Project 1

Jordan Hayes

EGR 424: Microcontroller Applications

Date Submitted: June 9, 2022

Instructor: Professor Parikh

**Scope**

The objectives of this project were to develop familiarity with ARM assembly language and the ARM development tool suite. Skills are demonstrated by studying the assembly representation of a C function. In assembly language, the short block of letters representing the instruction are read as macros containing the addresses of those functions in memory. The functions will be identified and their function and context within the program will be described. Background information on the registers and their purposes are listed below.

**Table A.1 Core Register Descriptions**

|  |  |
| --- | --- |
| **Register** | **Description** |
| **R0** | **General Purpose [Core]** |
| **R1** | **General Purpose [Core]** |
| **R2** | **General Purpose [Core]** |
| **R3** | **General Purpose [Core]** |
| **R4** | **General Purpose [Core]** |
| **R5** | **General Purpose [Core]** |
| **R6** | **General Purpose [Core]** |
| **R7** | **General Purpose [Core]** |
| **R8** | **General Purpose [Core]** |
| **R9** | **General Purpose [Core]** |
| **R10** | **General Purpose [Core]** |
| **R11** | **General Purpose [Core]** |
| **R12** | **General Purpose [Core]** |
| **R13** | **Main Stack Pointer, Process Stack Pointer** |
| **R14** | **Link Register** |
| **R15** | **Program Counter** |

**Table A.2 PSR Register Flag Descriptions**

|  |  |
| --- | --- |
| **Flag Bit** | **Description** |
| **N** | **Negative Result Indication**  **Stores bit 31 of the result of instruction** |
| **Z** | **Zero Result Indication**  **Is set to 1 if the result of the operation is zero** |
| **C** | **Result with Carry Indication**  **Stores the value of the carry bit if it occurred as the result of an addition or borrow** |
| **V** | **Overflow indication**  **Set to 1 if an overflow occured** |
|  |  |

**Code**

*The C code and Assembly code are shown below, respectively.*

The C function to be analyzed is called strpbrk. The function returns a pointer (char \*), and takes two constant char \* as input parameters.

Text

Description automatically generated**Figure x: Original C code**

The Assembly code was compiled using the -O3 optimization level.

Graphical user interface, text, application

Description automatically generatedText

Description automatically generated with medium confidence**Figure x: Assembly Code Generated by Compiler, Commented and Edited to Compile**

**Line-by-Line Analysis**

***First instruction on line 24***

|  |  |
| --- | --- |
| **Line: 24** | **Push {r4, lr}** |
| **Instruction** | **Push Multiple Registers** |
| **Description** | **Stores R4 and then LR to the stack** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000004** | **0x00000004** |
| **R3** | **0x00000000** | **0x00000000** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x20010000** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x00000598** | **0x0000059A** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction changes the value of the stack register to the next address in stack. When a BL or BX instruction performs a subroutine such as *strpbrk(),* LR is set to the return address. R4 is pushed as well. R4 is pushed because it is assumed by the compiler that it is an important input value needed later. LR is pushed as required by the instruction, which allows a return to the next instruction to be executed in the calling function.

|  |  |
| --- | --- |
| **Line: 25** | **ldrb r3, [r0, #0]** |
| **Instruction** | **Load Register Byte Immediate** |
| **Description** | **Since the offset is #0, the resulting address is the first byte of R0. The byte is zero extended and written to R3.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000004** | **0x00000004** |
| **R3** | **0x00000000** | **0x00000005** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x0000059A** | **0x0000059C** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** The instruction helps to complete the objection of the C function by loading the value of R0 into R3 to be compared in the next instruction. R3 corresponds to \*c, set to the same value as s2 as in line 38 of the C code.

|  |  |
| --- | --- |
| **Line: 26** | **cbnz r3, .L2** |
| **Instruction** | **Compare and Branch on Non-Zero** |
| **Description** | **Compares the value in Register R3 with zero. If the value is not zero, it branches to label .L2. If the value is zero, the instruction does not branch. Does not affect condition flags** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000004** | **0x00000004** |
| **R3** | **0x00000005** | **0x00000005** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x0000059C** | **0x000005B0** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction corresponds to line 39 in the C code. In the C code; the if statement checks that the value is zero. If it is, the next instructions will return a NULL pointer. If it is not, the C program moves on to the while loop structure. In the assembly, the program branches to its while loop structure as long as R3 is not zero.

|  |  |
| --- | --- |
| **Line: 27** | **mov r0, r3** |
| **Instruction** | **Move (register)** |
| **Description** | **Copies a value from register R3 to the destination, R0. Does not affect condition flags** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000023** | **0x00000000** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000004** | **0x00000004** |
| **R3** | **0x00000000** | **0x00000000** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x0000059E** | **0x000005A0** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction copies the value from R3 into R0 and corresponds to line 40 of the C code, because the C code initializes the NULL character returned as char \*, and R0 is the value returned for this function; this is the logic for having this instruction here.

|  |  |
| --- | --- |
| **Line: 28** | **Pop {r4, pc]** |
| **Instruction** | **Pop Multiple Registers** |
| **Description** | **Loads R4 and PC from the stack. When the registers loaded include PC, the word PC is treated as an address or exception return value. Bit<0> must be 1** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000000** | **0x00000000** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000004** | **0x00000004** |
| **R3** | **0x00000000** | **0x00000000** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x20010000** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005A0** | **0x00000586** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** Pop can take many encodings and registers. In this case, pop is used to extract R4, then PC is included in the POP list, and is treated here as an exception return in case an empty string or null character was passed to the function, the value of which is contained in R0 from the previous instruction.

|  |  |
| --- | --- |
| **Line: 30** | **Cmp r2, r4** |
| **Instruction** | **Compare (Register)** |
| **Description** | **Subtracts non shifted R4 from R2, updating the condition flags based on the result.**  **Z = 1 if the argument registers are equal.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000005** |
| **R3** | **0x20000004** | **0x20000004** |
| **R4** | **0x00000041** | **0x00000041** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005A2** | **0x000005A4** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **1** |
| **Z** | **0** | **0** |
| **C** | **1** | **0** |
| **V** | **0** | **0** |

**Big Picture:** This instruction is executed after a branch from assembly line 36, and corresponds to line 46 of the C code. Here N = 1, so the result of CMP subtraction was negative, meaning the second argument was larger. The registers represent variables s1 and c, and the values at their addresses are compared. This is important for both forms of the function because it determines whether a character exists in both strings.

|  |  |
| --- | --- |
| **Line: 31** | **Beq .L4** |
| **Instruction** | **Branch If Equal** |
| **Description** | **Causes a branch to the label .L4 if the result of the preceding CMP is zero, or Z = 1** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000005** |
| **R3** | **0x20000004** | **0x20000004** |
| **R4** | **0x00000041** | **0x00000041** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005A4** | **0x000005A6** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **1** | **1** |
| **Z** | **0** | **0** |
| **C** | **0** | **0** |
| **V** | **0** | **0** |

**Big Picture:** Enters this line as a result of continuation from line 30. The instruction corresponds to the break command on line 47 of the C code. If the values were equal, this instruction would send the program to label .L4, which returns the pointer to the character found.

|  |  |
| --- | --- |
| **Line: 32** | **Adds r3, r3, #1** |
| **Instruction** | **Add (Immediate)** |
| **Description** | **Adds immediate 1 to register R3 and writes the result to R3** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000005** |
| **R3** | **0x20000004** | **0x20000005** |
| **R4** | **0x00000041** | **0x00000041** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005A6** | **0x000005A8** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **1** | **0** |
| **Z** | **0** | **0** |
| **C** | **0** | **0** |
| **V** | **0** | **0** |

**Big Picture:** This instruction corresponds to the incrementation of the control loop variable in the C code. The value in R3 corresponds to the value of c, which is the address of one input string. The value is incremented by 1, moving the reference to the next character of the input string. This is important for the functions to search for whether a character in s2 occurs in s1

|  |  |
| --- | --- |
| **Line: 34** | **Ldrb r4, [r3, #0]** |
| **Instruction** | **Load Register Byte Immediate** |
| **Description** | **Since the offset is #0, the resulting address is the first byte of R3. The byte is zero extended and written to R4.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000005** |
| **R3** | **0x20000004** | **0x20000004** |
| **R4** | **0x00000000** | **0x00000041** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005A8** | **0x000005AA** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** The program branches here as a result of line 44 or continuation from line 32. The first byte of R3 is loaded into R4 to continue function. If continuing from line 32 (assembly), the incremented address in R3 is the next character in an input string and is copied to R4 to repeat this section of code under .L7, which ensures the next character is not zero.

|  |  |
| --- | --- |
| **Line: 35** | **Cmp r4, #0** |
| **Instruction** | **Compare (Immediate)** |
| **Description** | **Subtracts 0 from R4, updating the condition flags based on result, then discards result.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000005** |
| **R3** | **0x20000004** | **0x20000004** |
| **R4** | **0x00000041** | **0x00000041** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005AA** | **0x000005AC** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **0** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** The purpose of this section is to quickly check that the next character in the string s2 is also not a null character. The function will use the result of this comparison to determine whether or not to branch. The part of the program also checks whether the next character is NULL. The line corresponds to adherence to the continuation statement of the loop, where \*c must remain TRUE.

|  |  |
| --- | --- |
| **Line: 36** | **Bne .L5** |
| **Instruction** | **Branch if Not Equal** |
| **Description** | **Branches to label .L5 if the result of the preceding CMP is 0, or Z = 1** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000005** |
| **R3** | **0x20000004** | **0x20000004** |
| **R4** | **0x00000041** | **0x00000041** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005AC** | **0x000005A2** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **0** | **0** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture**: If it is not zero, the end of c (s2) has not been reached. The instruction then causes a branch to structure of code where the values are again compared.

|  |  |
| --- | --- |
| **Line: 37** | **B .L13** |
| **Instruction** | **Branch** |
| **Description** | **Branches to label .L13 unconditionally** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000005** |
| **R3** | **0x20000006** | **0x20000006** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005AE** | **0x000005C4** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** Enters here as a result of no branch on previous line. The instructions execution indicates a zero value was reached in the string. The program branches to .L13 where the variable is incremented. If the character is NULL, no character in s2 is the same as the character in s1 currently being compared by the for loop.

|  |  |
| --- | --- |
| **Line: 39** | **Mov r3, r1** |
| **Instruction** | **Move (Register)** |
| **Description** | **Copies a value from register R1 to the destination, R3. Does not affect condition flags.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000004** | **0x00000004** |
| **R3** | **0x00000005** | **0x20000004** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005B0** | **0x000005B2** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction is often branched to right away when input s1 is evaluated as non-zero. The input value is copied from R0 to R2.

|  |  |
| --- | --- |
| **Line: 41** | **Ldrb r2, [r0, #0]** |
| **Instruction** | **Load Register Byte Immediate** |
| **Description** | **Since the offset is #0, the resulting address is the first byte of R0. The byte is zero extended and written to R2.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000004** | **0x00000005** |
| **R3** | **0x20000004** | **0x20000004** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005B2** | **0x000005B4** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction corresponds to a helper for line 46 in C. Arrives here also from line 53 in assembly because it is the first statement in the for loop. The pointer s1 is compared to the pointer c, so a value must first be loaded into a register that represents s2.

|  |  |
| --- | --- |
| **Line: 42** | **Cbz r2, .L4** |
| **Instruction** | **Compare and Branch on Zero** |
| **Description** | **Compares value in register R2 with zero, if the result is 0, the program branches forward to label .L4. If R2 is anything but 0, the instruction does not branch. Does not affect condition flags.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000005** |
| **R3** | **0x20000004** | **0x20000004** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005B4** | **0x000005B6** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction corresponds to line 44 of the C code, and it ensures that the variable is not equal to zero. If R2 is zero, corresponding to variable c, the program branches to .L4 on line 45 of the assembly and eventually returns. The function of this instruction is evaluate whether or not to continue the loop.

|  |  |
| --- | --- |
| **Line: 43** | **Mov r3, r1** |
| **Instruction** | **Move (Register)** |
| **Description** | **Copies a value from register R1 to the destination, R3. Does not affect condition flags.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000005** |
| **R3** | **0x20000004** | **0x20000004** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005B6** | **0x000005B8** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction takes the value held in R1 and copies it to R3, setting the program up for another comparison in line 49 of the C code.

|  |  |
| --- | --- |
| **Line: 44** | **B .L7** |
| **Instruction** | **Branch** |
| **Description** | **Program branches to target address, label .L7** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000005** |
| **R3** | **0x20000004** | **0x20000004** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005B6** | **0x000005A8** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This assembly instruction branches the program to a section of assembly code that will compare the two input string characters.

|  |  |
| --- | --- |
| **Line: 46** | **Ldrb r3, [r3, #0]** |
| **Instruction** | **Load Register Byte Immediate** |
| **Description** | **Since the offset is #0, the resulting address is the first byte of R3. The byte is zero extended and written to R3.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000006** | **0x00000006** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000000** | **0x00000000** |
| **R3** | **0x20000006** | **0x00000000** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005BA** | **0x000005BC** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **0** | **0** |
| **C** | **0** | **0** |
| **V** | **0** | **0** |

**Big Picture:** Comes here from line 42 as a result of zero, or from line 31 if the characters evaluated were equal. Overwrites other values except for first byte in R3. This is how the function returns a pointer to a single character. R3 may be the character found in both strings, or a null character.

|  |  |
| --- | --- |
| **Line: 47** | **Cmp r3, #0** |
| **Instruction** | **Compare (Immediate)** |
| **Description** | **Subtracts 0 from value in R3, updating the condition flags based on result, then discards result.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000006** | **0x00000006** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000000** | **0x00000000** |
| **R3** | **0x00000000** | **0x00000000** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005BC** | **0x000005BE** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **0** | **1** |
| **C** | **0** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction corresponds to line 54 in the C code, comparing the value in R3 as representative of the variable \*c. The C code checks whether the variable is equal to a null character. The assembly does the same thing by comparing it to zero, and the Z bit is set high.

|  |  |
| --- | --- |
| **Line: 48** | **It eq** |
| **Instruction** | **If Equal, Then** |
| **Description** | **Begins an IT structure, where the condition to be met is Eq** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000006** | **0x00000006** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000000** | **0x00000000** |
| **R3** | **0x00000000** | **0x00000000** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005BE** | **0x000005C0** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction corresponds to line 54 in the C code, using the result of the preceding command to determine the execution of the following lines. It checks the value of Z bit.

|  |  |
| --- | --- |
| **Line: 49** | **Moveq r0, #0** |
| **Instruction** | **Move If Equal (Immediate)** |
| **Description** | **Copies the immediate value to the destination, R0, if the result of the CMP preceding IT is 0. Does not affect condition flags.** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000006** | **0x00000000** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000000** | **0x00000000** |
| **R3** | **0x00000000** | **0x00000000** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005C0** | **0x000005C2** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction corresponds to line 55 of the C code. If Z becomes 1 as a result of line 47 of the assembly code, this instruction is executed and zero is moved into R0 (setting it to NULL), which is the value returned by the function.

|  |  |
| --- | --- |
| **Line: 50** | **Pop {r4, pc}** |
| **Instruction** | **Pop Multiple Registers** |
| **Description** | **Loads R4 and PC from the stack. When the registers loaded include PC, the word PC is treated as an address or exception return value. Bit<0> must be 1** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000000** | **0x00000000** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000000** | **0x00000000** |
| **R3** | **0x00000000** | **0x00000000** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x20010000** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005C2** | **0x00000586** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **1** |
| **C** | **1** | **1** |
| **V** | **0** | **0** |

**Big Picture:** This instruction corresponds with the return statement on line 57 of the C code. In this case, pop is used to extract R4, then PC is included in the POP list, and is treated here as an an empty string or null character was passed to the function, the value of which is contained in R0 from the previous instruction.

|  |  |
| --- | --- |
| **Line: 52** | **Adds r0, r0, #1** |
| **Instruction** | **Add (Immediate)** |
| **Description** | **Adds the immediate value 1 to the register value R0, and stores the value into R0. The command updates the condition flags based on result** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000005** | **0x00000006** |
| **R3** | **0x20000006** | **0x20000006** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005C4** | **0x000005C6** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **1** | **0** |
| **C** | **1** | **0** |
| **V** | **0** | **0** |

**Big Picture**: Comes here from line 37. This instruction corresponds to the increment of s1++ in the C code and is used to sift through strings and ignore null characters.

|  |  |
| --- | --- |
| **Line: 53** | **B .L8** |
| **Instruction** | **Branche** |
| **Description** | **Branches program execution to label .L8** |

**Core Register Value Results**

|  |  |  |
| --- | --- | --- |
| **Register** | **Value Before Instruction** | **Value after Instruction** |
| **R0** | **0x00000005** | **0x00000005** |
| **R1** | **0x20000004** | **0x20000004** |
| **R2** | **0x00000006** | **0x00000006** |
| **R3** | **0x20000006** | **0x20000006** |
| **R4** | **0x00000000** | **0x00000000** |
| **R5** | **0x00000000** | **0x00000000** |
| **R6** | **0x00000000** | **0x00000000** |
| **R7** | **0x00000000** | **0x00000000** |
| **R8** | **0x00000000** | **0x00000000** |
| **R9** | **0x00000000** | **0x00000000** |
| **R10** | **0x00000000** | **0x00000000** |
| **R11** | **0x00000000** | **0x00000000** |
| **R12** | **0x000005F8** | **0x000005F8** |
| **Stack** | **0x2000FFF8** | **0x2000FFF8** |
| **Link Register** | **0x00000587** | **0x00000587** |
| **Program Counter** | **0x000005C6** | **0x000005B2** |

**PSR Flag Results**

|  |  |  |
| --- | --- | --- |
| **Flag Bit** | **Value Before Instruction** | **Value After Instruction** |
| **N** | **0** | **0** |
| **Z** | **0** | **0** |
| **C** | **0** | **0** |
| **V** | **0** | **0** |

**Big Picture:** This instruction is used to branch back into the for loop structure to compare all characters in c to the character from s1.

**Timing Analysis**

Each word of an instruction including its extension words must be "fetched" from memory. Each of these memory reads requires one clock cycle. In preparation for the next cycle, the program counter is incremented by 2 during each of these memory reads. The timing analysis below shows parameterized functions for clock cycles per instruction. Descriptions below.

Text

Description automatically generated

Parameter P can be 1-3 cycles, an average of 2 was taken for table analysis.

**Table D-1: Assembly Timing Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Line** | **Instruction** | **Notes** | **Function for Clock Cycles** | **Total Cycles** |
| **24** | **Push{r4, lr}** | **Stores exit to main address, 1 cycle per register** | **1 + N** | **2** |
| **25** | **Ldrb r3, [r0, #0]** | **Neighboring load and store single instructions can be pipelined and done in a single instruction** | **2** | **2** |
| **26** | **Cbnz r3, .L2** | **Conditional branch happens in 1 cycle if branch is not taken** | **1 + P** | **3** |
| **27** | **Mov r0, r3** | **Is an end case condition. Encountered if input is empty or null**  **Is 1+ P if moved to PC** | **1** | **1** |
| **28** | **Pop {r4, pc}** | **Exits function for end case. 1 cycle per register popped** | **1 + N** | **2** |
| **30** | **Cmp {r2, r4}** |  | **1** | **1** |
| **31** | **Beq .L4** | **Reference P param above. Only 1 cycle if branch is not taken.** | **1 + P** | **3** |
| **32** | **Adds r3, r3, #1** |  | **1** | **1** |
| **34** | **Ldrb r4, [r3,#0]** | **Neighboring load and store single instructions can be pipelined and done in a single instruction** | **2** | **2** |
| **35** | **Cmp r4, #0** |  | **1** | **1** |
| **36** | **Bne .L5** |  | **1+ P** | **3** |
| **37** | **B .L13** | **Unconditional Branch** | **1 + P** | **3** |
| **39** | **Mov r3, r1** |  | **1** | **1** |
| **41** | **Ldrb r2, [r0, #0]** |  | **2** | **2** |
| **42** | **Cbz r2, .L4** |  | **1 + P** | **3** |
| **43** | **Mov r3, r1** |  | **1** | **1** |
| **44** | **B .L7** |  | **1 + P** | **3** |
| **46** | **Ldrb r3, [r3, #0]** |  | **2** | **2** |
| **47** | **Cmp r3, #0** |  | **1** | **1** |
| **48** | **It** |  | **NA** | **NA** |
| **49** | **Moveq r0, #0** |  | **1** | **1** |
| **50** | **Pop {r4, PC}** |  | **1 + N** | **2** |
| **52** | **Adds r0, r0, #1** |  | **1** | **1** |
| **53** | **B .L8** |  | **1 + P** | **3** |

A wrapper program was created to run the subroutine, shown below.

**#include** "msp.h"

**#include** <stdio.h>

**#include** <string.h>

/\*\*

\* main.c

\*/

**void** **main**(**void**)

{

//WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer

//eturns a pointer to the character in str1 that matches one of the characters

//in str2, or NULL if no such character is found.

**char** \*str1 = "Hello";

**char** \*str2 = "World";

**char** \*result;

result = **strpbrk**(str1, str2);

**if**(result) {

**printf**("First matching character is: ");

}

**else** {

**printf**("Character not found");

}

**while**(1);

}

Changing the size of the inputs as well as their contents are the indicators for how long the processor will take to return the pointer. With each n characters in each input, the loop has a Big Oh equation of O(n) = n^2, since each character of str1 is compared with each individual character of str2. The equation represents a worst case scenario, since shorter strings can be entered and equivalent characters can be found.

**Big Picture**

The C library function finds the first character in s1 that matches any character specified in s2. The function checks s1 first to ensure it is not empty or NULL. If it is, a NULL pointer is returned. The while loop on line 42 is executed until the end of s1 is reached. The for loop on line 44 sets a local variable c to s2, its continuation requirement is that \*c does not point to NULL, and increments the character index pointed to in the string after iteration. If at line 46, the address of s1 is equal to the address of c, the loop is broken. The if on line 54 checks if the end of s2 was reached without finding a match. If it has, s1 is set to NULL, else, s1 is returned wherever a match was found and loop broken. There are contingencies within the C code, such as line 46, which caused a lot of confusion for me It was surprising, yet understandable, that the C code checks both parameters for null characters at all times, and the algorithm in which it is checked made the function slightly confusing. The lines of the assembly code were analyzed throughout this project and explained how they provide the same functionality through each line. The program begins by ensuring that s1 is not null or empty. .L2 is for a one-time move for the value at the start of s1 into a register. .L8 loads the second input value and begins the initialization of the control loop, it also serves as a double check for encountering the end of the string and breaking. .L7 is for reevaluation and continuation of the for loop. The for loop structure is continued under labels .L5 and is where each value is compared. .L13 is for skipping zero values within the string, and values are returned under label .L4.