***Efficient Embedded Course***

**LAB 2**

**LAB EXERCISE:**

**C AS IMPLEMENTED IN ASSEMBLY**

Note. The figures shown in solutions may vary subject to different experimental environments

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc87593270)

[1.1 Lab overview 1](#_Toc87593271)

[2 Learning Outcomes 1](#_Toc87593272)

[3 Requirements 1](#_Toc87593273)

[4 Procedure 2](#_Toc87593274)

[4.1 Assembly Code Listing 2](#_Toc87593275)

[4.1.1 Examine the function INIT\_LIST 2](#_Toc87593276)

[4.1.2 Examine the function FIND\_IN\_LIST 2](#_Toc87593277)

[4.2 Map File 3](#_Toc87593278)

[4.2.1 Default Program 3](#_Toc87593279)

[4.2.2 Specifying C Data as Read-Only to save RAM 4](#_Toc87593280)

# Introduction

## Lab overview

In this lab exercise, you will compile a C program and examine the assembly language program output of the compiler and the map file output of the linker.

# Learning Outcomes

* Modify and compile a C program and observe the assembly listing and the map file.

# Requirements

In this lab, we will be using the following hardware and software:

* **Keil µVision5 MDK IDE**
  + Please see the included Getting Started with Keil guide on how to download and install Keil.
* **STM32 Nucleo-L552ZE-Q**
  + For more information, click [here](https://www.st.com/en/evaluation-tools/nucleo-l552ze-q.html).

# Procedure

1. Open the CinAsmLab project.
2. Verify that the Compiler will create assembly code listings (Options->Listing->C Compiler Listing) with maximum optimization (Options->C/C++->Optimization Level 3 (-O3).
3. Compile the program and examine the listing and map files to answer the following questions.

## Assembly Code Listing

### Examine the function INIT\_LIST

1. Which context (if any) is saved after entering the function?

r4, r5, lr

1. Which instruction restores context and returns from the subroutine?

POP {r4,r5,pc}

1. Which register is used as the loop counter variable i?

r0

1. Which assembly instructions perform the operation i\*2000? Explain how they work.

MOVS r2, #0x7d loads 0x7d (125) into register r1.

LSLS r2, r2, #4 shifts register r2 left by 4 bits, multiplying by 16. The contents are now 16\*125 = 2000.

MULS r2, r0, r2 multiplies r0 (which contains i) by 2000, placing the result in r2.

1. Which assembly instructions access operand offset[i]? Explain how they work.

LDR r4,|L1. 104| loads r4 with the value stored at address |L1. 104| which is ||.data||. Looking in the map file shows us that .data starts at 0x2000\_00d8, and the value of **offset** is 0x2000\_00d8. So r4 points to the beginning of the array **offset**.

The array holds integers, each of which is four bytes long. To find the element address we need to calculate the address b

LSLS r3,r0,#2 shifts r0 (variable i, which is the column number) left by two bits, multiplying it by 4 because each array element is 4 bytes long.

The final address is the array’s start address (in r4) + 4\*column number (in r3)

LDR r5, [r4,r3] loads r5 from memory address r4+r3, giving us offset[i].

### Examine the function FIND\_IN\_LIST

1. Which context (if any) is saved after entering the function ?

No context is saved.

1. Which instruction restores context and returns from the subroutine? Why is this different from init\_list?

BX lr returns from the subroutine. No context is saved on the stack, so none needs to be restored. Hence the instruction BX lr is used.

1. Which register holds the argument key?

r0

1. Which instructions perform the loop repeat test? Explain how they work.

ADDS r0,r0,#1 adds one to variable i.

CMP r0,#0xa compares r0 with 10 (0xa) by subtracting 10 from r0.

BCC branches to label |L1.52| to repeat the loop if the carry flag is clear, indicating that 10-i did not result in a borrow, meaning that i is less than or equal to 10.

## Map File

### Default Program

1. Examine the Image Symbol Table (Global Symbols subsection) and complete the table below.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Starting Address** | **Size** |
| $Super$$main | 0x0000\_0851 | 26 |
| list\_init | 0x0000\_0809 | 46 |
| list\_find | 0x0000\_0837 | 26 |
| list | 0x2000\_0100 | 40 |
| offset | 0x2000\_00d8 | 40 |

1. Examine the “Memory Map of the image”. List the section name, size, type, attributes for each section from object main.o:

.text, 0x6c bytes, Code, Read-only

.data, 0x28 bytes, Data, Read-write

.bss, 0x28 bytes, Zero, Read-write

1. What is the size of the STACK? Does it match the value declared in the memory layout file (scatter file or startup.s file)?

0x0400 bytes = 1024 bytes. Yes, it matches this:

ARM\_LIB\_STACK (0x20000000 + 4096) EMPTY -0x0400

1. Examine the “Image Component Sizes” section of the map file. Complete the table to show how much memory space is needed for main.o.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Code | data | RO Data | RW Data | ZI Data | Debug |
| non-const offset | 108 | 10 | 0 | 40 | 40 | 1535 |

### Specifying C Data as Read-Only to save RAM

Change your C source code so that the offset array is declared as “const” and rebuild the project.

1. Examine the updated Image Symbol Table (Global Symbols subsection) and complete the table below. Identify what has changed and why.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Starting Address** | **Size** |
| $Super$$main | 0x0000\_0851 | 26 |
| list\_init | 0x0000\_0809 | 46 |
| list\_find | 0x0000\_0837 | 26 |
| list | 0x2000\_00d8 | 40 |
| offset | 0x0000\_0a84 | 40 |

The array “offset” is now located in ROM (at address 0x0000\_0a84) instead of in RAM (at address 0x2000\_00d8). The array “list” now resides in the first position in RAM (0x2000\_00d8) instead of after offset (0x2000\_0100) since offset has moved.

1. What changes do you see in main.txt?

offset is now in the area .constdata instead of .data:

AREA ||.constdata||, DATA, READONLY, ALIGN=2

1. Examine the updated “Memory Map of the image” List the section name, size, type, attributes for each section from object main.o:

.text, 0x6c bytes, Code, Read-only

.constdata, 0x28 bytes, Data, Read-only

.bss, 0x28 bytes, Zero, .read-write

1. Copy the values you determined in question 16 above into the first row. Then examine the “Image Component Sizes” section of the map file and complete the second row to show how much memory space is needed for main.o. What has changed, and why?

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Code | data | RO Data | RW Data | ZI Data | Debug |
| non-const offset | 108 | 10 | 0 | 40 | 40 | 1535 |
| const offset | 108 | 10 | 40 | 0 | 40 | 1539 |

This version of main.o needs no RW data, just 40 bytes of RO data to hold the const offset array. This reduces the total RAM requirements. Typically MCUs have much more Flash ROM than RAM, so this optimization is useful for saving RAM.