# Microprocessor Systems
# ARM Emulator Project
# Part: 1

## Introduction

The goal of the project is to emulate ARM Cortex M0 processor, by interpreting a subset of ARM Thumb instructions, using the programming language C, and graphics library SDL. An LCD screen is assumed to be attached as a peripheral device. Program takes its input as ARM assembly code, compiles it into ARM machine code using gcc-arm-none-eabi assembler, then interpretes the output machine code.

## Team members

Deniz Bashgoren: 040180902 (bashgoren18@itu.edu.tr)
C. Cem Belentepe: 040180255 (belentepe18@itu.edu.tr)

## Implementation

The emulator is non-strict: in cases where Cortex M0 would give HardFault, our emulator simply ignores the error. Since Cortex M0 operates only in Thumb mode, we ignore the T flag, and always keep it at 0. Branches always assume that the processor is in Thumb mode, so they keep the last bit zero. The N, Z, C, V flags are implemented according to the standard. The supported instructions are:

- Arithmetic: ADD, SUB, ADR, ADC, SBC, RSB, MUL, NEG
- Logical: AND, EOR, ORR, BIC, MVN, TST, CMP
- Bit shifting: LSL, LSR, ASR, ROR
- Memory: MOV, LDR, STR, PUSH, POP
- Branching: B, B{cond}, BL, BX, BLX

This implementation defines a new instruction, for the purpose of debugging. It takes advantage of an illegal bit sequence, and interprets it as a debug instruction. Adding `.hword 0xde00` in the code will trigger the emulator's debugger dialog. From there, one can see the state of the registers, and optionally, any part of the memory, by specifying which parts of the memory to view. Running the application with -debug flag after the path to the asm code, will automatically trigger the debugger on every instruction.
Another custom instruction is `.hword 0xde01`. Adding this sequence to the asm code will trigger a time counter. On every 60th execution, the current FPS will be printed on console.

## Discussion

The chosen processor, ARM Cortex M0, was one of the easiest ones to emulate, due to its limited number of instructions, fixed-size 32-bit registers, and relative ease to find documentation.

One of the difficult parts was to express the implementation logic correctly in C. Undefined and implementation defined behavior in C, especially casting rules, common arithmetic operators' underlying bit-level behavior was hard to reason about, and unpredictable at times.

Another difficult part was to find precise definitions of instructions. The given reference material was not sufficient for some parts. For example, the behavior of NEG, TST, BLX (label) was not documented properly, (as some were not even true opcodes - they were "pseudoinstructions" ). We had to use online references, and sometimes developer forums to get our answers.

A historically difficult part was to always consider the endianness of the data. We decided to systematically swap the bytes where needed, by limiting all the interaction with memory by two functions: load_from_memory, and store_to_memory. These functions would take for granted the endianness issues.

## Benchmark

A simple benchmark is created to see the performance difference with and without the optimizations of the GCC compiler. The recorded times are execution times (in seconds) of 100,000,000 instructions(the number is chosen arbitrarily) of the provided `color.s` assembly test file on a 2.4 GHz machine. Measurement of the benchmark is done by using the clock( ) function of the std header <time.h>.

|  | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| No optimization | 3.122 | 3.437 | 3.199 | 3.151 | 3.109 | 3.178 | 3.108 | 3.095 | 3.192 | 3.096 | 3.169 |
| Using -O 3 | 1.766 | 1.756 | 1.800 | 1.760 | 1.772 | 1.803 | 1.752 | 1.808 | 1.840 | 1.799 | 1.786 |

(the values are in seconds)

## <u>Test Programs</u>

Two simple test programs are provided along with the code, to demonstrate the validity and performance of the emulator. First is `color.s`, which displays tones of blue, green and red colors in order. The second file is `game_of_life.s`, an implementation of Conway's Game of Life on a 64x48 grid.

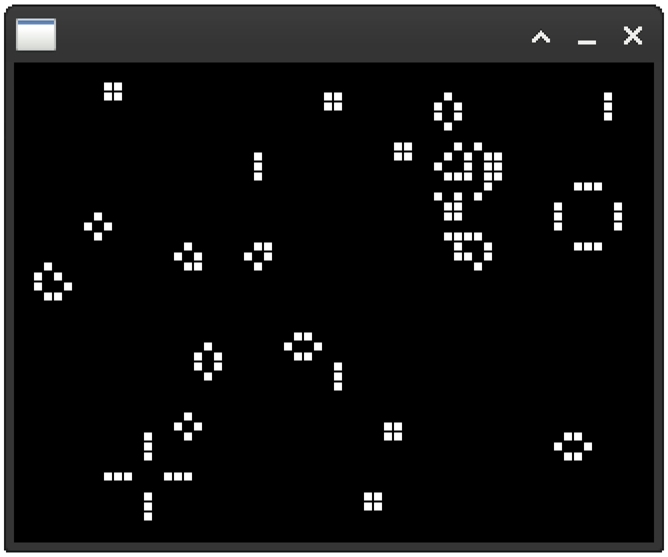

Figure 1: Conway's Game of Life

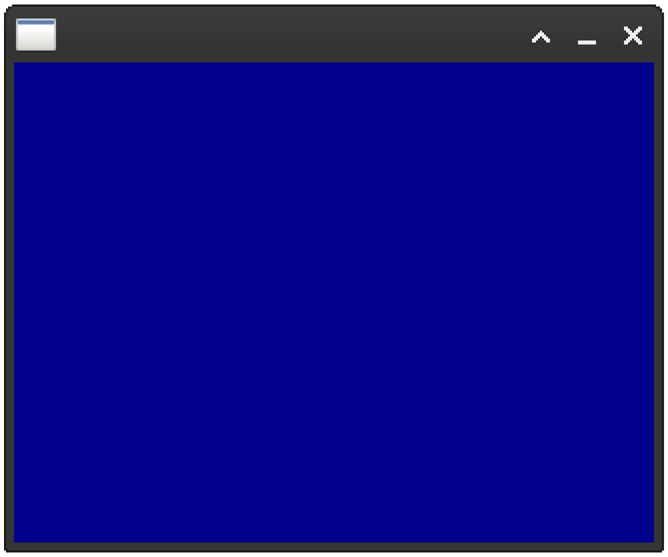

Figure 2: Color program