CPU Project

M-ARM and MARMalade

Operation, Specifications, and Syntax

A written guide detailing the operation of the assembler and instruction syntax for M-ARM, as well as MARMalade specifics
(Contains an example program!)

Matthew McCaughan
CPU Project
April 5th, 2024

Contents

1. Introduction

Thanks for taking the time to read the MARMalade and M-ARM assembly Handbook! I hope you find this handbook useful for operating the assembler effectively, identifying valid instructions, and learning about such a great new language and CPU!

2. CPU

2.1 Name and Design

My CPU's name is MARMalade, a play on words with M-ARM and marmalade. MARMalade contains 4 general purpose registers, M0, M1, M2, and M3, and they are referred to as such in the CPU and the assembly Language. MARMalade has three distinct functions: Loading, Adding, and Subtracting. Each instruction is represented by 8 bits (1 byte) of machine code in the CPU instruction memory.

3. Assembler

3.1 Walkthrough

The assembler for M-ARM is in the form of a python file titled **MarmAssembler.py**. Upon running the file, the program should run automatically and you will be greeted with text and directions, of which a snippet is provided below:

```
--------------------
INPUT FORMATTING:
Please input instuctions one at a time per prompt, when
   done, please input 'n' when prompted
!! Accepts up to 16 Instructions !!
!! Data Memory must be hardcoded !!
*****************************************************
Enter your instructions below:
Enter Instruction 1 :
```

Below "Enter Instruction 1 :" is where you will be entering your instructions! Instructions may only be entered one at a time, so enter the full instruction (with correct syntax please!). Correct instruction syntax is provided further in this document.

You will be greeted with another message (example provided):

```
Enter Instruction 1 :
LDR M0 2 00
Add another instruction? (y/n):
```

When facing this prompt, please input ANYTHING BUT an "n" if you want to add another instruction, and another input space will be added to allow you to type your next instruction! If you've added all of your instructions please type "n" and you will proceed. You will then be asked to confirm the contents of your input(s). Hopefully, you inputted the instructions right. If you've done everything right, the assembler should generate a .txt file for you. This will be your image file in which you will input into instruction memory in the .circ file. You may modify the name of the text file in the assembler on line 161, by default the assembler will create a file titled *Instructions.txt*.

3.2 Things to Remember

- M-ARM supports a maximum of 16 instructions per program.
- Please follow the syntax carefully for every instruction!
- The assembler can always just be re-run in the case of difficulties.
- The actual values in data memory must be hard-coded!

4. Instruction Set

4.1 Preface

M-ARM allows access to 4 registers within the CPU, each of these registers is an M register. Therefore the user has access to registers:

M0, M1, M2, and M3

These registers can be parameters in an instruction as well as the destination of an instruction. Every instruction is 8 bits long when translated to machine code, the highest 2 bits contain operation codes, then two bits for the destination register, then the last 4 bits for the parameter registers (2 bits each). M-ARM comes included with three exciting instructions to play with using these registers, and these are detailed below.

4.2 Loading

Loading takes data from data memory and moves it into the register, so that the register contains the data you accessed. This instruction is in the form of:

LDR Md Imm 00

In Which:

- **LDR** is the identifier for loading
- **Md** is the destination register for the data being loaded
- **Imm** is the offset of the stored data memory. Imm can be values from 0 to 3 and determines which position the instruction takes data from in data memory. For example, inputting '0' for Imm will access the actual data in the first address in data memory. There are 4 data memory addresses.
- **00** is the 4th parameter and MUST BE INCLUDED ON ALL LOADING INSTRUCTIONS just add it so the program assembles correctly.

4.3 Addition

Addition takes the data stored in two parameter registers, sums them, and stores that sum into the destination register. This instruction is in the form of:

ADD Md Ma Mb

In Which:

- **ADD** is the identifier for Addition
- **Md** is the destination register for the data being loaded
- **Ma** is the first parameter register
- **Mb** is the second parameter register

In other words, the instruction is **equivalent to Md = Ma + Mb**

4.4 Subtraction

Subtraction takes the data stored in two parameter registers, takes their difference, and stores that difference into the destination register. This instruction is in the form of:

SUB Md Ma Mb

In Which:

- **SUB** is the identifier for Subtraction
- **Md** is the destination register for the data being loaded
- **Ma** is the first parameter register
- **Mb** is the second parameter register

In other words, the instruction is **equivalent to Md = Ma - Mb**

4.5 Example Code

Here is a simple example program:

```
LDR M0 0 00
LDR M1 1 00
SUB M2 M1 M0
```

These three lines will store the difference of the values loaded in M1 and M0 into M2. Remember Data memory at addresses 0 and 1 must be hardcoded!

5. Conclusion

Hopefully all information provided in this document will be enough for you to generate programs and execute it in the CPU. Thanks for taking the time to read and review my project!