ELECTRICAL AND ELECTRONICS ENGINEERING DEPARTMENT

Programming with Subroutines and Parameter Passing

Experiment 2 - Programming with Subroutines and Parameter Passing

Objectives

A subroutine is a reusable program module. A main program can call or jump to the subroutine one or more times. The stack is used in several ways when subroutines are called. In this lab you will learn:

- How to write subroutines and call them from the main program
- Ways to pass parameters to and from subroutines
- How to create flowcharts and develop pseudo-codes

1 Background Information

1.1 Subroutines

A subroutine is a program module that is separate from the main or calling program. Frequently the subroutine is called by the main program many times, but the code for the subroutine only needs to be written once. When a subroutine is called, program control is transferred from the main program to the subroutine. When the subroutine finishes executing, control is returned to the main program. Using subroutines saves memory, but a greater benefit is improved program organization. In future labs, you will find it beneficial to use subroutines for various functions of your programs.

1.2 Calling Subroutines

You will recall, the program counter, or register PC (R15), always contains the address of the next instruction to be executed in a program. Calling a subroutine is similar to an unconditional branch as the program jumps to a different address other than the next address. The difference is that when the subroutine finishes executing, control goes back to the instruction after the subroutine call in the main program. A BL instruction causes the address of the next memory instruction to be pushed onto R14 (Link Register or LR) and the argument of BL to be loaded into the program counter. The argument of BL is the starting address of the subroutine. But, when you write your program, you just give it a name and the assembler figures out the rest. At the end of a subroutine, the instruction BX LR causes what was last stored in LR to be loaded into the program counter. In this way, the instruction after the BL in the main program is the next one executed. Therefore, it is important to recall that your subroutine should end with Branch and link with eXchance of the address stored in the Link Register, i.e., BX LR.

1.3 Coding with Subroutines

It is explained in Section 1.2 that a subroutine is called via BL instruction with the label of the subroutine as the operand. Thus, the starting address of the subroutine should be labeled. This can easily be done by using PROC and ENDP assembly directive. PROC indicates the starting of a procedure (subroutine) and ENDP designates the end of a procedure. The label for the line where PROC is placed actually is the label of the address of the first instruction of the subroutine. You may place your subroutines into the same source file where your main program exists. On the other hand, for more generic subroutines to be used in other projects, you may place your subroutines into separate source files so that different main programs can achieve them. In that case, you should tell the compiler that you are using subroutines from different source files in order to link them. This is done by using EXPORT and EXTERN/IMPORT assembler directives. EXPORT is used for the subroutines to make the label of the subroutine available to other source files. EXPORT directive should be followed by the name of the subroutine (e.g. EXPORT name_of_the_subroutine). EXPORT tells the compiler to remember the label to be possibly referred by other source files. EXTERN or IMPORT is used for the source files that are using labels (i.e. subroutines) that are defined in separate source files. EXTERN/IMPORT directive should be followed by the name of the subroutine that exists in one of the files added to the project (e.g. EXTERN name_of_the_subroutine). EXTERN/IMPORT tells the compiler to gather the address of the label to be referred by the current source file. An example of a subroutine coding can be:

```
;LABEL          DIRECTIVE       VALUE                           COMMENT

                                AREA    routines, CODE, READONLY
                                THUMB
                                EXPORT  Routine_name            ; make it available
                                                                ; to other sources
Routine_name    PROC
                ...             ...                             ; your routine
                BX              LR                              ; return
```

An example of a main program coding with subroutines placed in separate files can be:

```
;LABEL          DIRECTIVE       VALUE                   COMMENT

                AREA            main, CODE, READONLY
                THUMB
                EXTERN          Routine_name            ; Reference external subroutine
                                                        ; IMPORT can also be used
                EXPORT          __main
                ENTRY                                   ; execution starts from here
__main          PROC
                ...             ...                     ; your code
                BL              Routine_name            ; call to your subroutine
                ...             ...
                ENDP
```

If the subroutine and the main program are placed in the same source file, then EXPORT and EXTERN directives can be omitted.

1.4 Parameter Passing

A parameter is passed to the subroutine by leaving the data in a register, or memory, and allowing the subroutine to use it. A parameter is passed back to the main program by allowing the subroutine to change the data. This is the way parameters are passed in assembly language. When the parameter being passed to the subroutine is in a register, this is referred to as the call-by-value technique of passing parameters. If the data is in memory and the address is passed to the subroutine, this is called call-by-reference. It is important to document all parameter-passing details in subroutines.

1.5 Pseudo-code Generation

A pseudo-code uses an expressive, clear, and concise method to describe an algorithm. It may have short English phrases, arrows (←) to indicate storing of data into variables, or it may look somewhat like a high level programming language without the details of the syntax. Since the pseudo-code focuses on the algorithm to solve the problem, instead of the syntax, it abstracts out the problem solving stage from the code writing stage with appropriate instructions and syntax. Below is an example of a problem statement and the pseudo-code developed for it.

Problem: Write an assembly language source code to be executed on an MCU with two registers A and B as follows: The program should read three unsigned numbers at memory addresses `0x0000`, `0x0001`, and `0x0002`, sort them from largest to smallest, and store them to addresses `0x0003`, `0x0004`, and `0x0005`. i.e. at the end of the execution, memory addresses `0x0003` and `0x0005` should contain the largest and smallest numbers respectively.

Pseudo-code: Note that this is just one algorithm to solve this problem. A and B stand for register A and register B for convenience, but they do not have to be in a pseudo-code.

```
A ← 1st Number (0000)
B ← 2nd Number (0001)
If A < B
       A → Smallest (0005)
       B → Largest  (0003)
Else
       B → Smallest
       A → Largest
A ← Largest     (0003)
B ← 3rd Number  (0002)
If A < B
       B → Largest
       A → Middle (0004)
       Done.
Else
       B → Middle
       A ← Smallest
       If A < B
             Done.
       Else
             A → Middle
             B → Smallest
       Done.
```

1.6 Flowcharts

A flowchart illustrates the steps in a process. It contains different symbols to indicate various tasks that the program has to do. There are many dedicated symbols for different types of tasks, but not all have to be utilized for the algorithm to be clear. In general the tasks in the regular flow are indicated by rectangular boxes, and a decision point to change the program flow (If-Then-Else, Repeat-Until) is indicated by diamonds. For example the algorithm described in the previous section can be depicted in a flowchart as below.

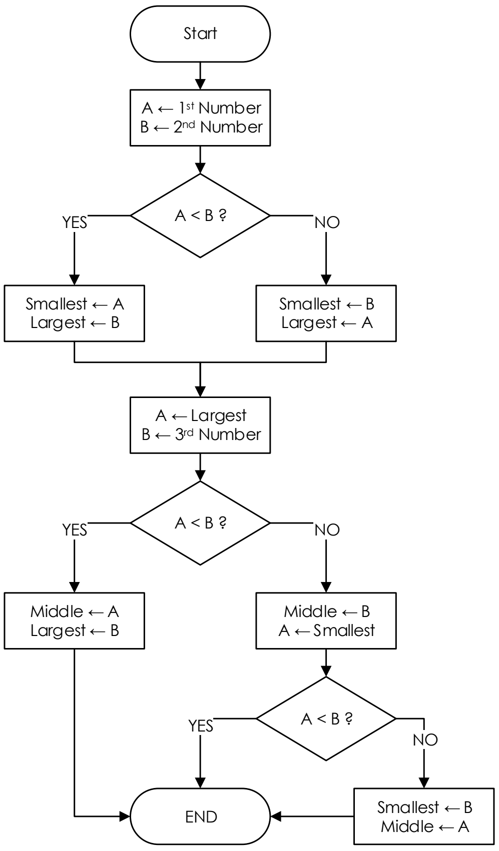

Figure 1: Example Flowchart for the Pseudo-code Given in 1.5

2 Preliminary Work

1. Write a subroutine, **CONVRT**, that converts an m-digit decimal number represented by n **bits** ($n < 32$) in register R4 into such a format that the ASCII codes of the digits of its decimal equivalent would listed in the memory starting from the location address of which is stored in register R5. When printed using OutStr, the printed number is to contain no leading 0s, that is, exactly m digits should be printed for an m-digit decimal number. Before writing the subroutine, the corresponding pseudo-code or flow chart is to be generated.
2. Write a program that, in an infinite loop, waits for a user prompt (any key to be pressed) and prints the decimal equivalent of the number stored in 4 bytes starting from the memory location **NUM**. Note that you may define **NUM** by using proper assembly directives. In this part, you are expected to use the subroutine you are written in Part-1. Explain which arguments should be passed and how.
3. Write a program for decimal number guessing using binary search method. The number is to be an integer in the range $(0, 2^n)$, i.e. $0 < number < 2^n$, where $n < 32$ and n is determined by a user-input. Then, the guessing phase is to be handled through a simple interface where the processor outputs its current guess in decimal base and calculate the next according to the user inputs, **D** standing for down, **U** standing for up, or **C** standing for correct. To fulfill the requirements given above, include the subroutine **CONVRT** from the Part-1 in your main program as well as a new subroutine **UPBND** that updates the search boundaries after each guess. Prior to writing the code itself, draw a flowchart of the main algorithm leaving the subroutine parts as black boxes.

3 Experimental Work

Attempt all the following items for full credit. While attempting each item, please follow your Teaching Assistant's (TA) instructions in order to make progress much more robustly. Notify your TA when you finished an item.

1. Verification of decimal number printing program: Execute the code you have written in the 2^{nd} step of the preliminary work (Section 2.2). Demonstrate your results to your lab instructor.
2. Verification of number guess program: Execute the code you have written in the 3^{rd} step of the preliminary work (Section 2.3). Demonstrate your results to your lab instructor.