

# Introduction to Looping

N. Narasimhamurthi

## 1 Objective

To become familiar with elementary loops and simple input/outputs functions.

### 1.1 Simple Input/Output

One of the basic functionality provided by any operating system is input/output routines. BUFFALO provides several useful functions for performing input/output. In this lab, we will look at two output functions provided by BUFFALO. Unlike in high level languages, functions in machine language are known by their addresses. The two functions we will be using are in ROM at locations `$FFB8` and `$FFBB`<sup>1</sup>. Rather than use these hard to remember and hard to recognize numbers, it is customary to give them meaningful names. Unless you have a good reason to do otherwise, it is best to use the name suggested by the vendor, in this case Motorola. The 'official' names for these functions are `OUTA` and `OUT1BYT` respectively. In assembly language, we make the connection between a name (technically known as a label) and a value using the `EQU` command as shown

```
OUTA          EQU      $FFB8
OUT1BYT       EQU      $FFBB
```

NOTE: Labels should be written starting from column 1. If a line does not have a label, it should start with a space or a tab or a comment character.

#### 1.1.1 The function `OUTA`

The function `OUTA` will transmit whatever is in register `A` over the serial communication line that is connected to the PC. What the PC does with this value depends on the terminal program that is used to communicate with the HC11. Under normal circumstances, the terminal program will interpret the value as an ASCII code and display the corresponding character on the screen.

---

<sup>1</sup>We will indicate HEX values with the prefix `$`. However, data that you would be entering in BUFFALO, as part of memory modify or register modify will be shown without the prefix `$` although it will be understood that the numbers are written in HEX.

**Exercise:** Power up HC11 and at the BUFFALO prompt try the following and write down what you see.

1. Issue the command RM (for register modify) and press the space bar till you see the **A** register. Enter the value 31 and press enter as shown below:

```
>rm
P-AAAA Y-AAAA X-AAAA A-AA B-AA C-DO S-004A
P-AAAA
Y-AAAA
X-AAAA
A-AA 31

>
```

Now execute the command

```
CALL FFB8
```

2. Repeat with the **A** register modified with the following values: 32, 33, 34, 21, 22, 23, 24, 25, 41, 42 43 44

If you are using the simulator, turn on the log feature (by pressing both the shift keys). If you are working with a real HC11, you can copy and paste the contents of the terminal screen.

### 1.1.2 The function OUT1BYT

This is a more involved output function. To start with, the value to be printed must be in memory. If the value is in a register you will have to store it in memory first. Next, the value in the **X** register should be the address where the value is stored. Thus this function should be told 'where' and not 'what'. When you call this function, the function will send *two* characters to the PC. If the terminal program interprets these two characters as ASCII codes and displays the corresponding two characters, then the display would be the value written in HEX. *In addition, the function will increment the value in the **X** register.* This is useful when displaying a series of memory locations.

**Exercise:**

1. Using the MM command (memory modify) enter the following values in memory locations 3000, 3001, ...: 30 31 32 33 41 42 43 44. Verify the values using the memory dump, MD, command.

2. Using RM, the register modify command, change the value in the **X** register to 3000
3. Execute the command CALL FFBB and write down what the output was and also the value in the **X** register after the command is executed.
4. Repeat the the command CALL FFBB and write down the output and the new value in the **X** register.
5. Repeat the last part until you have performed 7 calls to \$FFBB.

## 1.2 Branching

Conditional branching in HC11 is controlled by the state of one or more hardware flags. The state of a flag depends on the most recently executed instruction that affects the flag. Thus, if your branching depends on the result of some instruction, then it is your responsibility to make sure that none of the instructions between the instruction you are interested in and the branching instruction affects the flags that control the branching instruction. Thus, it is a good idea to follow the instruction that sets the flag by the branching instruction. In this lab, we will use the following conditional branch instructions:

BEQ	Branch if the Z flag is set
BNE	Branch if the Z flag is not set

Now, the Z flag is set after most instructions if the result of the instruction is a *zero*; or else it is cleared, i.e. not set. The two most important instructions that are often used to set/clear the flag are

CMP	Perform a subtraction and discard the answer. However, set the flags.
TST	Same as CMP except subtract the number zero

Thus, after the CMP command, the Z flag would be set if the two values that are compared are equal. Similarly, the TST command will compare a value (memory or register) with zero and set the Z flag if the value is zero.

### Exercise:

1. Using the HC11 reference book, identify 5 instructions that do *not* affect the **C** flag, but affects some other flag.
2. Using the HC11 reference book, can you identify any instruction that does *not* affect the **V** flag, but affects some other flag?

3. Using the HC11 reference book, identify 5 instructions that always clears the **C** flag.
4. Using the HC11 reference book, identify an instruction that always sets the **C** flag.

## 1.3 Looping

### 1.3.1 Counting loops

This is by far the simplest and most used loop structure. In a counting loop, we perform a specific operation a given number of times. A counter controls how many times the loop is executed. The counter could be stored in memory or kept in a register. If it is kept in a register, it is your responsibility to make sure that the register is not inadvertently changed either by your code or by some third party function that you call. If you decide to use a register, your best bet is to use either the **B** register or **Y** register. The structure of the loop is as follows:

- 1: Initialize the counter to the number of times the loop is to be performed
- 2: Perform any other initializations
- 3: Test if the counter is zero. If so quit the loop
- 4: Perform the desired task
- 5: Perform any re-initializations
- 6: Decrement the counter
- 7: Go back to 3
- 8: Come here when you quit the loop

Note that there are two places where the code jumps to. One to (3) and the other to (8). When writing the code in assembly language, we would need two labels. In the code that follows, I have used the labels **FOO** and **BAR**.

#### Exercises:

1. Assemble the following code in your PC, transfer the S19 file to HC11, run the program and write down the output of the program.

```

;Name:
;email:
;date:
;
OUTA      EQU      $FFB8

                ORG $2100
                LDAB  #$9 ; USING REGISTER B AS A COUNTER
                LDAA  #$31 ; OTHER INITIALIZATION
FOO        TSTB      ; SUBTRACT ZERO FROM B
                BEQ  BAR ; QUIT IF THE Z FLAG IS SET, I.E. B=0
                JSR  OUTA ; DO THE TASK
                INCA ; RE-INITIALIZE
                DECB ; DECREMENT THE COUNTER
                BRA  FOO ; GO BACK

BAR

                SWI

```

2. Modify the above program so that the program prints the upper case letters A to Z. Clearly indicate the changes you made.
3. The following function is equivalent to the function shown above<sup>2</sup>.

```

; continued from the previous function

                ORG $2200
                LDAB  #$9 ; USING REGISTER B AS A COUNTER
                LDAA  #$31 ; OTHER INITIALIZATION
JUBJUB      JSR  OUTA ; DO THE TASK
                INCA ; RE-INITIALIZE
                DECB ; DECREMENT THE COUNTER
                BNE  JUBJUB ; GO BACK

                SWI

```

---

<sup>2</sup>You can have as many functions as you want in the same file. Make sure that when you change the ORG, the functions do not overlap. You can determine this by looking at the LST file

Verify that CALL 2100 and CALL 2200 produces the same output. Explain why this is so, and how and why the loop terminates.

### 1.3.2 One, two! One, two! And through and through ... Marching through memory

Often loops are combined with marching through memory and operating on consecutive memory location. In this case, the **X** (and/or **Y**) register is initialized to a starting memory address. Inside the loop, memory is accessed using `IND,X` addressing mode. This operates on memory whose address is computed using the value in **X** register. At the bottom of the loop, **X** is incremented, so that next time around the loop, the operation is performed on the next memory location<sup>3</sup>. The following program shows prints using `OUTA` the values stored in 9 consecutive locations starting from location `$3000`. Note we are using `FCB` which is the assembly language equivalent of `memory modify`.

```

                                ORG    $2300

                                LDAB   #$09
                                LDX    #$3000 *Don't forget the #

VORPAL                          TSTB
                                BEQ    SWORD

                                LDAA   0,X
                                JSR    OUTA

                                INX
                                DECB
                                BRA    VORPAL

SWORD                          SWI

; FOLLOWING SAME AS MM 3000 FOLLOWED BY: 55 6F 66 4D 2D 44 62 72 6E
                                ORG    $3000
                                FCB    $55, $6F, $66, $4D, $2D, $44, $62, $72, $6E

```

Run the above program using `CALL 2300`<sup>4</sup>. Modify the above program as shown below and explain what the program does. Do you see why we do the `DEX` before

<sup>3</sup>In some cases the memory has to be accessed in the reverse order. In this case, **X** starts at the end, and at the bottom of the loop, **X** is decremented.

<sup>4</sup>Don't forget to assemble it and then transfer the S19 file to the HC11!

we access memory?

```

                                ORG    $2400
                                LDAB   #$09
                                LDX    $3000
                                ABX
SNICKER                        TSTB
                                BEQ    SNACK
                                DEX
                                LDAA   0,X
                                JSR    OUTA
                                DECB
                                BRA    SNICKER
SNACK                          SWI
                                ORG    $3000
                                FCB    $55, $6F, $66, $4D, $2D, $44, $62, $72, $6E
```

Now for some other useful examples. Explain what each of them does. Run the programs and using memory dump, verify that your explanation is correct. Each of the functions start with an ORG command.

```

                                ORG    $2500
                                LDAB   #$10
                                LDX    #$3000
CALLOOH                        TSTB
                                BEQ    CALLAY
                                LDAA   0,X
                                ADDA  $10,X
                                STAA  $20,X
                                INX
                                DECB
```

```

                                BRA CALLOOH
CALLAY
                                SWI

                                ORG $3000
                                FCB $55, $6F, $66, $4D, $44, $62, $72, $6E
                                FCB $41, $42, $43, $44, $45, $46, $47, $48
                                FCB 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

; NOTE: YOU CAN ENTER NUMBERS EITHER IN DECIMAL OR HEX!
; AFTER YOU RUN THE PROGRAM, DO
; MD 3000 302F
; TO SEE WHAT THE PROGRAM DOES

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                                ORG      $2600
                                LDAA    #$0A
                                JSR     OUTA
                                JSR     OUTA

                                LDAB    #$10
                                LDX     #$3020

KINGS
                                TSTB
                                BEQ     CABBAGES

                                JSR     OUT1BYT
                                LDAA    #$2C
                                JSR     OUTA
                                LDAA    #$20
                                JSR     OUTA

                                DECB
                                BRA     KINGS

CABBAGES
                                LDAA    #$0A
                                JSR     OUTA
                                JSR     OUTA

                                SWI

```