# Laboratory 2

# Compare-Two-Tables 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**

The code in Figure 1 implements an algorithm for counting how many 64h’s appear in a list of unsigned one-byte values. This list corresponds to the last term grades for the Microcomputer’s I quiz, and therefore only values between 00h and 64h would be possible (010 to 10010). The value FFh can be used as an end-of-array value, since there is no possibility for FFh (or 25510) to be a valid number in the array. Therefore, the array of values will be specified by supplying the address of the first element in address 3000h, and a value of FFh will indicate the end.

Next, we need a means of reporting the answer. In this course, this is usually done by storing it to a specific, predetermined address. Addresses 3000h and 3001h are already used (addresses require two bytes), so we will return the answer in 3002h. By only reserving one byte for the answer, our program will be limited to arrays up to 255 elements. The details discussed above are typically represented as a list of basic requirements, shown below.

1. The starting address of the input array is supplied in address 3000h.
2. The end of the array is indicated by the value FFh.
3. The program must correctly handle arrays up to 255 elements not including the end-of-array value.
4. The correct number of locations containing 64h is returned as a one-byte value in address 3002h.
5. You may assume that the inputs are valid and that the table does not overlap the outputs.

ORG $3000

table: ds.b 2 ; starting address of table of one-byte numbers

; terminated by FFh

answer: ds.b 1 ; count of 64h's found in "table"

ORG $3020 FYI ; TO FACILITATE DEBUGGING

; create a small array of data using "define constant byte"

data: dc.b $64,$45,$ff,$64

; don't mess with anything from here...

MyCode: SECTION

LDX table ; point to first byte of input table

MOVB #0,answer ; reset answer to 0

loop LDAA 0,X ; examine current byte

CMPA #$ff ; end-of-table delimiter found, end

BEQ endmain

CMPA #$64 ; compare byte to 100...

BNE not100

INC answer ; ...and increment count if found

not100 INX ; point to next byte in table

BRA loop

endmain BRA endmain

Figure 1 - Program to count 64h’s

The program in Figure 1 correctly meets these requirements. Perform the following tasks.

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 to create a list of four bytes in memory starting at 3020h so you don’t have to manually enter test data each time, and this would be removed after successfully debugging the program.*
2. Enter the value 30h and 20h into locations 3000h and 3001h respectively to point the program to the array that has already been defined.
3. Complete a program trace including the PC, the registers used, 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 changes must be made to the program to process a list that begins at address 3800h?

**Question 2**: What changes must be made to the program to store the answer at address 3100h?

**Question 3**: What changes must be made to the program to handle a list with a two-byte length? Hint: This means that the answer requires two bytes, and several changes are required.

1. Write a program that compares two equal-length arrays of two-byte numbers word-by-word and determines if all pairs of words are the same (i.e. the arrays are identical). Your program must meet the following requirements.
   1. The address of the first array is supplied in location 3000h.
   2. The address of the second array is supplied in location 3002h.
   3. The two-byte length of both arrays is supplied in location 3004h. The length is the number of elements, not the number of bytes.
   4. If the two arrays are identical, the program must store the result FFh in location 3006h, otherwise it must store the result 00h to location 3006h.
   5. Arrays with lengths of 0000h should be reported as identical.

Write your program in assembly using CodeWarrior and test your program. When it is working, demonstrate it to the instructor. It is recommended to simply edit the sample program in Lab02.

**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
* 50 points – Program from step 4

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 byte"

data1: dc.b $64,$45,$ff,$64

ORG $3120

; create a small array of data using "define constant byte"

data2: dc.b $64,$45,$64,$56,$60, $ff,$64

ORG $3220

; create a small array of data using "define constant byte"

data3: dc.b $64,$45, $64,$45$64,$45$64,$45$64,$45,$ff,$64

Notice that the labels must be different. Also, 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 3001 to run the program with data1, or enter 31h and 20h to use data1, and so on, without having to keep retyping different test cases.

* This program only has two answers, FFh for being the same, and 00h for being different. An easy (relatively speaking) approach is to assume that the two tables are the same, and start the program by storing FFh in the answer.

When the program compares two values and finds that they are the same, do nothing and continue to the next pair. If they are different, update the answer to 00h and exit. If the program reaches the end of the list, it means that all pairs were the same, and the initial guess of FFh was correct. No more action is needed. Since the length may be 0, the program should be written with a while-loop form, and this will gracefully handle the length of 0.