Lesson 07: Data Representation (W02D3)

Balboa High School

Michael Ferraro

August 26, 2015

Do Now

1. Write a UNIX/Linux command that would...
    1. copy all files ending in `.class` from `/home/student/Desktop/` to the current directory
    2. change the current directory from `/tmp` to the root directory. *There are two correct answers!*
2. What are one of the two core responsibilities of an operating system as described in last class' lesson slides?

Aim

Students will learn how data is represented in binary form by computers and how to convert back and forth between decimal and binary numbers.

Binary

Binary is a means of representing information through two values only, 0 and 1.

- Decimal numbers — ones whose digits range from 0→9 — can be expressed in binary form.
- Information in text:
  Each character → numerical value → binary number

Why do computers use binary numbers?

- In the CPU and in RAM, it's due to the nature of semiconductors (transistors)
  - Input to a gate is some voltage, either below some threshold or above it, considered to be a **0** or **1**
  - Output value of a gate is either
    - low (≈0V) and treated as a **0**, or
    - higher than some agreed-upon level, and treated as a **1**

Why do computers use binary numbers?

- On hard disk media, it's due to how data is stored magnetically

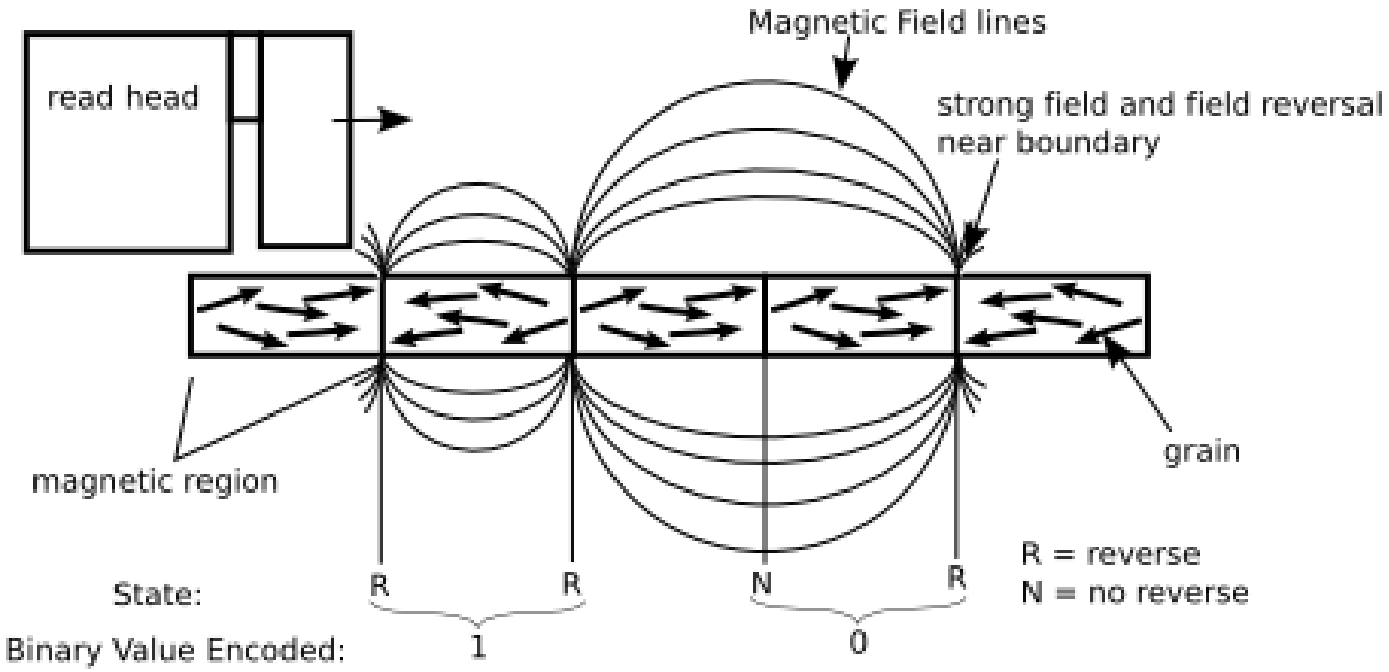


photo credit:
http://en.wikipedia.org/wiki/Image:MagneticMedia.png

Prelude to Binary: Decimal

- Each digit can have one of 10 possible values:
  $\{ 0, 1, 2, \ldots, 9 \}$
- If you list all whole numbers from 1 to 10, you see that you *need* to use a second place value — the *tens* place
- After you've used all possible combination of tens and ones — by the time you reach 99 — it's time to begin using the next place value, hundreds, and then start over with tens and ones (remember this idea when we count in binary...)

Binary: Base-2 Numbers

- Under this system, there are two possible values for a digit: $\{ 0, 1 \}$
- Like decimal, least significant digit is on the right
- What's a *bit*?

Binary: Base-2 Numbers

- Under this system, there are two possible values for a digit: $\{ 0, 1 \}$
- Like decimal, least significant digit is on the right
- What's a *bit*? **BI**nary digi**T**

Counting in Binary

_	_	_	_	decimal

Counting in Binary

_	_	_	_	decimal
0	0	0	0	0

Counting in Binary

_	_	_	_	decimal
0	0	0	0	0
0	0	0	1	1

Counting in Binary

_	_	_	_	decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2

Counting in Binary

_	_	_	_	decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3

Counting in Binary

_	_	_	_	decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4

Counting in Binary

_	_	_	_	decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5

Counting in Binary

_	_	_	_	decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6

Counting in Binary

_	_	_	_	decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

Counting in Binary

_	_	_	_	decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

Counting in Binary

_	_	_	_	decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

$\rightarrow$ **Now you finish up to 15.**

Counting in Binary — Solutions

				decimal
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Binary → Decimal, the Game

Let's play Cisco's Tetris-like binary numbers game...

Binary $\rightarrow$ Decimal Conversion (easy)

Example:

1 1 0 1

Binary $\rightarrow$ Decimal Conversion (easy)

Example:

1	1	0	1
2^3	2^2	2^1	2^0

Binary $\rightarrow$ Decimal Conversion (easy)

Example:

1	1	0	1
2^3	2^2	2^1	2^0
$8 +$	$4 +$	$0 +$	1

Binary $\rightarrow$ Decimal Conversion (easy)

Example:

$$\begin{array}{cccccc} 1 & 1 & 0 & 1 & \\ 2^3 & 2^2 & 2^1 & 2^0 & \\ 8 + & 4 + & 0 + & 1 = & 13 \end{array}$$

Binary $\rightarrow$ Decimal Conversion (easy)

Your turn:

1. 0 1 1 0
2. 1 0 0 1
3. 1 0 1 1 $\quad$ 1 1 0 1

Binary $\rightarrow$ Decimal Conversion (easy)

Your turn:

1. 0 1 1 0 $\leftarrow$ **6**
2. 1 0 0 1 $\leftarrow$ **9**
3. 1 0 1 1 1 1 0 1 $\leftarrow$ **189**

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
189

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
189

Binary Result: _ _ _ _ _ _ _ _

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
$189 \div 2 = 94$ R1

Binary Result: _ _ _ _ _ _ _ _

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
$189 \div 2 = 94$ R1

Binary Result: $\_ \quad \_ \quad \_ \quad \_ \quad \_ \quad \_ \quad \_ \quad \underline{1}$

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
$189 \div 2 = 94$ R1 ; $94 \div 2 = 47$ R0

Binary Result: $\_ \quad \_ \quad \_ \quad \_ \quad \_ \quad \_ \quad \_ \quad \underline{1}$

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
$189 \div 2 = 94$ R1 ; $94 \div 2 = 47$ R0

Binary Result: _ _ _ _ _ _ $\underline{0}$ $\underline{1}$

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
$189 \div 2 = 94$ R1 ; $94 \div 2 = 47$ R0 ; $47 \div 2 = 23$ R1

Binary Result: $\_ \quad \_ \quad \_ \quad \_ \quad \_ \quad \underline{1} \quad \underline{0} \quad \underline{1}$

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
$189 \div 2 = 94$ R1 ; . . . ; $23 \div 2 = 11$ R1

Binary Result: _ _ _ _ 1 1 0 1

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
$189 \div 2 = 94$ R1 ; . . . ; $11 \div 2 = 5$ R1

Binary Result: _ _ _ 1 1 1 0 1

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
$189 \div 2 = 94$ R1 ; . . . ; $5 \div 2 = 2$ R1

Binary Result: _ _ 1 1 1 1 0 1

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
$189 \div 2 = 94$ R1 ; . . . ; $2 \div 2 = 1$ R0

Binary Result: _ $\underline{0}$ $\underline{1}$ $\underline{1}$ $\underline{1}$ $\underline{1}$ $\underline{0}$ $\underline{1}$

Decimal $\rightarrow$ Binary Conversion

One method: *Repeated-Division-by-2 Algorithm*

Example:
$189 \div 2 = 84$ R1 ; . . . ; $1 \div 2 = 0$ R1

Binary Result: $\underline{1}\quad \underline{0}\quad \underline{1}\quad \underline{1}\quad \underline{1}\quad \underline{1}\quad \underline{0}\quad \underline{1}$

Decimal $\rightarrow$ Binary Conversion

Another method: *"Consumption"*

- Turn on a bit to *consume* that bit's value from the decimal number
- Always start with the highest-value bit and work to the right
- Live example: Converting 189_{10} to its base-2 equivalent...

Converting Decimal to Binary

Using your preferred method, convert these base-10 (decimal) values to binary.

1. 15
2. 16
3. 38
4. 255
5. 256

Converting Decimal to Binary

You try:

1. $15 \leftarrow 0\ 0\ 0\ 0\ \ 1\ 1\ 1\ 1$
2. $16 \leftarrow 0\ 0\ 0\ 1\ \ 0\ 0\ 0\ 0$
3. $38 \leftarrow 0\ 0\ 1\ 0\ \ 0\ 1\ 1\ 0$
4. $255 \leftarrow 1\ 1\ 1\ 1\ \ 1\ 1\ 1\ 1$
5. $256 \leftarrow 0\ 0\ 0\ 0\ \ 0\ 0\ 0\ 1\ \ 0\ 0\ 0\ 0\ \ 0\ 0\ 0\ 0$

Bytes

What is a **byte**?

Bytes

What is a **byte**? 8 bits!

Bytes

What is a **byte**? 8 bits!

$\rightarrow$ So how many bits does it take to represent 17?

Bytes

What is a **byte**? 8 bits!

$\rightarrow$ So how many bits does it take to represent 17?

$$17_{10}$$

Bytes

What is a **byte**? 8 bits!

$\rightarrow$ So how many bits does it take to represent 17?

$$17_{10} \rightarrow 10001_2$$

Bytes

What is a **byte**? 8 bits!

$\rightarrow$ So how many bits does it take to represent 17?

$$17_{10} \rightarrow 10001_2 \rightarrow 5 \text{ bits}$$

Bytes

What is a **byte**? 8 bits!

$\rightarrow$ So how many bits does it take to represent 17?

$$17_{10} \rightarrow 10001_2 \rightarrow 5 \text{ bits}$$

$\rightarrow$ How many bytes in memory are needed?

Bytes

What is a **byte**? 8 bits!

→ So how many bits does it take to represent 17?

$$17_{10} \rightarrow 10001_2 \rightarrow 5 \text{ bits}$$

→ How many bytes in memory are needed?

5 bits

Bytes

What is a **byte**? 8 bits!

$\rightarrow$ So how many bits does it take to represent 17?

$$17_{10} \rightarrow 10001_2 \rightarrow 5 \text{ bits}$$

$\rightarrow$ How many bytes in memory are needed?

$$5 \text{ bits} \cdot \frac{1 \text{ byte}}{8 \text{ bits}}$$

Bytes

What is a **byte**? 8 bits!

$\rightarrow$ So how many bits does it take to represent 17?

$$17_{10} \rightarrow 10001_2 \rightarrow 5 \text{ bits}$$

$\rightarrow$ How many bytes in memory are needed?

$$5 \cancel{\text{ bits}} \cdot \frac{1 \text{ byte}}{8 \cancel{\text{ bits}}} = \frac{5}{8} \text{ bytes}$$

Bytes

What is a **byte**? 8 bits!

$\rightarrow$ So how many bits does it take to represent 17?

$$17_{10} \rightarrow 10001_2 \rightarrow 5 \text{ bits}$$

$\rightarrow$ How many bytes in memory are needed?

$$5 \cancel{\text{ bits}} \cdot \frac{1 \text{ byte}}{8 \cancel{\text{ bits}}} = \frac{5}{8} \text{ bytes} = .625 \text{ bytes}$$

Bytes

What is a **byte**? 8 bits!

→ So how many bits does it take to represent 17?

$$17_{10} \rightarrow 10001_2 \rightarrow 5 \text{ bits}$$

→ How many bytes in memory are needed?

$$5 \cancel{\text{ bits}} \cdot \frac{1 \text{ byte}}{8 \cancel{\text{ bits}}} = \frac{5}{8} \text{ bytes} = \cancel{.625 \text{ bytes}}$$

can't have partial bytes — use a ceiling function (which rounds up):

Bytes

What is a **byte**? 8 bits!

$\rightarrow$ So how many bits does it take to represent 17?

$$17_{10} \rightarrow 10001_2 \rightarrow 5 \text{ bits}$$

$\rightarrow$ How many bytes in memory are needed?

$$\cancel{5 \text{ bits}} \cdot \frac{1 \text{ byte}}{\cancel{8 \text{ bits}}} = \frac{5}{8} \text{ bytes} = \cancel{.625 \text{ bytes}}$$

can't have partial bytes — use a ceiling function (which rounds up):

$$= \lceil .625 \text{ bytes} \rceil = \boxed{1 \text{ byte}}$$

How many bits or bytes does it take. . .

Assume the smallest unit of memory is 1 byte (i.e., no partial bytes). If you wanted to represent each of the following decimal numbers in binary, how many (a) bits and (b) bytes would be needed for each?

1. 15
2. 16
3. 38
4. 255
5. 256

How many bits or bytes does it take. . .

Assume the smallest unit of memory is 1 byte (i.e., no partial bytes). If you wanted to represent each of the following decimal numbers in binary, how many (a) bits and (b) bytes would be needed for each?

1. 15 $\leftarrow$ 4 bits, so 1 byte
2. 16 $\leftarrow$ 5 bits, so 1 byte
3. 38 $\leftarrow$ 6 bits, so 1 bytes
4. 255 $\leftarrow$ 8 bits, so 1 bytes
5. 256 $\leftarrow$ 9 bits, so 2 bytes

If you have _ bits, . . .

- If you have 2 bits, how many integers can you represent? Highest int.?
- If you have 3 bits?
- If you have 4 bits?

If you have _ bits, . . .

- If you have 2 bits, how many integers can you represent? Highest int.?
- If you have 3 bits?
- If you have 4 bits?
- *Highest integer: What's the general pattern?*

If you have _ bits, . . .

- If you have 2 bits, how many integers can you represent? Highest int.?
- If you have 3 bits?
- If you have 4 bits?
- *Highest integer: What's the general pattern?*
  $\rightarrow 2^n$ - 1, where n is # bits

But I don't want to store binary stuff!

- Storing strings of 0s and 1s doesn't interest me, so how could that be put to good use?

But I don't want to store binary stuff!

- Storing strings of 0s and 1s doesn't interest me, so how could that be put to good use?
- *How about a useful encoding?*

ASCII

- . . . is an encoding: characters represented by numbers

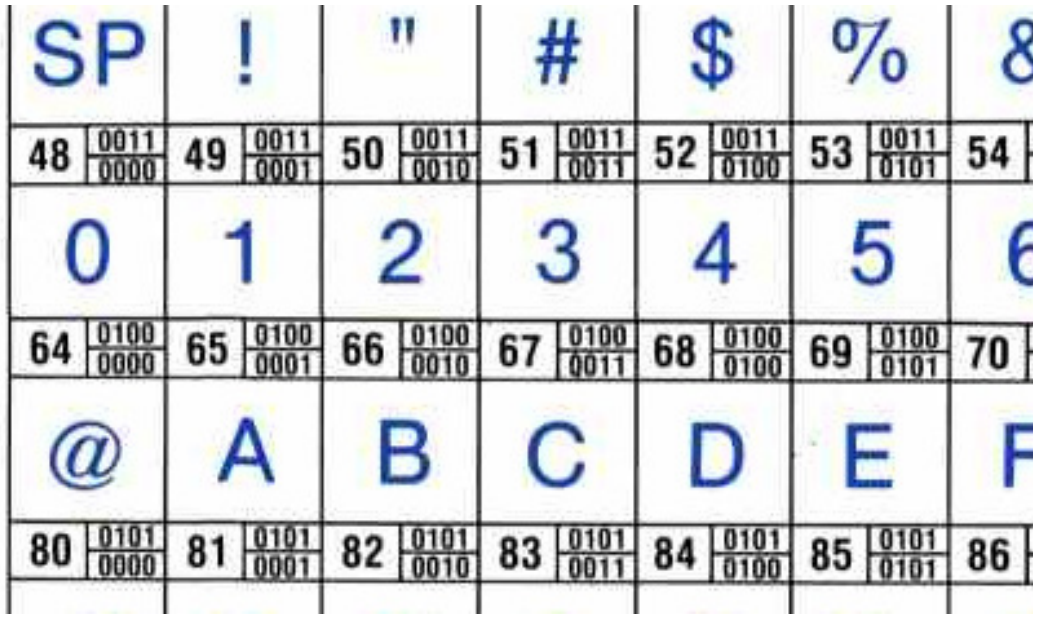
ASCII

- . . . is an encoding: characters represented by numbers
- those numbers can be stored using their binary values

ASCII

- . . . is an encoding: characters represented by numbers
- those numbers can be stored using their binary values
- ASCII stands for
  American Standard Code for Information Interchange

ASCII Table



SP	!	"	#	$	%	&
48 0011 0000	49 0011 0001	50 0011 0010	51 0011 0011	52 0011 0100	53 0011 0101	54
0	1	2	3	4	5	6
64 0100 0000	65 0100 0001	66 0100 0010	67 0100 0011	68 0100 0100	69 0100 0101	70
@	A	B	C	D	E	F
80 0101 0000	81 0101 0001	82 0101 0010	83 0101 0011	84 0101 0100	85 0101 0101	86

photo credit: http://www.ecowin.org/ascii.htm[1]

[1] Note that the image, as presented here, is not the entire original image; consult the URL in the photo credit for the whole image.

Example of ASCII

How would you store the message below?

H	e	l	l	o	,		w	o	r	l	d	!

Example of ASCII

How would you store the message below?

H	e	l	l	o	,		w	o	r	l	d	!
72	101	108	108	111	44	32	119	111	114	108	100	33

Example of ASCII

How would you store the message below?

H	e	l	l	o	,		w	o	r	l	d	!
72	101	108	108	111	44	32	119	111	114	108	100	33

- allow one byte to represent each character in memory or on disk (e.g., H is 72 in decimal and 0100 1000 as one byte of binary)

Example of ASCII

How would you store the message below?

H	e	l	l	o	,		w	o	r	l	d	!
72	101	108	108	111	44	32	119	111	114	108	100	33

- allow one byte to represent each character in memory or on disk (e.g., H is 72 in decimal and `0100 1000` as one byte of binary)
- storage of first 3 letters, **H-e-l**, would be:
  - `72, 101, 108`
  - `0100 1000, 0110 0101, 0110 1100`
  - `010010000110010101101100`

Limitations of ASCII

Limitations of ASCII

- What about other languages?
- ASCII can represent all English characters using one byte for each character
- Problem: using only 8 bits for each character means you can only have up to 2^8 (or 256) possible characters, which is far too small

Enter UNICODE

- What about other languages?
- ASCII can represent all English characters using one byte for each character
- Problem: using only 8 bits for each character means you can only have up to 2^8 (or 256) possible characters, which is far too small
- Enter UNICODE
  - Represents many world languages
  - Uses 16 bits (2 bytes) per character!
  - Java uses UNICODE!
    $\therefore$ *Internationalization* is possible when writing software in Java.

Hexadecimal

- Binary isn't a panacea[2] — inconvenient for people to represent data since large numbers require long strings of 0s and 1s
- Sometimes base-10 (decimal) isn't compact enough
- Enter hexadecimal, a **base-16** number system...
- Each hex digit (called a *nibble*) encodes one of 16 possible values:
  { `0, 1, 2, 3, ... , 9, a, b, c, d, e, f` }

[2] see http://dictionary.reference.com/browse/panacea

Counting in Hex

binary	decimal	hexadecimal
0000	0	00
0001	1	01
0010	2	02
0011	3	03
0100	4	04
0101	5	05
0110	6	06
0111	7	07
1000	8	08

binary	decimal	hexadecimal
1001	9	09
1010	10	0a
1011	11	0b
1100	12	0c
1101	13	0d
1110	14	0e
1111	15	0f
10000	16	10
10001	17	11

Hex on the Web

In web pages (or style sheets) we often use hex representations for colors:

Blue-violet	#8A2BE2
Blush	#DE5D83
Bole	#79443B
Bondi blue	#0095B6
Boston University Red	#CC0000
Brandeis blue	#0070FF

[3]

[3] excerpted from http://en.wikipedia.org/wiki/List_of_colors

How much can hex hold?

- each nibble holds one of 16 possible values
- if n nibbles are used, 16^n possible encoding possible
- Ex: Web colors range from #000000 to `#ffffff`

$$\underline{?} \times \underline{?} \times \underline{?} \times \underline{?} \times \underline{?} \times \underline{?}$$

$$\underline{16} \times \underline{16} \times \underline{16} \times \underline{16} \times \underline{16} \times \underline{16}$$

$$16^6$$

$$16{,}777{,}216$$

$\therefore$ $\approx$16.8 million colors can be represented using just 6 characters

Hex↔Bin Conversion

- Each nibble holds 4 bits — so that makes each nibble _____ of a byte. :)
- Simply group your binary value by fours, starting from right, and rewrite the value of each group as a hex value from 0→f
- Ex: Convert 01101011_2 to hexadecimal

  0110 1011

Hex↔Bin Conversion

- Each nibble holds 4 bits — so that makes each nibble _____ of a byte. :)
- Simply group your binary value by fours, starting from right, and rewrite the value of each group as a hex value from 0→f
- Ex: Convert 01101011_2 to hexadecimal

$$\begin{array}{cc} 0110 & 1011 \\ 6_{10} & 11_{10} \end{array}$$

Hex↔Bin Conversion

- Each nibble holds 4 bits — so that makes each nibble _____ of a byte. :)
- Simply group your binary value by fours, starting from right, and rewrite the value of each group as a hex value from 0→f
- Ex: Convert 01101011_2 to hexadecimal

0110	1011
6_{10}	11_{10}
6_{16}	b_{16}

Hex↔Bin Conversion

- Each nibble holds 4 bits — so that makes each nibble _____ of a byte. :)
- Simply group your binary value by fours, starting from right, and rewrite the value of each group as a hex value from $0 \rightarrow \mathtt{f}$
- Ex: Convert 01101011_2 to hexadecimal

$$\begin{array}{ll} 0110 & 1011 \\ 6_{10} & 11_{10} \\ 6_{16} & \mathtt{b}_{16} \end{array}$$

$$\therefore \quad 01101011_2 = 6\mathtt{b}_{16}$$

HW

Finish §3 of PS #1