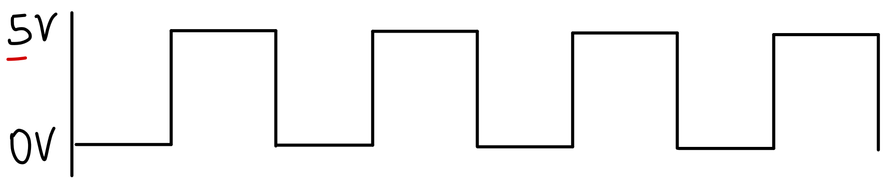
Number Systems

- Data are internally represented as sequence of bits (binary digits).

good separation of voltage



bits
1 – > 3·75V
buffer
0 – < 1·25V
⇓
also cheaper.

- Basic data type:
  - Int 70
  - float
  - double
  - char 'F'
  - bool True

  } 01000110 – different types means different ways of viewing a binary string
  ∴ declare data type!

- Bits:
  - "Binary Digits"
  - Either 0 or 1
  - Byte = 8 bits (By Eight)
  - Word = a set of byte that equals to one single unit of transfer from memory to CPU / within a CPU.
    power of 2
    i.e. 32 bit machine → Word = 4 bytes.
    64 bit machine → Word = 8 bytes
    } limits the range of data represented in one word.

- N bits ⇒ 2^N values.
- M values ⇒ $\lceil \log_2 M \rceil$ bits
  ceil so that it represents M values.
  (also ensure M is not short of one bit)

Weighted Positional Number System

Decimal ⇒ Base/radix = 10.

Symbols = $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

· Each position has a weight of power of 10:

i.e. $(7594.36)_{10}$ = digit × weight

$$
\begin{aligned}
& 7 \times 10^3 \\
+ & 5 \times 10^2 \\
+ & 9 \times 10^1 \\
+ & 4 \times 10^0 \\
\hline
+ & 3 \times 10^{-1} \\
+ & 6 \times 10^{-2}
\end{aligned}
$$

Binary ⇒ base 2 – prefix 0b.

Octal ⇒ base 8 – prefix 0 i.e. 032

Hexadecimal ⇒ base 16 – 0, 1, 2, ..., 9, A, B, C, D, E, F.

– prefix 0x.

Base-R to Decimal:

i.e. $1101.101_2 = 1\times2^3 + 1\times2^2 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-3}$

$= 13.625_{10}$

Decimal (Base-R) to Binary

For whole numbers – repeat div 2 method

fraction – repeat multiply 2 method.

i.e. $(43)_{10}$ = → $(101011)_2$.

43 / 2 = 21 r 1 ← LSB

21 / 2 = 10 r 1

⋮

.... = 0 r 1 ← MSB

$(0.3125)_{10}$ = (until desired number of decimal places)

	carry	
0.3125 × 2 = 0.625	0	MSB
0.625 × 2 = 1.25	1	
0.25 × 2 = 0.5	0	
0.5 × 2 = 1	1	LSB

$= (0.0101)_2$ //

· Base-R → Decimal → Base-S

"Bridge"

Shortcut for base – 2, 4, 8, 16 (pow of 2)

· $(1011\ 1101)_2$

grp of bits i.e. base 4

· Bin to Oct

$(10\ 111\ 011\ 001\ .\ 101\ 110)_2$

← grp · grp →

⇓

$(2\ 7\ 3\ 1\ .\ 5\ 6)_8$

· Oct to Bin ⇒ reverse

ASCII

- deprecated, replaced by Unicode

  American Standard Code for Information Interchange

- 7 bits, plus 1 parity bit

  odd / even

  ↓

  A $\xrightarrow{\text{odd}}$ B

  1 0 1 0 1 1 0 1 ⇒ Check if transmitted correctly

  odd no. of 1s.

  corruption 1 1 1 0 1 1 0 1

  ('flipped'

  ⇒ 6 one-bits

  agreed on odd parity

  ⇒ must have error

  ∴ For error detection upon transfer

- Character representation for 0 is different from int 0.

  ie. 00110000 vs 0000000

- Easy conversion: '9' − '0' = 9 (int)

- 01000110 — as 'int' = 70
  — as 'char' = 'F'

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

Negative Numbers

- Signed number: (same for positive number)
  - Sign and magnitude – 1-bit sign, 7-bit magnitude
    - 0 = +ve, 1 = -ve.
    - 2 zeroes → 00000000_{sm}, $+0_{10}$ and 10000000_{sm}, -0_{10} ⇒ good for limits
      sign and magnitude
    - Range: -127_{10} to $+127_{10}$
    - $\pm 2^{n-1}-1$ values represented for n-bits.
      minus one zero.
    - bad for arithmetic ⇒ add +ve and -ve cannot get 0.
    - $01111111 = +127_{10}$ (largest)
      $11111111 = -127_{10}$ (smallest)
  - 1's Complement
    - easy to implement
    - $-x_b = 2^n - x - 1$

      i.e.
      $-00001100_2 = 2^8 - 12 - 1$
      $= 243$
      $-12 = 11110011_{2s}$
      (flip all the bits)
    - largest: $01111111_{1s} = +127_{10}$
      Smallest: $10000000_{1s} = -127_{10}$
      Zeroes: 2, $00000000 = +0_{10}$
      $11111111 = -0_{10}$
      range: $-(2^{n-1}-\underset{-ve\ 0}{1})$ to $(2^{n-1}-\underset{+ve\ 0}{1})$

      Arithmetic: +ve + -ve → ±0
      however, 1 off for normal arithmetic in negative side.
      ⇒ due to negative zero.

2's Complement

- requires 1's complement to implement in circuit.
- $-x_b = 2^n - x$
- invert all the bits, add one. *

  Shortcut: 12 → -12
  12 : 0000 1100
  -12 : 1111 0100$_{2s}$
  inverted
- NOT gates
- i.e. 0000 1100 (12) → note: this is also in 2s (for +12)
  → 1111 0011
  \+ 1
  1111 0100$_{2s}$ (-12)
- Largest: $0111\ 1111 = +127_{10}$
  Smallest: $1000\ 0000 = -128_{10}$
  MSB still rep. the sign
  Zero: $0000\ 0000 = +0_{10}$
- Range: -2^{n-1} to $2^{n-1}-1$

2s complement addition and subtraction

- algorithm for A+B:
  - Binary addition of A+B
  - Ignore carry out of MSB.
  - Check for overflow
    - Overflow occurs if the 'carry in' and 'carry out' of MSB are different, or if the result is opposite sign of A and B.

$$
\begin{array}{rcccc}
 & x_3 & x_2 & x_1 & x_0 \\
 & a_3 & a_2 & a_1 & a_0 \\
+ & b_3 & b_2 & b_1 & b_0 \\
\hline
 & C_3 & C_2 & C_1 & C_0
\end{array}
$$

if both 1 or both 0 ⇒ overflow

if $x_3 \neq x_2$ ⇒ overflow

Range: -2^{n-1} to $2^{n-1}-1$

If $x \geq 2^{n-1} > 2^{n-1}-1$ ⇒ overflow

Detection: +ve + +ve → −ve

−ve + −ve → +ve

⇓

If sign different, will not overflow.

eg.

$$
\begin{array}{rl}
 & 0101_{2s} \rightarrow 5 \\
+ & 0110_{2s} \rightarrow 6 \\
\hline
 & 1011
\end{array}
$$

carry-in: 1

carry 0 (carry out)

diff. ∴ overflow

⇓ easier to detect with hardware

1011 = −ve ↓ 'negate' $-(0101)_2 = -5_{10}$

- A−B = A+(−B)

1's Complement Addition / Subtraction

- A+B
- perform binary addition
- If there is a carry out of MSB, add 1 to result
- Check for overflow ⇒ If result is opposite sign of A & B.

eg.

$$
\begin{array}{rl}
 & 1101 \\
+ & 1010 \\
\hline
 & 10111 \\
+ & 1 \\
\hline
 & 1000
\end{array}
$$

note: weight of 1's complement signed bit is $-2^{n-1}-1$

For 2's complement: -2^{n-1}

∴ −7

Excess Representation

- allows range of values to be evenly distributed between the positive and negative values.
- by simple translation (addition/subtraction)
- Excess-n
  → means that 0000 is −n

  1111 is n−1

- given n-bit, excess 2^{n-1}

  → evenly distributed between +ve and −ve.

Real number representation (fractions etc.)

- We always have a finite number of bits.

- Fixed point representation:

$$\underbrace{110}_{\text{always have some number of integer bits}} . \underbrace{\overset{0.5}{0}\overset{0.25}{1}}_{\text{and also have some number of fractional bits}}$$

always have some number of integer bits

and also have some number of fractional bits

- max accuracy of 0.25

2 different circuits

Cannot change hardware

- binary point cannot be moved

problem:

- limited range

∴ floating point representation:

- represent very large/small numbers
- very high accuracy
- Computer's representation of exponent notation.

Limited size.

IEEE 754.

Sign	exponent	mantissa

only for mantissa

- base is 2.
- Single* precision (32 bits): 1 bit sign
  (focus)
  8 bit exponent, excess-127
  23 bit mantissa.
- Double precision (64 bits): 1 bit sign
  11 bit exponent, excess-1023
  52 bit mantissa.

- Sign bit: 0 for positive
  1 for negative
- Mantissa (fraction)
  - normalised with implicit leading of 1 — gain 1 more bit of accuracy in mantissa.

i.e. $110.1_2 \rightarrow 1.\underbrace{101}_{\text{mantissa}}{}_2 \times 2^2$ * ← meaning of floating point rep.

$0.00101101_2 \rightarrow 1.01101_2 \times 2^{-3}$

hidden bit (not represented)

- Eg.

$\underline{-6.5_{10}}$

(1) Convert to binary

-110.1_2

(2) Calculate mantissa

$-1.\underbrace{101}_{\text{mantissa}}{}_2 \times 2^{2}$ — Store in excess-127

Sign bit $= \underline{1}$ (1 bit)

(3) Calculate exponent

Exponent $= 2 + 127 = 129$

$= \underbrace{100}_{C}\ \underbrace{0000}_{0}\ \underline{1}_2$ (8 bits)

(4) Mantissa $= \underbrace{101}_{D}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}$ (23 bits)

32 bits.

(5) Convert to hex: $0\text{x}C0D00000_{16}$

easier to convert back.