Reverse Polish Notation Calculator

Course: ENGG*3640 Microcomputer Interfacing

Instructor: Radu Muresan

Student Names/Numbers:

Ali Akhdar (1068542)

Bilal Ayyache (0988616)

Mohammed Al-Fakhri (0982745)

Emeka Madubuike (0948959)

Date: 08/10/2018

Contents

1. Introduction

The Purpose of lab 1 was to get familiar more familiar with using the Kiel Uvision program interface, to complete Lab 2, it was mandatory to understand how the stack functions can be used in assembly to implement a calculator. The biggest problems presented by the lab were using the stack memory and board memory and being able to convert between number systems. The program designed in this lab used RPN, reverse polish notation, to perform the basic arithmetic operations (*, +, /, -) through stack functions.

The RPN calculator, also known as Reverse Polish Notation, is a mathematical notation where operand is inputted first then their operators. For example, in regular algebraic notation; 30 + 70 = 100. In RPN; 30 70 + = 100. The simplest way of building a calculator like this is using stacks in assembly. Each argument the user inputs will get stored in the stack, when the operator is inputted, the inputs gets popped from the stack and the required operation is performed.

1.1. Lab Description

In this experiment, the main task of the lab was implementing an RPN calculator using the assembly language and the skills learned from lectures and Lab 1. To implement the calculator, an input was obtained by the user through Putty, the input is then converted from ASCII characters to binary and saved in the stack. The program then performs the arithmetic operation by popping the values from the stack. After operation is completed, the resulting values are then pushed into the stack. The results are then converted from binary back to ASCII characters so they can be displayed. Finally the program prints out the result once the user requests the output.

1.2. System Requirements

During this lab, the tools and equipment that were used are enumerated below:

- Kiel Uvision Program was used to create instructions to the K60 Microcontroller
- FreeScale TWR-K60D100M Microcontroller was used to implement instructions sent
- A Hi Speed USB 2.0 Connection cable between PC and board was used to connect the Microcontroller to the PC.
- Putty displayed the results by receiving commands through the COM port in which the USB 2.0 was connected to.

2. Background

2.1. Equipment

- **Kiel Uvision Program**: The Kiel Uvision program is an IDE that combines project management, source code editing, program debugging, and run-time environment into a single powerful environment. Using this environment, the user is able to easily and efficiently test, verify, debug, and optimize the code developed. During the first lab, the debug functionality helped in understanding how the code is acting.

- **K60 Microcontroller**: The K60 Microcontroller (Figure 2.1) from NXP contains a low power MCU core ARM Cortex-M4 that features an analog integration, serial communication, USB 2.0 full-speed OTG controller and 10/100 Mbps Ethernet MAC. These characteristics makes this

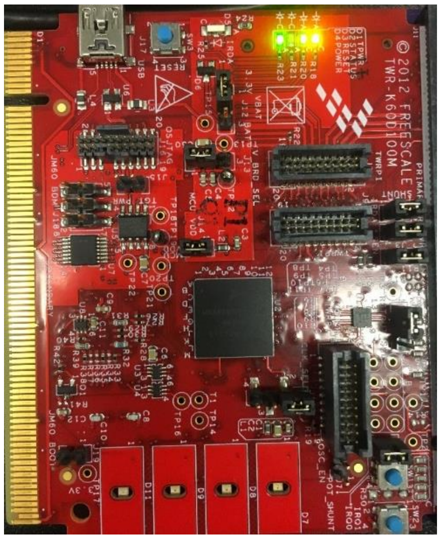


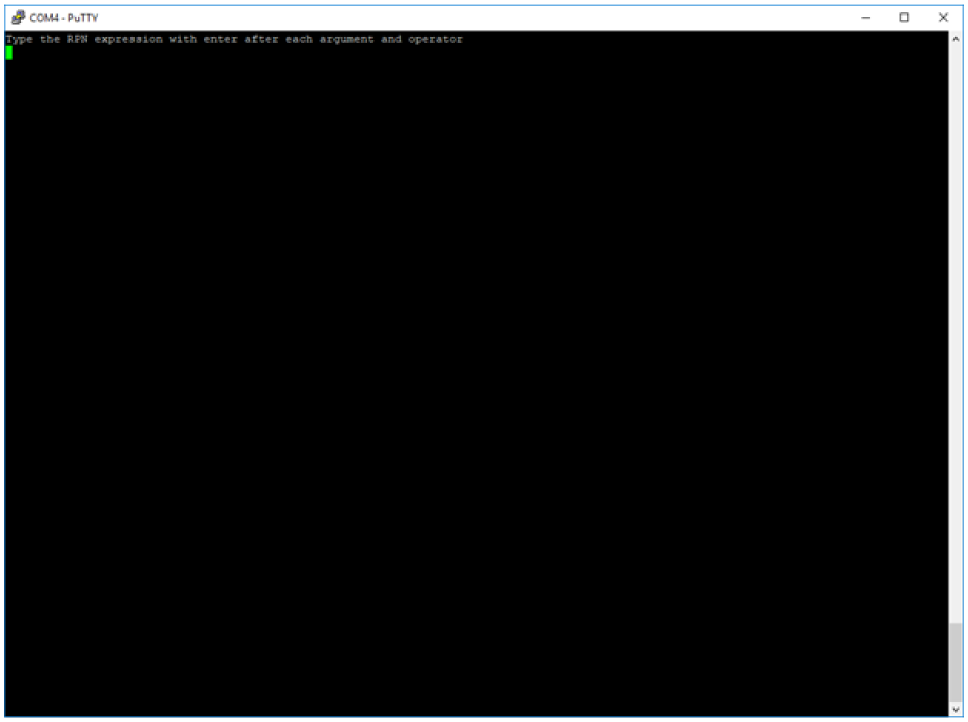
Figure 2.1 1K60 Microcontroller

Microcontroller suitable to preform task in a very efficient and fast manner. A USB connection was used to sync the microcontrollerwith the Keil uVision software that that was provided by the teaching assistant.

3. Implementation

During the implementation of this lab many obstacles were encountered. To overcome these obstacles, the code outline was written on a paper and debugged before typing into Keil for execution. The code consisted of the startup statement receiving inputs. After that, the conversion from ASCII to integer took place so the arithmetic operations could be run.

3.1. Lab Implementation Overview



```
Type the RPN expression with enter after each argument and operator
```

Figure 3 1Running the Code

Figure 3. displays the main menu of the RPN Calculator. The main menu prompts the user to input the RPN expression with an ‘enter’ after each argument and operator. This menu makes it easier for the user to interact with the program. Through this menu, the user can see the final results as the calculation's answer will be displayed on this screen

3.1.1. Software Implementation

In the main.c, we use collaborative programming to code functions using C-Programming. These functions were used to print out results/prompts on the main menu, collect input from the user and store values in registers. The following program is provided in figure A in the appendices section. A print, getchar, and putchar function were created for later use in *mymain.s*.

In *mymain.s* we have our assembly program that implements the lab. When the code is run, 64 bit space is immediately created and stored r7. Then the user is prompted to enter input. The input entered is loaded into registers r0 and r1 and then converted from its ASCII form to integer. After the inputs have been entered, the arithmetic operator is checked for, with the options being addition, subtraction, division, multiplication, and equal. If any of the four first options are selected first, as they should be, the function for that particular arithmetic operations is run on the entered inputs and the answer is stored in r0. If the input is equal then the result is converted from integer back to ASCII.

When converting from ASCII to hexadecimal it's a simple arithmetic function called *hexconversion* that subtracts 0x30 from the ASCII value giving us the hexadecimal equivalent. The back conversion however is not as simple. When the calculations are done the answer is left in binary and hence have to be converted back to ASCII to be displayed in Putty. The *BINtoASCII* function is where this happens.

3.1.2. Hardware Implementation

During the second lab, Implementation of hardware was minimal. The TWR-K60D100M microcontroller was to demonstrate the code implementation. Putty terminal was used to display the results on screen as was it communicating serially with the board this display was possible. The on-board crystal frequency was valued at 100 MHz. The above frequency value was used to calculate the register preload values.

3.2. Block Diagram

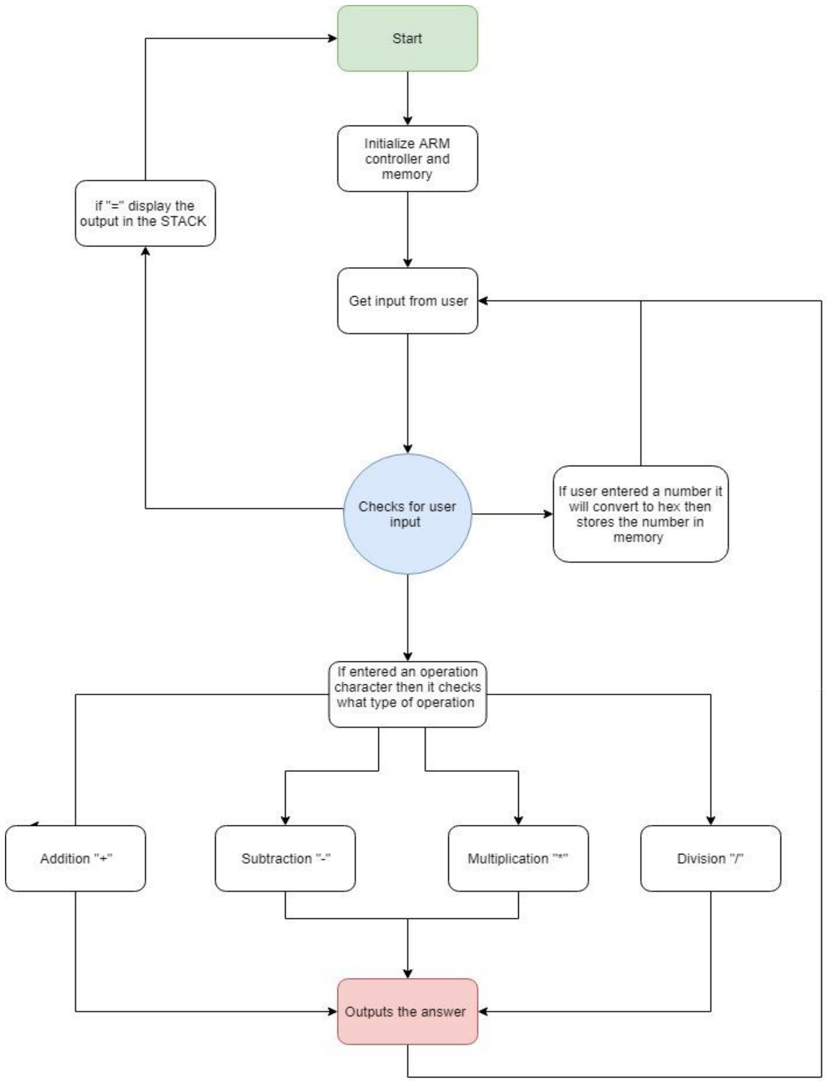
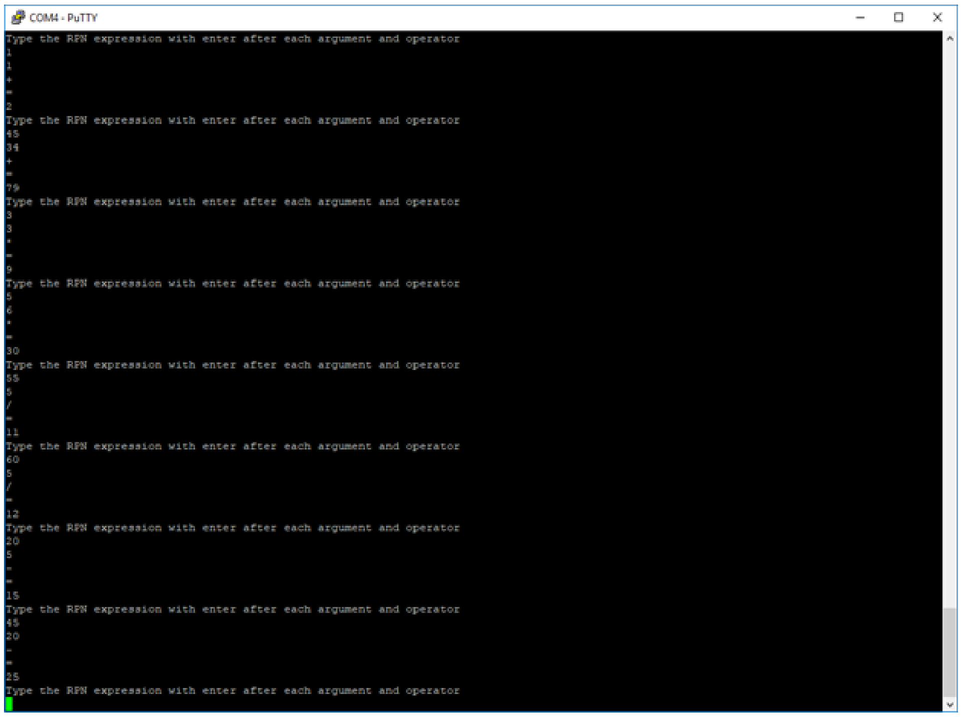

Start
Initialize ARM controller and memory
Get input from user
if "=" display the output in the STACK
Checks for user input
If user entered a number it will convert to hex then stores the number in memory
If entered an operation character then it checks what type of operation
Addition "+"
Subtraction "-"
Multiplication "*"
Division "/"
Outputs the answer

Figure 3. 1. Block Diagram

The first thing that the code does when it starts is to initialize the ARM controller and memory, after that it prompts the user to enter an input. If the user enters a number, then it will convert it hexadecimal and stores the number in the stack and prompts the user again for input. But if the user enters an operation character, it will check what operation needs to be done and outputs the answer when the "=" is pressed and repeats the process.

3.3. Simulation Results



COM4 - PuTTY

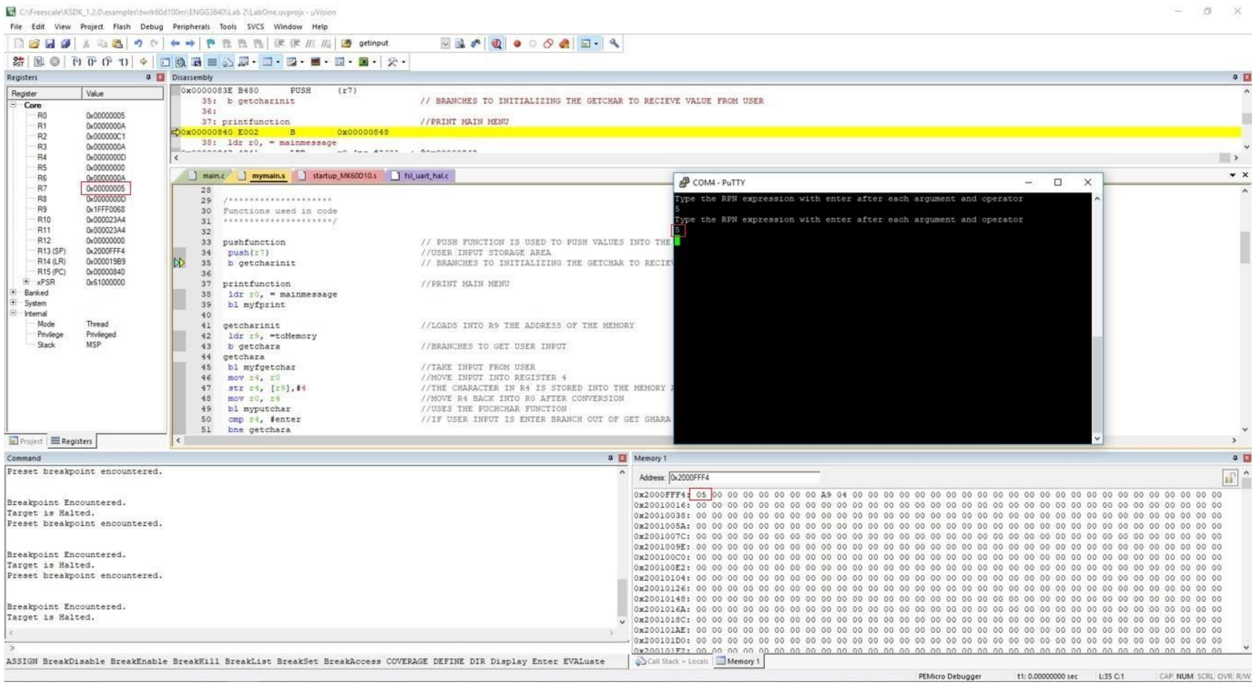
```
Type the RPN expression with enter after each argument and operator
1
1
+
=
2
Type the RPN expression with enter after each argument and operator
45
34
+
=
79
Type the RPN expression with enter after each argument and operator
3
3
*
=
9
Type the RPN expression with enter after each argument and operator
5
6
*
=
30
Type the RPN expression with enter after each argument and operator
55
5
/
=
11
Type the RPN expression with enter after each argument and operator
60
5
/
=
12
Type the RPN expression with enter after each argument and operator
20
5
-
=
15
Type the RPN expression with enter after each argument and operator
45
20
-
=
25
Type the RPN expression with enter after each argument and operator
```

Figure 3.3. 1 Results of running code with inputs

As it is shown in figure 3.3.1, after we run the code, the startup sentence will pop up alone. After the values are entered followed by the basic arithmetic operations the results will be shown after an "=" sign, and then it repeats itself with the startup sentence again.

3.4. Lab Requirements

After inputting the value 5 into the putty terminal. The value gets pushed to register 7. As shown above, the value 05 was stored in the stack pointer as seen in the memory address section above. This value can be changed manually using the debugger by changing the value or that specific register.



```
Registers
Register      Value
Core
  R0          0x00000005
  R1          0x0000000A
  R2          0x000000C1
  R3          0x0000000A
  R4          0x0000000D
  R5          0x00000000
  R6          0x0000000A
  R7          0x00000005
  R8          0x0000000D
  R9          0x1FFF0068
  R10         0x000023A4
  R11         0x000023A4
  R12         0x00000000
  R13 (SP)    0x2000FFF4
  R14 (LR)    0x000019B9
  R15 (PC)    0x00000840
  xPSR        0x61000000
Banked
System
Internal
  Mode        Thread
  Privilege   Privileged
  Stack       MSP
```

```
Disassembly
0x0000083E B480      PUSH     {r7}
    35:  b getcharinit                  // BRANCHES TO INITIALIZING THE GETCHAR TO RECIEVE VALUE FROM USER
    36:
    37: printfunction                   //PRINT MAIN MENU
0x00000840 E002      B        0x00000848
    38:  ldr r0, = mainmessage
```

```
/********************
Functions used in code
*********************/

pushfunction                            // PUSH FUNCTION IS USED TO PUSH VALUES INTO THE
 push{r7}                               //USER INPUT STORAGE AREA
 b getcharinit                          // BRANCHES TO INITIALIZING THE GETCHAR TO RECIEV

printfunction                           //PRINT MAIN MENU
 ldr r0, = mainmessage
 bl myfprint

getcharinit                             //LOADS INTO R9 THE ADDRESS OF THE MEMORY
 ldr r9, =toMemory
 b getchara                             //BRANCHES TO GET USER INPUT
getchara
 bl myfgetchar                          //TAKE INPUT FROM USER
 mov r4, r0                             //MOVE INPUT INTO REGISTER 4
 str r4, [r9],#4                        //THE CHARACTER IN R4 IS STORED INTO THE MEMORY A
 mov r0, r4                             //MOVE R4 BACK INTO R0 AFTER CONVERSION
 bl myputchar                           //USES THE PUCHCHAR FUNCTION
 cmp r4, #enter                         //IF USER INPUT IS ENTER BRANCH OUT OF GET GHARA
 bne getchara
```

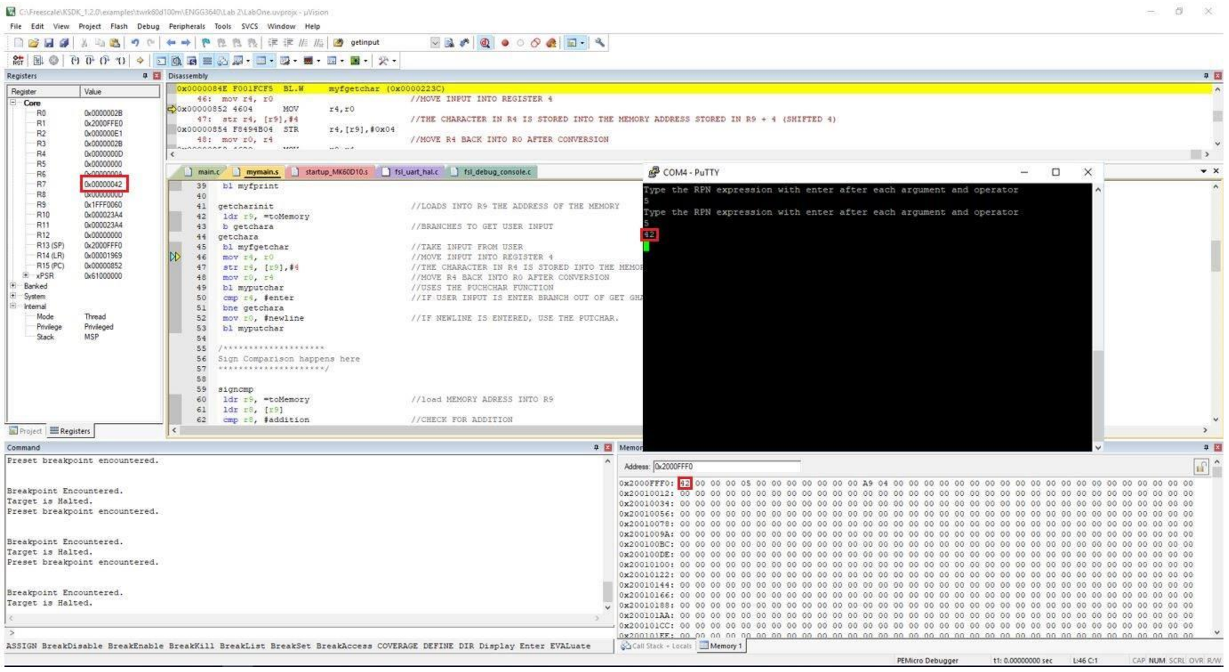
```
COM4 - PuTTY
Type the RPN expression with enter after each argument and operator
5
Type the RPN expression with enter after each argument and operator
5
```

```
Command
Preset breakpoint encountered.


Breakpoint Encountered.
Target is Halted.
Preset breakpoint encountered.


Breakpoint Encountered.
Target is Halted.
Preset breakpoint encountered.


Breakpoint Encountered.
Target is Halted.

ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR Display Enter EVALuate
```

```
Memory 1
Address: 0x2000FFF4
0x2000FFF4: 05 00 00 00 00 00 00 00 A9 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010038: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2001005A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2001007C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2001009E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200100C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200100E2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010104: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010126: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010148: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2001016A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2001018C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200101AE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200101D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200101F2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

PEMicro Debugger    t1: 0.00000000 sec    L:35 C:1
```

Figure 3.3 1 Stack Tracing 1

Register 7 is later set to 42 as the user inputs the second value. The second input is later pushed into the stack pointer, and shifted four bits for conversion purposes.



```
Registers
Register      Value
Core
  R0          0x0000002B
  R1          0x2000FFE0
  R2          0x000000E1
  R3          0x0000002B
  R4          0x0000000D
  R5          0x00000000
  R6          0x0000000A
  R7          0x00000042
  R8          0x0000000D
  R9          0x1FFF0060
  R10         0x000023A4
  R11         0x000023A4
  R12         0x00000000
  R13 (SP)    0x2000FFF0
  R14 (LR)    0x00001969
  R15 (PC)    0x00000852
  xPSR        0x61000000
Banked
System
Internal
  Mode        Thread
  Privilege   Privileged
  Stack       MSP
```

```
Disassembly
0x0000084E F001FCF5  BL.W     myfgetchar (0x0000223C)
    46:  mov r4, r0                     //MOVE INPUT INTO REGISTER 4
0x00000852 4604      MOV      r4,r0
    47:  str r4, [r9],#4                //THE CHARACTER IN R4 IS STORED INTO THE MEMORY ADDRESS STORED IN R9 + 4 (SHIFTED 4)
0x00000854 F8494B04  STR      r4,[r9],#0x04
    48:  mov r0, r4                     //MOVE R4 BACK INTO R0 AFTER CONVERSION
```

```
 bl myfprint

getcharinit                             //LOADS INTO R9 THE ADDRESS OF THE MEMORY
 ldr r9, =toMemory
 b getchara                             //BRANCHES TO GET USER INPUT
getchara
 bl myfgetchar                          //TAKE INPUT FROM USER
 mov r4, r0                             //MOVE INPUT INTO REGISTER 4
 str r4, [r9],#4                        //THE CHARACTER IN R4 IS STORED INTO THE MEMOR
 mov r0, r4                             //MOVE R4 BACK INTO R0 AFTER CONVERSION
 bl myputchar                           //USES THE PUCHCHAR FUNCTION
 cmp r4, #enter                         //IF USER INPUT IS ENTER BRANCH OUT OF GET GHA
 bne getchara
 mov r0, #newline                       //IF NEWLINE IS ENTERED, USE THE PUTCHAR.
 bl myputchar

/********************
Sign Comparison happens here
*********************/

signcmp
 ldr r9, =toMemory                      //load MEMORY ADRESS INTO R9
 ldr r8, [r9]
 cmp r8, #addition                      //CHECK FOR ADDITION
```

```
COM4 - PuTTY
Type the RPN expression with enter after each argument and operator
5
Type the RPN expression with enter after each argument and operator
5
42
```

```
Command
Preset breakpoint encountered.


Breakpoint Encountered.
Target is Halted.
Preset breakpoint encountered.


Breakpoint Encountered.
Target is Halted.
Preset breakpoint encountered.


Breakpoint Encountered.
Target is Halted.

ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR Display Enter EVALuate
```

```
Memory 1
Address: 0x2000FFF0
0x2000FFF0: 42 00 00 00 05 00 00 00 00 00 00 00 A9 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010012: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010034: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010056: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010078: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2001009A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200100BC: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200100DE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010122: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010144: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010166: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20010188: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200101AA: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200101CC: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200101EE: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

PEMicro Debugger    t1: 0.00000000 sec    L:46 C:1
```

Figure 3.3 2 Stack Tracing 2

4. Conclusion

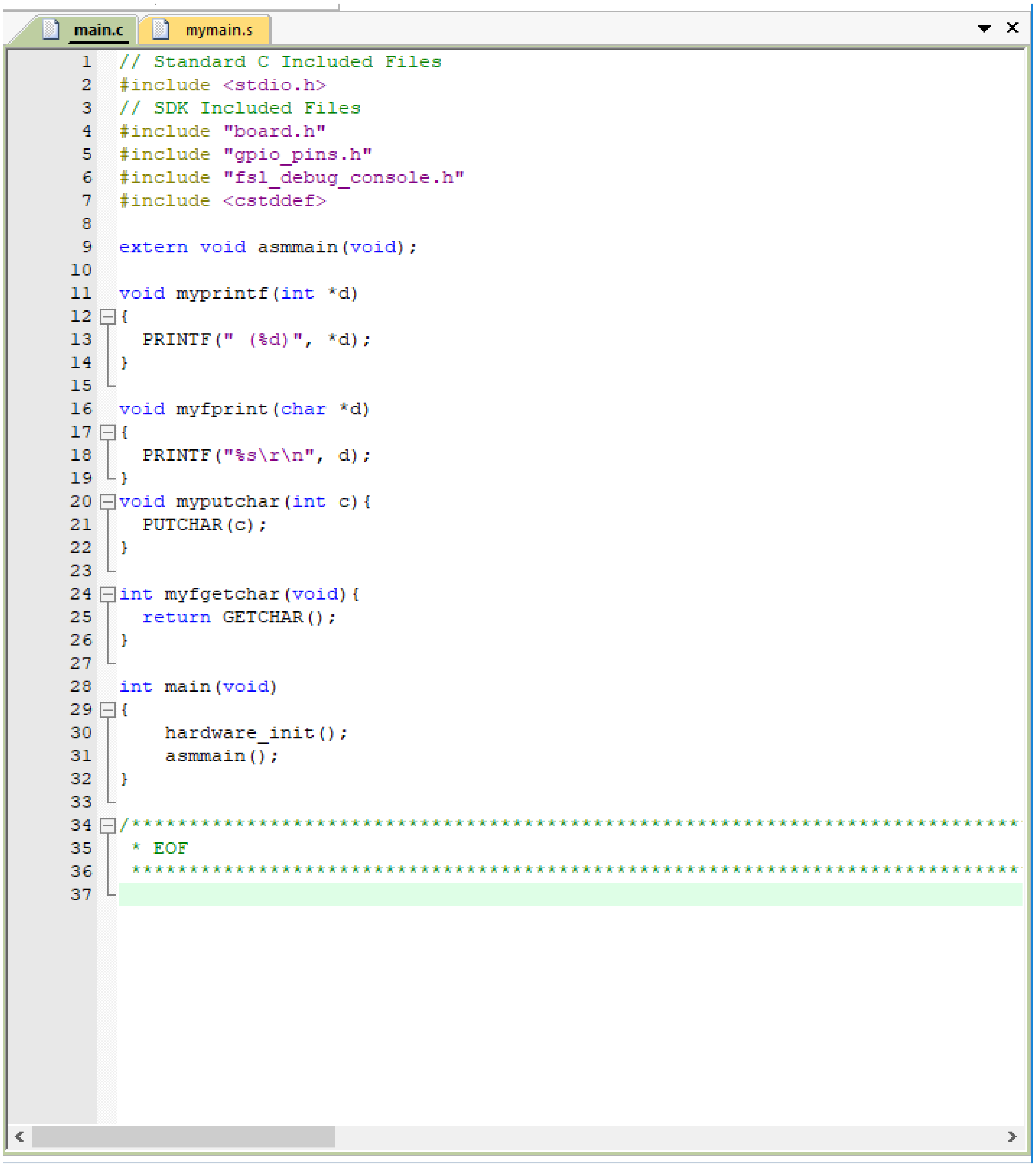
As a Conclusion, the reverse polish notation calculator is successfully implemented, it can receive and properly handle a large diversity of numeric input, including numbers of different lengths, up to 4 digits, as the digit is typed the source code pull each character once at a time and sends it to the board through assembly code. Once a carriage returns is encountered, the characters that are inputted are pulled into hex value and stored into the active stack. As any of the basic arithmetic operations (+, -, *, /) is pressed, the stack pops off two entries and performs the calculation and then shows the results. This code was tested thoroughly to ensure results are true.

References

[1] Radu Muresan, "ENGG3640: Microcomputer Interfacing Laboratory Manual, Version 2", University of Guelph, July 2016.

Appendices

A. Main.c

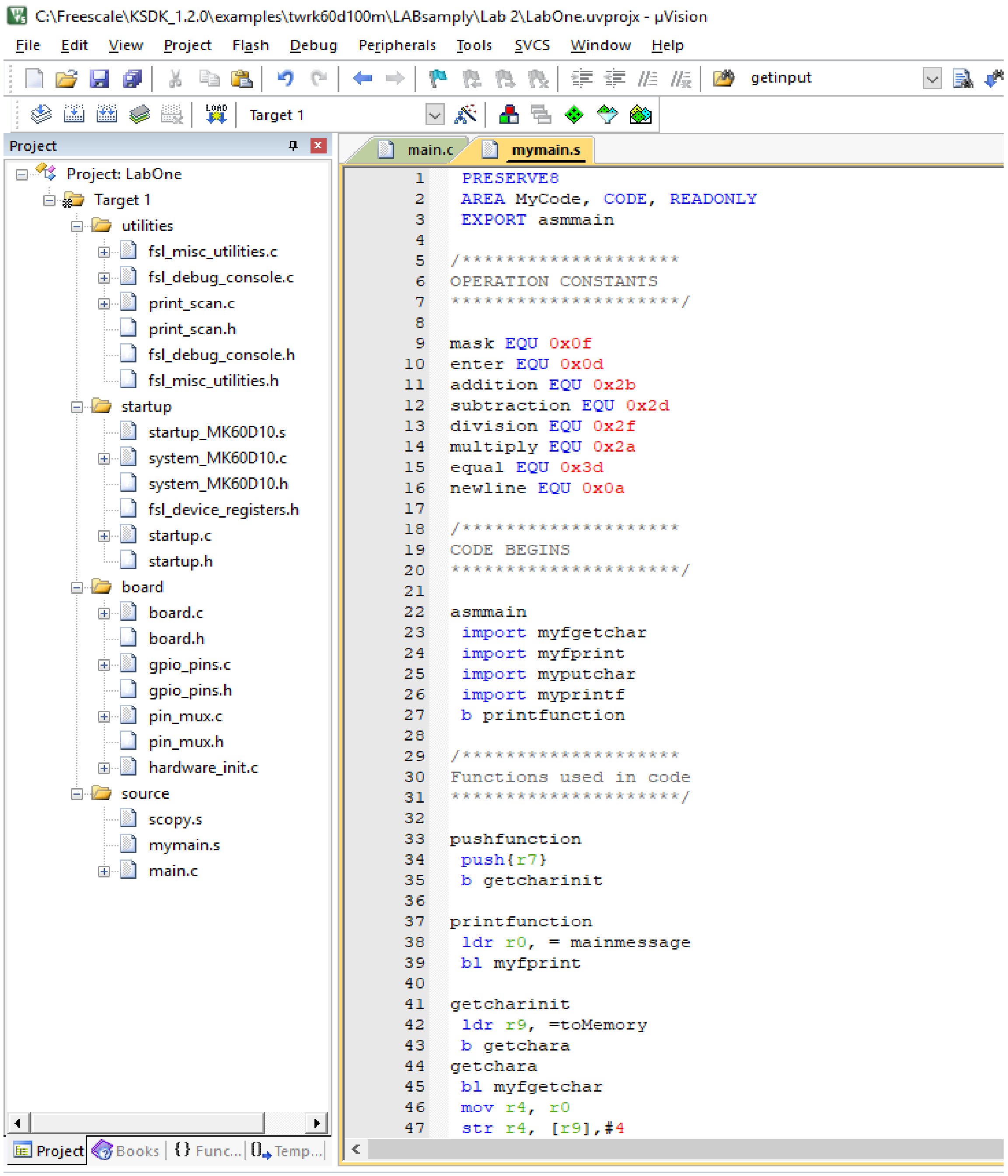


main.c mymain.s

```c
// Standard C Included Files
#include <stdio.h>
// SDK Included Files
#include "board.h"
#include "gpio_pins.h"
#include "fsl_debug_console.h"
#include <cstddef>

extern void asmmain(void);

void myprintf(int *d)
{
  PRINTF("  (%d)", *d);
}

void myfprint(char *d)
{
  PRINTF("%s\r\n", d);
}
void myputchar(int c){
  PUTCHAR(c);
}

int myfgetchar(void){
  return GETCHAR();
}

int main(void)
{
    hardware_init();
    asmmain();
}

/*****************************************************************************
 * EOF
 *****************************************************************************
```

Figure A. 1Main.c

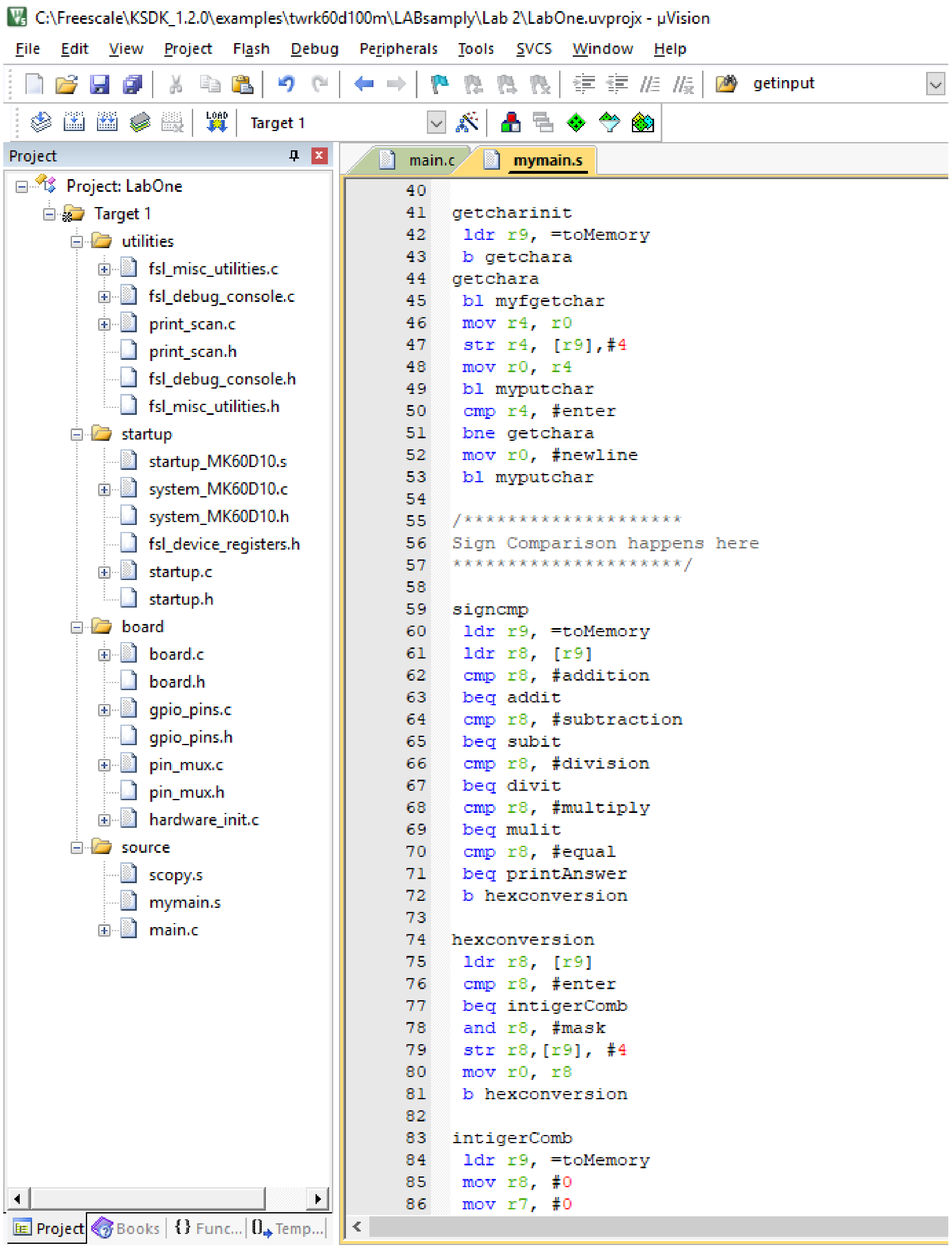
B. myMain.s



C:\Freescale\KSDK_1.2.0\examples\twrk60d100m\LABsamply\Lab 2\LabOne.uvprojx - µVision

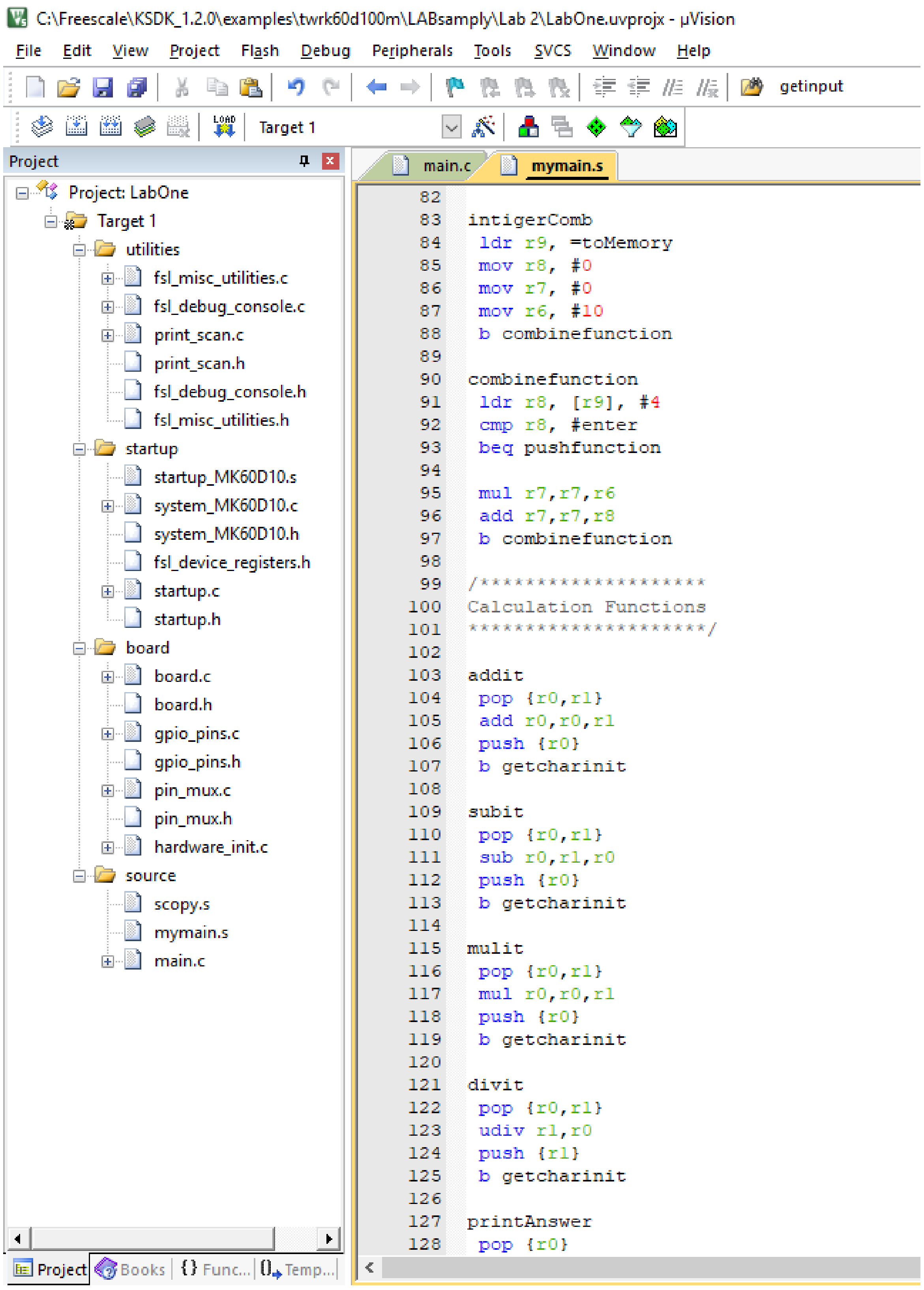
```
 PRESERVE8
 AREA MyCode, CODE, READONLY
 EXPORT asmmain

/********************
OPERATION CONSTANTS
********************/

mask EQU 0x0f
enter EQU 0x0d
addition EQU 0x2b
subtraction EQU 0x2d
division EQU 0x2f
multiply EQU 0x2a
equal EQU 0x3d
newline EQU 0x0a

/********************
CODE BEGINS
********************/

asmmain
 import myfgetchar
 import myfprint
 import myputchar
 import myprintf
 b printfunction

/********************
Functions used in code
********************/

pushfunction
 push{r7}
 b getcharinit

printfunction
 ldr r0, = mainmessage
 bl myfprint

getcharinit
 ldr r9, =toMemory
 b getchara
getchara
 bl myfgetchar
 mov r4, r0
 str r4, [r9],#4
```

Figure B. 1 myMain.s



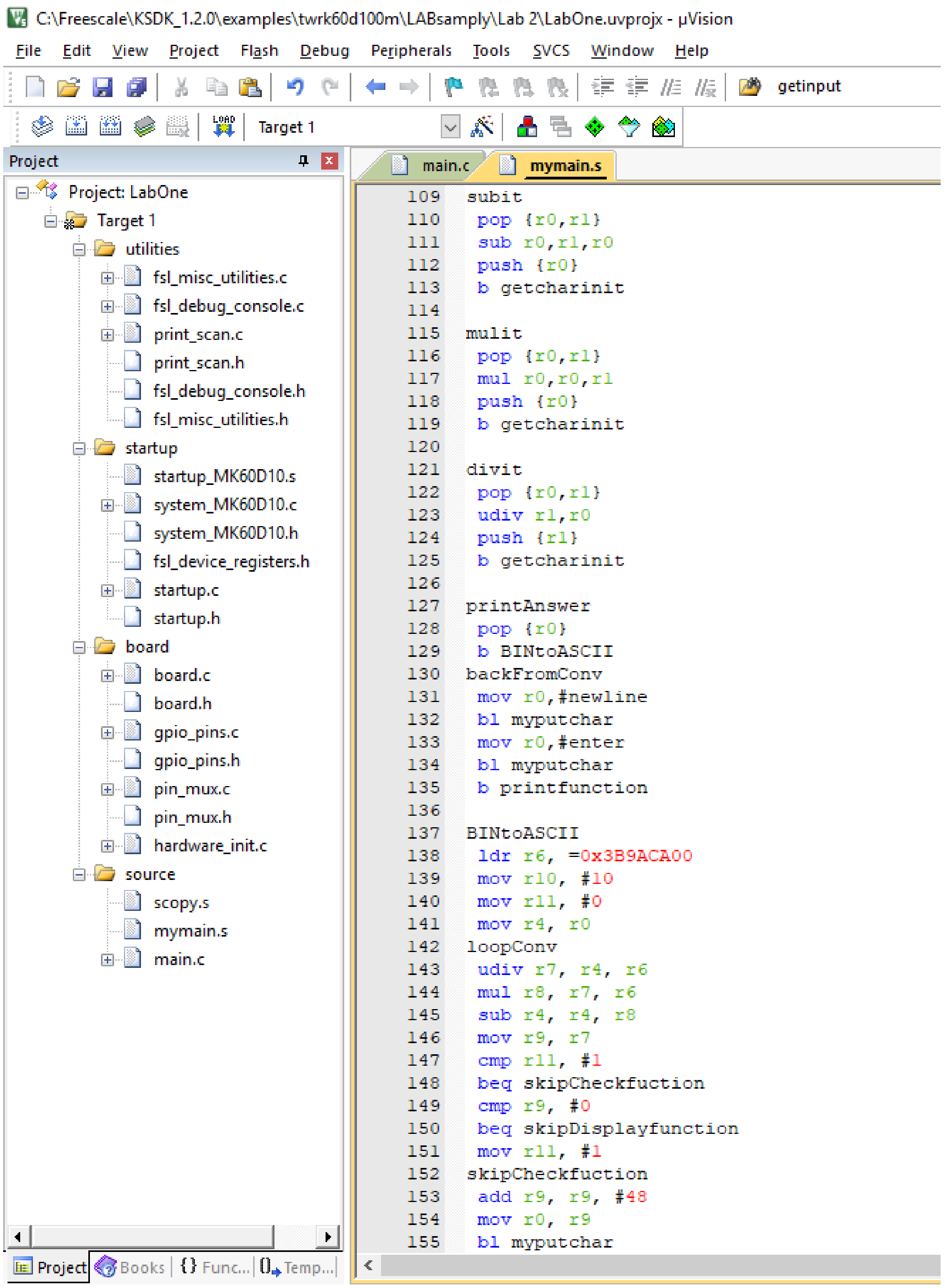
```

getcharinit
 ldr r9, =toMemory
 b getchara
getchara
 bl myfgetchar
 mov r4, r0
 str r4, [r9],#4
 mov r0, r4
 bl myputchar
 cmp r4, #enter
 bne getchara
 mov r0, #newline
 bl myputchar

/*********************
Sign Comparison happens here
*********************/

signcmp
 ldr r9, =toMemory
 ldr r8, [r9]
 cmp r8, #addition
 beq addit
 cmp r8, #subtraction
 beq subit
 cmp r8, #division
 beq divit
 cmp r8, #multiply
 beq mulit
 cmp r8, #equal
 beq printAnswer
 b hexconversion

hexconversion
 ldr r8, [r9]
 cmp r8, #enter
 beq intigerComb
 and r8, #mask
 str r8,[r9], #4
 mov r0, r8
 b hexconversion

intigerComb
 ldr r9, =toMemory
 mov r8, #0
 mov r7, #0
```

Figure B. 2 myMain.s


C:\Freescale\KSDK_1.2.0\examples\twrk60d100m\LABsamply\Lab 2\LabOne.uvprojx - µVision

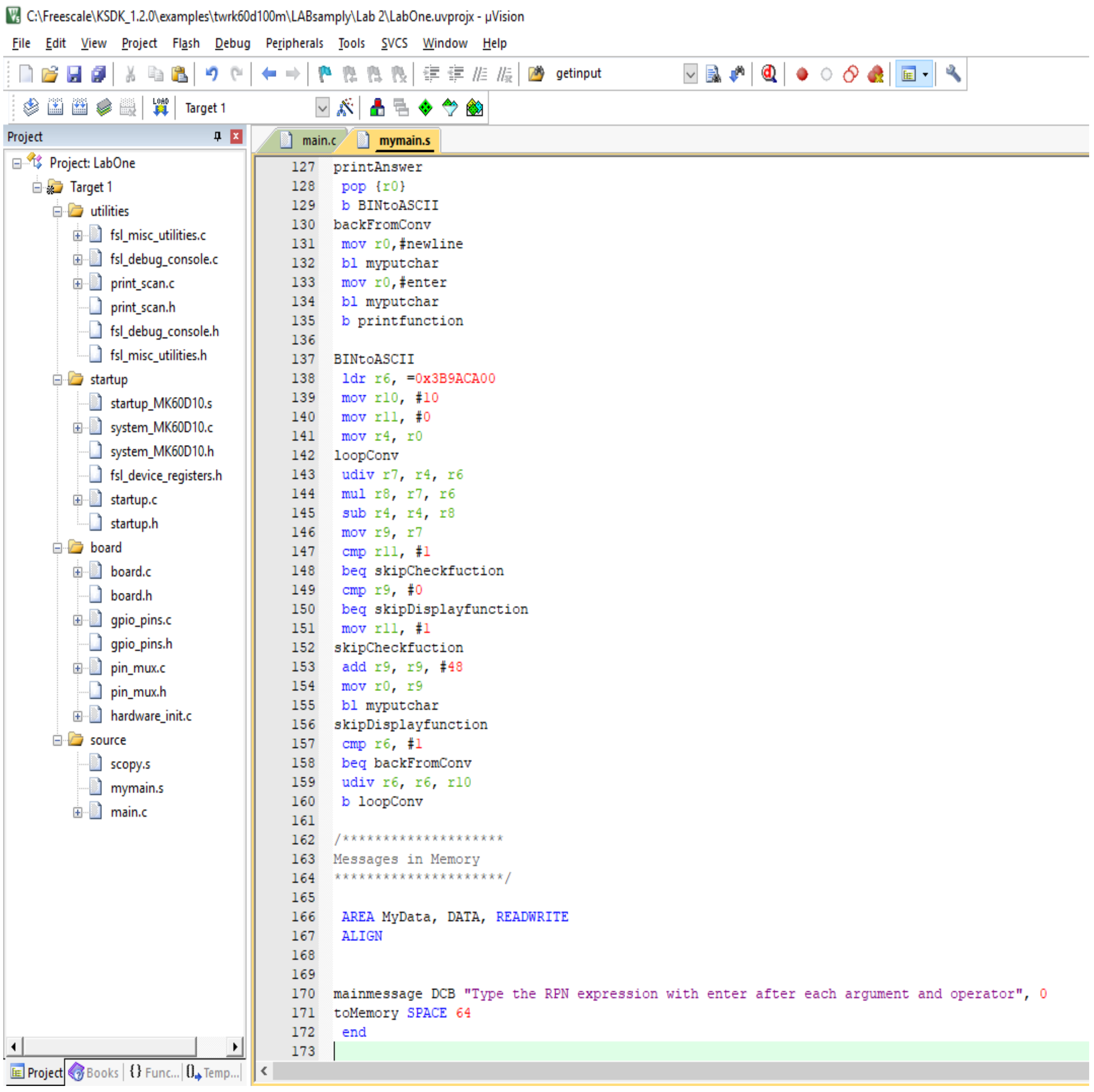
```
intigerComb
 ldr r9, =toMemory
 mov r8, #0
 mov r7, #0
 mov r6, #10
 b combinefunction

combinefunction
 ldr r8, [r9], #4
 cmp r8, #enter
 beq pushfunction

 mul r7,r7,r6
 add r7,r7,r8
 b combinefunction

/********************
Calculation Functions
*********************/

addit
 pop {r0,r1}
 add r0,r0,r1
 push {r0}
 b getcharinit

subit
 pop {r0,r1}
 sub r0,r1,r0
 push {r0}
 b getcharinit

mulit
 pop {r0,r1}
 mul r0,r0,r1
 push {r0}
 b getcharinit

divit
 pop {r0,r1}
 udiv r1,r0
 push {r1}
 b getcharinit

printAnswer
 pop {r0}
```

Figure B. 3 myMain.s



```
subit
 pop {r0,r1}
 sub r0,r1,r0
 push {r0}
 b getcharinit

mulit
 pop {r0,r1}
 mul r0,r0,r1
 push {r0}
 b getcharinit

divit
 pop {r0,r1}
 udiv r1,r0
 push {r1}
 b getcharinit

printAnswer
 pop {r0}
 b BINtoASCII
backFromConv
 mov r0,#newline
 bl myputchar
 mov r0,#enter
 bl myputchar
 b printfunction

BINtoASCII
 ldr r6, =0x3B9ACA00
 mov r10, #10
 mov r11, #0
 mov r4, r0
loopConv
 udiv r7, r4, r6
 mul r8, r7, r6
 sub r4, r4, r8
 mov r9, r7
 cmp r11, #1
 beq skipCheckfuction
 cmp r9, #0
 beq skipDisplayfunction
 mov r11, #1
skipCheckfuction
 add r9, r9, #48
 mov r0, r9
 bl myputchar
```

Figure B. 4 myMain.s



```
printAnswer
 pop {r0}
 b BINtoASCII
backFromConv
 mov r0,#newline
 bl myputchar
 mov r0,#enter
 bl myputchar
 b printfunction

BINtoASCII
 ldr r6, =0x3B9ACA00
 mov r10, #10
 mov r11, #0
 mov r4, r0
loopConv
 udiv r7, r4, r6
 mul r8, r7, r6
 sub r4, r4, r8
 mov r9, r7
 cmp r11, #1
 beq skipCheckfuction
 cmp r9, #0
 beq skipDisplayfunction
 mov r11, #1
skipCheckfuction
 add r9, r9, #48
 mov r0, r9
 bl myputchar
skipDisplayfunction
 cmp r6, #1
 beq backFromConv
 udiv r6, r6, r10
 b loopConv

/********************
Messages in Memory
********************/

 AREA MyData, DATA, READWRITE
 ALIGN


mainmessage DCB "Type the RPN expression with enter after each argument and operator", 0
toMemory SPACE 64
 end

```

Figure B. 5 myMain.s