THINKERSMITH™

Traveling Circuits

Lesson 1

Binary Baubles

a one hour
introductory activity
adapted for CSED Week 2013

COMPUTER SCIENCE
Education Week
WWW.CSEDWEEK.ORG

PO Box 42186, Eugene, OR, 97404

Disclaimer

Neither Thinkersmith nor any other party involved in the creation of this curriculum can be held responsible for damage, mishap, or injury incurred because of this lesson. Adult supervision and supervisor caution is recommended at all times. When necessary, every effort has been made to locate copyright and permission information.

Lesson 1: Binary Baubles

Main Goal: Introduce ***binary*** and the idea of ***coding***.

Overview: This lesson uses the concept of binary to illustrate how a computer codes data that will be stored for use later. It incorporates the idea of opposites, then describes versions of opposites (like up/down and off/on) that can substitute for the stereotypical ones and zeros.

As a final activity, this lesson introduces a coding exercise where students learn to ***encode*** their initials in a form that they can take home.

Objectives:

Students will

- Learn that a computer does not understand language the way that we do
- Find out that all kinds of information can be stored with different combinations of just two choices.
- Learn to encode letters into binary
- Learn how to decode binary into letters

Materials and Preparation:

Estimated lesson time: 1 hour
Estimated prep time: 15 min

Materials

- Computer Image Pack
- ASCII Encoder Card (1 per student)
- Paper Initial Cut-Out (1 per student)
- Colored Markers or Crayons (1 or 2 per student)

Preparation

- Print out one Computer Image Pack
- Print an ASCII Encoder Card for each student
- Print age appropriate Initial Cut-Outs on card stock and cut one per student

Key Lesson Vocabulary:

ASCII - "American Standard Code for Information Interchange" encoding of characters.

Binary - A notation that utilizes only two options for each selection.

Bit - Short for "Binary Digit". It is one digit's location in a binary number.

Code/Coding - Transformation from one representation to another.

Decode - Convert a coded message into something familiar.

Encode - Convert a familiar message into code.

Lesson Plan

Introduce: Explaining the usefulness of binary is a great way to help wrap a student's mind around computer science. Begin this lesson by telling the class that they are going to learn how to understand the language of the computer.

Does a computer communicate the same way that we do? Does it understand English? Spanish? No. Deep down, a computer is translating every piece of information that it gets into its own simply processed language called binary.

The **Hardware Lecture** in this workbook is a good way to introduce the idea that data is transformed before it's stored. Next, challenge the class to figure out how to decode some letters that have been encoded as binary. Finally, allow students to practice transforming their own information.

Steps:

1. Give the class a look at how binary is used in computers.
2. Challenge the class to decode a letter of your choosing.
3. Challenge the class to encode a letter for you to decode.
4. Allow the class to create artwork with their initials encoded in binary.

Adjustments:

Grades K-2

- Use the "light" portion of the **Sample Script**. Too much detail may lose young students.
- Large Initial Cut-Out strips have been provided for young students to encode a single initial. If hand-coloring the squares feels too intense, you can provide 3/4" squares of colored paper to attach with glue sticks.
- Copying the Initial Cut-Outs on thick card stock turns them into personalized bookmarks!

Grades 3-6

- Tailor **Hardware Lecture** to your class' attention span.
- This age group generally does just fine with the small, square Initial Cut Outs.

Grades 7+

- This age group can be very interested in the **Hardware Lecture**. Suggest a follow-up segment that includes YouTube research if they hunger for more than you are comfortable teaching.
- This age group usually enjoys completing one or more of the Initial Cut Out squares.

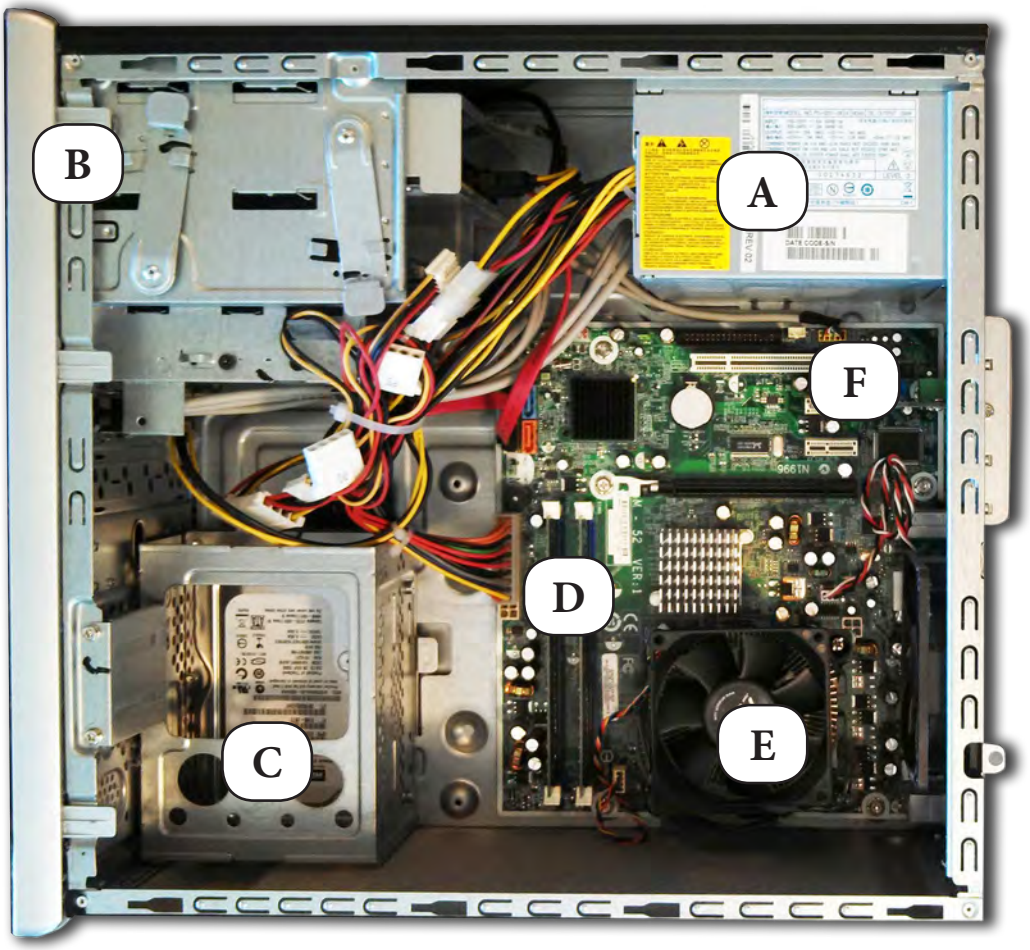
Hardware Lecture:

For this section, we have provided a **Computer Image Pack**. If you have access to the inside of a computer, deconstructed hard drive, and/or R+ - DVD disk, feel free to use those as examples instead of photographs.

It is important to make sure that the students know what is happening inside of a computer. If you are comfortable in this area, you can use this section as a road map for your own lecture. Otherwise, a **Sample Script** has been provided at the end of the booklet.

Inside the machine:

What does the inside of a computer look like? There are no magic monkeys or miniature wizards operating the thing. Take a look at the inside of any standard desktop machine (Image 1 in **Computer Image Pack**).



A) Power Supply B) Expansion Bays for CD/DVD, etc. C) Hard Drive
D) RAM slots E) Processor/CPU with Fan F) Expansion Card Slots

You will see that an intricate series of wires carries power and information from one place to another. Once you know that, you may realize why "off" and "on" are so important.

Do computers actually speak English? Do they store information as sentences that we could understand? No. Computers transmit and store information by encoding it into combinations of digits with only two choices per spot. Most people think of those choices as zeros and ones (like in the movie "The Matrix"). In reality, information can be encoded many ways. Hard Disk Drives (Image 2 in **Computer Image Pack**) use a combination of magnetic positives and magnetic negatives. CDs and DVDs (Image 3 in **Computer Image Pack**) use laser light that either reflects or does not reflect back. Other storage devices may use other options, but the method is still binary.

Now that the need for binary is clear, it's time to practice encoding simple data, like letters [1].

Letter	Binary	Letter	Binary
A	■□■■ ■■■□	N	■□■■ □□□■
B	■□■■ ■■□■	O	■□■■ □□□□
C	■□■■ ■■□□	P	■□■□ ■■■■
D	■□■■ ■□■■	Q	■□■□ ■■■□
E	■□■■ ■□■□	R	■□■□ ■■□■
F	■□■■ ■□□■	S	■□■□ ■■□□
G	■□■■ ■□□□	T	■□■□ ■□■■
H	■□■■ □■■■	U	■□■□ ■□■□
I	■□■■ □■■□	V	■□■□ ■□□■
J	■□■■ □■□■	W	■□■□ ■□□□
K	■□■■ □■□□	X	■□■□ □■■■
L	■□■■ □□■■	Y	■□■□ □■■□
M	■□■■ □□■□	Z	■□■□ □■□■

Depiction of ASCII Encoder Card

1) Your class may be extremely curious about encoding numbers, letters, images and sound. Some details have been provided at the end of this lesson for your convenience. See **Info Section** *for more on those formats.*

Example

Beginning: To boost confidence before the main activity, it can be helpful to go over a few examples together. One of the best ways to help the class understand this is to read an encoded letter aloud, then see if they can figure out the decoded character with help of their ASCII Encoder Card.

In this part of the guide, Thinkersmith intentionally uses the terms "off" and "on" or "color" and "don't color" rather than "one" and "zero". Since this activity is often done in elementary school, we prefer not to mess with a student's burgeoning sense of arithmetic. However, if you are teaching this activity to students in grades 6+ the words "one" and "zero" are perfectly appropriate choices for your vocabulary [2].

Choose any letter from the ASCII Encoder Card. In this example, we will use 'K':

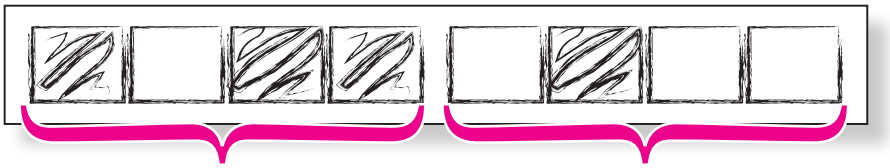
K ■□■■ □■□□

Keep the selected letter to yourself until after you have read the binary combination out loud. Challenge the students to listen as you call out the bits. The letter K might sound like:

"Color", "Don't Color", "Color", "Color"
"Don't Color", "Color", "Don't Color", "Don't Color"

If the class is able to shout out the correct letter upon direction, then you can continue with another example or two, and move on to the next segment.

If the students remain perplexed, draw the encoded letter on the board or overhead projector. Encourage the students to match the first set of four boxes, before they try to match the second set.



Match this group of bits first *Then match these bits*

The class may find it interesting to discover that the letters A-O begin with ■□■■, while P-Z begin with ■□■□. Now, assist the class in narrowing down to the letter that you have chosen. Keep giving examples until the class is able to correctly guess a letter without your assistance.

2) This is a great place to explain what bits are. See **Info Section** *for details.*

Completion: Once the students are comfortable guessing your letter, let them turn the tables. Leave the room and have them come up with an encoding as a group (with the help of either a teaching assistant or a designated volunteer). Give them a two minute time limit. This prevents them from deliberating over the selection for too long and it also lets you know when you can return, in case no one comes to get you.

Using your ASCII Encoder Card[3], decode the letter in the same way that the class did earlier. If the letter that you decode is different from what they intended, return to the beginning of the example and try again. Otherwise, congratulate the crew and transition into the activity. Sometimes, individual students decide that *they* would like a turn leaving the room and returning to guess an encoded letter. This is a good way to add time to the overall lesson without going into additional technical detail.

If everyone is still enjoying themselves and time permits, you can pair the students to encode letters or words for each other. It can be a lot of fun to work in small groups to create encodings, then trade with other groups to solve them. This is also a helpful technique to pull around the last handful of students who feel like they are having a hard time grasping the concept of binary encoded ASCII.

3 - If you're feeling brave, you can try to guess without the ASCII Encoder Card by doing a little mental binary math. See **Info Section**.

The Exercise

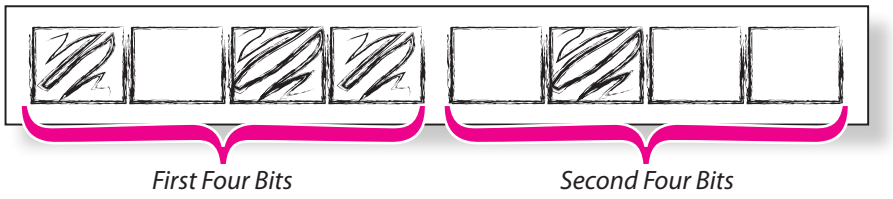
Hand Out: Make sure that every student has an ASCII Encoder Card, Paper Initial Cut-Out, and marker.

Let them know that this section is where they get to make a take-home reminder of the lesson. While small paper squares are simple and convenient, it is also possible to adhere 1-inch squares of quad paper (graph paper with 4 squares per linear inch) to a 1-inch square magnet and give the students a more useful bauble. The lesson can also be done with quad paper that has been run through a sticker machine or printed on shrink film in order to make this activity a doorway for more involved projects.

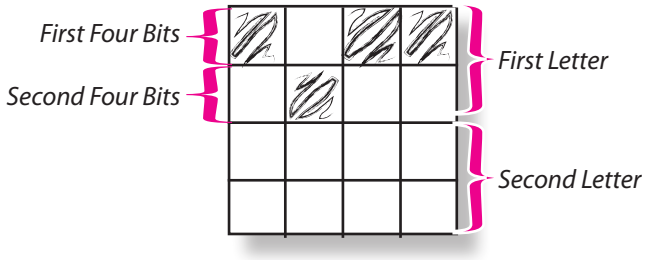
With a younger class (grades K-2) you can modify this section to the larger bookmark sized binary strips, or use beads on pipe-cleaners.

The rest of this lesson will proceed as if the students have been given the square Paper Initial Cut-Outs.

Prepare: Often, students need help making the jump from a linear encoding system (where all 8-bits are presented on one line):



to the two-line encoding required for the Paper Initial Cut-Out squares:



To thwart confusion, show them that the bits on the ASCII Encoder Card are broken up into two sections of size four. The first four bits will be the top row, and the second four bits will be their second row.

Help them understand that each letter has 8 bits (8 individual squares) and that the Paper Initial Cut-Outs have 16 squares, so they should be able to encode two letters. Many choose to encode their first and last initial.

Create:

This activity is meant to solidify encoding and decoding concepts for the students. It is usually done as an individual exercise, but can be modified easily for pairs by allowing students to help one another determine which bits go onto which row.

Students will select two letters (just one if using the bookmark style strips) to encode onto their Paper Initial Cut-Outs. Once their letters have been chosen, they can either take the intermediate step of writing the encoding on a blank piece of paper, or they can encode directly onto their Cut-Outs. Using a marker, each student will fill in the bits of their Cut-Outs according to the pattern for their selected letter.

Generally, the black squares (considered "off") are the squares that get colored, but occasionally students will flip the code and color the squares that are considered "on". This is perfectly alright, as long as they make their method clear.

Occasionally, students will "mess-up" on their patterns. If you have enough Paper Initial Cut-Outs to allow them a new one, it is okay to do so. If you find yourself limited in the number of Paper Initial Cut-Outs available, then challenge the student to figure out what their alternate encoding means in ASCII[4] and let them think of a new idea for what their encoding might symbolize.

When the students are done with their Paper Initial Cut-Out you can encourage them to share it with others or find a prominent place to display it.

Adjustments:

Grades K-2

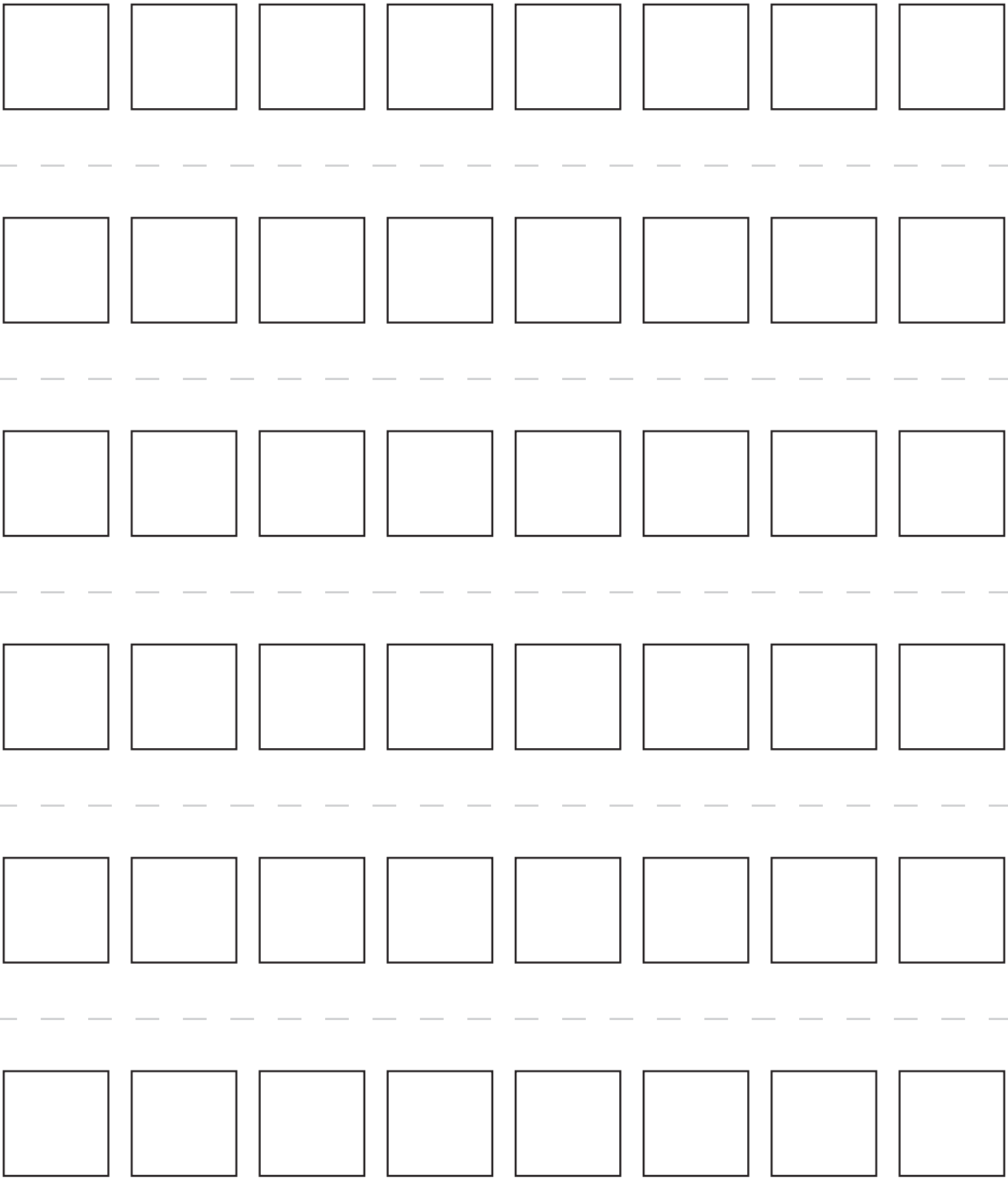
- Let younger students work on just one 8-bit line, representing one letter.
- If the class is having a tough time, choose one letter that means something to all of you (like the school's initials) to code together as a class.

Grades 7+

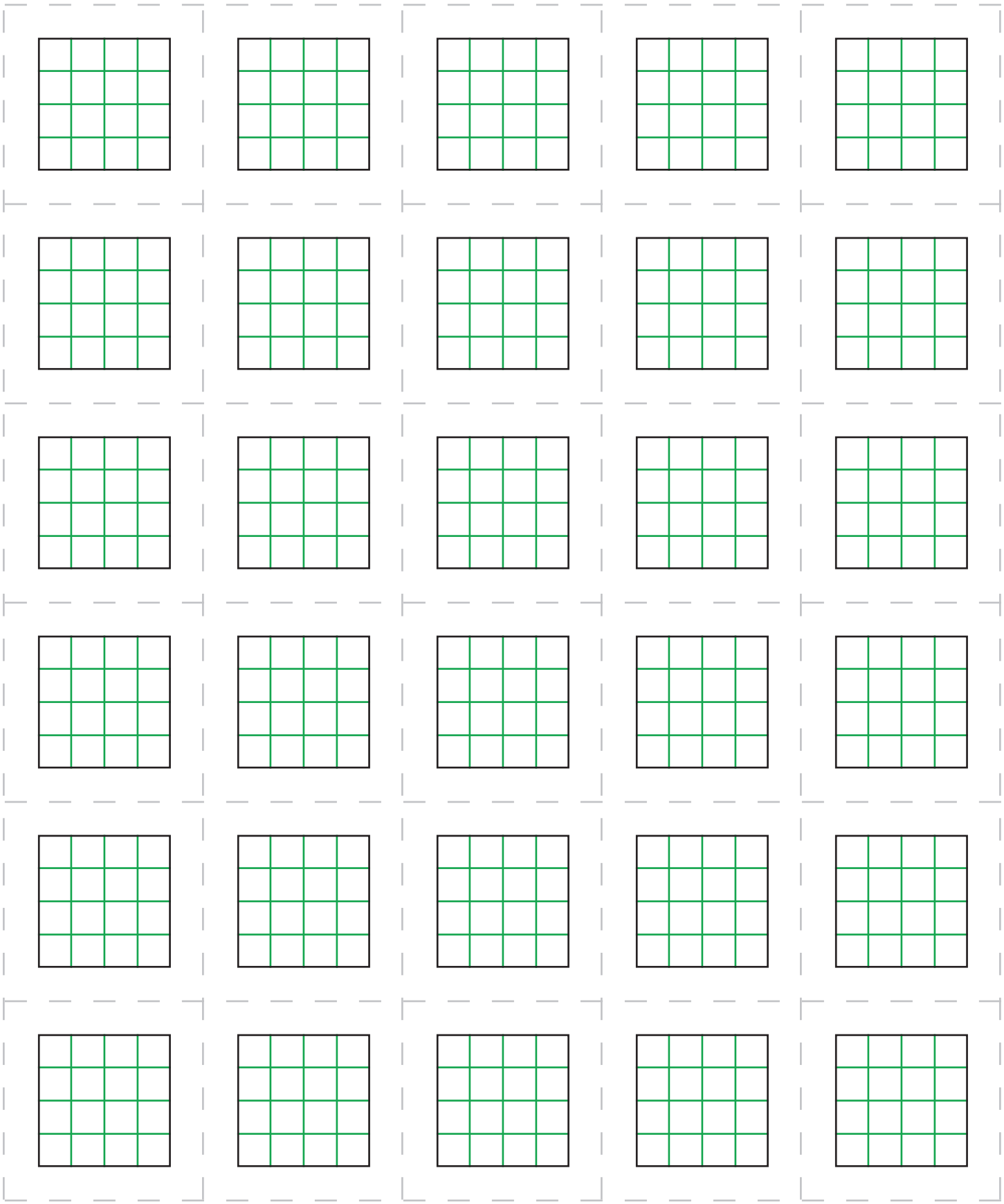
- Encourage older students to create longer messages on graph paper or modified binder paper.

4 - A more complete ASCII chart has been included for such a look-up. See **Info Section**.

Initial Cut-Out Strips



Initial Cut-Out Squares



ASCII Alphabet in Binary

Letter	Binary
A	01000001
B	01000010
C	01000011
D	01000100
E	01000101
F	01000110
G	01000111
H	01001000
I	01001001
J	01001010
K	01001011
L	01001100
M	01001101
N	01001110
O	01001111
P	01010000
Q	01010001
R	01010010
S	01010011
T	01010100
U	01010101
V	01010110
W	01010111
X	01011000
Y	01011001
Z	01011010

ASCII Alphabet in Binary

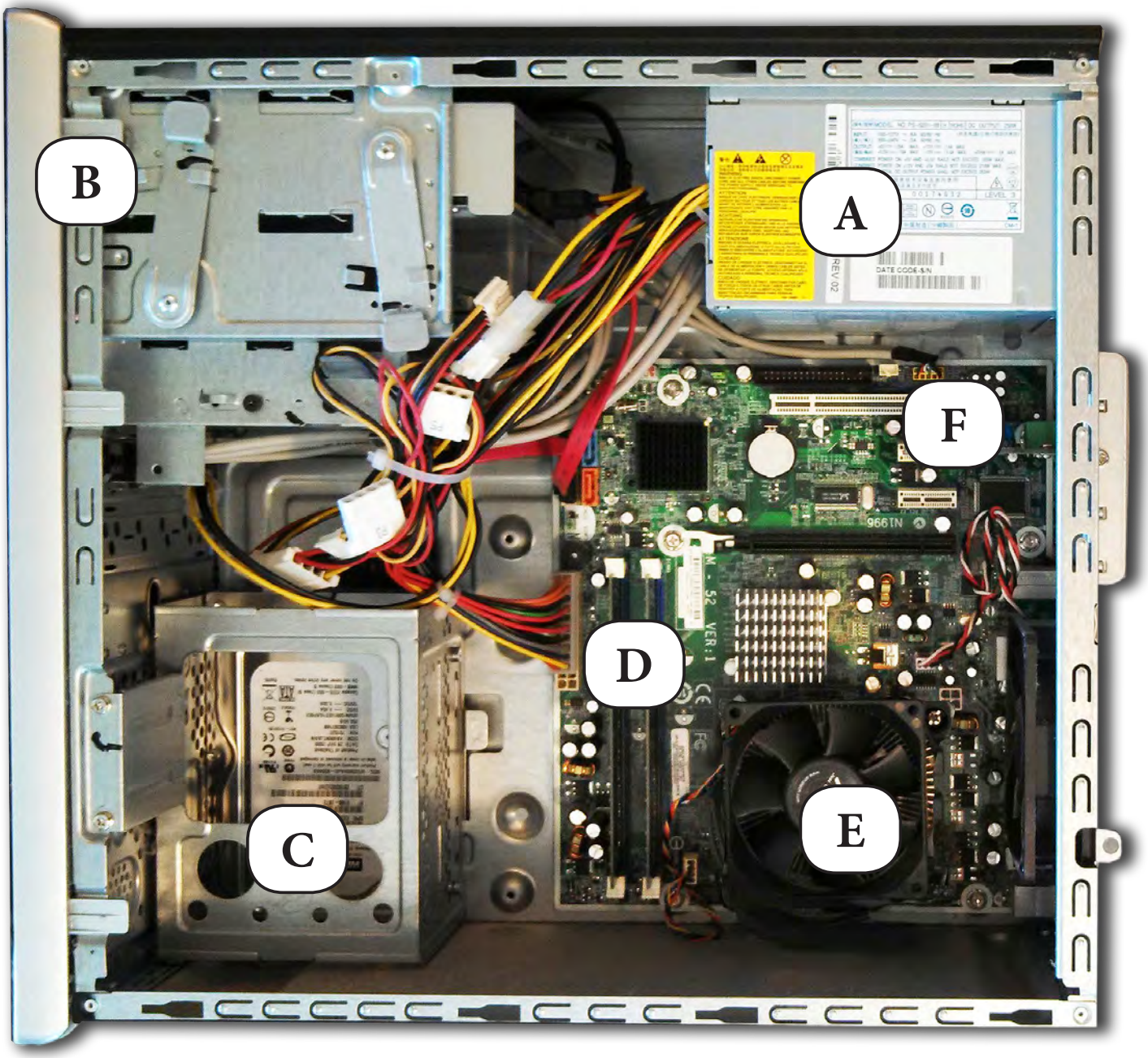
Letter	Binary
A	01000001
B	01000010
C	01000011
D	01000100
E	01000101
F	01000110
G	01000111
H	01001000
I	01001001
J	01001010
K	01001011
L	01001100
M	01001101
N	01001110
O	01001111
P	01010000
Q	01010001
R	01010010
S	01010011
T	01010100
U	01010101
V	01010110
W	01010111
X	01011000
Y	01011001
Z	01011010

ASCII Alphabet in Binary

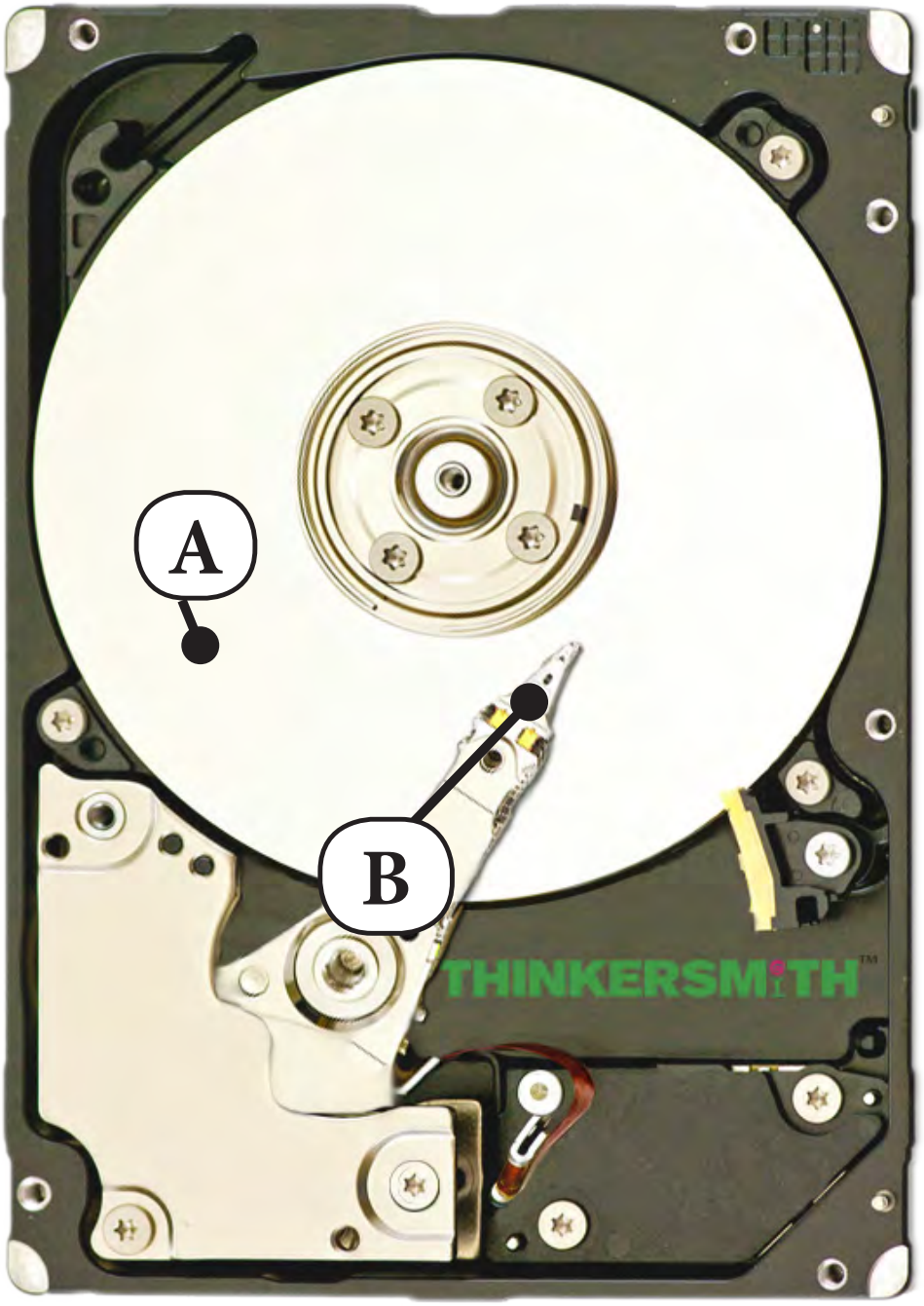
Letter	Binary
A	01000001
B	01000010
C	01000011
D	01000100
E	01000101
F	01000110
G	01000111
H	01001000
I	01001001
J	01001010
K	01001011
L	01001100
M	01001101
N	01001110
O	01001111
P	01010000
Q	01010001
R	01010010
S	01010011
T	01010100
U	01010101
V	01010110
W	01010111
X	01011000
Y	01011001
Z	01011010

ASCII Alphabet in Binary

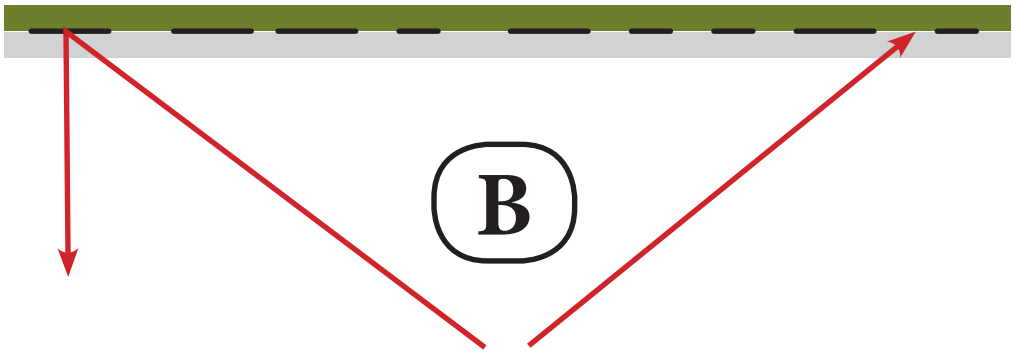
Letter	Binary
A	01000001
B	01000010
C	01000011
D	01000100
E	01000101
F	01000110
G	01000111
H	01001000
I	01001001
J	01001010
K	01001011
L	01001100
M	01001101
N	01001110
O	01001111
P	01010000
Q	01010001
R	01010010
S	01010011
T	01010100
U	01010101
V	01010110
W	01010111
X	01011000
Y	01011001
Z	01011010



A) Power Supply *B) Expansion Bays for CD/DVD, etc.* *C) Hard Drive*
D) RAM slots *E) Processor/CPU with Fan* *F) Expansion Card Slots*



A) *Spinning Storage Platter* *B) Read/Write Head*



A) Front View B) Side View with Laser Direction

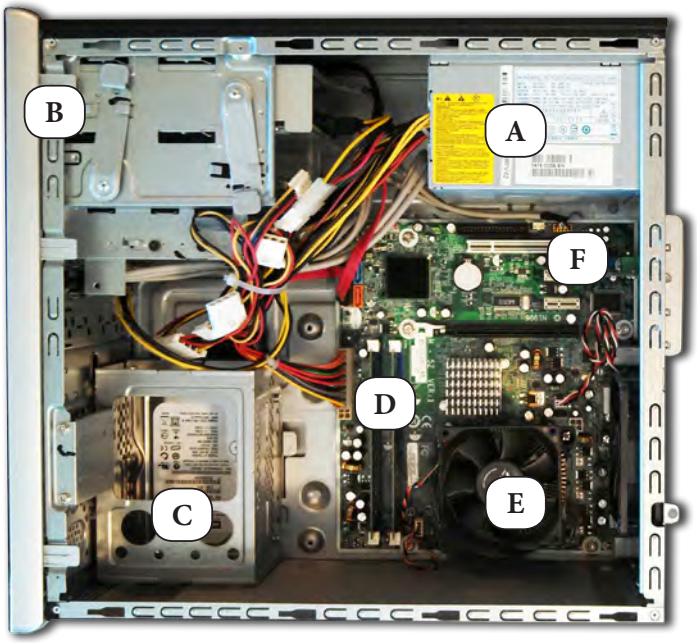
Sample Script

Light: "How many of you have ever seen the inside of a computer? Can you describe what it looks like?

(Pause for answers. Eventually, someone will get to the wires.)

Yes, that's right. There are lots of wires inside of a computer. Those wires do different things. Some of them carry power around the machine, and others carry information. Let's take a look:

(If you have a real computer handy, feel free to remove the back and put it on display. Otherwise, you can use a printout or projection of Image 1 in the **Computer Image Pack**.)



Do you see that big silver box with all of the wires under the label A? That's the **Power Supply**. It converts the power that comes from the wall outlet into something that the computer can use. It then feeds that power to the rest of the computer parts.

Next, look at D. That is the **RAM**, which stands for "Random Access Memory". This is a really fast storage space where the computer can keep information that it uses often. When the RAM is full, or when you are done using information for a while, the info can get stored to the **Hard Disk Drive**, which we see hidden under the C label.

(There is a picture of the inside of a hard disk drive inside the **Computer Image Pack** labeled Image 2).

Light Continued:

Now, if you look at label E, you'll see what looks like a tiny fan. That is where the **CPU** (Central Processing Unit) sits. Sometimes called the **Processor**, the CPU is full of wires — more than three million of them — each 1/100th the thickness of a human hair. The processor gets extremely hot, since those wires leak a little bit of electricity every time an instruction is performed, and the average processor in 2013 performs about 178 million instructions per second. You will almost always see a fan near, or even on top of, the processor trying to keep it cool.

Next:

This is where the **Light** version of the script concludes. To save time or intensity, you can skip ahead to the **Wrap-Up**.

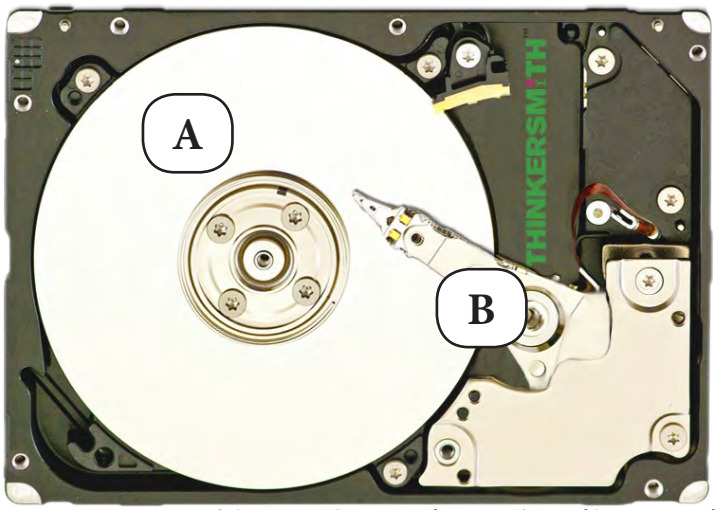
Detailed :

A processor sends information flying through each of its wires and through **Transistors**, which are like wires with gates on them. Transistors either let electricity through or don't, depending on what the instructions tell them to do. Those instructions, which were originally written in a language that humans can understand, get encoded to ones and zeros, and those ones and zeros become patterns of off and on inside the CPU.

By the time your CPU has performed a command, like moving the mouse cursor, multiplying numbers, or printing to the screen, the command has become nothing more than an encoded combination of on and off.

Let's talk about how those encoded messages get stored for use later.

Take for example, the hard disk drive (Image 2 in **Computer Image Pack**).

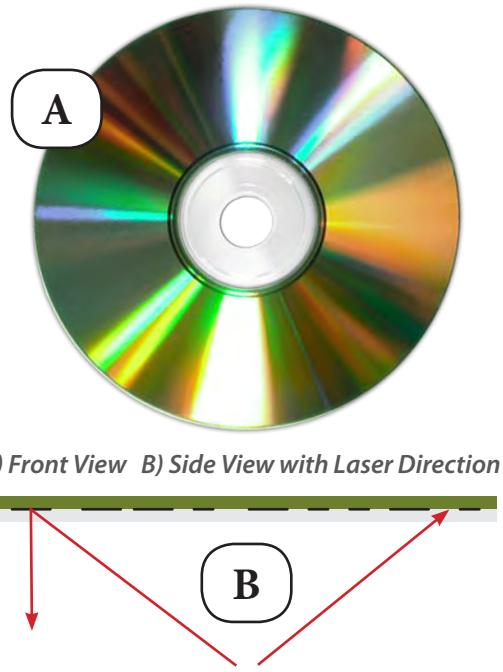


A) Spinning Storage Platter B) Read/Write Head

Detailed Continued:

The two most important elements on a hard disk drive are the spinning storage platter (A) and the read/write head (B). The head actually magnetizes sectors of the spinning platter to store information as a combination of magnetic positive and magnetic negative bits. This is the hard disk's version of ones and zeros.

Another example of storing data using binary comes in the form of **CDs** and **DVDs** (Image 3 in the **Computer Image Pack**).



A) Front View B) Side View with Laser Direction

CDs, DVDs, and even Blu-Ray disks work by directing a laser either back to a laser-reader, or off course so that the light never returns. The presence or absence of reflected light is read as a binary encoding in the same way as on and off or one and zero.

Wrap-Up:

Now that we understand a little more about binary, it's time to understand how you can use it to encode information. For this activity, I will be handing everyone an ASCII Encoder Card to help us go back and forth between letters and their binary code.

Continue:

This is where the lesson becomes a group activity. From here, you can pick up at the beginning of the **Example** section and work your way through the samples and into the art project.

If students have additional questions on hardware, binary, or encoding, encourage them to do some research and present their findings to the class in a follow-up session.

Info Section

Representation Introduction:

It can be difficult to fathom how binary can be used to represent all forms of information that a computer can store. Some encoding methods may seem more intuitive than others, but here is a brief sampling of how each of the popular media types might be transformed.

Numbers:

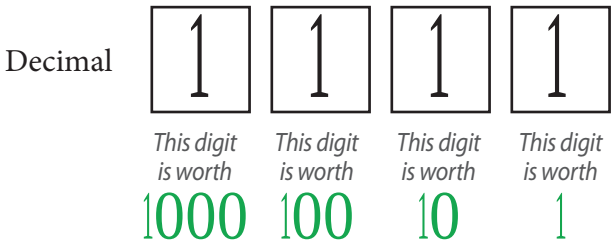
When mathematicians first learn about binary, it usually happens in reference to the numbers that we already know. Most of the current human population counts in ***decimal*** (a method that uses a base of ten options, 0-9, for each digit) likely because we have 10 fingers. Binary, on the other hand, uses a method where there are only two options for each digit, 0 & 1.

To get an idea of the difference, take this line of dots:

If you were going to display the number in decimal that you had counted to at any given moment, you would be fine with a single digit display from zero all the way through nine.

but as soon as you tried to count the next dot, you would encounter a problem. You have run out of original characters (0-9) to symbolize your number of dots. Fortunately, it is quite acceptable to add another digit to the left that keeps track of how many times you have cycled through the digits to the right.

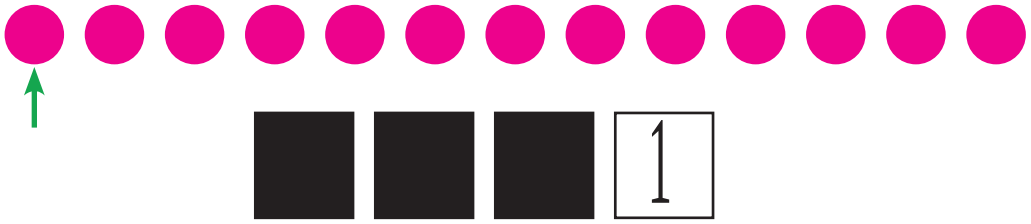
This will keep you happy up through 99 dots. At that point, you would need to add a third digit, and so on. Each digit increases in value as you continue to build to the left.



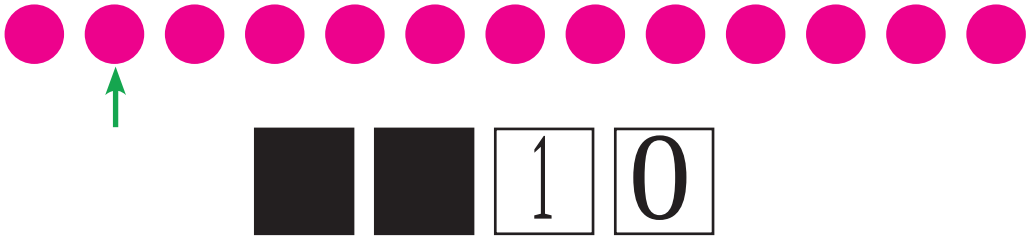
Those values happen by multiplying the base (in this case 10) times the value of the digit to the right. So, every digit is worth 10 times more than the one before.

Numbers Continued:

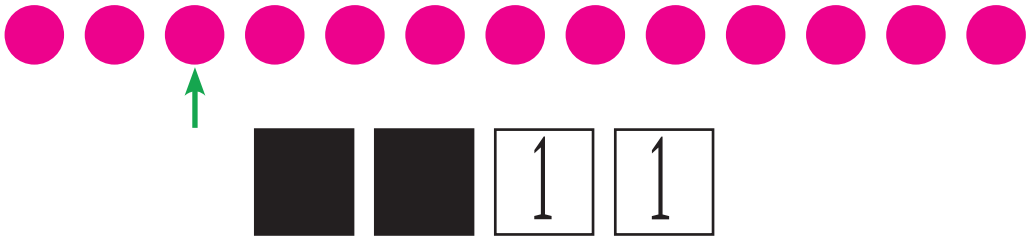
Now imagine that you were counting the same dots with only two options for each digit, 0 and 1.



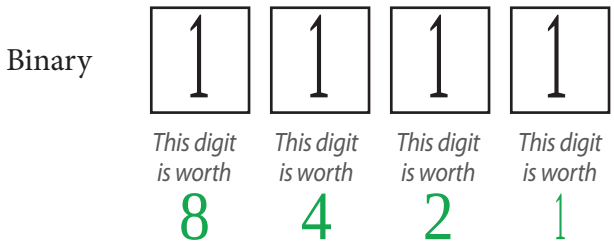
You don't get very far before you need another digit.



It's almost unbelievable, but the decimal number 2 is a two digit number in binary! So is 3.



From here, binary representations of decimal numbers get very big, very fast. If you take a look at the digit values below, you will see why.



The largest decimal number that you can represent with four binary digits is 15 in decimal!

Just as in decimal, the binary digit values happen by multiplying your base (in this case 2) times the value of the digit to the right. So, every digit is worth only 2 times more than the one before.

That is how numbers are represented in binary.

Numbers Continued:

Counting in binary has a very predictable rhythm. Take a look at the binary representations of the first 50 decimal numbers.

Decimal	Binary	Decimal	Binary
0	000000	25	011001
1	000001	26	011010
2	000010	27	011011
3	000011	28	011100
4	000100	29	011101
5	000101	30	011110
6	000110	31	011111
7	000111	32	100000
8	001000	33	100001
9	001001	34	100010
10	001010	35	100011
11	001011	36	100100
12	001100	37	100101
13	001101	38	100110
14	001110	39	100111
15	001111	40	101000
16	010000	41	101001
17	010001	42	101010
18	010010	43	101011
19	010011	44	101100
20	010100	45	101101
21	010101	46	101110
22	010110	47	101111
23	010111	48	110000
24	011000	49	110001

If you look carefully at the sequential binary numbers, you'll see that the bits alternate very smoothly. The right-most bit alternates with every iteration. The second bit from the right alternates every second iteration. The third bit alternates every fourth iteration, the fourth bit every eighth, and so on, with each further bit flipping after two times the iterations of the one before it.

Letters:

Once you understand numbers, letters are very easy to grasp. In ASCII, letters are assigned to numbers and those numbers are encoded into binary. As it happens, numbers are assigned sequentially so letters that differ by one spot in the alphabet are also next to each other numerically.

Letter	ASCII Code	Binary
A	065	01000001
B	066	01000010

Colors:

Representing colors inside the computer happens with combinations of red, green and blue. It's as if you have three light bulbs, one in each of these colors, and you can change the color of your entire room by changing how brightly each shines. Each light bulb can go from zero (off) to 255 (all the way on). To understand the binary color representation, you just need to encode the level of red, green, and blue as a 24 bit binary number.

R: 255 G:0 B:0 = 11111111 00000000 00000000

R: 0 G:255 B:0 = 00000000 11111111 00000000

R: 0 G:0 B:255 = 00000000 00000000 11111111

R: 255 G:255 B:0 = 11111111 11111111 00000000

R: 0 G:255 B:255 = 00000000 11111111 11111111

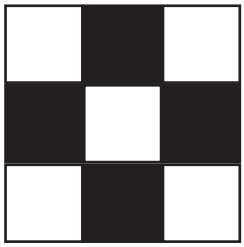
Images:

To understand how it's possible to encode an image as zeros and ones, you must first understand how ***pixels*** work. Pixels are the individual squares that make up a computer screen. Each pixel has control over it's own colors (the same red, green, and blue from above).

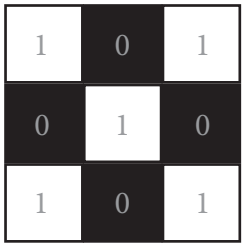
To encode an entire picture there are several levels of color options. The easiest way to begin is in black and white. When encoding a black and white image, there are only two possible options for each pixel, off and on. Those options are often encoded as zero and one, respectively.

Images Continued:

Take this sample image:

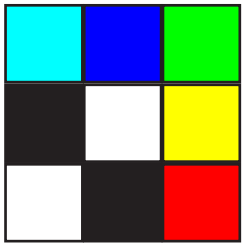


It could be thought of as a nine pixel screen that is capable of showing two options in each pixel. If off (black) is 0, and on (white) is 1, then you could easily represent the image above using a code like this:

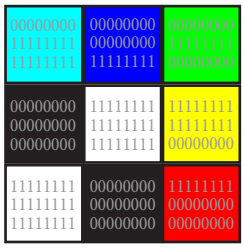


101 010 101

A similar technique can be used on images with full web color. Imagine a screen like this:



Now, instead of having only two choices for each pixel, all web color options are available, requiring the 24 bit representation that goes along with them. Encoding these images would look a little more like this:

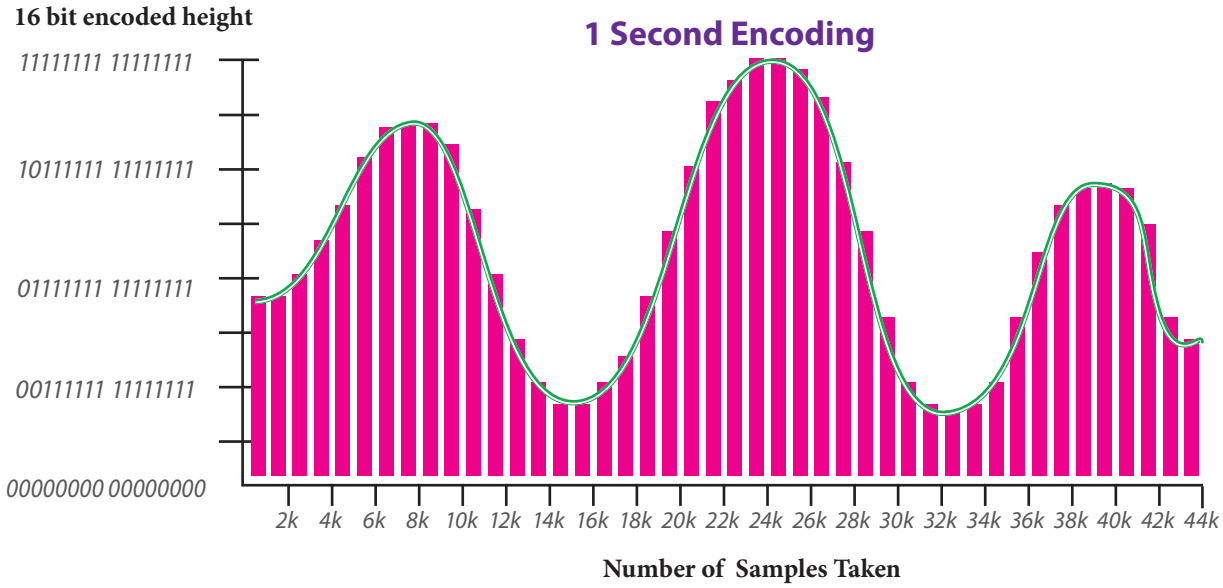


00000000 11111111 11111111 00000000 00000000 11111111 00000000 11111111 00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111 11111111 00000000 11111111 11111111 11111111 00000000 00000000 00000000 11111111 00000000 00000000

Music:

The binary representation of sound is quite different from the representation of images. Colors and pixels won't help here.

Instead, digital recordings are created by taking samples of sounds at a specific rate. "CD quality" sound has a digital sampling rate of 44.1 kilohertz, which is 44,100 times per second. This means that 44,100 times each second, a program measures and records the height of the song's sound wave, then translates the height to a binary representation. The MP3 translation uses 16 bits for each sound channel. As such, a CD quality, stereo MP3 that is 4 minutes long would take up roughly 338,688,000 bits!



1 second encoding =

16 bits per channel × 2 channels × 44,100 samples

= 1,411,200 bits each second

4 minute song =

1,411,200 bits per second × 60 seconds × 4 minutes

= 338,688,000 bits in the song

Bits Explained: What 's with this bit business, anyway?

The word bit is essentially a contraction of the words "binary" and "digit". A bit is merely one slot where a binary option can be placed.

0

A one bit number

0 1 1 0 1 0 0 1

An eight bit number

For ease of speaking, certain quantities of bits have been given catchy nicknames. For instance, four bits is also called a "nibble" or "nybble", eight bits is called a "byte", and eight thousand bits is known as a "kilobyte"[5].

Below is a list of terms and the original bit quantities associated with them:

#Bits	Symbol	Term
1	b	bit
4	n	nibble
8	B	byte
1000	kB	kilobyte
1000^2	MB	megabyte
1000^3	GB	gigabyte
1000^4	TB	terabyte
1000^5	PB	petabyte
1000^6	EB	exabyte
1000^7	ZB	zettabyte

5 - Using one convention, the word kilobyte represents 1000 bits, which you may see written as "1 kB". The word kibibyte, written "1 KB" often refers to kilobinary, which is $2^{10} = 1024$ bits.

Binary Math:

Math with binary is actually more simple than decimal math, due to the fact that you are dealing with ones and zeros. Other than that, the arithmetic is quite similar. Note the example:

$$\begin{array}{r} 5 \\ +\ 3 \\ \hline 8 \end{array} \qquad\qquad \begin{array}{r} 101 \\ +\ 11 \\ \hline 1000 \end{array}$$

You'll notice from the **Numbers** section that 8 in decimal does, in fact, equal 1000 binary. Subtraction, multiplication and division follow suit.

Binary math can be helpful when attempting to decode ASCII letters. If, for example, you memorize just one encoding (like: A = 0100 0001) and you are given another encoded letter (suppose it's 0100 1101) then you can do some binary subtraction to figure out what that letter is. Example:

$$\begin{array}{r} 0100\ 1101 \\ -\ 0100\ 0001 \\ \hline 1100 \end{array}$$

With the table from the **Numbers** section, you will note that 1100 in binary is 12 in decimal. That means that the number provided is 12 after the letter A.

Just like that, you can calculate that the encoded number must be an 'M'.

Expanded ASCII Table:

!	00100001	A	01000001	a	01100001	¢	10100010
"	00100010	B	01000010	b	01100010	£	10100011
#	00100011	C	01000011	c	01100011	¤	10100100
$	00100100	D	01000100	d	01100100	¥	10100101
%	00100101	E	01000101	e	01100101	¦	10100110
&	00100110	F	01000110	f	01100110	§	10100111
'	00100111	G	01000111	g	01100111	¨	10100111
(	00101000	H	01001000	h	01101000	©	10101001
)	00101001	I	01001001	i	01101001	ª	10101010
*	00101010	J	01001010	j	01101010	«	10101011
+	00101011	K	01001011	k	01101011	¬	10101100
,	00101100	L	01001100	l	01101100	®	10101110
-	00101101	M	01001101	m	01101101	¯	10101111
.	00101110	N	01001110	n	01101110	°	10110000
/	00101111	O	01001111	o	01101111	±	10110001
0	00110000	P	01010000	p	01110000	²	10110010
1	00110001	Q	01010001	q	01110001	³	10110011
2	00110010	R	01010010	r	01110010	´	10110100
3	00110011	S	01010011	s	01110011	µ	10110101
4	00110100	T	01010100	t	01110100	¶	10110110
5	00110101	U	01010101	u	01110101	·	10110111
6	00110110	V	01010110	v	01110110	¸	10111000
7	00110111	W	01010111	w	01110111	¹	10111001
8	00111000	X	01011000	x	01111000	º	10111010
9	00111001	Y	01011001	y	01111001	»	10111011
:	00111010	Z	01011010	z	01111010	¼	10111100
;	00111011	[	01011011	{	01111011	½	10111101
<	00111100	\	01011100	|	01111100	¾	10111110
=	00111101	]	01011101	}	01111101	¿	10111111
>	00111110	^	01011110	~	01111110		
?	00111111	_	01011111	€	10000000		
@	01000000	`	01100000	¡	10100001		

COMPUTER SCIENCE
Education Week
WWW.CSEDWEEK.ORG

For more lessons, please visit
www.thinkersmith.org