电子科技大学
格拉斯哥学院
Glasgow College, UESTC

UESTC4019: Real-Time Computer Systems and Architecture

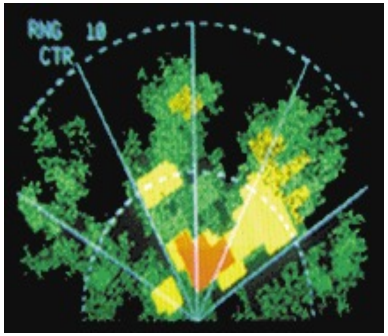
Lecture 3

Computer Number System

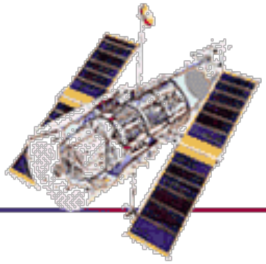
Part -1

Binary Digital Systems

- The world is analogue
  - Physical variables are continuous (apart from at the quantum level)
  - What we see, hear, feel - a continuous range of values
  - It is difficult to faithfully copy an analogue property
- Discrete variables are easier to quantify
- Problems where digital & analogue meet
  - cannot map infinite precision to discrete values in finite time
- Binary digital system is how computer manipulate representations of things
  - Such as, numbers, characters, pixels, money, position, instructions ….



Ground-proximity warning system



What do Computer do with Number

- Computer uses binary data to represent:
  - Numbers: Integers, Negative integers, reals, floating point i.e. 1,2,3,4,… -1, -2, -3 … 1.23, … $1.23\text{x}10^4$
  - Characters: A, B, 0, 1, a, b, c …
  - Instructions: Move, add, subtract, OR, AND …
  - Logical: True, False

Radix System (Base)

- Radix(Base) defines a set of symbols used to represent numbers in the system
- Radix point is the reference point that determine the value of each digit
  - Decimal symbols: 0,1,2,3,…9; Radix point: Decimal point
  - Binary symbols: 0,1; Radix point: Binary point
  - Hexadecimal symbols: 0,1,2,3,4..9,A,B,C,D,E,F; Radix point: Hexa-decimal point

Radix Representation

- Number representation:
  - $d_{31}d_{30} \ldots d_2d_1d_0$ is a 32 digit number
  - value = $(d_{31} \times B^{31}) + (d_{30} \times B^{30}) + \ldots + (d_2 \times B^2) + (d_1 \times B^1) + (d_0 \times B^0)$

    d: symbol(digit) & B: Radix(Base)

- In general, the relationship between a digit, its position and the base of the system is given by :
  - *Digit x Base Position*^{N}

Examples

- **Decimal Numbers: Base 10**
  - Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (10 symbols)
  - Example: $\mathbf{3271_{10}}$ =
- **Binary Numbers : Base 2**
  - Digits (Binary bits): 0, 1 (2 symbols)
  - Example: $\mathbf{1011010_2}$ =

    =
  - Note: **7 bits** (digit) binary number could turn into ONLY a **2** digit decimal number
- **Hex Numbers : Base 16**
  - Digits (Binary bits): 0, 1, 2,3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. (16 symbols)
  - Example: $\mathbf{11C_{16}}$ =
  - Note: 1 Hex Digit is equivalent to 4 bits binary.

Examples

- **Decimal Numbers: Base 10**
  - Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (10 symbols)
  - Example: $\mathbf{3271_{10}} = (\mathbf{3}x10^3) + (\mathbf{2}x10^2) + (\mathbf{7}x10^1) + (\mathbf{1}x10^0)$
- **Binary Numbers : Base 2**
  - Digits (Binary bits): 0, 1 (2 symbols)
  - $\mathbf{1011010_2} = (\mathbf{1}x2^6 + \mathbf{0}x2^5 + \mathbf{1}x2^4 + \mathbf{1}x2^3 + \mathbf{0}x2^2 + \mathbf{1}x2 + \mathbf{0}x1)_{10}$
    $= (64 + 16 + 8 + 2)_{10} = 90_{10}$
  - Note: **7 bits** (digit) binary number could turn into ONLY a **2** digit decimal number
- **Hex Numbers : Base 16**
  - Digits (Binary bits): 0, 1, 2,3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. (16 symbols)
  - $\mathbf{11C_{16}} = (\mathbf{1}x16^2 + 1x16^1 + 12x16^0)_{10} = (256 + 16 + 12)_{10} = 284_{10}$
  - Note: 1 Hex Digit is equivalent to 4 bits binary.

Hex to Binary

Hex	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Hex	Binary
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Binary/Hex Conversions

- Hex digits are in one-to-one correspondence with groups of four binary digits:

```
11    1010 0101 0110 . 1110 0010 1111 10
```

Binary/Hex Conversions

- Hex digits are in one-to-one correspondence with groups of four binary digits:

0011 1010 0101 0110 . 1110 0010 1111 1000

Binary/Hex Conversions

- Hex digits are in one-to-one correspondence with groups of four binary digits:

0011 1010 0101 0110 . 1110 0010 1111 1000

3 A 5 6 . E 2 F 8

Binary/Hex Conversions

- Hex digits are in one-to-one correspondence with groups of four binary digits:

0011 1010 0101 0110 . 1110 0010 1111 1000

3 A 5 6 . E 2 F 8

Conversion is a simple table lookup!

- Zero-fill on left and right ends to complete the groups!

Works because $16 = 2^4$ (power relationship)

Two Interpretations

unsigned ⟵ 10100111_2 ⟶ signed

- Signed vs. unsigned is a matter of interpretation; thus a single bit pattern can represent two different values
- Allowing both interpretations is useful:
  - Some data (e.g., count, age) can never be negative, and having a greater range is useful.

Two Interpretations

$$167_{10} \xleftarrow{\text{unsigned}} 10100111_2 \xrightarrow{\text{signed}} -89_{10}$$

- Signed vs. unsigned is a matter of interpretation; thus a single bit pattern can represent two different values.
- Allowing both interpretations is useful:
  - Some data (e.g., count, age) can never be negative, and having a greater range is useful.

Signed and Unsigned Number

- Unsigned number
  - For N-bit unsigned integer, the range is given by 0 to $(2^N-1)_{10}$
  - For 8-bits unsigned integer, the range is 0 to $(2^8-1)_{10}=255_{10}$
- Signed number (Two's complement)
  - For N-bit signed integer, the range is given by $-(2^{N-1})_{10}$ to $+(2^{N-1} - 1)_{10}$
  - For 8-bits signed integer, the range is $-(2^{8-1})_{10}$ to $+(2^{8-1}- 1)_{10}$=> -128_{10} to $+127_{10}$

Two's Complement

- Two's complement is the way most computer chooses to represent integers
- To get negative notation of an integer
  - invert the digits, and add one to the result
- Examples

  +28 (00011100) -> -28 (11100100)
  - 2’s Complement of value hex 0xFFFFFFFF is 0x00000001
- Arithmetic with 2’s Complement
  - Example 28 -28 = 0

  +28 00011100

  -28 11100100 +

  0 00000000 (8 bits)

Representation Width

Be Careful! You must be sure to pad the original value out to the full representation width before applying the algorithm!

Apply algorithm

Expand to 8-bits

Wrong: +25 = 11001 → 00111 → 00000111 = +7

Right: +25 = 11001 → 00011001 → 11100111 = -25

If positive: Add leading 0's
If negative: Add leading 1's

Apply algorithm

Signed and Unsigned Numbers in C

- C declaration `int`
  - Declares a signed number
  - Uses two's complement

- C declaration `unsigned int`
  - Declares an unsigned number
  - Treats 32-bit number as unsigned integer, most significant bit is part of the number, not a sign bit

- NOTE:
  - Hardware does all arithmetic in 2's complement.
  - It is up to programmer to interpret numbers as signed or unsigned.
  - Hardware provides some information to interpret numbers as signed or unsigned

Part -2

Floating Point Number

- Why floating-point numbers are needed
  - Since computer memory is limited, you cannot store numbers with infinite precision, no matter whether you use binary fractions or decimal ones: at some point you have to cut off
  - But how much accuracy is needed?
  - Where is it needed?
  - How many integer digits and how many fraction digits?

Floating Point Number

- To an engineer building a highway, it does not matter whether it’s 10 meters or 10.0001 meters wide - his measurements are probably not that accurate in the first place

- To someone designing a microchip, 0.0001 meters (a tenth of a millimeter) is a *huge* difference - But he’ll never have to deal with a distance larger than 0.1 meters

- A physicist needs to use the speed of light (about 300000000) and Newton’s gravitational constant (about 0.0000000000667) together in the same calculation

Floating Point Number

- To satisfy the engineer and the chip designer, a number format has to provide accuracy for numbers at very different magnitudes. However, only relative accuracy is needed.
- To satisfy the physicist, it must be possible to do calculations that involve numbers with different magnitudes
- Basically, having a fixed number of integer and fractional digits is not useful - and the solution is a format with a floating point

How Floating-point Numbers Work

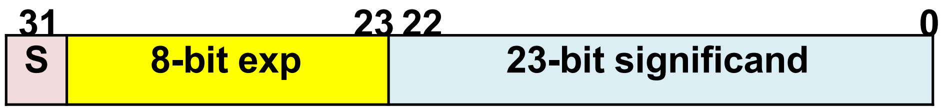
- The idea is to compose a number of two main parts:
  - A significand that contains the number’s digits. Negative significands represent negative numbers
  - An exponent that says where the decimal (or binary) point is placed relative to the beginning of the significand. Negative exponents represent numbers that are very small (i.e. close to zero).
- Such a format satisfies all the requirements:
  - It can represent numbers at wildly different magnitudes (limited by the length of the exponent)
  - It provides the same relative accuracy at all magnitudes (limited by the length of the significand)
  - It allows calculations across magnitudes: multiplying a very large and a very small number preserves the accuracy of both in the result

How Floating-point Numbers Work

- Decimal floating-point numbers usually take the form of scientific notation with an explicit point always between the 1st and 2nd digits
- The exponent is either written explicitly including the base, or an **e** is used to separate it from the significand

Significand	Exponent	Scientific notation	Fixed-point value
1.5	4	$1.5 \cdot 10^4$	15000
-2.001	2	$-2.001 \cdot 10^2$	-200.1
5	-3	$5 \cdot 10^{-3}$	0.005
6.667	-11	6.667e-11	0.00000000006667

IEEE-754 standard floating point



Single precision

- MSB is sign-bit (same as fixed point)
- 8-bit exponent in bias-127 integer format (i.e. add 127 to it)
- 23-bit signifcand to represent only the fractional part of the mantissa. The MSB of the mantissa is ALWAYS '1', therefore it is not stored

(The *mantissa*, also known as the *significand*, represents the precision bits of the number)

$$1.175 \times 10^{-38}\ (2^{-126}) < |x| < 3.4 \times 10^{38}\ (\sim 2 \times 2^{127})$$

Three components:

Base is implied

$$\pm\ \text{significand} \times 2^{\text{exponent}}$$

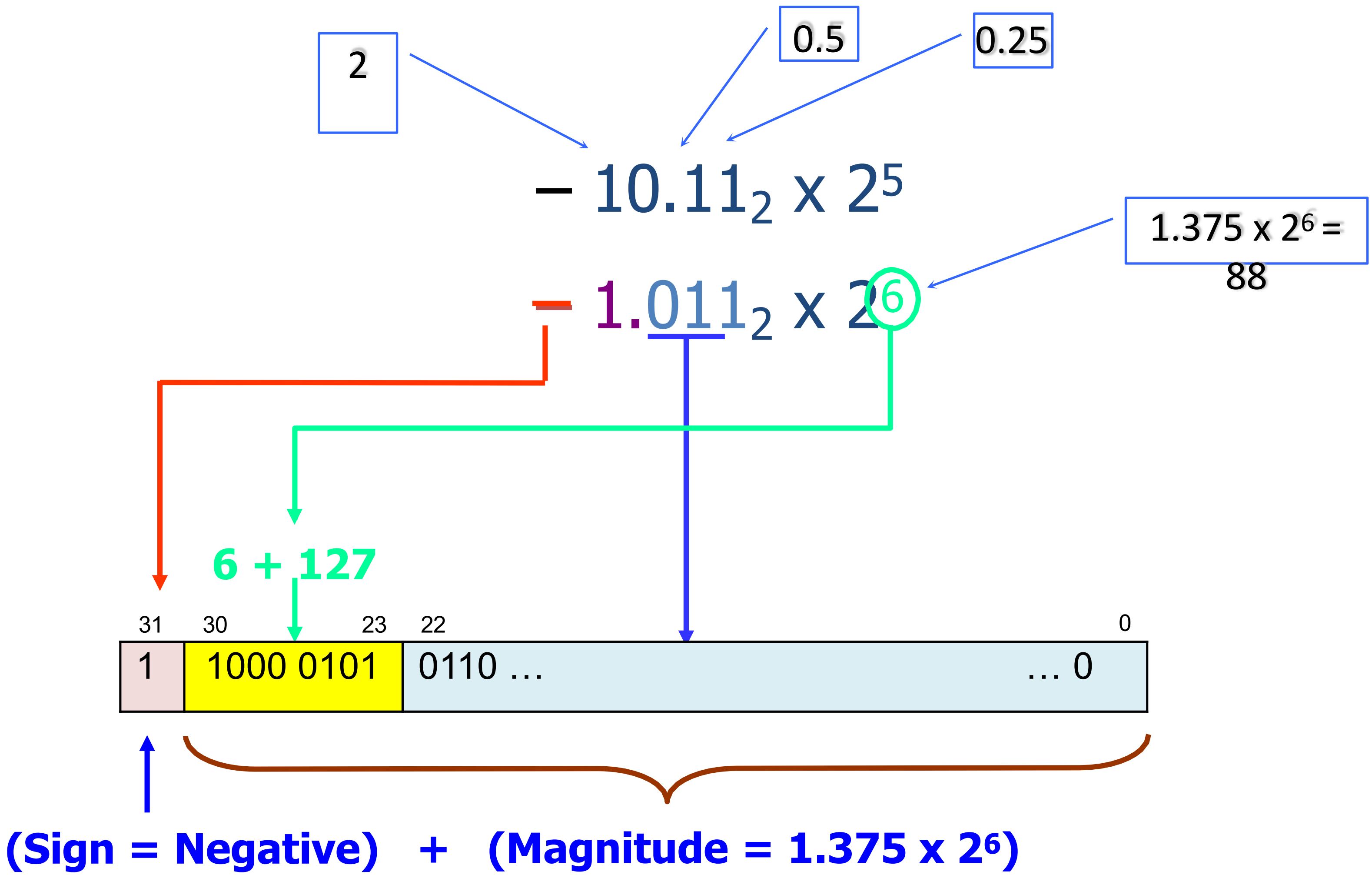
Sign

A biased integer value

An unsigned fractional value

Example: -88 =

Example: -88 = -2.75 x 2^5



IEEE-754 Double Precision format

63	62–52	51 … 0
S	11-bit exp (e)	52-bit significand

Double precision

- MSB is sign-bit (same as fixed point)
- 11-bit exponent in bias-1023 integer format (i.e. add 1023 to it)
- 52-bit signifcand to represent only the fractional part of the mantissa.

  The MSB of the mantissa is ALWAYS ‘1’, therefore it is not stored

  (The *mantissa*, also known as the *significand*, represents the precision bits of the number)

$$2.2250738585072014 \times 10^{-308} < |x| < 1.7976931348623157 \times 10^{308}$$

$$(2^{-1022}) \qquad\qquad (2 \times 2^{1023})$$

$$(-1)^{sign}\left(1 + \sum_{i=1}^{52}(b_{52-i} \times 2^{-i})\right) \times 2^{(e-1023)}$$

Fixed vs. Floating

- Floating-Point:

  Pro: Large dynamic range determined by exponent; resolution determined by significand

  Con: Implementation of arithmetic in hardware is complex (slow)

- Fixed-Point:

  Pro: Arithmetic is implemented using regular integer operations of processor (fast)

  Con: Limited range and resolution

Fixed-Point Reals

Three components:

Implied binary point

$0\cdots00.00\cdots0$

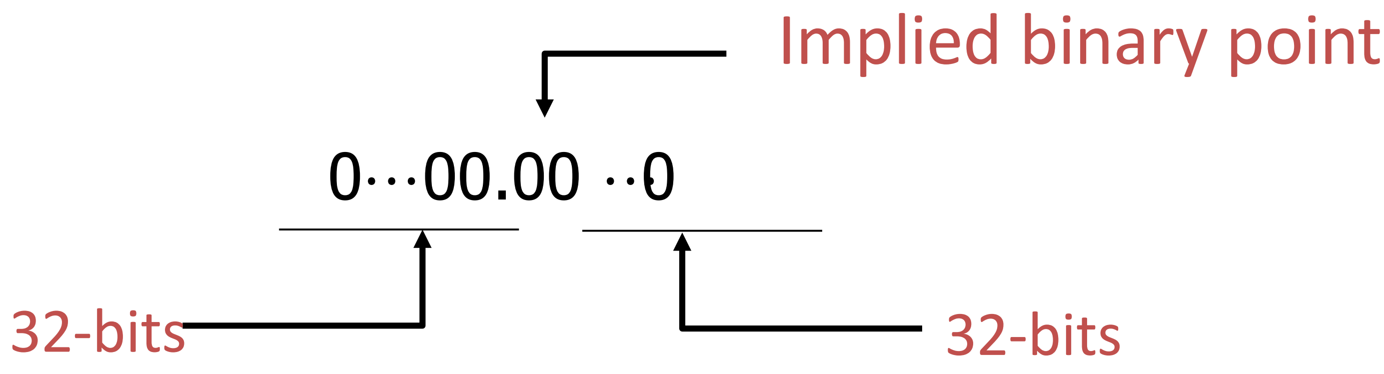
Whole part

Fractional part

Fixed-Point & Scale Factors

- The position of the binary point is determined by a *scale factor*
- Different variables can have different scale factors
- Determine scale factor by expected range and required resolution
- Programmer must keep track of scale factors! (Tedious)

32.32 Format



- This format uses lots of bits, but memory is relatively cheap and it supports both very large and very small numbers
- If all variables use this same format (i.e., a common scale factor), programming is simplified
- This is the strategy used in many 3D graphic engines Examples, Sony PlayStation, Nintendo video games, openGL etc .

Representation of Characters -ASCII CODE

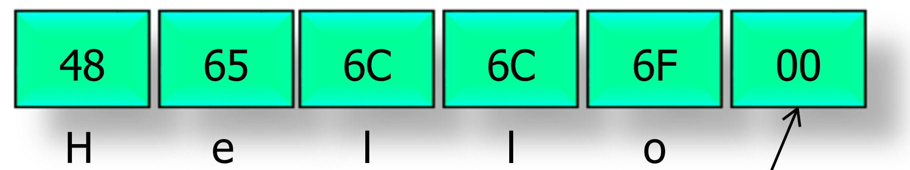
- 8 bit bytes represent characters, nearly every computer uses American Standard Code for Information Interchange (ASCII)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	16	10	DLE	32	20	Space	48	30	0	64	40	@	80	50	P	96	60	`` ` ``	112	70	p
1	01	SOH	17	11	DC1	33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
2	02	STX	18	12	DC2	34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
3	03	ETX	19	13	DC3	35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
4	04	EOT	20	14	DC4	36	24	$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
5	05	ENQ	21	15	NAK	37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
6	06	ACK	22	16	SYN	38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
7	07	BEL	23	17	ETB	39	27	'	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
8	08	BS	24	18	CAN	40	28	(	56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
9	09	HT	25	19	EM	41	29	)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
10	0A	LF	26	1A	SUB	42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
11	0B	VT	27	1B	ESC	43	2B	+	59	3B	;	75	4B	K	91	5B	[	107	6B	k	123	7B	{
12	0C	FF	28	1C	FS	44	2C		60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	|
13	0D	CR	29	1D	GS	45	2D	-	61	3D	=	77	4D	M	93	5D	]	109	6D	m	125	7D	}
14	0E	SO	30	1E	RS	46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
15	0F	SI	31	1F	US	47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	DEL

Character/Strings in C

- Characters normally combined into strings, which have variable length
  - Examples: “ab”, “IBM computer”, etc…
- How to represent a variable length string?
  - C uses **0, ‘\0’** (**null** in ASCII) to mark **end of string**
- How many bytes to represent the string ‘Cope’
- What are the values of the bytes for the string ‘Cope’
  - 67, 111, 112, 101, 0 Dec
  - 43, 6F, 70, 65, 0 Hex
- String in C program – an example.

  String simply an array of char



C uses a terminating “NUL” byte of all zeros at the end of the string.

```
void strcpy (char x[], char y[]){
  int i = 0; /* declare,initialize i*/

  while ((x[i] = y[i]) != '\0') /* 0 */
   i = i + 1; /* copy and test byte */
  }
```

Character Constants in C

- To distinguish a character that is used as data from an identifier that consists of only one character long:

  x		is an identifier.

  ‘x’		is a character constant.

- The value of ‘x’ is the ASCII code of the character x.

- Character Escapes - A way to represent characters that do not have a corresponding graphic symbol.

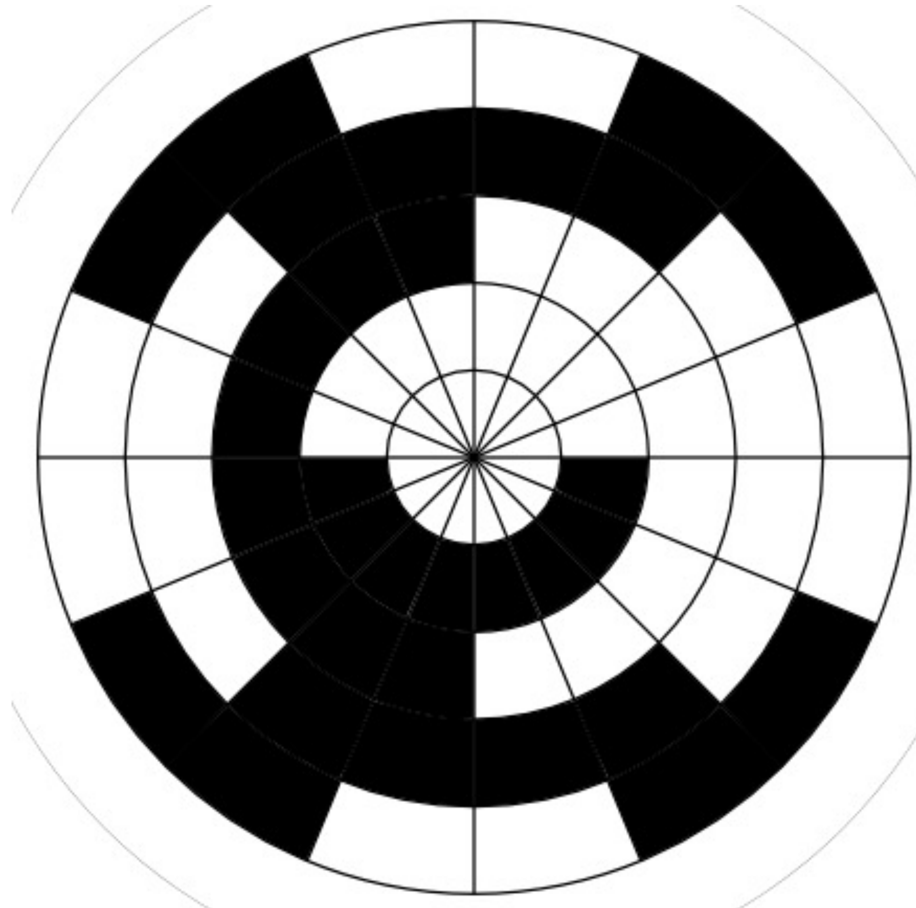
constant	Hex	Character	Constant	Hex	Character
‘\a’	07	Alert/bell	‘\v’	0B	Vert. tab
‘\b’	08	Backspace	‘\\’	5C	backslash
‘\f’	0C	Form feed	‘\’’	27	Single quote
‘\n’	0A	New line	‘\”’	22	Double quote
‘\r’	0D	Carriage return	‘\?’	3F	Question mark
‘\t’	09	Horz. tab			

PC 101 Keyboard Scancode (Typical)

Key	Down	Up	Key	Down	Up	Key	Down	Up	Key	Down	Up
Esc	1	81	[{	1A	9A	, <	33	B3	***center***	4C	CC
1 !	2	82	] }	1B	9B	. >	34	B4	***right***	4D	CD
2 @	3	83	Enter	1C	9C	/ ?	35	B5	***+***	4E	CE
3 #	4	84	Ctrl	1D	9D	R shift	36	B6	***end***	4F	CF
4 $	5	85	A	1E	9E	* PrtSc	37	B7	***down***	50	D0
5 %	6	86	S	1F	9F	alt	38	B8	***pgdn***	51	D1
6 ^	7	87	D	20	A0	space	39	B9	***ins***	52	D2
7 &	8	88	F	21	A1	CAPS	3A	BA	***del***	53	D3
8 *	9	89	G	22	A2	F1	3B	BB	/	E0 35	B5
9 (	0A	8A	H	23	A3	F2	3C	BC	***enter***	E0 1C	9C
0)	0B	8B	J	24	A4	F3	3D	BD	F11	57	D7
- _	0C	8C	K	25	A5	F4	3E	BE	F12	58	D8
= +	0D	8D	L	26	A6	F5	3F	BF	ins	E0 52	D2
Bksp	0E	8E	; :	27	A7	F6	40	C0	del	E0 53	D3
Tab	0F	8F	' "	28	A8	F7	41	C1	home	E0 47	C7
Q	10	90	` ~	29	A9	F8	42	C2	end	E0 4F	CF
W	11	91	L shift	2A	AA	F9	43	C3	pgup	E0 49	C9
E	12	92	\ |	2B	AB	F10	44	C4	pgdn	E0 51	D1
R	13	93	Z	2C	AC	NUM	45	C5	left	E0 4B	CB
T	14	94	X	2D	AD	SCRL	46	C6	right	E0 4D	CD
Y	15	95	C	2E	AE	***home***	47	C7	up	E0 48	C8
U	16	96	V	2F	AF	***up***	48	C8	down	E0 50	D0
I	17	97	B	30	B0	***pgup***	49	C9	R alt	E0 38	B8
O	18	98	N	31	B1	-	4A	CA	R ctrl	E0 1D	9D
P	19	99	M	32	B2	***left***	4B	CB	Pause	E1 1D 45 E1 9D C5	-

Gray Code

Gray Code					Binary			
2^3	2^2	2^1	2^0	Position	2^3	2^2	2^1	2^0
0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1
0	0	1	1	2	0	0	1	0
0	0	1	0	3	0	0	1	1
0	1	1	0	4	0	1	0	0
0	1	1	1	5	0	1	0	1
0	1	0	1	6	0	1	1	0
0	1	0	0	▶ 7 ◀	0	1	1	1
1	1	0	0	▶ 8 ◀	1	0	0	0
1	1	0	1	9	1	0	0	1
1	1	1	1	10	1	0	1	0
1	1	1	0	11	1	0	1	1
1	0	1	0	12	1	1	0	0
1	0	1	1	13	1	1	0	1
1	0	0	1	14	1	1	1	0
1	0	0	0	15	1	1	1	1



References

1. **The Principles of Computer Hardware**
   - **Alan Clements** 3rd Edition Oxford University Press (Chapter 4)
2. **Digital Systems Principles and Applications**
   - **RJ Tocci & NS Widmer** 8th Edition Prentice Hall (chapter 6)
3. **IEEE 754-2008**
   - en.*wikipedia.org/wiki/IEEE_754-2008, last assessed Jan 2012.*
4. **Keyboard Scancodes**
   - **Andries Brouwer,** http://www.win.tue.nl/~aeb/linux/kbd/scancodes.html#toc14; last assessed Jan 2012.