# Lab 02: Practicing ARM Assembly Language

My name: _____Brodric Young_____

My lab partner(s) name(s) _____Karl Richards_____

## Objectives

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

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 (those with an 'S' added) also change the Status flags (used as comparisons for conditional execution), in which case you should also state "flags updated" as one of the results. Each row is 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`<br>`R1 = 0xBEEFFACE` | `R1 = 0x01234567` | `----------` |
| | `STRH R0, [R1]` | `R0 = 0x0000ABCD`<br>`R1 = 0x20000000` | `Address 0x20000001 = 0xAB`<br>`Address 0x20000000 = 0xCD` | `----------` |
| 1A | `MOV R1, #0xFA05` | `R1 = 0x01234567` | `R1 = 0x0000FA05` | |
| 1B | `MOVS R1, #10` | `R1 = 0x01234567` | `R1 = 0x0000000A`<br>`Flags updated` | |
| 1C | `MOVT R1, #0xFA` | `R1 = 0x01234567` | `R1 = 0x00FA4567` | |
| 1D | `LDRB R2, [R1]` | `R1 = 0x20000000`<br>`R2 = 0xBEEFFACE`<br>`Address 0x20000003 = 0xAA`<br>`Address 0x20000002 = 0xBB`<br>`Address 0x20000001 = 0xCC`<br>`Address 0x20000000 = 0xDD` | `R2 = 0x000000DD` | |
| 1E | `STR R2, [R1, #4]` | `R1 = 0x20000000`<br>`R2 = 0xBEEFFACE` | `Address 0x20000007 = 0xBE`<br>`Address 0x20000006 = 0xEF`<br>`Address 0x20000005 = 0xFA`<br>`Address 0x20000004 = 0xCE` | |

## Part 2: ALU Instructions [5 points]

| | Instruction(s) | Previous Values | Result (what changed?) | Second Try |
|---|---|---|---|---|
| 2A | `ADD R2, R1, R3` | `R1 = 0x00000005`<br>`R2 = 0x00000002`<br>`R3 = 0x00000007` | `R2 = 0x0000000C` | |
| 2B | `ADDS R4, R4, #100` | `R4 = 0x00000100` | `0x00000164`<br>`Flags updated` | |
| 2C | `AND R9, R2, #0xFF00` | `R2 = 0xFACEBEAD`<br>`R9 = 0xBEEFFACE` | `R9 = 0x0000BE00` | |
| 2D | `UDIV R2, R0, R1` | `R0 = 0x00000005`<br>`R1 = 0x00000002`<br>`R2 = 0x00000007` | `R2 = 0x00000002` | |
| 2E | `ORR R2, R0, #00000010b` | `R0 = 0x00001111`<br>`R2 = 0xBEEFFACE` | `R2 = 00001113` | |

For Parts 3 & 4, complete the "Code" columns *without using an emulator*. Some starting code is provided for you. Add more lines of code to complete the task. **Don't fill out the "Second Try" columns until you get to Part 5.**

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

Write an assembly language program that swaps the value in Register 1 (#7) with the value in Register 2 (#4). The values should change places. The code should work for any values in R1 or R2. *Make sure to comment your code.*

| Code: | Second Try |
|---|---|
| `MOV R1, #7  ;R1 holds the number 7`<br>`MOV R2, #4  ;R2 holds the number 4`<br>`MOV R3, R1  ;Move R1 into R3`<br>`MOV R1, R2  ;Move R2 into R1`<br>`MOV R3, R2  ;Move R3 into R2` | |

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

Write assembly language that stores the 32-bit value from R1 (0xBEEFFACE) to memory at address 0x20000000 (you'll need to get 0x20000000 into a register first), then reads a *half word* (16-bits) from that same memory location into R2, and then stores a *byte* (8-bits) from R2 into memory at address 0x20000004. Don't use the number 0x20000004. Instead use an **offset** from the register that holds 0x20000000**.** *Make sure to comment your code.*

| Code: | Second Try |
|---|---|
| `MOV  R1, #0xFACE`<br>`MOVT R1, #0xBEEF  ; R1 = 0xBEEFFACE`<br>`MOV R3, #0x0000`<br>`MOVT R3, #0x2000  ; R3 = 0x20000000`<br>`STR R1, [R3]  ; Store R1 at [R3]`<br>`LDRH R2, [R3]  ; R2 = 0x0000FACE`<br>`STRB R2, [R3, #4] ; store R2 at [R3, #4]` | |

## Part 5: Peer Review and Corrections

A. Peer Review with another team
- Find another team to peer review your answers to Parts 1-4. Peer review their answers, too.
- Each team will let the other team know what corrections they believe need to be made.
- **Important: Do not change your original answers.** All corrected answers should go in "Second Try."
- 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.

B. After the peer review, use an emulator to double check your answers.
- Again, do not change your first-try answers. Any corrections should update the "Second Try" column.

## Part 6: Lab Code

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

With your original lab partner(s), write an assembly program that is at least 15 lines long. **Do not use A.I. such as ChatGPT to help you with this assignment.** **Do not use an emulator until part 6(C).** You may choose what your program does, but you should try to do something useful, even if it is simple. Comment your code.

Here are some simple ideas, but try to think of your own: a program that puts the Fibonacci sequence (1, 1, 2, 3, 5, 8…) into an array in memory; a program that creates a list of perfect squares (1, 4, 9, 16, 25…) and places them into an array in memory; a program that put the numbers from 1 to `n` in an array, and then sums those values; a program that receives the side lengths of rectangles and calculates the perimeter and area; a program that converts the string "HELLO WORLD" to lowercase "hello world" using ASCII math; etc.

| Assembly Code: | Second Try |
|---|---|
| `MOV R0, #0`<br>`MOVT R0, #0x2000 ; R0 = 0x20000000`<br>`MOV R1, #0x0401`<br>`MOVT R1, #0x1009 ; R1 = 0x10090401`<br>`MOV R2, #0x2419`<br>`MOVT R2, #0x4031 ; R2 = 0x40312419`<br>`STR R1, [R0]`<br>`STR R2, [R0, #4]`<br><br>`LDRB R3, [R0] ; R3 = 1 (0x1)`<br>`LDRB R4, [R0, #1] ; R4 = 4 (0x4)`<br>`ADD R11, R3, R4 ; R11 = 5 (0x5)`<br><br>`LDRB R3, [R0, #2] ; R3 = 9 (0x9)`<br>`LDRB R4, [R0, #3] ; R4 = 16 (0x10)`<br>`ADD R10, R3, R4 ; R10 = 25 (0x19)`<br><br>`ADD R11, R11, R10 ; R11 = 30 (0x1E)`<br><br>`LDRB R3, [R0, #4] ; R3 = 25 (0x19)`<br>`LDRB R4, [R0, #5] ; R4 = 36 (0x24)`<br>`ADD R10, R3, R4 ; R10 = 61 (0x3D)`<br><br>`ADD R11, R11, R10 ; R11 = 91 (0x5B)`<br><br>`LDRB R3, [R0, #6] ; R3 = 49 (0x31)`<br>`LDRB R4, [R0, #7] ; R4 = 64 (0x40)`<br>`ADD R10, R3, R4 ; R10 = 113 (0x71)` | `MOV R0, #0`<br>`MOVT R0, #0x2000 ; R0 = 0x20000000`<br>`MOV R1, #0x0401`<br>`MOVT R1, #0x1009 ; R1 = 0x10090401`<br>`MOV R2, #0x2419`<br>`MOVT R2, #0x4031 ; R2 = 0x40312419`<br>`STR R1, [R0]`<br>`STR R2, [R0, #4]`<br><br>`LDRB R3, [R0] ; R3 = 1 (0x1)`<br>`LDRB R4, [R0, #1] ; R4 = 4 (0x4)`<br>`ADD R11, R3, R4 ; R11 = 5 (0x5)`<br><br>`LDRB R3, [R0, #2] ; R3 = 9 (0x9)`<br>`LDRB R4, [R0, #3] ; R4 = 16 (0x10)`<br>`ADD R10, R3, R4 ; R10 = 25 (0x19)`<br><br>`ADD R11, R11, R10 ; R11 = 30 (0x1E)`<br><br>`LDRB R3, [R0, #4] ; R3 = 25 (0x19)`<br>`LDRB R4, [R0, #5] ; R4 = 36 (0x24)`<br>`ADD R10, R3, R4 ; R10 = 61 (0x3D)`<br><br>`ADD R11, R11, R10 ; R11 = 91 (0x5B)`<br><br>`LDRB R3, [R0, #6] ; R3 = 49 (0x31)`<br>`LDRB R4, [R0, #7] ; R4 = 64 (0x40)`<br>`ADD R10, R3, R4 ; R10 = 113 (0x71)` |

```
ADD R11, R11, R10 ; R11 = 204 (0xCC)
UDIV R11, R11, #8 ; R11 = 25 (0x19)
```

```
ADD R11, R11, R10 ; R11 = 204 (0xCC)
MOV R10, #8
UDIV R11, R11, R10 ; R11 = 25 (0x19)
STR R11, [R0, #8]
```

(lab continues onto the next page)

## 6(B) Create a C Code Version [3 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:**

```
#include <stdio.h>

int main(void) {
    // store array of 8 different perfect squares
    int squares[8] = {1, 4, 9, 16, 25, 36, 49, 64};
    int sum = 0;

    // sums up all perfect squares in the array
    for (int i = 0; i < 8; i++) {
        sum += squares[i];
    }

    // calculate average
    int avg = sum / 8;

    // display result
    printf("Average: %d", avg);
}
```

## 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 any points missed on your first try.