Spring 2022

Haroutun Haroutunian
Andres Lopez

California State University Northridge
Department of Electrical and Computer Engineering

Experiment 8
Writing and Calling Subroutines

Professor Flynn
1 April, 2022

Introduction

In this experiment, we were introduced to subroutines which are small programs or functions that perform common tasks. These are similar to functions in high level languages such as C++ , Java, Python etc.. Subroutines are useful for iterative programming tasks that need to be performed on multiple occasions and improve the program flow.

Procedure

In task 1, students are to write a complete program which will create a "running" LED pattern on the EduBoard, utilizing all 8 LEDs connected from P0.8 through P0.15. In other words, each LED must turn on back to back with a short delay between toggles through a separate file which will act as a subroutine. In figure 8.1, programmers utilize the EQU mnemonic to define the IO0DIR, PIN, SET, CLR, PINSEL0, and DELAY_LED labels, similarly done in previous labs. Students imported "DELAY," which is the delay subroutine written in ARM7 assembly language, and "delay_C," which is the delay subroutine written in C++. The following symbols are utilized for the task, determining the functionality of the pins.

- PINSEL0 = assigns the pins P0.8-P0.15 as GPIO(General Purpose Input/Output) pins
- IO0DIR = used to set the direction of the pins as with inputs or outputs
- IO0PIN =sends a 0 or 1 to the port pins connected to the LEDs
- IO0SET = used to set a "1" to a given bit in the IO0PIN memory address
- IO0CLR = used to clear a bit in the IO0PIN register, by placing "1" in memory
- DELAY_LED = Used to determine how long the delay should occur

```
1       GLOBAL user_code
2       IMPORT  Reset_Handler
3       IMPORT  DELAY
4   PINSEL0     EQU 0XE002C000
5   IO0DIR      EQU 0XE0028008
6   IO0PIN      EQU 0XE0028000
7   IO0SET      EQU 0XE0028004
8   IO0CLR      EQU 0XE002800C
9   DELAY_LED   EQU 300000
```

Figure 8.1: Beginning of code and Label initialization

In order to begin with the tasks, the LED pins on the EduBoard must be set as GPIO and set as inputs, as the tasks require the LEDs to toggle. As shown in figure 8.2, the 8 LED pins are set as inputs. Once set to inputs, register 8, which asks as a mask, sets P0.8 through P0.15, which undergoes negative logic and turns the LED's off.

```
12  user_code
13  ;TASK1  AND TASK2
14      LDR r4,=PINSEL0
15      LDR r5,=IO0DIR
16      LDR r6,=IO0SET
17      LDR r7,=IO0CLR
18      ;PINSEL0
19      LDR r8,=0xFFFF
20      LDR r9,[r4]
21      AND r9,r9,r8
22      STR r9,[r4]
23      ;IO0DIR
24      LDR r8,=0xFF00
25      LDR r9,[r5]
26      ORR r9,r9,r8
27      STR r9,[r5]
28
29      STR r8,[r6] ;TURN LEDs off
```

Figure 8.2: Initialization of 8 LED pins

As mentioned in the lab manual, the LEDs must turn on in a "running" pattern. The program must turn on the first LED and wait a specified amount of time before turning on the next LED. On line 30 in figure 8.3, students load Register 8 with 0x100, which points to P0.8, which is the first LED that must be toggled. Register 9 is then loaded with the value of 8, which refers to the number for iterations the loop must be performed. A loop will simplify the program since the LEDs must toggle with the same delay in between. Lines 32 through 41 is the loop in which the LEDs are toggled. Register 0 is utilized as the time delay between each toggle. Register 8 is utilized as a mask when storing into Register 7 to toggle P0.8 LED. Line 35 branch links to the file where the DELAY subroutine is located, as seen in figure 8.4. Once delayer, the respective LED is then turned off and Register 8 which contains the mask to turn on the first LED is incremented. Incrementing Register 8 by 1 would point to P0.9, which is the second LED pin that must be toggled. Register 9 is then decremented to keep track of the number of iterations left. Line 39 would act as a conditional statement which continues the loop until Register 9 reaches "0." Line 40 restarts the entire loop, so the toggling of the LEDs goes on indefinitely.

```
30  LOOP1   LDR r8,=0x100n
31          MOV r9,#0x8;counter
32  LED_RUN LDR r0,=DELAY_LED
33          STR r8,[r7] ;TURN ON LED
34
35          BL  DELAY   ;BRACH LINK TO DELAY SUBROUTINE
36          STR r8,[r6] ;TURN OFF LED
37          LSL r8,r8,#0x1 ; NEXT LED
38          SUBS    r9,r9,#0x1
39          BNE LED_RUN ;
40          B   LOOP1   ;
41  STOP    B   STOP
42      END
43
```

Figure 8.3: Running LED loop

Figure 8.4 presents the delay subroutine written in ARM7 Assembly language, which is located in a separate file. This subroutine must be globalized in order to be imported and utilized in other files, such as task 1 in figure 8.3. All subroutines include similar beginnings and endings, which are lines 9-10 and 13-14. STMEA and LDMEA mnemonics store and load multiple registers in "Empty Ascending" order, respectively. A stack is utilized to save the registers and link register from the calling file, preventing alterations in the subroutine. Similarly, MRS mnemonic stores the current CPSR into a register, protecting flags from the calling program to be altered in the subroutine. In other words, the 4 lines allows programmers to go into a subroutine, perform tasks, and get out as if nothing happened. Lines 11-12 refers to the delay loop, which "grabs" the contents of Register 0 from the calling program, decrements until 0 then returns back.

```
1  ;delay subrountine
2      GLOBAL DELAY
3      IMPORT  user_code
4      IMPORT  LED_TOGGLE
5      IMPORT  ALL_LED
6      AREA    mycode,CODE,READONLY
7
8  DELAY
9      STMEA   SP!,{r4-r9,LR}
10     MRS r1,CPSR
11 LOOP1   SUBS    r0,r0,#0x1  ;decrement loop
12     BNE LOOP1
13     MSR CPSR_F,r1   ;restore flags
14     LDMEA   SP!,{r4-r9,PC}
15 STOP    B   STOP
16     END
17
18         void    delay_C(int delay_time)
19 {
20     while(delay_time !=0)
21     {
22         delay_time--;
23     }
24 }
```

Figure 8.4: ARM7 Assembly Language written delay subroutine.

Task 2 requires students to perform the "running" pattern of the LEDs in Task 1 but instead of writing the delay subroutine in Assembly language, it must be written in C language. As shown in figure 8.5, the only thing that changes from figure 8.3 are lines 35 and 36. PRESERVE8 directive specifies that the current file requires or preserves eight-byte alignment of the stack. Line 36 branch links to the C file which consists of the delay subroutine.

```
30  LOOP1   LDR r8,=0x100
31          MOV r9,#0x8;counter
32  LED_RUN LDR r0,=DELAY_LED
33          STR r8,[r7] ;TURN ON LED
34          PRESERVES
35          BL  delay.c ;BRACH LINK TO DELAY SUBROUTINE
36          STR r8,[r6] ;TURN OFF LED
37          LSL r8,r8,#0x1 ; NEXT LED
38          SUBS    r9,r9,#0x1
39          BNE LED_RUN ;
40          B   LOOP1   ;
41  STOP    B   STOP
42      END
```

Figure 8.5: Task 2 running LED loop

In the figure below, the delay subroutine named delay_C from figure 8.4 is written in C language. The code passes the value of Register 0, which is DELAY_LED, into delay_time, decrements it and returns back to the calling program. The code decrements delay_time until it reaches 0, since the "while" loop is true until delay_time doesn't equal 0.

```
1 void  delay_C(int delay_time)
2 {
3   while(delay_time !=0)
4   {
5     delay_time--;
6   }
7 }
```

Figure 8.6: C written delay subroutine

In task 3, students are to write a complete program in which the first 4 LEDs are initialized to be turned off, then toggle through a push-button. Pressing the button once would either turn the LEDs on or off. As shown in figure 8.7, a Read-Modify-Write formatted program would be utilized to specify that only the 4 LEDs and PIN 14 are needed throughout task 3. From lines 13-17, PINSEL0 is modified to specify the needed pins. From lines 19-23, pin 14 is initialized as an input so it can be used as the push-button. P0.8-11 are then initialized as outputs, which are the LEDs. From lines 25-29, the 4 LEDs are set off, as specified in the lab manual

```
4       IMPORT Reset_Handler
5       IMPORT DELAY
6   PINSEL0 EQU 0XE002C000
7   IO0DIR  EQU 0XE0028008
8   IO0PIN  EQU 0XE0028000
9   DELAY_SWITCH    EQU 200000
10      AREA    mycode,CODE,READONLY
11
12  TOGGLE
13      LDR r4,=PINSEL0
14      LDR r5, [r4]
15      BIC r5,r5,#0x30000000   ;MODIFY BITS 28-29 FOR PIN 14
16      BIC r5,r5,#0x00FF   ;MODIFY BITS 16-23 FOR PINS 8-11
17      STR r5,[r4]
18
19      LDR r4,=IO0DIR
20      LDR r5, [r4]
21      BIC r5,r5,#0x4000
22      ORR r5,r5,#0xF00;MODIFY PINS 8-11 TO BE OUTPUTS
23      STR r5,[r4] ;Write]
24
25      LDR r7,=0xF00   ;LEDs 8-11
26      LDR r4,=IO0PIN
27      LDR r5, [r4]    ;
28      ORR r5,r5,r7    ;TURN OFF LEDS
29      STR r5,[r4]
30
```

Figure 8.7: Task 3 initialization of Pins

Utilizing knowledge from previous labs when toggling LEDs with a push-button, students use the IO0PIN to read whether the button is pressed. Line 35 of figure 8.8 branches to itself which constantly checks the status of the button. Register 0 is also utilized in this task as the clock delay which passes to the delay subroutine to prevent debouncing, as shown previously in figure 8.4. Debouncing was described in a previous lab as a small ripple of movement caused by a mechanical switch due to a small series of short contacts. If debouncing were to be removed from the code, it would cause the LEDs to rapidly toggle and cause confusion in the experiment. Lines 39 - 51 checks whether the button is pushed or not as well as turns the LEDs on or off. In line 43, if the button is released, then it would branch to itself constantly until it is pushed, which would continue on through the program. Once the code reaches line 48, the LED's would turn off if the button was pushed, then reach line 51 and check whether the button was pressed again. Lines 53 - 55 utilizes the ORR mnemonic to turn the LEDs off if they are on, which is called on line 48.

```
31  PUSHCHECK           ;check if pin 14 is pushed
32      LDR r7,[r4] ;IO0DIR contents
33      BIC r7,r7,#0x4000   ;MODIFIES BIT 14 AS INPUT
34      STR r7,[r4]
35      LDR r6,[r5] ;READS CONTENTS OF IO0PIN
36      AND r8,r6,#0x4000   ;READS 14 BIT IN ADDRESS
37      CMP r8,#0x4000  ;CHECKS FOR PIN , 1 OFF, 0 ON
38      LDR r7,[r4] ;READ CONTENTS IN IO0DIR
39      ORR r7,r7,#0x4000   ;MODIFY BIT 14 AS OUTPUT
40      STR r7,[r4]
41      BEQ PUSHCHECK
42      LDR r0,=DELAY_SWITCH
43  STOP1   BL  DELAY   ;prevent bouncing
44
45
46  RELEASE LDR r7,[r4]
47      BIC r7,r7,#0x4000   ;MODIFIES BIT 14 AS INPUT
48      STR r7,[r4]
49      LDR r6,[r5]
50      AND r8,r6,#0x4000
51      CMP r8,#0x4000  ;CHECK PIN 14(1 =OFF)   (0 = ON)
52      LDR r7,[r4]
53      ORR r7,r7,#0x4000   ;MODIFY PIN 14 TO BE OUTPUT
54      STR r7,[r4]
55      BNE RELEASE ;WAITS UNTIL BUTTON RELEASED
56      LDR r0,=DELAY_SWITCH
57  STOP2   BL  DELAY   ;prevent bouncing
58      LDR r6,[r5]
59      AND r8,r6,#0xFF00
60      CMP r8,#0xFF00  ;LEDs OFF
61  STOP3   BNE TURNOFF
62      BIC r6,r6,#0xFF00
63      STR r6,[r5]
64      B   PUSHCHECK
65
66  TURNOFF
67      ORR r6,r6,#0xFF00
68      STR r6,[r5]
69      B   PUSHCHECK
70
71  STOP    B   STOP
72      END
```

Figure 8.8: Task 3 push-button check and toggle of LEDs

Task 4 requires students to write a complete program which would repeat task 3 but with all 8 LEDs, P0.8-P0.15. The program would run through a problem if task 3 was completely repeated due to P0.14 because of an LED and a push-button pin. Thus, P0.14, would need to be set as an input while the other 7 pins are outputs. Once P0.14 is pushed, then it would turn into an output pin and act accordingly with the other pins. The beginning of the program is repeated as shown in figure 8.9 with task 3 of initializing all pins as outputs to turn all LEDs off initially.

```
6  PINSEL0 EQU 0XE002C000
7  IO0DIR  EQU 0XE0028008
8  IO0PIN  EQU 0XE0028000
9  DELAY_SWITCH    EQU 200000
10     AREA    mycode,CODE,READONLY
11 ALL_LED
12     LDR r4,=PINSEL0 ;GPIO
13     LDR r6,=0xFFFF
14     LDR r5,[r4] ;PINSEL0
15     AND r5,r5,r6    ;modify bits 8-15
16     STR r5,[r4]
17
18     LDR r4,=IO0DIR  ;I/O
19     LDR r5,[r4]
20     ORR r5,r5,#0xFF00   ;modify bits 8-15 to be output
21     STR r5,[r4]
22
23     LDR r5,=IO0PIN
24     LDR r6,[r5]
25     ORR r6,r6,#0xFF00   ;sends 1 to pins 8-15 to turn off
26     STR r6,[r5]
```

Figure 8.9: Initializing all LED pins to outputs

Likewise, within the PUSHCHECK label in figure 8.10 consists of checking P0.14 push-button in order to determine whether the LED's should turn off or on. Rather than simply reading P0.14, it would need to be converted as an input since it was set as an output initially to turn the LED off. Once it is converted, it is then read and compared Register 8, which is loaded with the AND of Register 6 (contents of IO0PIN) and #0x4000. After checking P0.14 push-button, it is then converted back to an output to toggle the LED.

```
29 PUSHCHECK           ;check if pin 14 is pushed
30     LDR r7,[r4] ;IO0DIR contents
31     BIC r7,r7,#0x4000   ;MODIFIES BIT 14 AS INPUT
32     STR r7,[r4]
33     LDR r6,[r5] ;READS CONTENTS OF IO0PIN
34     AND r8,r6,#0x4000   ;READS 14 BIT IN ADDRESS
35     CMP r8,#0x4000  ;CHECKS FOR PIN , 1 OFF, 0 ON
36     LDR r7,[r4] ;READ CONTENTS IN IO0DIR
37     ORR r7,r7,#0x4000   ;MODIFY BIT 14 AS OUTPUT
38     STR r7,[r4]
39     BEQ PUSHCHECK
40     LDR r0,=DELAY_SWITCH
41 STOP1   BL  DELAY   ;prevent bouncing
```

Figure 8.10: Push-button check

The "RELEASE" label repeats the task 3 label but does the same process as "PUSHCHECK" by converting P0.14 to an output and back to an input. In figure 8.11, line 51, if the button is released, it would load Register 0 with the clock delay and branch link to the delay subroutine found in a separate file to prevent debouncing. Lines 54-56 checks P0.8-P0.15 if the LEDs are on or off, and if they're on, they would be turned off by branching to the "TURNOFF" label from line 57. Line 58 would clear the bits with the BIC mnemonic then continue to line 60 to check whether the button is pressed. Lines 62-65 consists of the "TURNOFF" label, which simply ORR's Register 6 (LED status) with 0xFF00, then stored into Register 5. Finally, the program would then branch to check the status of the push-button.

```
46  RELEASE LDR r7,[r4]
47      BIC r7,r7,#0x4000   ;MODIFIES BIT 14 AS INPUT
48      STR r7,[r4]
49      LDR r6,[r5]
50      AND r8,r6,#0x4000
51      CMP r8,#0x4000  ;CHECK PIN 14(1 =OFF)   (0 = ON)
52      LDR r7,[r4]
53      ORR r7,r7,#0x4000   ;MODIFY PIN 14 TO BE OUTPUT
54      STR r7,[r4]
55      BNE RELEASE ;WAITS UNTIL BUTTON RELEASED
56      LDR r0,=DELAY_SWITCH
57  STOP2   BL  DELAY   ;prevent bouncing
58      LDR r6,[r5]
59      AND r8,r6,#0xFF00
60      CMP r8,#0xFF00  ;LEDs OFF
61  STOP3   BNE TURNOFF
62      BIC r6,r6,#0xFF00
63      STR r6,[r5]
64      B   PUSHCHECK
65
66  TURNOFF
67      ORR r6,r6,#0xFF00
68      STR r6,[r5]
69      B   PUSHCHECK
70
71  STOP    B   STOP
72      END
```

Figure 8.11: Push-button check and toggle of LED's task 4

Discussion

This experiment required a subroutine that was in charge of continuously running LEDs and turning them on and off. At first, we did not utilize the stack for this task. Although the program had no issues compiling, debug showed undesirable results. The Stack is a memory region within the program/process. This part of the memory gets allocated when a process is created. We use Stack for storing temporary data such as registers of subroutines.The stack is mostly to keep calls/returns in order(which needs to save the state of the CPU) and push existing values in registers it will use and pop them before returning. We were not certain as to why we got different results than the theoretical ones but we believe not using the stack is the reason why we ran into issues since data would be overwritten or lost without the Link Register.

Conclusion

Utilizing subroutines and branches in this experiment allows for the flexibility to make robust programs. Subroutines are convenient and expansive methods to create functions to organize our code. Subroutines, just like functions, can be used to optimize code and minimize repetitive code. They also have the ability to be used recurvesely. Students learned that subroutines can also be utilized from one programming language to another, for example, delay_C within the experiment was written in C language and the "calling program" was written in ARM7 Assembly.