COSC2473

Number Systems

Data representation & Binary Numbers

<u>Other Number Systems and Characters</u>

Other Number Representations

Consider 1234_{10} and 1234_5

$$1234_{10} = 1*10^3 + 2 * 10^2 + 3 * 10^1 + 4 * 10^0$$

$$12345 = 1*5^3 + 2 * 5^2 + 3 * 5^1 + 4 * 5^0$$

Notice that while they are written the same, they are not the same value.

In General a number to base b,

$$n_3n_2n_1n_0\ _b = n_3 * b^3 + n_2 * b^2 + n_1 * b^1 + n_0 * b^0$$

Octal representation

- Expecting human programmers to read/write binary numbers is fraught with danger
  - Humans find binary difficult to work with accurately

- Need another number representation that is easier for humans to use, and easy to convert to/from binary
  - Octal

- Binary is base 2. Octal is base 8
  - Available symbols are 0,1,2,3,4,5,6,7

Octal representation

- Consider 2053_8

  $2 * 8^3 + 0 * 8^2 + 5 * 8^1 + 3 * 8^0$

  $2 * 512 + 0 * 64 + 5 * 8 + 3 * 1 = 1067_{10}$

- Binary for 1067_{10}

  10000101011

  – break the binary into groups of 3 (from LSB)

  10 000 101 011

  – write binary triplets as base_{10}

  2 0 5 3

Hexadecimal representation

- But notice that for octal, an 8 bit byte is represented by a pattern of nn nnn nnn and for 16 bits it is worse, being n nnn nnn nnn nnn mmm   which is still awkward.

- Hexadecimal numbers can be used to represent binary nibbles om groups of 4 which fits better with common word formats of 8,16, 24, 32,64,128 bits,

- With 4 bits, we can represent 16 symbols
  - Decimal only has 10 symbols 0..9
  - So, add 6 more symbols A,B,C,D,E,F

Hexadecimal representation

- With 4 bits we can represent:

- $0000 = 0_{16}$
- $0001 = 1_{16}$
- $0010 = 2_{16}$
- $0011 = 3_{16}$
- $0100 = 4_{16}$
- $0101 = 5_{16}$
- $0110 = 6_{16}$
- $0111 = 7_{16}$

- $1000 = 8_{16}$
- $1001 = 9_{16}$
- $1010 = A_{16}$
- $1011 = B_{16}$
- $1100 = C_{16}$
- $1101 = D_{16}$
- $1110 = E_{16}$
- $1111 = F_{16}$

Hexadecimal representation

- Consider binary for 1067_{10}

  010000101011

  - break the binary into groups of 4 (from LSB)

    0100 0010 1011

  - write binary quartets as $base_{16}$

    4 2 B

  - Binary 010000101011 is hexadecimal 42B

Other Number Schemes

- There are several other older number representations. We mention their names, but will discuss them in detail.
  - BCD – Binary Coded Decimal
  - EBCDIC – IBM's version of ASCII (see below)
  - Packed Decimal – similar to BCD

- Floating Point numbers
  - There are also binary coding systems for representing real numbers (eg 12.095, -42.333) or very large numbers (eg 1.3×10^{11}, 9.7×10^{-4})
  - They use a method called "Floating Point"
    - Numbers are almost never exact, but have a fixed precision
    - single-precision, double-precision or quad-precision

- These topics are outside the scope of this course

Characters

- There are 2 main codes in use for representing characters as binary
  - ***ASCI*** I (American Standard Code for Information Interchange) used by many older programming languages
  - ***Unicode*** is a newer series of codes, ranging in size from 8 bit, 16 or 32 bit. The larger number of bits allows a huge number of (non-European) characters to be represented. Currently over 110000 characters have been defined. Used by the Java programming language and .NET as well as Windows 10+ , recent Apple OS's
    - Comes is : UTF-8, (most common) , UTF-16, UTF-32
- An old code you might hear about was EBCDIC: used in some IBM mainframe syst

Characters

- ASCII is a 7 bit code
  (often extended to 8 bits)
- Some ASCII characters:
  01000001 = 'A'
  01000010 = 'B'
  01000011 = 'C'
  ...
  01011010 = 'Z'
  ...
  01100001 = 'a'
  01100010 = 'b'

- Note: 1 is not '1' !!

  00110000 = '0'
  00110001 = '1'
  00110010 = '2'

  00100000 = ' '
  00100001 = '!'

  00100100 = '$'
  01000000 = '@'

  00000111 = ring bell !!

ASCII Character Code Chart

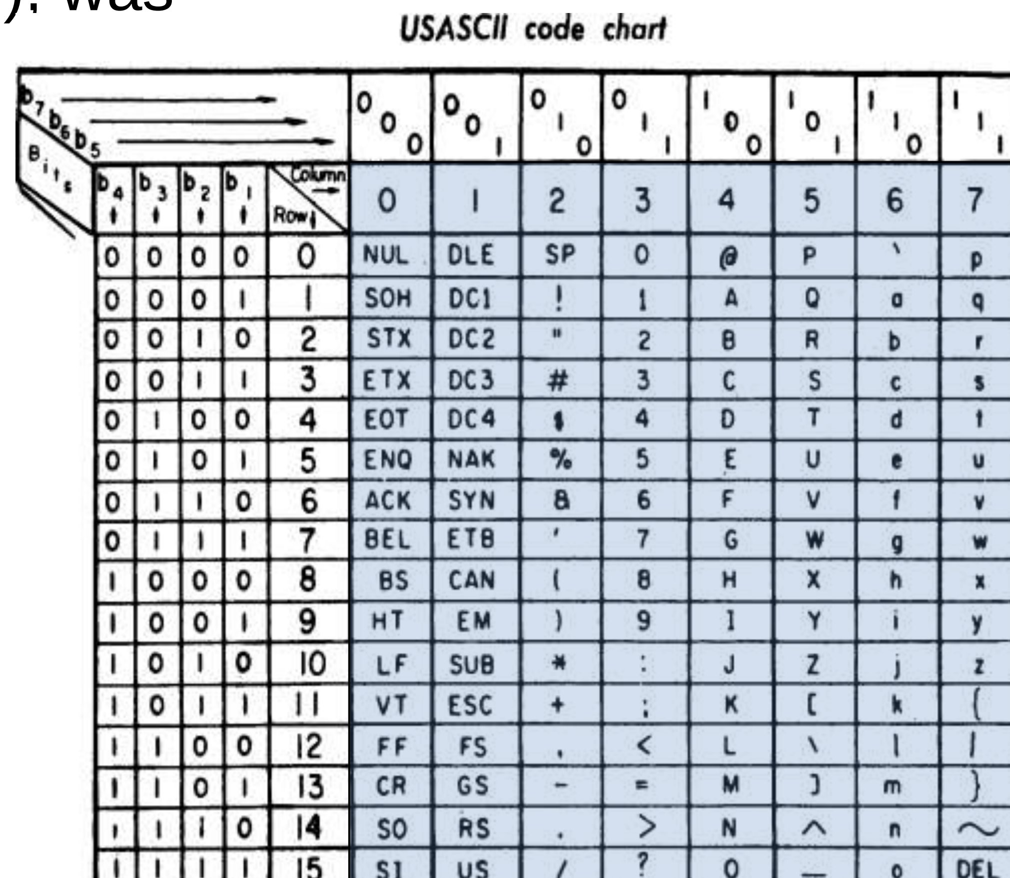
- If you look at a code chart, you will see an overall organisation. ASCII (American Standard Code for Information Interchange), was one of the first chart.

<u>Bits 6-4 of character data</u>

- <u>Bits 6-4 EX Description</u>
- 000-001 00-1F Device Control
- 010-011 20-3F Symbols + Digits
- 100-101 40-5A Upper Case + Sym
- 110-111 60-7A Lower case + Sym
- The MSB was reserved to be a parity bit (see next lectures)

Typical Binary Dump

```
Location  Hex values                Char Val
00000000 43 4F 53 43  2D 32 34 37 COSC-247
00000008 33 20 49 6E  74 72 6F 64 3 Introd
00000010 75 63 74 69  6F 6E 20 74 uction t
```



USASCII code chart

b7 b6 b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4	b3	b2	b1	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

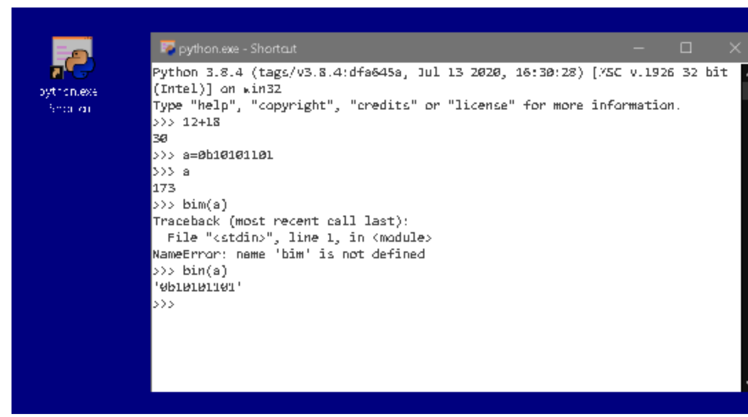
https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcR880d-xrwTX517IpQ5FUuR6uUAYU84Mv6ngg&usqp=CAU

Other Characters Codes

- Some web pages use different coding schemes, especially
  - Large Alphabet Languages (mainly CJK)
  - Example: www.cnd.org
  - use “view ->developer -> view source”
  - You will find a reference to gb 2312, which this browser understands to be a commonly used coding standard for Chinese (Guo Biao 2312)
- Again it is all binary data, the important thing is the program (browser) knows how to interpret it.

Python Code Snippets

- In this course, we use MicroPython within the BBC Micro-bit devices, but the language can also be easily installed in PCs and Unix, and students can use code snippets supplied here to demonstrate some of the issues described in this lecture series.



- Pictured at left is a typical Python window in which code snippets can be attempted. This is a role that languages such as BASIC have fulfilled in the past.
- The following and subsequent slides in the lecture series will show some code snippets for students to play with.

Python Code

- Python numbers can be expressed in 4 different bases:
  - Denoted by a leading ‘0’ (zero) followed by letter ‘b’,’o’,’x’, or ’X’
  - If there is no leading 0, it is assumed to be decimal
  - Binary.

```
>>> a = 0b10101101
>>> format(a,’b’)
‘10101101’
```

  - Octal.

```
>>> a = 0o255
>>> format(a,’o’)
‘255’
```

  - Decimal

```
>>> a = 173
```

  - Hexadecimal

```
>>> a = 0xad = 0XAD
>>> format(a,’X’)         # Upper case X
‘AD’
>>> format(a,’x’)         # lower case x
‘ad’
```

Python Code - Overflow

- If we implement negative numbers using 2's complement, what happens if we add 1 to the largest number possible?
- Python numbers effectively have unlimited precision due to how they are implemented, but we simulate using objects from the ctypes library.
  - Let's use the C type '**`ubyte`**' which is unsigned 8 bits

```
>>> from ctypes import *
```
Unsigned Byte = 8 bits

```
>>> c = c_ubyte(0b10101101)1
c_ubyte(173)
>>> format(c.value,'b')
'10101101'
```

Unsigned Byte = 8 bits

1	0	1	0	1	1	0	1

  - Since c is an object, we need to use c.value.
- Suppose we try to convert a number that is too large
  - Use **`byte`** instead which is 7 bits so too small for 173.

```
>>> c = c_byte(173)
>>> c
c_byte(-83)
>>> format(c.value,'b')
'10101101'
```

Signed Byte = 7 bits + sign bit

1	0	1	0	1	1	0	1

Python Code (Convert #1)

- Simple program to convert number n to base b

```
>>> def baseN(n, b, al="0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"):
...     s=""
...     while n > 0:
...        s = al[n % b] + s
...        n = n // b
...     return s
...
>>> baseN(25555,60)
'75T'
```

- Simple program to convert string s of base b to number

```
>>> def base10(s, b, al="0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"):
...     sum = 0
...     i = 0
...     while i < len(s):
...        sum = sum*b + al.find(s[i])
...        i += 1
...     return sum
...
>>> base10("75T",60)
25555
```

Quiz

- Given 01000001 is binary for ASCII ‘A’, what character is represented by 01000100 ?

- Research:
  - ASCII is a 7-bit Character code.. What was the 8th bit originally used for?
  - Unicode comes in several overall coding schemes:
    - UTF-8, UTF-16 and UTF-32
    - When would you use each of these?

- If you were to use a 5-bit word to represent character data, would you have enough values to represent it all.

- The characters “1537” using 8-bit ASCII would occupy 4 bytes. Since “

  1537”=01000001 01000101 01000011 01000111
  - Is there a simple way to convert to binary value?
  - How many bis would it need?

- Try to understand the Python conversion code in the previous slides