Slide Set 1

for ENEL 353 Fall 2019

Steve Norman, PhD, PEng

Electrical & Computer Engineering
Schulich School of Engineering
University of Calgary

Fall Term, 2019

Contents

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

Outline of Slide Set 1

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

About these slides

This is the first of around ten to twelve large sets of slides that will be used for lectures in Section 02 of ENEL 353 in Fall 2019.

It will usually take several lectures to get through a single set of slides. For example, I expect that it will take about 4 lectures to get through this first set.

Reading these slides online is **not** a good substitute for attending lectures—in most lectures I will do some important hand-written work using the document camera. Please come to lectures prepared to take some notes.

Typographical conventions

Either **bold text** or bright red text will be used for emphasis.

Italics will be used two different ways.

One word or a few words in italics will be used to formally or informally define a term.

Example: A *bit* is the basic unit of information in a digital system; the value of a bit is either 0 or 1.

An entire sentence in italics indicates a pause to elaborate a concept or solve a problem under the document camera.

Outline of Slide Set 1

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

Introduction to number systems

What does **702.53** mean?

Seven hundred and two point five three, of course! Why even ask the question?

We've all used numbers in this form so often that it's possible we've forgotten the **underlying rules** for the number system we use every day.

The decimal system

702.53 stands for ...

$$
\begin{aligned}
& 7 \times 10^{2} \\
+\ & 0 \times 10^{1} \\
+\ & 2 \times 10^{0} \\
+\ & 5 \times 10^{-1} \\
+\ & 3 \times 10^{-2}
\end{aligned}
$$

This an example of the *decimal system*, by far the most common system used by humans for representing numbers.

The decimal system

What is the general pattern used for a number in the decimal system?

Ten is called the *base* or the *radix* of the decimal system.

It's probably not a coincidence that most of us have ten fingers and ten is the radix of the number system we use in daily life!

The plural of radix is *radices*.

About the word "decimal"

In technical discussion of number systems, *decimal* means "base ten, with numbers written using digits $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$."

The word *decimal* does NOT mean "has a dot to separate the integer part of a number from the fraction part."

So $28 + 79 = 107$ *is* an example of decimal addition, even though you don't see any "decimal points".

Outline of Slide Set 1

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

Radices other than ten . . .

For any integer $r \geq 2$, you can create a number system with radix r. Such a system would be called a "base r" number system.

You would need symbols for digits ranging from 0 up to $r - 1$.

If $r \leq$ ten, you can just use the digits we already have.

If $r >$ ten, you have to make up new symbols for ten, eleven, twelve, etc.

General format for a base r number

Write it as two lists of digits separated by a dot:

$$d_n d_{n-1} d_{n-2} \cdots d_1 d_0 \,.\, d_{-1} d_{-2} \cdots d_{-p}$$

Each digit d_k must be ≥ 0 and $< r$.

Mathematically, the number is ...

$$\sum_{k=-p}^{n} d_k r^k$$

Textbook notation for indicating a radix

Numbers in the course textbook, course notes, quizzes, tests, etc., will often have a base ten subscript to make it clear what radix is in use.

So 17_{10} is just the number we all know as seventeen. Usually it would be written as 17, with no subscript.

What are 17_8 *and* 17_{16} *in base ten?*

How about 18_8 *and* 19_8*?*

Base two numbers, also called *binary* numbers

Base two is important because it allows design of fast and efficient electronic circuits for arithmetic: addition, subtraction, multiplication, etc.

A binary digit is called a *bit*. So a bit is a thing that can have one of two values: 0 or 1.

Binary number example: What is 1011.01_2 *in base ten?*

Conversion from base r to base ten

Just use the formula:

$$\sum_{k=-p}^{n} d_k r^k$$

We've already used it for 1011.01_2.

Another example: What is 235_{16} *in base ten?*

Conversion from base ten to base r: integers only

This algorithm is important to know in ENEL 353; make sure you get lots of practice with it.

The algorithm is: Using base ten, repeatedly divide by r until you get a **quotient** of zero. The **remainders** you get along the way are the digits of the base r number.

Example: Convert 13_{10} *to binary.*

Example: Convert 87_{10} *to base eight.*

(Note that this method matches the paragraph that starts, "Working from the right . . . ," in Example 1.5 on page 13 of Harris & Harris.)

Conversion from base ten to base r: numbers with fractions

We won't need this algorithm in ENEL 353, but it's nice to know that it exists.

The algorithm is: Using base ten, repeatedly multiply by r. The integer parts of the numbers you get along the way are the digits of the base r number. Fractional parts are used in subsequent multiplications.

Example: Convert 0.6875_{10} *to binary.*

Conversion from base ten to base r: more about numbers with fractions

The algorithm is not guaranteed to terminate in a finite number of steps! For example, try to convert 0.6_{10} to binary ...

multiplication	product	integer part	remark
0.6×2	1.2	1	0.1_2 so far
0.2×2	0.4	0	0.10_2 so far
0.4×2	0.8	0	0.100_2 so far
0.8×2	1.6	1	0.1001_2 so far
0.6×2	1.2	1	Uh-oh ... a cycle!

Conversion from base ten to base r: more about numbers with fractions

The algorithm tells us that
$0.6_{10} = 0.1001\ 1001\ 1001\ 1001\ \cdots_2$.

The problem is NOT that the algorithm is somehow defective.

So what is the real problem here?

Octal and hexadecimal systems

The radix 8 system is called *octal* and so uses digits 0, 1, 2, 3, 4, 5, 6 and 7.

The radix 16 system is called *hexadecimal* or "hex" and uses digits 0, 1, 2, . . . , 8, 9, and, *um, wait a minute, what comes after 9?*

Example: What is $3A9.C_{16}$*?*

Conversion between binary, octal, and hex

Conversion is **easy!** 3 bits make one octal digit; 4 bits make one hex digit.

Example: Express 11010_2 *in octal and hex.*

More examples: Express 153_8 *and* $5D_{16}$ *in binary.*

One more example: Convert 487_{10} *to hex and octal.*

Learn these tables!

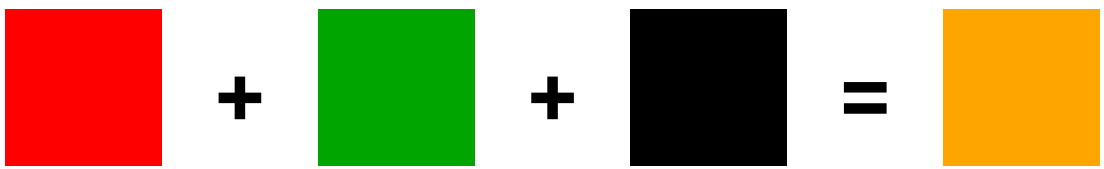
octal digit	bit pattern
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

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

Hex gets used a lot!

It's a convenient way for humans to describe binary data.

Example: HTML color encoding uses two hex digits for brightness of each of red, green, and blue. #FFA500 (FF_{16} means maximum red, $A5_{16}$ means quite a lot of green, and 00_{16} means no blue at all) is a shade of orange ...



Review of conversion between binary and hex and conversion between binary and octal

Hex to binary: Replace each hex digit with the equivalent 4-bit binary pattern.

Binary to hex, step 1: If necessary, add leading zeros so you can make groups of 4 bits.

Binary to hex, step 2: Replace each group of 4 bits with the equivalent hex digit.

For octal, see above, but use 3-bit groups instead of 4-bit groups.

Why does this work? See the next slide ...

Detailed demonstration of conversion from binary to hex: $1010\,0110\,1011_2 = \mathrm{A6B}_{16}$

$$
\begin{aligned}
1010\,0110\,1011_2 \;=\;& \quad 1\times 2^{11} + 0\times 2^{10} + 1\times 2^{9} + 0\times 2^{8} \\
&+ \quad 0\times 2^{7} + 1\times 2^{6} + 1\times 2^{5} + 0\times 2^{4} \\
&+ \quad 1\times 2^{3} + 0\times 2^{2} + 1\times 2^{1} + 1\times 2^{0} \\
\\
=\;& \quad (1\times 2^{3} + 0\times 2^{2} + 1\times 2^{1} + 0\times 2^{0})\times 2^{8} \\
&+ \quad (0\times 2^{3} + 1\times 2^{2} + 1\times 2^{1} + 0\times 2^{0})\times 2^{4} \\
&+ \quad (1\times 2^{3} + 0\times 2^{2} + 1\times 2^{1} + 1\times 2^{0})\times 2^{0} \\
\\
=\;& \quad 10\times 16^{2} \\
&+ \quad 6\times 16^{1} \\
&+ \quad 11\times 16^{0}
\end{aligned}
$$

Outline of Slide Set 1

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

Signed and unsigned number systems

Signed and *unsigned* are adjectives used to describe **number systems**.

A *signed* system has some negative numbers, zero, and some positive numbers.

An *unsigned* system has only zero and positive numbers.

About the words *signed* and *unsigned*

From the previous slide: *Signed* and *unsigned* are adjectives used to describe **number systems**.

They are also useful words for describing **numerical types** in computer programming systems.

They are **NOT** fancy synonyms for "negative" and "positive", and should **NEVER** used to describe individual values.

AVOID saying things like "-42 is signed" and "37 is unsigned". Statements like that just don't make sense!

Outline of Slide Set 1

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

Unsigned addition

Let's add 268_{10} and 437_{10}, so we can quickly review the decimal addition algorithm we learned as school kids:

$$\begin{array}{rr} \textit{carries} & \textit{1}\ \ \textit{1} \\ & 2\ 6\ 8 \\ + & 4\ 3\ 7 \\ \hline & \textit{7}\ \ \textit{0}\ \ \textit{5} \end{array}$$

Note that we have to use rules such as "$8 + 7$ is 5 with a carry of 1," and "$1 + 6 + 3$ is 0 with a carry of 1."

Unsigned binary addition

Addition in base two (or with any other radix) can be done with essentially the same procedure as is used for decimal addition.

Let's demonstrate by adding two four-bit numbers: $0111_2 + 0110_2$.

To do the work, we use rules such as

- $0 + 1 + 0$ is 1 with a carry of 0,
- $0 + 1 + 1$ is 0 with a carry of 1,
- and so on.

Doing one column of unsigned binary addition

$$\begin{array}{rcc} & & C_{\text{in}} \\ & & A \\ + & & B \\ \hline & C_{\text{out}} & S \end{array}$$

Each column has three input bits: C_{in} (carry in), A and B.

Each column has two output bits: S (sum) and C_{out} (carry out).

One column of unsigned binary addition: A table of outputs generated by all possible combinations of input bits

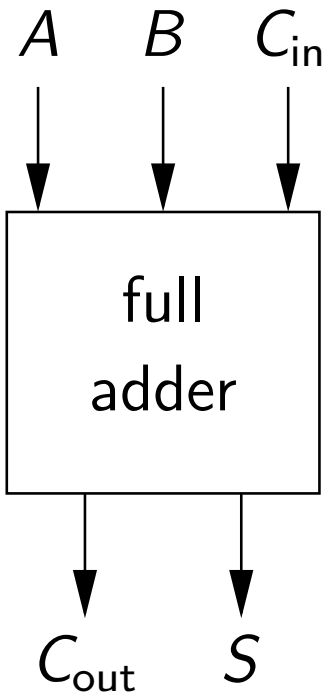
A	B	C_{in}	C_{out}	S
0	0	0	?	?
0	0	1	?	?
0	1	0	?	?
0	1	1	?	?
1	0	0	?	?
1	0	1	?	?
1	1	0	?	?
1	1	1	?	?

Let's replace all the ? symbols with correct bit values.

This kind of table (without any ? symbols) is usually called a *truth table*. We'll read and write a lot of truth tables in ENEL 353.

The full adder: Our first digital logic circuit!

An electronic circuit can be built to match the truth table ...



Inputs and outputs are communicated on **wires**.

For example, voltage near 0.0V could indicate a bit value of 0, and voltage near 3.3V could indicate a bit value of 1.

A four-bit unsigned binary adder

Suppose we have lots of full adder circuits available, and we want to build a circuit to add the binary numbers $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$. Let's call the 4-bit sum $S_3S_2S_1S_0$.

How many full adders do we need, and how should we wire them together?

The *width* of a number system

In everyday work with decimal numbers we tend to use as many or as few digits as we need—we DON'T typically think of numbers as having a fixed number of digits.

But in digital circuits and computer systems, binary numbers usually DO have a fixed number of bits, or *width*.

In the previous example of binary addition, the numbers were all **4 bits wide**.

Outline of Slide Set 1

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

Overflow in fixed-width number systems

Addition (and other kinds of arithmetic) in a fixed-width number system can produce results that DON'T match the "everyday math" result of the computation.

For example, let's add 1011_2 *and* 0110_2 *with a 4-bit adder.* Note that "everyday math" says $11_{10} + 6_{10} = 17_{10}$.

A practical demonstration of overflow

In C, `int` and `unsigned int` types typically have a fixed width of 32 bits ...

```
#include <stdio.h>
int main(void) {
  // Note: u after all the digits indicates that
  // the type of a constant is unsigned.
  unsigned int a = 4294967295u;
  unsigned int b = 2u;
  printf("%u + %u is %u\n", a, b, a + b);
  return 0;
}
```

Output: `4294967295 + 2 is 1`

What happened in the 32-bit addition?

Note that $4294967295_{10} = \mathrm{FFFFFFFF}_{16}$. I deliberately picked a very big number for a to make this example work.

carry in:	1111 1111 1111 1111 1111 1111 1111 1100
a:	1111 1111 1111 1111 1111 1111 1111 1111
b:	0000 0000 0000 0000 0000 0000 0000 0010
sum:	0000 0000 0000 0000 0000 0000 0000 0001

The carry out from the leftmost column is not used—the sum must be exactly 32 bits wide.

Outline of Slide Set 1

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

Signed numbers

Often digital hardware must be able to handle the possibility of **negative numbers**.

How can we use sequences of 1's and 0's to represent numbers that might be negative, might be zero, and might be positive?

The most obvious system is called *sign/magnitude* (sometimes called "sign-and-magnitude").

Another system, not so obvious, **but used much more often**, is called *two's-complement*.

Sign/magnitude numbers

We use sign/magnitude in daily life with decimal numbers, such as $+5$, -37, 44. (If the sign is left out, we assume that it's $+$.)

For binary numbers, we can use 1 to stand for $-$ and 0 to stand for $+$.

Let's look at some examples of binary sign/magnitude representation.

What is the range of an n-bit sign/magnitude system?

Drawbacks of sign/magnitude systems

It's annoying that **zero has two different representations**: $000\cdots00$ and $100\cdots00$. Ideally, two numbers should be equal only if they have identical bit patterns.

Complicated logic is needed to add two numbers that have opposite signs.

What makes a number format good?

In digital systems design it is **important** that arithmetic circuits are **small** and **efficient**.

It is **not important** that numbers be easy for humans to read!

By these criteria, sign/magnitude is **not** a winning format—circuits for the common operations of comparison and addition are more complicated than necessary.

Bit numbering, the MSB, and the LSB

Each bit in an n-bit pattern can be given a number to identify the position of the bit within the pattern. The range of bit numbers runs from 0 up to and including $n - 1$.

Bit $n - 1$, on the far left, is called the MSB (most significant bit).

Bit 0, on the far right, is called the LSB (least significant bit).

Let's draw a diagram.

For the eight-bit pattern 01110101, what are the bit numbers and values of the MSB and LSB? What is the value of bit 4?

Outline of Slide Set 1

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

Two's complement systems for signed integers

Two's complement is **way, way more widely used** than sign/magnitude or any other method for representing signed integers.

Examples:

- With current processor chips, operating systems, and C and C++ compilers, the `int` type is almost certain to be 32-bit two's-complement.
- In Java programming, the `int` type is guaranteed to be 32-bit two's-complement, and the `long` type is guaranteed to be 64-bit two's-complement.

Two's complement systems: Zero and positive numbers

In two's complement, zero and positive numbers are just like sign/magnitude: The MSB is 0, and all the other bits are used for the magnitude.

What are the 5-bit two's-complement representations of 0 *and* 9_{10}*?*

Two's complement systems: Negative numbers

Suppose that a is a positive n-bit number.

Then the bit pattern for the n-bit two's-complement representation of $-a$ is the same as the bit pattern for the **unsigned** number $2^n - a$.

Example: What is the 5-bit two's complement representation of -10_{10}*?*

Let's compare 5-bit two's-complement and sign/magnitude rep's of $+10_{10}$ *and* -10_{10}.

Two's complement systems: Finding the *n*-bit pattern for a negative number

The algorithm is:

1. Get the *n*-bit binary representation for the magnitude of the number.
2. Invert (in other words, *complement*) each bit. Each 0 changes to 1, and each 1 changes to 0.
3. Add 1 to the result using *n*-bit unsigned addition. (That implies ignoring the carry out of the MSB.)

Let's demonstrate for the 5-bit rep of -10_{10}.

What happens if we try to find the 5-bit rep of -0 *("negative zero")?*

6-bit two's complement examples

What are the 6-bit two's-complement representations of -17_{10} *and* 23_{10}*?*

Attention: **A width must be specified.** It does **not** make sense to ask, "What are the two's-complement representations of -17_{10} and 23_{10}?" without any particular width in mind.

Two's-complement negation

Review from two slides back . . .

1. Get the n-bit binary representation for the magnitude of the number.
2. Invert (in other words, *complement*) each bit. Each 0 changes to 1, and each 1 changes to 0.
3. Add 1 to the result using n-bit unsigned addition. (That implies ignoring the carry out of the MSB.)

Steps 2 and 3 together are called *two's-complement negation*.

Let's try two's-complement negation starting with a negative number.

Two good things (among many) about two's complement

(1) There is only one way to represent zero, and it's obvious: all bits are 0.

(2) Perhaps surprising, but true and **extremely useful**: An **unsigned** binary adder circuit will correctly perform two's-complement addition!

Let's demonstrate this with our 6-bit two's-complement bit patterns for -17_{10} *and* 23_{10}.

Why does a circuit for unsigned addition also work for two's-complement addition?

It might seem like we've picked a system more or less at random for representing negative numbers.

Therefore it might seem like a **fluky accident** that an adder designed for unsigned numbers will handle negative numbers correctly!

In fact, straightforward but time-consuming mathematics proves why this is true. Your textbook and lectures both skip the proofs!

Sum of a finite geometric series

A fact from mathematics: For any r,

$$\sum_{i=0}^{k-1} r^i = 1 + r + r^2 + \ldots + r^{k-2} + r^{k-1} = r^k - 1.$$

What does that say about the largest ("most positive") number in an n-bit two's-complement system?

What if we negate the largest number? Does that give us the smallest ("most negative") number in the system?

(Attention: In the interest of saving time, ENEL 353 lectures **skip the proofs** of key facts about how two's complement works.)

Ranges for two's-complement systems

For an n-bit two's-complement system, the minimum value is $-(2^{n-1})$ and the maximum value is $2^{n-1} - 1$. Here are some example ranges for various choices of n:

width n	minimum value	maximum value
4	$-2^3 = -8$	$2^3 - 1 = 7$
8	?	?
16	?	?
32	$-2{,}147{,}483{,}648$	$2{,}147{,}483{,}647$

The last row in the table gives the range of a typical C or C++ `int` type.

Overflow in two's-complement addition (often called "signed overflow")

What is the largest number in a 6-bit two's-complement system?

So, something will go wrong if we try to add $23_{10} + 23_{10}$ in 6-bit two's complement. *Let's see what happens . . .*

Facts about overflow in two's-complement addition

Overflow **cannot** happen when two numbers with **opposite signs** are added.

When two numbers with the **same sign** are added, overflow has occurred if and only if **the sign of the result is obviously wrong**.

Another way to detect overflow, not stated in Harris & Harris: Overflow has occurred if and only if the **carry in** to the MSB column **does not match** the **carry out** from that column.

Unsigned overflow vs. signed overflow in binary addition

Unsigned overflow and signed overflow are NOT THE SAME THING!

With an n-bit adder you can have neither, one but not the other, or both, depending on what the input bit patterns are.

Ways to detect **unsigned** overflow:

- Carry out of MSB column is 1.
- n-bit sum is smaller than either of the n-bit numbers being added.

Unsigned overflow vs. signed overflow in binary addition: 4-bit examples

In addition of 1100 and 1101 is there unsigned overflow? Is there signed overflow?

What about in each of these three additions?

carry:	1 0 0 0 0	1 1 1 1 0	0 1 1 0 0
	1 1 0 0	1 1 1 1	0 1 1 0
+	1 0 1 0	0 0 0 1	0 0 1 1
	0 1 1 0	0 0 0 0	1 0 0 1

Outline of Slide Set 1

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

Codes—BCD and Gray Codes

In this course, a *code* is a *system* for representing numbers or symbols with sequences of 1's and 0's.

Codes we've seen so far in this course: unsigned binary, sign/magnitude, two's complement.

Some other important codes: ASCII code for text, BCD code for base ten numbers, Gray codes.

BCD: Binary Coded Decimal

Each digit in a BCD-encoded decimal number is represented by a 4-bit "word".

For example, in 16-bit BCD, how would 1985_{10} *be encoded?*

decimal digit	BCD code "word"
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

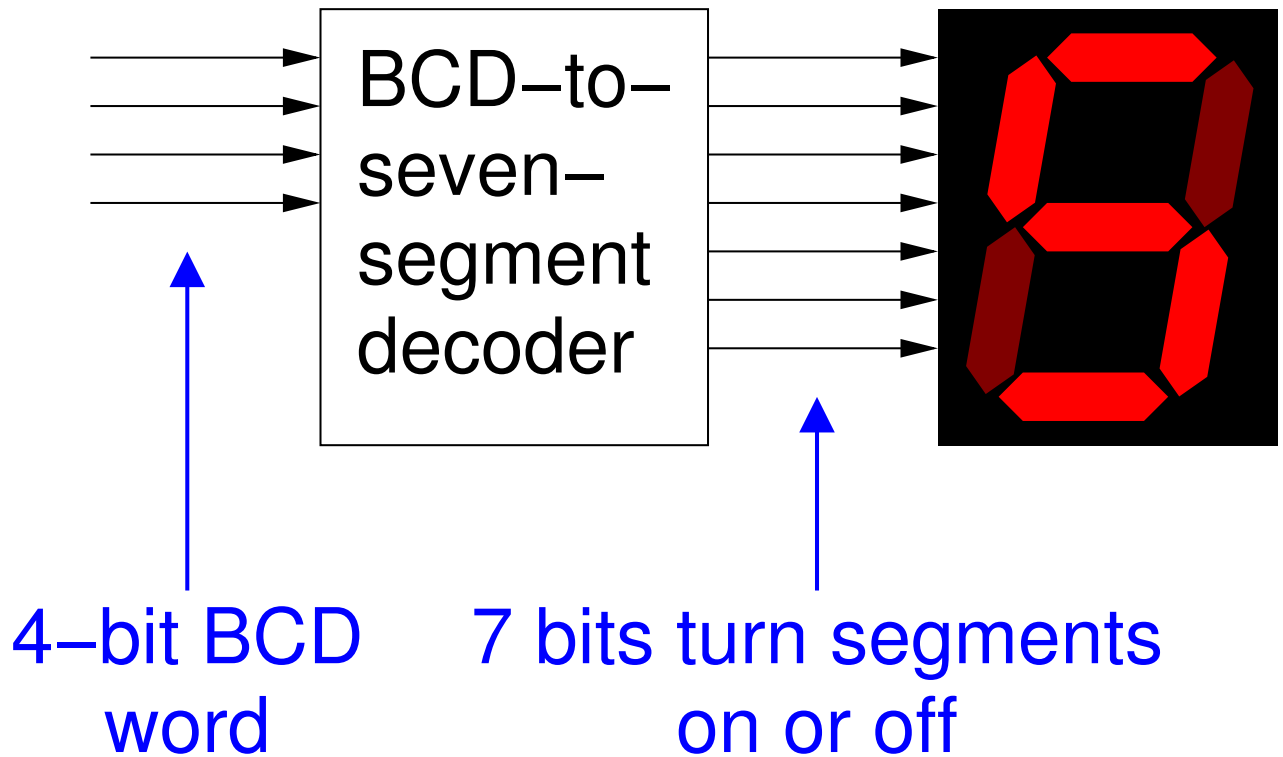
An application of BCD encoding: Simple calculators

Keyboard input is base ten, and so is display output. To keep the design as simple as possible, it makes sense to store numbers and do arithmetic in base ten.

Photo from
`www.oldcalculatormuseum.com/us1.html`

How each digit in the calculator is displayed

Each digit uses a "7-segment display"—a collection of 7 LEDs. (More modern equipment uses LCDs instead of LEDs—LCDs use much less power.)



BCD Arithmetic Algorithms, Circuits and Software

A BCD-based calculator needs logic circuits and microprocessor software to do BCD arithmetic: addition, subtraction, multiplication, division.

Past years' versions of ENEL 353 (2012 and before) spent a little time explaining how to do addition wth BCD-encoded numbers. **Students in ENEL 353 in Fall 2019 MAY be tested on how BCD representation works, but will NOT be tested on BCD addition.**

Gray codes

In an n-bit Gray code, each code word differs from the previous one in only one bit position. Also the first and last codes differ in only one bit position.

That is NOT true for n-bit unsigned binary encoding!

number	3-bit unsigned binary code	3-bit Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Construction of Gray codes

Start with 1-bit Gray code ...

number	1-bit Gray code
0	0
1	1

Duplicate previous Gray code in reverse order ...

number	incomplete 2-bit Gray code
0	0
1	1
2	1
3	0

1st half gets leading 0, 2nd half gets leading 1 ...

number	complete 2-bit Gray code
0	00
1	01
2	11
3	10

Repeat this process until the code has the desired number of bits.

Construction of Gray codes, continued

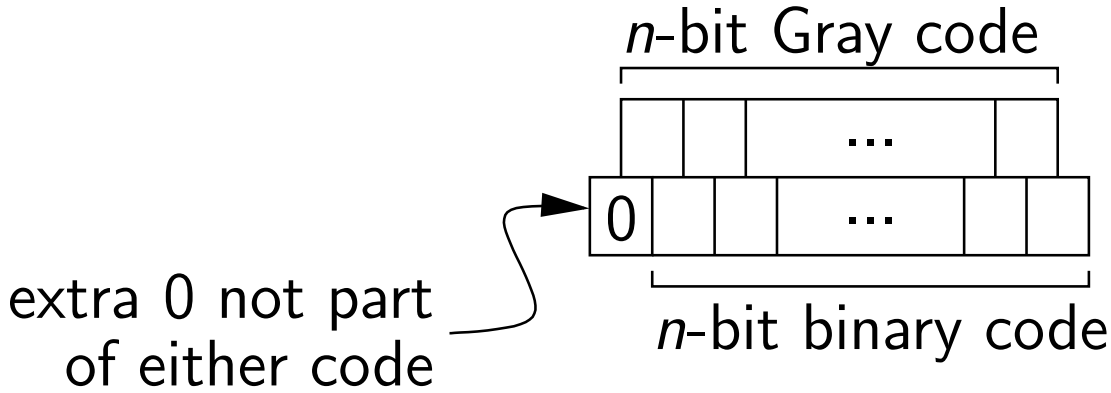
Let's use the construction algorithm to generate the 3-bit Gray code.

Do it yourself: *Starting with the 3-bit Gray code, use the construction algorithm to generate the 4-bit Gray code, and check your work against the table on the next slide.*

4-bit Gray code

number	Gray code	number	Gray code
0	0000	8	1100
1	0001	9	1101
2	0011	10	1111
3	0010	11	1110
4	0110	12	1010
5	0111	13	1011
6	0101	14	1001
7	0100	15	1000

Gray code to unsigned binary code conversion, unsigned binary code to Gray code conversion



The Gray code bit is 0 if the adjacent binary code bits match each other. Otherwise the Gray code bit is 1.

Let's convert 4-bit binary 0110 to Gray code, and 6-bit Gray code 101110 to binary.

Outline of Slide Set 1

About these slides

Introduction to number systems

Radices other than ten ...

Signed and unsigned number systems

Unsigned addition

Overflow in fixed-width number systems

Signed numbers

Two's complement systems for signed integers

Codes—BCD and Gray Codes

Shaft Encoders

Shaft encoders: An application of Gray codes

A *shaft encoder* is a device for reporting the angular position of some rotating piece of equpiment.

A *digital shaft encoder* reports the angle as a **sequence of bits**.

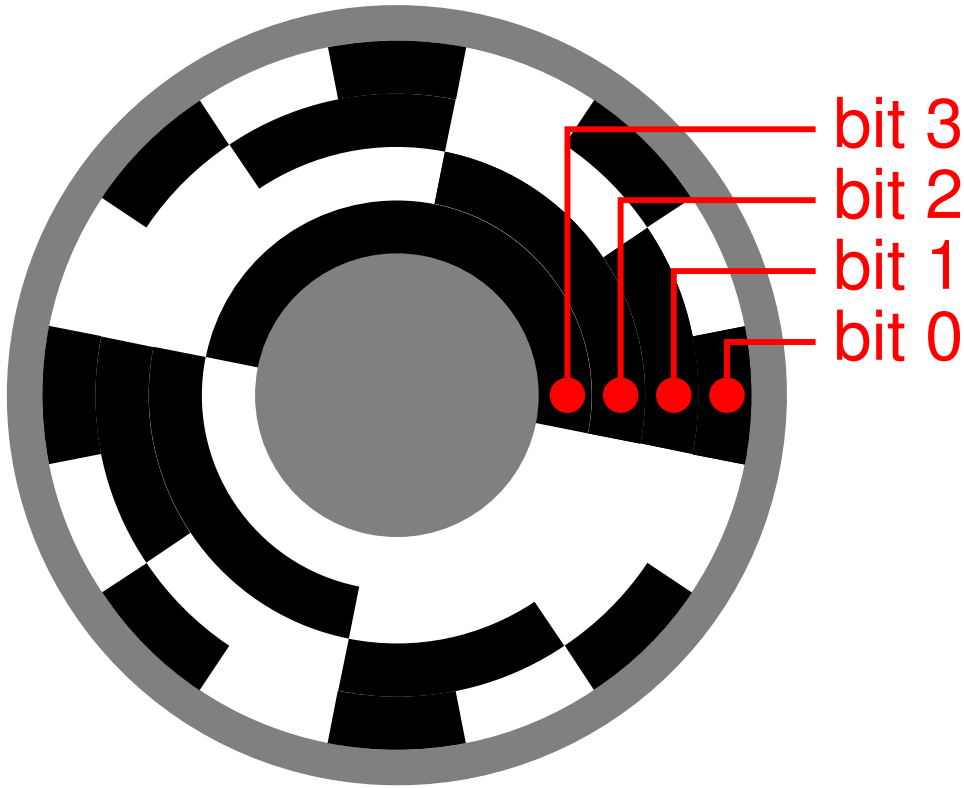
As an example, let's consider the design of a 4-bit shaft encoder. It will measure the rotation of a spinning disk with a precision of one-sixteenth of a revolution.

We'll divide the surface of the disk into sixteen regions that will look somewhat like pie wedges.

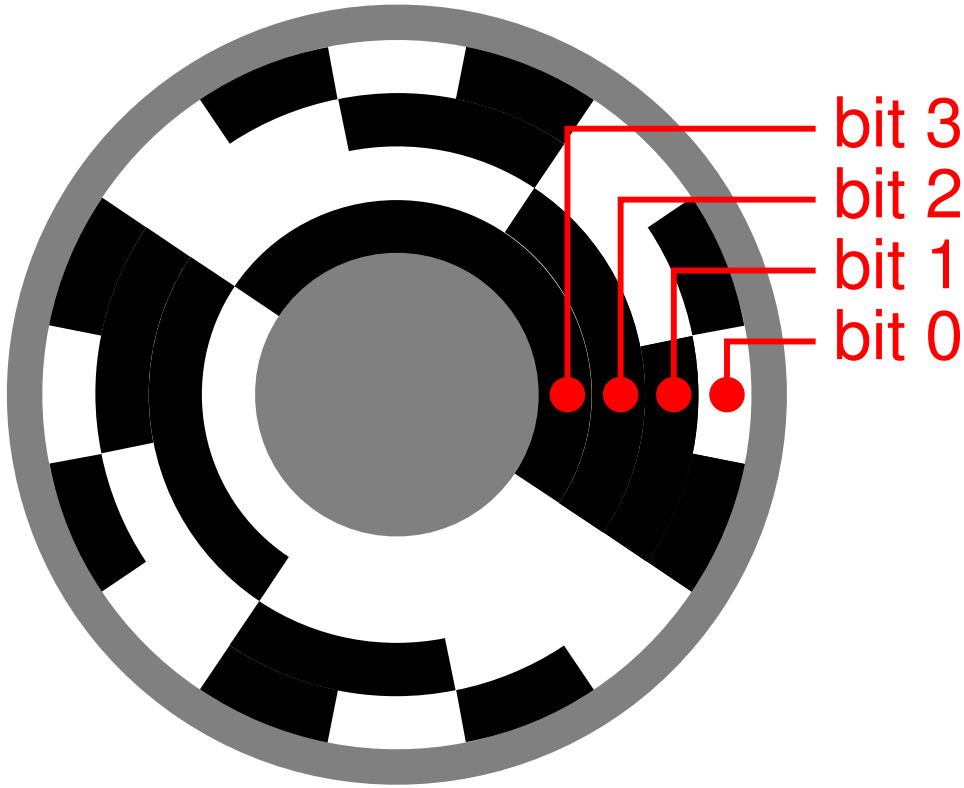
Black and white paint will be used to paint bit patterns on to each wedge. For example, black-black-white-black will represent 0010.

Optical sensors will be used to detect whether the paint under each sensor is black or white.

One possible arrangement is shown on the next slide.

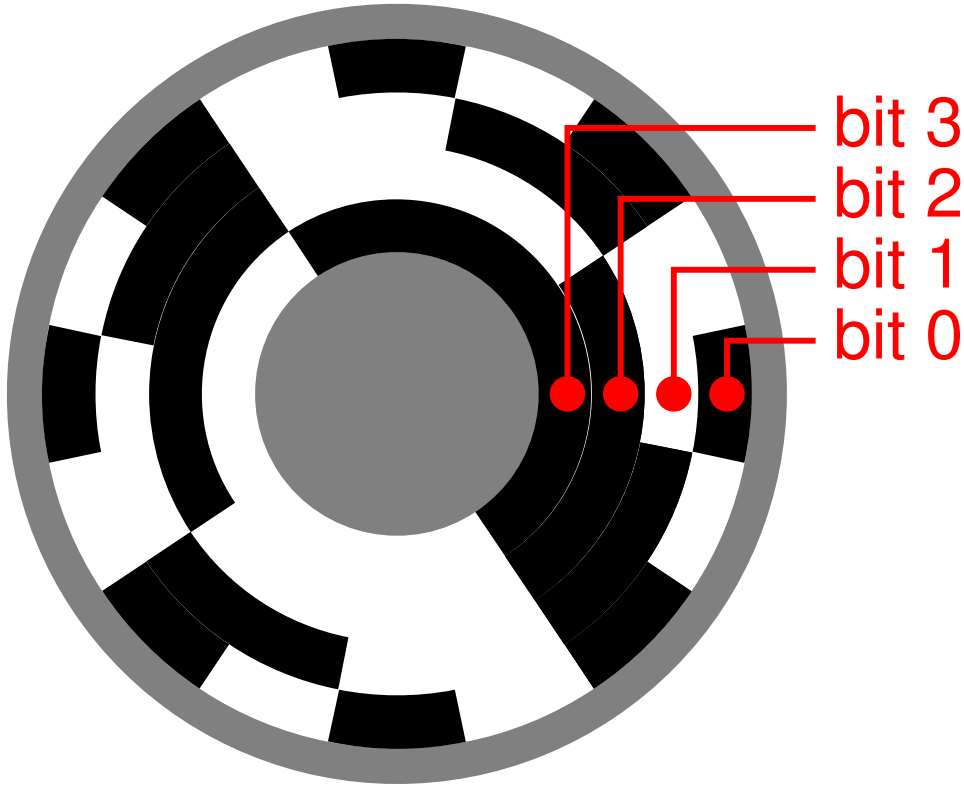


What bit pattern will the sensors report when the disk is in the position shown above?



The disk has rotated a little.

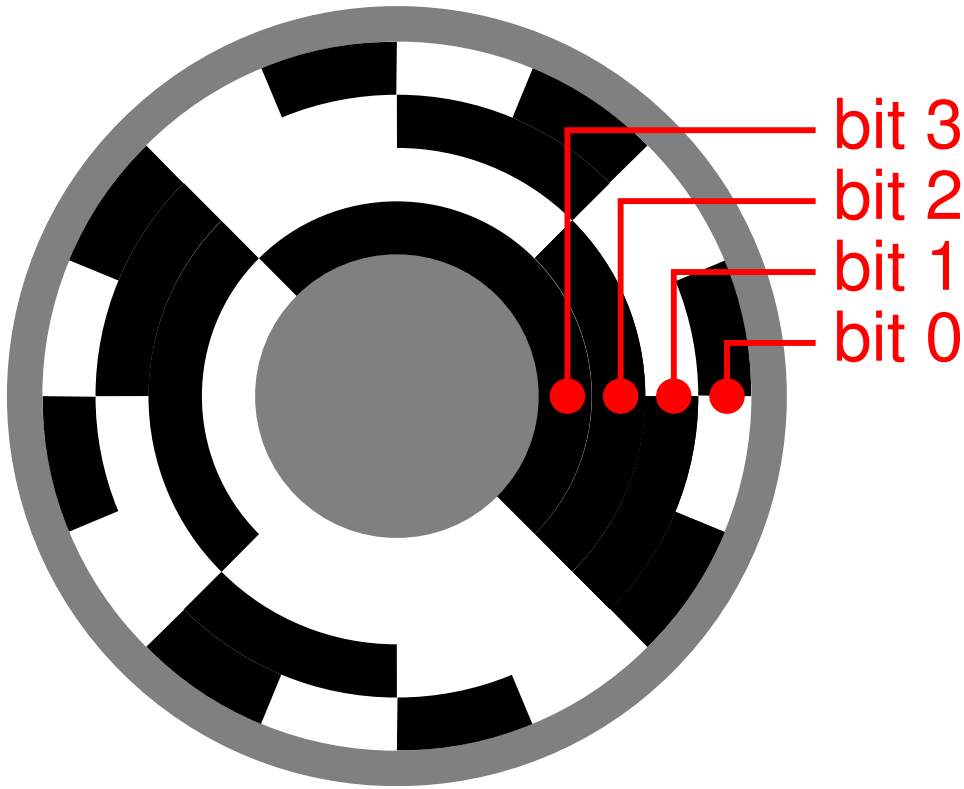
Now what bit pattern will the sensors report?



The disk has rotated a little more.

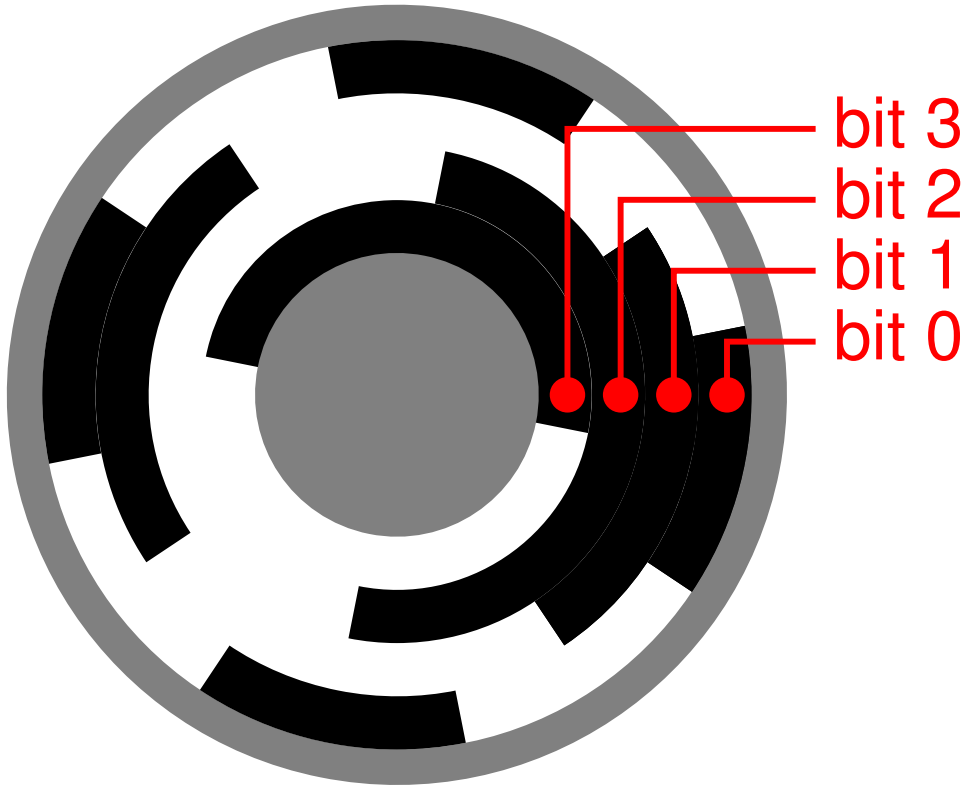
Now what bit pattern will the sensors report? And if the disk continues to rotate clockwise, what is the sequence of bit patterns that will pass under the sensors?

There's a problem with this scheme! The paint won't be perfect, and the sensors won't be perfectly aligned.



What bit patterns could the sensors report when they are over the "wedge boundary" shown above?

Let's change the paint to use a 4-bit **Gray code**.



How does this eliminate the problem presented on the previous slide?

Practical shaft encoders use Gray codes

Measuring to the nearest one-sixteenth of a revolution is not very precise. But it might be useful for a weather instrument indicating one of 16 possible wind directions: N, NNE, NE, ENE, . . . , NNW.

Very precise shaft encoders can measure very small changes in angles in industrial equipment. For example, a 12-bit shaft encoder measures rotation angle to within $360^\circ/2^{12} = 0.088^\circ$.

A web search for "Gray code shaft encoder" will produce some very pretty images!