COMPUTER LOGIC AND DIGITAL CIRCUIT DESIGN (PHYS306/COSC330): UNIT 1.3

Jordan Hanson

February 27, 2018

Whittier College Department of Physics and Astronomy

SUMMARY

Reading: Digital Fundamentals (DF) Ch. 2 (see Moodle)

1. Number representation
2. Binary conversions
3. Binary arithmetic
4. The floating-point system
5. Hexadecimals, Binary-Coded Decimals (BCD), Gray codes, and ASCII

Homework: exercises 5-16, 19-32, 35-38, 45, 53, 58 Ch. 2 (DF) (two weeks).

NUMBER REPRESENTATION

Questions:

- A simple question: how many students do we have in this class?
- Una simple pregunta: ¿Cuántos estudiantes tenemos en esta clase?
- Une question simple: Combien d'étudiants est-ce que nous avons dans cette classe?

What languages do computers speak? How can we store and transmit numbers through circuits? (We cannot use voltage magnitudes).

Consider the number 37

Numbers	Expanded notation
0d37	$3 \times 10^1 + 7 \times 10^0$
0b100101	$1 \times 2^5 + 1 \times 2^2 + 1 \times 2^0$
0x25	$2 \times 16^1 + 5 \times 16^0$

Consider the number 412

Numbers	Expanded notation
0d412	$4 \times 10^2 + 1 \times 10^1 + 2 \times 10^0$
0b110011100	$1 \times 2^8 + 1 \times 2^7 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2$
0x100	1×16^2

Number representations - Digits, weights, and a common base

- A number is written with **digits** that have **weights**.
- A **weight** is a power of a **base**.
- At right, the **base** is ten, and the **weights** are 10^2, 10^1, and 10^0.
- The **digits** are four, one, and two.
- The digits cannot represent numbers larger than the base.

Expanded notation

- $412 =$ $4 \times 10^2 + 1 \times 10^1 + 2 \times 10^0$

Number representations - (Aside) please use scientific notation, and here is why: digit minimization!

- Scientific notation **factors the largest weight**.
- Results in weights that are less than one.
- Weights that are less than one go to the right of the **decimal point**.
- Return here with *floating-point* representation.

Expanded notation

- $412,000,000 = 4 \times 10^8 + 1 \times 10^7 + 2 \times 10^6 + 0 \times 10^5 + 0 \times 10^4 + 0 \times 10^3 + 0 \times 10^2 + 0 \times 10^1 + 0 \times 10^0$
- $412,000,000 = 4.12 \times 10^8 = (4 \times 10^0 + 1 \times 10^{-1} + 2 \times 10^{-2}) \times 10^8$

Number representations - (Aside) please use scientific notation, and here is why: arithmetic operations with large numbers!

1. $4200 \times 4200 = (4.2 \times 10^3)^2 = (4.2)^2 \times 10^6 \approx 16 \times 10^6$
2. $\approx 17.6 \times 10^6$ if you account for the 0.2 ...

---

1. $4000/3000 = 4 \times 10^3 \times \frac{1}{3} \times 10^{-3} = \frac{4}{3}$
2. $\frac{4}{3} \approx 1.33$

Expand the following numbers to expanded decimal notation:

- -10.432
- 800,000,144

Expand the following numbers to expanded binary notation:

- 10011010
- 11110000

Convert the following decimal numbers to binary notation:

- 260
- 560

Volunteer to board? - Key is explaining how you did the binary conversions

BINARY CONVERSION

How did you do the conversions to binary? Is there a systematic what to do this?

- *Successive Approximation* - like a number puzzle (*Sum of Weights Method*)
- *Successive Division Method* - Example of an algorithm

Successive approximation technique (does this remind you of doing division in your head?) 260...$2^8 = 256$. Now we need four more...$4 = 2^2$. So $2^8 + 2^2 = 0b10000100$

What is 328 divided by 3? Ok try 100 because three times one hundred is close...

Successive Division Method

$0d412 = 0b110011100 = 2^8 + 2^7 + 2^4 + 2^3 + 2^2$

Algorithm:

1. Divide the decimal number by 2, and write down the remainder. This is the *least-significant bit* or LSB.
2. Keep dividing and recording the remainders in order, until you reach a dividend of 1.
3. $1/2 = 0r1$, so the *most-significant bit*, or MSB, is always 1.

Convert 412 to binary using the successive division method.

Convert the numbers at right to binary.

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

1. 93
2. 189
3. 270

Note: how many bits do you need for that last one? For binary, show that the highest representable number with n bits is $2^n - 1$.

Notice that we get a *repeating pattern* when we compare the conversion of 189 and 93. Why?

In decimal notation, we represent numbers $\in [0-1]$ with digits to the right of the *decimal point*.

In expanded notation:
$42.42 = 4 \times 10^1 + 2 \times 10^0 \ \ . \ \ 4 \times 10^{-1} + 2 \times 10^{-2}$

We have a similar notation in other number systems:
$101.11 = 2^2 + 2^0 \ \ . \ \ 2^{-1} + 2^{-2}$

Successive Multiplication Method

$0d0.48 = 0b0.011110101 = 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-9}$

Algorithm:

1. Multiply the decimal fraction by 2, and write down the *carry* (the 1 to the left of the decimal point, else 0). This is the *most-significant bit* or MSB.
2. Keep multiplying and recording the carries in order, to the desired precision.
3. Watch for repeating patterns.

Convert 0.48 to binary using the successive multiplication method until you identify a repeating pattern.

Convert the numbers at right to binary.

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

1. 0.64
2. 0.125

Which result should have a smaller number of digits, and why?

BINARY ARITHMETIC

Binary arithmetic, like decimal arithmetic, relies upon *carries* and *borrows*. For example in decimal:

- $8 + 7 = 5c1 = 15$
- $3 + 4 = 7c0 = 7$
- $8 - 2 = 6b0 = 6$
- $10 - 5 = (0b10) - 5 = 5$
- $13 - 7 = (0b10) - 4 = 6$

In binary there are limited combinations, so we may form a set of addition and subtraction rules that describe all possibilities:

- $0 + 0 = 0$
- $1 + 0 = 1$
- $0 + 1 = 1$
- $1 + 1 = 0c1$

- $0 - 0 = 0$
- $1 - 0 = 1$
- $0 - 1 = 0b1$
- $1 - 1 = 0$

In the fourth rule for the addition set, the carry works the same way as a decimal carry: we add that 1 to the digit corresponding to the next power of 2. Similarly, in the third rule of the subtraction set, we subtract the 1 on the left side of the equation from a 1 from the digit corresponding to the next highest power of 2 available.

Complete the following additions and subtractions:

- $10 + 11$
- $111 + 1$
- $111 + 111$
- $1010 + 101$
- $10 - 11$
- $111 - 1$
- $111 - 111$
- $1010 - 101$

Binary multiplication stems from another set of rules:

- $0 \times 0 = 0$
- $1 \times 0 = 0$
- $0 \times 1 = 0$
- $1 \times 1 = 1$

Multiply:

1. 10×10
2. 110×10

Binary division is exactly the same as decimal long division. Let's work these examples on the board:

- 145/12
- 1101/101
- 45/3
- 111/10

Of what logic operation does binary multiplication remind you?

We need a few tools to improve our arithmetic techniques in order to install these functions into circuits. Consider two actions:

- NOT to the bit sequence of a number: **1's compliment**
- Add 1 to the 1's compliment: **2's compliment**

Logical operator representation of these actions for 8-bits:

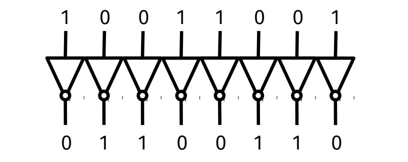

Figure 1: Representation of 1's compliment.

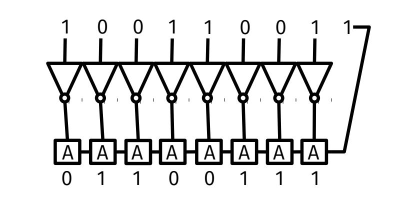

Figure 2: Representation of 2's compliment.

Exercises, and a realization for the purpose of 2's compliment.

Take the 1's compliment of the right-hand number, and complete the addition:

- $111 + (000)$
- $111 + (111)$

Take the 2's compliment of the right-hand number, and complete the addition:

- $111 + (000)$
- $111 + (111)$

Same, but drop the MSB:

- $1010 + (1010)$
- $1100 + (1100)$

For what purpose are we going to use the 2's compliment?

The answer: representing negative numbers. For *2's compliment form signed binary numbers, the MSB is the* ***sign bit***. To convert to decimal, give the MSB a negative weight:

Example: 10101010
This is a negative number because the MSB is a 1. Converting in expanded form: $-2^7 + 2^5 + 2^3 + 2^1 = -128 + 32 + 8 + 2 = -86$.

Consider the effect on the maximum *range* of binary numbers after sacrificing the MSB to flag numbers less than zero. For 8-bits unsigned numbers, the range is [0-255]:

n-bits	8	16
Lowest unsigned	0	0
Highest unsigned	255	65535
Lowest signed	-128	-32768
Highest signed	127	32767

Table 1: The ranges of unsigned and signed numbers using *n* digits.

By changing the role of the MSB, we are not reducing the *range* of numbers, just its *location*. Related to *conservation of information*...

2’s compliment, signed number representation and conversion, arithmetic with negative numbers

---

1. Write -278 in binary, 2’s compliment form.
2. Convert -302 and 65 each to binary, and add them.
3. This number is in 2’s compliment form: 11001101. Convert it to decimal.

THE FLOATING-POINT SYSTEM

Who has taken software programming?[1]
In software code, we allocate memory for variables which could be decimal numbers. In C++, this might look like

float *var* = 15.237;

double *var* = 1.61*E* − 19;

unsigned int *var* = 8;

long int *var* = 3200000000;

We will now learn the origin and purpose of these keywords.

[1] Normally, this is a pre-requisite for this course, but in this innaugural year we are more flexible.

A *floating-point number* is a general system for storage of wide ranges of real numbers in binary. The structure of a floating-point number:

N	Binary	Scientific Notation	Sign bit	Exponent	Mantissa
-10.23	-1010.0011101	-1.0100011101×2^3	1	10000010	01000111010000000000000
22.3	10110.01001100	$1.011001001100 \times 2^4$	0	10000011	01100100110000000000000

Table 2: For **float** or **single** precision, the number of bits allowed for the sign, exponent and mantissa is **32**. The exponent is $3 + 127$ in binary. We *bias* the exponent by 127 so that the range of exponents can be both negative and positive.

- Float or single precision: 32 bits
- Double precision: 64 bits

The structure of a floating-point number:

N	Binary	Scientific Notation	Sign bit	Exponent	Mantissa
22.3	10110.01001100	$1.011001001100 \times 2^4$	0	10000011	01100100110000000000000
6.825	1010.11010011	1.01011010011×2^3	0	10000010	01011010011000000000000

Table 3: Note that the MSB in scientific notation is **dropped** in the mantissa. Why?

Convert the following numbers into **floats**:

- 8.125
- -4.0625

Addition and subtraction with signed numbers: either number can be positive/negative, and the larger/smaller number by magnitude. Thus, 4 combinations are possible:

...	Negative	Positive
Larger	A	B
Smaller	C	D

Table 4: Four categories, two numbers, so 6 initial pairs: AB AC AD BC BD CD. However, AB and CD aren't real choices, so the list is: AC AD BC BD.

Bottom line: both negative, both positive, one negative one positive (two cases)

The results:

1. Both negative $\rightarrow$ discard the carry
2. One positive, one negative, and the negative number has a larger magnitude
3. One positive, one negative, and the positive number has a larger magnitude $\rightarrow$ discard the carry
4. Both positive

Example of negative-negative case: $-13 - 9 = -22$.

1. Identify case: both are negative, with magnitudes of 13 and 9.
2. Decide n-bits. Ensure that $2^n - 1$ is much larger than other numbers. Let's choose $n = 8$.
3. Convert the first magnitude (13) to 2's compliment form: $2s(00001101) = 11110011$
4. Convert the second magnitude (9) to 2's compliment form: $2s(00001001) = 11110111$
5. Add them, and in this case, drop the final carry: 11101010
6. Check: $2s(11101010) = 00010110 = 22$

HEXADECIMALS, BCD, GRAY-CODES, AND ASCII

Why do we use Hexadecimals?

`https://youtu.be/DqfnMzDhi38`

In case you get your butt stranded on Mars, and need to devise a rag-tag communication system from a camera-pole, is why.

Astronaut Watney needs a way to cram more information into shorter-length words...hexadecimals

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	1111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Table 5: Decimal, binary and Hexadecimal representations of numbers. What do you notice that is different between the representations?

Because $2^4 = 16$, we may group bits in bit sequences in groups of four, and read out the hexadecimal conversion:

$0b11010010 = 1101\ 0010 = (13)\ (2) = 0xD2$

On page 71 of DF: “It should be clear that it is much easier to deal with a hexadecimal number than with the equivalent binary number. Since conversion is so easy, the hexadecimal system is widely used for representing binary numbers in programming, printouts, and displays.”

It is probably important. What follows is a few examples of how I’ve encountered hexadecimals.

HEXADECIMALS, BCD, GRAY-CODES, AND ASCII

```
set key at 500,73 box on samplen 2 spacing 1.5 font ”Courier,16” width 2
set xrange [1:1e3]
set yrange [0:180]
set format x ”10^{%T}”
set rmargin 8
set lmargin 13
set tmargin 2
set bmargin 6
set xlabel ”Frequency [MHz]” font ”Courier,28” offset 0,-2,0
set ylabel ”Phase (deg)” font ”Courier,28” offset -5,0,0
set ytics scale 2 font ”Courier,28”
set xtics scale 2 font ”Courier,28” offset 0,-1,0
set style line 6 linecolor rgb ”#000000” linewidth 5.000 dashtype 3 pointtype 2
set style line 7 linecolor rgb ”#BBBBBB” linewidth 5.000 dashtype 3 pointtype 2
set linestyle 1 lc rgb ”#000000” lw 5
set linestyle 2 lc rgb ”#444444” lw 5
set linestyle 3 lc rgb ”#888888” lw 5
set linestyle 4 lc rgb ”#BBBBBB” lw 5
set linestyle 5 lc rgb ”#DDDDDD” lw 5
set terminal postscript color enhance
```

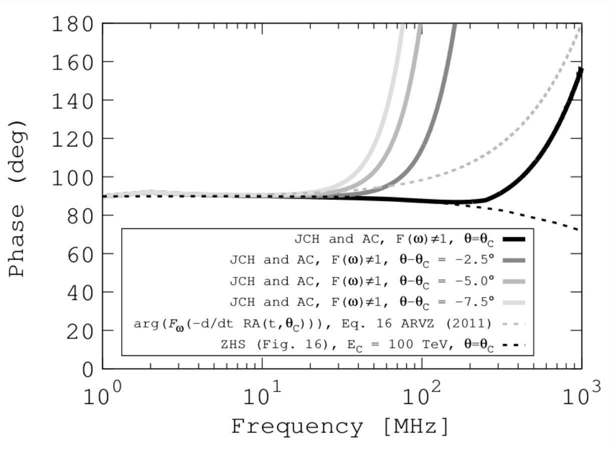

Figure 3: A figure from a recent paper we published about neutrino signals in ice. The hexadecimals were used to encode the line-colors.

A math problem: If the RGB (red, green, blue) codes have the following format 0xA1B2C3, where each hex word (two characters) represents an amount of red, green, and blue (sequentially), how many distinct shades of each are technically possibe? How many colors are therefore possible (even if a standard monitor could not resolve them)?

Volunteer to board?

In graduate school, I was responsible for designing and deploying a housekeeping integrate circuit (HK board). The task of this device was to control which subsystems are activated in an ARIANNA station, managing power consumption.

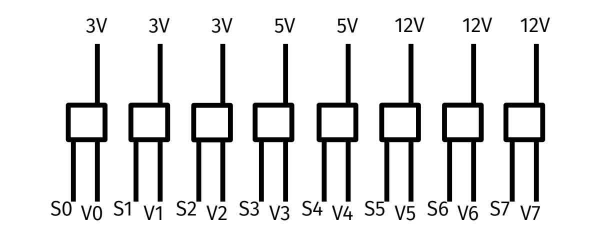

Figure 4: Symbolic representation of a relay bank. *Raising* the various signals S_i to a HIGH state connects the sub-system voltages V_i to power sources at the given voltages, and setting them to LOW state powers down the subsystems.

In software, the C-code interpreted a *bit-mask* in hexadecimal, which the single-board computer (SBC) would send to the digital IO (input/output), connected to the S_i lines. For example, 0*b*11100000 activated only the 3V subsystems.

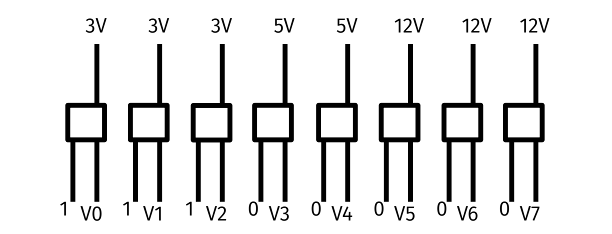

Figure 5: The S_i lines were controlled by the system digital IO. The above bit-mask was sent from the C-code through the digital IO.

However, to save space on a hard drive that only had 4 GB of space, with potentially thousands of defined functions that had to send bit masks, the manufacturer designed the base bitmasking function input as a hex word (two hex characters). In this case, it would have been $0xE0 = 0b11100000$.

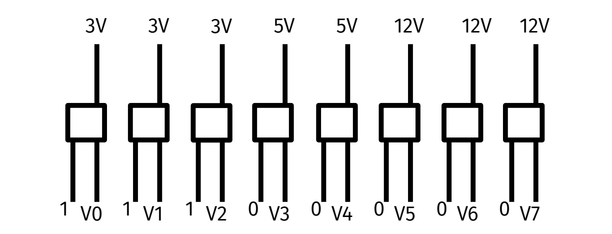

Figure 6: A bitmask of $D0$.

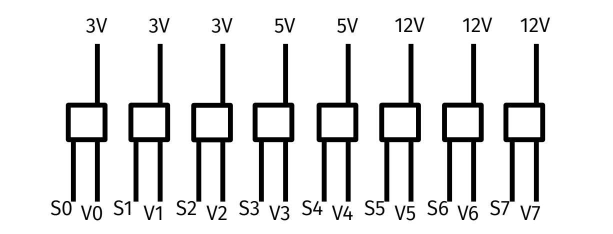

Figure 7: The general case for the relay bank.

1. What hex word would activate only the 5V systems?
2. What hex word would activate only the 12V sytems?
3. What hex word would activate just S_1, S_4, and S_7?

Other codings with binary and hexadecimals:

- Binary coded decimals
- Gray codes
- ASCII

Binary coded decimals (BCD):

The 8421 BCD system is often used in digital displays, readouts and number pads. It combines the familiarity of decimal digits with the utility of binary.

For example:
$0d234 = 0010\ 0011\ 0100$ in BCD.

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Table 6: The 8421 coding for BCD. Each four-digit bit sequence represents one decimal digit.

Quick exercise: Write your first name as a decimal string ($A = 1$, $B = 2$, etc.) and convert it to BCD.

Jordan corresponds to 10 15 18 04 01 14 so 00010000 00010101 00011000 00000100 00000001 00010100

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

Table 7: The 8421 coding for BCD. Each four-digit bit sequence represents one decimal digit.

The Gray Code:

The Gray code is a non-arithmetic code word system, related to binary, that minimizes transition errors by restricting one bit change per transition.

Binary to Gray Code conversion

1. The MSB is conserved.
2. Add consecutive binary bits starting with the MSB to obtain Gray Code bits.
3. Discard carries.

Binary	Gray Code
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
...	...

Table 8: The 4-bit Gray Code.

The Gray Code:

The Gray code is a non-arithmetic code word system, related to binary, that minimizes transition errors by restricting one bit change per transition.

Gray Code to Binary conversion

1. The MSB is conserved.
2. Add newest binary bit to next Gray Code bit to obtain next new binary bit.
3. Discard carries.

Binary	Gray Code
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
...	...

Table 9: The 4-bit Gray Code.

Other codings with binary and hexadecimals:

- Binary coded decimals
- Gray codes
- ASCII - American Standard Code for Information Interchange

HEXADECIMALS, BCD, GRAY-CODES, AND ASCII

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Figure 8: An ASCII table, used in computer science for encoding text from standard input.

Example: compiling the following C++ code on some machines still *rings the bell*.

```
#include <stdio.h>
using namespace std;
int main()
{
char y = 7;
printf("%c\n",y);
return 0;
}
```

Often used in systems and code involving exchanging strings of text.

CONCLUSION

Reading: Digital Fundamentals (DF) Ch. 2 (see Moodle)

1. Number representation
2. Binary conversions
3. Binary arithmetic
4. The floating-point system
5. Hexadecimals, Binary-Coded Decimals (BCD), Gray codes, and ASCII

Homework: exercises 5-16, 19-32, 35-38, 45, 53, 58 Ch. 2 (DF) (two weeks).