CMSC 313 Lecture 03

- **Multiple-byte data**
  - big-endian vs little-endian
  - sign extension
- **Multiplication and division**
- **Floating point formats**
- **Character Codes**

For the following questions, *show all of your work*. It is not sufficient to provide the answers.

Exercise 1. Convert the following numbers.

a. 62347_{10} to unsigned binary

b. $8DF6_{16}$ to base 2

c. 41.375_{10} to base 4

d. 10011101.0101_2 to base 10

Exercise 2. Convert each of the following numbers to 8-bit signed magnitude, 8-bit one's complement, 8-bit two's complement and 8-bit excess 128 formats.

a. $(-122)_{10}$

b. $(-31)_{10}$

c. $(-16)_{10}$

d. 127_{10}

Exercise 3. Find the decimal equivalents for the following 8-bit two's complement numbers.

a. 1000 0001

b. 0111 1011

c. 1111 0001

d. 0010 1010

Exercise 4. Perform two's complement addition on the following pairs of numbers. In each case, indicate whether an overflow has occured.

a. 1110 1011 + 0110 1001

b. 1110 1011 + 1111 1111

c. 1000 1100 + 1100 0001

d. 0111 1001 + 0000 1001

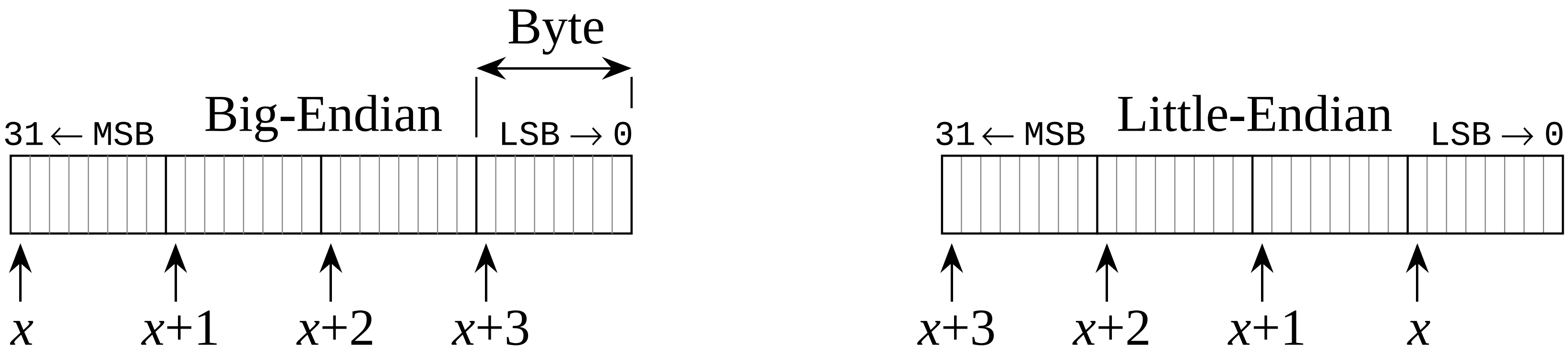
Common Sizes for Data Types

- A byte is composed of 8 bits. Two nibbles make up a byte.
- Halfwords, words, doublewords, and quadwords are composed of bytes as shown below:

Bit	0
Nibble	0110
Byte	10110000
16-bit word (halfword)	11001001 01000110
32-bit word	10110100 00110101 10011001 01011000
64-bit word (double)	01011000 01010101 10110000 11110011
	11001110 11101110 01111000 00110101
128-bit word (quad)	01011000 01010101 10110000 11110011
	11001110 11101110 01111000 00110101
	00001011 10100110 11110010 11100110
	10100100 01000100 10100101 01010001

Big-Endian and Little-Endian Formats

- In a byte-addressable machine, the smallest datum that can be referenced in memory is the byte. Multi-byte words are stored as a sequence of bytes, in which the address of the multi-byte word is the same as the byte of the word that has the lowest address.
- When multi-byte words are used, two choices for the order in which the bytes are stored in memory are: most significant byte at lowest address, referred to as *big-endian*, or least significant byte stored at lowest address, referred to as *little-endian*.



Word address is *x* for both big-endian and little-endian formats.

Two's Complement Sign Extension

Decimal	8-bit	16-bit
+5	0000 0101	0000 0000 0000 0101
-5	1111 1011	1111 1111 1111 1011

- **Why does sign extension work?**

-x is represented as 2^8 - x in 8-bit

-x is represented as 2^{16} - x in 16-bit

$2^8 - x + ??? = 2^{16} - x$

$??? = 2^{16} - 2^8$

```
   1 0000 0000 0000 0000 = 65536
-              1 0000 0000 =   256
------------------------------------
     1111 1111 0000 0000 = 65280
```

Multiplication Example

- **Multiplication of two 4-bit unsigned binary integers produces an 8-bit result.**

$$
\begin{array}{cccccccccll}
 & & & & 1 & 1 & 0 & 1 & & (13)_{10} & \text{Multiplicand M} \\
 & & & \times & 1 & 0 & 1 & 1 & & (11)_{10} & \text{Multiplier Q} \\
\hline
 & & & & 1 & 1 & 0 & 1 & & & \\
 & & & 1 & 1 & 0 & 1 & & & & \\
 & & 0 & 0 & 0 & 0 & & & & & \text{Partial products} \\
 & 1 & 1 & 0 & 1 & & & & & & \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & & (143)_{10} & \text{Product P}
\end{array}
$$

- **Multiplication of two 4-bit signed binary integers produces only a 7-bit result (each operand reduces to a sign bit and a 3-bit magnitude for each operand, producing a sign-bit and a 6-bit result).**

Multiplication of Signed Integers

- Sign extension to the target word size is needed for the negative operand(s).
- A target word size of 8 bits is used here for two 4-bit signed operands, but only a 7-bit target word size is needed for the result.

$$
\begin{array}{rrrrrrrrl}
 & & & & 1 & 1 & 1 & 1 & \quad (-1)_{10} \\
 & & & \times & 0 & 0 & 0 & 1 & \quad (+1)_{10} \\
\hline
 & & & & 1 & 1 & 1 & 1 & \\
 & & & 0 & 0 & 0 & 0 & & \\
 & & 0 & 0 & 0 & 0 & & & \\
 & 0 & 0 & 0 & 0 & & & & \\
\hline
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \quad (+15)_{10}
\end{array}
$$

(Incorrect; result should be –1)

$$
\begin{array}{rrrrrrrrrl}
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \quad (-1)_{10} \\
\times & & & & & 0 & 0 & 0 & 1 & \quad (+1)_{10} \\
\hline
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & \\
 & 0 & 0 & 0 & 0 & 0 & 0 & & & \\
 & 0 & 0 & 0 & 0 & 0 & & & & \\
\hline
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \quad (-1)_{10}
\end{array}
$$

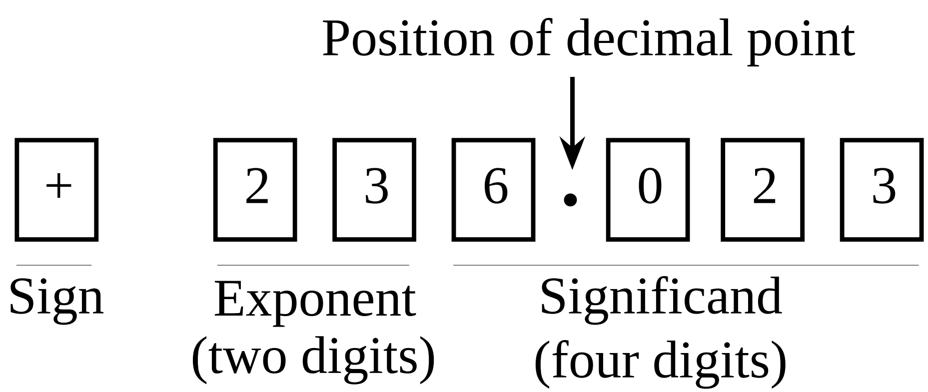
Example of Base 2 Division

- $(7 / 3 = 2)_{10}$ with a remainder R of 1.
- Equivalently, $(0111/ 11 = 10)_2$ with a remainder R of 1.

```
      0 0 1 0  R1
    ____________
1 1 | 0 1 1 1
        1 1
      ______
          0 1
```


Base 10 Floating Point Numbers

- **Floating point numbers allow very large and very small numbers to be represented using only a few digits, at the expense of precision. The precision is primarily determined by the number of digits in the fraction (or *significand*, which has integer and fractional parts), and the range is primarily determined by the number of digits in the exponent.**
- **Example (+6.023 $\times$ 10^{23}):**



Normalization

- The base 10 number 254 can be represented in floating point form as 254×10^0, or equivalently as:

  25.4×10^1, or

  2.54×10^2, or

  $.254 \times 10^3$, or

  $.0254 \times 10^4$, or

  infinitely many other ways, which creates problems when making comparisons, with so many representations of the same number.
- Floating point numbers are usually *normalized*, in which the radix point is located in only one possible position for a given number.
- Usually, but not always, the normalized representation places the radix point immediately to the left of the leftmost, nonzero digit in the fraction, as in: $.254 \times 10^3$.

Floating Point Example

- Represent $.254 \times 10^3$ in a normalized base 8 floating point format with a sign bit, followed by a 3-bit excess 4 exponent, followed by four base 8 digits.
- Step #1: Convert to the target base.

  $.254 \times 10^3 = 254_{10}$. Using the remainder method, we find that $254_{10} = 376 \times 8^0$:

  254/8 = 31 R 6

  31/8 = 3 R 7

  3/8 = 0 R 3
- Step #2: Normalize: $376 \times 8^0 = .376 \times 8^3$.
- Step #3: Fill in the bit fields, with a positive sign (sign bit = 0), an exponent of 3 + 4 = 7 (excess 4), and 4-digit fraction = .3760:

0 111 . 011 111 110 000

Error, Range, and Precision

- In the previous example, we have the base $b = 8$, the number of significant digits (not bits!) in the fraction $s = 4$, the largest exponent value (not bit pattern) $M = 3$, and the smallest exponent value $m = -4$.
- In the previous example, there is no explicit representation of 0, but there needs to be a special bit pattern reserved for 0 otherwise there would be no way to represent 0 without violating the normalization rule. We will assume a bit pattern of 0 000 000 000 000 000 represents 0.
- Using b, s, M, and m, we would like to characterize this floating point representation in terms of the largest positive representable number, the smallest (nonzero) positive representable number, the smallest gap between two successive numbers, the largest gap between two successive numbers, and the total number of numbers that can be represented.

Error, Range, and Precision (cont')

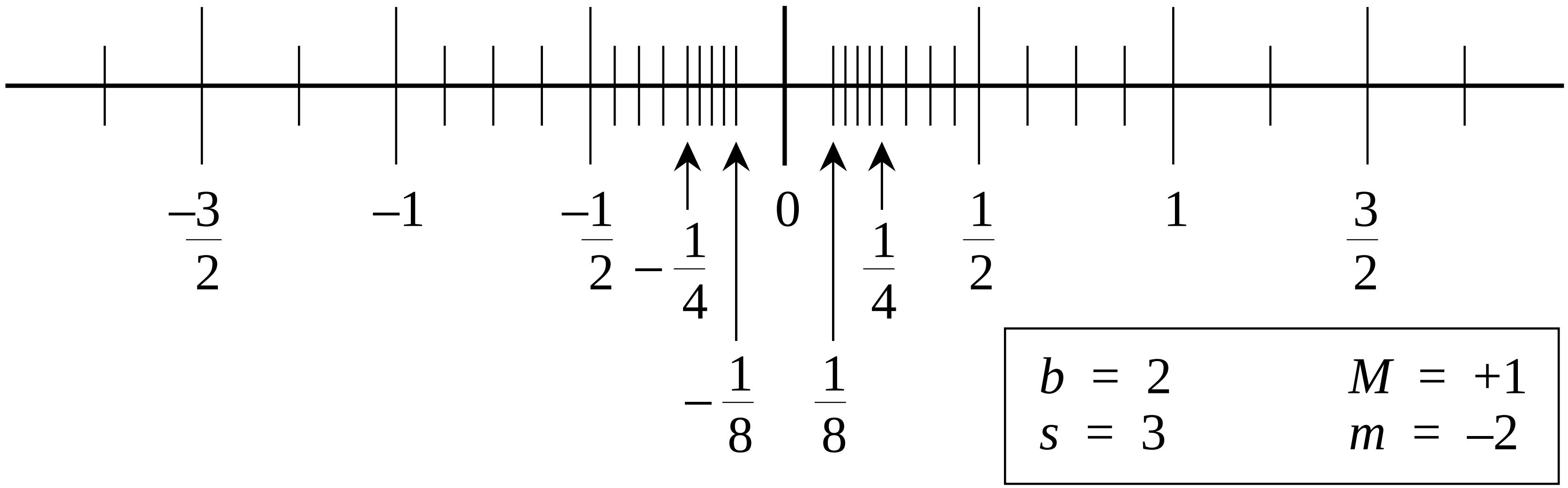
- **Largest representable number:** $b^M \times (1 - b^{-s}) = 8^3 \times (1 - 8^{-4})$
- **Smallest representable number:** $b^m \times b^{-1} = 8^{-4 - 1} = 8^{-5}$
- **Largest gap:** $b^M \times b^{-s} = 8^{3 - 4} = 8^{-1}$
- **Smallest gap:** $b^m \times b^{-s} = 8^{-4 - 4} = 8^{-8}$

Error, Range, and Precision (cont’)

$$\underset{\substack{\uparrow\\ \text{Sign bit}}}{\overset{\text{A}}{2}} \times \overset{\text{B}}{\underbrace{((M - m) + 1)}_{\text{The number of exponents}}} \times \overset{\text{C}}{\underbrace{(b - 1)}_{\text{First digit of fraction}}} \times \overset{\text{D}}{\underbrace{b^{s-1}}_{\text{Remaining digits of fraction}}} + \underset{\substack{\uparrow\\ \text{Zero}}}{\overset{\text{E}}{1}}$$

- **Number of representable numbers: There are 5 components: (A) sign bit; for each number except 0 for this case, there is both a positive and negative version; (B) (*M* - *m*) + 1 exponents; (C) b - 1 values for the first digit (0 is disallowed for the first normalized digit); (D) b^{s-1} values for each of the *s*-1 remaining digits, plus (E) a special representation for 0. For this example, the 5 components result in: $2 \times ((3 - 4) + 1) \times (8 - 1) \times 8^{4-1} + 1$ numbers that can be represented. Notice this number must be no greater than the number of possible bit patterns that can be generated, which is 2^{16}.**

Example Floating Point Format



- Smallest number is 1/8
- Largest number is 7/4
- Smallest gap is 1/32
- Largest gap is 1/4
- Number of representable numbers is 33.

Gap Size Follows Exponent Size

- The relative error is approximately the same for all numbers.
- If we take the ratio of a large gap to a large number, and compare that to the ratio of a small gap to a small number, then the ratios are the same:

A large gap $\longrightarrow$

A large number $\longrightarrow$

$$\frac{b^{M-s}}{b^{M} \times (1 - b^{-s})} = \frac{b^{-s}}{1 - b^{-s}} = \frac{1}{b^{s} - 1}$$

A small gap $\longrightarrow$

A small number $\longrightarrow$

$$\frac{b^{m-s}}{b^{m} \times (1 - b^{-s})} = \frac{b^{-s}}{1 - b^{-s}} = \frac{1}{b^{s} - 1}$$

Conversion Example

- **<u>Example:</u>** Convert $(9.375 \times 10^{-2})_{10}$ to base 2 scientific notation
- Start by converting from base 10 floating point to base 10 fixed point by moving the decimal point two positions to the left, which corresponds to the -2 exponent: .09375.
- Next, convert from base 10 fixed point to base 2 fixed point:

$$
\begin{array}{llcll}
.09375 & \times & 2 & = & 0.1875 \\
.1875 & \times & 2 & = & 0.375 \\
.375 & \times & 2 & = & 0.75 \\
.75 & \times & 2 & = & 1.5 \\
.5 & \times & 2 & = & 1.0
\end{array}
$$

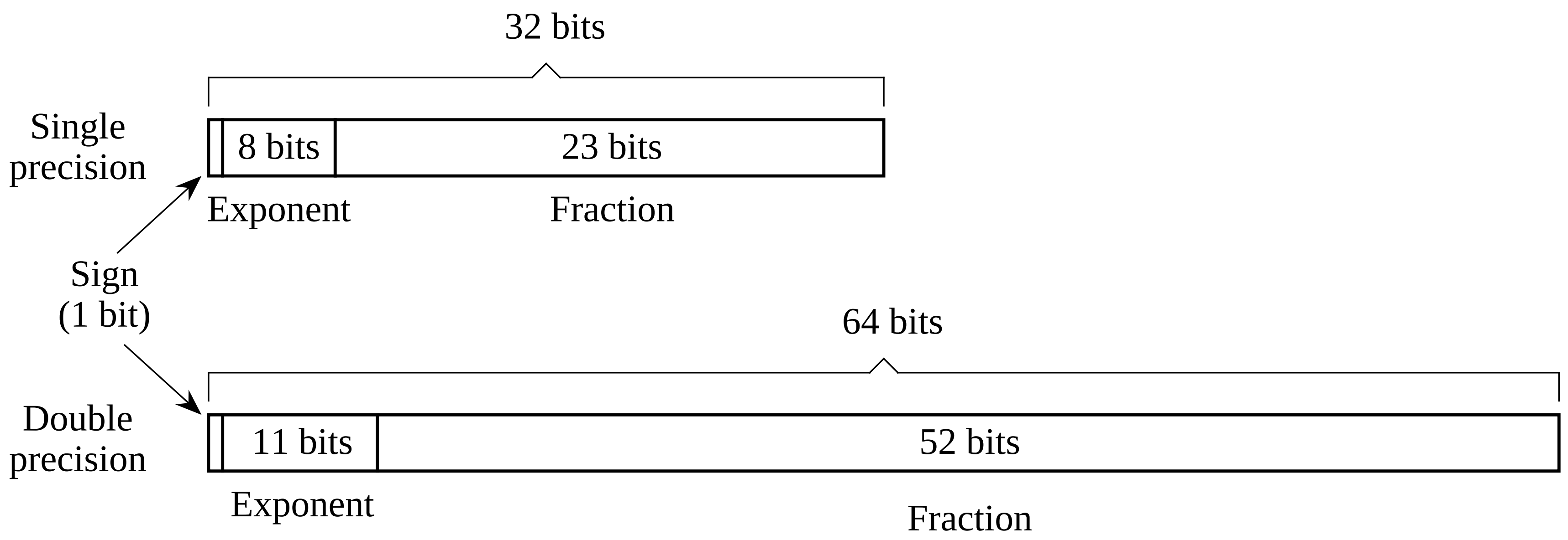
- Thus, $(.09375)_{10} = (.00011)_2$.
- Finally, convert to normalized base 2 floating point:

$$.00011 = .00011 \times 2^0 = 1.1 \times 2^{-4}$$

IEEE-754 32-bit Floating Point Format

- sign bit, 8-bit exponent, 23-bit mantissa
- normalized as 1.xxxxx
- leading 1 is hidden
- 8-bit exponent in excess 127 format
  - NOT excess 128
  - `0000 0000` and `1111 1111` are reserved
- +0 and -0 is zero exponent and zero mantissa
- `1111 1111` exponent and zero mantissa is infinity

IEEE-754 Floating Point Formats



IEEE-754 Examples

	Value	Bit Pattern: Sign	Exponent	Fraction
(a)	$+1.101 \times 2^{5}$	0	1000 0100	101 0000 0000 0000 0000 0000
(b)	-1.01011×2^{-126}	1	0000 0001	010 1100 0000 0000 0000 0000
(c)	$+1.0 \times 2^{127}$	0	1111 1110	000 0000 0000 0000 0000 0000
(d)	$+0$	0	0000 0000	000 0000 0000 0000 0000 0000
(e)	-0	1	0000 0000	000 0000 0000 0000 0000 0000
(f)	$+\infty$	0	1111 1111	000 0000 0000 0000 0000 0000
(g)	$+2^{-128}$	0	0000 0000	010 0000 0000 0000 0000 0000
(h)	+NaN	0	1111 1111	011 0111 0000 0000 0000 0000
(i)	$+2^{-128}$	0	011 0111 1111	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

IEEE-754 Conversion Example

- Represent -12.625_{10} in single precision IEEE-754 format.
- Step #1: Convert to target base. $-12.625_{10} = -1100.101_2$
- Step #2: Normalize. $-1100.101_2 = -1.100101_2 \times 2^3$
- Step #3: Fill in bit fields. Sign is negative, so sign bit is 1. Exponent is in excess 127 (not excess 128!), so exponent is represented as the unsigned integer 3 + 127 = 130. Leading 1 of significand is hidden, so final bit pattern is:

1 1000 0010 . 1001 0100 0000 0000 0000 000

Floating Point Arithmetic

- Floating point arithmetic differs from integer arithmetic in that exponents must be handled as well as the magnitudes of the operands.
- The exponents of the operands must be made equal for addition and subtraction. The fractions are then added or subtracted as appropriate, and the result is normalized.
- Ex: Perform the floating point operation: $(.101 \times 2^3 + .111 \times 2^4)_2$
- Start by adjusting the *smaller* exponent to be equal to the larger exponent, and adjust the fraction accordingly. Thus we have $.101 \times 2^3 = .010 \times 2^4$, losing $.001 \times 2^3$ of precision in the process.
- The resulting sum is $(.010 + .111) \times 2^4 = 1.001 \times 2^4 = .1001 \times 2^5$, and rounding to three significant digits, $.100 \times 2^5$, and we have lost another 0.001×2^4 in the rounding process.

Floating Point Multiplication/Division

- Floating point multiplication/division are performed in a manner similar to floating point addition/subtraction, except that the sign, exponent, and fraction of the result can be computed separately.
- Like/unlike signs produce positive/negative results, respectively. Exponent of result is obtained by adding exponents for multiplication, or by subtracting exponents for division. Fractions are multiplied or divided according to the operation, and then normalized.
- Ex: Perform the floating point operation: $(+.110 \times 2^5) / (+.100 \times 2^4)_2$
- The source operand signs are the same, which means that the result will have a positive sign. We subtract exponents for division, and so the exponent of the result is 5 – 4 = 1.
- We divide fractions, producing the result: 110/100 = 1.10.
- Putting it all together, the result of dividing $(+.110 \times 2^5)$ by $(+.100 \times 2^4)$ produces $(+1.10 \times 2^1)$. After normalization, the final result is $(+.110 \times 2^2)$.

ASCII Character Code

- **ASCII is a 7-bit code, commonly stored in 8-bit bytes.**
- **"A" is at 41_{16}. To convert upper case letters to lower case letters, add 20_{16}. Thus "a" is at 41_{16} + 20_{16} = 61_{16}.**
- **The character "5" at position 35_{16} is different than the number 5. To convert character-numbers into number-numbers, subtract 30_{16}: 35_{16} - 30_{16} = 5.**

00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	´	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

NUL	Null	FF	Form feed	CAN	Cancel
SOH	Start of heading	CR	Carriage return	EM	End of medium
STX	Start of text	SO	Shift out	SUB	Substitute
ETX	End of text	SI	Shift in	ESC	Escape
EOT	End of transmission	DLE	Data link escape	FS	File separator
ENQ	Enquiry	DC1	Device control 1	GS	Group separator
ACK	Acknowledge	DC2	Device control 2	RS	Record separator
BEL	Bell	DC3	Device control 3	US	Unit separator
BS	Backspace	DC4	Device control 4	SP	Space
HT	Horizontal tab	NAK	Negative acknowledge	DEL	Delete
LF	Line feed	SYN	Synchronous idle		
VT	Vertical tab	ETB	End of transmission block		

EBCDIC Character Code

- **EBCDIC is an 8-bit code.**

00	NUL	20	DS	40	SP	60	–	80		A0		C0	{	E0	\
01	SOH	21	SOS	41		61	/	81	a	A1	~	C1	A	E1	
02	STX	22	FS	42		62		82	b	A2	s	C2	B	E2	S
03	ETX	23		43		63		83	c	A3	t	C3	C	E3	T
04	PF	24	BYP	44		64		84	d	A4	u	C4	D	E4	U
05	HT	25	LF	45		65		85	e	A5	v	C5	E	E5	V
06	LC	26	ETB	46		66		86	f	A6	w	C6	F	E6	W
07	DEL	27	ESC	47		67		87	g	A7	x	C7	G	E7	X
08		28		48		68		88	h	A8	y	C8	H	E8	Y
09		29		49		69		89	i	A9	z	C9	I	E9	Z
0A	SMM	2A	SM	4A	¢	6A	‘	8A		AA		CA		EA	
0B	VT	2B	CU2	4B		6B	,	8B		AB		CB		EB	
0C	FF	2C		4C	<	6C	%	8C		AC		CC		EC	
0D	CR	2D	ENQ	4D	(	6D	_	8D		AD		CD		ED	
0E	SO	2E	ACK	4E	+	6E	>	8E		AE		CE		EE	
0F	SI	2F	BEL	4F	|	6F	?	8F		AF		CF		EF	
10	DLE	30		50	&	70		90		B0		D0	}	F0	0
11	DC1	31		51		71		91	j	B1		D1	J	F1	1
12	DC2	32	SYN	52		72		92	k	B2		D2	K	F2	2
13	TM	33		53		73		93	l	B3		D3	L	F3	3
14	RES	34	PN	54		74		94	m	B4		D4	M	F4	4
15	NL	35	RS	55		75		95	n	B5		D5	N	F5	5
16	BS	36	UC	56		76		96	o	B6		D6	O	F6	6
17	IL	37	EOT	57		77		97	p	B7		D7	P	F7	7
18	CAN	38		58		78		98	q	B8		D8	Q	F8	8
19	EM	39		59		79		99	r	B9		D9	R	F9	9
1A	CC	3A		5A	!	7A	:	9A		BA		DA		FA	|
1B	CU1	3B	CU3	5B	$	7B	#	9B		BB		DB		FB	
1C	IFS	3C	DC4	5C	·	7C	@	9C		BC		DC		FC	
1D	IGS	3D	NAK	5D	)	7D	'	9D		BD		DD		FD	
1E	IRS	3E		5E	;	7E	=	9E		BE		DE		FE	
1F	IUS	3F	SUB	5F	¬	7F	"	9F		BF		DF		FF	

STX	Start of text	RS	Reader Stop	DC
DLE	Data Link Escape	PF	Punch Off	DC
BS	Backspace	DS	Digit Select	DC
ACK	Acknowledge	PN	Punch On	CU
SOH	Start of Heading	SM	Set Mode	CU
ENQ	Enquiry	LC	Lower Case	CU
ESC	Escape	CC	Cursor Control	SY
BYP	Bypass	CR	Carriage Return	IF
CAN	Cancel	EM	End of Medium	EC
RES	Restore	FF	Form Feed	ET
SI	Shift In	TM	Tape Mark	NA
SO	Shift Out	UC	Upper Case	SM
DEL	Delete	FS	Field Separator	SC
SUB	Substitute	HT	Horizontal Tab	IG
NL	New Line	VT	Vertical Tab	IR
LF	Line Feed	UC	Upper Case	IU

Unicode Character Code

- **Unicode is a 16-bit code.**

0000	NUL	0020	SP	0040	@	0060	`	0080	Ctrl	00A0	NBS	00C0	À	00E0	à
0001	SOH	0021	!	0041	A	0061	a	0081	Ctrl	00A1	¡	00C1	Á	00E1	á
0002	STX	0022	"	0042	B	0062	b	0082	Ctrl	00A2	¢	00C2	Â	00E2	â
0003	ETX	0023	#	0043	C	0063	c	0083	Ctrl	00A3	£	00C3	Ã	00E3	ã
0004	EOT	0024	$	0044	D	0064	d	0084	Ctrl	00A4	¤	00C4	Ä	00E4	ä
0005	ENQ	0025	%	0045	E	0065	e	0085	Ctrl	00A5	¥	00C5	Å	00E5	å
0006	ACK	0026	&	0046	F	0066	f	0086	Ctrl	00A6	¦	00C6	Æ	00E6	æ
0007	BEL	0027	'	0047	G	0067	g	0087	Ctrl	00A7	§	00C7	Ç	00E7	ç
0008	BS	0028	(	0048	H	0068	h	0088	Ctrl	00A8	¨	00C8	È	00E8	è
0009	HT	0029	)	0049	I	0069	i	0089	Ctrl	00A9	©	00C9	É	00E9	é
000A	LF	002A	*	004A	J	006A	j	008A	Ctrl	00AA	ª	00CA	Ê	00EA	ê
000B	VT	002B	+	004B	K	006B	k	008B	Ctrl	00AB	«	00CB	Ë	00EB	ë
000C	FF	002C	´	004C	L	006C	l	008C	Ctrl	00AC	¬	00CC	Ì	00EC	ì
000D	CR	002D	-	004D	M	006D	m	008D	Ctrl	00AD	–	00CD	Í	00ED	í
000E	SO	002E	.	004E	N	006E	n	008E	Ctrl	00AE	®	00CE	Î	00EE	î
000F	SI	002F	/	004F	O	006F	o	008F	Ctrl	00AF	¯	00CF	Ï	00EF	ï
0010	DLE	0030	0	0050	P	0070	p	0090	Ctrl	00B0	°	00D0	Đ	00F0	¶
0011	DC1	0031	1	0051	Q	0071	q	0091	Ctrl	00B1	±	00D1	Ñ	00F1	ñ
0012	DC2	0032	2	0052	R	0072	r	0092	Ctrl	00B2	²	00D2	Ò	00F2	ò
0013	DC3	0033	3	0053	S	0073	s	0093	Ctrl	00B3	³	00D3	Ó	00F3	ó
0014	DC4	0034	4	0054	T	0074	t	0094	Ctrl	00B4	´	00D4	Ô	00F4	ô
0015	NAK	0035	5	0055	U	0075	u	0095	Ctrl	00B5	µ	00D5	Õ	00F5	õ
0016	SYN	0036	6	0056	V	0076	v	0096	Ctrl	00B6	¶	00D6	Ö	00F6	ö
0017	ETB	0037	7	0057	W	0077	w	0097	Ctrl	00B7	·	00D7	×	00F7	÷
0018	CAN	0038	8	0058	X	0078	x	0098	Ctrl	00B8	¸	00D8	Ø	00F8	ø
0019	EM	0039	9	0059	Y	0079	y	0099	Ctrl	00B9	¹	00D9	Ù	00F9	ù
001A	SUB	003A	:	005A	Z	007A	z	009A	Ctrl	00BA	º	00DA	Ú	00FA	ú
001B	ESC	003B	;	005B	[	007B	{	009B	Ctrl	00BB	»	00DB	Û	00FB	û
001C	FS	003C	<	005C	\	007C	|	009C	Ctrl	00BC	1/4	00DC	Ü	00FC	ü
001D	GS	003D	=	005D	]	007D	}	009D	Ctrl	00BD	1/2	00DD	Ý	00FD	Þ
001E	RS	003E	>	005E	^	007E	~	009E	Ctrl	00BE	3/4	00DE	ý	00FE	þ
001F	US	003F	?	005F	_	007F	DEL	009F	Ctrl	00BF	¿	00DF	§	00FF	ÿ

NUL	Null	SOH	Start of heading	CAN	Cancel	SP	Space
STX	Start of text	EOT	End of transmission	EM	End of medium	DEL	Delete
ETX	End of text	DC1	Device control 1	SUB	Substitute	Ctrl	Control
ENQ	Enquiry	DC2	Device control 2	ESC	Escape	FF	Form feed
ACK	Acknowledge	DC3	Device control 3	FS	File separator	CR	Carriage return
BEL	Bell	DC4	Device control 4	GS	Group separator	SO	Shift out
BS	Backspace	NAK	Negative acknowledge	RS	Record separator	SI	Shift in
HT	Horizontal tab	NBS	Non-breaking space	US	Unit separator	DLE	Data link escape
LF	Line feed	ETB	End of transmission block	SYN	Synchronous idle	VT	Vertical tab

Next Time

- Basic Architecture of Intel Pentium Chip
- “Hello World” in Linux Assembly
- Addressing Modes