Introduction to Computer Science
Lesson 3

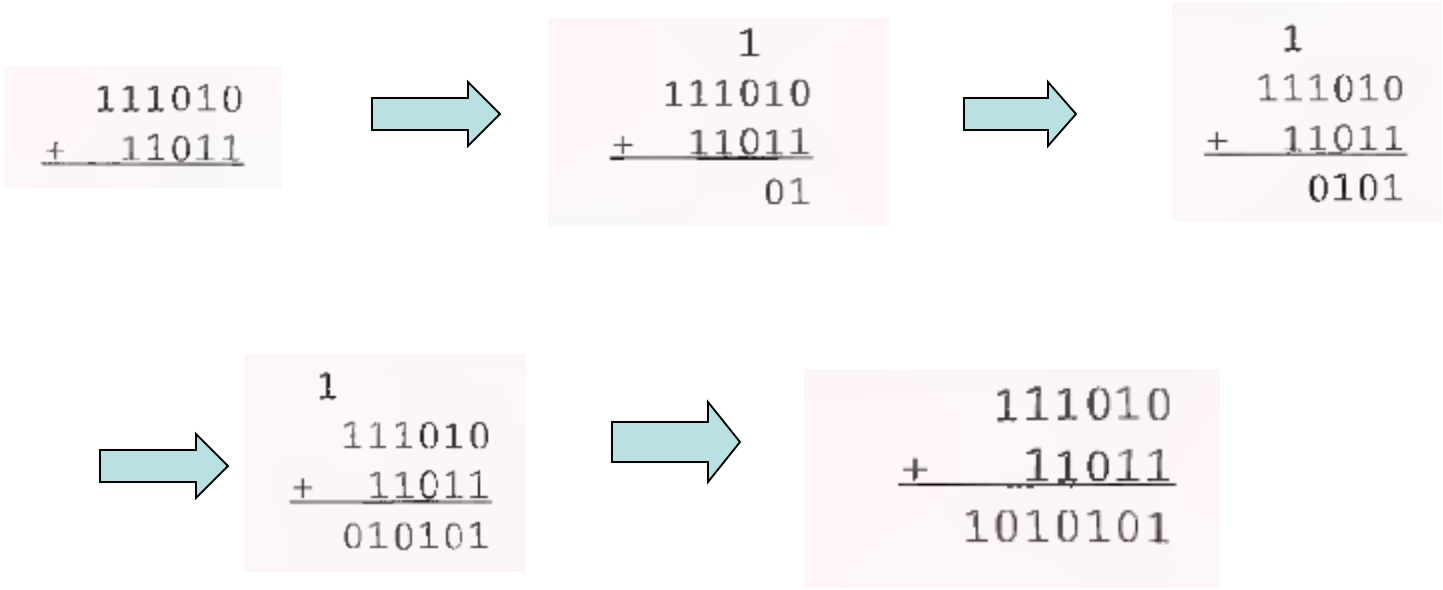
BSc in Computer Science
University of New York, Tirana

Assoc. Prof. Marenglen Biba

From last lesson: the binary addition facts

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$

Binary addition



Storing Integers

- **Two's complement notation:** The most popular means of representing integer values
- **Excess notation:** Another means of representing integer values
- Both can suffer from **overflow** errors.

Two's Complement Notation

- The most popular system for representing integers within today's computers is **two's complement notation**.
- This system uses a **fixed number of bits** to represent each of the values in the system.
- In today's equipment, it is common to use a two's complement system in which each value is represented by a pattern of **32 bits or 64 bits**.
- Such a large system allows a wide range of numbers to be represented.

Figure 1.21 Two's complement notation systems

- Note that in a two's complement system, the **leftmost bit** of a bit pattern indicates the **sign** of the value represented.
- Thus, the leftmost bit is often called the sign bit. In a two's complement system, **negative values** are represented by the patterns whose sign bits are **1**; **nonnegative values** are represented by patterns whose sign bits are **0**.

a. Using patterns of length three

Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4

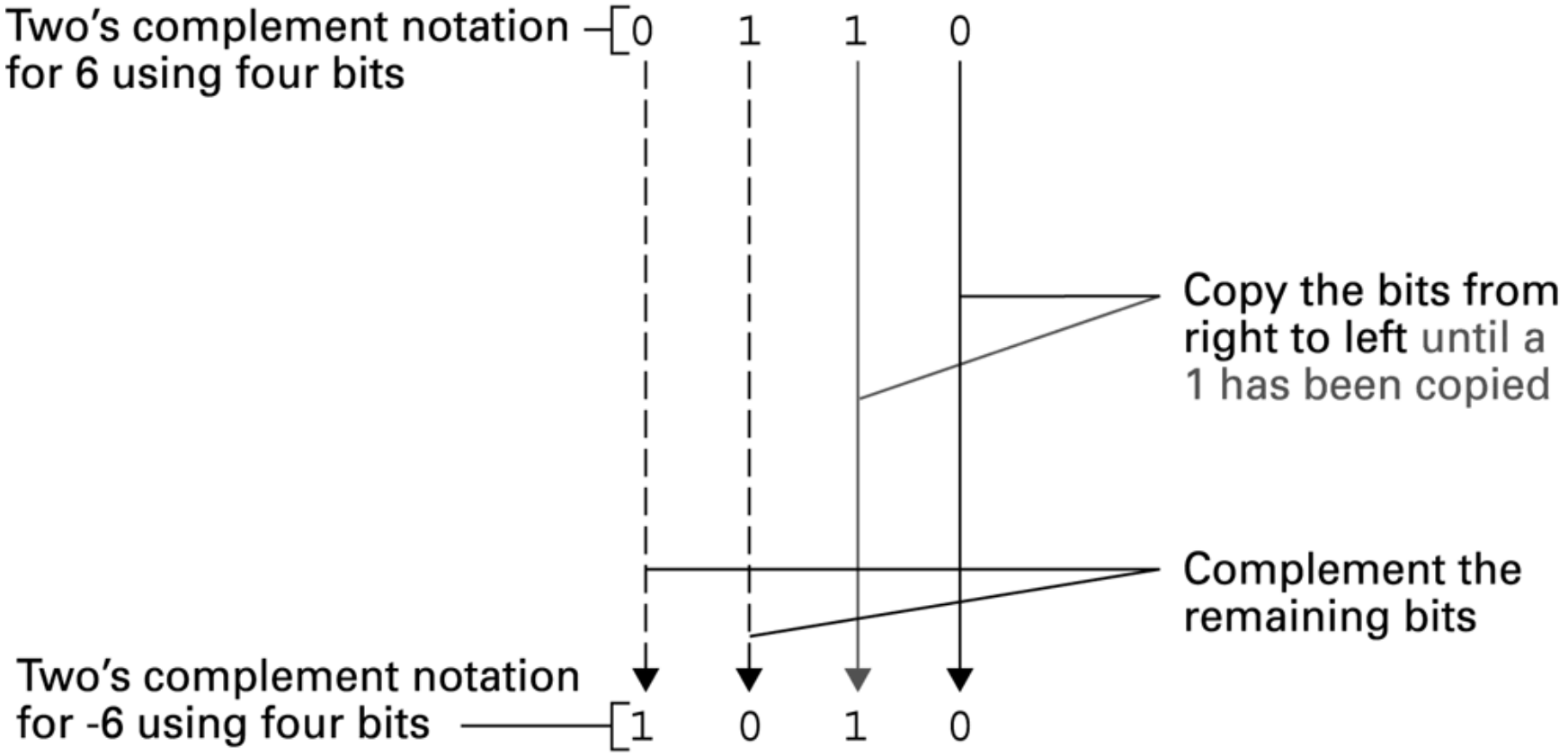
b. Using patterns of length four

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

Properties of two's complement notation

- In a two's complement system, there is a **convenient relationship** between the patterns representing positive and negative values of the same magnitude.
- **They are identical when read from right to left, up to and including the first 1.**
- **From there on, the patterns are complements of one another.**
- The complement of a pattern is the pattern obtained by changing all the 0s to 1s and all the 1s to 0s; 0110 and 1001 are complements.
- For example, in the four-bit system the patterns representing 2 and -2 both end with 10, but the pattern representing 2 begins with 00, whereas the pattern representing -2 begins with 11.
- This observation leads to an **algorithm** for converting back and forth between bit patterns representing positive and negative values of the same magnitude. **We merely copy the original pattern from right to left until a 1 has been copied, then we complement the remaining bits as they are transferred to the final bit pattern.**

Figure 1.22 **Coding the value -6 in two's complement notation using four bits**

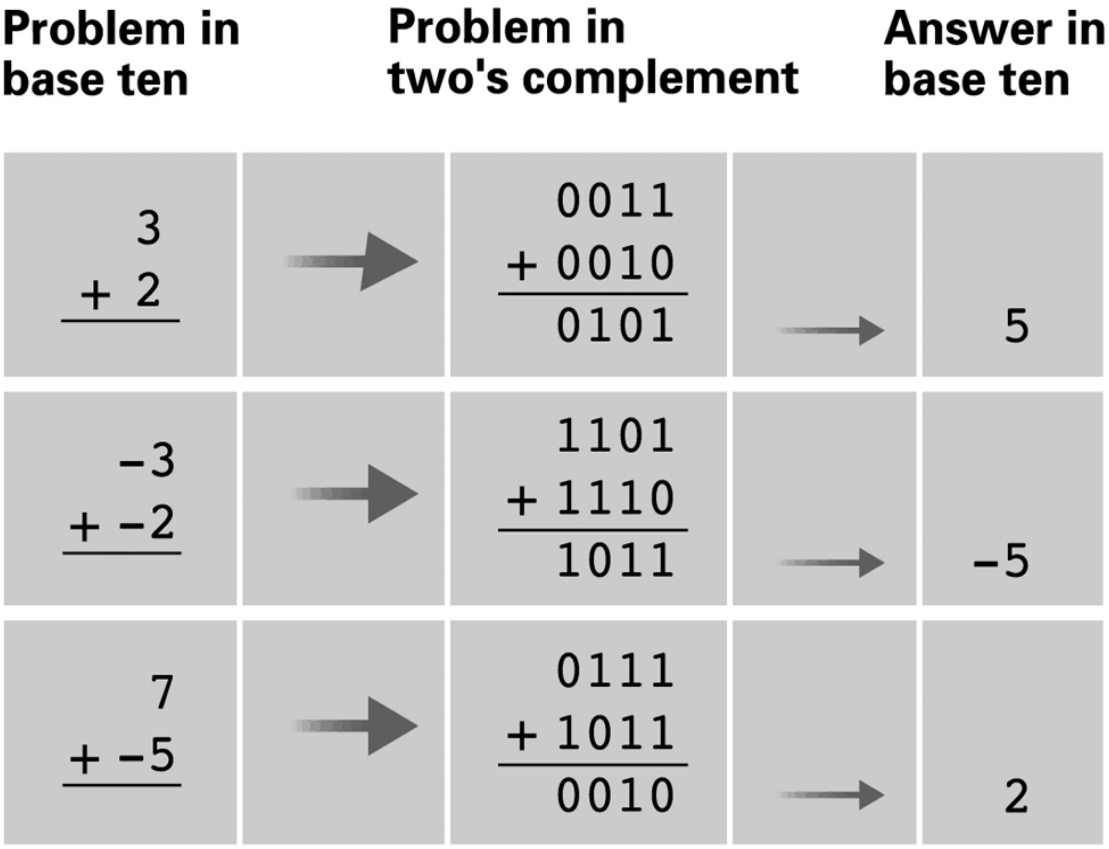


Addition in two's complement

- To add values represented in **two's complement** notation, we apply the same algorithm that we used for binary addition, except that all bit patterns, including the answer, are the same length.

- This means that when adding in a two's complement system, any **extra bit generated on the left of the answer by a final carry must be truncated**.
  - Thus "adding" 0101 and 0010 produces 0111, and "adding" 0111 and 1011 results in 0010 (0111 + 1011 = 10010,which is truncated to 0010).

Figure 1.23 Addition problems converted to two's complement notation



The Problem of Overflow

- One problem we have avoided in the preceding examples is that in any two's complement system there is a **limit to the size of the values that can be represented.**

- When using two's complement with patterns of four bits, the largest positive integer that can be represented is 7, and the most negative integer is -8.

- In particular, the value 9 can not be represented, which means that we cannot hope to obtain the correct answer to the problem 5 + 4. In fact, the result would would appear as -7. **This phenomenon is called overflow.**

- That is, **overflow** is the problem that occurs when a computation produces a value that falls outside the range of values that can be represented. When using two's complement notation, this might occur **when adding two positive values or when adding two negative values.**

Overflow

- Of course, because most computers use two's complement systems with **longer bit patterns** than we have used in our examples, larger values can be manipulated **without causing an overflow**.

- Today, it is common to use patterns of 32 bits for storing values in two's complement notation, allowing for positive values as large as 2,147,483,647 to accumulate before overflow occurs.

- If still larger values are needed, **longer bit patterns** (**64 bits**) can be used or perhaps the **units of measure can be changed.**
  - For instance, finding a solution in terms of miles instead of inches results in smaller numbers being used and might still provide the accuracy required.

Overflow ☺

- On September 19, 1989, a hospital computer system malfunctioned after years of reliable service.
- Close inspection revealed that this date was 32,768 ($=2^{15}$) days after January 1, 1900, and the machine was programmed to compute dates based on that starting date.
- Thus, because of overflow, September 19, 1989 produced a negative value - a phenomenon for which the computer's program was not designed to handle.

Binary subtraction

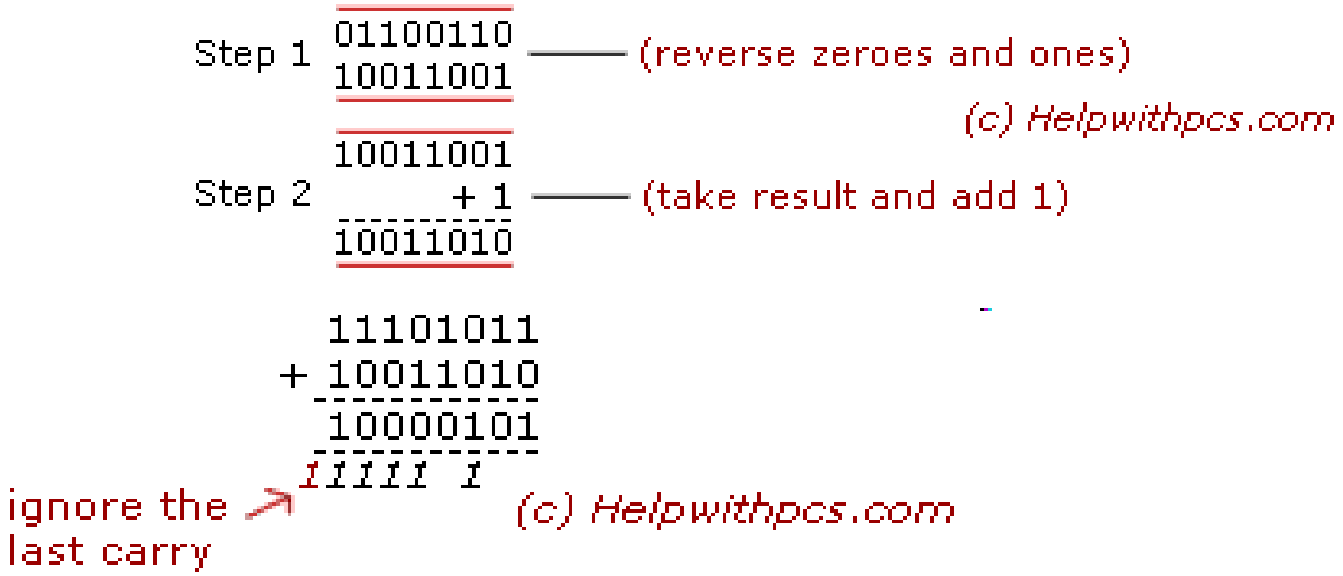
- The most common way of subtracting binary numbers is done by first taking the second value (the number to be subtracted) and apply what is known as **two's complement**, this is done in two steps:
  - 1. complement each digit in turn (change 1 for 0 and 0 for 1).
  - 2. add 1 (one) to the result.
  - note**:** step 1. by itself is known as **one's complement**.

- By applying these steps you are effectively **turning the value into a negative number**, and as when dealing with decimal numbers, if you add a negative number to a positive number then you are effectively subtracting to the same value.

  In other words 25 + (-8) = 17, which is the same as writing 25 - 8 = 17.

Subtraction example

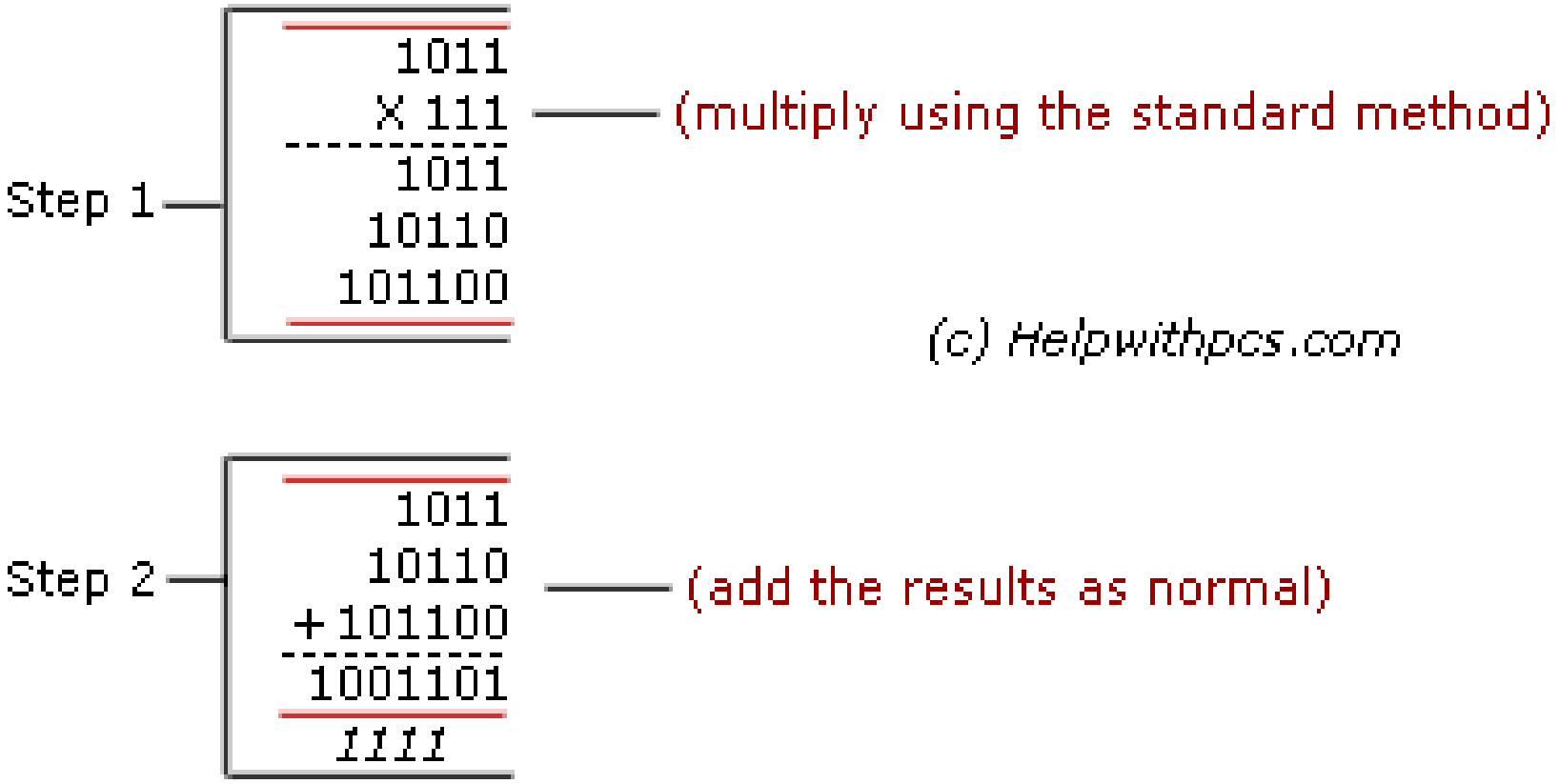
- An example, let's do the following subtraction **11101011 - 01100110** (235_{10} - 102_{10}).



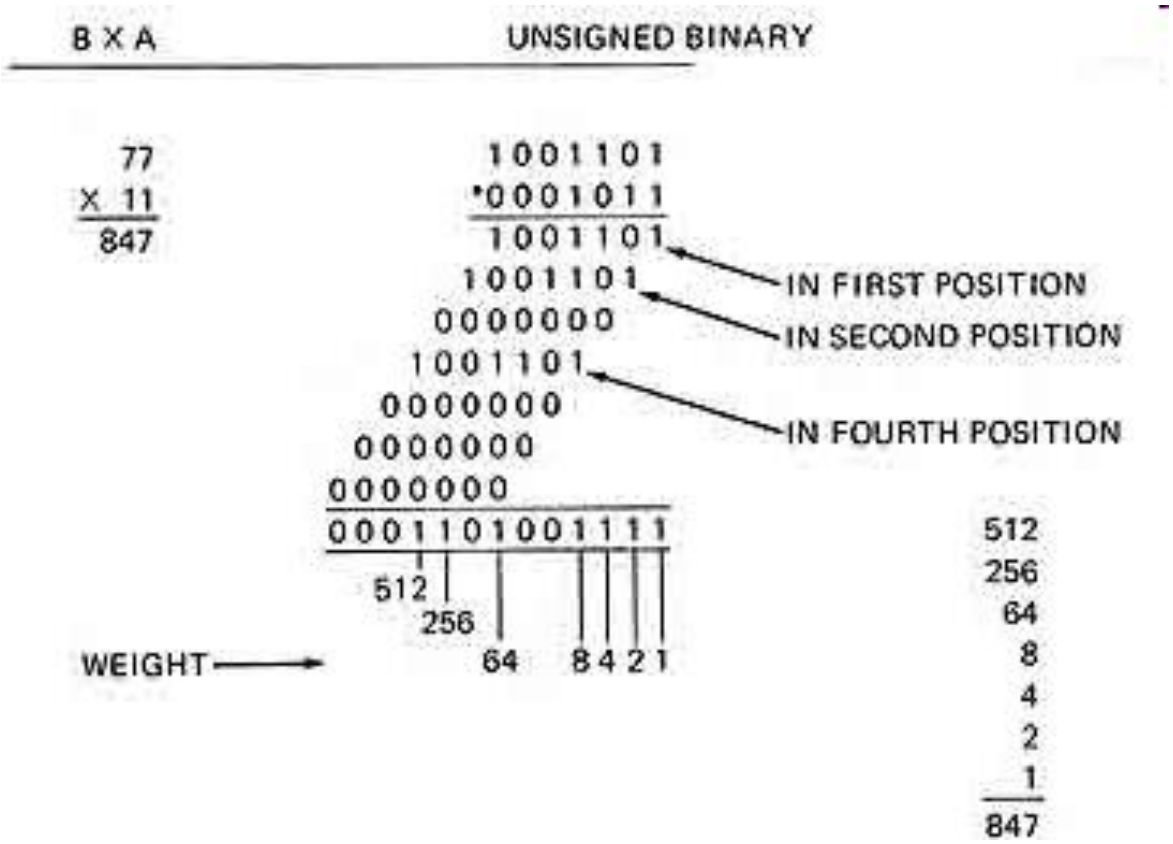
This gives us **10000101**, now we can convert this value into decimal, which gives $\mathbf{133_{10}}$

So the full calculation in decimal is 235_{10} - 102_{10} = 133_{10} (correct !!)

Binary multiplication



Binary multiplication



Excess notation

- Another method of representing integer values is **excess notation**. As is the case with two's complement notation, each of the values in an excess notation system is represented by a **bit pattern of the same length.**

- We observe that the first pattern with a **1 as its most significant bit** appears approximately halfway through the list.
  - We pick this pattern to represent zero; the patterns following this are used to represent 1, 2, 3, . . . ; and the patterns preceding it are used for -1, -2, -3, . . .

- The resulting code, when using patterns of length four, is shown in Figure 1.24. There we see that the value 5 is represented by the pattern 1101 and -5 is represented by 0011. **(Note that the difference between an excess system and a two's complement system is that the sign bits are reversed.)**

Figure 1.24 An excess eight conversion table

Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	−1
0110	−2
0101	−3
0100	−4
0011	−5
0010	−6
0001	−7
0000	−8

- The system represented in Figure 1.24 is known as **excess eight notation**.
- In each case, you will find that the binary interpretation **exceeds** the excess notation interpretation by the value 8.
- For example, the pattern **1100** in binary notation represents the **value 12,** but in our excess system it represents 4;
- 0000 in binary notation represents 0, but in the excess system it represents negative 8.
- In a similar manner, an excess system based on patterns of length five would be called excess 16 notation, because the pattern 10000 for instance, would be used to represent zero rather than representing its usual value of 16.

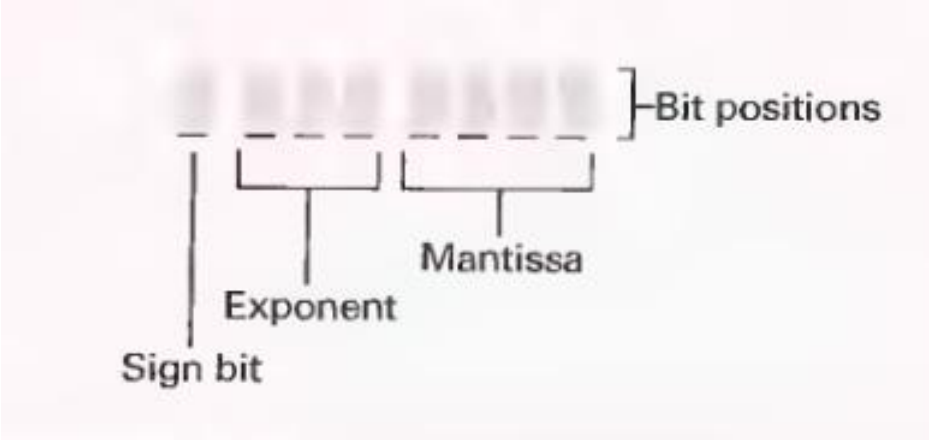
Figure 1.25 An excess notation system using bit patterns of length three

- Excess four notation

Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

Floating point

- In contrast to the storage of integers, the storage of a value with a fractional part requires that we store not only the pattern of 0s and 1s representing its binary representation but also the position of the radix point.
- A popular way of doing this is based on scientific notation and is called **floating-point** notation.
- We first designate the high-order bit of the byte as the **sign bit**. Once again, a 0 in the sign bit will mean that the value stored is nonnegative, and a 1 will mean that the value is negative.
- Next, we divide the remaining seven bits of the byte into two groups, or fields, the **exponent field** and the **mantissa field**. Let us designate the three bits following the sign bit as the exponent field and the remaining four bits as the mantissa field.



Floating point

- We can explain the meaning of the fields by considering the following example.
- Suppose a byte consists of the bit pattern 01101011. Analyzing this pattern with the preceding format, we see that the sign bit is 0, the exponent is 110, and the mantissa is 1011. To decode the byte, we first extract the mantissa and place a radix point on its left side, obtaining

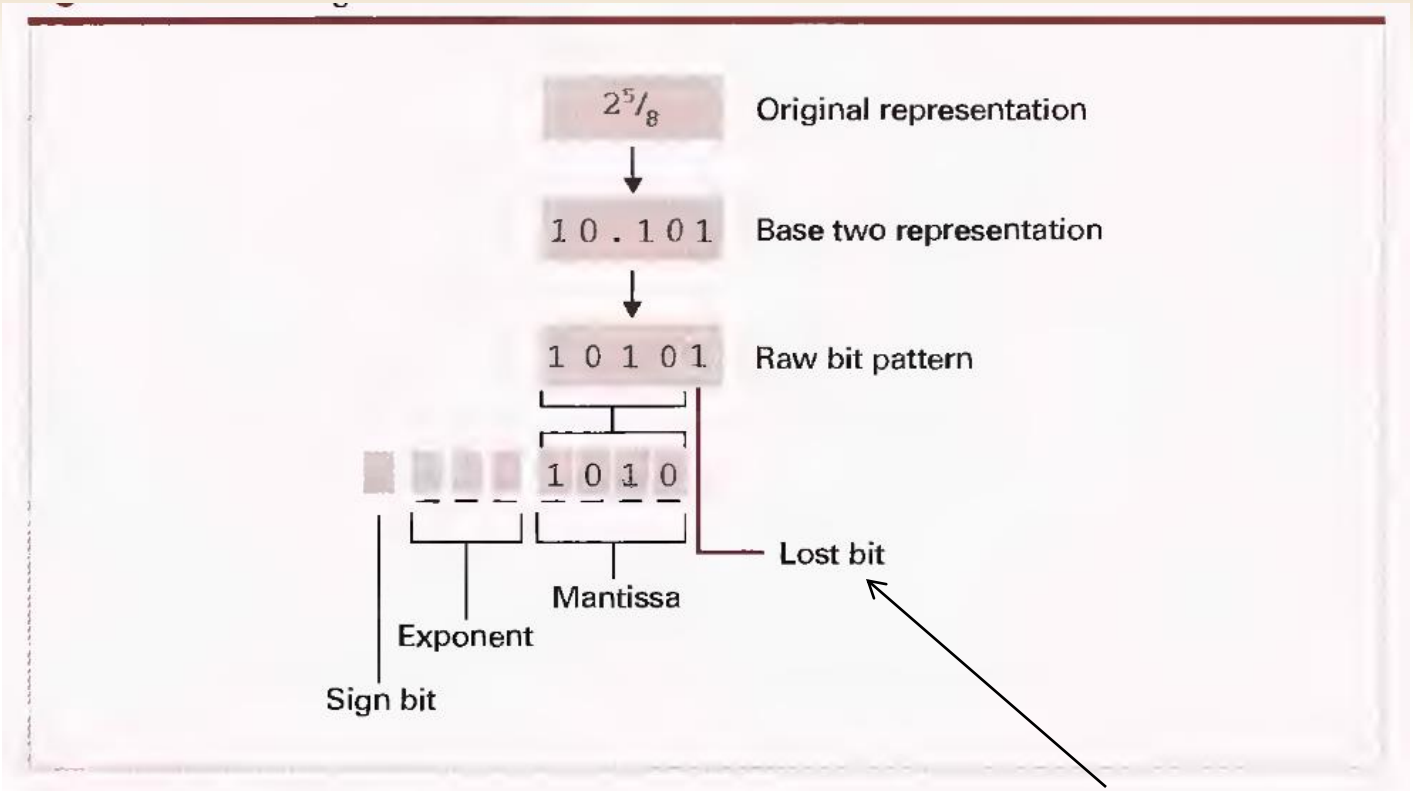
.1011

- Next, we extract the contents of the exponent field (110) and interpret it as an integer stored using the three-bit excess method. Thus the pattern in the exponent field in our example represents a positive 2. This tells us to move the radix in our solution to the right by two bits. (A negative exponent would mean to move the radix to the left.) Consequently, we obtain

10.11

- which is the binary representation for 2 3/4. Next, we note that the sign bit in our example is 0; the value represented is thus nonnegative. We conclude that the byte 01101011 represents 2 3/4.
- Had the pattern been 11101011 (which is the same as before except for the sign bit), the value represented would have been -2 3/4.

Truncation error

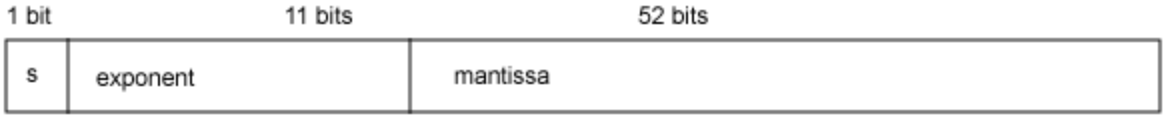


- The significance of such errors can be reduced by using a longer mantissa field.
- In fact, most computers manufactured today use at least 32 bits for storing values in floating-point notation instead of the 8 bits we have used here.

IEEE Floating point representation

IEEE Floating Point Representation

IEEE Double Precision Floating Point Representation



Data compression

- Data compression schemes fall into two categories.
- Some are **lossless**, others are **lossy**.
- **Lossless** schemes are those that do not loose information in the compression process.
- **Lossy** schemes are those that may lead to the loss of information.
  - **Lossy** techniques often provide more compression than lossless ones and are therefore popular in settings in which minor errors can be tolerated, as in the case of images and audio.

Data compression

- Purpose: reduce the data size so that data can be stored and transmitted efficiently.
- For example,
  - 0000000011111111 can be compressed as (0,9,1,8)
  - 123456789 can be compressed as (1,1,9)
  - AABAAAABAAC can be compressed as 11011111011100, where A, B, C are encoded as 1, 01, and 00 respectively.

Run-length encoding

- In cases where the data being compressed consist of **long sequences of the same value,**
  - the compression technique called **run-length encoding**, which is a lossless method, is popular.

- **It is the process of replacing sequences of identical data elements with a code indicating the element that is repeated and the number of times it occurs in the sequence.**
  - For example, less space is required to indicate that a bit pattern consists of 253 ones, followed by 118 zeros, followed by 87 ones than to actually list all 458 bits.

Frequency-dependent encoding

- Another lossless data compression technique is **frequency-dependent encoding**
  - A system in which the length of the bit pattern used to represent a data item is **inversely related to the frequency of the item's use.**
  - Such codes are examples of variable-length codes, meaning that items are represented by patterns of different lengths as opposed to codes such as Unicode, in which all symbols are represented by 16 bits.
  - **David Huffman** is credited with discovering an **algorithm** that is commonly used for **developing frequency-dependent codes**, and it is common practice to refer to codes developed in this manner as **Huffman codes**.
  - In turn, most frequency-dependent codes in use today are Huffman codes.

Huffman Code

- As an example of frequency-dependent encoding, consider the task of encoded English language text.

- In the English language the letters e, t, a, and i are used more frequently than the letters z, q, and x.
  - So, when constructing a code for text in the English language, space can be saved by using **short bit patterns to represent the former letters and longer bit patterns to represent the latter ones.**

- **The result would be a code in which English text would have shorter representations than would be obtained with uniform-length codes.**

Differential Encoding

- In some cases, the stream of data to be compressed consists of units, **each of which differs only slightly from the preceding one**.
- An example would be consecutive frames of a motion picture. In these cases, techniques using relative encoding, also known as **differential encoding**, are helpful.
- These techniques record the **differences between consecutive data units rather than entire units**;
  - that is, each unit is encoded in terms of its relationship to the previous unit.
- Relative encoding can be implemented in either lossless or lossy form depending on whether the differences between consecutive data units are encoded precisely or approximated.

Dictionary encoding

- Still other popular compression systems are based on **dictionary encoding techniques**.
- Here the term dictionary refers to a collection of building blocks from which the message being compressed is constructed, and the message itself is **encoded as a sequence of references to the dictionary**.
- We normally think of dictionary encoding systems as **lossless** systems,
  - but there are times when the entries in the dictionary are only **approximations** of the correct data elements, resulting in a **lossy** compression system.

Lempel-Ziv-Welsh (LZW) encoding

- A variation of dictionary encoding is **adaptive dictionary encoding** (also known as **dynamic dictionary encoding**).
- In an adaptive dictionary encoding system, the dictionary is **allowed to change** during the encoding process.
- A popular example is **Lempel-Ziv-Welsh (LZW)** encoding (named after its creators, Abraham Lempel, Jacob Ziv, and Terry Welsh).
- To encode a message using LZW; one starts with a dictionary containing the basic building blocks from which the message is constructed, but as larger units are found in the message, they are **added** to the dictionary-meaning that future occurrences of those units can be encoded as single, rather than multiple, dictionary references.

Compressing Images

- GIF: Good for cartoons
- JPEG: Good for photographs
- TIFF: Good for image archiving

GIF

- One system known as **GIF** (short for Graphic Interchange Format and pronounced "Giff" by some and "Jiff" by others) is a dictionary encoding system that was developed by CompuServe.
- It approaches the compression problem by reducing the number of colors that can be assigned to a pixel to only 256.
- The red-green-blue combination for each of these colors is encoded using three bytes, and these 256 encodings are stored in a table (a dictionary) called the palette.
- Each pixel in an image can then be represented by a single byte whose value indicates which of the 256 palette entries represents the pixel's color. (Recall that a single byte can contain any one of 256 different bit patterns.)
- Note that GIF is a lossy compression system when applied to arbitrary images because the colors in the palette may not be identical to the colors in the original image.

JPEG

- Another popular compression system for images is **JPEG (pronounced "JAYpeg").**
- It is a standard developed by the **Joint Photographic Experts Group** (hence the standard's name) within ISO.
- JPEG has proved to be an effective standard for compressing color photographs and is widely used in the photography industry, as witnessed by the fact that most digital cameras use JPEG as their default compression technique.

JPEG standard

- The JPEG standard actually encompasses several methods of image compression, each with its own goals.
- In those situations that require the utmost in precision, JPEG provides a lossless mode.
- However, JPEG's lossless mode does not produce high levels of compression when compared to other JPEG options.
- Moreover, other JPEG options have proven very successful, meaning that JPEG's lossless mode is rarely used.
- Instead, the option known as JPEG's baseline standard (also known as JPEG's lossy sequential mode) has become the standard of choice in many applications.

JPEG – Step 1

- Image compression using the JPEG baseline standard requires a sequence of steps, some of which are designed to take advantage of a human eye's limitations.
- In particular, the human eye is more sensitive to changes in brightness than to changes in color.
- So, starting from an image that is encoded in terms of luminance and chrominance components, the first step is to average the chrominance values over two-by-two pixel squares.
- This reduces the size of the chrominance information by a factor of four while preserving all the original brightness information.

- The result of step 1 is a significant degree of compression without a noticeable loss of image quality.

JPEG – Step 2

- The next step is to divide the image into eight-by-eight pixel blocks and to compress the information in each block as a unit.
- This is done by applying a mathematical technique known as the discrete cosine transform, whose details need not concern us here.
- The important point is that this transformation converts the original eight-by-eight block into another block whose entries reflect how the pixels in the original block relate to each other rather than the actual pixel values.
- Within this new block, values below a predetermined threshold are then replaced by zeros, reflecting the fact that the changes represented by these values are too subtle to be detected by the human eye.

JPEG – Step 3

- At this point, more traditional run-length encoding, relative encoding, and variable-length encoding techniques are applied to obtain additional compression.

- All together, JPEG's baseline standard normally compresses color images by a factor of at least 10, and often by as much as 30, without noticeable loss of quality.

TIFF

- Another data compression system associated with images is **TIFF (short** for Tagged Image File Format).

- However, the most popular use of TIFF is not as a means of data compression but instead as a standardized format for storing photographs along with related information such as date, time, and camera settings.

- In this context, the image itself is normally stored as red, green, and blue pixel components without compression.

TIFF

- The TIFF collection of standards does include data compression techniques, most of which are designed for compressing images of text documents in facsimile applications.
- These use variations of run-length encoding to take advantage of the fact that text documents consist of long strings of white pixels.
- The color image compression option included in the TIFF standards is based on techniques similar to those used by GIF, and are therefore not widely used in the photography community.

Compressing Audio and Video

- MPEG
  - High definition television broadcast
  - Video conferencing
- MP3
  - Temporal masking
  - Frequency masking

MPEG

- The most commonly used standards for encoding and compressing audio and video were developed by the **Motion Picture Experts Group (MPEG)** under the leadership of ISO.
- In turn, these standards themselves are called MPEG.
- MPEG encompasses a variety of standards for different applications.
- For example, the demands for high definition television (HDTV) broadcast are distinct from those for video conferencing in which the broadcast signal must find its way over a variety of communication paths that may have limited capabilities.
- And, both of these applications differ from that of storing video in such a manner that sections can be replayed or skipped over.

MPEG

- The techniques employed by MPEG are well beyond the scope of this course, but in general, video compression techniques are based on video being constructed as a sequence of pictures in much the same way that motion pictures are recorded on film.
- To compress such sequences, only some of the pictures, called I-frames, are encoded in their entirety.
- The pictures between the I-frames are encoded using relative encoding techniques.
- That is, rather than encode the entire picture, only its distinctions from the prior image are recorded.
- The I-frames themselves are usually compressed with techniques similar to JPEG.

MP3

- The best known system for compressing audio is **MP3,** which was developed within the MPEG standards.
- In fact, the acronym *MP3 is short for MPEG layer 3.*
- Among other compression techniques, MP3 takes advantage of the properties of the human ear, removing those details that the human ear cannot perceive.
- One such property, called **temporal masking:**
  - is that for a short period after a loud sound, the human ear cannot detect softer sounds that would otherwise be audible.
- Another, called **frequency masking:**
  - is that a sound at one frequency tends to mask softer sounds at nearby frequencies.
- By taking advantage of such characteristics, MP3 can be used to obtain significant compression of audio while maintaining near CD quality sound.

MPEG and MP3

- Using MPEG and MP3 compression techniques, video cameras are able to record as much as an hour's worth of video within 128MB of storage and portable music players can store as many as 400 popular songs in a single GB.

- But, in contrast to the goals of compression in other settings, the goal of compressing audio and video is not necessarily to save storage space.

- Just as important is the goal of obtaining encodings that allow information to be transmitted over today's communication systems fast enough to provide timely presentation.

Speed of communication

- If each video frame required a MB of storage and the frames had to be transmitted over a communication path that could relay only one KB per second, there would be no hope of successful video conferencing.
- Thus, in addition to the quality of reproduction allowed, audio and video compression systems are often judged by the transmission speeds required for timely data communication.
- These speeds are normally measured in **bits per second (bps).**
- Common units include Kbps (kilo-bps, equal to one thousand bps), Mbps (mega-bps, equal to one million bps), and Gbps (gigabps, equal to one billion bps).
- Using MPEG techniques, video presentations can be successfully relayed over communication paths that provide transfer rates of 40 Mbps.
- MP3 recordings generally require transfer rates of no more than 64 Kbps.

Communication Errors

- When information is transferred back and forth among the various parts of a computer, or transmitted from the earth to the moon and back, or, for that matter, merely left in storage, a chance exists that the **bit pattern ultimately retrieved may not be identical to the original one.**

- Particles of dirt or grease on a magnetic recording surface or a malfunctioning circuit may cause data to be **incorrectly recorded or read**.

- Pressure or forces on a transmission path may **corrupt** portions of the data.

- And, in the case of some technologies, normal background radiation can **alter** patterns stored in a machine's main memory.

Communication Errors

- To resolve such problems, a variety of **encoding** techniques have been developed to allow the detection and even the **correction of errors.**

- Today, because these techniques are largely built into the internal components of a computer system, they are not apparent to the personnel using the machine.
  - Nonetheless, their presence is important and represents a significant contribution to scientific research.

Parity bit

- A simple method of detecting errors is based on the principle that if each bit pattern being manipulated has an **odd** number of 1s and if a pattern with an **even** number of 1s is encountered, **an error must have occurred.**

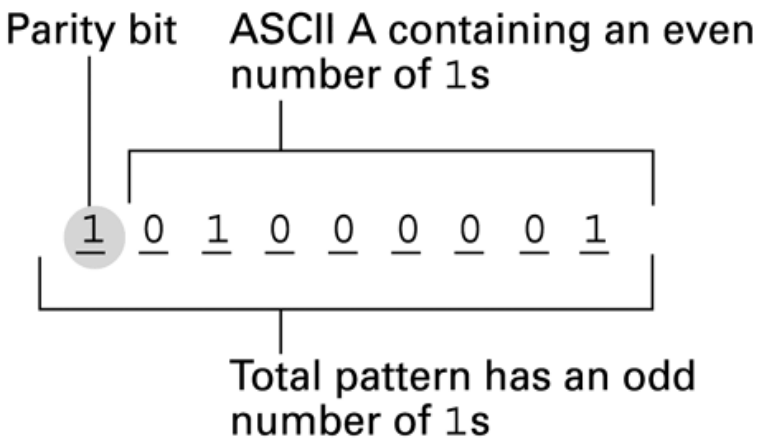
- To use this principle, we need an encoding system in which each pattern contains an **odd** number of 1s. This is easily obtained by first adding an additional bit, called a **parity bit**, to each pattern in an encoding system already available (perhaps at the high-order end).

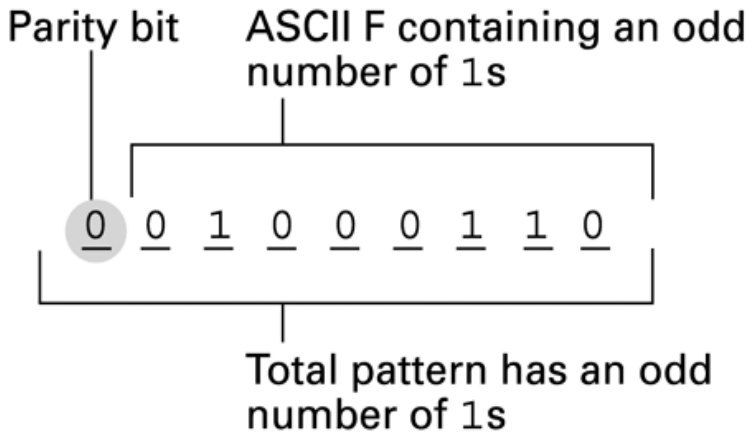
- In each case, we assign the value 1 or 0 to this new bit so that the entire resulting pattern has an odd number of 1s.

- Once our encoding system has been modified in this way, a pattern with an **even number of 1s indicates that an error has occurred** and that the pattern being manipulated is incorrect.

Error detection

- During transmission, error could happen.
  - For example, bit 0 $\rightarrow$1 or bit 1 $\rightarrow$0.
- How could we know there is an error?
  - Adding a parity bit (even versus odd)





Checkbyte

- The straightforward use of parity bits is simple but it has its **limitations.**
- If a pattern originally has an odd number of 1s and suffers **two errors**, it will still have an odd number of 1s, and thus the parity system **will not detect the errors.**
- In fact, straightforward applications of parity bits **fail to detect any even number of errors within a pattern**.

- One means of minimizing this problem is sometimes applied to long bit patterns, such as the string of bits recorded in a sector on a magnetic disk.
- In this case the pattern is accompanied by a **collection of parity bits making up a checkbyte.**
  - **Each bit within the checkbyte is a parity bit associated with a particular collection of bits scattered throughout the pattern.**

Checkbyte

- For instance, one parity bit may be associated with every eighth bit in the pattern starting with the first bit, while another may be associated with every eighth bit starting with the second bit.

- In this manner, a collection of errors concentrated in one area of the original pattern is more likely to be detected, since it will be in the scope of several parity bits.

- Variations of this checkbyte concept lead to **error detection** schemes known as **checksums** and **cyclic redundancy checks (CRC).**

Parity bits in memory

- Today it is not unusual to find **parity bits** being used in a computer's main memory.
- Although we envision these machines as having memory cells of **eight-bit capacity**, in reality each has a capacity of **nine bits**, one bit of which is used as a parity bit.
- Each time an eight-bit pattern is given to the memory circuitry for storage, the circuitry adds a parity bit and **stores the resulting nine-bit pattern.**
- When the pattern is later retrieved, the **circuitry checks the parity of the nine-bit pattern.**
  - If this does not indicate an error, then the memory removes the parity bit and confidently returns the remaining eight-bit pattern.
  - Otherwise, the memory returns the eight data bits with a warning that the pattern being returned may not be the same pattern that was originally entrusted to memory.

Error Correction Codes (Figure 1.29)

- Although the use of a parity bit allows the detection of an error, it does not provide the information **needed to correct the error.**
- **Error-correcting codes** can be designed so that errors can be not only detected but also corrected.
- After all, intuition says that we cannot correct errors in a received message unless we already know the information in the message.
- However, a simple code with such a corrective property is:

Symbol	Code
A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

Hamming distance

- Hamming distance (named after R W Hamming, who pioneered the search for error-correcting codes after becoming frustrated with the lack of reliability of the early relay machines of the1940s) between two patterns is the number of bits in which the patterns differ.

- For example, the Hamming distance between the patterns representing A and B in the code in Figure 1.29 is four, and the Hamming distance between B and C is three.

- The important feature of the code in Figure 1.29 is that any two patterns are separated by Hamming distance of at least three.

Hamming Distance

- If a single bit is modified in a pattern from Figure 1.29, the error can be detected since the result will not be a legal pattern.
  - **We must change at least three bits in any pattern before it will look like another legal pattern**.

- Moreover, we can also figure out what the original pattern was.
  - **After all, the modified pattern will be a Hamming distance of only one from its original form but at least two from any of the other legal patterns.**

Error correction with Hamming distance

- Thus, to decode a message that was originally encoded using Figure 1.29, we simply compare each received pattern with the patterns in the code until we find one that is **within a distance of one from the received pattern.**
- We consider this to be the correct symbol for decoding. For example, **if we received the bit pattern 010100** and compared this pattern to the patterns in the code, we would obtain the table in Figure1.30. Thus, we would conclude that the character transmitted must have been a D because this is the closest match.

Character	Code	Pattern received	Distance between received pattern and code
A	0 0 0 0 0 0	0 **1** 0 **1** 0 0	2
B	0 0 1 1 1 1	0 **1** **0** 1 **0** **0**	4
C	0 1 0 0 1 1	0 1 0 **1** **0** **0**	3
D	0 1 1 1 0 0	0 1 **0** 1 0 0	**1** — Smallest distance
E	1 0 0 1 1 0	**0** **1** 0 1 **0** 0	3
F	1 0 1 0 0 1	**0** **1** **0** **1** 0 **0**	5
G	1 1 0 1 0 1	**0** 1 0 1 0 **0**	2
H	1 1 1 0 1 0	**0** 1 **0** **1** **0** 0	4

End of class

- Readings
  - Book: Chapter 1