is less than n, 0s are added to the left.

Example

Store 7 in an 8-bit memory location using unsigned representation.

Solution

First change the integer to binary, $(111)_2$. Add five 0s to make a total of eight bits, $(00000111)_2$. The integer is stored in the memory location.

Change 7 to binary	→						1	1	1
Add five bits at the left	→	0	0	0	0	0	1	1	1

Example

Store 258 in a 16-bit memory location

Solution

First change the integer to binary $(100000010)_2$. Add seven 0s to make a total of sixteen bits, $(0000000100000010)_2$. The integer is stored in the memory location.

Change 258 to binary	→								1	0	0	0	0	0	0	1	0
Add seven bits at the left	→	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0

Example

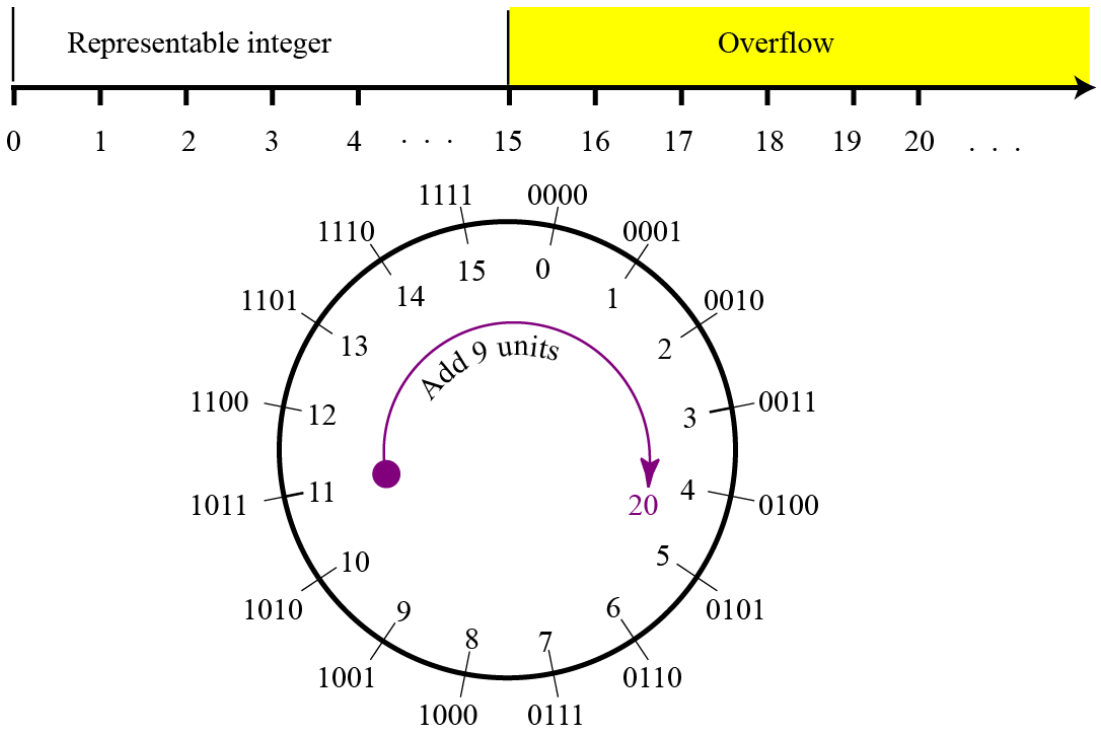
What is returned from an output device when it retrieves the bit string 00101011 stored in memory as an unsigned integer?

Solution

The binary integer is converted to the unsigned integer 43.

Overflow in unsigned integers

What happens if we try to store an integer that is larger than $2^4 - 1 = 15$ in a memory location that can only hold four bits.



Overflow in unsigned integers

If you are representing numbers using 4 bits as unsigned binary, the minimum value is 0 (i.e., $\mathbf{0000}_2$, while the maximum value is 15 ($\mathbf{1111}_2$).

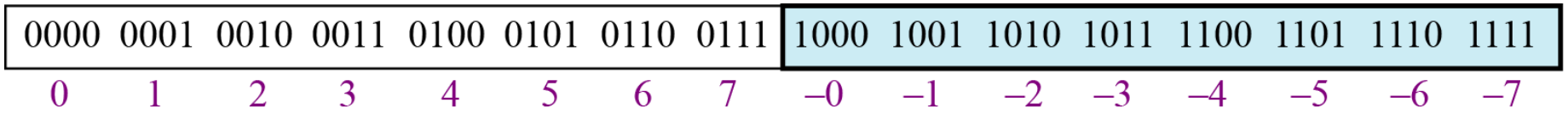
Any operation that results in a number larger than 15 or smaller than 0 has *overflowed*.

For example, if you summed 14 + 14 (in UB, that's 1110 + 1110), the resulting value is 28, but the value 28 has no representation using 4 bits (since the maximum representable value is 15). Thus, there is overflow.

You might wonder, well, why not use 5 bits? Hardware has fixed number of bits. If it's 4 bits, then that's all you have to work with.

Sign-and-magnitude representation

In this method, the available range for unsigned integers (0 to $2^n - 1$) is divided into two equal sub-ranges. The first half represents positive integers, the second half, negative integers.



In sign-and-magnitude representation, the leftmost bit defines the sign of the integer. If it is 0, the integer is positive. If it is 1, the integer is negative.

Example

Store +28 in an 8-bit memory location using sign-and-magnitude representation.

Solution

The integer is changed to 7-bit binary. The leftmost bit is set to 0. The 8-bit number is stored.

Change 28 to 7-bit binary		0	0	1	1	1	0	0
Add the sign and store	0	0	0	1	1	1	0	0

Example

Store –28 in an 8-bit memory location using sign-and-magnitude representation.

Solution

The integer is changed to 7-bit binary. The leftmost bit is set to 1. The 8-bit number is stored.

Change 28 to 7-bit binary		0	0	1	1	1	0	0
Add the sign and store	1	0	0	1	1	1	0	0

Example

Retrieve the integer that is stored as 01001101 in sign-and-magnitude representation.

Solution

Since the leftmost bit is 0, the sign is positive. The rest of the bits (1001101) are changed to decimal as 77. After adding the sign, the integer is +77.

Example

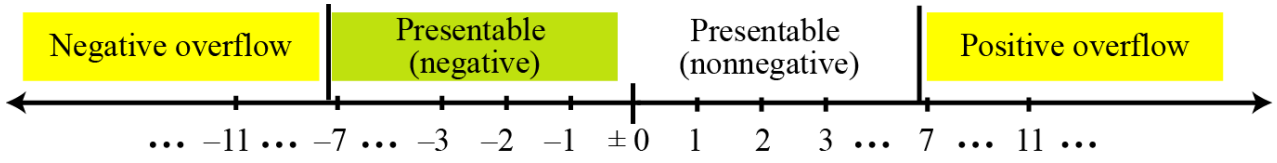
Retrieve the integer that is stored as 10100001 in sign-and-magnitude representation.

Solution

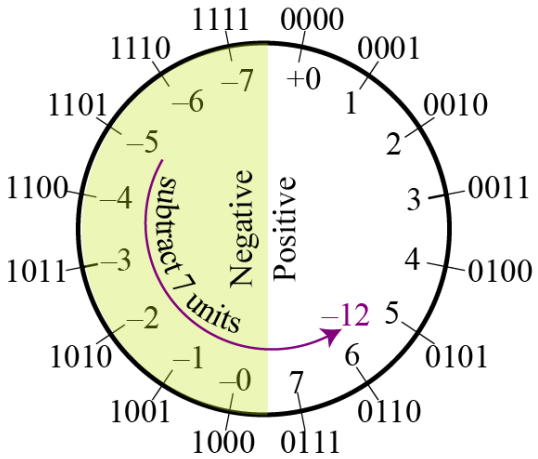
Since the leftmost bit is 1, the sign is negative. The rest of the bits (0100001) are changed to decimal as 33. After adding the sign, the integer is −33.

Overflow in sign-and-magnitude representation

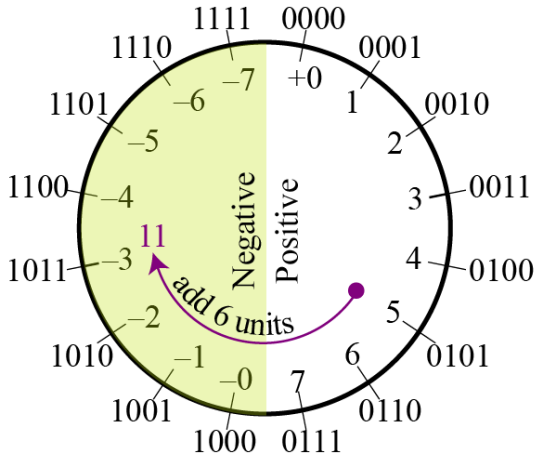
Both positive and negative overflow when storing an integer in sign-and-magnitude representation using a 4-bit memory location.



a. Linear characteristic of an integer in sign-and-magnitude format



b. Negative overflow



c. Positive overflow

Two’s complement representation

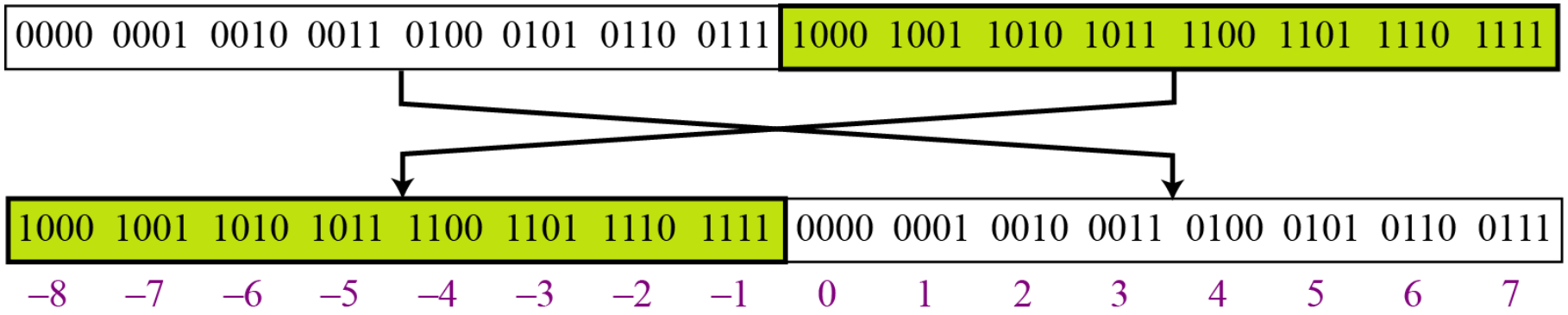
Almost all computers use two’s complement representation to store a signed integer in an n-bit memory location.

In this method, the available range for an unsigned integer of (0 to 2n − 1) is divided into two equal sub-ranges.

The first sub-range is used to represent nonnegative integers, the second half to represent negative integers.

The bit patterns are then assigned to negative and nonnegative (zero and positive) integers.

Two’s complement representation



In two’s complement representation, the leftmost bit defines the sign of the integer. If it is 0, the integer is positive. If it is 1, the integer is negative.

One’s Complementing

Before we discuss this representation further, we need to introduce two operations.

The first is called one’s complementing or taking the one’s complement of an integer. The operation can be applied to any integer, positive or negative. This operation simply reverses (flips) each bit. A 0-bit is changed to a 1-bit, a 1-bit is changed to a 0-bit.

Example

The following shows how we take the one’s complement of the integer 00110110.

Original pattern	0	0	1	1	0	1	1	0
After applying one’s complement operation	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**

Example

The following shows that we get the original integer if we apply the one's complement operations twice.

Original pattern	0	0	1	1	0	1	1	0
One's complementing once	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
One's complementing twice	0	0	1	1	0	1	1	0

Two's Complementing

The second operation is called two's complementing or taking the two's complement of an integer in binary. This operation is done in two steps. First, we copy bits from the right until a 1 is copied; then, we flip the rest of the bits.

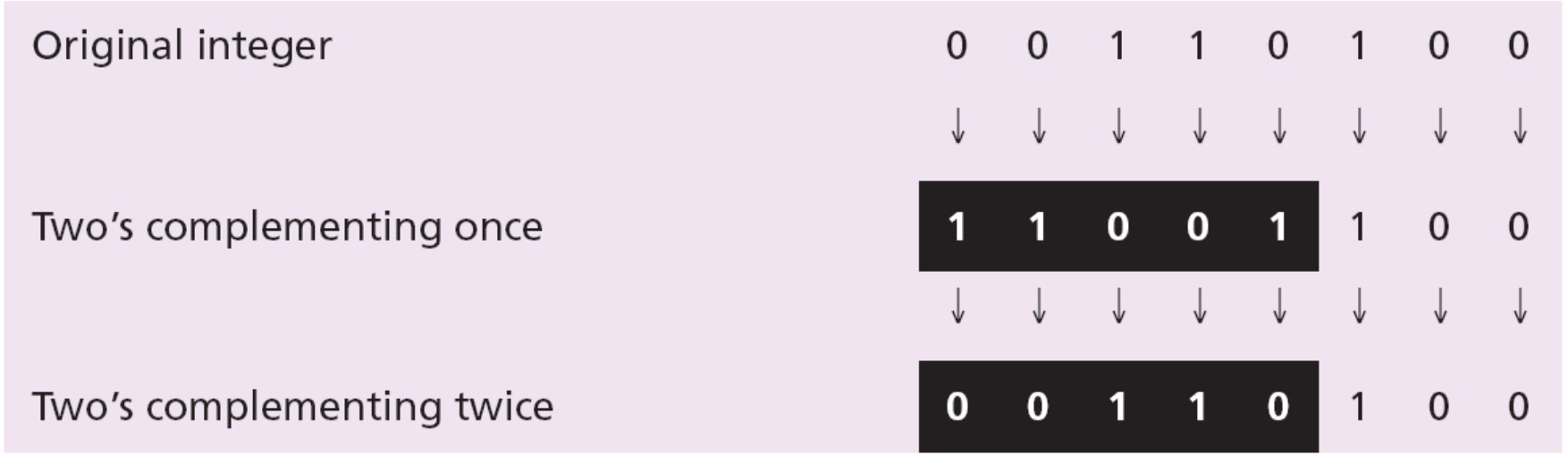
Example

The following shows how we take the two's complement of the integer 00110100.

Original integer	0	0	1	1	0	1	0	0
	↓	↓	↓	↓	↓	↓	↓	↓
Two's complementing once	**1**	**1**	**0**	**0**	**1**	1	0	0

Example

The following shows that we always get the original integer if we apply the two's complement operation twice.



An alternative way to take the two's complement of an integer is to first take the one's complement and then add 1 to the result.

Example

Store the integer 28 in an 8-bit memory location using two's complement representation.

Solution

The integer is positive (no sign means positive), so after decimal to binary transformation no more action is needed. Note that five extra 0s are added to the left of the integer to make it eight bits.

Change 28 to 8-bit binary	0	0	0	1	1	1	0	0

Example

Store −28 in an 8-bit memory location using two's complement representation.

Solution

The integer is negative, so after changing to binary, the computer applies the two's complement operation on the integer.

Change 28 to 8-bit binary	0	0	0	1	1	1	0	0
	↓	↓	↓	↓	↓	↓	↓	↓
Apply two's complement operation	**1**	**1**	**1**	**0**	**0**	1	0	0

Example

Retrieve the integer that is stored as 00001101 in memory in two's complement format.

Solution

The leftmost bit is 0, so the sign is positive. The integer is changed to decimal and the sign is added.

Leftmost bit is 0. The sign is positive	0	0	0	0	1	1	0	1
Integer changed to decimal								13
Sign is added (optional)								+13

Example

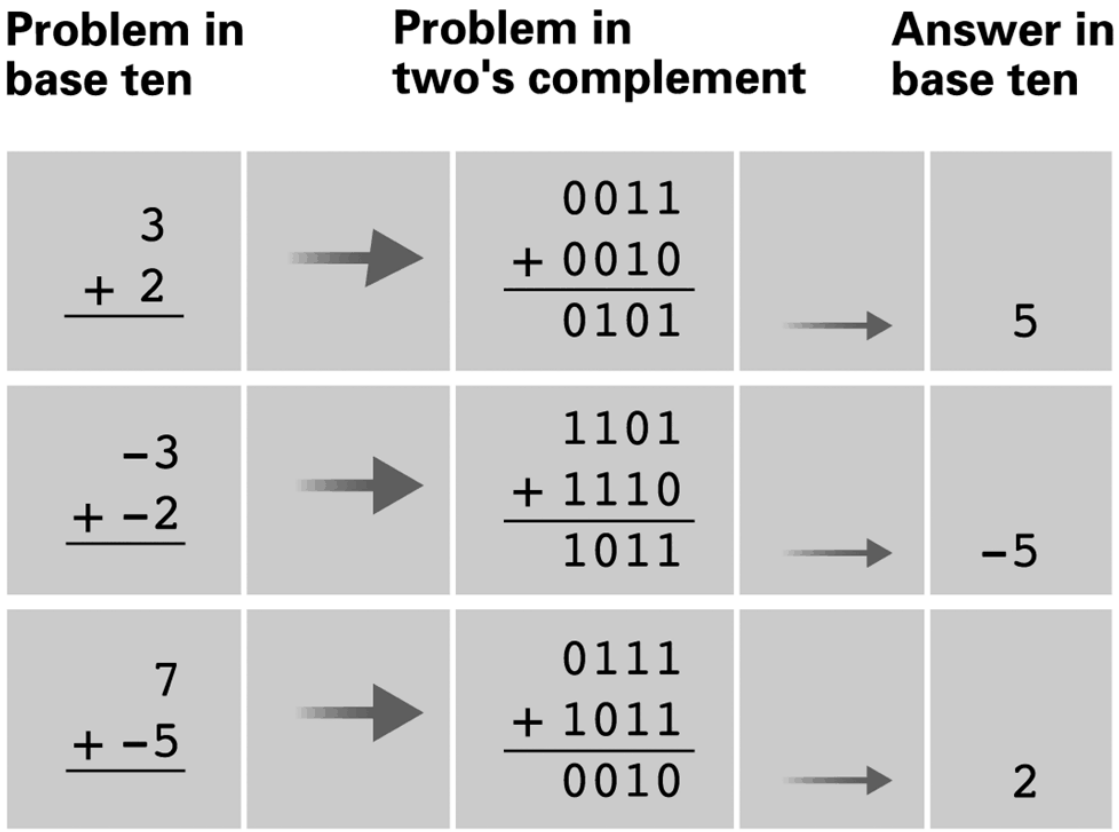
Retrieve the integer that is stored as 11100110 in memory using two's complement format.

Solution

The leftmost bit is 1, so the integer is negative. The integer needs to be two's complemented before changing to decimal.

Leftmost bit is 1. The sign is negative	1	1	1	0	0	1	1	0
	↓	↓	↓	↓	↓	↓	↓	↓
Apply two's complement operation	**0**	**0**	**0**	**1**	**1**	**0**	1	0
Integer changed to decimal								26
Sign is added								–26

Addition problems converted to two's complement notation



Addition problems converted to two's complement notation

56 + (−19)

Solution

56 = 111000 = 0111000 (Did you forget the sign bit?)

19 = 0010011

Two's complement of 0010011 = 1101101

```
   1 1 1        carry row
   0 1 1 1 0 0 0
+  1 1 0 1 1 0 1
 ---------------
   1 0 1 0 0 1 0 1
```

After truncating the overflow '1', we get the positive number 0100101, which equals 37.

References

- Computer Fundmental –Pradeep K. Sinha & Priti Sinha

  Foundations of Computer Science @ Cengage Learning
- Computer Science: An Overview -Eleventh Edition –by J. Glenn Brookshear

ANNOUNCEMENTS

❖ The **Lecture 4** were posted Online Facebook Group last week.

Please read them carefully.

❖**Sheet # 3** were posted online this week.

❖**Submissions of Sheet #3** is during next week's Labs

❖**Quiz # 3** will be held in the week after

Thank you