Running the Bases

Activity Overview:

This activity explores the number bases, base-2, base-10, base-16, and ASCII. Students use a TI-Innovator Hub, TI-RGB Array, and TI-Nspire to display on the array a byte in binary. The activity is strongly aligned with AP Computer Science Principles but has applications in many other classes.

AP Computer Science Principles Objectives incorporated in this Activity:

Enduring Understandings:

- EU 2.1 – A variety of abstractions built on binary sequences can be used to represent all digital data.
- EU 2. – Multiple levels of abstraction are used to write programs or create other computational artifacts.

Learning Objectives:

- LO 2.1.1 – Describe the variety of abstractions used to represent data.
- LO 2.1.2 – Explain how binary sequences are used to represent digital data.
- LO 2.2.1 – Develop an abstraction when writing a program or creating other computational artifacts.

Essential Knowledge:

EK2.1.1A, EK2.1.1B, EK2.1.1C, EK2.1.1D, EK2.1.1E, EK2.1.1F, EK2.1.1AG

EK2.2.1.C, EK2.2.1.D, EK2.2.1.F

Activity Background:

Tool building is an essential aspect of humanity, and the computer is the ultimate modern tool. Humans have big brains, five senses, agile fingers, vocal cords and eat food; computers don't have or do any of these. Computers have integrated circuits, cameras, microphones, speakers, displays, keyboards, mice, and consume electricity; humans don't have or do any of these. How do our human brains communicate with the computer's integrated circuits? The computer has two fundamental components, software and hardware. The essential component of hardware is the integrated circuit, while the essential component of the software is logical operations on binary numbers.

The transistor, invented in 1947 by William Schockley at Bell Labs, is the basic building block of all modern digital devices. The transistor is an electronic device that can be turned "on" when an electrical voltage is applied to a contact within the device called the gate and turned off when zero Volts is applied to the gate. When the transistor is "on," electricity can flow through a path within the device, and when "off", electrical flow is blocked. The transistor functions like a water faucet; the difference is, it controls electricity instead of water. Since the transistor has two states of either "on" or "off," these two states may be abstracted with the two numbers zero and one of the binary number system. Alan Turing, a mathematician, working in the 1940s, devised a way of solving any logical problem using operations on just these two digits. Shockley and Turing, along with many others, created the hardware and software of the modern computer. One problem facing the early computer industry was the enormous size of computers. One of the first working computers in the late 1940s was ENIAC, it occupied several rooms and had very little computing power compared to today's computers and smartphones.

In 1958 Jack Kilby, while working at Texas Instruments, invented a method for building multiple transistors on a small wafer of silicon crystal called an integrated circuit or IC commonly called a "chip." Modern integrated circuits can have over 100 million transistors on a single crystal chip of silicon. Some powerful computers may have hundreds of these ICs on the circuit boards inside the computer case. This complex system of integrated circuits and circuit boards is the hardware of the computer. The "hardware" of a computer does nothing until it is programmed by "software." "Software" sets all of the transistors within the IC to either on or off according to the logic of the program. When a computer is programmed, a human abstracts a problem using a programming language such as python, which ultimately becomes a large set of binary numbers. When the user executes a program, the gates of the transistors within the chips are all switched to either on or off. Then electricity flows through the transistors in a unique and logical path, determined by the settings of all the gates. The answer to the problem is then interpreted from the resulting unique flow of electricity through the transistors.

Additional hardware devices are required to enter a program into the computer's ICs and to interpret the result of the execution. These devices are called input/output, or I/O, peripherals. Examples include monitors, speakers, mice, keyboards, and networking connections. All of these devices either consume or produce a stream of binary numbers. For example, a monitor requires several binary numbers to color a small section of the screen called a pixel. The digital video card of a computer is constantly streaming pixel coordinates and color values to the monitor to paint the screen in a way meaningful to humans. When a user watches a streaming 4k video movie, there is a tremendous amount of abstraction and flow of visual and aural information into and out of the binary number system.

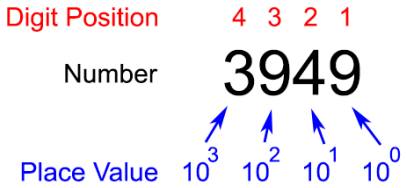
The binary number system is the beginning of all software that programs hardware. The state of an individual transistor, represented by either a 0 or 1, is called a bit. Individual bits are assembled into logical groups of four, called a nibble. Further grouping of two nibbles is referred to as a byte, and two bytes are referred to as a word. These groupings of binary digits are the fundamental units of software. The eight bits of a byte can be arranged in 2^8 or 256 different ways. Thus, a byte can represent a decimal number from 0 to 255. Often programmers may further abstract a nibble using the hexadecimal, base 16, number system. Since hexadecimal number system possesses 16 or 2^4 different digits, a single hexadecimal number can represent 4 bits and a byte can be represented as a two-digit hexadecimal value. A two-digit hexadecimal number is easier to read than an eight-bit binary number. Another layer of abstraction can occur with the ASCII representation of binary numbers. Two hexadecimal digits represent a single familiar ASCII character (see ASCII set table inclded in resorces). Using the ASCII character set provides a method for written text to be represented as binary numbers. A word processing file is a digital representation of the written language contained within the document file.

When a word processing file is created, the clicking of a key on the computer's keyboard is the initial action that sets the state of eight transistors within the computer's integrated circuits, called a register. When a key is pressed down, a voltage or ground is applied to each gate of the register's eight transistors changing each transistor's state to either on or off. The state of the eight transistors within the register represents a single byte of data that can be abstracted as an ASCII character. In this way, each key on the keyboard corresponding to one ASCII character. All characters of the alphabet, number, and punctuation marks are represented in this way. As an example, when the letter "A" is pressed, the register's transistors are set to the state of "off-on-off-off-off-off-off-on", and this abstracts to the binary number "01000001" which abstracts to the decimal number 65 which then abstracts to the ASCII character "A."
Once the byte is set within the register, the state can be passed to a location within memory transistors to free keyboard to read another keypress. A word processing software is a program that can manipulate, transform and store all of the bytes of data generated by the clicking of keys during the human's writing composition.

The following six coding challenges explore the binary, decimal, hexadecimal, and ASCII abstractions of digital data. Students will have hands-on experience of setting the color and location of a pixel within a display in a way similar to how video is streamed and also to generate text using binary, decimal, hexadecimal and ASCII characters.

Number Bases and ASCII Characters:

The base of a number system represents how many different digits or symbols are used to represent any number. The position of the digit within the number determines its value.



Base-10, decimal, there are ten digits, zero through nine. The value of each digit is found by multiplying that digit by ten raised to the power of its position minus one. For example, 3949 is equal to:

$$3 \times 10^3 + 9 \times 10^2 + 4 \times 10^1 + 9 \times 10^0 = 3949$$

Base-2, binary, there are two digits zero and one. The value of each digit is found by multiplying that digit by two raised to the power of its position minus one. For example, 1001 is equal to:

$$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9$$

Base-16, hexadecimal, there are sixteen symbols zero through F. The first five letters of the alphabet are used as digits to represent the values of 10 thru 15. The value of each digit is found by multiplying that digit by 16 raised to the power of its position minus one. For example A0F9 is equal to:

$$A \times 16^3 + 0 \times 16^2 + F \times 16^1 + 9 \times 16^0$$
$$10 \times 16^3 + 0 \times 16^2 + 15 \times 16^1 + 9 \times 16^0 = 41209$$

ASCII, there are two hundred and fifty-six different alphanumeric characters. These characters are in a lookup table and each character has an associated number. For example, the character “A” has a lookup number of 65 and “a” has a lookup number of 97. There are two functions on the calculator to look up these numbers. In a calculator page, the command ord(“A”) will give 65 and ord(“a”) will give 97. The complimentary function char(65) will give “A” and char(97) will give “A”.

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
32	20	40	[space]	64	40	100	@	96	60	140	`
33	21	41	!	65	41	101	A	97	61	141	a
34	22	42	"	66	42	102	B	98	62	142	b
35	23	43	#	67	43	103	C	99	63	143	c
36	24	44	$	68	44	104	D	100	64	144	d
37	25	45	%	69	45	105	E	101	65	145	e
38	26	46	&	70	46	106	F	102	66	146	f

Command Background:

Command	Example	Behavior
CONNECT RGB	`Send "CONNECT RGB"`	Associates a TI RGB Array with a software object named RGB.
SET RGB <pixel # R G B>	`Send "SET RGB 0 255 0 0"`	Turns on the 0 position pixel to red.
SET RGB ALL < R G B>	`Send "SET RGB ALL 0 255 0"`	Turns on ALL of the pixels to green.
SET RGB CLEAR	`Send "SET RGB CLEAR"`	Turns off all of the array's pixels.
Wait<time>	`Wait 0.1`	Pauses the program execution for 1/10 second
Char<ascii number>	`Char(97)`	Returns the ASCII character, "a," associated with the denumber 97.
<"string 1"> & <"string 2">	`msg:= "H" & "i"`	The ampersand operator, &, joins the "H" and "i" together and stores result to the variable msg. Displaying msg will show "Hi" on the screen.
DispAt <line #> , <"text"> , <variable name>	`DispAt 3, "BASE 16 = ", hex`	When variable hex has a value of 255, the following line is displayed on the calculator: Base 16 = 0hFF

For <counter >,<start>,<end>,[<step>]
<statements>
EndFor
	`For n,1,10,1`
	`  DispAt 3,n`
	`EndFor`
		Runs a loop 10 times, starting at 1 and ending at 10 with an increment of 1. Executes the statement in the block each cycle.

If <Boolean expression> Then
<statements>
EndIf
	`If dec≥2^3 Then`
	`  Send "SET RGB 3 255 0 0"`
	`  dec:=dec-2^3`
	`EndIf`
		If the value stored in dec is greater or equal to 2^3, then turn on the third pixel red and decrement the value in dec by 2^3.

eval(<variable>)	`Send "SET RGB eval(n) 255 0 0"`	Evaluates the variable named n and substitutes the value into the command string.

`Define myProgram(parameters)=`
`Prgm`
`  <statement>`
`EndPrgm`
	`Define myProgram(row, msg)=`
	`Prgm`
	`  DispAt row, msg`
	`EndPrgm`
		Running myProgram (3, "hello") on the calculator page will display the "hello" on line 3 of the calculator

PROJECTS WITH THE TI-INNOVATOR™ SYSTEM (TI-NSPIRE CX) **The TI-RGB Array**

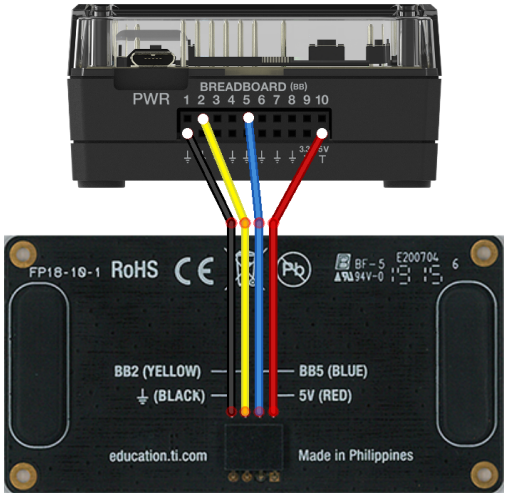
Supplies and Equipment:

- TI Nspire CX or CX II
- Unit to Unit cable
- TI Innovator Hub
- TI RGB Array

Setup Project Model:

1. Carefully Insert the TI RGB Array wire-harness into the array receptacle on the back of the board, noting the following color coding:
   a. 5V – Red
   b. BB5 – Blue
   c. BB2 – Yellow
   d. (ground) ⏚ – Black

2. Carefully Insert the opposite end of the wire harness into the following pins on the TI Innovator Hub.
   a. Red – 5V
   b. Blue – BB5
   c. Yellow – BB2
   d. Black – ⏚ (ground)



The Student Activity

Sit in small groups of 2-3 partners with your calculators and share a TI-Innovator Hub with TI-RGB Array attached.

Challenge 1: Write a program to turn pixel zero to the following colors:

- red
- yellow
- Try to make it very dim red.
- Try different pixels numbers.
- Try to display a pattern for five seconds and then clear the array.

The Teacher Activity

Organize groups of two to four students, depending on the available equipment. Every student in the group should have their calculator to write and test their programs on the group shared Hub and Array. When a student wants to test a program, they plug the Hub with Array into their calculator and test. When finished, the Hub and array are detached and passed to another group partner who is ready to test.

Guidance during challenge 1:

1. Ensure students connect the TI-RGB Array correctly.
2. SET RGB has four parameters; the pixel number and the red, green, and blue colors the pixel emits.
3. The intensity of the color is given by the magnitude of the value. For example, bright red is 255 while dim red could be 20.

```
Define c1()=
Prgm
Send "CONNECT RGB"
Send "SET RGB 0 255 0 0"
Send "SET RGB 1 255 255 0"
Send "SET RGB 2 20 0 0"
Wait 5
Send "SET RGB CLEAR"
EndPrgm
```

Challenge 2: Use a For-EndFor loop to Write a program to turn on and then off (blink) each pixel in numeric order. Choose an appropriate wait time between each pixel. Choose your favorite color!

Guidance during challenge 2: Be sure to point out pixel position numbering on TI-RGB Array circuit board. The eval() command can be used to set the pixel number based on the loop index variable.

```
Define c2()=
Send "CONNECT RGB"
For n,0,15
  Send "SET RGB eval(n) 255 128 0"
  Wait 0.5
  Send "SET RGB eval(n) 0 0 0"
  Wait 0.5
EndFor
EndPrgm
```

Challenge 3:
Assemble a Flippy-Do and attach to the TI_RGB Array. Complete the Flippy-Do Worksheet.

Guidance during challenge 3: The Flipply-Do activity is intended to be an introduction to binary numbers. Students are encouraged to discover a pattern first and then learn the number system later. Once students understand the binary number system, they program the TI_RGB Array to display several 4-bit binary numbers and check their work with the Flippy-Do.

Example code in "Running the Bases" companion program.

Challenge 4:
Run the "Binary Blocks" program and observe the method that is used to convert a decimal number into its binary representation.

Write a program using If-Then-EndIf statements to convert a decimal number into a binary number using the same method as in the "Binary Blocks" program.

Guidance during challenge 4: The goal of the "Binary Blocks" activity is for students to observe how any decimal number can be represented as a sum of powers of two.
Essential observations are:

- the method must proceed from the greatest power of two to the least power of two.
- a decision is made for each power of two asking "does this block fit ?"
- the sum of the binary block lengths is equal to the overall length of the decimal block.
- a decimal number can be represented as a sum of powers of two.
- The algorithm for the "Binary Blocks" activity should be used to solve challenge 4.

Example code in "Running the Bases" companion program.

Challenge 4 optional extension:
Write a program that accepts a decimal value from 0 to 2^{16} -1 (65535) as a parameter and uses a loop to display the binary representation or the TI-RGB Array.

Guidance during challenge 4 optional extension: A byte is 8 bits, and 2 bytes is 1 word. There are 16 pixels on the array,16 bits can represent as 0b11111111 11111111 other words 2^{16} - 1 (65535) decimal values can be displayed on the array extending the method from C4.

Steps in the program:

1. Associate the hardware with the software object.
2. Turn off all pixels on the array.
3. Loop through the 16 pixels (0 – 15) on the array. The position number printed on the board is also the power of 2 in the binary display.
4. Start with the most significant bit, n=15, test If 2^n is a term in decimal value, if it is, turn on the pixel at position n and subtract 2^n from decimal value. The loop will continue testing each subsequent power of 2 until the least significant bit in the zero position.

```
Define c4_extension(dec)=
Prgm
Send "CONNECT RGB"
Send "SET RGB ALL 0 0 0"
For n,15,0,-1
  If dec≥2^n Then
    Send "SET RGB eval(n) 255 0 0"
    dec:=dec-2^n
    wait .1
  EndIf
EndFor
EndPrgm
```

Challenge 5:
Write a program that accepts three variables; R, G, and B. Use those three variables to set all of the pixels on the TI-RGB array.

Guidance during challenge 5:
Please read the guidance for the extension below for background on color. This challenge uses three separate bytes for setting the color. The optional extension is a more challenging program that requires parsing a hexadecimal triplet into three separate bytes.

```
Define c5(r,g,b)=
Prgm
Send "CONNECT RGB "
Send "SET RGB ALL eval(r) eval(g) eval(b)"
EndPrgm
```

Challenge 5 optional extension:
Write a program that accepts a hexadecimal triplet and converts the triplet into three separate R, G, and B bytes. Use the R, G, and B bytes to set all of the pixels on the TI-RGB Array.

Guidance during challenge 5 optional extension:
A single byte ranges in values from 0 to 11111111 in binary, or 0 to 255 in decimal, or 0 to FF in hexadecimal. Since one byte is two hexadecimal digits, six hexadecimal digits are three bytes, referred to as a hex-triplet. RGB color on computers is represented with a single hex-triplet. Each byte, two hex digits, of the hex-triplet corresponds to one of the color channels, red, green, and blue. The most significant byte, furthermost left, corresponds to red. The middle byte is green, while the least significant byte is blue. Sometimes this color depth is referred to as 24-bit color because of 28 x 28 x 28 = 224 and represents 16777216 different colors! To limit the number of colors on the WWW, a subset of the possible colors is standardized as the X11 named colors space. Their hex-triplet code references these named colors.

Since the TI-RGB Array command Send "SET ALL 255 128 64" has separate R, G, and B values, the six-digit hex-triplet, must be separated into the three individual bytes. This method of setting a color on the RGB Array, is also how color TV works. When a movie is streamed in 4K digital format, the location of the pixel on the screen and the 6 hex-triplet are streamed to the monitor to turn each pixel on with the appropriate color to render one frame of the movie. The 4K standard is 3840 (4k) pixels x 2160 pixels, each with 24-bit color, a hex-triplet.

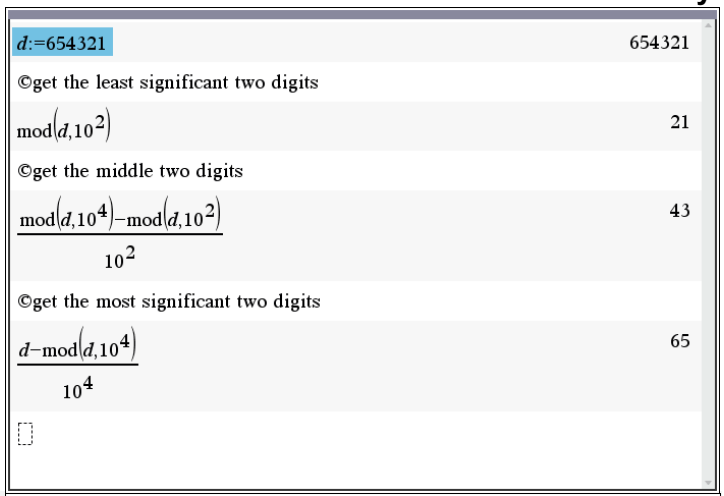
Modular Division gives the remainder of a regular

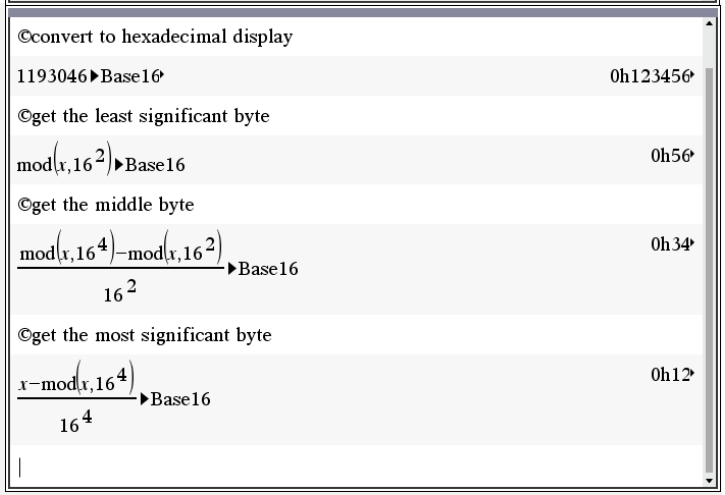
```
Define c5_extension(hex)=
Prgm
Send "CONNECT RGB"
r:=((hex-mod(hex,16^(4)))/(16^(4)))
g:=((mod(hex,16^(4))-
mod(hex,16^(2)))/(16^(2)))
b:=mod(hex,16^(2))
Send "SET RGB ALL eval(r) eval(g) eval(b)"
EndPrgm
```

arithmetic division. The modular division operator is %. TI-Nspire uses the mod() function to operate.For example: 10%3=1, 9%3=0, 8%3=2
An application of modular division is the conversion between 24-hour and 12-hour time:17:00%12= 5:00 PM. The modular division is a method to separate the hex-triplet into the three bytes needed to set the TI-RGB Array color.

On the first image to the right, look at the example calculations and notice how modular division is used to separate a six-digit decimal-triplet into three two-digit decimal numbers. Decimal is used as an introduction example. The following example will use hexadecimal.

On the second image to the right, the same method as used on the previous page is used on a six-digit hex-triplet to separate the three-byte number into three separate bytes. The example will separate the number 0h123456 into three separate bytes 0h12, 0h34, and 0h56. Remember, on the TI-Nspire; hexadecimal numbers are labeled with a 0h before the value.





PROJECTS WITH THE TI-INNOVATOR™ SYSTEM (TI-NSPIRE CX)

The TI-RGB Array

Challenge 6: Convert the following set of ASCII codes into a string of characters and display the result on the calculator screen.

code:={67,111,100,105,110,103,32,105,115,32,67,111,111,108,33}

Use the char() function to return the character associated with an ASCII number and the ampersand, &, to join a new character to the string.

Guidance during challenge 6: Students will need to copy the ASCII code given in the challenge into their program. The code should be stored as a list. In TI-BASIC, a list is defined with curly braces {} and the first indexed position is 1. To reference a value within the array, the list name, followed with square brackets and the index returns the value. Two functions are useful when working with ASCII. The char() function returns the ASCII character as a string of the corresponding ASCII value. Conversely, the ord() function returns an integer for the coresponding string character. An ASCII table is provided in the suplamentary materials of this activity and may be a useful student handout.

```
Define c6()=
Prgm
code:={67,111,100,105,110,103,32,105,115,32,
67,111,111,108}
length:=dim(code)
message:=""
For n,1,length
  message:=message&char(code[n])
EndFor
Disp message
EndPrgm
```

Challenge 6 optional extension: Use the Challenge 4 extension program to display each ASCII code on the TI-RGB Array as each ASCII code is converted and added to the message string. Allow a 1-second wait to slow the program for viewing as each character is converted.

code:={67,111,100,105,110,103,32,111,110,32,97,32,84,73,32,67,97,108,99,117,108,97,116,111,114,32,119,105,116,104,32,116,104,101,32,84,73,45,73,110,110,111,118,97,116,111,114,8482,32,72,117,98,32,97,110,100,32,82,71,66,32,65,114,114,97,121,32,105,115,32,70,117,110,32,58,41}

Guidance during challenge 6 extension: This challenge calls the program created in challenge 4 extension to turn on the TI-RGB array with the binary code coresponding to each character in the code array. As in the previous challengs, studens will need to copy the array given in the challenge to their program. In additon, the c4_extension program must coexist in the same problem as the c6_extension program in order to call one program from another. To call a program, enter the name of the program along with the parameter and the code within the called function will execute and return to calling program upon completion.

```
Define c6_extension()=
Prgm
code:={67,111,100,105,110,103,32,111,110,32,
97,32,84,73,32,67,97,108,99,117,108,97,116,1
11,114,32,119,105,116,104,32,116,104,101,32,
84,73,45,73,110,110,111,118,97,116,111,114,8
482,32,72,117,98,32,97,110,100,32,82,71,66,3
2,65,114,114,97,121,32,105,115,32,70,117,110
,32,58,41}
message:=""
For i,1, dim(code)
  message:=message&char(code[i])
  c4_extension(code[i])
  DispAt 3,message
Wait 0.1
EndPrgm
```