

The ARM Software and Stack

Computer Architecture Exploitation and Security

Table of Contents

[Objectives 2](#_Toc46935891)

[Background Reading 2](#_Toc46935892)

[Important Information 2](#_Toc46935893)

[Introduction 3](#_Toc46935894)

[Problem 1 3](#_Toc46935895)

[Problem 2 5](#_Toc46935896)

[Problem 3 5](#_Toc46935897)

[Problem 4 7](#_Toc46935898)

L*abs must be submitted by the due date for full credit. After due date late submissions will be accepted for a period of one week (seven days) and the grade will be reduced by ten percent (10%) per day after due day.* ***Assignments that are submitted more than seven days late will receive a grade of zero (0).***

I certify that the work submitted in this assignment is my own and that it has not been taken in whole or in part from any other source. I understand that the penalty for plagiarism will include a grade of zero (0) for this assignment plus disciplinary action in accordance with SAIT policies.

**EVALUATION**:

|  |  |  |
| --- | --- | --- |
| Analyze and modify ARM code | 15 |  |
| Analyze immediate values | 10 |  |
| Opcode | 10 |  |
| Stack Analysis | 15 |  |
| TOTAL MARK | 50 |  |

Computer Architecture Exploitation and Security

The ARM Software and Stack

Objectives

This lab focuses on the following objectives:

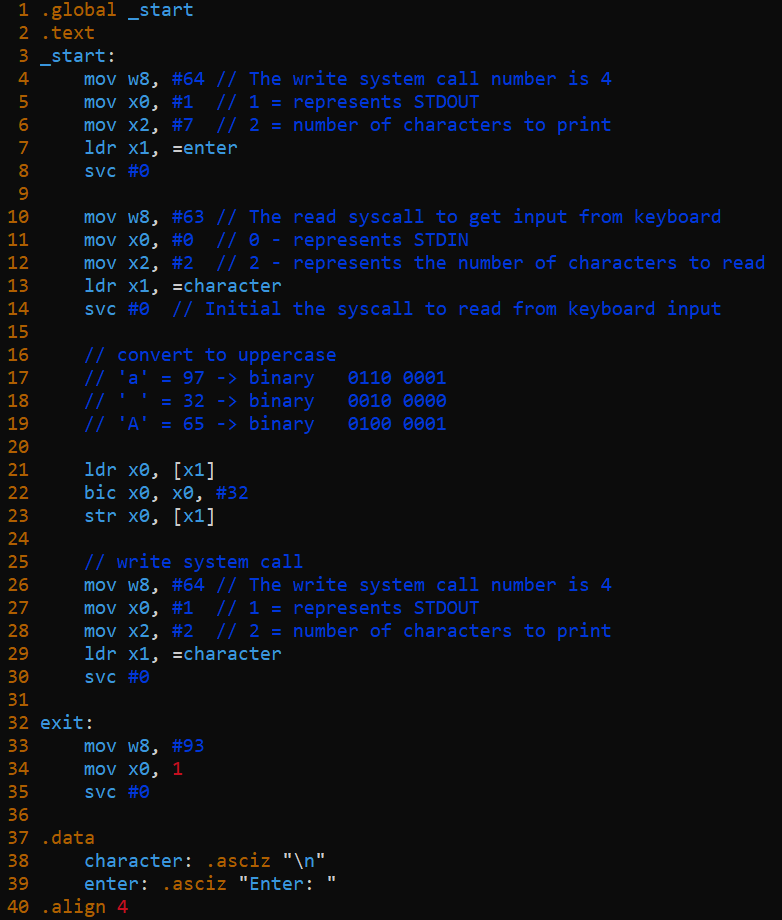
* Identify data type
* Use Memory instructions ldr and str
* Analyze stack instructions and functions
* Differentiate memory addressing modes
* Identify instructions opcode and decode

.

Background Reading

* DDI0406C ARM Architecture Reference Manual ARMv7-A and ARMv7-R Edition
* DDI0344K Cortex-A8 Technical Reference Manual
* DEN0013D ARM Cortex-A Series Programmer's Guide
* <https://developer.arm.com/documentation/dui0204/j/writing-arm-assembly-language?l>
* <https://developer.arm.com/documentation/ddi0487/latest>

# Problem 1 Analyze and Modify ARM code \_\_\_/15



1. Us **as** and **ld** to generate the executable
2. Run, analyze the code and the output. Answer the following questions:
   1. What does the program do? **1pt**

**It takes a character and prints out the same character uppercased**

* 1. What is the purpose of register (w8)? **1pt**

**w8 stores the system call number**

* 1. What registers are used to pass the **parameters** to the system calls? **1pt**

**x0 and x2**

* 1. In the data section the **asciz** directive is used. What is the difference between directives **asciz** and **ascii**? **1pt**

**asciz is the same as ascii nut each string is followed by a zero byte**

* 1. Explain what the **bic** instruction is used for? **1pt**

**In this code it performs an AND bitwise operation on the value of x1 (which is the character entered) with the value of 32 and stores the result in x0, which uppercases the character entered.**

* 1. Why is the purpose of **.align 4** directive? **1pt**

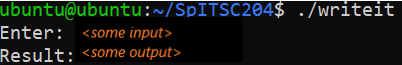
**Aligns the data on a 4 byte boundary**

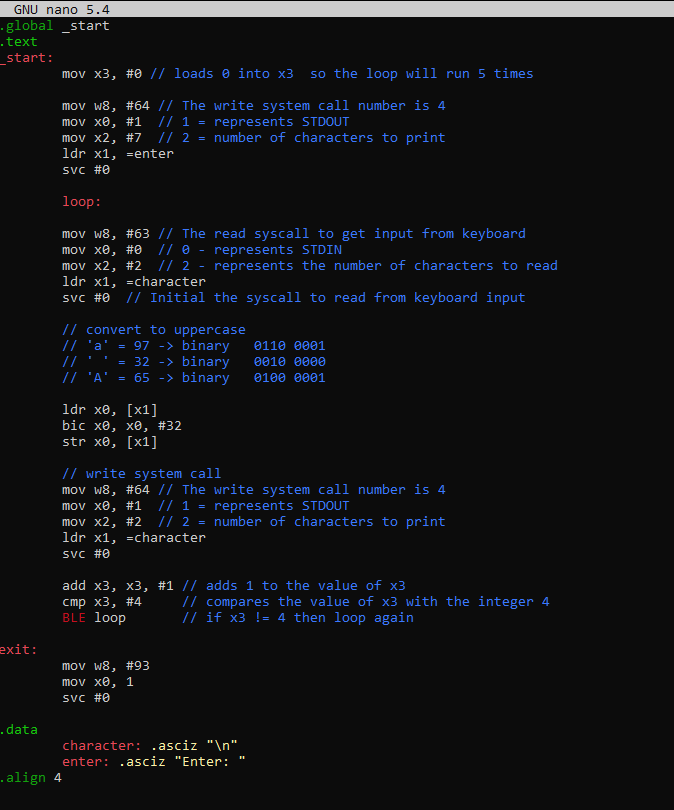
* 1. Explain the instruction **ldr x1,=character 1pt**

**Loads the register x1 with the value stored in character**

1. Modify the code in a way that converts 5 characters in a loop.

Write a loop **simple**, loop max 3-5 lines,that runs 5 times and reads, converts your input. **3pts**

1. Modify the code so that the output looks like the image below:
2. Provide the following screen captures
   1. Created code

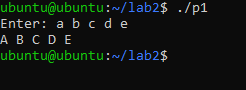


* 1. Provide the answers under respective question (color)

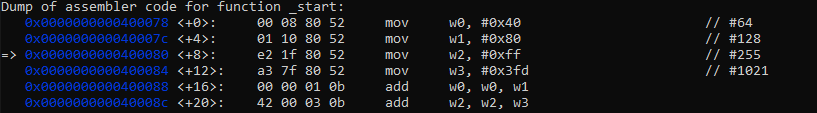
**Answers are under the respective questions above**

* 1. Provide modified code and the results

**Modified code is posted above**

****

# Problem 2 Analyze immediate values \_\_\_15

1. Write ARM assembly code that perform the following **7pts**
   1. Move the value 64 to W0
   2. Move the value 128 to W1
   3. Move the value 255 to W2
   4. Move the value 1021 to W3
   5. Add W0 = W0 + W1
   6. Add W2 = W2 + W3
   7. Assemble the code
   8. Use gdb to analyze the respective commands to verify the instructions and values in each register.
2. Convert the any 3 instructions opcode into binary and identify the respective bits for each instruction and respective operands. Use as reference the last slide of Module 7 “ ARM Opcode e.g. For ARMv7” **8pts**
3. Provide the screen capture of objdump results
4. Provide the conversion and the identified bits for each instruction and the respective operands after the objdump capture

**So I used the slides to answer this question but you mentioned that the instructions are in 32 bit and mine are in 64 bit so this may be incorrect**

Condition I Opcode S Rn Rd Op2

I if set to 1 the second operand is an immediate value

S if set to 1 the conditions status can be updated

**52 80 08 00**

**0101 00 1 0100 0 0000 0000 100000000000**

**52 80 10 01**

**0101 00 1 0100 0 0000 0001 000000000001**

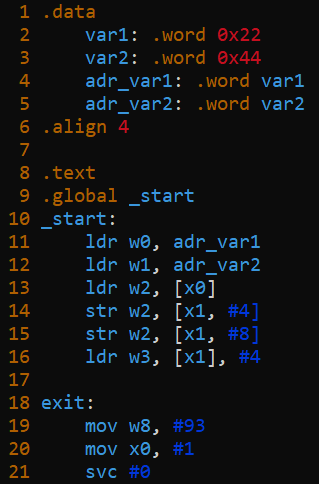
**52 80 1f e2**

**0101 00 1 0100 0 0000 0001 111111100010**

# Problem 3 Stack Analysis \_\_\_15

**Memory Addressing Modes**

1. Create and analyze the following program:



1. Identify the differentiate memory addressing modes being used in the code
2. Comment the identified modes on the lines in which they appear
3. In your comments identify the **base address register**?
4. Use the gdb debugger and create a break at \_start. Use si, info registers and x commands.
   1. Identify the addresses of var1 and var2

**var1: 0x4100e0**

**var2: 0x4100e4**

* 1. Record x0, x1 and x2 values before x2 is stored in memory

**x0: 0x4100e0**

**x1: 0x4100e4**

**x2: 0x22**

* 1. Use disass to identify the address of <adr\_var1> and <adr\_var2>

**adr\_var1: 0x4100e8**

**adr\_var2: 0x4100ec**

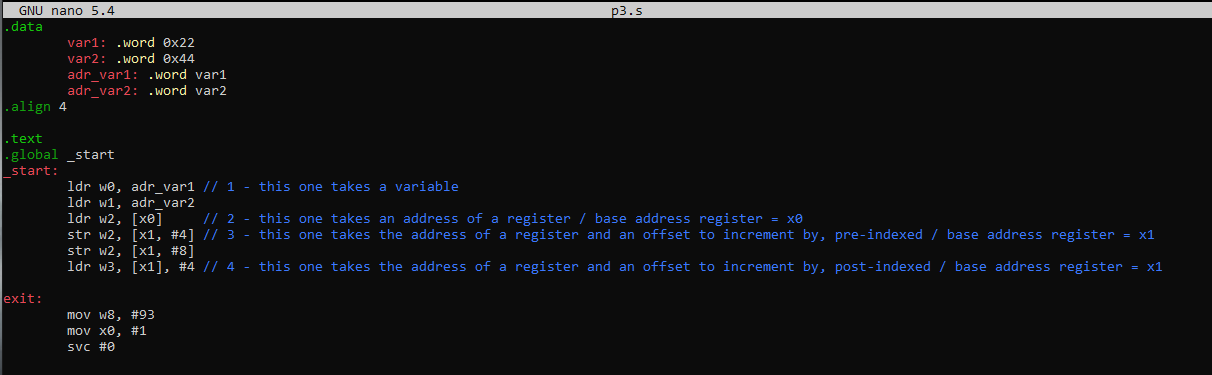
* 1. Use x to verify the content of these addresses and compare it with the values in x0 and x1

**They are the same values (x0 = adr\_var1, x1 = adr\_var2)**

* 1. Use si and info registers command to analyze the registers results after each str and ldr instruction. Observe the base address register behavior and record its value after each str and ldr instruction. Explain the results

**Before being loaded the values were at 0, after being loaded the values stayed the same over the course of the program**

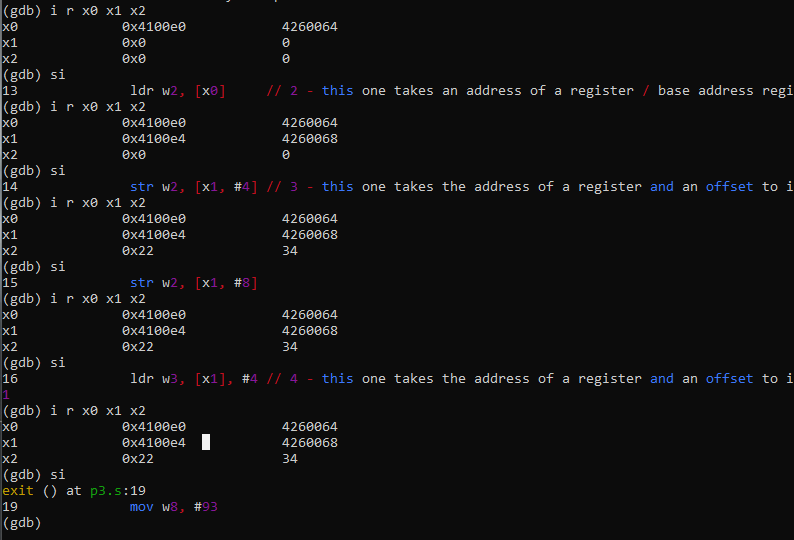
1. Attach the following screen captures:
   1. Commented program



* 1. Underline or highlight the base address register

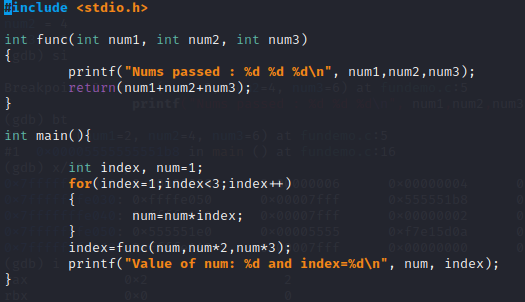
**Base address register is identified in the comments**

* 1. Debugger analysis that demo the changes of base address register



**Stack Analysis**

1. Create the following C code or if you already had the code copy it into raspberry pi

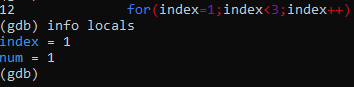


1. Compile it and use the debugger to analyze the stack
2. Use the respective debugger commands to run the program and provide the screen captures that demo the following
   1. Prologue and epilogue of the function called func





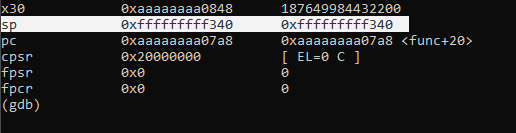
* 1. Info locals



* 1. Function func parameters



* 1. Function func frame. Value of sp and link register. Is there an fp (frame pointer)



* 1. Return address.



1. What is the addressing mode use by str and ldr instructions

**They appear to use pre-index addressing**

1. Which one is the base address register?

**It appears to be sp (the stack pointer)**