Unit 3

Binary Representation

ANALOG VS. DIGITAL

Analog vs. Digital

- The analog world is based on continuous events. Observations can take on (real) any value.

- The digital world is based on discrete events. Observations can only take on a finite number of discrete values

Analog vs. Digital

- Q. Which is better?
- A. Depends on what you are trying to do.

- Some tasks are better handled with analog data, others with digital data.
  - Analog means continuous/real valued signals with an infinite number of possible values
  - Digital signals are discrete [i.e. 1 of n values]

Analog vs. Digital

- How much money is in my checking account?
  - Analog: Oh, some, but not too much.
  - Digital: $243.67

Analog vs. Digital

- How much do you love me?
  - Analog: I love you with all my heart!!!!
  - Digital: 3.2×10^3 MegaHearts

The Real (Analog) World

- The real world is inherently analog.
- To interface with it, our digital systems need to:
  - Convert analog signals to digital values (numbers) at the input.
  - Convert digital values to analog signals at the output.
- Analog signals can come in many forms
  - Voltage, current, light, color, magnetic fields, pressure, temperature, acceleration, orientation

Digital is About Numbers

- In a digital world, numbers are used to represent all the possible discrete events (i.e. 1-of-n possibilities)
  - Numerical values (5.7, 1923.8, …)
  - Computer instructions (ADD, SUB, BLE, …)
  - Characters ('a', 'b', 'c', …)
  - Conditions (on, off, ready, paper jam, …)
- Numbers allow for easy manipulation
  - Add, multiply, compare, store, …
- Results are repeatable
  - Each time we add the same two number we get the same result

DIGITAL REPRESENTATION

Interpreting Binary Strings

- Given a string of 1's and 0's, you need to know the *representation system* being used, before you can understand the value of those 1's and 0's.
- Information (value) = Bits + Context (System)

01000001 = ?

Unsigned Binary system

65_{10}

BCD System

41_{BCD}

ASCII system

'A'$_{ASCII}$

Binary Representation Systems

- Integer Systems
  - Unsigned
    - Unsigned (Normal) binary
  - Signed
    - Signed Magnitude
    - 2's complement
    - *Excess-N**
    - *1's complement**
- *Floating Point**
  - *For very large and small (fractional) numbers*

- Codes
  - Text
    - ASCII / Unicode
  - Decimal Codes
    - BCD (Binary Coded Decimal) / (8421 Code)

* = Not fully covered in this class

OVERVIEW

4 Skills

- We will teach you 4 skills that you should know and be able to apply with confidence
  - Convert a number in any base (base r) to decimal (base 10)
  - Convert a decimal number (base 10) to binary
  - Use the shortcut for conversion between binary (base 2) and hexadecimal (base 16)
  - Understand the finite number of combinations that can be made with n bits (binary digits) and its implication for codes including ASCII and Unicode

Number Systems

- Number systems consist of
  1. A base (radix) r
  2. r coefficients [0 to r-1]
- Human System: Decimal (Base 10): 0,1,2,3,4,5,6,7,8,9
- Computer System: Binary (Base 2): 0,1
- Human systems for working with computer systems (shorthand for human to read/write binary)
  - Octal (Base 8): 0,1,2,3,4,5,6,7
  - Hexadecimal (Base 16): 0-9,A,B,C,D,E,F (A thru F = 10 thru 15)

Skill 1: Converting to Decimal

- Number systems use implied place values (weights) that are powers of the base

$$(\underset{10^2=100}{\_\_\_\_}\ \underset{10^1=10}{\_\_\_\_}\ \underset{10^0=1}{\_\_\_}\ .\ \underset{10^{-1}=0.1}{\_\_\_\_})_{10} \qquad (\underset{2^3=8}{\_\_\_}\ \underset{2^2=4}{\_\_\_}\ \underset{2^1=2}{\_\_\_}\ \underset{2^0=1}{\_\_\_}\ .\ \underset{2^{-1}=0.5}{\_\_\_\_}\ \underset{2^{-2}=0.25}{\_\_\_\_})_2$$

- Converting from **another base** to **decimal** requires summing the product of each digit with its implied place value

$$(\underset{8}{1}\ \underset{4}{1}\ \underset{2}{1}\ \underset{1}{0}\ .\ \underset{0.5}{0}\ \underset{0.25}{1})_2 = (14.25)_{10}$$

Conversion Examples 1

$(746)_8 = 7*8^2 + 4*8^1 + 6*8^0$

$= 448 + 32 + 6 = 486_{10}$

$(1A5)_{16} = 1*16^2 + 10*16^1 + 5*16^0$

$= 256 + 160 + 5 = 421_{10}$

Skill 2: Converting from Decimal

- Number systems use implied place values (weights) that are powers of the base

$(\underline{\quad}\ \underline{\quad}\ \underline{\quad}.\underline{\quad})_{10}$ $(\underline{\quad}\ \underline{\quad}\ \underline{\quad}\ \underline{\quad}.\underline{\quad}\ \underline{\quad})_{2}$

$10^2=100\ \ 10^1=10\ \ 10^0=1\ \ 10^{-1}=0.1$ $2^3=8\ \ 2^2=4\ \ 2^1=2\ \ 2^0=1\ \ 2^{-1}=0.5\ \ 2^{-2}=0.25$

- Converting from **decimal** to **another base** requires finding the digits that, when multiplied times each place value, will sum to the desired value (i.e. making change).

$(29.75)_{10} = (\underline{\quad}\ \underline{\quad}\ \underline{\quad}\ \underline{\quad}\ \underline{\quad}.\underline{\quad}\ \underline{\quad})_{2}$

16 8 4 2 1 0.5 0.25

Conversion Examples 2

$(87)_{10} = 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1 = (01010111)_2$

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

128 64 32 16 8 4 2 1

$(77)_{10} = 4\ \ 13 = (4D)_{16}$

$16^1\ 16^0$

16 1

$(4.375)_{10} = 1\ 0\ 0.\ 0\ 1\ 1 = (100.011)_2$

$2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}$

4 2 1 1/2 1/4 1/8

Shortcuts for Converting Binary (r=2), Hexadecimal (r=16) and Octal (r=8)

SHORTHAND FOR BINARY

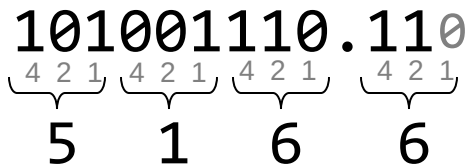
Binary, Octal, and Hexadecimal

- Octal (base 8 = 2^3)
- 1 Octal digit (?)$_8$ can represent: 0 – 7
- 3 bits of binary (? ? ?)$_2$ can represent: 000-111 = 0 – 7
- Conclusion… 1 Octal digit = 3 bits

- Hex (base 16=2^4)
- 1 Hex digit (?)$_{16}$ can represent: 0-F (0-15)
- 4 bits of binary (? ? ? ?)$_2$ can represent: 0000-1111= 0-15
- Conclusion… 1 Hex digit = 4 bits

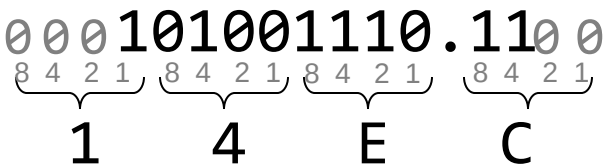
Skill 3: Binary to Octal or Hex

- Make groups of 3 bits starting from radix point and working outward
- Add 0's where necessary
- Convert each group of 3 to an octal digit



516.6_8

- Make groups of 4 bits starting from radix point and working outward
- Add 0's where necessary
- Convert each group of 4 to a hex digit



$14E.C_{16}$

Octal or Hex to Binary

- Expand each octal digit to a group of 3 bits

317.2_8

4 2 1 4 2 1 4 2 1 4 2 1

$011\,001\,111.010_2$

11001111.01_2

- Expand each hex digit to a group of 4 bits

$D93.8_{16}$

8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1

$1101\,1001\,0011.1000_2$

110110010011.1_2

Hexadecimal Representation

- Since values in modern computers are many bits, we use hexadecimal as a shorthand notation (4 bits = 1 hex digit)
  - 11010010 = D2 hex or 0xD2 if you write it in C/C++
  - 0111011011001011 = 76CB hex or 0x76CB if you write it in C/C++

ASCII & Unicode

BINARY CODES

Binary Representation Systems

- Integer Systems
  - Unsigned
    - Unsigned (Normal) binary
  - Signed
    - Signed Magnitude
    - 2's complement
    - *1's complement**
    - *Excess-N**
- *Floating Point**
  - *For very large and small (fractional) numbers*
- Codes
  - Text
    - ASCII / Unicode
  - Decimal Codes
    - BCD (Binary Coded Decimal) / (8421 Code)

* = Not covered in this class

Skill 4: Unique Combinations

- Given *n* digits of base *r*, how many unique numbers can be formed? $\mathbf{r^n}$ What is the range? [**0 to $\mathbf{r^n-1}$**]
- Use the examples below to generalize the relationship

2-digit, decimal numbers (r=10, n=2)					0-9	0-9	**100 combinations: 00-99**
3-digit, decimal numbers (r=10, n=3)				0-9	0-9	0-9	**1000 combinations: 000-999**
4-bit, binary numbers (r=2, n=4)			0-1	0-1	0-1	0-1	**16 combinations: 0000-1111**
6-bit, binary numbers (r=2, n=6)	0-1	0-1	0-1	0-1	0-1	0-1	**64 combinations: 000000-111111**

Main Point: Given n digits of base r, $\mathbf{r^n}$ unique numbers can be made with the range [0 - (r^n-1)]

Approximating Large Powers of 2

- Often need to find decimal approximation of a large powers of 2 like 2^{16}, 2^{32}, etc.
- Use following approximations:
  - $2^{10} \approx 10^3$ (1 thousand) = 1 Kilo-
  - $2^{20} \approx 10^6$ (1 million) = 1 Mega-
  - $2^{30} \approx 10^9$ (1 billion) = 1 Giga-
  - $2^{40} \approx 10^{12}$ (1 trillion) = 1 Tera-
- For other powers of 2, decompose into product of 2^{10} or 2^{20} or 2^{30} and a power of 2 that is less than 2^{10}
  - 16-bit half word: 64K numbers
  - 32-bit word: 4G numbers
  - 64-bit dword: 16 million trillion numbers

$2^{16} = 2^6 * 2^{10}$
$\approx 64 * 10^3 = 64{,}000$

$2^{24} = 2^4 * 2^{20}$
$\approx 16 * 10^6 = 16{,}000{,}000$

$2^{28} = 2^8 * 2^{20}$
$\approx 256 * 10^6 = 256{,}000{,}000$

$2^{32} = 2^2 * 2^{30}$
$\approx 4 * 10^9 = 4{,}000{,}000{,}000$

Binary Codes

- Using binary we can represent any kind of information by coming up with a code
- Using *n* bits we can represent 2^n distinct items

Colors of the rainbow:
- Red = 000
- Orange = 001
- Yellow = 010
- Green = 100
- Blue = 101
- Purple = 111

Letters:
- 'A' = 00000
- 'B' = 00001
- 'C' = 00010

.
.
.

- 'Z' = 11001

ASCII Code

- Used for representing text characters
- Originally 7-bits but usually stored as 8-bits = 1-byte in modern computer systems
- Example:
  - `"Hello\n"`
  - Each character is converted to ASCII equivalent
    - 'H' = 0x48, 'e' = 0x65, …
    - \n = newline character is represented by either one or two ASCII character
      - LF (0x0A) = line feed (moves cursor down a line)
      - CR (0x0D) = carriage return character (moves cursor to start of current line)
      - Newline for Unix / Mac = LF only
      - Newline for Windows = CR + LF

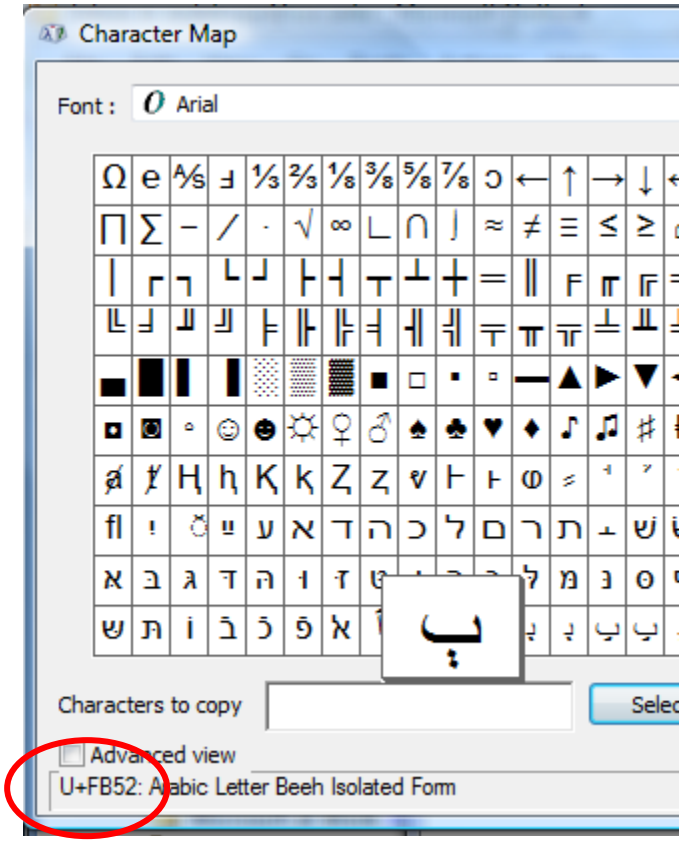
ASCII Table

'M' = 0x4D = 0100 1101

LSD/MSD	0	1	2	3	4	5	6	7
0	NULL	DLW	SPACE	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	TAB	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

UniCode

- ASCII can represent only the English alphabet, decimal digits, and punctuation
  - 7-bit code => 2^7 = 128 characters
  - It would be nice to have one code that represented more alphabets/characters for common languages used around the world
- Unicode
  - Up to 32-bit Code => up to 4 billion combinations
  - 137,000 character defined for many languages
  - Used by Java as standard character code



Unicode hex value
(i.e. FB52 => 1111101101010010)

Review: Chars and Ints

- The C/C++ language supports types `char` and `int`. Let's see what you know about these:

True/False	Question
False	In C/C++: '1' and 1 are the same. - '1' is replaced with its equiv. ASCII number: 0x31 - Thus, storing '1' or 0x31 or 49 to a char variable are all equivalent (the hardware does not store/record how the bits will be interpretated…that is up to the code)
True	chars and ints have different ranges of the values they can represent. - Range of char: -128 to +127 - Range of int on Arduino: ±32K
False	chars are only for ASCII characters (e.g. `char c = '1';` and not `char c = 1;`) - Chars can store integers in the range -128 to 127

Application: Chars and Ints

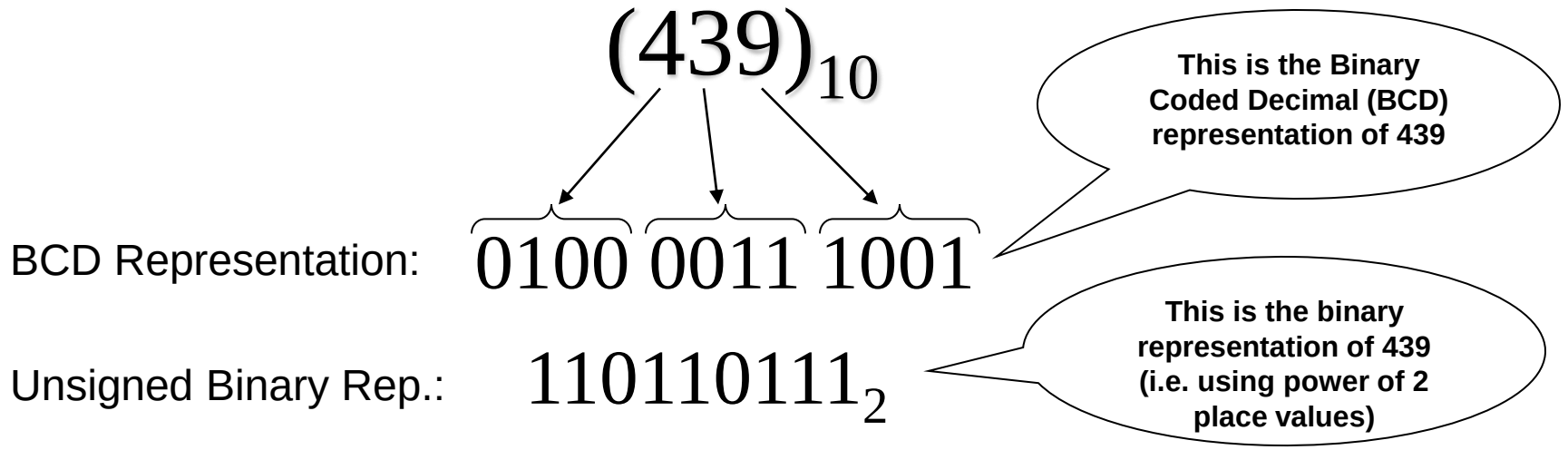
- Lesson 1: A char is just an 8-bit storage location (it can hold integer values or ASCII values)
  - If a function uses a char for an argument or return value, you can still pass integer values (see example to the right)
- Lesson 2: For output (display), we must use ASCII
  - Code will take/expect a certain "interpretation" of a char (either as an ASCII digit for input/output related code) or as a small integer (for processing related code)
- Lesson 3: We can use arithmetic to manipulate chars.

```
char invert(char x)
{
  if(x == '1') return '0';
  else return '1';
}
int main(){
  char y = invert('1');  // passes 0x31=49
}

char invert(char x)
{
  if(x == 1) return 0;
  else return 1;
}
int main(){
  char y = invert(1);    // passes 0x1 = 1
}
```

```
int main(){
  char c = 1;
  putchar(c);    // prints nothing/garbage
}

int main(){
  char c = '1';
  putchar(c);      // prints 1
  c = '0' + 7;
  putchar(c);      // prints 7
}
```

BCD (If Time Permits)

- Rather than convert a decimal number to binary which may lose some precision (i.e. 0.1_{10} = infinite binary fraction), BCD represents each decimal digit as a separate group of bits (exact decimal precision)
  - Each digits is represented as a separate 4-bit number (using place values 8,4,2,1 for each dec. digit)
  - Often used in financial and other applications where decimal precision is needed



Important: Some processors have specific instructions to operate on #'s represented in BCD

Summary

- Convert Base r to Base 10
  - Apply place values (powers of r)
  - $\mathbf{N_r => \Sigma_i(a_i*r^i) => D_{10}}$
- Convert Base 10 to Base r
  - "Make change" using powers of r as the weights/denominations
- Base 2 (Bin) ⇔ Base 16 (Hex)
  - Group or expand 1 hex digit to/from 4 bits
  - Start at binary point and work outward

BACKUP – MORE DETAILS

Using positional weights/place values

BASE R TO BASE 10

Skill 1: Converting Base r to Decimal

- 934.7_{10} = 9 3 4 . 7 = 934.7_{10}
  - $10^2=100$ $10^1=10$ $10^0=1$ $10^{-1}=0.1$
- 1101.1_2 = 1 1 0 1 . 1 = $8+4+1+0.5=13.5_{10}$
  - $2^3=8$ $2^2=4$ $2^1=2$ $2^0=1$ $2^{-1}=0.5$
- $3B.4_{16}$ = 3 B . 4 = $48 + 11 + 0.25=59.25_{10}$
  - $16^1=16$ $16^0=1$ $16^{-1}=0.0625$

Main Point: To convert any base to decimal (base 10), apply the implicit place values (weights) which are just the powers of the base and sum each digit times its place value.

General Conversion From Base r to Decimal

- A number in base r has place values/weights that are the powers of the base
- Denote the coefficients as: a_i

Generalized approach:

$$(a_3a_2a_1a_0.a_{-1}a_{-2})_r = a_3*r^3 + a_2*r^2 + a_1*r^1 + a_0*r^0 + a_{-1}*r^{-1} + a_{-2}*r^{-2}$$

Left-most digit = Most Significant Digit (MSD)

Right-most digit = Least Significant Digit (LSD)

Example:

$$(1001.01)_r = 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 + 0*2^{-1} + 1*2^{-2}$$

$$N_r \Rightarrow \Sigma_i(a_i*r^i) \Rightarrow D_{10}$$

Number in base r

Decimal Equivalent

Examples

$(746)_8 = 7*8^2 + 4*8^1 + 6*8^0$

$= 448 + 32 + 6 = 486_{10}$

$(1A5)_{16} = 1*16^2 + 10*16^1 + 5*16^0$

$= 256 + 160 + 5 = 421_{10}$

$(AD2)_{16} = 10*16^2 + 13*16^1 + 2*16^0$

$= 2560 + 208 + 2 = (2770)_{10}$

Binary Examples

$$(\underset{\mathbf{8}}{1}\underset{\mathbf{4}}{0}\underset{\mathbf{2}}{0}\underset{\mathbf{1}}{1}.\underset{\mathbf{.5}}{1})_2 = 8 + 1 + 0.5 = 9.5_{10}$$

$$(\underset{\mathbf{128}}{1}\underset{\mathbf{32}}{0}\underset{\mathbf{16}}{1}10001)_2 = 128 + 32 + 16 + 1 = 177_{10}$$

Powers of 2

- It helps to memorize the first 11 powers of 2

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$

1024	512	256	128	64	32	16	8	4	2	1

"Making change"

BASE 10 TO BASE 2 OR BASE 16

Skill: Decimal to Base r

- To convert a decimal number, *x,* to binary:
  - Only coefficients of 1 or 0. So simply find place values that add up to the desired values, starting with larger place values and proceeding to smaller values and place a 1 in those place values and 0 in all others
  - Similar to how one would **make change**

25_{10} =	0	1	1	0	0	1
	32	16	8	4	2	1

For 25_{10} the place value 32 is too large to include so we include 16. Including 16 means we have to make 9 left over. Include 8 and 1.

Decimal to Unsigned Binary

73_{10}=

0	1	0	0	1	0	0	1
128	64	32	16	8	4	2	1

87_{10}=

0	1	0	1	0	1	1	1

145_{10}=

1	0	0	1	0	0	0	1

0.625_{10}=

1	0	1	0	0
.5	.25	.125	.0625	.03125

Decimal to Another Base

- To convert a decimal number, *x*, to base r:
  - Use the place values of base r (powers of r). Starting with largest place values, fill in coefficients that sum up to desired decimal value without going over.

75_{10} =

0	4	B	hex
256	16	1	