DIV=14, BCOR=0mm, headinclude=true, footinclude=false

Computer Design Laboratory Manual

Department of Electrical & Computer Engineering

September 30, 2025

Contents

Learning Objectives

- Understand the fundamental concepts of ARM architecture and assembly language programming.
- Execute and debug assembly code effectively using the **Keil uVision5** development environment.
- Implement basic assembly programs for data manipulation and control flow operations.

Experiment Overview

This experiment introduces the ARM Cortex-M4 architecture and assembly language programming. Students will learn about the architecture's key features, including its registers, memory model, and instruction set. The experiment will also cover the basics of writing, assembling, and debugging assembly code using the **Keil uVision5** IDE. By the end of this experiment, students will have a foundational understanding of ARM assembly language and be able to write simple programs to manipulate data and control program flow.

1 Theoretical Background

1.1 Cortex-M4 Architecture

1.1.1 Registers Overview

The Cortex-M4 architecture includes a set of general-purpose registers (R0-R12), a stack pointer (SP), a link register (LR), a program counter (PC), and a program status register (xPSR), all of which are 32-bit registers. The general-purpose registers are used for data manipulation and temporary storage during program execution. The SP is used to manage the call stack, while the LR holds the return address for function calls. The PC points to the next instruction to be executed, and the xPSR contains flags and status information about the processor state.

Program Status Register (xPSR)

The xPSR holds the current state of the processor, including condition flags (Negative, Zero, Carry, Overflow), interrupt status, and execution state. These flags are updated based on the results of arithmetic and logical operations, allowing for conditional branching and decision-making in programs.

1.1.2 Memory Model

The Cortex-M4 uses a flat memory model, where all memory locations are accessible through a single address space. This model simplifies programming and allows for efficient access to data and instructions. The memory is divided into several regions, including code memory (for storing instructions), data memory (for storing variables), and peripheral memory (for interfacing with hardware components). The architecture supports both little-endian and big-endian data formats, with little-endian being the default.

Table 1: Cortex-M4 Memory Regions (ARMv7-M)

Region	Address Range	Description
Code	0x0000_0000 – 0x1FFF_FFFF	Flash / code memory
SRAM	0x2000_0000 – 0x3FFF_FFFF	On-chip static RAM
Peripheral	0x4000_0000 – 0x5FFF_FFFF	Memory-mapped peripheral registers
External RAM	0x6000_0000 – 0x9FFF_FFFF	External RAM (if implemented)
External Device	0xA000_0000 – 0xDFFF_FFFF	External devices / memory (if impleme
PPB (Private Peripheral Bus)	0xE000_0000 – 0xE00F_FFFF	Cortex-M4 internal peripherals (NVIC sTick, MPU, SCB)
System	0xE010_0000 – 0xFFFF_FFFF	System region (reserved/system-level)

1.2 Assembly Language Basics

2 Procedure