Laboratory 2

Copying Null-Terminated Lists Program

Concepts:

- Writing programs in assembly using CodeWarrior

Objectives:

- Develop a program to meet a specific list of requirements in assembly using the CodeWarrior IDE, and debug it for correctness.

Files Needed:

- lab02.zip from Blackboard

Introduction:

An array is a group of items or elements stored in consecutive memory locations. Arrays are also called tables, lists, and sometimes strings, and we will often use these terms interchangeably at the assembly level. There are two pieces of information that define an array: where the array is stored in memory and the size and type of each array element. For this course, we will assume that arrays only contain one type of element. For example, we may have an array of all one-byte unsigned numbers, an array of all two-byte signed numbers, etc.

There are three main methods of defining where an array is stored in memory.

- Starting address and length: We supply the memory address where the first element is located in memory and the number of items.

- Starting address and final address: We supply the memory address of the first element and the memory address of the last element.

- Starting address with end-of-array value: We supply the memory address of the first element of the array. After the last element of the array, a predefined value is stored indicating that the end of the array has been reached. (Note that this terminating value cannot be a valid value for an array element.)

Assignment:

A common form of storing list of characters is a “null-terminated string”. This means that a null, or 0x00, is used to denote the end of the string. The null is included in the memory storage for the string, but it is not reported in the length. In ASCII, each character is assigned a value from 0x01 to 0x7F. When a program determines the length of an ASCII string, it starts at the first character and counts non-zero bytes (or encodings) until it reaches the 0x00. For example, the figure below shows how the word “Hello” is stored in memory. Note that it uses 6 bytes of storage, but a high-level programming language would state the length as 5. The escape sequence “/0” is often used to explicitly show the null as a character.

‘H’	‘e’	‘l’	‘l’	‘o’	/0
0x48	0x65	0x6C	0x6C	0x6F	0x00

The program provided in lab02.zip determines the length of a null-terminated string. It meets the following requirements:

1. The starting address of the list is supplied in address 3000h.

2. The end of the array is indicated by the value 00h.

3. The program must correctly handle arrays up to 255 elements not including the end-of-array value.

4. The length is returned as a one-byte value in address 3002h.

5. Assume that the inputs are valid. (i.e. only values from 00h to 7Fh are present, the array does not overlap the input/output locations, etc.)

6. (for practical reasons) Use only assembly instructions already covered in lecture.

```
          ORG  $3020; create a small array of data using "define constant byte"
          dc.b  $64,$45,$ff,$00

          ORG  $3040; create a small array of data using "define constant byte"
          dc.b  $64,$45,$64,$56,$60,$2f,$00

          ORG  $3060; create a small array of data using "define constant byte"
          dc.b  $64,$45,$64,$45,$64,$45,$64,$45,$64,$45,$27,$00

          ORG  $3100
          dcb.b 500, $ff
          dc.b $00

; code section
MyCode:    SECTION
main:
_Startup:
Entry:
           LDAB #0              ; This will count the length
           LDX $3000            ; point X at the first character
loop:      LDAA 0,X             ; Grab a byte from the array
           BEQ foundEnd         ; If it's 0x00, we found the end
           INCB                 ;...otherwise increment the length
           INX                  ; point to the next character
           BRA loop             ; and check again
foundEnd:  STAB $3002           ; output the final length
endmain:   BRA endmain
```

Figure 1 - Program to count 64h's

Perform the following steps.

1. Download lab02.zip and unzip it. Open the project file in CodeWarrior, and you should see that *main.asm* contains the program from Figure 1. Note that the file uses *dc.b* and *dcb.b* to place data in memory so that you don't have to do it manually. We will cover these statements in detail later in the term.

2. Start the debugger. Using the memory window, manually enter the values 30h and 20h into locations 3000h and 3001h respectively. Enter the value $FF in memory location 3002h. This ensures that we see the program change the output.

3. Generate a line-by-line program trace showing the values of the PC, the A, B, and X registers, and the N, Z, V, and C condition code bits from the beginning of the code until *"endmain BRA endmain"* is reached.

Question 1: What change(s) must be made to the program to process a list that begins at address 3040h?

Question 2: What change(s) must be made to the program to store the answer at address 3010h?

Question 3: What change(s) must be made to the program to handle a list with a two-byte length, given that the output value would now be written to 3002h and 3003h? Hint: This means that the answer requires two bytes, and several changes are required.

4. Modify the program to handle a two-byte length as asked in Question 3. Try the program on the data that starts at 3100h. This *dcb.b* causes 500_{10} bytes of FFh to be placed in memory, and the *dc.b* places a 00h after that. This creates a null-terminated string with a length of 500.

   You do not need to demonstrate this code, but it must be submitted with the lab.

5. Start a new project. Write a program that copies the values from a null-terminated source string to a destination string. It must meet the following requirements:
   - The address of the source array is provided in 3000h.
   - The address of the destination array is provided in 3002h.
   - The source array is assumed to have a valid 00h terminator. The array data may be any value from 01h to FFh, and it is all valid.
   - The program must copy all of the bytes from the source array to the destination array, and the destination array must also have the null-terminator at the end. (Hint: It may be easier to add the 00h at the end instead of explicitly copying it.)
   - The program may assume all inputs are valid. I.e. arrays do not overlap, input address point to valid memory, etc.

6. Test your data on the four null-terminated arrays provided in Figure 1 (i.e. cut and paste them into your new project.) You must provide one of the starting addresses in 3000h, and you must point to enough space for the destination in 3002h. Also test your program on a string of length 0 (i.e. just the null character).

Question 4: How many bytes of code is your assembly program?

Deliverables/Scoring:

Successful demonstration of the program is required for acceptance of the lab report, then

- 15 points - Compliance with posted lab report guidelines.
- 15 points - Answers to questions.
- 20 points - Program trace from step 3
- 20 points - Program from step 4
- 30 points - Program from step 5

Submit the deliverables according to the lab report guidelines posted on Blackboard.

Helpful Hints:

- You can have multiple sets of data in the code at the same time, shown below.

```
ORG  $3020    ; create a small array of data using "define constant "
dc.b  $64,$45,$ff,$00

ORG  $3040    ; create a small array of data using "define constant "
 dc.b  $64,$45,$64,$56,$60,$2f,$00

ORG  $3060    ; create a small array of data using "define constant "
dc.b  $64,$45,$64,$45,$64,$45,$64,$45,$64,$45,$27,$00
```

Each data set is given a new, known address, and they are chosen so that they don't overlap. You can now enter 30h and 20h in 3000h and 3001h to run the program with the first set of data (length of 3), or enter 31h and 40h to use the second data set with length 6, and so on, without having to keep retyping different test cases. You may use this format to enter your own cases.