Lab 02: Blinking Your BOE-Bot

When You've Never Programmed Before

INTRODUCTION

What do your computer, tablet, and smartphone have in common? Microprocessors! You're pretty used to smart devices, which can handle multiple tasks pretty much all at the same time. But for your computer to be, well, a computer, it needs more than a microprocessor. You need memory (RAM), storage (hard drive), a graphics card—just to name a few things! The microprocessor by itself is like a brain without a body: smart, but not terribly capable.

What do your microwave, calculator, RC cars, and most household appliances have in common? Microcontrollers! The difference is that a microcontroller is specifically designed to one thing (or just a few related things), without needing a complex operating system.

When you think about robots, maybe you think about the life-like humanoid robots you see in movies—which are fantastic, but mostly fiction. Everyday robots usually don't resemble humans: form-follows-function means that they will look like what they are designed to do. A robotic probe designed to explore the surface of Mars really shouldn't look anything like a person, and neither should an industrial robot designed to paint auto body parts!

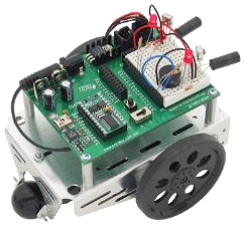
And you would never load your Mars probe with a program to paint auto body parts. The microcontroller in your calculator needs to perform tasks (like graphing or trig functions) that the microcontroller in your washing machine will never need, and even though you probably would never call your calculator or laundry machines "robots," they share exactly the same kind of one-track-mind microcontroller programming.

OBJECTIVES

The specific goals for this exercise are:

- Verifying that your BOE-Bots are properly constructed
- Writing a simple program to test the microcontroller
- Wiring a simple circuit to light an LED
- Programming loops to make the BOE-bot to blink

ASSEMBLY



The robots we'll be using are called BOE-Bots, and are designed to give us a first look at how to design a device to solve a specific problem, and how to implement that solution with simple engineering and programming.

Look carefully at the assembled bot. The microcontroller chip is actually pretty small, but that's where the processor is, along with the memory.

To make the bot "do" anything, you need a power source. There's a battery pack mounted on the underside.

To make your bot "go" anywhere, you need wheels. Notice that the wheels are attached to servo motors, which are plugged in to sockets on the circuit board. The microcontroller can tell the wheels to spin clockwise or counterclockwise.

Notice also that you have extra ports, where you can attach other devices. And a "breadboard" for electronic circuits means that we can also add sensors.

Now do a fastener check! Replace any missing fasteners, and make sure that everything is adequately tightened.

HELLO WORLD!

Once you have a completely assembled bot, you need to make sure the brain works! Open the MacBS2 Basic Stamp editor, then plug in your bot using the USB cable.

This is pretty universally the first program you write in whatever language you are learning...how to display some output. Type the following (but don't type the line numbers on the left!):

```
01)   'Program: Hello World!
02)   '{$STAMP BS2}
03)   '{$PBASIC 2.5}
04)   DEBUG "Hello world!", CR
05)   END
```

This will compile cleanly and run, but what do the lines mean?

Line 01: Using a single quote to begin a line indicates a comment. Comments are statements that the controller reads, but doesn't try to execute. This comment provides a title for the program.

Lines 02 and 03: These control directives are necessary to every program you write, and are always automatically inserted whenever you open a new editor window. They are identifying which BasicStamp microcontroller (BS2) is being used, and which version of the PBasic language is being used to communicate with it (2.5). Never edit or delete these lines!

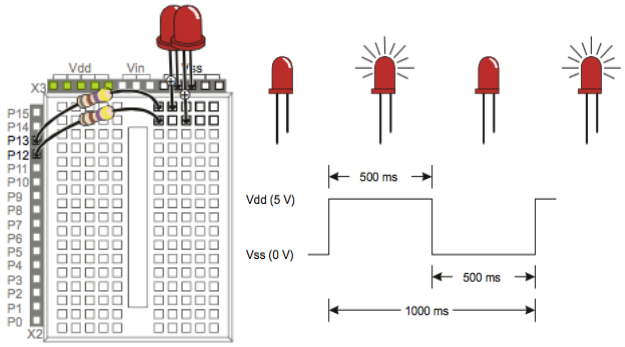
Line 04: DEBUG displays a text string (enclosed in double quotes) to the output terminal. CR goes to the next line.

Line 05: Get into the habit of always closing your programs with an END statement, even if you think it's obvious that the program is at an end.

LIGHTS!

Displaying this "Hello world!" message is not all that useful, but having the ability to display something is important!

If your bot isn't connected to the computer, though, it can't display a screen message to get your attention. But a flashing light? That would probably get your attention.



We can use the onboard power and the breadboard to attach some LED lights, and write a program to control when they turn on and off. Once we know how to turn them on/off, we can decide why we might want to flash the lights, and then incorporate that into solving a bigger problem.

- Use the wiring diagram above to wire up two LEDs to one of your bots. Note that you do not have to use the same LED positions, but you *do* need to have a complete circuit to light the bulbs. You can use a plain wire as a jumper to place one of the LEDs at the far end of the breadboard.
- Select your resistors carefully! You must use the proper resistor (notice the color band coding!).
- Notice how each bulb is wired to a separate P receptacle; while you might not necessarily need to use P12 and P13 as shown, as we wire up more attachments, we want to be sure we aren't "double dipping."

You do not need to save your "Hello World!" program to submit. This next set of programs, which will control the LEDs, is the source code you should save and submit for credit. Get in the habit of saving and re-using your last program to complete your next one! Don't try to strip the code below out of the .pdf document, there will be invisible formatting that makes the IDE editor choke! (Also, please never manually enter the individual line numbers! The IDE automatically numbers the lines for you.)

```
01)     'Program: Don't Blink!
02)     '{$STAMP BS2}         'Stamp directive
03)     '{$PBASIC 2.5}        'Language directive
04)     DEBUG "Don't blink!", CR
05)     HIGH 3.               'Power on to Pin 3
06)     DEBUG "Pin 3 light on!"
07)     PAUSE 500             'Wait half a sec
08)     LOW 3                 'Power off to Pin 3
09)     DEBUG "Pin 3 light off!"
10)     PAUSE 500             'Wait half a sec
11)     HIGH 11               'Power on to Pin 11
12)     DEBUG "Pin 11 light on!"
13)     PAUSE 500             'Wait half a sec
14)     LOW 11                'Power off to Pin 11
15)     DEBUG "Pin 11 light off!"
16)     END                   'End of program
```

Notice that the comments don't get displayed when you run the program, only the DEBUG strings display to the screen!

LOOPS

Flashing the light one time? Maybe useful. But we can also control the number of times we want a flash. Or we can program a blink to just keep blinking until a user responds.

Let's say we want a signal: a particular number of flashes of a particular LED. We can write some code that will trigger only one LED, and flash it exactly the number of times we decide. Let's look at an iteration, or loop structure:

```
01)     'Program: Blink! Blink! Blink!
02)     '{$STAMP BS2}         'Stamp directive
03)     '{$PBASIC 2.5}        'Language directive
04)     index  VAR Byte       'Memory for counter
05)     blinks VAR Byte       'Memory for total
06)     blinks = 3            'Number of loops
07)     FOR index = 1 TO blinks
08)       'What needs repeating?
09)       'You figure out what lines go here!
10)     NEXT                  'Advances index by 1
11)     END                   'End of program
```

That seems like a lot of code…and a lot of comments!

Start to notice that we can keep recycling code! We can use the same pieces, put together in different ways in different programs. Having a lot of comments makes sure that you always know what any particular piece of code supposed to do, even if you wrote it ages ago!

This is also why you always want to save the programs you write, especially once you know they work properly. It's a good practice to reuse/recycle rather than reinvent/rewrite the same bits and pieces of code that get used often. In programming, copy/paste should be the keystrokes that your fingers fly to automatically!

TWO LOOPS!

Now add a second loop to your program. Edit the source code so that the first loop blinks one LED (you choose how many times), followed by a second loop which blinks the other LED. Again, copy/paste should be your best friend!

SAVE AND SUBMIT

Be sure to save your source code frequently! Always save to your own UCA Google drive. If you are using the UCA computers, save the program on the Desktop—but always save it in a second location. If it's on your Google drive, you won't need to be in the lab to access the files!

- Use the proper file name: Whatever you have named your program in the source code, you must use the correct filename for submission. Name your two-loop program lastnameLAB02, obviously using your own last name. It should already/automatically have the .bs2 file extension. Never submit word processor documents or .pdf files!
- Submit electronically: Your program is due no later than 6:00 PM on Tuesday, 06 February 2024. Everyone must submit via the Blackboard Assignment.
- Complete your kit: If you don't have two complete robots, or you are missing any accessories, please round out your kit. Make you're your bots aren't missing any fasteners! You can obtain the parts you need from the accessory boxes on the lab shelves. *Spoiler alert:* You will eventually need both robots operating simultaneously, so you need to test your program on both!

GRADING RUBRIC

Your submitted source code will be graded using the assessment rubric below.

ASSESSMENT	CRITERIA / VALUE		POINTS EARNED
Lab 02 Program: Blink! DUE: Tue 06 Feb 24 (filename: lastnameLAB02.bs2)	Compilation: Program compiles cleanly	8 points	
	Execution: Program executes correctly: Two correctly programmed loops Each blinker is controlled separately	12 points	
	Annotation: Program is sufficiently commented	10 points	