CPG 5 Coding and Programming

Introduction to Game Programming

Paul Sinnett, London South Bank University <paul.sinnett@gmail.com>

Table of Contents

Week 3

Representing numbers and other things

This week's subject areas are:

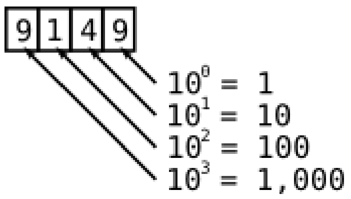
- binary numbers
- hexadecimal
- logical operations with numbers
- encoding text, pictures, and audio

Binary numbers

The binary number system is the natural system for representing numbers in a computer built on Boolean logic. The 0 maps to false and the 1 maps to true.

The decimal system that we use every day has ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. To represent numbers larger than 9, we add a column to the left and digits placed there represent ten times their named value. Each column we add to the left represents ten times the value of the previous column.

Figure 1. A 4 digit decimal number

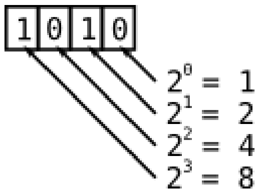


In the number 9,149 the digits represent:

- 9 × 1
- 4 × 10
- 1 × 100
- 9 × 1,000

The binary system is analogous to that, except it has just two digits: 0 and 1. To represent numbers larger that 1, we add a column to the left and digits placed there represent two times their named value. Each column we add to the left represents two times the value of the previous column.

Figure 2. A 4 digit binary number



In the binary number 1010_2 the digits represent:

- 0 × 1
- 1 × 2
- 0 × 4
- 1 × 8

Practicality of binary

Technically we could build a computer that encoded decimal numbers as its natural number system.

Each decimal digit would need to be encoded using a combination of wires. We could do this for each of the digits using the Boolean logic we already know. However, it would take a minimum of 4 wires to represent 1 decimal digit compared to 1 wire to represent 1 binary digit.

With 8 wires in a binary computer we can represent any decimal number from 0 to 255. With 8 wires a decimal based encoding can only manage numbers in the range 0 to 99. The difference quickly accumulates.

But the real problem with using a decimal based encoding for numbers is that it would significantly complicate the circuits required to perform the mathematical operations.

Hexadecimal

Binary is a good fit for building computers, but it has a couple of disadvantages too. Firstly, binary numbers are much longer in digits than the decimal numbers, this can make them hard for programmers to read and interpret.

Two other number systems were commonly used in the early days of computers: base 8 (octal) and base 16 (hexadecimal.)

Base 8 had the advantage that it required no extra digits, we simply dropped the 8 and 9 from decimal. Numbers in base 8 are much shorter than binary numbers and much easier for humans to distinguish. However, base 8 splits the binary numbers up into groups of 3 bits which is an odd number.

By contrast, hexadecimal splits the binary into groups of 4 bits and retains the advantages of being shorter and easier to distinguish. The disadvantage is that we need to add 6 new digits. For digits representing values greater than 9, we use the first 6 letters of the alphabet.

Over the years, hexadecimal has become the dominant system for representing numbers in a way that is friendly to both humans and computers. However, many languages (the C family of languages included) still retain the ability to interpret numbers in base 8.

Unintended use of base 8

To indicate a hexadecimal number in members of the C family of languages, the numbers must be written with `0x` as a prefix. So the hexadecimal representation of the number 16 (10_{16}) must be written as `0x10`.

However, in most of the C family of languages, octal is written with just `0` as the prefix. So the octal representation of the number 8 (10_8) must be written as `010`.

This will cause problems if you try to keep your code neat by zero padding the numbers:

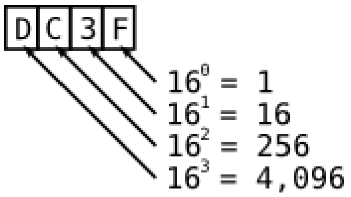
```
int score = 010;
int bonus = 050;
int super = 100;
```

The above code will produce decimal values of 8, 40, and 100 in many C family languages. To pad safely in these languages, use whitespace instead of zero for padding:

```
int score =  10;
int bonus =  50;
int super = 100;
```

The hexadecimal system is also analogous to the decimal system. It has the all the digits 0 to 9 and 6 more: A, B, C, D, E, and F. To represent numbers larger that F, we add a column to the left and digits placed there represent 16 times their named value. Each column we add to the left represents 16 times the value of the previous column.

Figure 3. A 4 digit hexadecimal number



In the binary number $DC3F_{16}$ the digits represent:

- F × 1
- 3 × 16
- C × 256
- D × 4,096

Hexadecimal has the useful attribute that 1 hexadecimal digit represents exactly 4 bits. This means an 8 bit byte is encoded with exactly 2 hexadecimal digits. The following table shows the conversion between hexadecimal, decimal, and binary:

Table 1. Conversion between decimal, hexadecimal, and binary

Decimal	**Hexadecimal**	**Binary**
0	0_{16}	0000_2
1	1_{16}	0001_2
2	2_{16}	0010_2
3	3_{16}	0011_2
4	4_{16}	0100_2
5	5_{16}	0101_2
6	6_{16}	0110_2
7	7_{16}	0111_2
8	8_{16}	1000_2
9	9_{16}	1001_2
10	A_{16}	1010_2
11	B_{16}	1011_2
12	C_{16}	1100_2
13	D_{16}	1101_2
14	E_{16}	1110_2
15	F_{16}	1111_2

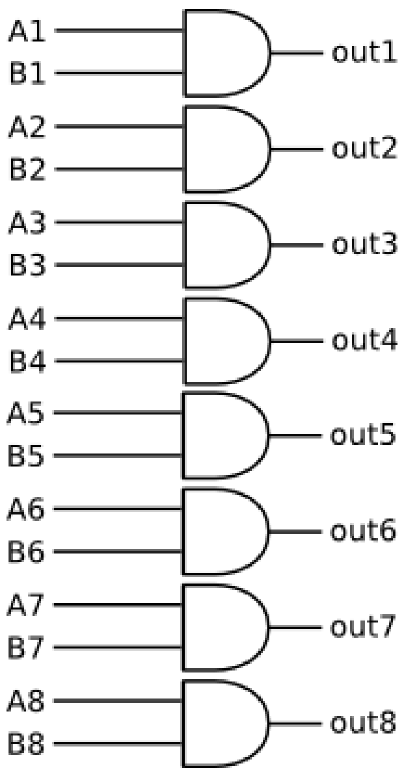
Conversion between decimal, hexadecimal, and binary

The 8 bit byte eventually won out as the standard unit of computer data with the popularity of the 8 bit computer. (An 8 bit computer is one where 8 bits is used in the design of the CPU, its registers, and its addressing.) This doubled to 16, 32, and 64, but we have retained the 8 bit byte as the standard unit of measurement.

Logical operations with numbers

If we represent numbers in binary, we can use all of the Boolean logical operations we looked at last week, by duplicating the gates for each of the bits in the number:

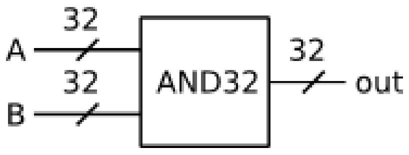
Figure 4. An 8 bit arithmetic AND



Abstraction

Drawing out many duplicate gates is tedious, error-prone, and unnecessarily confusing in a larger circuit diagram. So we often use a short hand for drawing operations that apply the same operation on multiple inputs and outputs.

Figure 5. A 32 bit arithmetic AND



The slash across the input and output lines, simply means that these lines represent multiple lines.

So the arithmetic versions of the 3 Boolean operations work in the same way as the single bit versions, but apply to each bit in the input or inputs. These are also known as bitwise versions of the Boolean operations.

In the C family of languages, the bitwise versions of operations are: ~ (called the tilde) for NOT operation, a single & for the AND, and a single | for the OR.

Parallel processes

Because each output bit is connected only to the corresponding input bit or bits, you can think of this as performing 8 (or 16, 32, or 64) single bit Boolean operations in parallel. So if you arrange your data as a bitmap, you can process your bits in blocks rather than one at a time.

Encoding text, pictures, and audio

By using binary representation we can design circuits that can do things with numbers. But we can also use binary to represent text, pictures, music, and more.

Ada Lovelace

The Countess of Lovelace was the daughter of the poet Lord Byron. She became interested in mathematics and worked on Charles Babbage's analytical engine. She is sometimes credited as the first programmer. This view might not be entirely historically accurate. But (more importantly from a game programming point of view) she did imagine uses for the engine that went beyond the production of numerical tables.

Table 2. ASCII conversion between hexadecimal bytes and text

	0x0	**0x1**	**0x2**	**0x3**	**0x4**	**0x5**	**0x6**	**0x7**	**0x8**	**0x9**	**0xA**	**0xB**	**0xC**	**0xD**	**0xE**	**0xF**
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

ASCII conversion between hexadecimal bytes and text

Example 1. Encoding a simple text string

To encode the string "Hello world!" in hexadecimal using ASCII encoding, you would use the following sequence:

```
48 65 6C 6C 6F 20 77 6F 72 6C 64 21 00
```

The NUL code at the end signals the end of the string. This is the usual method for ending a string in the C family of languages. For this reason they are sometimes called NUL terminated strings or zero terminated strings.

Example 2. Encoding a bitmap

A bitmap is a 2 dimensional arrangement or array of bits. If you consider your 8 bit byte as a horizontal line of 8 bits you can encode an 8 by 8 pixel bitmap with 8 bytes of 2 digit hexadecimal code.

Here is the hexadecimal code for a single capital letter A:

```
7C C6 C6 FE C6 C6 C6 00
```

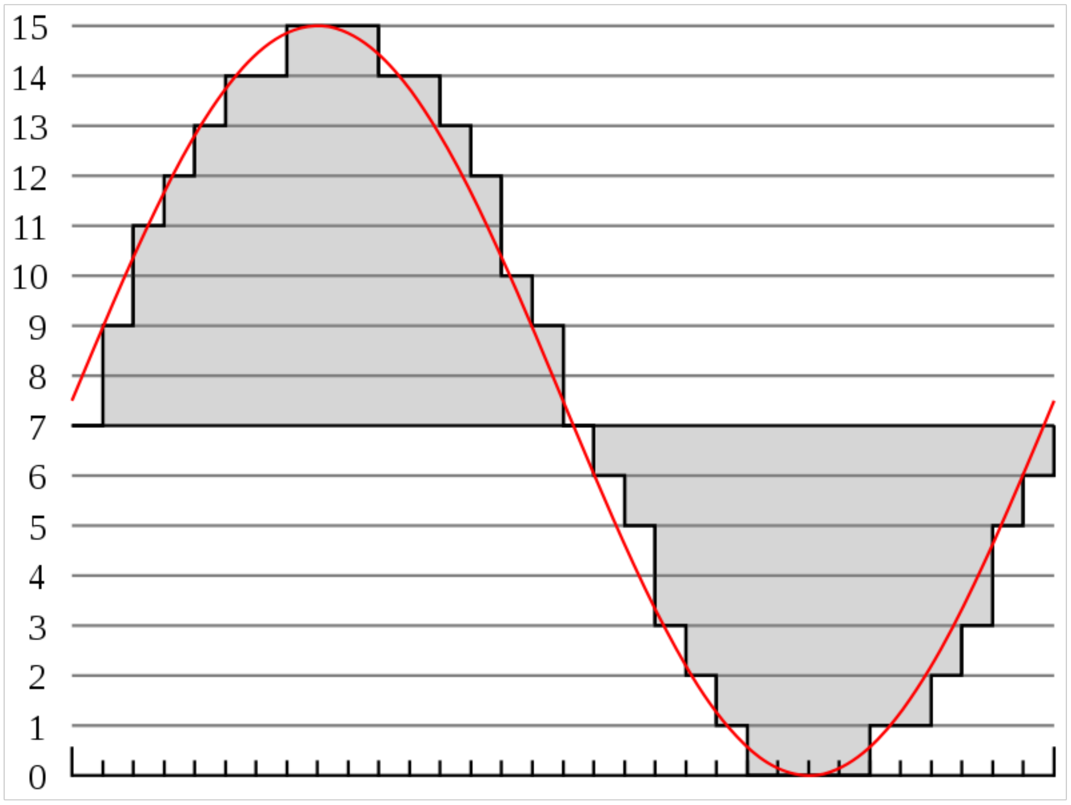
By combining the representation of text coded in ASCII and a table of 8 by 8 pixel bitmaps, a computer can represent our language visually allowing it to print or display letters on a screen.

Bitmap formats have evolved over time. The currently dominant encodings for bitmaps (more accurately now called pixel maps) are 24 bit or 32 bit "true colour" formats. These formats use 8 bits to represent the levels of red, green, and blue for each pixel. In 32 bit formats, an extra channel of data is also included known as the alpha channel. This extra per pixel data can be used for other purposes, but it is most commonly used to provide a mask. Masks are useful when composing images in layers.

Although we no longer commonly use purely textual displays on computers, the visual displays of almost all modern devices are still encoded as pixel maps and the text in simple text files is usually encoded in an ASCII compatible representation.

It is also possible to encode audio waveforms into binary streams. The most common form of audio encoding tests the amplitude of the audio wave at some fixed high frequency and stores the result in some number of bits. Here is an approximation of an audio sine wave using 4 bits to encode the amplitude:

Example 3. A 4 bit audio sample



With all of these encodings, the format of the encoding must be agreed upon somehow. The is usually achieved with a header block of some kind. This is a small section of data that identifies the encoding scheme of the data bytes that follow.

Exercise

For this week's exercise I'd like you to encode some data in hexadecimal representation. I'd like you to attempt to encode a text string in ASCII for some of the messages you'd like to see in your game. Use the zero terminated encoding system of the C family of languages, as in the "Hello world!" example above.

I'd also like you to encode a bitmap that you could use to display either a sprite or some of the lettering in your game. Since 1 bit per pixel bitmaps are no longer commonly used, I've created the following ASCII encoding for this bitmap:

Example 4. An 8 x 8 bitmap encoding for the letter A

```
BITMAP 8 8
7C C6 C6 FE C6 C6 C6 00
```

To create a bitmap file in this encoding scheme, you need to create a plain text file. You can use notepad for this. The file must begin with the text "BITMAP" followed by two space separated decimal numbers representing the width and height of the bitmap in pixels. After this come the space separated hexadecimal byte codes. The width in pixels you specify for your bitmap defines how many bytes you need to encode each line of the image. If you don't use a multiple of 8 for the width, round up the number of bytes required. Additional pixels in these bytes are ignored.

I will upload a viewer program for these files to source control. To view a bitmap with this viewer in Windows, drag and drop the bitmap file onto the View.exe program. If you have any problems creating these files or running the viewer we'll go through this in the class.

You must submit plain text files containing your ASCII encoded strings and bitmaps to source control. Don't forget to keep track of your time and also submit your log files. The deadline for submission is Sunday.