BINARY DATA REPRESENTATION

Michael D'Argenio – mjdargen@ncsu.edu
Electrical Engineering – SS 2019 – Duke TIP

How do we represent data in a computer?

- At the lowest level, a computer is an electronic machine.
  - Works by controlling the flow of electrons
- Easy to recognize two conditions:
  1. presence of a voltage – we'll call this state "1"
  2. absence of a voltage – we'll call this state "0"
- Could base state on *value* of voltage, but control and detection circuits more complex.
  - Early computer iterations attempted trinary and quinary systems.
  - Eventually settled on binary system.

Computer is a binary digital system.

- Basic unit of information is the *binary digit*, or *bit*.
- Values with more than two states require multiple bits.
  - A collection of two bits has four possible states:
    - 00, 01, 10, 11
  - A collection of three bits has eight possible states:
    - 000, 001, 010, 011, 100, 101, 110, 111
  - A collection of **n** bits has $\mathbf{2^n}$ possible states.

Digital system:

- finite number of symbols

Binary (base two) system:

- has two states: 0 and 1

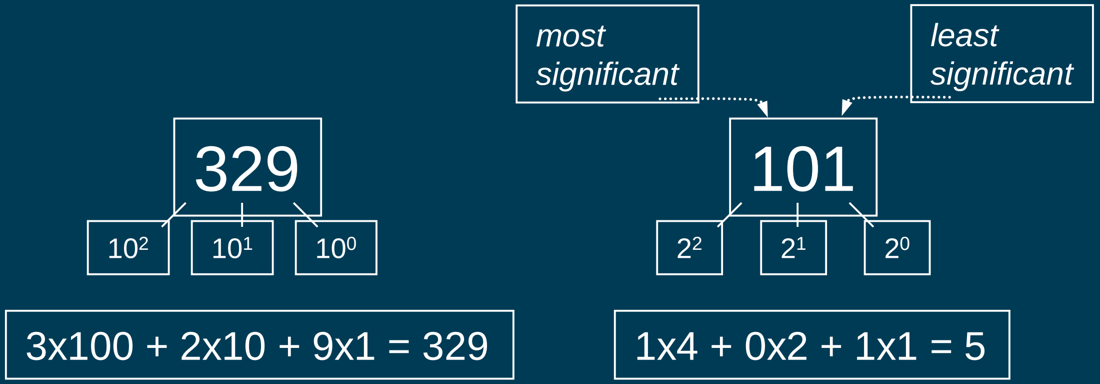
What kinds of data do we need?

- Numbers – signed, unsigned, integers, floating point, complex, rational, irrational, …
- Text – characters, strings, …
- Images – pixels, colors, shapes, …
- Sound
- Logical – true, false
- Instructions
- …

- Data type: *representation* and *operations* within the computer

Unsigned Integers

- Non-positional notation - could represent a number (“5”) with a string of ones (“11111”)
- Weighted positional notation
  - Use in our decimal number system: “329”
  - “3” is worth 300, because of its position, while “9” is only worth 9



Unsigned Integers (cont.)

- An *n*-bit unsigned integer represents 2^n values: from 0 to 2^n-1.

2^2	2^1	2^0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Unsigned Binary Arithmetic

- Base-2 addition – just like base-10!
  - add from right to left, propagating carry

```
   10010        10010        1111
+   1001     +   1011     +     1
   11011        11101       10000
```

carry

```
   10111
+    111
```

Signed Integers

- With n bits, we have 2^n distinct values.
  - assign about half to positive integers (1 through 2^{n-1}) and about half to negative (-2^{n-1} through -1)
- Most significant bit (MSB) indicates sign: 0=positive, 1=negative
- Positive integers
  - just like unsigned – zero in MSB to show it's positive.
    - 00101 = 5
- Negative integers
  - sign-magnitude – one in MSB to show it's negative. Other bits are the same as unsigned.
    - 10101 = -5
- Note there are other signed notations we won't cover.

Hexadecimal Notation

- It is often convenient to write binary (base-2) numbers as hexadecimal (base-16) numbers instead.
  - fewer digits - four bits per hex digit
  - less error prone - easy to corrupt long string of 1's and 0's

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

Binary	Hex	Decimal
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Converting from Binary to Hexadecimal

- Every four bits is a hex digit.
- Start grouping from right-hand side

```
 011 1010 1000 1111 0100 1101 0111
  ⇓    ⇓    ⇓    ⇓    ⇓    ⇓    ⇓
  3    A    8    F    4    D    7
```

This is not a new machine representation, just a convenient way to write the number.

Fractions: Fixed-Point

- How can we represent fractions?
- Use a “binary point” to separate positive from negative powers of two -- just like “decimal point.”

$2^{-1} = 0.5$

$2^{-2} = 0.25$

$2^{-3} = 0.125$

```
  00101000.101  (40.625)
+ 11111110.110  (-1.25)
  00100111.011  (39.375)
```

No new operations -- same as integer arithmetic.

Very Large & Very Small: Floating-Point

- Large values: 6.023×10^{23} -- requires 79 bits
- Small values: 6.626×10^{-34} -- requires >110 bits
- Use equivalent of "scientific notation": $F \times 2^E$
- Need to represent F (*fraction*), E (*exponent*), and sign.
- IEEE 754 Floating-Point Standard (32-bits):

1b	8b	23b
S	Exponent	Fraction

$$N = (-1)^S \times 1.\text{fraction} \times 2^{\text{exponent}-127}, 1 \le \text{exponent} \le 254$$
$$N = (-1)^S \times 0.\text{fraction} \times 2^{-126}, \text{exponent} = 0$$

Floating Point Example

- Single-precision IEEE floating point number:

  1 01111110 10000000000000000000000

  sign *exponent* *fraction*

  - Sign is 1 – number is negative.
  - Exponent field is 01111110 = 126 (decimal).
  - Fraction is 0.100000000000… = 0.5 (decimal).

- Value = $-1.5 \times 2^{(126-127)} = -1.5 \times 2^{-1} = -0.75$.

Text: ASCII Characters

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

Other Data Types

- Text strings
  - sequence of characters, terminated with NULL (0)
  - typically, no hardware support
- Image
  - array of pixels
    - monochrome: one bit (1/0 = black/white)
    - color: red, green, blue (RGB) components (e.g., 8 bits each)
    - other properties: transparency
  - hardware support:
    - typically none, in general-purpose processors
    - MMX -- multiple 8-bit operations on 32-bit word
- Sound
  - sequence of fixed-point numbers

Operations: Arithmetic and Logical

- Recall: a data type includes *representation* and *operations*.
- In the instructions our computer operates, we have operands (values) and opcodes (type of operation).
- We will have a number of arithmetic operations including:
  - Addition
  - Subtraction
  - Sign Extension
- Important to keep track of overflow.
- Multiplication, division, etc., can be built from these operations.
- Logical operations are also useful:
  - AND
  - OR
  - NOT