Kennesaw State University (KSU)
College of Computing and Software Engineering (CCSE)
CS-3503 Computer Organization & Architecture

Lab-6: Calling ARM assembly subroutine from C/C++ program and ARM processor performance analysis

Name: _____ **Date:** _____

Learning Objectives:

- Create a program in C/C++ and assembly that calls an assembly subroutine from a C/C++ program.
- Analyze the performance of the ARM processor based on experimental results.

Introduction

This laboratory exercise introduces you to the coding framework used to integrate assembly subroutines into C/C++ programs.

Why Integrate Assembly Subroutines into C/C++ Programs?

Reasons for integrating assembly subroutines into C/C++ programs include:

1) Optimization of a performance critical function.
   - Manually programming in assembly may result in improved performance compared to compiler-generated code.
2) Availability of processor-specific instructions that a compiler may not utilize.
3) Direct access to hardware.

Calling Assembly Subroutines from a C/C++ Program

To call an assembly subroutine stored in a different source file from a C/C++ program the following requirements must be satisfied:

1) Assembly code must use the "EXPORT" or "GLOBAL" directive to make the assembly subroutine accessible by the C/C++ program.
2) C/C++ program must declare the assembly subroutine as a function using the "extern" keyword.

Also, the input arguments and return value of the C/C++ function corresponding to the assembly subroutine map to specific register locations according to the ARM embedded application binary interface (EABI). As shown in Fig. 1, the first four, 32-bit input arguments map to registers R0 – R3, respectively, and a 32-bit return value maps to R0.

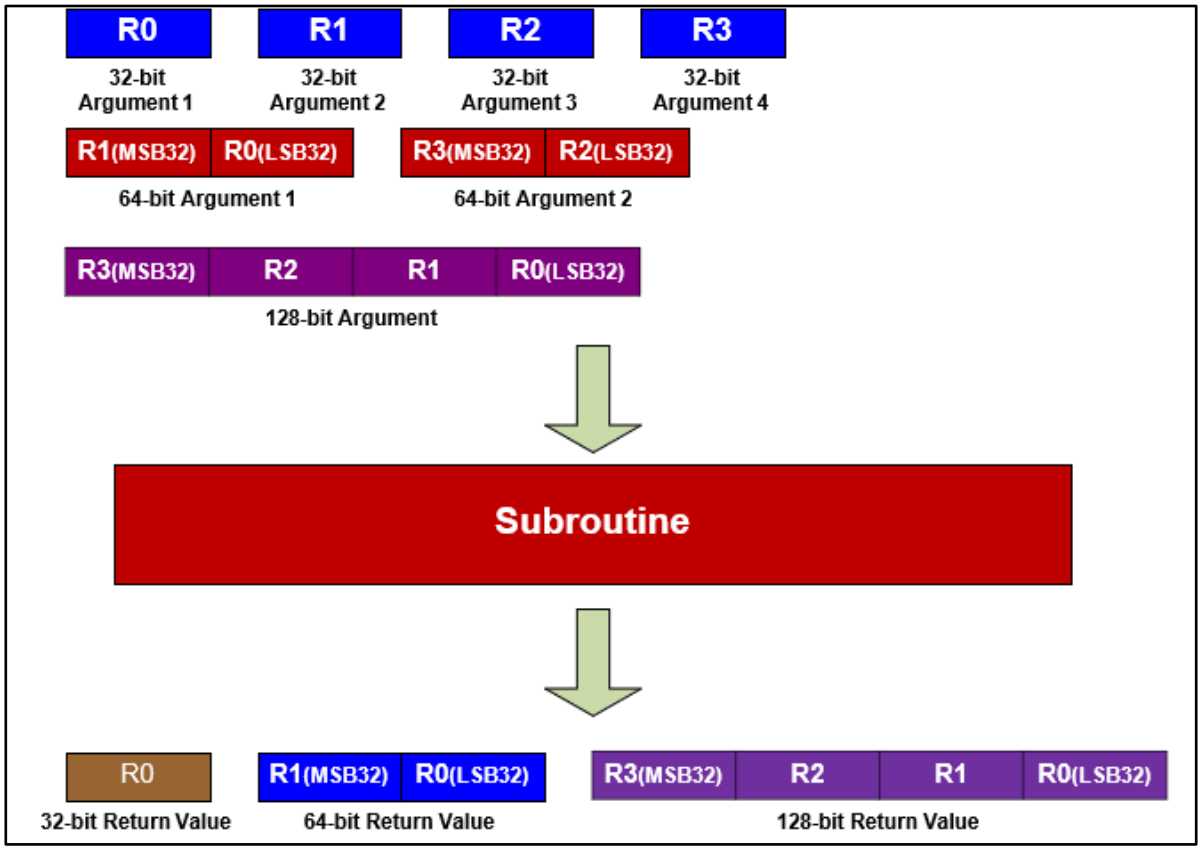


Figure 1 – Register to Argument and Register to Return Value Mappings for ARM EABI

Prelab

For your prelab activity, you will create a project that consists of 2 files: main.cpp and add_asm.s. The code in main.cpp will add two integers together and print the sum to the terminal screen. The two integers are added by calling an assembly subroutine written in add_asm.s. The code for each file is listed below

main.cpp

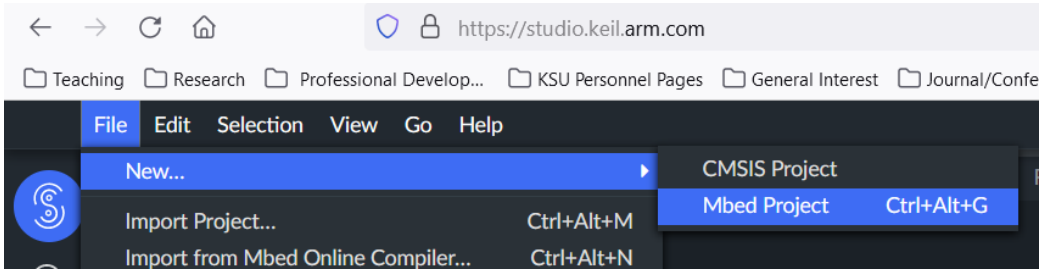
```
#include "mbed.h"

extern "C" int add_asm(int a, int b);

int main() {
    while (true) {
        printf("add_asm computes %d + %d = %d\n", 2, 3, add_asm(2,3));
         wait(0.5);
    }
}
```

add_asm.s

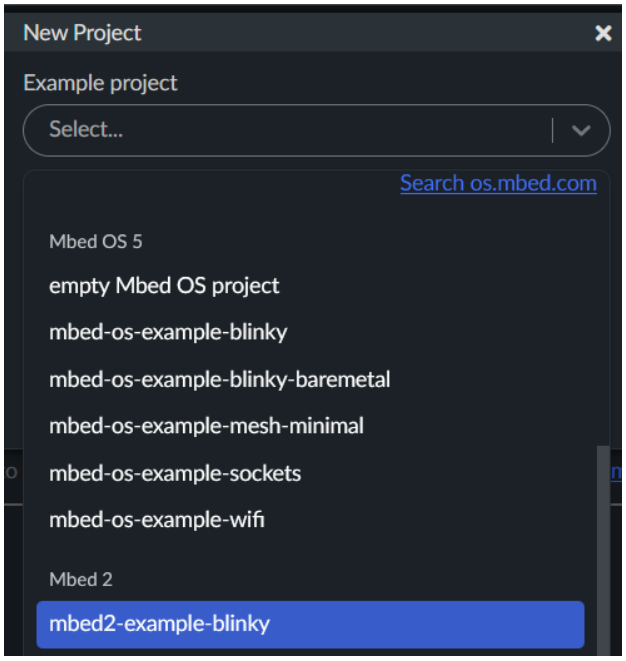
```
            AREA  |.text|, CODE, READONLY

add_asm     PROC
            EXPORT add_asm
            ADD   R0, R1
            BX    LR
            ENDP
            ALIGN
            END
```

You will use Arm Keil Studio to create this project. To do this successfully, follow the instructions below.

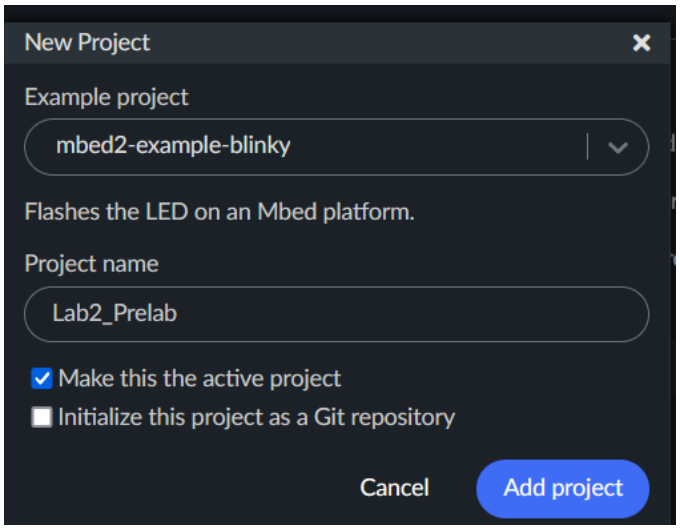
1. Create a new Mbed project in Arm Keil Studio called Lab_6_Prelab by doing the following:
   i. **Select File > New > Mbed Project option** in the upper left corner of the IDE.



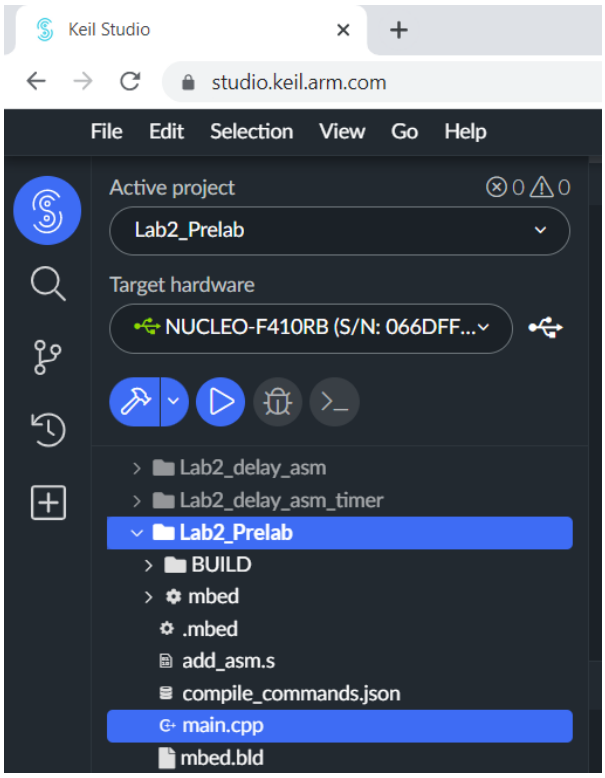
   ii. In the **New Project** dialog box, **under Example project, select Mbed 2 → mbed2-example-blinky**.



iii. **Enter Lab_6_Prelab** under **Project Name** and **click Add Project**.

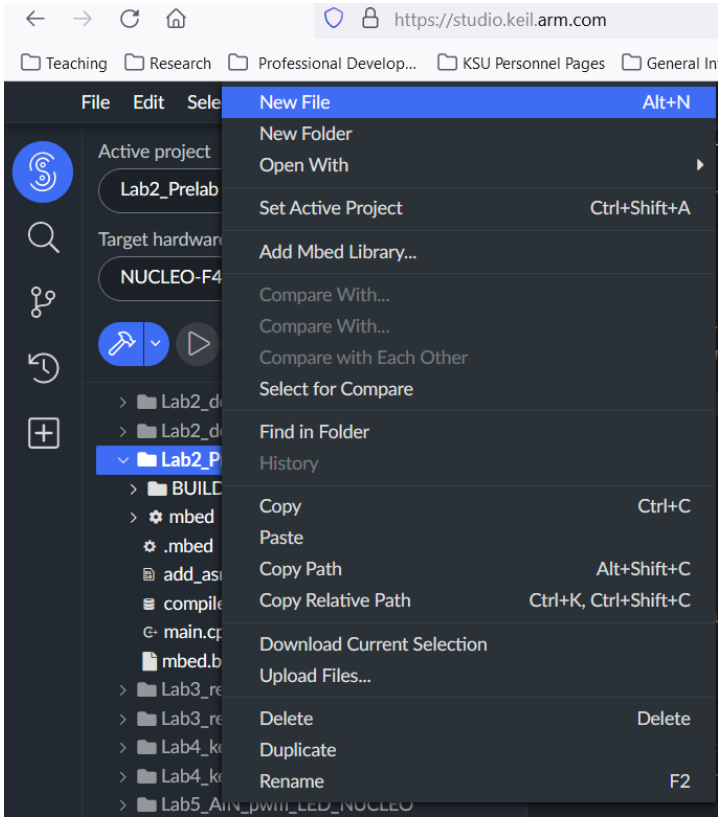


iv. This will create a new project folder in your Project Workspace tree with the name Lab_6_Prelab. **Select the main.cpp file.**

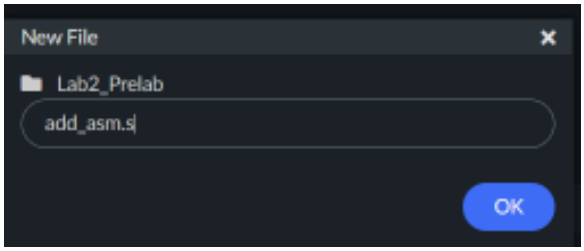


v. **Copy and paste the code** given **for main.cpp** into this file and **save it**.

vi. **Right-click** the **Lab_6_Prelab Folder** and **select New File**.



vii. Under the New File dialog box, e**nter add_asm.s** and **click OK**.



viii. **Copy and paste the code** given **for add_asm.s** into this file and **save it.**

ix. Your program is now ready to compile, build and run. **Click**  **button to run** (build and download) the **project**. **Sometimes this fails**, if this is the case **click**  **button** to build the project, then manually download the project.

x. **Open your terminal program** (Putty or CoolTerm). Verify that the terminal displays the following every 0.5 s.:

```
add_asm computes 2 + 3 = 5
```

xi. Take a screenshot of terminal output.

Laboratory Procedure:

For the lab, you will use the C++ program and assembly subroutine structure introduced in the prelab to create a program that allows you to control on/off time of your board's blinking led. Your project will consist of 2 files: main.cpp and delay_asm.s, which are given below.

The code in main.cpp creates a DigitalOut object `myled` that you map to `LED1` of your microcontroller. You invert the state of `myled` within the `while(true)` loop to turn your led on and off repetitively. The call to assembly subroutine `delay_asm()` creates a delay before the next iteration of the `while(true)` loop, which means that `delay_asm()` controls the on/off time of `LED1`. The Timer object `t` is used to measure the on/off time in software and enables you to print the on/off time to the terminal screen.

main.cpp

```
#include "mbed.h"

DigitalOut myled(LED1);
Timer t;


extern "C" void delay_asm(void);

int main() {
    int begin, end;

    while(true) {
        t.start();              // start timer
        begin = t.read_ms();

        myled = !myled;         // invert LED state
        delay_asm();            // sets led on/off time

        end = t.read_ms();
        t.stop();               // stop timer
        printf("LED on/off time = %d ms \n", end - begin);
    }
}
```

The code in delay_asm.s uses register `R0` to control the delay time. It does this by loading an initial value into `R0`, then repetitively decrements the value stored in `R0` by 1 until `R0 == 0`. When `R0 == 0`, execution switches back to main.cpp. This means that the execution time of `delay_asm()` is determined by the initial value of `R0`.

delay_asm.s

```
            AREA    |.text|, CODE, READONLY

delay_asm   PROC
            EXPORT delay_asm
            LDR R0, =0x00800000     ;set initial value of R0
            MOV R1, #1
LOOP        SUBS    R0, R1
            BNE LOOP
            BX  LR
            ENDP
            ALIGN
            END
```

1. Using the procedure described in the Prelab, **create** a **program** called **Lab_6_delay_asm in** the **Arm Keil Studio** IDE. Lab_6_delay_asm will have 3 components.
    a. main.cpp (code given above)
    b. delay_asm.s (code given above)
    c. Mbed 2 library
2. **Build/Run the program**.

3. **Record** the **on/off time** of your LED using your terminal application.

    LED on/off Time: _____

4. The initial value of `R0` determines the on/off time of the LED. **Convert** the **initial value of `R0` into decimal** and **compute the ratio** of `R0` value to LED on/off time.
    Ratio of R0 value to LED on/off Time: _____

5. Modify the initial value of R0 and observe the change in on/off time of the LED.
6. Use the ratio from 4 to determine the initial value of R0 in hexadecimal that will give an LED on/off time of 1 ms.

    R0 Setting for 1 ms on/off Time: _____

Discussion Questions

1. Refer to Table 3.1. Cortex-M4 Instruction Set Summary at the url below, and describe in your own words the operations accomplished by the `MOV` and `SUBS` instructions in the delay_asm.s file.
   https://developer.arm.com/documentation/ddi0439/b/Programmers-Model/Instruction-set-summary/Cortex-M4-instructions?lang=en

2. The largest value that can be stored in `R0` is `0xFFFFFFFF`. What is the longest on/off time (in seconds) you can achieve using `delay_asm()`?
3. An inspection of Table 3.1 will show that each instruction takes a specific number of clock cycles to execute. Use the clock cycle information from Table 3.1 to determine the number of clock cycles it takes to execute all the instructions in `delay_asm()`. Note the following:
   a. The executable instructions run from line 5 to 9.
   b. Instructions on lines 7 and 8 repeat multiple times until `R0 == 0`.
   c. Assume that $P = 1$ for computing the number of cycles corresponding to the BNE instruction.
4. Using the result from 3 and your on/off time measurement, estimate the clock rate of your processor in Hz (clock cycles per second).

Report Requirements

Prelab

Screenshot of terminal output (30 points)

Lab Results

- Video of blinking LED (10 points)
  Please post the video link only when you submit under the D2L Assignments folder.
- LED on/off time (10 points)
- Ratio of R0 value to LED on/off Time (10 points)
- R0 Setting for 1 ms on/off Time. (10 points)

Answers to Discussion Questions

1. 10 points
2. 10 points
3. 5 points
4. 5 points

No formal lab report is needed for this lab.