“Zeros”, “Ones”, and the Morse Code

Subject Area(s)	Number & operations, science & technology
Associated Unit	None
Associated Lesson	None
Activity Title	“Zeros”, “Ones”, and the Morse Code
Header	Insert image 1 here, right justified to wrap

Image 1
ADA Description: A portrait of Samuel Morse.
Caption: Figure1: Samuel Morse, a famous American inventor.
Image file name: morse_keyer1.bmp
Source/Rights: Copyright © National Inventors Hall of Fame Foundation, Inc.

Grade Level	9 (7-8)
Activity Dependency	None
Time Required	150 minutes
Group Size	3
Expendable Cost per Group	US$0

Summary

Students will learn about the binary number system computers use to store data, in which numbers are represented as sequences of zeros and ones. Using the Basic Stamp 2 (BS2) microcontroller, students will experiment with Morse code – one of the oldest communication methods still in use today. In the process, students learn about how only two signaling elements – the “dit” and the “dah”, or 0 and 1, can be used to encode arbitrarily complex messages. The activity introduces a number of useful commands used to program the BS2 microcontroller, as well as concepts of computer memory, data types, and data transfer for input and output.

Engineering Connection
The binary number system is often used by electrical and computer engineers. For example, design of communication systems is a branch of electrical engineering. Communication involves efficient transfer of messages, or information, over distance from a transmitter to a receiver. The transmitter is concerned with reliable acquisition of information, proper information processing – often using binary codes, and use of appropriate means of sending information to the receiver; the receiver performs the inverse operations. Computer engineers design computer systems to efficiently operate on numbers stored in binary format.

Engineering Category
Category 2: relates math concept to engineering

Category 4: provides complete engineering design process

Keywords
Binary numbers, computer, microcontroller, basic stamp, communication system

Educational Standards
- NewYork State science: T1.4b, T1.5a, Information Systems 3.2
- New York State technology: engineering design, information systems, computer technology

Pre-Requisite Knowledge None

Learning Objectives
After this activity, students should be able to:
- Explain the difference between the decimal and binary number systems, and be able to perform needed conversions
- Be able to program simple tasks using the BS2 microcontroller

Materials List
Each group needs:
- One Parallax Board of Education #28803 with a properly installed BS2 microcontroller
- One laptop or desktop computer with the Parallax PBASIC Editor software and the USB driver (if using USB version of the Board of Education)
- One piezoelectric speaker (Parallax Basic Analog Kit #19678)

To share with the entire class:
- Jumper wires as needed

Introduction/Motivation

Although we don’t usually think of this, the binary number system is all around us, and is in almost every piece of technology we use. Examples include portable music players, kitchen appliances like microwave ovens, computers, and even ordinary indoor light switches. The reason the binary number system is so useful is because it is very simple to understand and use, and is also easy to use in technology like computers. Also, as we will learn in this activity, the binary number system is used in Morse code – a method of communication over long distances that has been in use for more than 100 years, and still in use today.

The binary number system can be used to represent numbers as sequences of zeros and ones. The basic element of a binary number system is a bit, which can take the values 0 or 1. The bit can be used to answer a so-called binary question. Who can give an example of a question that requires an either “yes” or “no”, or binary, answer? (Answers: Has the library book been returned? Will the meeting take place in the morning or afternoon? Is there any news?) One can use the binary digit 1 to encode a “yes” answer, and the digit 0 to encode a “no” answer. However, a single bit cannot be used to represent several answers. For example, consider the following question. There are four fruit on the table: an apple, banana, orange, and kiwi. Which one fruit will Pete prefer? Since there are four possibilities, there are also four answers. We can use two bits together to index the possibilities, and assign each two-bit combination to each distinct answer. One way to do this is to assign the binary number 00 to denote “apple”, 01 “banana”, 10 “orange”, and 11 “kiwi”. Similarly, a question that requires one of eight distinct answers can be answered by a three-bit binary number. Let’s list all possible three-bit binary numbers. (Answer: 000, 001, 010, 011, 100, 101, 110, 111.) Notice that each of these binary numbers corresponds to one of the integer numbers 0, 1, …, 7. In other words, a three-bit binary number can have 8 integer values.

Given a binary number, how can we convert it to an integer? One way is to look at the bits from right to left, and form the integer as a sum. For example, for a three-bit binary number, the integer is equal to

integer value = $\text{bit}(2)*2^2 + \text{bit}(1)*2^1 + \text{bit}(0)*2^0$,

Notice that we count the bits from right to left, and the rightmost bit is considered to be the zero-th bit. As an example, for the binary number 110, we have the integer

$1*2^2 + 1*2^1 + 0*2^0 = 4 + 2 + 0 = 6$.

Now, given an integer, how can we convert it to a binary number? A simple procedure is to divide the number by 2, and record the remainder, which will be either a 1 or a 0, since we are dividing by the even number 2. We keep the remainder, and divide the result by 2 again, recording the remainder. We continue this process until the result of division is a 0. For example, suppose we want to represent an integer 9 as a binary number. It follows that

9/2 = 4 with remainder 1,

4/2 = 2 with remainder 0,

2/2 = 1 with remainder 0,

1/2 = 0 with remainder 1.

Therefore, our binary representation of the number 9 is 1001. You can verify this answer by expressing 1001 as an integer using

$1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 8 + 0 + 0 + 1 = 9$.

In Morse code, letters of the alphabet and other characters like numbers can be represented as sequences of binary numbers, although each bit in the sequence is sometimes called a “dit” or a “dah”. In writing, a “dit” is represented by a dot (.), and a “dah” is represented by a dash (-). For example, the capital letter E can be represented as a single bit 0, a “dit”, or equivalently a dot (.). Other letters have different representations that are listed in Morse code tables. An interesting feature of Morse code is that the more often used letters have shorter representations, while the less often used letters, like Q, have longer Morse codes. In this activity, we will implement the capital alphabet letters A through Z, and the numbers 0 through 9. Notice that the integer numbers in Morse code are not represented as binary numbers, for example, the number 3 in Morse code is represented as a five-bit number 00011 (. . . - -), and **not** as the binary number 11.

Vocabulary / Definitions

Word	Definition
Binary number	A number in which each digit or element is either a zero or a one.
Decimal number	A number in which each digit is a number from 0 to 9.
Bit	The basic element in a binary number system.
Nib	A four-bit binary number, as used in the Basic Stamp programs.
Byte	An eight-bit binary number.
ASCII code	A code used in information processing systems like computers, where each letter, number, and symbol is represented using 7 bits (the smallest standard variable size to store 7 bits is a byte). ASCII stands for American Standard Code for Information Interchange.
Flow diagram	A high-level pictorial representation of a computer program.
Routine	Part of a computer program that is designed to accomplish some task(s). It is often called a function.
Microcontroller	A small computer that can interface with the outside world and execute user-specified commands.
Basic Stamp 2	An example of a microcontroller.
Memory	A location inside a computer or microcontroller where binary numbers can be stored and retrieved when they are needed by a program.
Register	Another name for a location in memory.
Lookup table	A collection of data or numbers stored in computer memory, each entry of which can be retrieved given its memory location.
EEPROM	Type of computer memory that retains its contents even when the computer like a Basic Stamp is powered down. EEPROM stands for Electronically Erasable Programmable Read Only Memory.
RAM	Type of computer memory that retains its contents only when the computer like a Basic Stamps is powered up. RAM stands for Random Access Memory, and is used to store numbers used in a program.

LED	Light emitting diode, an electrical device that emits light when properly connected to an electrical energy source.
Resistor	An electrical component that acts to limit the amount of electrical energy flowing through it.
Transducer	An element that converts one form of energy to another. An example is the conversion of electrical energy to sound energy.
Piezoelectric material	A material that changes physical shape when connected to an electrical energy source.
Piezoelectric speaker	A speaker that uses a piezoelectric material as a transducer.
Schematic diagram	A diagram used to depict an electric circuit, which can be replicated by an engineer in the lab.

Procedure

Background

Before the Activity

- Make copies of the procedure section and attached handouts.

With the Students

1. Divide the class into groups of three students. Team members will work together on all aspects of the activity.
2. Introduce and demonstrate how to use the Basic Stamp software to write, save, and run programs. Demonstrate the Basic Stamp board, in particular the on/off switch (for this activity, the on/off switch should be set to position 2), and the white prototyping "breadboard" area used to build circuits. Discuss safety and precautions.
3. Demonstrate action of the piezoelectric speaker by building the circuit in the right panel of Image 2, loading and running the program in Image 3. The speaker should emit a 3 kHz tone of 1 second duration. Illustrate the use of the **FREQOUT** command. Change the frequency output from 3 kHz to 1 kHz. Let students comment on how the pitch and frequency of the sound changes. Comment on how the first two lines of commented code are necessary in every Basic Stamp program.

Insert image 2 here, centered

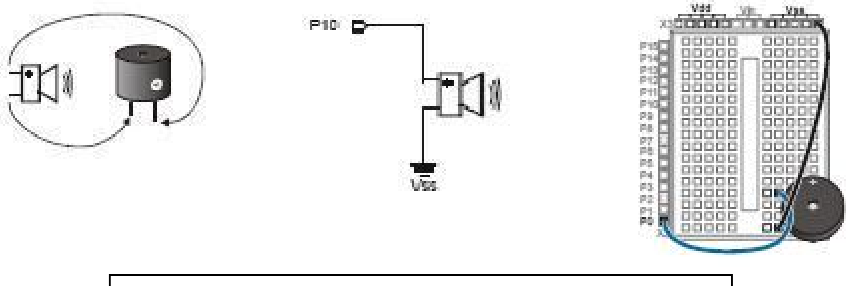
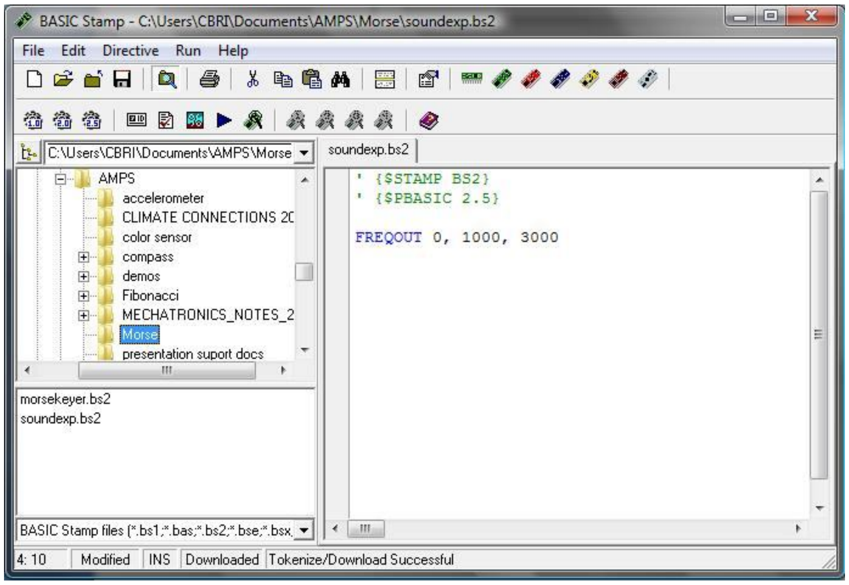


Image 2
ADA Description: An image of a piezoelectric speaker, together with the Basic Stamp schematic.
Caption: A piezoelectric speaker together with its schematic.
Image file name: morse_keyer2.jpg
Source/Rights: © Parallax Inc.

Insert image 3 here, centered


BASIC Stamp - C:\Users\CBRI\Documents\AMPS\Morse\soundexp.bs2
File Edit Directive Run Help
C:\Users\CBRI\Documents\AMPS\Morse
AMPS
accelerometer
CLIMATE CONNECTIONS 20
color sensor
compass
demos
Fibonacci
MECHATRONICS_NOTES_2
Morse
presentation suport docs
morsekeyer.bs2
soundexp.bs2
BASIC Stamp files (*.bs1;*.bas;*.bs2;*.bse;*.bsx
soundexp.bs2
4: 10 Modified INS Downloaded Tokenize/Download Successful

```
' {$STAMP BS2}
' {$PBASIC 2.5}

FREQOUT 0, 1000, 3000
```

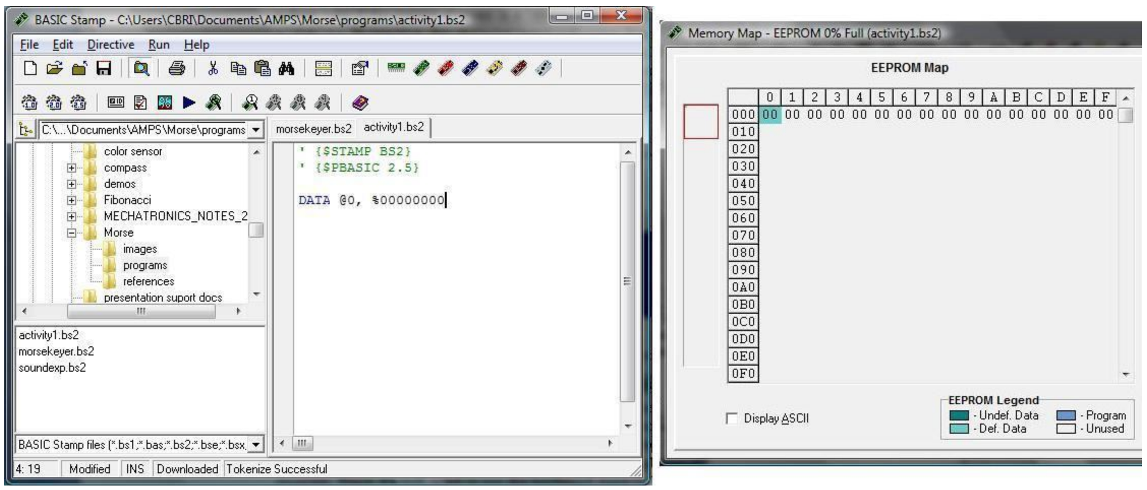
Image 3
ADA Description: A program demonstrating how to use a piezoelectric speaker.
Caption: A program to play a 3 kHz signal.
Image file name: morse_keyer3.jpg
Source/Rights: none

4. Pass out the Basic Stamp boards and laptops. Pass out one piezoelectric speaker and two jumper wires to each group.
5. Have students build a functioning Morse code generator by following the engineering design process below.
6. **Hardware setup:** Students build the circuit in Image 2. The speaker can be connected to any of the input/output pins 0-15, although Image 2 shows examples using pin 10 and pin 0. The Basic Stamp board should then be connected to a power source (9 Volt battery or an AC adapter) and a computer through a serial (USB) cable.
7. **Problem statement:** The Morse code generator is to output a codeword (refer to Morse character set handout) for each letter of the alphabet (A, B, C, …, Z), and each of the numbers (0, 1, 2, …, 9). In this activity, we are implementing only capitalized letters. Each character has a Morse code, with some characters having shorter codewords than others. Given a codeword, the Basic Stamp needs to look at each codeword element from left to right, and output a short tone if the current element is a “dit” (.), and a longer tone if the element is a “dah” (-). In practice, the character input will come from the computer keyboard. This means that the Basic Stamp should take a keyboard input, interpret it as a codeword

using a lookup table located in its memory, and then output a sequence of short and long tones. After this, the Basic Stamp should wait for the next keyboard input, and repeat the procedure when it receives a new character. **For discussion:** In teams, have students come up with flow diagrams illustrating how the program will perform the outlined steps. Compare and discuss the solutions with class. Refer to the flow diagram handout.

8. **EEPROM memory organization:** Recall that a lookup table is a collection of memory locations with specified addresses. EEPROM memory on the Basic Stamp can store 2048 bytes of information, which can include the program as well as other data. Each byte has a corresponding address, from 0 to 255 (notice that 256*8 = 2048). When a program is downloaded into the Basic Stamp, it is stored in the highest addresses in the EEPROM, starting with address 255 and working downward. This means that the lower addresses starting with 0 are available to store data.
    - **Experiment:** To illustrate how EEPROM memory is organized, open a new file using a PBASIC editor, and enter the program shown in the left window in Image 4. This program uses the command **DATA** to enter a byte into memory with address 0, having the value 00000000. Notice that the symbol @ is required before the address 0, and the symbol % is required before the binary number 00000000. Look in the help menu to learn about the command **DATA**. Load and run the program by going to the **Run** menu and then clicking **run**. After the program loads, click on the **Run** menu, and click on **Memory Map** to bring up a screen in the right panel in Image 4. Observe that the 0th memory location now contains a binary number, highlighted in green.

Insert image 4 here, centered



```
' {$STAMP BS2}
' {$PBASIC 2.5}

DATA @0, %00000000
```

```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
010
020
030
040
050
060
070
080
090
0A0
0B0
0C0
0D0
0E0
0F0
```

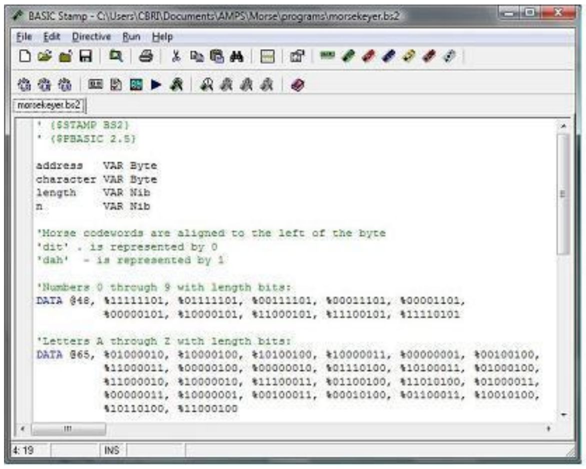
BASIC Stamp - C:\Users\CBRI\Documents\AMPS\Morse\programs\activity1.bs2
File Edit Directive Run Help
color sensor
compass
demos
Fibonacci
MECHATRONICS_NOTES_2
Morse
images
programs
references
presentation suport docs
activity1.bs2
morsekeyer.bs2
soundexp.bs2
BASIC Stamp files (*.bs1;*.bas;*.bs2;*.bse;*.bsx
4: 19 Modified INS Downloaded Tokenize Successful
Memory Map - EEPROM 0% Full (activity1.bs2)
EEPROM Map
Display ASCII
EEPROM Legend
Undef. Data
Def. Data
Program
Unused

Image 4
ADA Description: A program demonstrating the use of the command DATA.
Caption: A program to store one binary number in EEPROM memory.
Image file name: morse_keyer4.jpg
Source/Rights: none

- **EEPROM memory use:** We will use ASCII code (see ASCII character set handout) to store and address each character. For example, we will store the letter A in the EEPROM memory location 65, the letter B in the memory location 66, and so on up to the letter Z, which will be stored at location 90. The numbers will be stored similarly. For example, we with store 0 at location 48. Also, we will store the length of each Morse codeword together with its binary representation. For example, for the letter E, the memory location 69 will contain the codeword 0 aligned to the left of the byte, and its codeword length 1 aligned to the right of the byte – 00000001. What binary number will be stored in the EEPROM memory location 76? (Answer: the ASCII character for 76 is the letter L. Looking up its Morse codeword we obtain 0100, with length 4. Because the binary representation of the number 4 is 100, we will store the number 01000100 at location 76 in the EEPROM.)

- **RAM memory use:** We will need to define four variables to use in the program.
    1. **address** – a byte-sized variable that will store the codeword address.
    2. **character** – a byte-sized variable to store the Morse codeword. This is due to the fact that the largest codeword length is 5 (for each of the numbers 0, 1, …, 9). Therefore the smallest variable required is a byte.
    3. **length** – a nib-sized (4-bit) variable to hold the codeword length. Since the codeword length is variable (for example, from one bit for letter E to five bits for the number 7), we will need a variable that can take on decimal values up to 5, and the smallest such variable supported by the Basic Stamp is a 4-bit variable called a nib.
    4. **n** – since we will be outputting tones for each bit in a given codeword, we need a variable to store the position of the current bit, and increment it for each bit in the codeword. The largest codeword length is 5, and therefore this variable will likewise be a nib.
- **Variable declaration and EEPROM initialization:** We must first declare variables needed in a program before using them. Create a new file named **morsekeyer.bs2**, and declare four variables as shown in Image 5. Then initialize the EEPROM with the binary numbers corresponding to Morse codewords for 0, 1, …, 9 and their lengths, starting at the ASCII address 48. Finally, initialize the EEPROM starting at address 65 with data for the letters A through Z. Load and run the program, and observe the memory map. In the **EEPROM Map** window in Image 5, notice how the 10 numbers are stored starting at the hexadecimal location 030 (we have not introduced the hexadecimal number system, but the hexadecimal number 030 is equivalent to the decimal number 48). Also notice how the letters are stored starting at the hexadecimal address 041 (decimal 65). Now observe the **RAM Map** and notice that the two-byte register named REG0 has been reserved to contain contents of the byte-sized variables **address** and **character**. Also, the register named REG1 has been reserved to contain the nib-sized variables **length** and **n**.

Insert image 5 here, centered


BASIC Stamp - C:\Users\CBRI\Documents\AMPS\Morse\programs\morsekeyer.bs2
File Edit Directive Run Help
morsekeyer.bs2
4: 19 INS

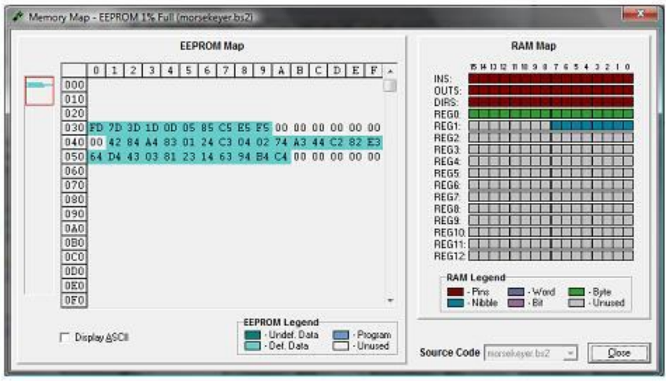
```
' {$STAMP BS2}
' {$PBASIC 2.5}

address   VAR Byte
character VAR Byte
length    VAR Nib
n         VAR Nib

'Morse codewords are aligned to the left of the byte
'dit' . is represented by 0
'dah'  - is represented by 1

'Numbers 0 through 9 with length bits:
DATA @48, %11111101, %01111101, %00111101, %00011101, %00001101,
          %00000101, %10000101, %11000101, %11100101, %11110101

'Letters A through Z with length bits:
DATA @65, %01000010, %10000100, %10100100, %10000011, %00000001, %00100100,
          %11000011, %00000100, %00000010, %01110100, %10100011, %01000100,
          %11000010, %10000010, %11100011, %01100100, %11010100, %01000011,
          %00000011, %10000001, %00100011, %00010100, %01100011, %10010100,
          %10110100, %11000100
```



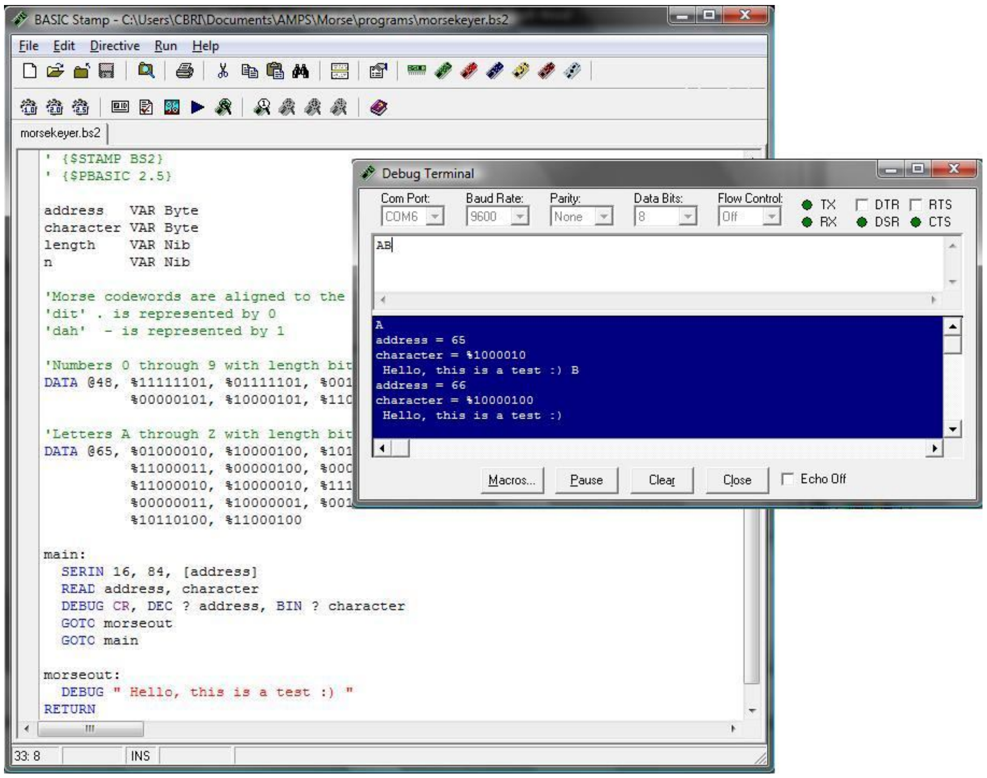
```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000
010
020
030 FD 7D 3D 1D 0D 05 85 C5 E5 F5 00 00 00 00 00 00
040 00 42 84 A4 83 01 24 C3 04 02 74 A3 44 C2 82 E3
050 64 D4 43 03 81 23 14 63 94 B4 C4 00 00 00 00 00
060
070
080
090
0A0
0B0
0C0
0D0
0E0
0F0
```

Image 5
ADA Description: A program morsekeyer.bs2 that declares variables and initializes the EEPROM.
Caption: First update of the Morse keyer program.
Image file name: morse_keyer5.jpg
Source/Rights: Copyright 2009 Pavel Khazron. Used with permission.

- **The main program:** At this point, we are ready to write code for the main part of the program. Programmers often name a piece of code that carries out the most important functions in a program as **main**. In our case, commands in the **main** block are required to perform the following tasks.
    1. The Basic Stamp is to wait for and properly receive a keyboard input, which will be an address encoded in ASCII format.
    2. Given the address, a byte corresponding to this address is to be retrieved from EEPROM memory.
    3. The retrieved byte needs to be partitioned into a Morse codeword and a codeword length. A short or a long tone needs to be played on the piezoelectric speaker for each bit in the codeword.
    4. The Basic Stamp is to go back and wait for the next keyboard stroke.
    5. The first step above can make use of the PBASIC command **SERIN**. This command is used for serial communication between the Basic Stamp and the outside world. Step (ii) is easily carried out using the command **READ**. Step (iii) is the most involved part of the program, and is discussed later. Finally, the **GOTO** command is used to take the Basic Stamp back to Step (i) indefinitely. Use the help menu to read about the commands **SERIN**, **READ**, and **GOTO**. Understanding all of the help information on the **SERIN** command is not necessary. Add the main code into your program **morsekeyer.bs2**, as shown in Image 6. Commands after the main label carry out Steps (1), (2), and (4). Step (3) will be coded at the location of the statement **DEBUG " Hello, this is a test :) "**. Load and run the program. When run, the program opens a so-called debug terminal. Enter some capitalized letter or a number into the white area inside the debug terminal. The ASCII address in binary, and the contents of the corresponding memory location are printed in the blue area of the

terminal. The command **DEBUG** is used to display this data in order to test how the program operates.

Insert image 6 here, centered


BASIC Stamp - C:\Users\CBRI\Documents\AMPS\Morse\programs\morsekeyer.bs2
File Edit Directive Run Help
morsekeyer.bs2
Debug Terminal
Com Port: COM6
Baud Rate: 9600
Parity: None
Data Bits: 8
Flow Control: Off
TX RX DTR DSR RTS CTS
Macros... Pause Clear Close Echo Off
33: 8 INS

```
' {$STAMP BS2}
' {$PBASIC 2.5}

address   VAR Byte
character VAR Byte
length    VAR Nib
n         VAR Nib

'Morse codewords are aligned to the
'dit' . is represented by 0
'dah'  - is represented by 1

'Numbers 0 through 9 with length bit
DATA @48, %11111101, %01111101, %001
          %00000101, %10000101, %110

'Letters A through Z with length bit
DATA @65, %01000010, %10000100, %101
          %11000011, %00000100, %000
          %11000010, %10000010, %111
          %00000011, %10000001, %001
          %10110100, %11000100

main:
  SERIN 16, 84, [address]
  READ address, character
  DEBUG CR, DEC ? address, BIN ? character
  GOTO morseout
  GOTO main

morseout:
  DEBUG " Hello, this is a test :) "
RETURN
```

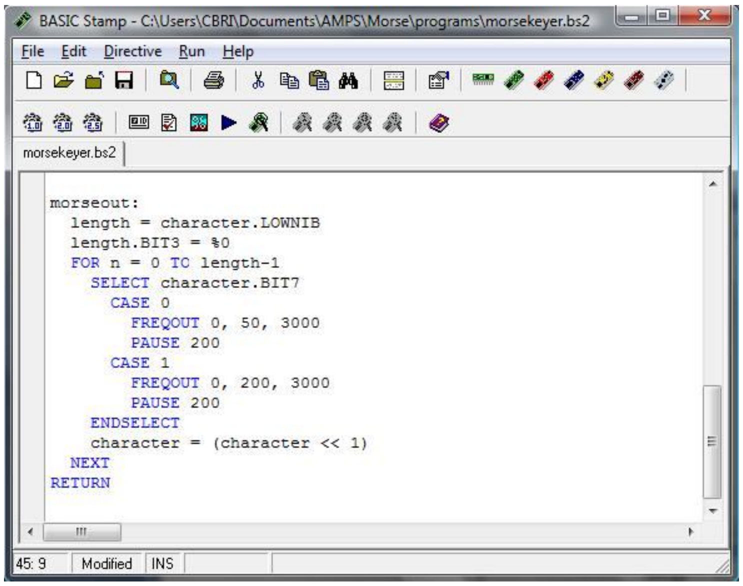
```
AB
```

```
A
address = 65
character = %1000010
 Hello, this is a test :) B
address = 66
character = %10000100
 Hello, this is a test :)
```

Image 6
ADA Description: The program in file morsekeyer.bs2 updated with the main routine.
Caption: Second update of the Morse keyer program.
Image file name: morse_keyer6.jpg
Source/Rights: Copyright 2009 Pavel Khazron. Used with permission.

- **The routine morseout:** Given a Morse codeword and its length contained inside a byte-sized variable character, we need to program a routine to output Morse code to the piezoelectric speaker. Refer to Image 7. Since the length of the Morse codeword is aligned to the right of the byte in the variable character, we initialized the length variable with the lower (rightmost) four bits of the character variable, using the modifier **LOWNIB**. Since (the leftmost) bit 4 of the length variable is not valid (Why?), we set it to binary zero on the next line. Next, we use a **FOR NEXT** statement to loop length number of times. For example, for a length 5 codeword, we will loop once for each of the 5 bits in the Morse codeword. We test each bit using the **SELECT** statement according to its value (0 or 1). If the bit is a 0, we send a short 3 kHz tone to the speaker connected to pin 0, followed by a 200 millisecond pause, while if the bit is a 1, we output a long 3 kHz tone. Finally, we use the right-shift operator << to shift the character variable one bit to the right. In the case when character contains 10110100, right-shifting by 1 would give the result 01101000. Update your program as in Figure 7. Load and run the program on the Basic Stamp. When the debug terminal window appears, enter either a capital letter or a number. Listen to the Morse code and compare with the selected Morse code character set handout. Do the sequences of short and long tones correctly communicate each of the letters and numbers? If you enter a lowercase letter, or press any key other than a number or an uppercase letter, the Basic Stamp program will crash (Why?). In this case, press the reset push-button switch located on the Basic Stamp board to reset the program.

Insert image 7 here, centered


BASIC Stamp - C:\Users\CBRI\Documents\AMPS\Morse\programs\morsekeyer.bs2
File Edit Directive Run Help
morsekeyer.bs2
45: 9 Modified INS

```
morseout:
  length = character.LOWNIB
  length.BIT3 = %0
  FOR n = 0 TO length-1
    SELECT character.BIT7
      CASE 0
        FREQOUT 0, 50, 3000
        PAUSE 200
      CASE 1
        FREQOUT 0, 200, 3000
        PAUSE 200
    ENDSELECT
    character = (character << 1)
  NEXT
RETURN
```

Image 7
ADA Description: The program in file morsekeyer.bs2 updated with the morseout routine.
Caption: Third update of the Morse keyer program.
Image file name: morse_keyer7.jpg
Source/Rights: Copyright 2009 Pavel Khazron. Used with permission.

Attachments

ASCII_Character_Set.doc

Morse_Character_Set.doc

Morse_Code_Flow_Diagram.doc

Morse_Code_Solutions.doc

soundexp.bs2

activity1.bs2

morsekeyer.bs2

Safety Issues
Don’t handle the Basic Stamp microcontroller and power circuitry on the board, and don’t perform the activity around food or liquids.

Troubleshooting Tips
If the Basic Stamp board or the PBasic editor crashes, close and restart the editor, and cycle the power to the board.

Investigating Questions

Assessment

Pre-Activity Assessment
Sharing Experiences: Ask students to share their programming experience. Write down collective experiences on the board. Conduct a discussion on the binary number system. Write down any technical terms raised by participants.

Activity Embedded Assessment
Analysis: Encourage students to draw connections between programs and the attached flow diagrams. Encourage creative thinking or different programming approaches from those suggested in the activity. Note new or original solutions.

Post-Activity Assessment
What Have We Learned?: Conduct a discussion with the same theme as in the pre-activity assessment, and note new lessons, skills, or techniques students have developed.

Activity Extensions
The Morse code standard also contains codewords for symbols like a period, comma, and exclamation mark. Is it possible to modify the program developed in this exercise to use those symbols? If yes, discuss a workable approach, and if not, provide a supporting explanation. In addition, compute the character rate (in letters or numbers per second) of the Morse code generator in this activity. How can we modify this rate?

Activity Scaling None

Additional Multimedia Support None

References
Basic analog and digital Manual, ver 1.4, Accessed September 16, 2009. http://www.parallax.com/Portals/0/Downloads/docs/books/edu/Web-BasicAnalogDigital-v1.4.pdf

StampWorks Manual v2.1. Accessed September 16,2009.
http://www.parallax.com/Portals/0/Downloads/docs/books/sw/Web-SW-v2.1.pdf.

Redirect URL	http://gk12.poly.edu/amps-cbri/
Owner	Pavel Khazron
Contributors	Pavel Khazron
Copyright	Copyright © 2009 by Polytechnic Institute of NYU. The development of this activity was supported by Project AMPS under a GK-12 Fellows grant 0807286 from the National Science Foundation.

Version: August, 2009