MANAGEMENT AND ANALYSIS OF PHYSICS DATASET (MOD. A)

Number Systems
Arithmetic Operations

Binary numbers

- Computers doesn’t work with decimal numbers
  - Binary numbers are more reliable -> two-states discrimination
- Decimal number system uses ten *digits* (from 0 to 9)
  - Reset-and-carry
    - The units reset to zero and carry to the tens…
- Binary number system uses two *digits* (0 and 1)
  - Still Reset-and-carry

Binary numbers

- Computers doesn’t work with decimal numbers
    - Binary numbers are more reliable -> two-states discrimination
- Decimal number system uses ten *digits* (from 0 to 9)
    - Reset-and-carry
        - The units reset to zero and carry to the tens…
- Binary number system uses two *digits* (0 and 1)
    - Still Reset-and-carry
- Base or radix of a number system is the number of digits
- *Bit* stands for binary digit
- Binary is less compact than decimal

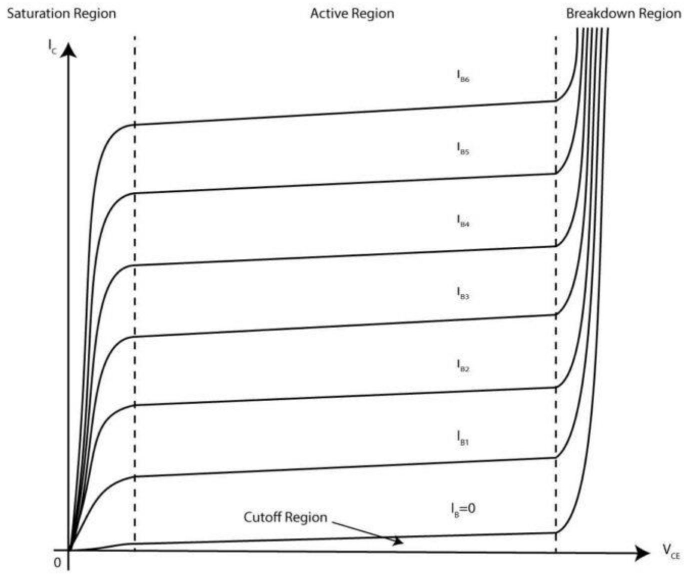
0	0	0	0	0	0
0	1	0	0	0	1
0	2	0	0	1	0
0	3	0	0	1	1
0	4	0	1	0	0
0	5	0	1	0	1
0	6	0	1	1	0
0	7	0	1	1	1
0	8	1	0	0	0
0	9	1	0	0	1
1	0	1	0	1	0
1	1	1	0	1	1
1	2	1	1	0	0
1	3	1	1	0	1
1	4	1	1	1	0
1	5	1	1	1	1

Binary numbers

- Computers doesn’t work with decimal numbers
  - Binary numbers are more reliable -> two-states discrimination
- Decimal number system uses ten *digits* (from 0 to 9)
  - Reset-and-carry
    - The units reset to zero and carry to the tens…
- Binary number system uses two *digits* (0 and 1)
  - Still Reset-and-carry
- Base or radix of a number system is the number of digits
- *Bit* stands for binary digit
- Binary is less compact than decimal
- They are equivalent (ten pebbles are 10_{10} pebbles or 1010_2 pebbles)

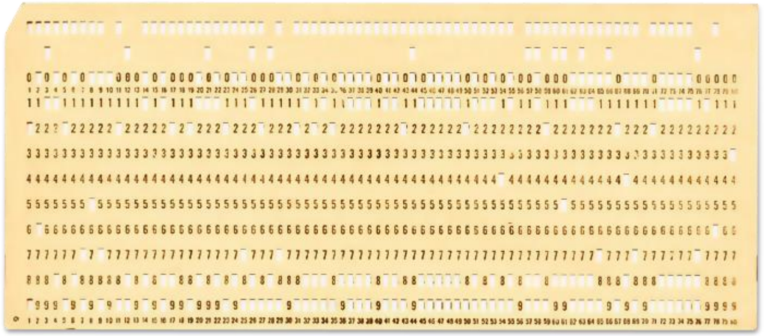
Binary numbers

- Computer can store Program and Data only in a binary form
- Transistors have parameters that can vary of more than 50% with temperature and among them for manufacturing differences
- Two-state design uses only two working points
  - For instance, cutoff and saturation



Binary numbers

- Computer can store Program and Data only in a binary form
- Transistors have parameters that can vary of more than 50% with temperature and among them for manufacturing differences
- Two-state design uses only two working points
  - For instance, cutoff and saturation
- Punched cards are another example of Program and Data storing



Binary numbers

- 4 bit are a *nibble*
- 8 bit are a *byte*
- 16 bit are a *halfword*
- 32 bit are a *(single)word*
- 64 bit are a *doubleword*
- 128 bit are a *quadword*
- A *string* is a group of character (either letters or digits) written one after the other
- *Word* can also refer to a string of bits of a specific length (i.e., a 24-bit word)
- The abbreviation *K* stands for 1024 (powers of two!)
  - 64K is 65536
- *b* is the abbreviation of bit, *B* of byte
  - 1 MB = 1048576 B = 8388608 b

Binary-to/from-decimal

- Both systems use positional notation
  - Decimal -> power of ten
  - Binary -> power of two
- $38572 = 3 \cdot 10^4 + 8 \cdot 10^3 + 5 \cdot 10^2 + 7 \cdot 10^1 + 2 \cdot 10^0$
- $10110100 = 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$
- Double-dabble algorithm

19		
9	1	
4	1	
2	0	
1	0	
0	1	-> 10011

Hexadecimal numbers

- Number system with base 16
  - Digits are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- Shorter representation than binary
- Easy to convert
  - Nibble by nibble
  - A3C = 101000111100
- It uses positional notation as well
  - $A3C = A \cdot 16^2 + 3 \cdot 16^1 + C \cdot 16^0 = 2620_{10}$
- Hex-dabble algorithm

  2620
  163 C
  10 3
  0 A -> A3C

0	0	0	0	0	0	0
0	1	0	0	0	1	1
0	2	0	0	1	0	2
0	3	0	0	1	1	3
0	4	0	1	0	0	4
0	5	0	1	0	1	5
0	6	0	1	1	0	6
0	7	0	1	1	1	7
0	8	1	0	0	0	8
0	9	1	0	0	1	9
1	0	1	0	1	0	A
1	1	1	0	1	1	B
1	2	1	1	0	0	C
1	3	1	1	0	1	D
1	4	1	1	1	0	E
1	5	1	1	1	1	F

Exercises

- Convert 82_{10} in binary
- Convert $10A4_{16}$ in binary
- Convert 650_{10} in hex

Arithmetic Operations

Binary addition

- Simple additions
  - $0 + 0 = 0$
  - $0 + 1 = 1$
  - $1 + 0 = 1$
  - $1 + 1 = 10$ (zero, carry one)
  - $1 + 1 + 1 = 11$ (one, carry one)
- Larger numbers

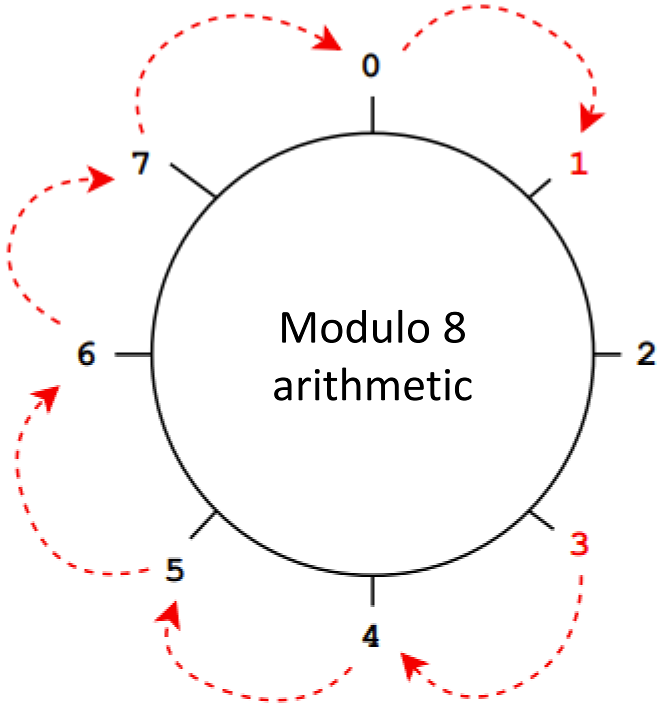
$$
\begin{array}{r}
11100\ + \\
11010\ = \\
110110
\end{array}
$$

Binary subtraction

- Simple subtractions
  - $0 - 0 = 0$
  - $1 - 0 = 1$
  - $1 - 1 = 0$
  - $10 - 1 = 1$
- Larger numbers
  $1101 -$
  $1010 =$
  0011

Modular arithmetic

- When the set of number is finite, we use modular arithmetic
- Numbers “wrap around”
- $a \equiv b \ (mod\ n) \rightarrow a = kn + b$
  - $25\ mod\ 8 = 1$
  - $24\ mod\ 8 = 0$
  - $17\ mod\ 3 = 2$
- Examples of operations on $3\ bit\ (mod\ 2^3)$
  - $3 + 2 = 5$
  - $3 + 6 = 1$
  - $7 - 5 = 2$
  - $2 - 5 = 5$



$3 + 6 = 1$

Signed binary numbers

- Simply setting one as MSB (MSB as sign)
  - For instance, in an 8-bit representation: 9 -> 00001001   -9 -> 10001001
  - +0 and -0
  - $9 + (-9) \neq +0$ $9 + (-9) \neq -0$
- Inverting all the bits (one's complement)
  - For instance, in an 8-bit representation: 9 -> 00001001   -9 -> 11110110
  - +0 and -0
  - 9 + (-9) = -0
- Inverting all the bits and adding one (two's complement)
  - For instance, in an 8-bit representation: 9 -> 00001001   -9 -> 11110111
  - Only one 0
  - 9 + (-9) = 0   (plus a carry)
  - In general, all the sums work: 9 + (-3) = 6   00001001 + 11111101 = 100000110

2’s complement operations

- Given $k\ bit$, we can represent the numbers in the interval $[-2^{k-1}, 2^{k-1}-1] \subset \mathbb{Z}$
- The most significant bit tells us if the number is positive or negative
- For instance, 8 bit -> 256 numbers from -128 to 127
  - -2+5 -> 11111110+
    00000101=
    100000011
  - 5-3 -> 00000101+
    11111100=
    100000010

10000000	-128
…	…
11111110	-2
11111111	-1
00000000	0
00000001	1
00000010	2
…	…
01111111	127

Binary multiplication

- Same as decimal multiplication
- For instance

```
     1101 ·
     1011=
     1101
    1101
   0000
  1101
 10001111
```

- If M has n-bit and Q has m-bit, $M \cdot Q$ has n+m bit

Binary division

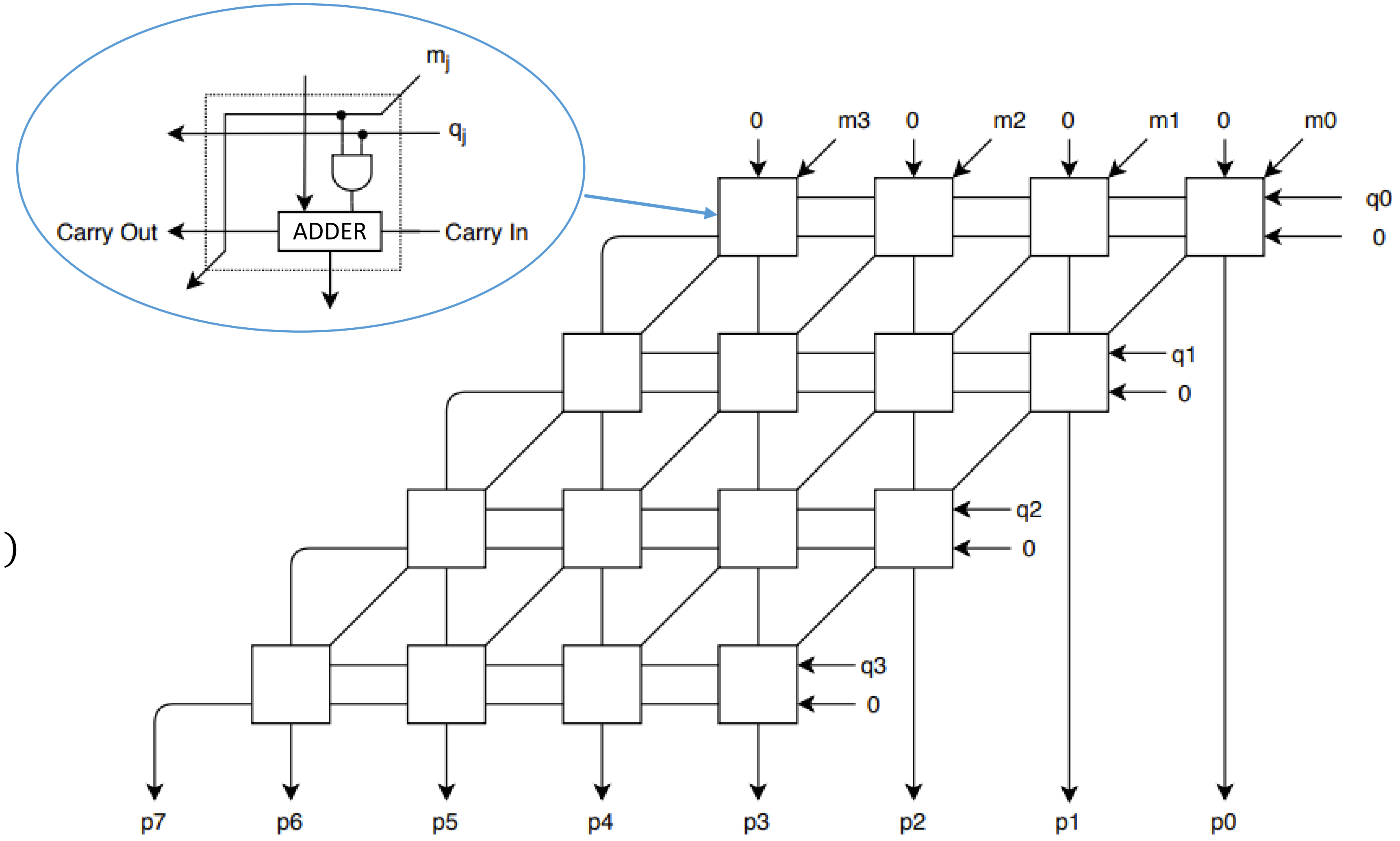
- Same as decimal division
- For instance

```
100010010    /  1101
  1101           10101
    10000
     1101
         1110
         1101
            1
```

- If M has n-bit and P has m-bit, M/P has n-m bit

Parallel multiplier

$P = M \cdot Q$
$M = (m_3, m_2, m_1, m_0)$
$Q = (q_3, q_2, q_1, q_0)$
$P = (p_7, p_6, p_5, p_4, p_3, p_2, p_1, p_0)$



Exercises

- Calculate the binary sum 11100+10011
- Calculate the binary subtraction 100101-11011
- Calculate the binary multiplication 1011·1001
- Calculate the binary division 1111÷110
- Calculate 10011-11100 using 2's complement method

Bit manipulation

Bitwise operations

- Operators that work on a bit array at the level of its individual bit
  - NOT (complement)
    - NOT 10010101 = 01101010
  - AND
    - 0111 AND 1010 = 0010
  - OR
    - 0011 OR 0100 = 0111
  - XOR
    - 0010 XOR 1010 = 1000
- Quite often a single register has several bits with different functionalities
- To manipulate the single bit, we do bit-masking

Bitwise operations

Register 135. Reset/Freeze/Memory Control

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Name	RST_REG	NewFreq	Freeze M	Freeze VCADC		N/A		RECALL
Type	R/W	R/W	R/W	R/W		R/W		R/W

Bit	Name	Function
7	RST_REG	**Internal Reset.** 0 = Normal operation. 1 = Reset of all internal logic. Output tristated during reset. Upon completion of internal logic reset, RST_REG is internally reset to zero. **Note:** Asserting RST_REG will interrupt the I^2C state machine. It is not the recommended approach for starting from initial conditions.
6	NewFreq	**New Frequency Applied.** Alerts the DSPLL that a new frequency configuration has been applied. This bit will clear itself when the new frequency is applied.
5	Freeze M	**Freezes the M Control Word.** Prevents interim frequency changes when writing RFREQ registers.
4	Freeze VCADC	**Freezes the VC ADC Output Word.** May be used to hold the nominal output frequency of an Si571.
3:1	N/A	Always Zero.
0	RECALL	**Recall NVM into RAM.** 0 = No operation. 1 = Write NVM bits into RAM. Bit is internally reset following completion of operation. **Note:** Asserting RECALL reloads the NVM contents in to the operating registers without interrupting the I^2C state machine. It is the recommended approach for starting from initial conditions.

Bit-masking

- For setting a bit to 1, we do an OR with a mask done by all 0s except the bit we want to set
  - Set bit 3 of register R0

R0	B7	B6	B5	B4	B3	B2	B1	B0	**OR**
Mask	0	0	0	0	1	0	0	0	**=**
R0	B7	B6	B5	B4	1	B2	B1	B0	

  - R0 OR 0x08

Bit-masking

- For clearing a bit to 0, we do an AND with a mask done by all 1s except the bit we want to clear
  - Clear bit 3 of register R0

R0	B7	B6	B5	B4	B3	B2	B1	B0	**AND**
Mask	1	1	1	1	0	1	1	1	**=**
R0	B7	B6	B5	B4	0	B2	B1	B0	

  - R0 AND 0xF7

Bit-masking

- For inverting a bit, we do a XOR with a mask done by all 0s except the bit we want to invert
  - Invert bit 3 of register R0

R0	B7	B6	B5	B4	B3	B2	B1	B0	**XOR**
Mask	0	0	0	0	1	0	0	0	**=**
R0	B7	B6	B5	B4	$\overline{B3}$	B2	B1	B0	

  - R0 XOR 0x08

Bit-masking

- For testing a bit, we do an AND with a mask done by all 0s except the bit we want to test and then we check if the result is equal to 0
  - Test bit 3 of register R0

R0	B7	B6	B5	B4	B3	B2	B1	B0	**AND**
Mask	0	0	0	0	1	0	0	0	**=**
R0	0	0	0	0	B3	0	0	0	

  - R0 AND 0x08 $\stackrel{?}{=}$ 0x00

Exercises

- Check if the MSB of a 32-bit register is set
- Get the third bit (starting from the LSB) of an 8-bit register

Real numbers

Real numbers representation

- As for $\mathbb{Z}$, with a finite number of bit we can represent only a subset of $\mathbb{R}$
- Any number belonging to $\mathbb{R}$ must be approximated
- Two main techniques
  - Fixed point
  - Floating point

Fixed point

- Given k bit, we set a number of bit $r < k$ for the fractional part
- The weight of each bit of the fractional part is given by $2^{-1}, 2^{-2}, \ldots$
- The integer part will have $k - r$ bit
- For instance, if $k = 8$ and $r = 4$ the number 00110110 means:

$$\begin{aligned} &00110110 = 0011.0110 \\ &= 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} \\ &= 2 + 1 + 0.25 + 0.125 = 3.375 \end{aligned}$$

- Real to binary conversion algorithm
  - Usual algorithm for the integer part
  - Fractional part multiplied by two, extraction of the integer part and so on
  - For instance, if $k = 8$ and $r = 4$ the number 3.375 becomes:

$$\begin{array}{llll} & 3 \to 0011 & & \searrow \\ 3.375 & & & \quad 0011.0110 \\ & 0.375 \to 0.750 & \to 0 & \nearrow \\ & \quad 1.5 & \to 1 & \\ & \quad 1 & \to 1 & \\ & \quad 0 & \to 0 & \end{array}$$

Floating point

- Fixed point represents a too small subset of $\mathbb{R}$
- For being more flexible floating point uses a *significand* or *mantissa* and an *exponent*
- A typical example is IEEE 754 the `float` of C/C++

  1 bit for sign S

  8 bit for exponent E

  23 bit for mantissa M

  $(-1)^{S} \cdot 1.M \cdot 2^{E-127}$
- For instance

  $BFA00000_{16} = 10111111101000000000000000000000$

  $= (-1)^{1} \cdot 1.01_{2} \cdot 2^{127-127} = -1.25$

Sum of floating points

- $M_1 \cdot 2^{E_1} + M_2 \cdot 2^{E_2}$
- Let's assume $E_1 < E_2$, we get M_1' as a right shift of $M_1$of $E_2 - E_1$ positions
- The mantissa of the sum will be $M_s = M_1' + M_2$
- So, the sum will be $M_S \cdot 2^{E_2}$
- Finally, the result is normalized to the form $1.M \cdot 2^E$

$$1.0001 \cdot 2^2 + 1.1101 \cdot 2^4 =$$
$$0.010001 \cdot 2^4 + 1.1101 \cdot 2^4 =$$
$$10.000101 \cdot 2^4 =$$
$$1.0000101 \cdot 2^5$$

Multiplication of floating points

- $M_1 \cdot 2^{E_1} \ \cdot \ M_2 \cdot 2^{E_2} = (M_1 \cdot M_2) \cdot 2^{E_1+E_2}$
- The result is then normalized to the form $1.M \cdot 2^E$

- $M_1 \cdot 2^{E_1} \ / \ M_2 \cdot 2^{E_2} = (M_1/M_2) \cdot 2^{E_1-E_2}$
- The result is then normalized to the form $1.M \cdot 2^E$

$1.0001 \cdot 2^2 \cdot 1.1101 \cdot 2^4 =$
$(1.0001 \cdot 1.1101) \cdot 2^{(2+4)} =$
$1.11101101 \cdot 2^6$

ASCII code

- To input and output data to/from a computer we need numbers, letters and other symbols
- *American Standard Code for Information Interchange* is a standard input/output code
- 7-bit code
- 0x48 0x65 0x6C 0x6C 0x6F -> `Hello`
- 0x48 0x65 0x6C 0x6C 0x77 0x08 0x6F -> `Hello`

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	*[NULL]*	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	*[START OF HEADING]*	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	*[START OF TEXT]*	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	*[END OF TEXT]*	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	*[END OF TRANSMISSION]*	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	*[ENQUIRY]*	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	*[ACKNOWLEDGE]*	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	*[BELL]*	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	*[BACKSPACE]*	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	*[HORIZONTAL TAB]*	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	*[LINE FEED]*	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	*[VERTICAL TAB]*	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	*[FORM FEED]*	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	*[CARRIAGE RETURN]*	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	*[SHIFT OUT]*	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	*[SHIFT IN]*	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	*[DATA LINK ESCAPE]*	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	*[DEVICE CONTROL 1]*	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	*[DEVICE CONTROL 2]*	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	*[DEVICE CONTROL 3]*	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	*[DEVICE CONTROL 4]*	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	*[NEGATIVE ACKNOWLEDGE]*	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	*[SYNCHRONOUS IDLE]*	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	*[ENG OF TRANS. BLOCK]*	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	*[CANCEL]*	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	*[END OF MEDIUM]*	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	*[SUBSTITUTE]*	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	*[ESCAPE]*	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	*[FILE SEPARATOR]*	76	4C	1001100	114	L	124	7C	1111100	174	|
29	1D	11101	35	*[GROUP SEPARATOR]*	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	*[RECORD SEPARATOR]*	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	*[UNIT SEPARATOR]*	79	4F	1001111	117	O	127	7F	1111111	177	*[DEL]*
32	20	100000	40	*[SPACE]*	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

Exercises

- Find the decimal value of the binary number 11101.011
- Calculate $23.75_{10} + 4.5_{10}$ using floating point