Lab 02: Intro to ARM Assembly Language

My name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
My lab partner(s) for Parts 3-6: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Objectives

* Review and practice ARM data movement instructions.
* Review and practice ARM ALU instructions.
* Analyze and write simple ARM assembly code.
* Peer review

**Note:** Parts 1 and 2 should be completed individually before Wednesday’s class to take advantage of Wednesday’s peer review and teamwork. If those parts are not complete before class, you should still come to class.

For Parts 1 & 2, fill out the “Result” column *without using an emulator*. The first two rows are completed for you as an example. When a register value changes, write all 32 bits of the new register value in hex. When memory changes, list each address that changed. Some instructions change the flags, in which case you should state “flags updated” as one of the results. Treat each row as a separate problem. **Don’t fill out the “Second Try” columns until you get to Part 5.**

# Part 1: Data Movement Instructions [5 points]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Instruction(s)** | **Previous Values** | **Result (what changed?)** | **Second Try** |
|  | MOV R1, R0 | R0 = 0x01234567 R1 = 0xBEEFFACE | R1 = 0x01234567 | -------------------- |
|  | STRH R0, [R1] | R0 = 0x0000ABCD R1 = 0x20000000 | Address 0x20000001 = 0xAB Address 0x20000000 = 0xCD | -------------------- |
| 1A | MOV R1, #0xFA05 | R1 = 0x01234567 |  |  |
| 1B | MOVS R1, #10 | R1 = 0x01234567 |  |  |
| 1C | MOVT R1, #0xFA | R1 = 0x01234567 |  |  |
| 1D | LDRB R2, [R1] | R1 = 0x20000000 R2 = 0xBEEFFACE Address 0x20000003 = 0xAA  Address 0x20000002 = 0xBB  Address 0x20000001 = 0xCC  Address 0x20000000 = 0xDD |  |  |
| 1E | STR R2, [R1, #4] | R1 = 0x20000000 R2 = 0xBEEFFACE |  |  |

# Part 2: ALU Instructions [5 points]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Instruction(s)** | **Previous Values** | **Result (what changed?)** | **Second Try** |
| 2A | ADD R2, R1, R3 | R1 = 0x00000005 R2 = 0x00000002 R3 = 0x00000007 |  |  |
| 2B | ADDS R4, R4, #100 | R4 = 0x00000100 |  |  |
| 2C | AND R9, R2, #0xFF00 | R2 = 0xFACEBEAD R9 = 0xBEEFFACE |  |  |
| 2D | UDIV R2, R0, R1 | R0 = 0x00000005 R1 = 0x00000002 R2 = 0x00000007 |  |  |
| 2E | ORR R2, R0, #00000010b | R0 = 0x00001111 R2 = 0xBEEFFACE |  |  |

**Stop here. You will complete the rest with a lab partner during class on Wednesday.**

# Part 3: Practice Writing Assembly Code - Swapping values [5 points]

Write an assembly language program that swaps the value in Register 1 with the value in Register 2. (The values should change places). *Make sure to comment your code.*

|  |  |
| --- | --- |
| **Code:** | **Second Try** |
|  |  |

# Part 4: Practice Writing Assembly Code – Memory Copy [5 points]

Write an assembly language program that reads the 32-bit value in memory starting at address 0x20000000 and then copies it to memory starting at address 0x20000004. **Use an offset.** *Make sure to comment your code.*

|  |  |
| --- | --- |
| **Code:** | **Second Try** |
|  |  |

# Part 5: Peer Review and Corrections (to be done in class on Wednesday)

* For Parts 1 & 2:
  + Peer-review each other’s answers for Parts 1 and 2. **Do not change your original answers.**
  + In the “second try” columns, add corrections to anything that needs to be fixed/improved. Sufficiently corrected answers there can help you earn back any points missed on the first try.
* For Parts 3 & 4:
  + Use an emulator to check your code in Parts 3 & 4. **Do not change your original code.**
  + In the “second try” columns, add corrections to anything that needs to be fixed/improved.
* You may now also use an emulator to double check your answers to Parts 1 & 2.

# Part 6: Lab Code

6(A) Create an ARM Assembly Program [10 points]

With your lab partner(s), write an assembly program that is at least 15 lines long. **Do not use ChatGPT to help you with this assignment.** You may choose what your program does, but you should try to make it do something at least somewhat useful, even if simple. Do not use an emulator to test it yet. Comment your code. You may look ahead to the readings on branching instructions if you would like to include those.

Here are some simple ideas, but try to think of your own: a program that puts the numbers 1-10 in an array in memory, a program that reads in the radius of a sphere and calculates the volume, a program that converts the string “HELLO WORLD” to “hello world” using ASCII math, etc.

|  |  |
| --- | --- |
| **Assembly Code:** | **Second Try** |
|  |  |

(lab continues onto the next page)

### 6(B) Create a C Code Version [5 points]

With your lab partner(s), determine an equivalent ‘C’ program that performs approximately the same operations as your assembly language program. It doesn’t have to be line-by-line equivalent, just have it do a similar task. Comment your code.

|  |
| --- |
| **C Code:** |
|  |

### 6(C) Test with an Emulator

Now, using one of the ARM emulators in Canvas, test your assembly code in part 6(A). If after testing your code you decide to make changes, don’t change the existing code, instead place the updated code in the “Second Try” column. Corrections to your code there can help you earn back points missed on your first try.

Note that the emulator doesn’t support certain operations, like UDIV. If you use an unsupported operation, just test the other parts of your program. Note also that the emulator limits immediates in MOV to 12 bits; so, to have an immediate with 16 bits, use MOVW. (On the actual microcontroller, you won’t need to worry about these issues.)