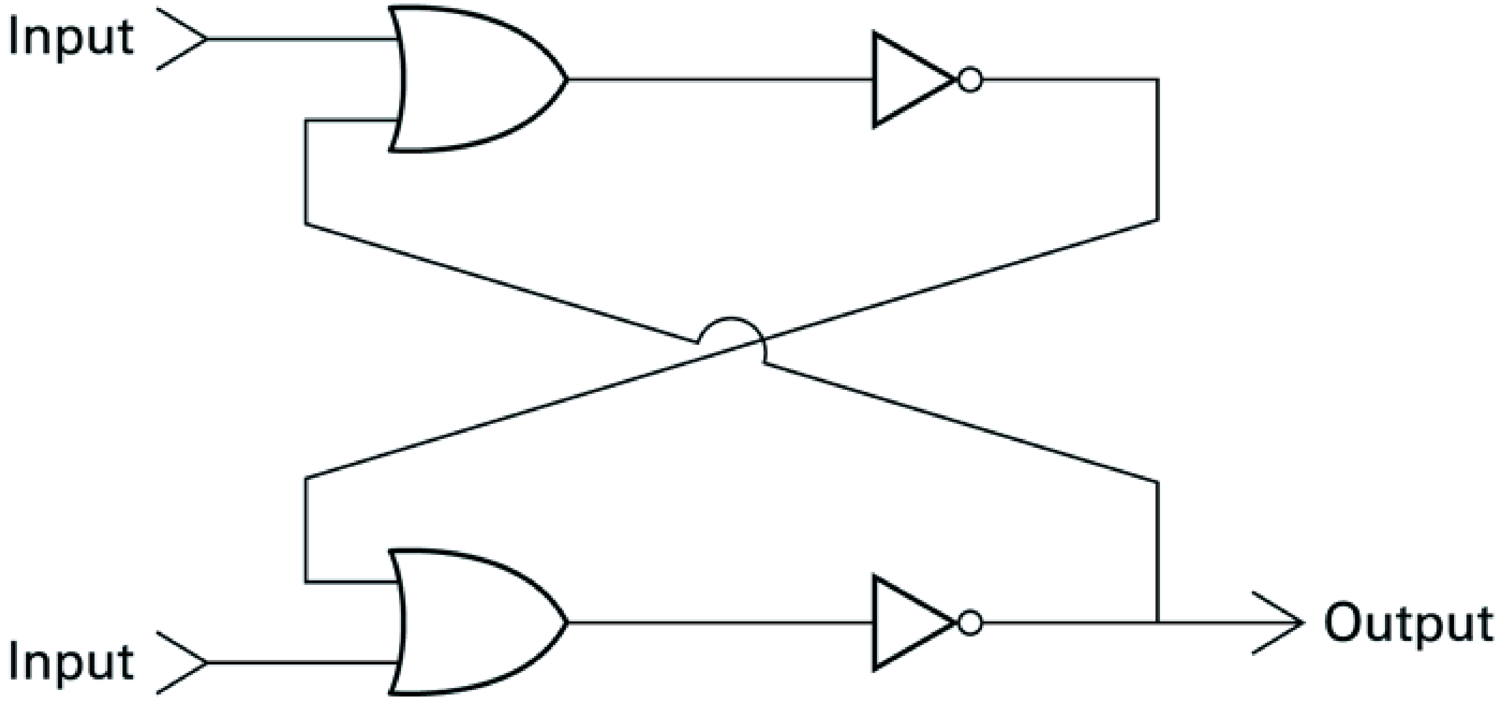
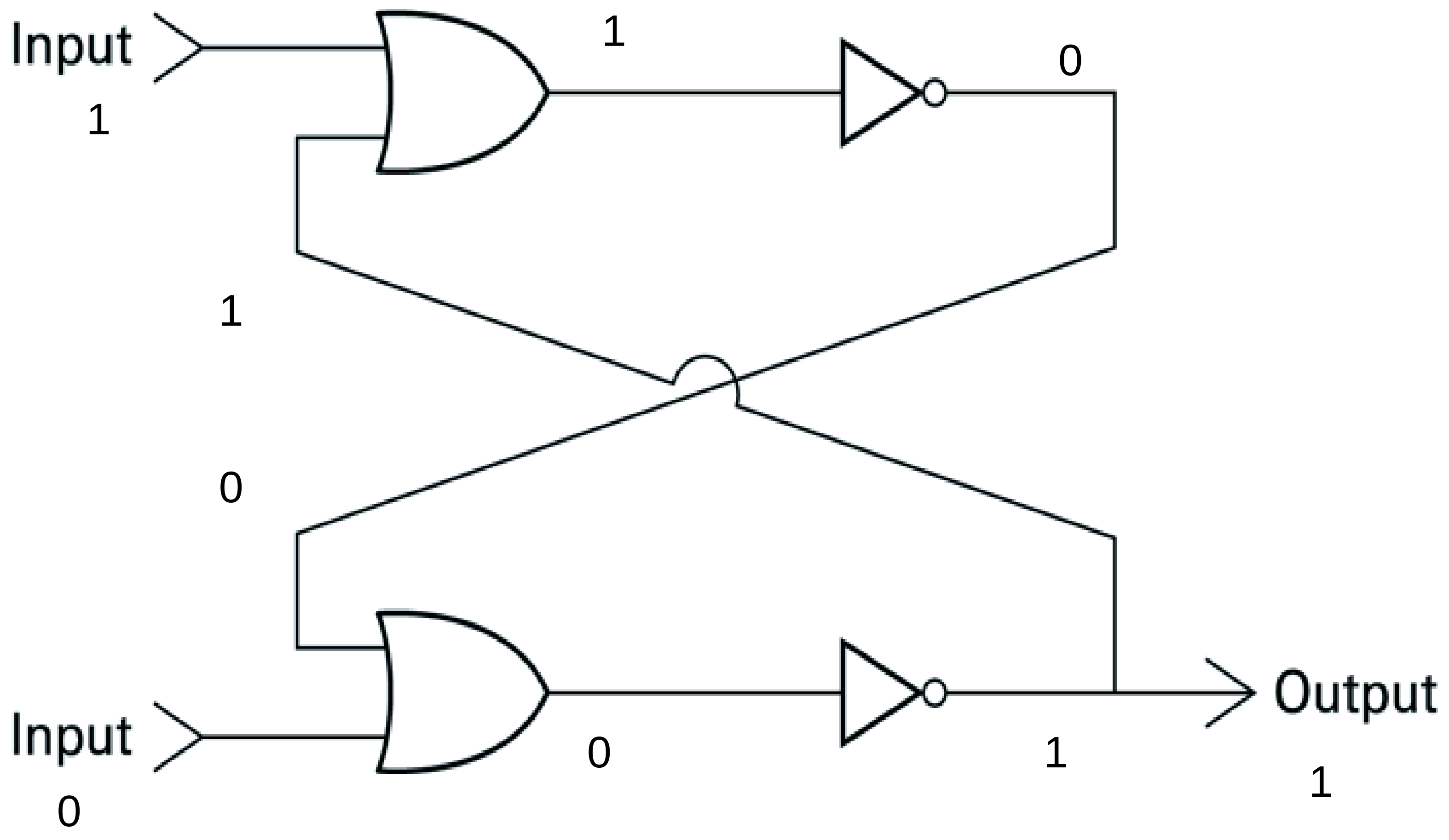
Another way of constructing Flip-Flop

Assignment: **Write outputs & sequence of steps on the basis of following inputs?**

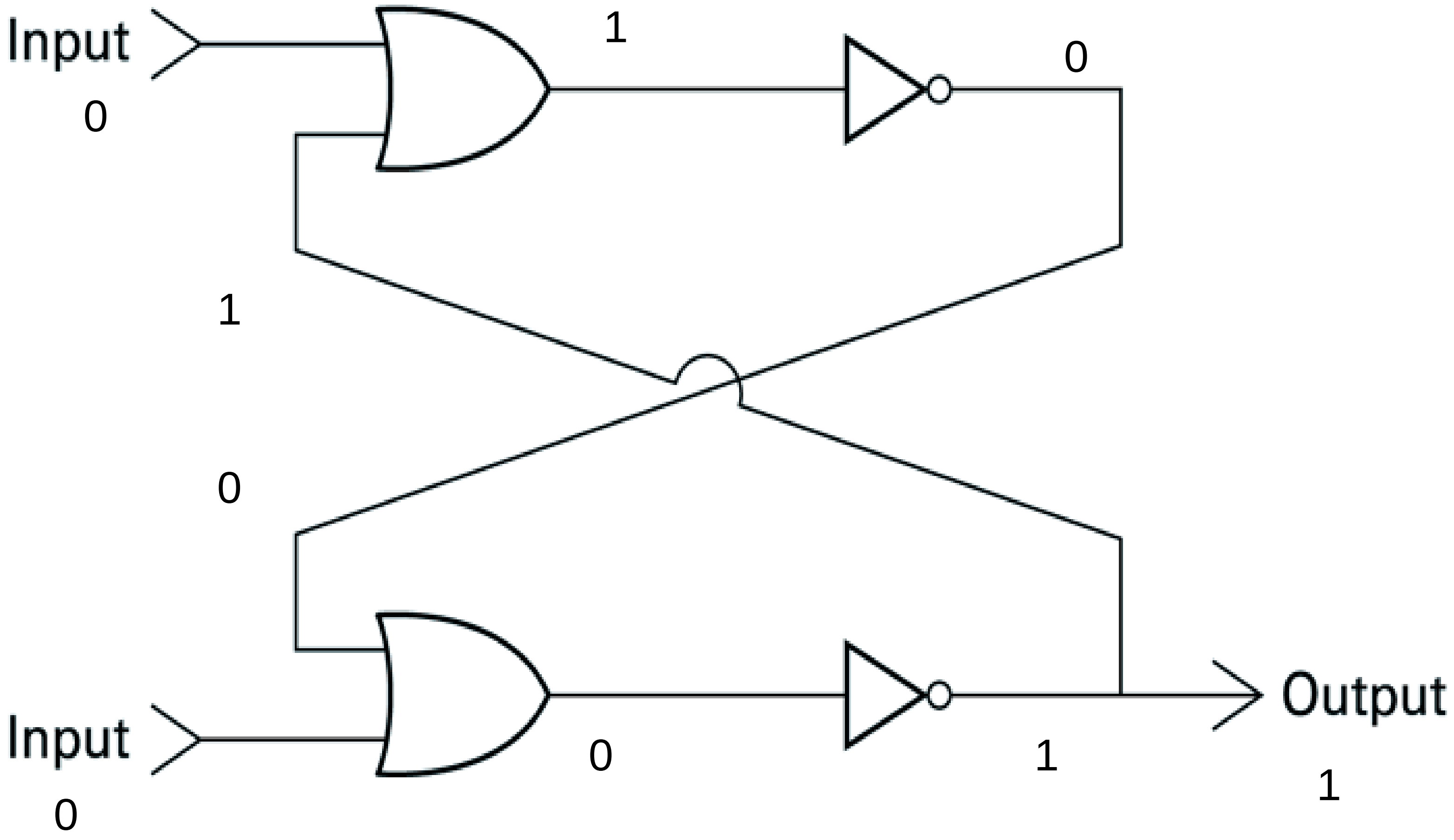
- **upper input = 1 and lower input = 0**
- **upper input = 0 and lower input = 0**



Solution:



Solution:

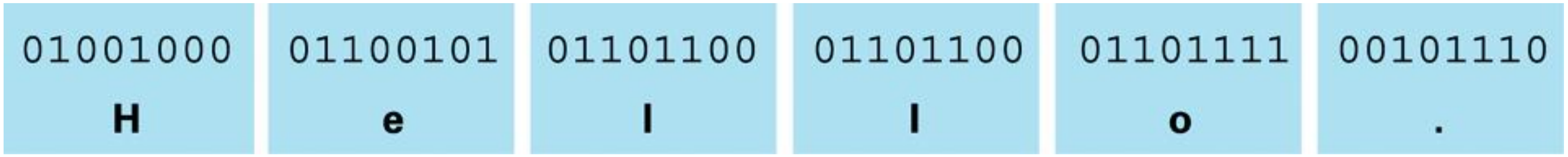


1.4 Representing Information as Bit Patterns

- Now that we know how to store single bits, we can consider how *information* can be encoded as *bit patterns*
- Different encoding systems exist for different types of information
  - numbers, text, images, sound, …
- Encoding systems more and more standardized
  - American National Standards Institute (ANSI)
  - International Organization for Standardization (ISO)

1.4 Representing Text

- Each symbol represented by a unique bit *pattern*
- Text represented by long *stream of patterns*
- Today’s standard coding system:
  - ASCII (American Standard Code for Information Interchange)
  - Bit patterns of length 7 (generally extended by 1 bit)
  - See ASCII-table in Appendix A.



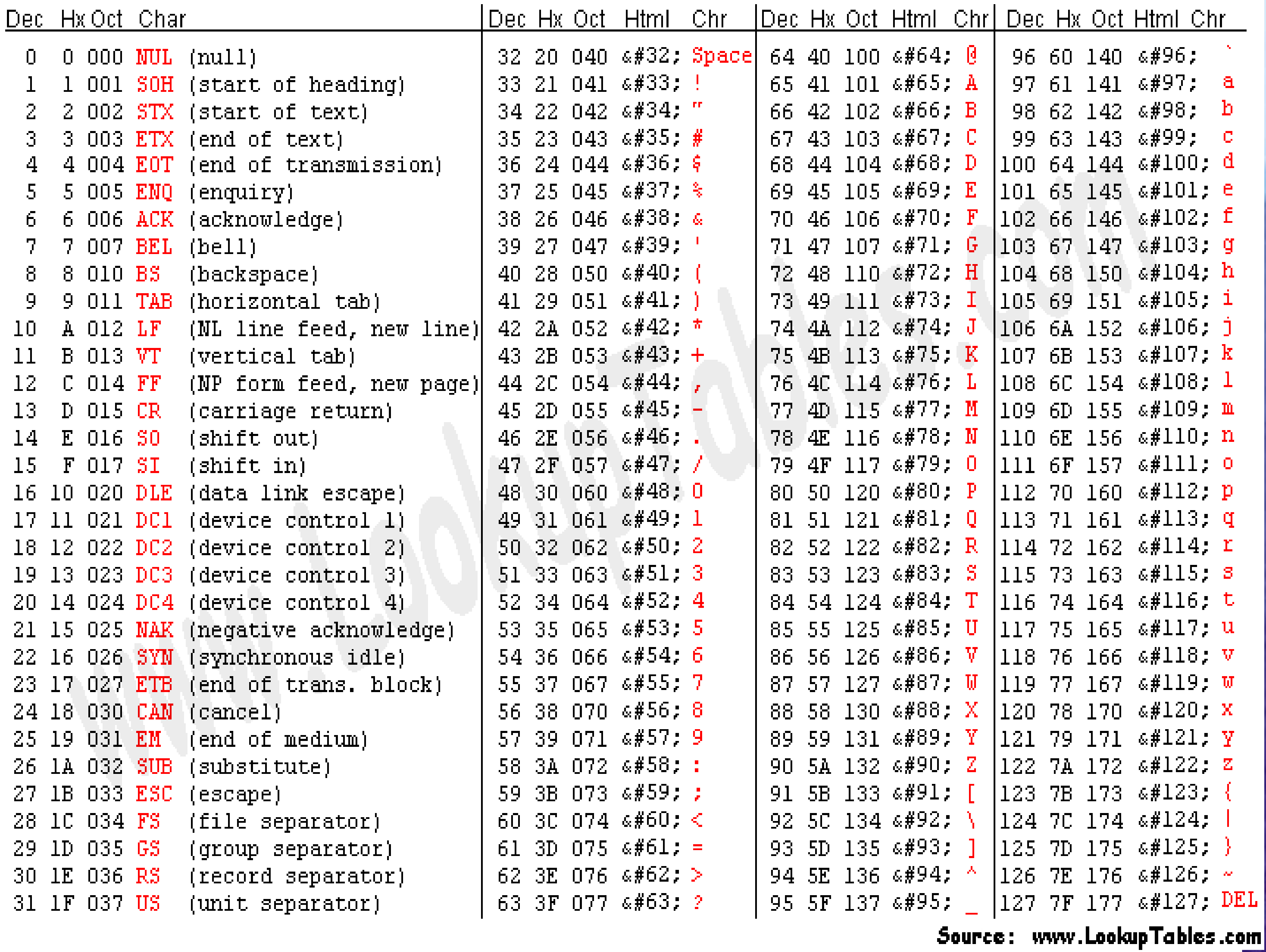
01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

ASCII

- ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose. Below is the ASCII character table and this includes descriptions of the first 32 non-printing characters.

ASCII

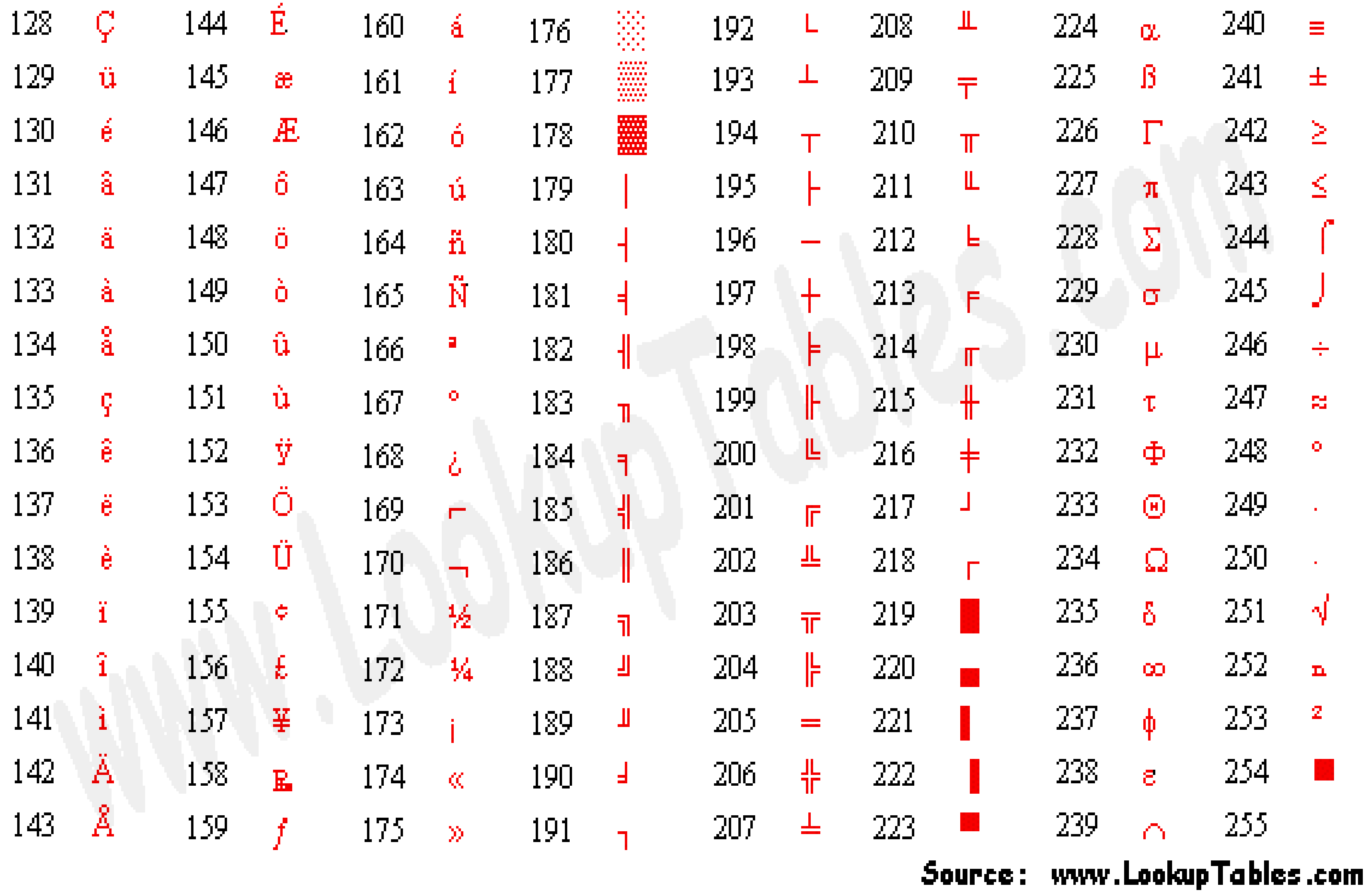
- ASCII was actually designed for use with teletypes and so the descriptions are somewhat obscure. If someone says they want your CV however in ASCII format, all this means is they want 'plain' text with no formatting such as tabs, bold or underscoring - the raw format that any computer can understand. This is usually so they can easily import the file into their own applications without issues. Notepad.exe creates ASCII text, or in MS Word you can save a file as 'text only



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Extended ASCII Codes



128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

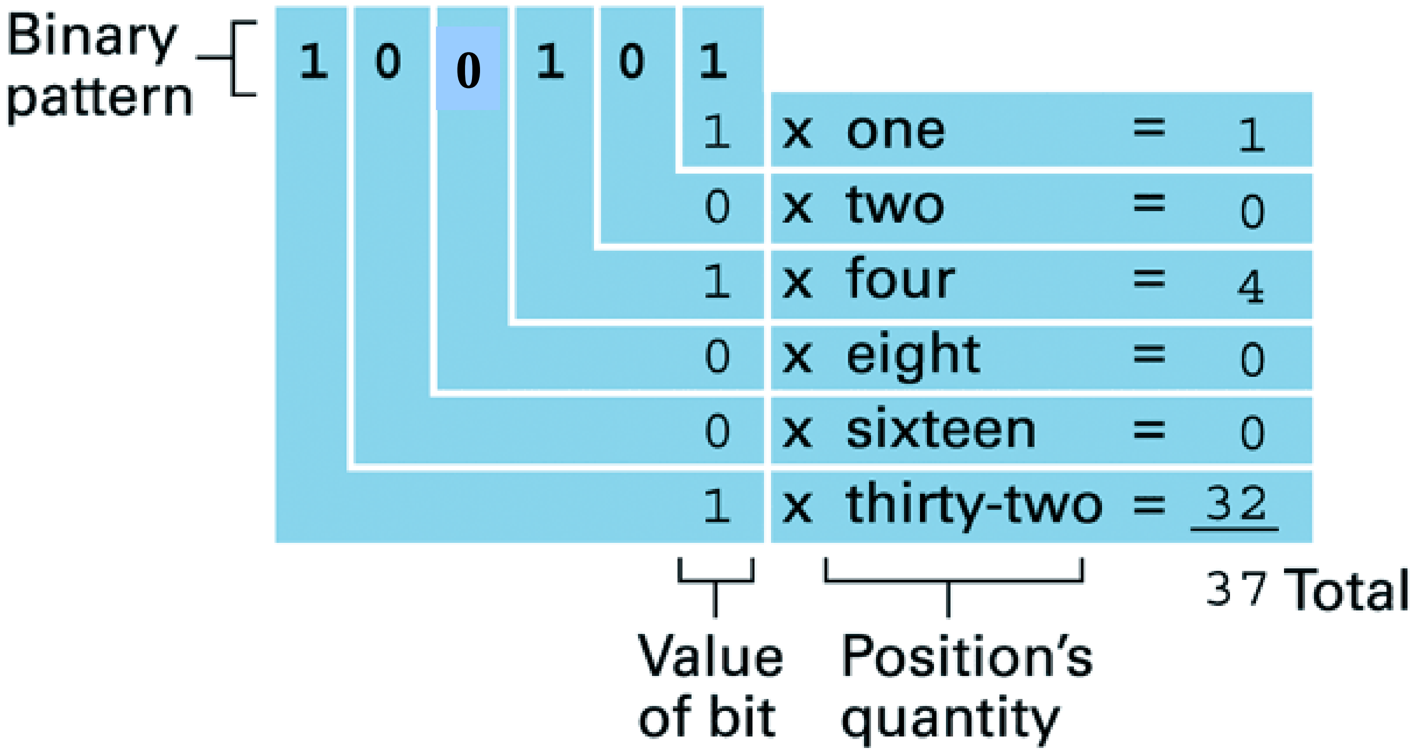
Source: www.LookupTables.com

Link

1.4 Representing Numbers

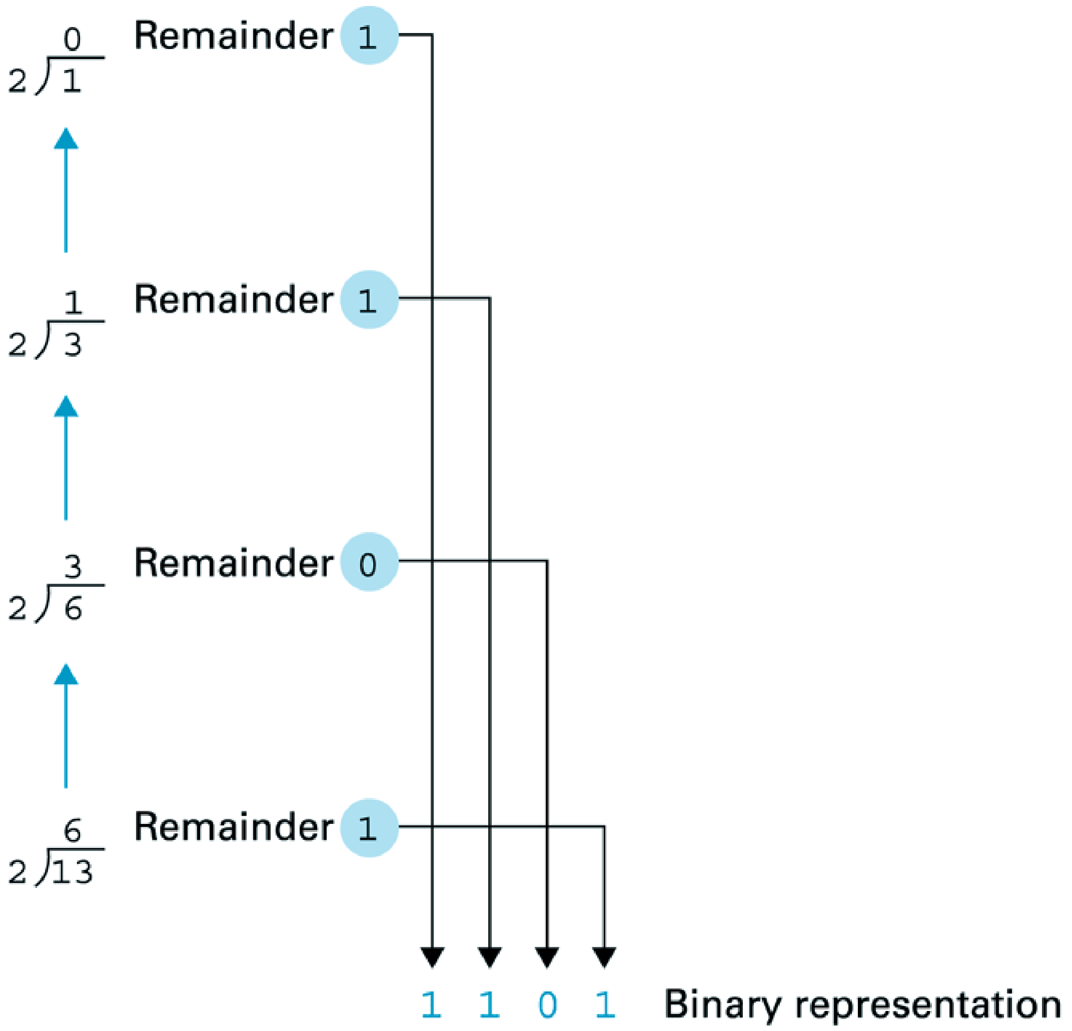
- ASCII-encoding inefficient for numeric values
- Consider storing the value 25:
  - In ASCII: 00110010 00110101 (16 bits)
  - Worse: largest 16-bit number would be 99
- More efficient approach is to use *binary system*
  - uses digits 0 and 1, incl. factor 2 for all bit-positions
- Compare decimal system
  - uses digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, incl. factor 10 for each decimal position

1.4 Decoding the Binary Representation 100101



– $1×2^5 + 0×2^4 + 0×2^3 + 1×2^2 + 0×2^1 + 1×2^0 = 37$

1.4 Obtaining the binary representation of 13



$2\overline{)1}$ quotient 0 — Remainder 1
$2\overline{)3}$ quotient 1 — Remainder 1
$2\overline{)6}$ quotient 3 — Remainder 0
$2\overline{)13}$ quotient 6 — Remainder 1
1 1 0 1 Binary representation

1.5 The Binary System: Addition

- Knowing how numeric values are encoded, we can consider how to do calculations
- Binary addition:

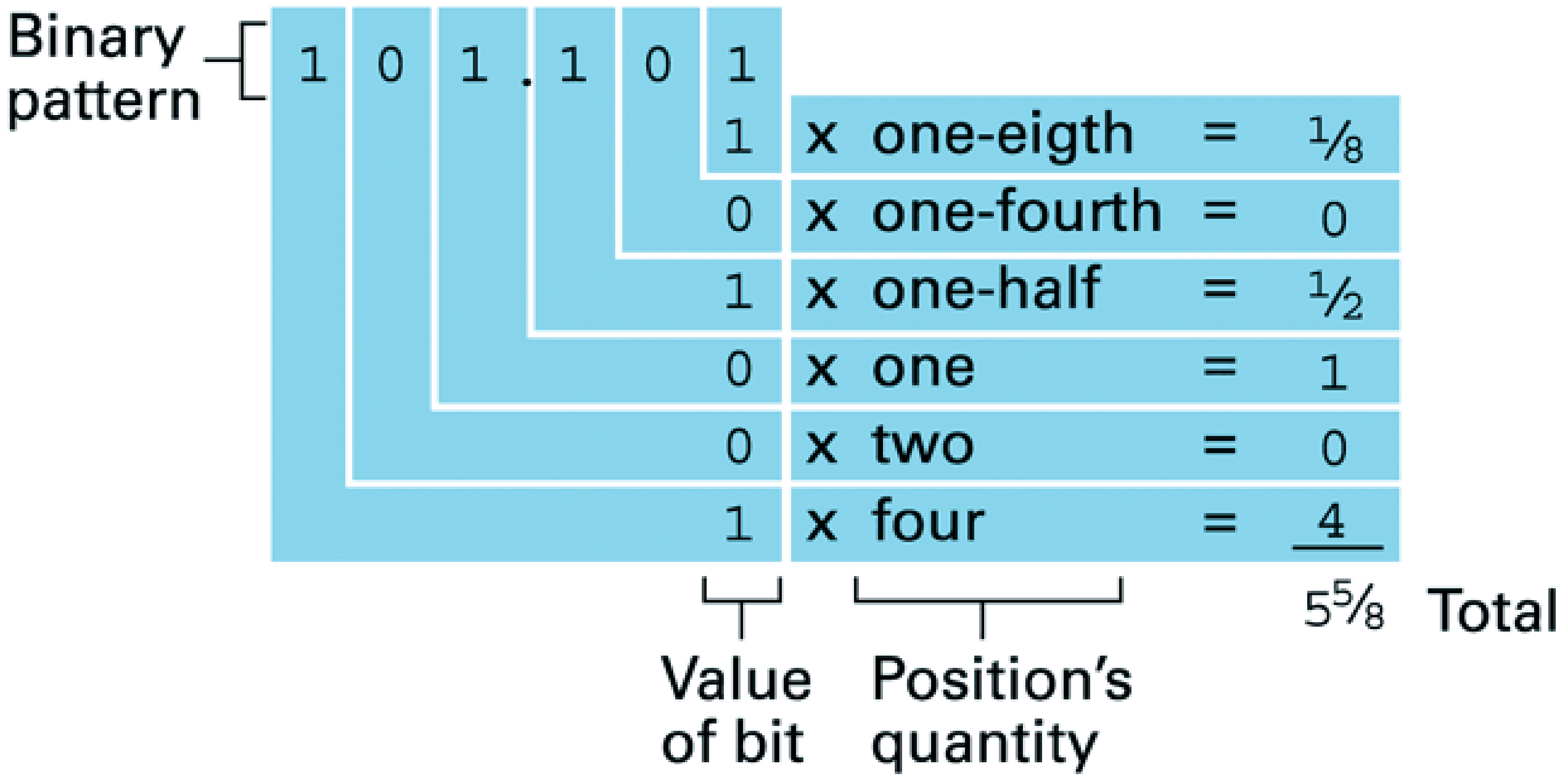
$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$

- Example:

$$\begin{array}{r} 0\,0\,1\,1\,1\,0\,1\,0 \\ +\;\;0\,0\,0\,1\,1\,0\,1\,1 \\ \hline 0\,1\;0\,1\;0\,1\;0\,1 \end{array} \qquad (58 + 27 = 85)$$

1.5 Fractions in the Binary System

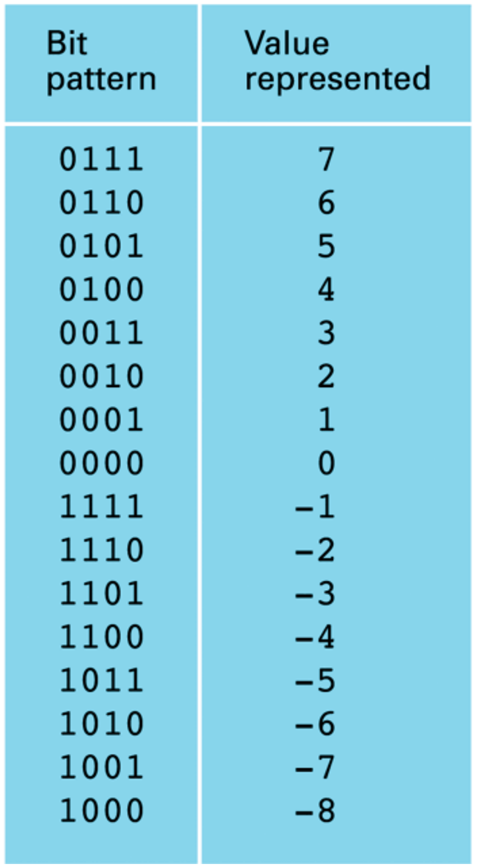
- Radix point has same role as in decimal system



1.6 Storing Integers: Two’s Complement Notation

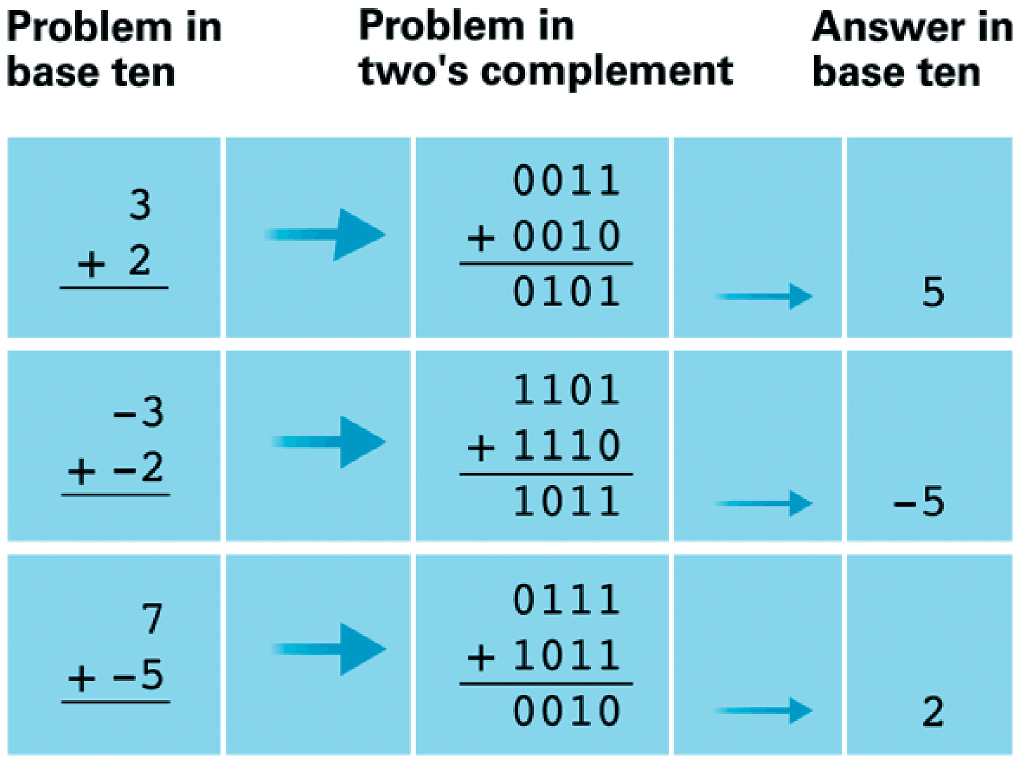
- In general: values of 32 bits
- Includes negative numbers
- Leftmost bit indicates the sign
  - *sign bit*
- Note:
  - Positive and negative numbers are identical from right to left up to & including first ‘1’; from there on are complements of one another

b. Using patterns of length four



Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

1.6 Addition in two's complement notation



Problem in base ten		Problem in two's complement		Answer in base ten
$\begin{array}{r} 3 \\ +\ 2 \\ \hline \end{array}$	→	$\begin{array}{r} 0011 \\ +\ 0010 \\ \hline 0101 \end{array}$	→	5
$\begin{array}{r} -3 \\ +\ -2 \\ \hline \end{array}$	→	$\begin{array}{r} 1101 \\ +\ 1110 \\ \hline 1011 \end{array}$	→	−5
$\begin{array}{r} 7 \\ +\ -5 \\ \hline \end{array}$	→	$\begin{array}{r} 0111 \\ +\ 1011 \\ \hline 0010 \end{array}$	→	2

- Note: no circuitry for subtraction needed!
- Note: *overflow* errors: 0101 + 0100 = 1001 (5 + 4 = -7)

Chapter 1: Problem 6

How many cells can be in a computer's main memory if each cell's address can be represented by 3 hexadecimal digits?

- Three digits:
  - 3 positions, each of which can be one of 16 values (from the range: 0, 1, …, 9, A, B, C, D, E, F)
  - smallest: 000 = $0 \times 16^2 + 0 \times 16^1 + 0 \times 16^0 = 0$
  - largest: FFF = $15 \times 16^2 + 15 \times 16^1 + 15 \times 16^0 = 4095$
  - So, total number of unique addresses = $16^3 = 4096$

Chapter 1: Problem 23

Here's a message in ASCII. What does it say?

```
01010111  01101000  01100001  01110100  00100000  01100100
01101111  01100101  01110011  00100000  01101001  01110100
00100000  01110011  00110001  01111001  00111111
```

- Each block of 8 bits represents one character:
  - See ASCII table in Appendix A
  - Example: 01010111 = ‘W’
  - Message says: ‘What does it s1y?’
  - Note: 00110001 = ‘1’, while 01100001 = ‘a’…

Chapter 1: Problem 28

a. Write the number 14 by representing the 1 and 4 in ASCII.

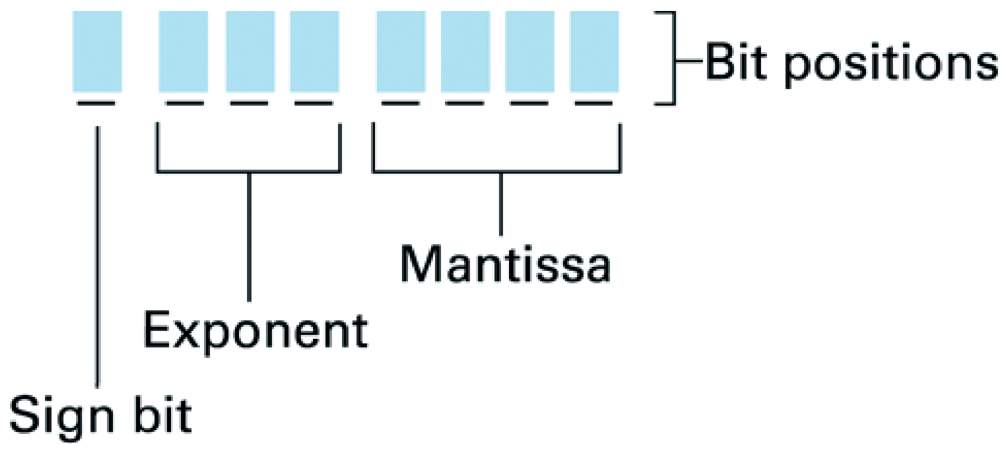
b. Write the number 14 in binary representation.

- a. See ASCII Table in Appendix A:
  - 14 = 00110001 00110100

- b. In binary system each ‘1’ represents a power of 2:
  - 14 = 8 + 4 + 2 = $\mathbf{1}\times 2^3 + \mathbf{1}\times 2^2 + \mathbf{1}\times 2^1 + \mathbf{0}\times 2^0$ => binary: 1110

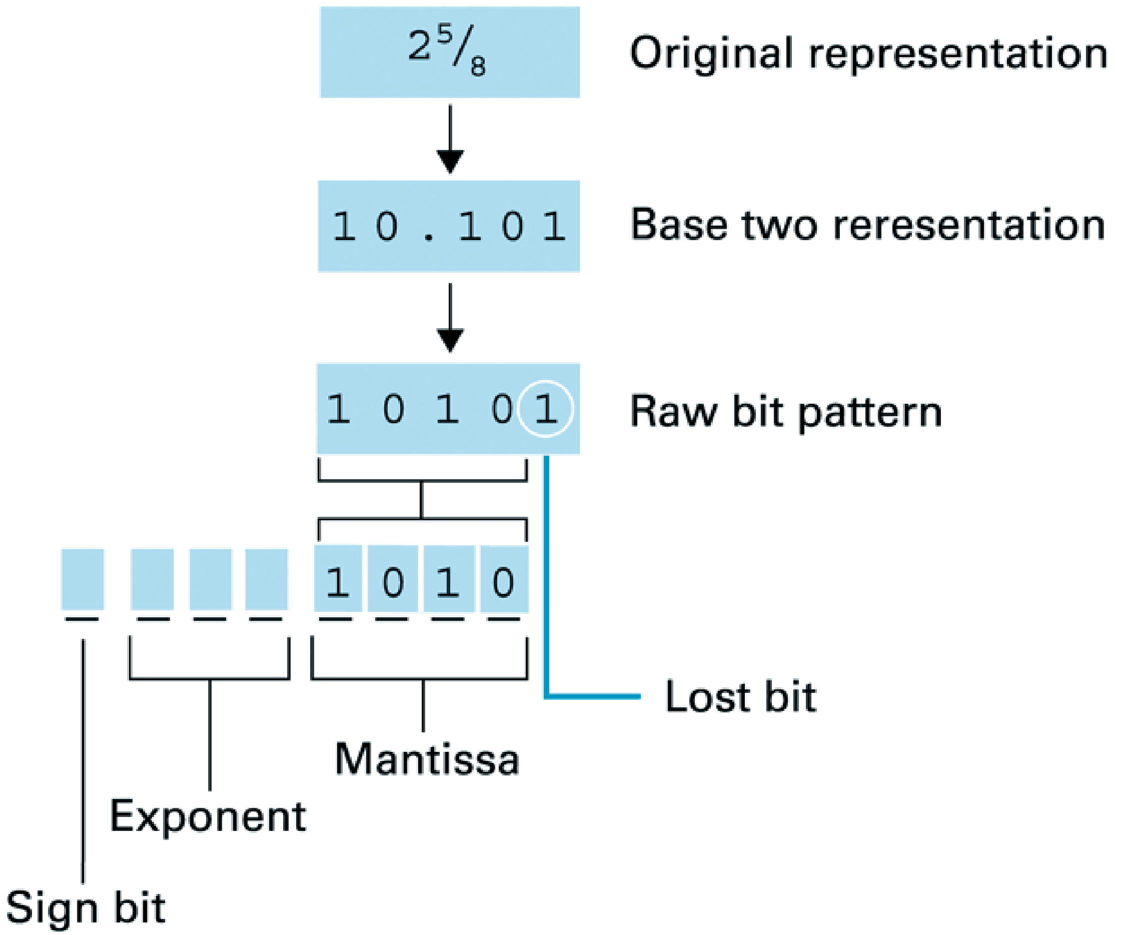
1.7 Storing Fractions: Floating-point Notation

- In contrast to integers, fractions require storage of the radix point
  - *Floating-point* notation



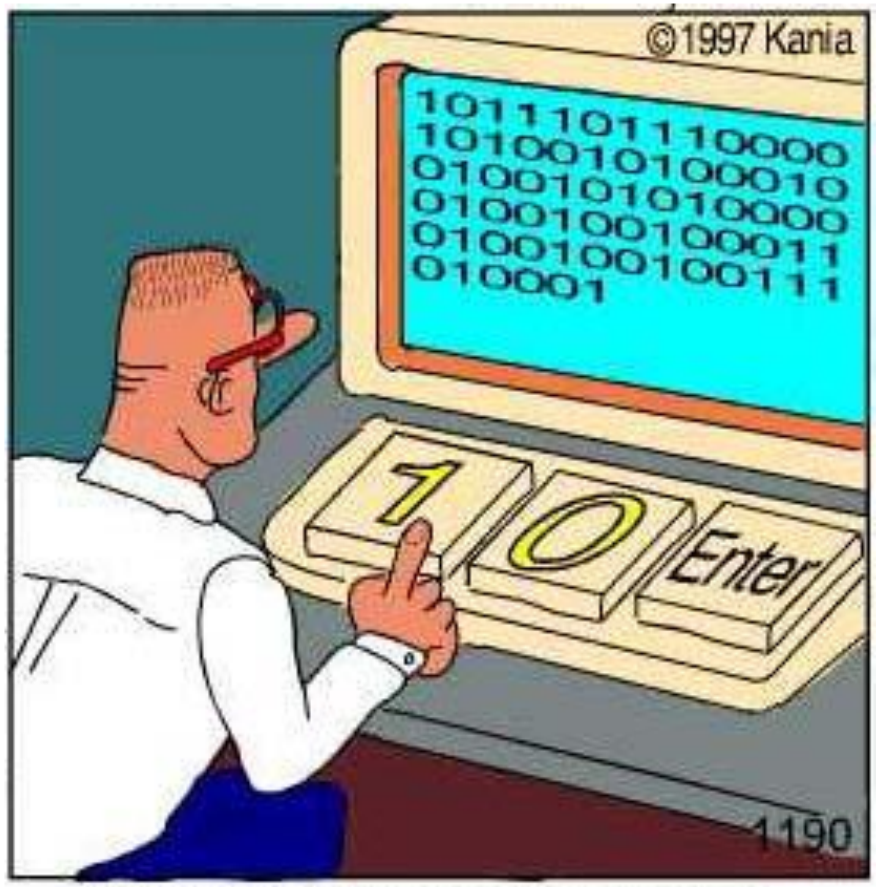
- Example: 1 110 1011 = -10.11 = -2.75

1.7 Truncation Errors: Coding the value 2 $^{5/8}$



1.7 Truncation Errors (cont'd)

- Significance of truncation errors reduced by using larger mantissa & exponent fields (32bits)
- Problem of nonterminating expansion (e.g. 1/3)
  - worse in binary than in decimal system (e.g. 1/10)
- Interesting:
  - 2 1/2 + 1/8 + 1/8 = 2 1/2
  - 1/8 + 1/8 + 2 1/2 = 2 3/4
- When adding numbers, order may be important
  - rule: add smaller values first!



Real programmers code in binary.

Chapter 1: Conclusions

- Information stored as *streams of bits*
- Bit streams stored in main memory or on mass storage devices - each with different degree of random access (and thus: speed)
- Meaning of bit streams application dependent
- Standardized representations exist for (a.o):
  - text, numeric values, images, sounds, …
- For numeric values: overflow and truncation errors may make life difficult sometimes...