Lab 2: Introduction to the TM4C123 Microcontroller and the Keil uVision Software Development Tool

Nima Karimian

Preparation

Install and run Keil uVision on your personal computer. The software installation instruction can be downloaded from the *Software* section on Canvas.

Review the content of lecture 3 and 4

Purpose

The general purpose of this lab is to familiarize you with the TM4C123 microcontroller and the software development steps using the uVision simulator. Starting with Lab 3, we will use uVision for both simulation and debugging on the real board, but for this lab, we will use just the simulator. You will learn how to develop and simulate assembly programs using uVision. Through a graphical user interface, the programmer can monitor program execution and the status of the simulated microcontroller including the CPU registers and memory. Software skills you will learn include variable initialization, memory access instructions, arithmetic operations, condition code flags, and branching.

References

- ➢ Cortex-M3/M4F Instruction Set Technical User's Manual. (Canvas -> Reference Materials)
- ➢ Thumb-2 Instruction Set Quick Reference Card (Canvas-> Reference Materials)

Demonstration and Submission

You will have one week to complete the lab. You can discuss with your group members and complete the lab work together. Every group will need to write and submit a lab report to Canvas->Labs->Lab2 report submission. The lab report should include

- The students' names, emails and IDs
- The answers of all the questions asked in the lab handout
- Discussion and suggestions: Through your lab experiments, what have you learned? Do you have any suggestions for future labs or improvement on the learning experiences?

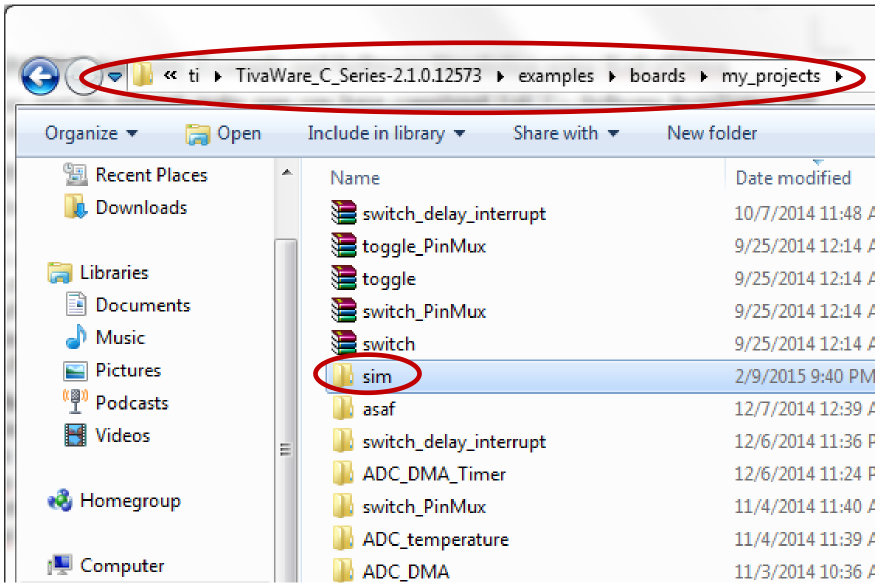
If you finish the lab experiments during the lab time, please demonstrate your results to the instructor. The instructor may ask questions regarding your program. After the demonstration, you can leave. **The latest demonstration time will be the beginning of next lab. The lab report is due at 6pm on the day of your next lab.**

Again, you can work with your group members on all the lab activities, but make sure you understand all the materials.

Procedure

Before you start the tutorial, make sure you have completed *Lab 1 – Software Installation* and installed all the software tools.

1. Download the project *sim.zip* from Canvas->Labs->Lab2->sim. This is a sample project that calculates the sum, difference, and absolute difference of two 32-bit signed numbers NUM1 and NUM2 and then stores the results in memory. It is assumed that you already install the TivaWare library in C:\ti\TivaWare_C_Series-2.x.x.xxxxx. First create a new directory "my_projects" under C:\ti\TivaWare_C_Series-2.x.x.xxxxx\examples\boards and then save and uncompress the sample project *sim.zip* under "C:\ti\TivaWare_C_Series-2.x.x.xxxxx\examples\boards\my_projects" as shown in Figure 1.



Name	Date modified
switch_delay_interrupt	10/7/2014 11:48
toggle_PinMux	9/25/2014 12:14
toggle	9/25/2014 12:14
switch_PinMux	9/25/2014 12:14
switch	9/25/2014 12:14
sim	2/9/2015 9:40 PM
asaf	12/7/2014 12:39
switch_delay_interrupt	12/6/2014 11:36
ADC_DMA_Timer	12/6/2014 11:24
switch_PinMux	11/4/2014 11:40
ADC_temperature	11/4/2014 11:39
ADC_DMA	11/3/2014 10:36

Figure 1

2. Start the Keil µVision IDE by double-clicking the icon on your desktop or by selecting it from the Windows Start Menu. When the IDE loads, it was a blank screen (see Figure 2).

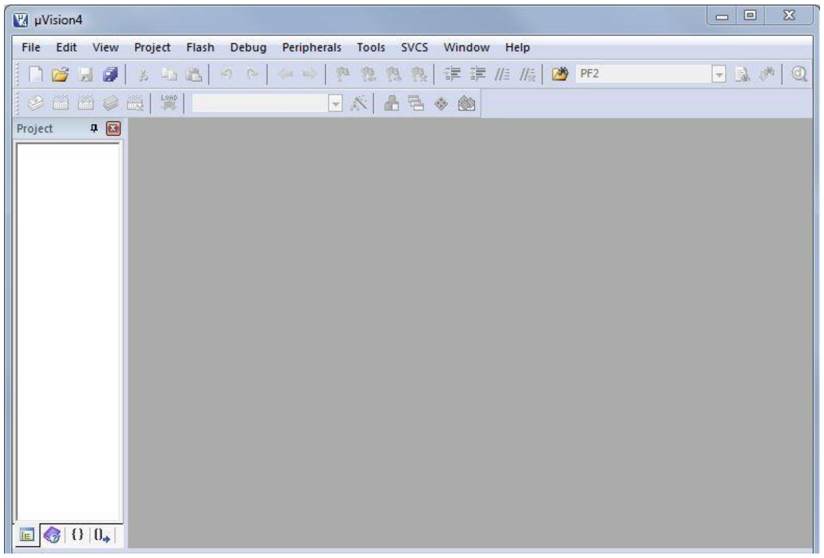


Figure 2

3. Open the project in Keil uVision. From the Project menu, select Open Project… (See Figure 3).

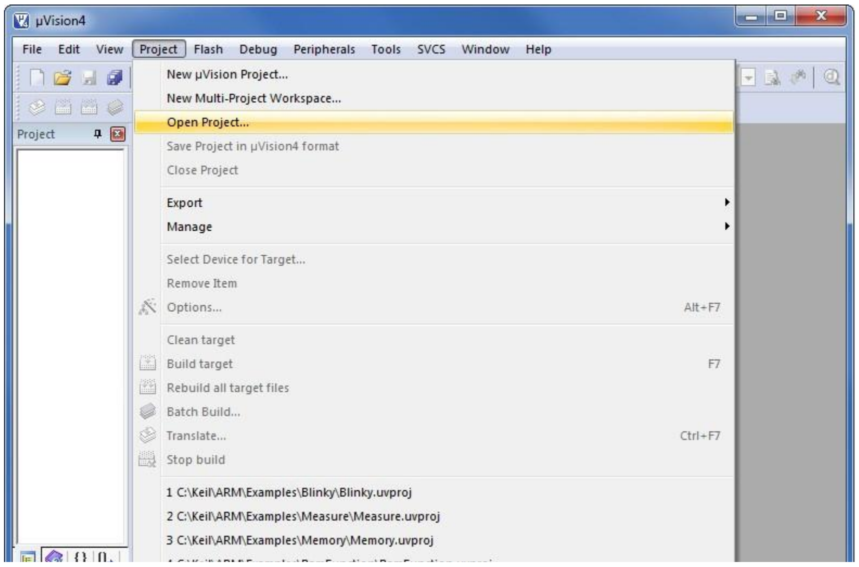
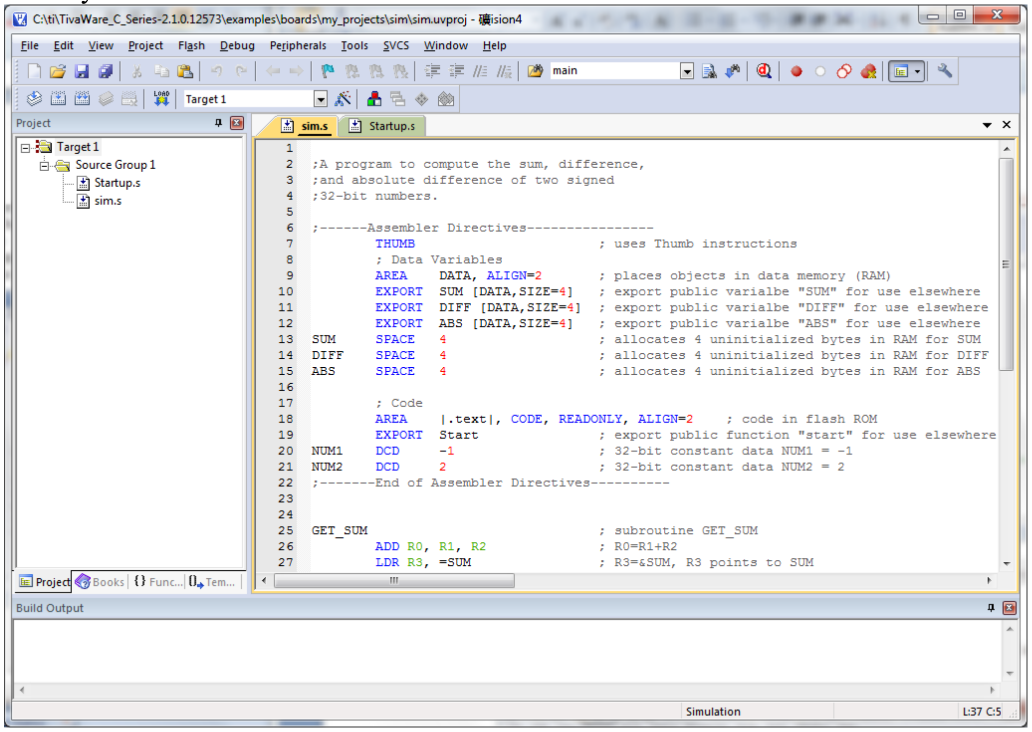


Figure 3

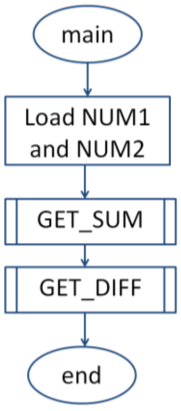
4. Use the dialog box to navigate to the *sim* project. From the location where you installed TivaWare, the *sim* project should be located in: *C:\ti\TivaWare_C_Series-2.1.0.12573\examples\boards\my_projects\sim.* Select the *sim.uvproj* project file and click Open. The project opens in the IDE (see Figure 4).In the *Project* window on the left side, there are two .s files under *Source Group 1*. **Startup.s** contains assembly source code to define the stack, reset vector, and interrupt vectors. All projects in this class will include a **Startup** file, and you do not need to make any changes to the **Startup.s** file for this lab. The **sim.s** file contains the assembly source code for this lab.

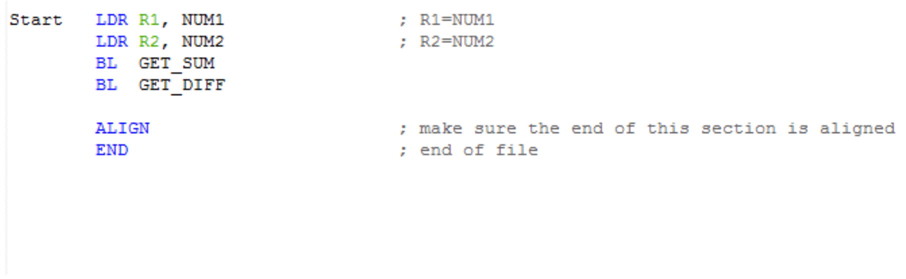


```

;A program to compute the sum, difference,
;and absolute difference of two signed
;32-bit numbers.

;------Assembler Directives----------------
        THUMB                         ; uses Thumb instructions
        ; Data Variables
        AREA    DATA, ALIGN=2         ; places objects in data memory (RAM)
        EXPORT  SUM [DATA,SIZE=4]     ; export public varialbe "SUM" for use elsewhere
        EXPORT  DIFF [DATA,SIZE=4]    ; export public varialbe "DIFF" for use elsewhere
        EXPORT  ABS [DATA,SIZE=4]     ; export public varialbe "ABS" for use elsewhere
SUM     SPACE   4                     ; allocates 4 uninitialized bytes in RAM for SUM
DIFF    SPACE   4                     ; allocates 4 uninitialized bytes in RAM for DIFF
ABS     SPACE   4                     ; allocates 4 uninitialized bytes in RAM for ABS

        ; Code
        AREA    |.text|, CODE, READONLY, ALIGN=2    ; code in flash ROM
        EXPORT  Start                 ; export public function "start" for use elsewhere
NUM1    DCD     -1                    ; 32-bit constant data NUM1 = -1
NUM2    DCD     2                     ; 32-bit constant data NUM2 = 2
;-------End of Assembler Directives----------


GET_SUM                               ; subroutine GET_SUM
        ADD R0, R1, R2                ; R0=R1+R2
        LDR R3, =SUM                  ; R3=&SUM, R3 points to SUM
```

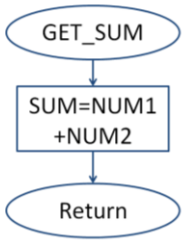
Figure 4

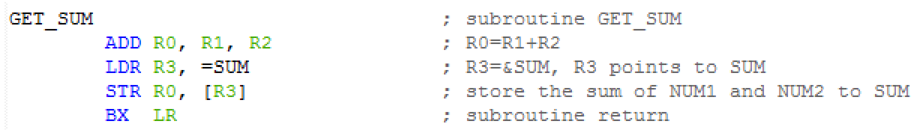
5. Understand the sample project *sim*. The sample project *sim* implements an assembly program that calculates the sum, difference, and absolute difference of two constant 32-bit signed numbers NUM1 and NUM2 and then stores the results in three defined variables SUM, DIFF, and ABS allocated in the RAM. Figure 5 shows the flowchart of the program. The program consists of a main routine and two subroutines GET_SUM and GET_DIFF. GET_SUM calculates the sum of NUM1 and NUM2 and stores the result into the variable SUM. GET_DIFF calculates the difference of NUM1 and NUM2 and stores the result into the variable DIFF. If DIFF is less than 0, the program calculates the absolute different of NUM1 and NUM2 by calculating 0-DIFF, otherwise the absolute difference is equal to DIFF. The absolute difference is then stored in the variable ABS. If you are querying the usage of any instruction, you can search for the instruction (ctrl+F5) in the *Cortex-M3/M4F Instruction Set Technical User's Manual* or the *Thumb-2 Instruction Set Quick Reference Card* (both can be downloaded from Canvas->Reference Materials).



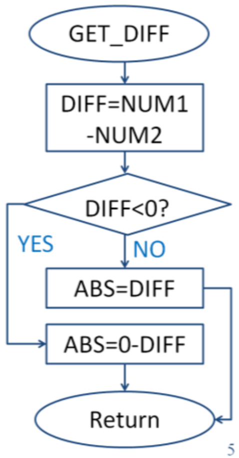


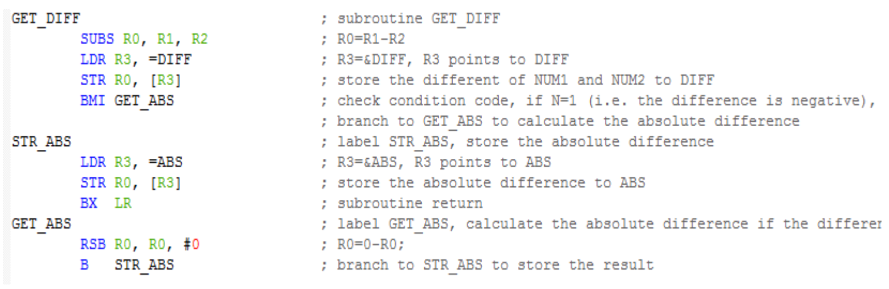
```
Start   LDR R1, NUM1                   ; R1=NUM1
        LDR R2, NUM2                   ; R2=NUM2
        BL  GET_SUM
        BL  GET_DIFF

        ALIGN                          ; make sure the end of this section is aligned
        END                            ; end of file
```





```
GET_SUM                                     ; subroutine GET_SUM
        ADD R0, R1, R2                      ; R0=R1+R2
        LDR R3, =SUM                        ; R3=&SUM, R3 points to SUM
        STR R0, [R3]                        ; store the sum of NUM1 and NUM2 to SUM
        BX  LR                              ; subroutine return
```

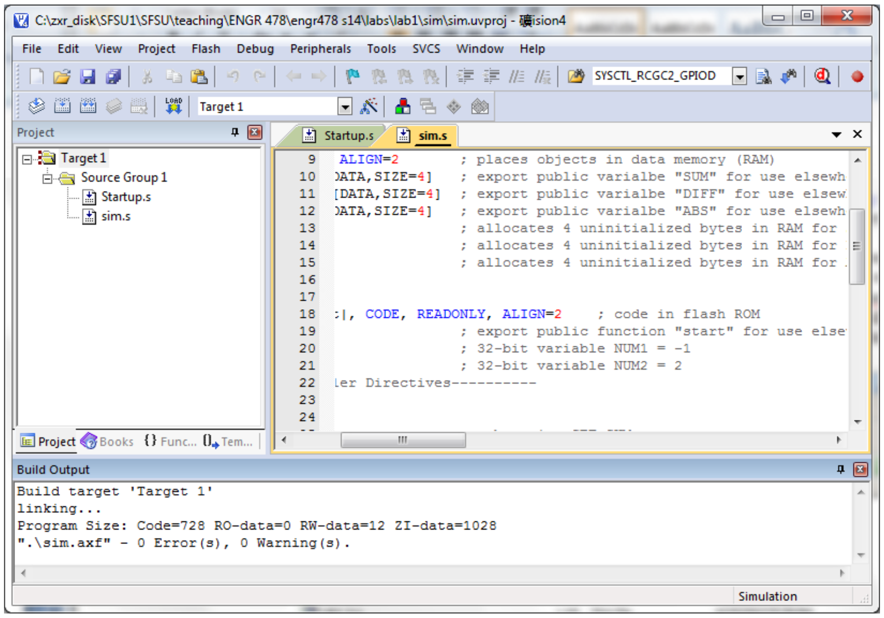




```
GET_DIFF                          ; subroutine GET_DIFF
        SUBS R0, R1, R2           ; R0=R1-R2
        LDR R3, =DIFF             ; R3=&DIFF, R3 points to DIFF
        STR R0, [R3]              ; store the different of NUM1 and NUM2 to DIFF
        BMI GET_ABS               ; check condition code, if N=1 (i.e. the difference is negative),
                                  ; branch to GET_ABS to calculate the absolute difference
STR_ABS                           ; label STR_ABS, store the absolute difference
        LDR R3, =ABS              ; R3=&ABS, R3 points to ABS
        STR R0, [R3]              ; store the absolute difference to ABS
        BX  LR                    ; subroutine return
GET_ABS                           ; label GET_ABS, calculate the absolute difference if the differer
        RSB R0, R0, #0            ; R0=0-R0;
        B   STR_ABS               ; branch to STR_ABS to store the result
```

Figure 5

6. Compile the project by clicking *Project->Build target*. This should show 0 error and 0 warning in the *Build Output* window (see Figure 6).

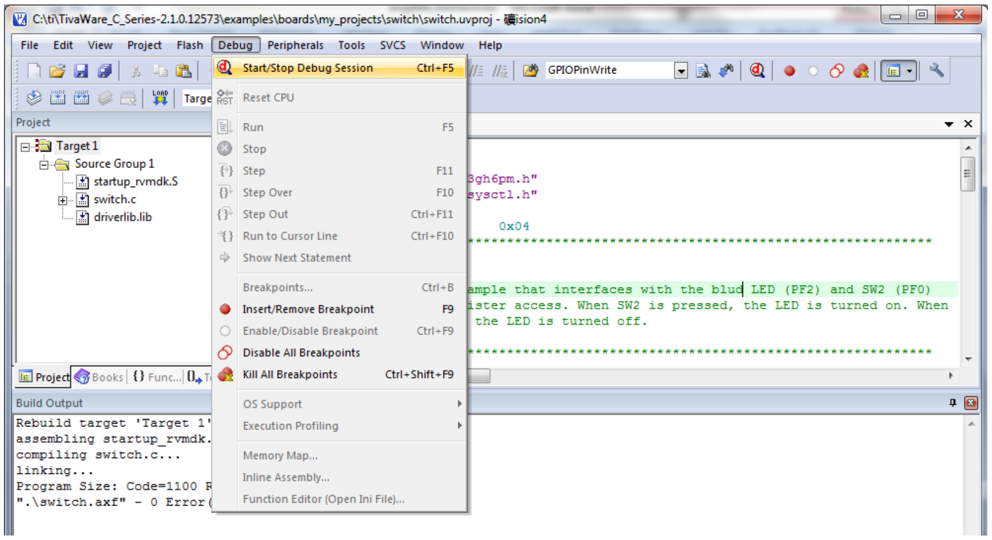

C:\zxr_disk\SFSU1\SFSU\teaching\ENGR 478\engr478 s14\labs\lab1\sim\sim.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
SYSCTL_RCGC2_GPIOD
Target 1
Project
Target 1
Source Group 1
Startup.s
sim.s
Startup.s
sim.s
Project Books Func... Tem...
Build Output
Simulation

```
 ALIGN=2          ; places objects in data memory (RAM)
DATA,SIZE=4]      ; export public varialbe "SUM" for use elsewh
[DATA,SIZE=4]     ; export public varialbe "DIFF" for use elsew
DATA,SIZE=4]      ; export public varialbe "ABS" for use elsewh
                  ; allocates 4 uninitialized bytes in RAM for
                  ; allocates 4 uninitialized bytes in RAM for
                  ; allocates 4 uninitialized bytes in RAM for


:|, CODE, READONLY, ALIGN=2     ; code in flash ROM
                  ; export public function "start" for use else
                  ; 32-bit variable NUM1 = -1
                  ; 32-bit variable NUM2 = 2
ler Directives----------
```

```
Build target 'Target 1'
linking...
Program Size: Code=728 RO-data=0 RW-data=12 ZI-data=1028
".\sim.axf" - 0 Error(s), 0 Warning(s).
```

Figure 6

7. Now you can start to simulate the program. Click *Debug->Start/Stop Debug Session* (see Figure 7), then click OK. The simulation starts. You will see several monitor windows as shown in Figure 8 including the *Registers Window, Disassembly Window, source code window, and Memory 1 Window.*

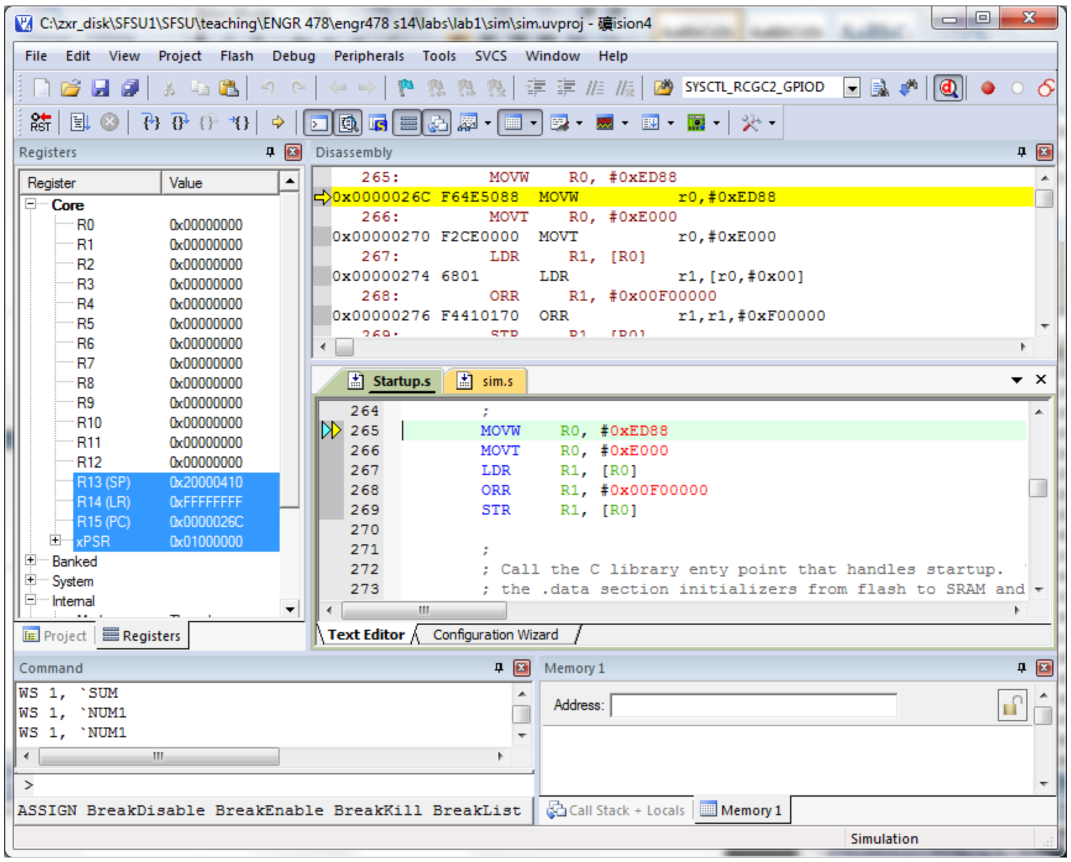

C:\ti\TivaWare_C_Series-2.1.0.12573\examples\boards\my_projects\switch\switch.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Start/Stop Debug Session Ctrl+F5
Reset CPU
Run F5
Stop
Step F11
Step Over F10
Step Out Ctrl+F11
Run to Cursor Line Ctrl+F10
Show Next Statement
Breakpoints... Ctrl+B
Insert/Remove Breakpoint F9
Enable/Disable Breakpoint Ctrl+F9
Disable All Breakpoints
Kill All Breakpoints Ctrl+Shift+F9
OS Support
Execution Profiling
Memory Map...
Inline Assembly...
Function Editor (Open Ini File)...
GPIOPinWrite
Targe
Project
Target 1
Source Group 1
startup_rvmdk.S
switch.c
driverlib.lib
Project Books Func...
Build Output

```
3gh6pm.h"
sysctl.h"

    0x04
*****************************************************************

ample that interfaces with the blud LED (PF2) and SW2 (PF0)
ister access. When SW2 is pressed, the LED is turned on. When
 the LED is turned off.

*****************************************************************
```

```
Rebuild target 'Target 1'
assembling startup_rvmdk.
compiling switch.c...
linking...
Program Size: Code=1100 R
".\switch.axf" - 0 Error(
```

Figure 7

8. In the source code editing window, you will see the symbol ⏩ points to the instruction "MOVW R0, #0xED88" in **Startup.s**. This indicates the first instruction to be executed after the CPU is reset.


C:\zxr_disk\SFSU1\SFSU\teaching\ENGR 478\engr478 s14\labs\lab1\sim\sim.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
SYSCTL_RCGC2_GPIOD
Registers
Disassembly
Startup.s
sim.s
Text Editor
Configuration Wizard
Project
Registers
Command
Memory 1
Address:
Call Stack + Locals
Simulation

Register	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000410
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000026C
xPSR	0x01000000
Banked	
System	
Internal	

```
   265:               MOVW     R0, #0xED88
0x0000026C F64E5088  MOVW              r0,#0xED88
   266:               MOVT     R0, #0xE000
0x00000270 F2CE0000  MOVT              r0,#0xE000
   267:               LDR      R1, [R0]
0x00000274 6801      LDR               r1,[r0,#0x00]
   268:               ORR      R1, #0x00F00000
0x00000276 F4410170  ORR               r1,r1,#0xF00000
```

```
          ;
          MOVW     R0, #0xED88
          MOVT     R0, #0xE000
          LDR      R1, [R0]
          ORR      R1, #0x00F00000
          STR      R1, [R0]

          ;
          ; Call the C library enty point that handles startup.
          ; the .data section initializers from flash to SRAM and
```

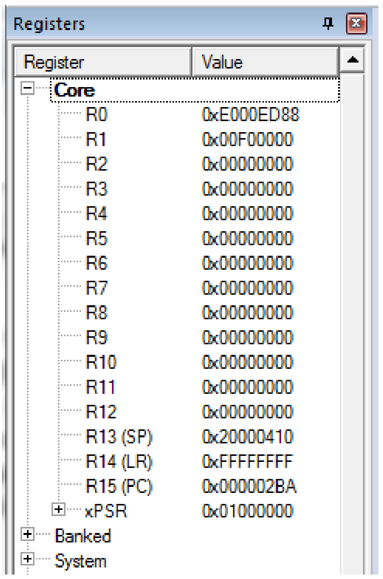
```
WS 1, `SUM
WS 1, `NUM1
WS 1, `NUM1
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList
```

Figure 8

9. The uVision Debugger is powerful. While debugging, developers have full access to the source code, can modify the source code, and can control and analyze program execution on high-level (C/C++) or assembly level. Please refer to *uVision Help* (Help->uVision Help->uVision User's Guide) for more information of the debug menu and commands. In this activity, you will be using the "Step" command to execute the assembly program instruction by instruction, and meanwhile you will closely watch the value changes of CPU registers and memory and record the values of involved registers and memory locations for each step.

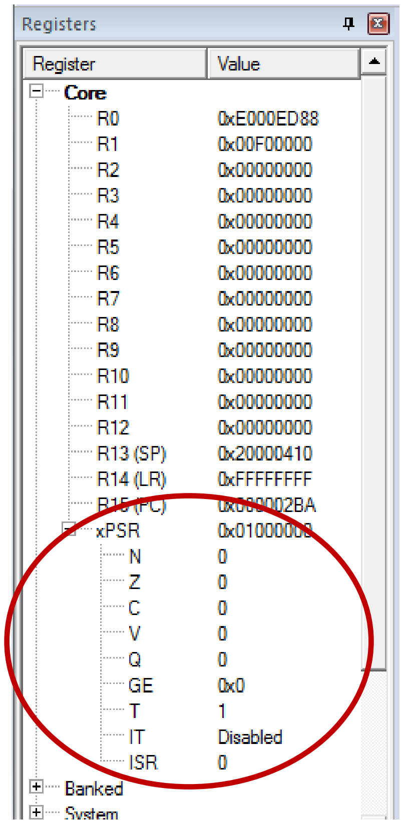
10. **Registers** window (Figure 9). The Registers window shows the content of registers, lists microcontroller operation modes, and system and internal states. You are able to track the contents stored in the general purpose registers (R0-R12) and the special registers (SP, LR, PC, and PSR) when the software program is executed. You can also expand the xPSR register to view the individual bits in the program status register as shown in Figure 10. The condition flags N, Z, C, and V are set by arithmetic and logical instructions and used for conditional code execution.

Condition Code Bits		***Indicates***
N	negative	Result is negative
Z	zero	Result is zero
V	overflow	Signed overflow
C	carry	Unsigned overflow



Register	Value
Core	
R0	0xE000ED88
R1	0x00F00000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000410
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x000002BA
xPSR	0x01000000
Banked	
System	

Figure 9

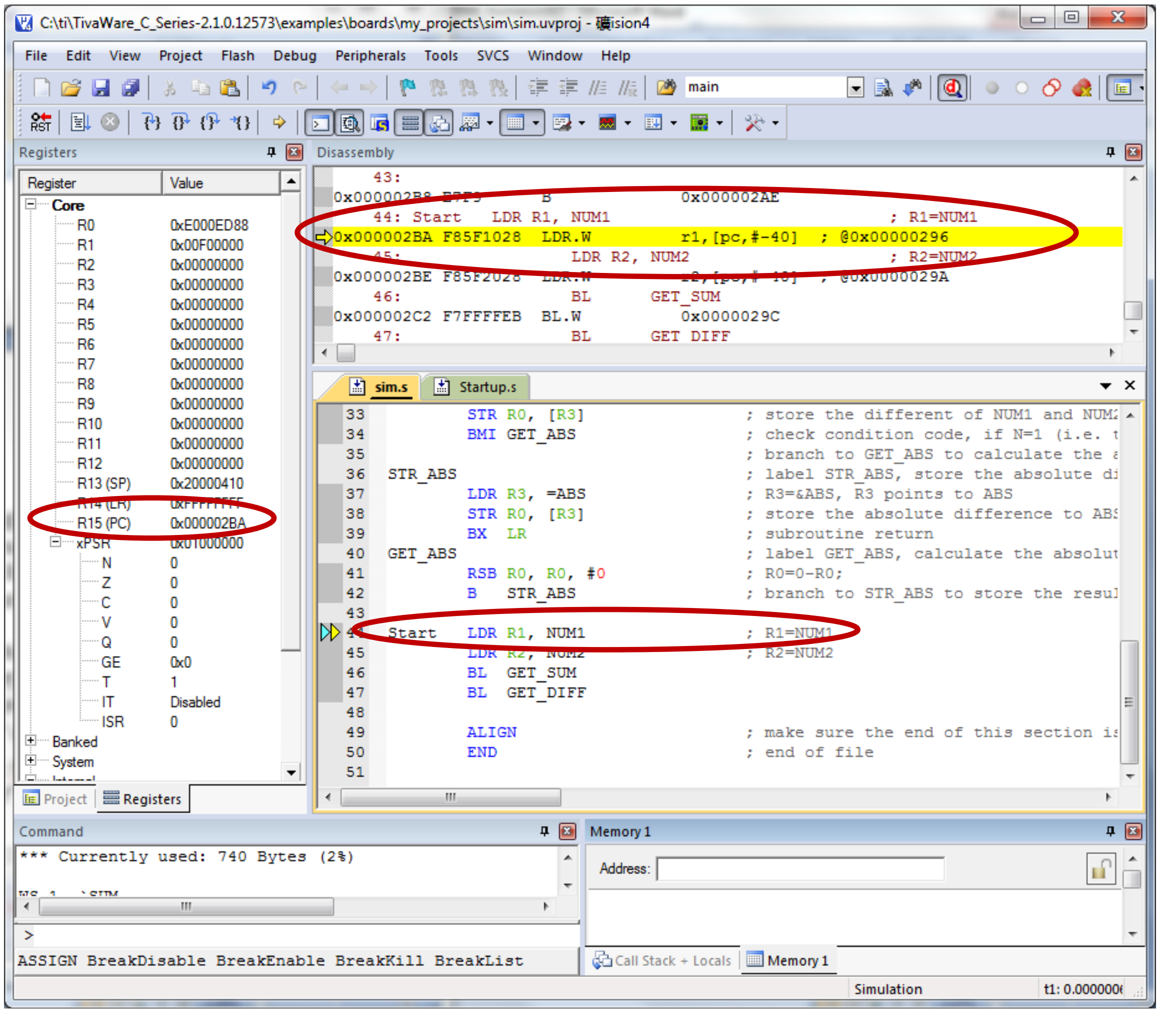


Register	Value
Core	
R0	0xE000ED88
R1	0x00F00000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000410
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x000002BA
xPSR	0x01000000
N	0
Z	0
C	0
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0
Banked	
System	

Figure 10

11. **Disassembly** Window. The Disassembly Window shows the program execution in assembly code. You can see the memory address, the machine code, and the assembly code of each instruction in this window. The currently executing instruction is always highlighted in the window. Figure 11 shows an example in which the program counter (R15 PC) points to the first instruction to be executed in sim.s: “Start   LDR R1, NUM1”. We can see in the Registers Window, the content in PC is 0x0000.02BA, which is the memory address of the instruction to be

executed. We can also see in the Disassembly Window, the line starting with 0x0000.02BA is highlighted, meaning this instruction is to be executed. “F85F1028” is the machine code of the instruction, which is the content stored in address 0x0000.02BA. “LDR.W r1, [pc, #-40]” is another format of “LDR R1, NUM1”, which indicates that the addressing mode of the instruction is PC relative.



Register	Value
Core	
R0	0xE000ED88
R1	0x00F00000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000410
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x000002BA
xPSR	0x01000000
N	0
Z	0
C	0
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0
Banked	
System	

```
    43:
0x000002B8 E7F5      B              0x000002AE
    44: Start    LDR R1, NUM1                        ; R1=NUM1
0x000002BA F85F1028  LDR.W          r1,[pc,#-40]  ; @0x00000296
    45:                  LDR R2, NUM2                ; R2=NUM2
0x000002BE F85F2028  LDR.W          r2,[pc,#-40]  ; @0x0000029A
    46:                  BL      GET_SUM
0x000002C2 F7FFFFEB  BL.W           0x0000029C
    47:                  BL      GET_DIFF
```

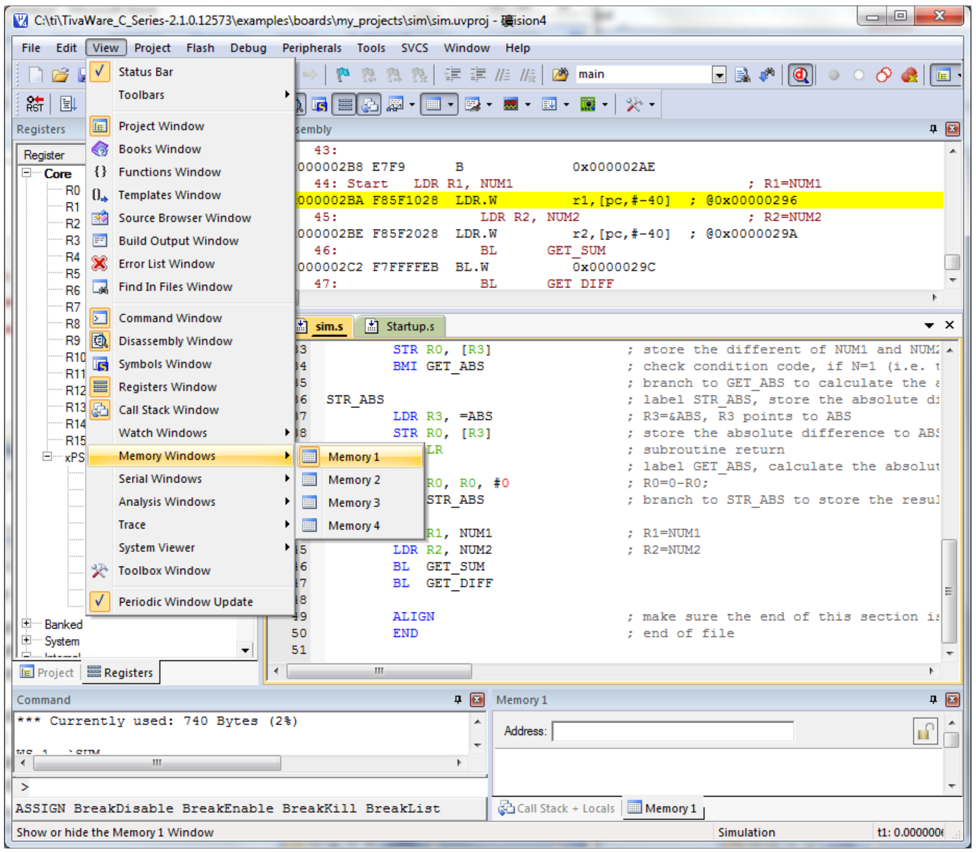
```
          STR R0, [R3]                      ; store the different of NUM1 and NUM2
          BMI GET_ABS                       ; check condition code, if N=1 (i.e. t
                                            ; branch to GET_ABS to calculate the a
STR_ABS                                     ; label STR_ABS, store the absolute di
          LDR R3, =ABS                      ; R3=&ABS, R3 points to ABS
          STR R0, [R3]                      ; store the absolute difference to ABS
          BX  LR                            ; subroutine return
GET_ABS                                     ; label GET_ABS, calculate the absolut
          RSB R0, R0, #0                    ; R0=0-R0;
          B   STR_ABS                       ; branch to STR_ABS to store the resul

Start     LDR R1, NUM1                      ; R1=NUM1
          LDR R2, NUM2                      ; R2=NUM2
          BL  GET_SUM
          BL  GET_DIFF

          ALIGN                             ; make sure the end of this section is
          END                               ; end of file

```

```
*** Currently used: 740 Bytes (2%)
```

```
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList
```

Figure 11

12. **Memory** Windows. The Memory windows show the content of memory areas. You can open a memory window by clicking View -> Memory Windows -> Memory 1 as shown in Figure 12. You can enter a memory address and view the content stored in that address. For example, the address of the 32-bit variable SUM in the sample program is 0x2000.0000 because SUM is the first variable defined in the RAM. If you enter 0x20000000 (don’t forget type 0x) in the Memory 1 Window as shown in Figure 13, you will see the value of SUM stored in 0x20000000-0x20000003 is 00 00 00 00 at the beginning.



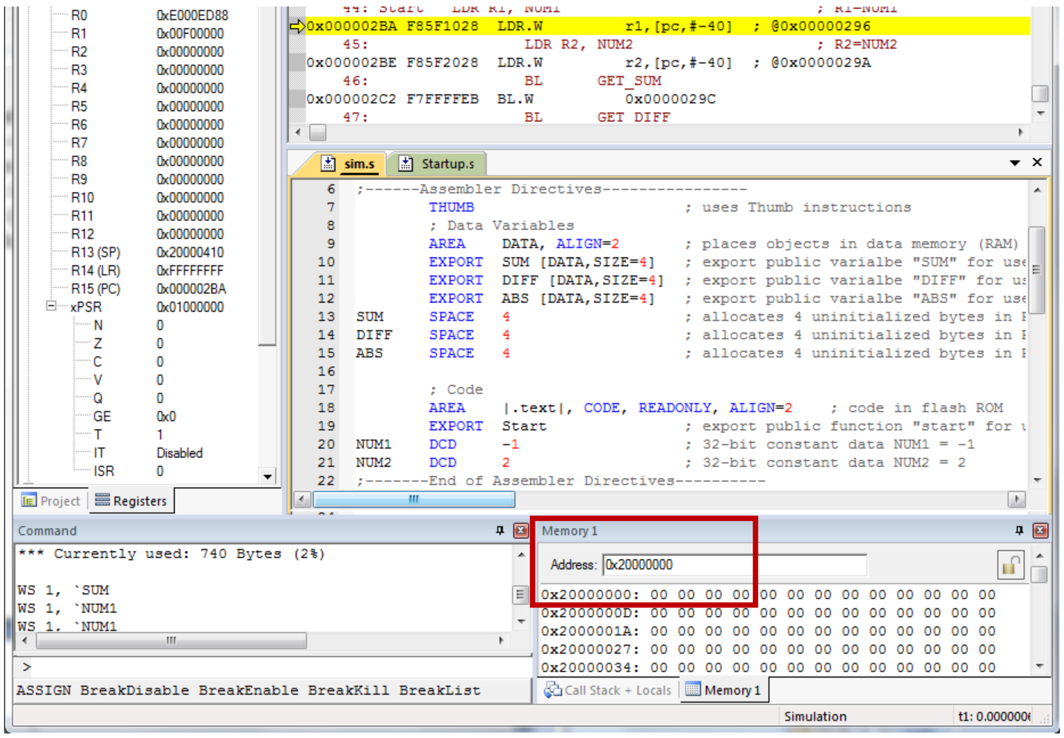
```
    43:
000002B8 E7F9      B         0x000002AE
    44: Start   LDR  R1, NUM1                    ; R1=NUM1
000002BA F85F1028  LDR.W     r1,[pc,#-40]  ; @0x00000296
    45:           LDR R2, NUM2                   ; R2=NUM2
000002BE F85F2028  LDR.W     r2,[pc,#-40]  ; @0x0000029A
    46:           BL    GET_SUM
000002C2 F7FFFFEB  BL.W      0x0000029C
    47:           BL    GET_DIFF
```

```
        STR R0, [R3]        ; store the different of NUM1 and NUM2
        BMI GET_ABS         ; check condition code, if N=1 (i.e. t
                            ; branch to GET_ABS to calculate the a
STR_ABS                     ; label STR_ABS, store the absolute di
        LDR R3, =ABS        ; R3=&ABS, R3 points to ABS
        STR R0, [R3]        ; store the absolute difference to ABS
        LR                  ; subroutine return
                            ; label GET_ABS, calculate the absolut
        R0, R0, #0          ; R0=0-R0;
        STR_ABS             ; branch to STR_ABS to store the resul

        R1, NUM1            ; R1=NUM1
        LDR R2, NUM2        ; R2=NUM2
        BL  GET_SUM
        BL  GET_DIFF

        ALIGN               ; make sure the end of this section is
        END                 ; end of file
```

```
*** Currently used: 740 Bytes (2%)

ASSIGN BreakDisable BreakEnable BreakKill BreakList
```

Show or hide the Memory 1 Window   Simulation   t1: 0.000000

Figure 12



R0	0xE000ED88
R1	0x00F00000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000410
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x000002BA
xPSR	0x01000000
N	0
Z	0
C	0
V	0
Q	0
GE	0x0
T	1
IT	Disabled
ISR	0

```
0x000002BA F85F1028  LDR.W      r1,[pc,#-40]  ; @0x00000296
    45:            LDR R2, NUM2                   ; R2=NUM2
0x000002BE F85F2028  LDR.W      r2,[pc,#-40]  ; @0x0000029A
    46:            BL    GET_SUM
0x000002C2 F7FFFFEB  BL.W       0x0000029C
    47:            BL    GET_DIFF
```

```
;------Assembler Directives----------------
        THUMB                         ; uses Thumb instructions
        ; Data Variables
        AREA    DATA, ALIGN=2         ; places objects in data memory (RAM)
        EXPORT  SUM [DATA,SIZE=4]     ; export public varialbe "SUM" for use
        EXPORT  DIFF [DATA,SIZE=4]    ; export public varialbe "DIFF" for us
        EXPORT  ABS [DATA,SIZE=4]     ; export public varialbe "ABS" for use
SUM     SPACE   4                     ; allocates 4 uninitialized bytes in R
DIFF    SPACE   4                     ; allocates 4 uninitialized bytes in R
ABS     SPACE   4                     ; allocates 4 uninitialized bytes in R

        ; Code
        AREA    |.text|, CODE, READONLY, ALIGN=2    ; code in flash ROM
        EXPORT  Start                 ; export public function "start" for u
NUM1    DCD     -1                    ; 32-bit constant data NUM1 = -1
NUM2    DCD     2                     ; 32-bit constant data NUM2 = 2
;-------End of Assembler Directives----------
```

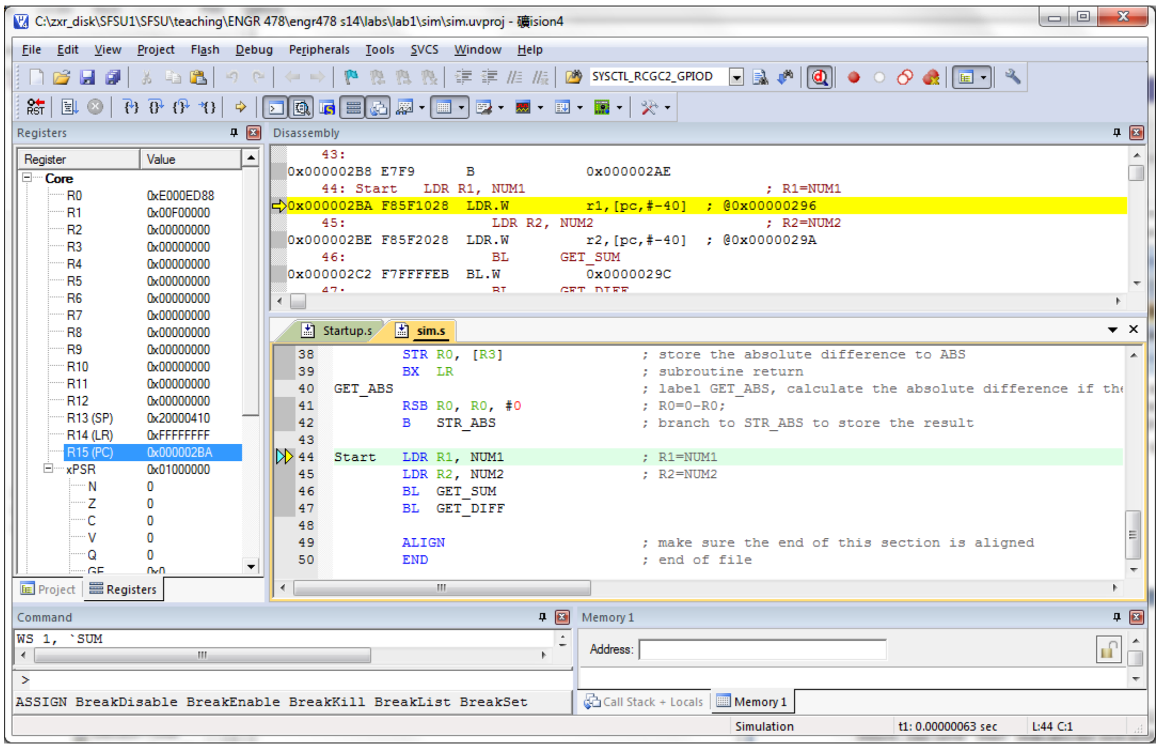
```
*** Currently used: 740 Bytes (2%)

WS 1, `SUM
WS 1, `NUM1
WS 1, `NUM1

ASSIGN BreakDisable BreakEnable BreakKill BreakList
```

```
Address: 0x20000000
0x20000000: 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000000D: 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000001A: 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000027: 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000034: 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Simulation   t1: 0.000000

Figure 13

13. To simulate the program step by step, click *Debug->Step* or the button. Every time you click it, one instruction will be executed, and the registers and memory will be updated accordingly. The program counter R15(PC) (displayed in the *Registers* window on the left panel) always points to the next instruction to be executed. After you run the "Step" command for a few times, you will see the symbol jumps from the instruction "B Start" in **Startup.s** to "Start LDR R1, NUM1" in **sim.s** as shown in the figure below. The label "Start" indicates the first instruction to be executed in the user developed code **sim.s**.


C:\zxr_disk\SFSU1\SFSU\teaching\ENGR 478\engr478 s14\labs\lab1\sim\sim.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
SYSCTL_RCGC2_GPIOD
Registers
Disassembly
Startup.s
sim.s
Project
Registers
Command
Memory 1
Address:
Call Stack + Locals
Simulation
t1: 0.00000063 sec
L:44 C:1

Register	Value
Core	
R0	0xE000ED88
R1	0x00F00000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000410
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x000002BA
xPSR	0x01000000
N	0
Z	0
C	0
V	0
Q	0
GE	0x0

```
    43:
0x000002B8 E7F9      B           0x000002AE
    44: Start   LDR R1, NUM1                        ; R1=NUM1
0x000002BA F85F1028  LDR.W       r1,[pc,#-40]  ; @0x00000296
    45:              LDR R2, NUM2                   ; R2=NUM2
0x000002BE F85F2028  LDR.W       r2,[pc,#-40]  ; @0x0000029A
    46:              BL      GET_SUM
0x000002C2 F7FFFFEB  BL.W        0x0000029C
    47:              BL      GET_DIFF
```

```
        STR R0, [R3]                ; store the absolute difference to ABS
        BX  LR                      ; subroutine return
GET_ABS                             ; label GET_ABS, calculate the absolute difference if the
        RSB R0, R0, #0              ; R0=0-R0;
        B   STR_ABS                 ; branch to STR_ABS to store the result

Start   LDR R1, NUM1                ; R1=NUM1
        LDR R2, NUM2                ; R2=NUM2
        BL  GET_SUM
        BL  GET_DIFF

        ALIGN                       ; make sure the end of this section is aligned
        END                         ; end of file
```

```
WS 1, `SUM
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet
```

6. Run the "step" command instruction by instruction until the end of the program. Record the values all the important registers and memory locations, and then answer the questions below.

Questions

1. Create a table as shown below to record the values of important registers and variables when simulating the program step by step.

Steps	PC	LR	NZCV	R0	R1	R2	R3	SUM	DIFF	ABS
1										
2										
…										

2. What are the addresses of variables **SUM**, **DIFF**, and **ABS**? What values are stored at these addresses when the simulation is completed?

3. In this program, consider the following items including SUM, DIFF, ABS, NUM1, NUM2, and all the instructions, which are stored in RAM and which are stored in ROM?

4. What is the address of instruction "**BL GET_SUM**" in memory? What does this instruction do? After executing this instruction, what are the values stored in PC and LR? Why?

5. What does the instruction "**BMI GET_ABS**" do?

6. After executing "**SUBS R0, R1, R2**", what are the values of the condition code flags? Why?

7. Edit **sim.s**, modify NUM1 and NUM2 to be 5 and 3 respectively and simulate the program again. What are the values stored in SUM, DIFF, and ABS after the simulation?

8. Add a new function to the sample program which compares the values of NUM1 and NUM2 and stores the larger value into a new variable **LARGER**. You need to

a. Define a new 32-bit variable LARGER

b. Design a subroutine **GET_LARGER**, draw the flowchart of the subroutine, and write assembly code to implement this subroutine

c. Compile and simulate your program to validate your code. You must demonstrate your results to me

In your lab report, please include:

- The assembly program listing with detailed comments and the flow chart of the program.
- The execution results of your program.