University of Dhaka

Department of Computer Science and Engineering

CSE 3113: Microprocessor and Assembly Lab

Report of Tasks from Lab 2

Submitted By

Muhaiminul Islam Ninad
Roll: 43

Submitted To:

Dr. Upama Kabir
Professor, Dept. of CSE, University of Dhaka

Dr. Mosarrat Jahan
Associate Professor, Dept. of CSE, University of Dhaka

Mr. Jargis Ahmed,
Lecturer, Dept. of CSE, University of Dhaka

Mr. Palash Roy,
Lecturer, Dept. of CSE, University of Dhaka

Submission Date: April 27, 2025

Contents

1 Task 1: Addition of Two 16-bit Variables

1.1 Code with Comments

The following assembly code adds the contents of the 16-bit variable X to the contents of the 16-bit variable Y and places the result in the 16-bit variable Result:

```
    AREA STACK, NOINIT, READWRITE, ALIGN=3
    SPACE 1024                  ; Reserve 1024 bytes for stack

    AREA |.vectors|, CODE, READONLY
    EXPORT __Vectors
__Vectors
    DCD __stack_top             ; Initial stack pointer
    DCD Reset_Handler           ; Reset handler
    DCD 0                       ; NMI handler (placeholder)
    DCD 0                       ; HardFault handler (placeholder)

    AREA |startup|, CODE, READONLY
__stack_top EQU STACK + 1024
    EXPORT Reset_Handler
Reset_Handler
    BL main                     ; Call main
    B .                         ; Loop forever if main returns

    AREA |.data|, DATA, READWRITE
X       DCW 0x1234              ; Define 16-bit value for X
Y       DCW 0x4321              ; Define 16-bit value for Y
Result  DCW 0x0001              ; Result placeholder

    AREA |.text|, CODE, READONLY
main
    LDR r0, =X                  ; Load address of X
    LDRH r1, [r0]               ; Load value of X into r1

    LDR r0, =Y                  ; Load address of Y
    LDRH r2, [r0]               ; Load value of Y into r2

    ADD r3, r1, r2              ; Add X and Y, store result in r3

    LDR r0, =Result             ; Load address of Result
    STRH r3, [r0]               ; Store result in memory
STOP
    B STOP                      ; Infinite loop
    END
```

1.2 Code Explanation

The assembly code can be broken down into several key components:

1.2.1 Memory and Stack Setup

- `AREA STACK` directive allocates 1024 bytes for the stack
- Vector table defines essential handlers including the reset handler

1.2.2 Data Section

- Variables X and Y are defined as 16-bit values (0x1234 and 0x4321)
- Result is initialized with a placeholder value

1.2.3 Main Logic

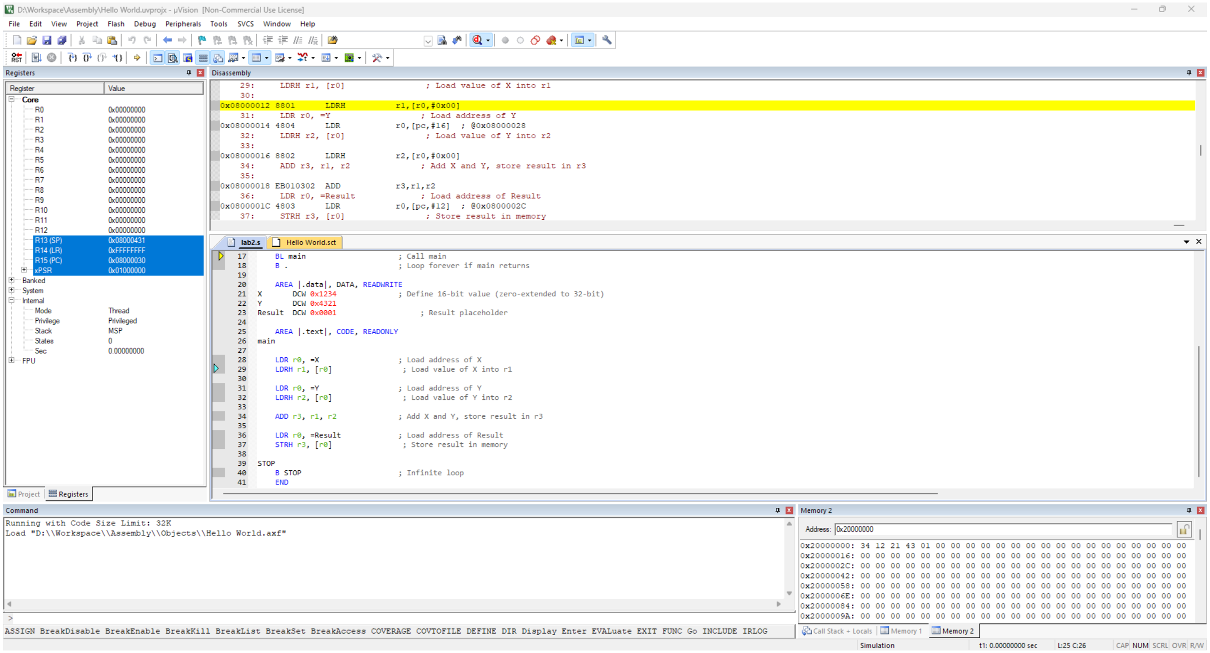
- The address of X is loaded into register r0
- The 16-bit value of X is loaded into register r1 using LDRH (Load Register Halfword)
- The address of Y is loaded into register r0
- The 16-bit value of Y is loaded into register r2 using LDRH
- The ADD instruction adds r1 and r2, storing the result in r3
- The address of Result is loaded into register r0
- The STRH (Store Register Halfword) instruction stores the 16-bit value in r3 to the Result variable

1.2.4 Expected Results

Adding 0x1234 (4660 in decimal) to 0x4321 (17185 in decimal) yields 0x5555 (21845 in decimal), which will be stored in the Result variable.

1.3 System State Screenshots

1.3.1 System State After Code Loading



D:\Workspace\Assembly\Hello World.uvprojx - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
lab2.s
Hello World.sct
Command
Memory 2
Project
Registers
Call Stack + Locals
Memory 1
Memory 2

```
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)      0x08000431
  R14 (LR)      0xFFFFFFFF
  R15 (PC)      0x08000030
  xPSR          0x01000000
Banked
System
Internal
  Mode          Thread
  Privilege     Privileged
  Stack         MSP
  States        0
  Sec           0.00000000
FPU
```

```
    29:     LDRH r1, [r0]                ; Load value of X into r1
    30:
0x08000012 8801      LDRH          r1,[r0,#0x00]
    31:     LDR r0, =Y                  ; Load address of Y
0x08000014 4804      LDR           r0,[pc,#16]  ; @0x08000028
    32:     LDRH r2, [r0]                ; Load value of Y into r2
    33:
0x08000016 8802      LDRH          r2,[r0,#0x00]
    34:     ADD r3, r1, r2              ; Add X and Y, store result in r3
    35:
0x08000018 EB010302  ADD           r3,r1,r2
    36:     LDR r0, =Result             ; Load address of Result
0x0800001C 4803      LDR           r0,[pc,#12]  ; @0x0800002C
    37:     STRH r3, [r0]                ; Store result in memory
```

```
    BL main                     ; Call main
    B .                         ; Loop forever if main returns

    AREA |.data|, DATA, READWRITE
X       DCW 0x1234              ; Define 16-bit value (zero-extended to 32-bit)
Y       DCW 0x4321
Result  DCW 0x0001                  ; Result placeholder

    AREA |.text|, CODE, READONLY
main

    LDR r0, =X                  ; Load address of X
    LDRH r1, [r0]                ; Load value of X into r1

    LDR r0, =Y                  ; Load address of Y
    LDRH r2, [r0]                ; Load value of Y into r2

    ADD r3, r1, r2              ; Add X and Y, store result in r3

    LDR r0, =Result             ; Load address of Result
    STRH r3, [r0]                ; Store result in memory

STOP
    B STOP                      ; Infinite loop
    END
```

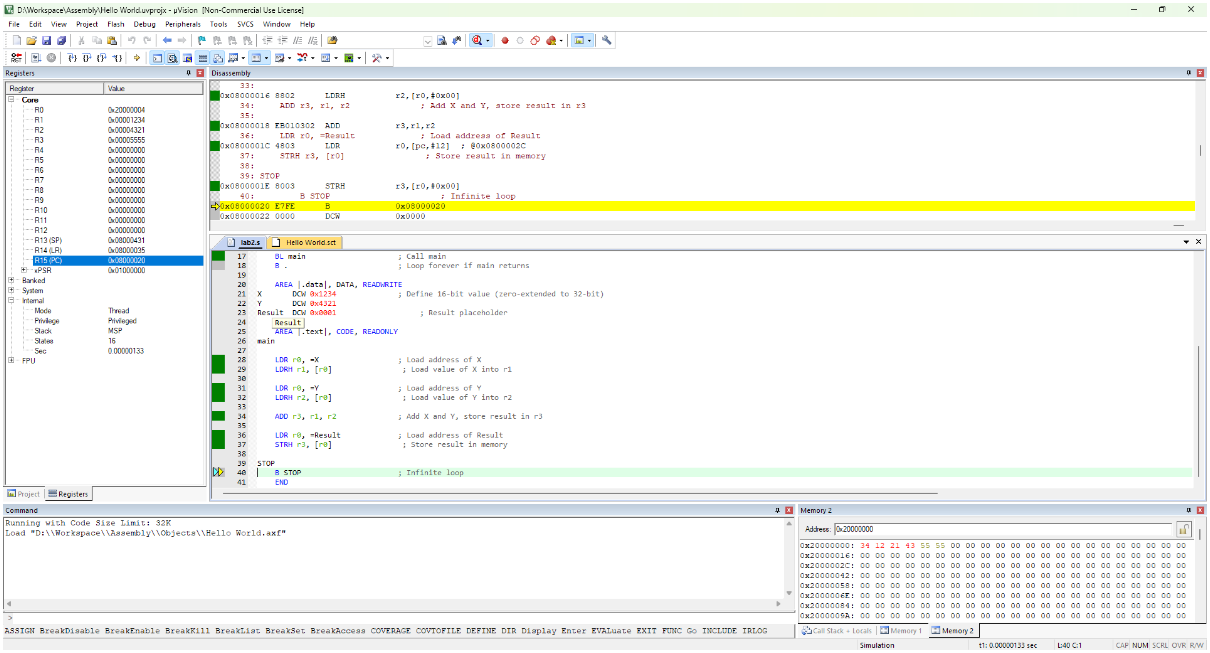
```
Running with Code Size Limit: 32K
Load "D:\\Workspace\\Assembly\\Objects\\Hello World.axf"
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE DEFINE DIR Display Enter EVALuate EXIT FUNC Go INCLUDE IRLOG
```

```
Address: 0x20000000
0x20000000: 34 12 21 43 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000002C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000042: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000058: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000006E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000084: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000009A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Simulation    t1: 0.00000000 sec    L:25 C:26    CAP NUM SCRL OVR R/W
```

Figure 1: Memory contents after loading the assembly code

1.3.2 System State After Code Execution



D:\Workspace\Assembly\Hello World.uvprojx - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
lab2.s
Hello World.sct
Result
Command
Memory 2
Project
Registers
Call Stack + Locals
Memory 1
Memory 2

```
Register        Value
Core
  R0            0x20000004
  R1            0x00001234
  R2            0x00004321
  R3            0x00005555
  R4            0x00000000
  R5            0x00000000
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x08000431
  R14 (LR)      0x08000035
  R15 (PC)      0x08000020
  xPSR          0x01000000
Banked
System
Internal
  Mode          Thread
  Privilege     Privileged
  Stack         MSP
  States        16
  Sec           0.00000133
FPU
```

```
    33:
0x08000016 8802      LDRH          r2,[r0,#0x00]
    34:     ADD r3, r1, r2              ; Add X and Y, store result in r3
    35:
0x08000018 EB010302  ADD           r3,r1,r2
    36:     LDR r0, =Result             ; Load address of Result
0x0800001C 4803      LDR           r0,[pc,#12]  ; @0x0800002C
    37:     STRH r3, [r0]                ; Store result in memory
    38:
    39: STOP
0x0800001E 8003      STRH          r3,[r0,#0x00]
    40:         B STOP                     ; Infinite loop
0x08000020 E7FE      B             0x08000020
0x08000022 0000      DCW           0x0000
```

```
    BL main                     ; Call main
    B .                         ; Loop forever if main returns

    AREA |.data|, DATA, READWRITE
X       DCW 0x1234              ; Define 16-bit value (zero-extended to 32-bit)
Y       DCW 0x4321
Result  DCW 0x0001                  ; Result placeholder

    AREA |.text|, CODE, READONLY
main

    LDR r0, =X                  ; Load address of X
    LDRH r1, [r0]                ; Load value of X into r1

    LDR r0, =Y                  ; Load address of Y
    LDRH r2, [r0]                ; Load value of Y into r2

    ADD r3, r1, r2              ; Add X and Y, store result in r3

    LDR r0, =Result             ; Load address of Result
    STRH r3, [r0]                ; Store result in memory

STOP
    B STOP                      ; Infinite loop
    END
```

```
Running with Code Size Limit: 32K
Load "D:\\Workspace\\Assembly\\Objects\\Hello World.axf"
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE DEFINE DIR Display Enter EVALuate EXIT FUNC Go INCLUDE IRLOG
```

```
Address: 0x20000000
0x20000000: 34 12 21 43 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000002C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000042: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000058: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000006E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000084: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000009A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Simulation    t1: 0.00000133 sec    L:40 C:1    CAP NUM SCRL OVR R/W
```

Figure 2: Memory contents after executing the assembly code

2 Task 2: Arithmetic Operations on Two Variables

2.1 Code with Comments

The following assembly code performs addition, subtraction, and multiplication on two 16-bit variables X and Y:

```
    AREA STACK, NOINIT, READWRITE, ALIGN=3
    SPACE 1024                  ; Reserve 1024 bytes for stack

    AREA |.vectors|, CODE, READONLY
    EXPORT __Vectors
__Vectors
    DCD __stack_top             ; Initial stack pointer
    DCD Reset_Handler           ; Reset handler
    DCD 0                       ; NMI handler (placeholder)
    DCD 0                       ; HardFault handler (placeholder)

    AREA |startup|, CODE, READONLY
__stack_top EQU STACK + 1024

    EXPORT Reset_Handler
Reset_Handler
    BL main                     ; Call main
    B .                         ; Loop forever if main returns

    AREA |.data|, DATA, READWRITE
X           DCW 0x4321          ; Define 16-bit value
Y           DCW 0x1234          ; Define another 16-bit value
Result_Add  DCW 0x0000          ; Placeholder for Addition result
Result_Sub  DCW 0x0000          ; Placeholder for Subtraction result
Result_Mul  DCD 0x0000          ; Placeholder for Multiplication result

    AREA |.text|, CODE, READONLY
main
    ; Load X into r1
    LDR r0, =X
    LDRH r1, [r0]

    ; Load Y into r2
    LDR r0, =Y
    LDRH r2, [r0]

    ; -------- Addition --------
    ADD r3, r1, r2              ; r3 = r1 + r2
    LDR r0, =Result_Add
    STRH r3, [r0]

    ; -------- Subtraction --------
    SUB r3, r1, r2              ; r3 = r1 - r2
    LDR r0, =Result_Sub
    STRH r3, [r0]

    ; -------- Multiplication --------
    MUL r3, r1, r2              ; r3 = r1 * r2
    LDR r0, =Result_Mul
    STR r3, [r0]                ; Store full 32 bits (since
   multiplication can overflow 16 bits)
```

```


STOP
    B STOP                      ; Infinite loop
    END
```

2.2 Code Explanation

This code extends the previous example to include three arithmetic operations:

2.2.1 Data Section

- X is defined as 0x4321 (17185 in decimal)
- Y is defined as 0x1234 (4660 in decimal)
- Three result variables are defined for different operations

2.2.2 Main Logic

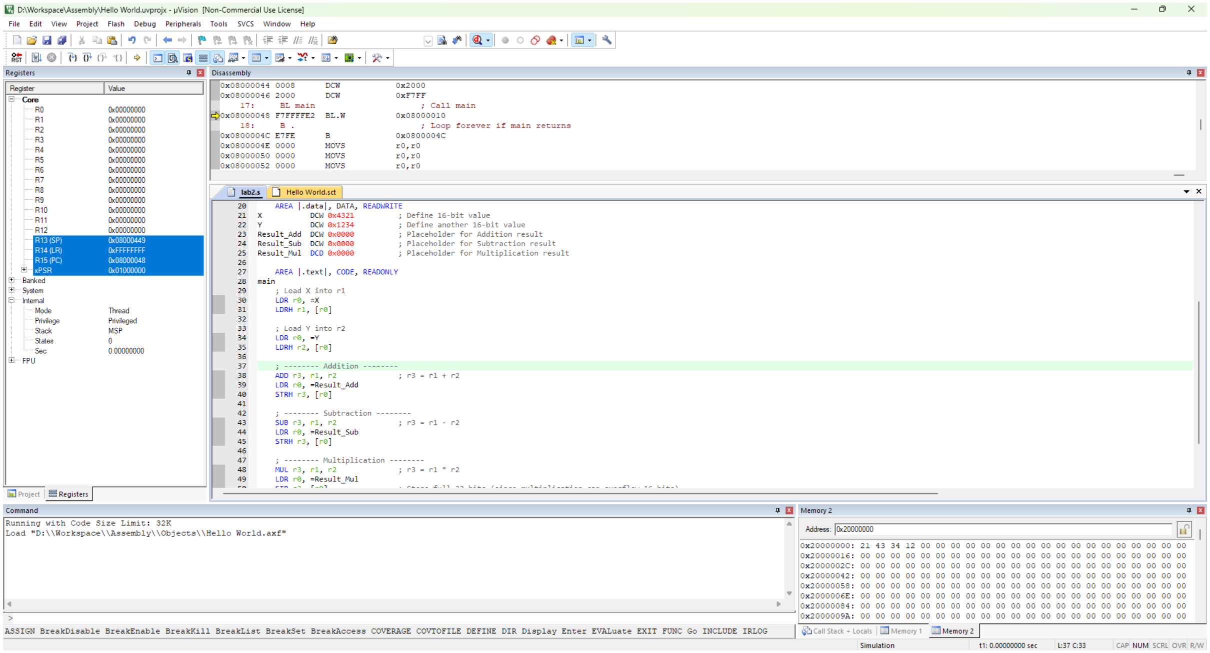
- Values of X and Y are loaded into registers r1 and r2 respectively
- Addition: r1 + r2 stored in r3, then stored in the Addition variable
- Subtraction: r1 - r2 stored in r3, then stored in the Subtraction variable
- Multiplication: r1 * r2 stored in r3, then stored in the Multiplication variable

2.2.3 Expected Results

- Addition: 0x4321 + 0x1234 = 0x5555 (17185 + 4660 = 21845)
- Subtraction: 0x4321 - 0x1234 = 0x30ED (17185 - 4660 = 12525)
- Multiplication: 0x4321 * 0x1234 = 0x4ECA564 (17185 * 4660 = 80,082,100)
  - Note: The multiplication result requires 32 bits to store properly, which is why we use STR instead of STRH for the result.

2.3 System State Screenshots

2.3.1 System State After Code Loading


D:\Workspace\Assembly\Hello World.uvprojx - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
lab2.s
Hello World.sct
Command
Memory 2
Project
Registers
Call Stack + Locals
Memory 1
Memory 2

```
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)    0x08000449
  R14 (LR)    0xFFFFFFFF
  R15 (PC)    0x08000048
  xPSR        0x01000000
Banked
System
Internal
  Mode        Thread
  Privilege   Privileged
  Stack       MSP
  States      0
  Sec         0.00000000
FPU
```

```
0x08000044 0008       DCW          0x2000
0x08000046 2000       DCW          0xF7FF
    17:      BL main                    ; Call main
0x08000048 F7FFFFE2   BL.W         0x08000010
    18:      B .                        ; Loop forever if main returns
0x0800004C E7FE       B            0x0800004C
0x0800004E 0000       MOVS         r0,r0
0x08000050 0000       MOVS         r0,r0
0x08000052 0000       MOVS         r0,r0
```

```
    AREA |.data|, DATA, READWRITE
X           DCW 0x4321          ; Define 16-bit value
Y           DCW 0x1234          ; Define another 16-bit value
Result_Add  DCW 0x0000          ; Placeholder for Addition result
Result_Sub  DCW 0x0000          ; Placeholder for Subtraction result
Result_Mul  DCD 0x0000          ; Placeholder for Multiplication result

    AREA |.text|, CODE, READONLY
main
    ; Load X into r1
    LDR r0, =X
    LDRH r1, [r0]

    ; Load Y into r2
    LDR r0, =Y
    LDRH r2, [r0]

    ; -------- Addition --------
    ADD r3, r1, r2              ; r3 = r1 + r2
    LDR r0, =Result_Add
    STRH r3, [r0]

    ; -------- Subtraction --------
    SUB r3, r1, r2              ; r3 = r1 - r2
    LDR r0, =Result_Sub
    STRH r3, [r0]

    ; -------- Multiplication --------
    MUL r3, r1, r2              ; r3 = r1 * r2
    LDR r0, =Result_Mul
```

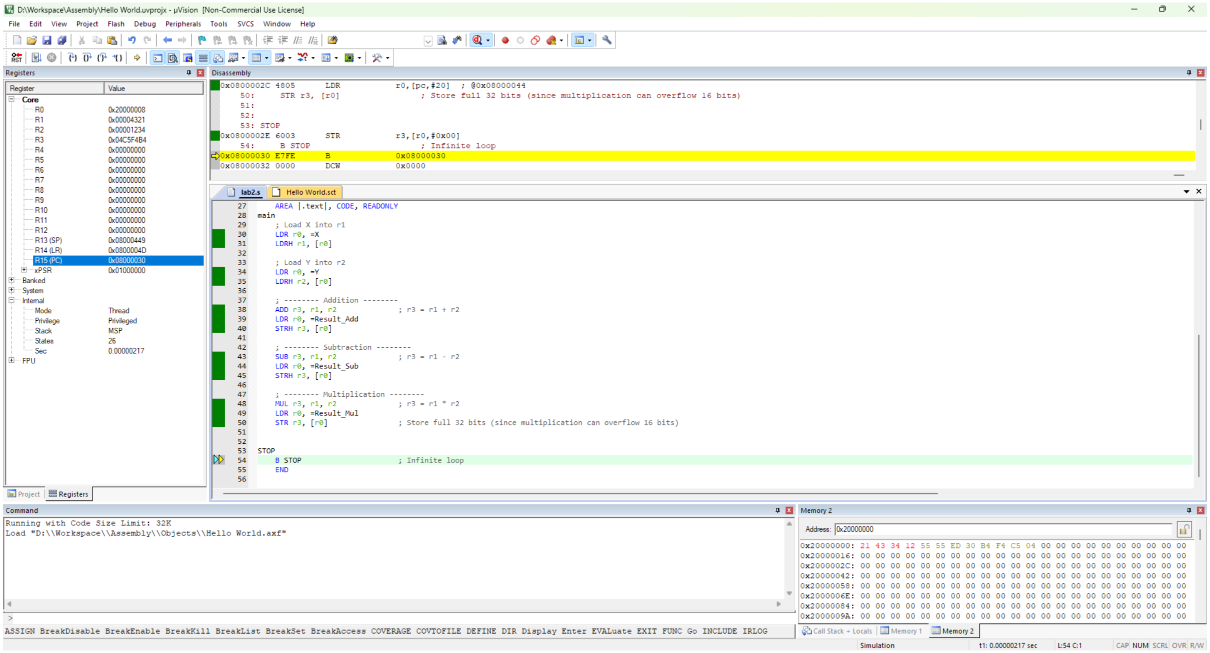
```
Running with Code Size Limit: 32K
Load "D:\\Workspace\\Assembly\\Objects\\Hello World.axf"
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE DEFINE DIR Display Enter EVALuate EXIT FUNC Go INCLUDE IRLOG
```

```
Address: 0x20000000
0x20000000: 21 43 34 12 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000002C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000042: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000058: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000006E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000084: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000009A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Simulation    t1: 0.00000000 sec    L:37 C:33    CAP NUM SCRL OVR R/W
```

Figure 3: Memory contents after loading the assembly code

2.3.2 System State After Code Execution


D:\Workspace\Assembly\Hello World.uvprojx - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
lab2.s
Hello World.sct
Command
Memory 2
Project
Registers
Call Stack + Locals
Memory 1
Memory 2

```
Register      Value
Core
  R0          0x20000008
  R1          0x00004321
  R2          0x00001234
  R3          0x04C5F4B4
  R4          0x00000000
  R5          0x00000000
  R6          0x00000000
  R7          0x00000000
  R8          0x00000000
  R9          0x00000000
  R10         0x00000000
  R11         0x00000000
  R12         0x00000000
  R13 (SP)    0x08000449
  R14 (LR)    0x0800004D
  R15 (PC)    0x08000030
  xPSR        0x01000000
Banked
System
Internal
  Mode        Thread
  Privilege   Privileged
  Stack       MSP
  States      26
  Sec         0.00000217
FPU
```

```
0x0800002C 4805       LDR          r0,[pc,#20]  ; @0x08000044
    50:      STR r3, [r0]               ; Store full 32 bits (since multiplication can overflow 16 bits)
    51:
    52:
    53: STOP
0x0800002E 6003       STR          r3,[r0,#0x00]
    54:      B STOP                     ; Infinite loop
0x08000030 E7FE       B            0x08000030
0x08000032 0000       DCW          0x0000
```

```
    AREA |.text|, CODE, READONLY
main
    ; Load X into r1
    LDR r0, =X
    LDRH r1, [r0]

    ; Load Y into r2
    LDR r0, =Y
    LDRH r2, [r0]

    ; -------- Addition --------
    ADD r3, r1, r2              ; r3 = r1 + r2
    LDR r0, =Result_Add
    STRH r3, [r0]

    ; -------- Subtraction --------
    SUB r3, r1, r2              ; r3 = r1 - r2
    LDR r0, =Result_Sub
    STRH r3, [r0]

    ; -------- Multiplication --------
    MUL r3, r1, r2              ; r3 = r1 * r2
    LDR r0, =Result_Mul
    STR r3, [r0]                ; Store full 32 bits (since multiplication can overflow 16 bits)


STOP
    B STOP                      ; Infinite loop
    END

```

```
Running with Code Size Limit: 32K
Load "D:\\Workspace\\Assembly\\Objects\\Hello World.axf"
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE DEFINE DIR Display Enter EVALuate EXIT FUNC Go INCLUDE IRLOG
```

```
Address: 0x20000000
0x20000000: 21 43 34 12 55 55 ED 30 B4 F4 C5 04 00 00 00 00 00 00 00 00 00 00
0x20000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000002C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000042: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000058: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000006E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000084: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000009A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Simulation    t1: 0.00000217 sec    L:54 C:1    CAP NUM SCRL OVR R/W
```

Figure 4: Memory contents after executing the assembly code

3 Task 3: Finding the Smaller of Two Integer Numbers

3.1 Code with Comments

The following assembly code finds the smaller of two integer numbers:

```
    AREA STACK, NOINIT, READWRITE, ALIGN=3
    SPACE 1024                  ; Reserve 1024 bytes for stack

    AREA |.vectors|, CODE, READONLY
    EXPORT __Vectors
__Vectors
    DCD __stack_top             ; Initial stack pointer
    DCD Reset_Handler           ; Reset handler
    DCD 0                       ; NMI handler (placeholder)
    DCD 0                       ; HardFault handler (placeholder)

    AREA |startup|, CODE, READONLY
__stack_top EQU STACK + 1024
    EXPORT Reset_Handler
Reset_Handler
    BL main                     ; Call main
    B .                         ; Loop forever if main returns

    AREA |.data|, DATA, READWRITE
Num1        DCW 0x0018          ; First number (24 in decimal)
Num2        DCW 0x002A          ; Second number (42 in decimal)
SmallerNum  DCW 0x0000          ; Will store the smaller number

    AREA |.text|, CODE, READONLY
main
    ; Load the values of Num1 and Num2
    LDR r0, =Num1               ; Load address of Num1
    LDRH r1, [r0]               ; Load value of Num1 into r1

    LDR r0, =Num2               ; Load address of Num2
    LDRH r2, [r0]               ; Load value of Num2 into r2

    ; Compare the two numbers
    CMP r1, r2                  ; Compare r1 (Num1) with r2 (Num2)
    BLS Num1IsSmaller           ; Branch if r1 <= r2 (Num1 is smaller or equal)

    ; If we get here, Num2 is smaller
    LDR r0, =SmallerNum         ; Load address of SmallerNum
    STRH r2, [r0]               ; Store Num2 as the smaller number
    B Done                      ; Branch to Done

Num1IsSmaller
    LDR r0, =SmallerNum         ; Load address of SmallerNum
    STRH r1, [r0]               ; Store Num1 as the smaller number

Done
STOP
    B STOP                      ; Infinite loop
    END
```

3.2 Code Explanation

This code compares two numbers and finds the smaller one:

3.2.1 Data Section

- Num1 is defined as 0x0018 (24 in decimal)
- Num2 is defined as 0x002A (42 in decimal)
- SmallerNum is initialized to store the result

3.2.2 Main Logic

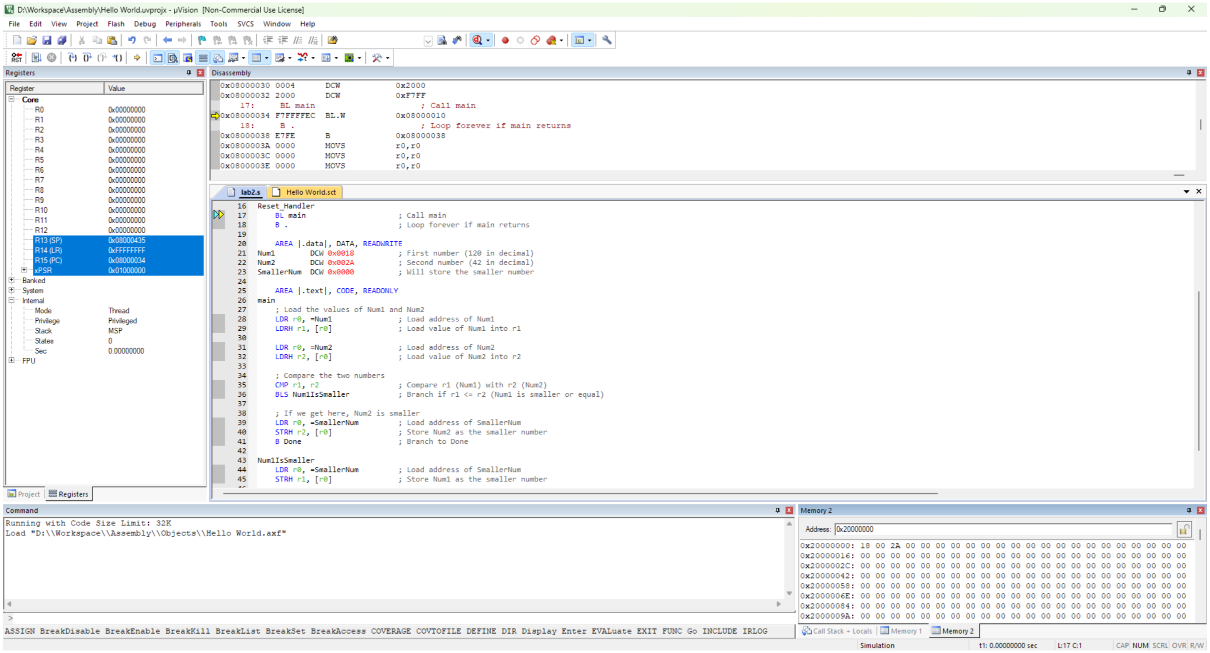
- Values of Num1 and Num2 are loaded into registers r1 and r2 respectively
- CMP instruction compares r1 and r2
- BLS (Branch if Lower or Same) instruction checks if r1 $\leq$ r2
- If r1 $\leq$ r2, the code branches to Num1IsSmaller
- Otherwise, r2 is assumed to be smaller and stored in SmallerNum
- At Num1IsSmaller, r1 is stored in SmallerNum

3.2.3 Expected Results

Since Num1 (24) is smaller than Num2 (42), the value 0x0018 (24) will be stored in SmallerNum.

3.3 System State Screenshots

3.3.1 System State After Code Loading



```
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)      0x08000435
  R14 (LR)      0xFFFFFFFF
  R15 (PC)      0x08000034
  xPSR          0x01000000
Banked
System
Internal
  Mode          Thread
  Privilege     Privileged
  Stack         MSP
  States        0
  Sec           0.00000000
FPU
```

```
0x08000030 0004      DCW          0x2000
0x08000032 2000      DCW          0xF7FF
    17:     BL main                    ; Call main
0x08000034 F7FFFFEC  BL.W         0x08000010
    18:     B .                        ; Loop forever if main returns
0x08000038 E7FE      B            0x08000038
0x0800003A 0000      MOVS         r0,r0
0x0800003C 0000      MOVS         r0,r0
0x0800003E 0000      MOVS         r0,r0
```

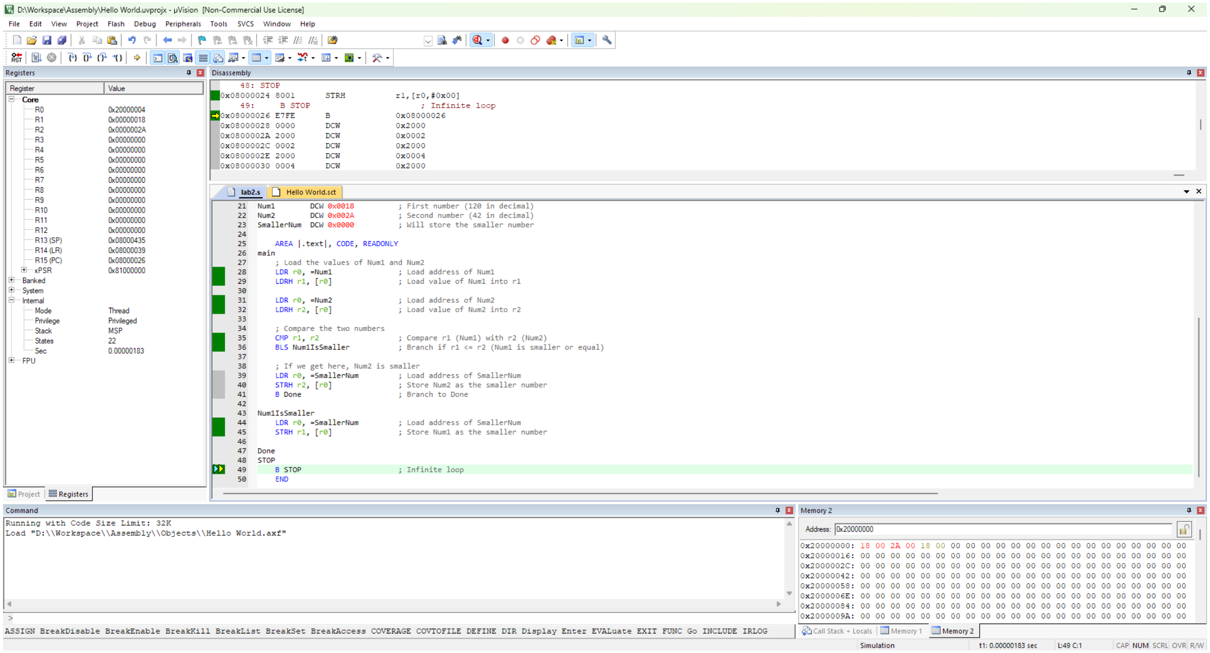
```
Reset_Handler
    BL main                     ; Call main
    B .                         ; Loop forever if main returns

    AREA |.data|, DATA, READWRITE
Num1        DCW 0x0018          ; First number (120 in decimal)
Num2        DCW 0x002A          ; Second number (42 in decimal)
SmallerNum  DCW 0x0000          ; Will store the smaller number

    AREA |.text|, CODE, READONLY
main
    ; Load the values of Num1 and Num2
    LDR r0, =Num1               ; Load address of Num1
    LDRH r1, [r0]               ; Load value of Num1 into r1

    LDR r0, =Num2               ; Load address of Num2
    LDRH r2, [r0]               ; Load value of Num2 into r2

    ; Compare the two numbers
    CMP r1, r2                  ; Compare r1 (Num1) with r2 (Num2)
    BLS Num1IsSmaller           ; Branch if r1 <= r2 (Num1 is smaller or equal)

    ; If we get here, Num2 is smaller
    LDR r0, =SmallerNum         ; Load address of SmallerNum
    STRH r2, [r0]               ; Store Num2 as the smaller number
    B Done                      ; Branch to Done

Num1IsSmaller
    LDR r0, =SmallerNum         ; Load address of SmallerNum
    STRH r1, [r0]               ; Store Num1 as the smaller number
```

```
Running with Code Size Limit: 32K
Load "D:\\Workspace\\Assembly\\Objects\\Hello World.axf"
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE DEFINE DIR Display Enter EVALuate EXIT FUNC Go INCLUDE IRLOG
```

```
Address: 0x20000000
0x20000000: 18 00 2A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000002C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000042: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000058: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000006E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000084: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000009A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 5: Memory contents after loading the assembly code

3.3.2 System State After Code Execution



```
Register        Value
Core
  R0            0x20000004
  R1            0x00000018
  R2            0x0000002A
  R3            0x00000000
  R4            0x00000000
  R5            0x00000000
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000000
  R13 (SP)      0x08000435
  R14 (LR)      0x08000039
  R15 (PC)      0x08000026
  xPSR          0x81000000
Banked
System
Internal
  Mode          Thread
  Privilege     Privileged
  Stack         MSP
  States        22
  Sec           0.00000183
FPU
```

```
    48: STOP
0x08000024 8001      STRH         r1,[r0,#0x00]
    49:     B STOP                     ; Infinite loop
0x08000026 E7FE      B            0x08000026
0x08000028 0000      DCW          0x2000
0x0800002A 2000      DCW          0x0002
0x0800002C 0002      DCW          0x2000
0x0800002E 2000      DCW          0x0004
0x08000030 0004      DCW          0x2000
```

```
Num1        DCW 0x0018          ; First number (120 in decimal)
Num2        DCW 0x002A          ; Second number (42 in decimal)
SmallerNum  DCW 0x0000          ; Will store the smaller number

    AREA |.text|, CODE, READONLY
main
    ; Load the values of Num1 and Num2
    LDR r0, =Num1               ; Load address of Num1
    LDRH r1, [r0]               ; Load value of Num1 into r1

    LDR r0, =Num2               ; Load address of Num2
    LDRH r2, [r0]               ; Load value of Num2 into r2

    ; Compare the two numbers
    CMP r1, r2                  ; Compare r1 (Num1) with r2 (Num2)
    BLS Num1IsSmaller           ; Branch if r1 <= r2 (Num1 is smaller or equal)

    ; If we get here, Num2 is smaller
    LDR r0, =SmallerNum         ; Load address of SmallerNum
    STRH r2, [r0]               ; Store Num2 as the smaller number
    B Done                      ; Branch to Done

Num1IsSmaller
    LDR r0, =SmallerNum         ; Load address of SmallerNum
    STRH r1, [r0]               ; Store Num1 as the smaller number

Done
STOP
    B STOP                      ; Infinite loop
    END
```

```
Running with Code Size Limit: 32K
Load "D:\\Workspace\\Assembly\\Objects\\Hello World.axf"
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE DEFINE DIR Display Enter EVALuate EXIT FUNC Go INCLUDE IRLOG
```

```
Address: 0x20000000
0x20000000: 18 00 2A 00 18 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000016: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000002C: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000042: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000058: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000006E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000084: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000009A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 6: Memory contents after executing the assembly code

4 Conclusion

This lab demonstrated the implementation of basic arithmetic operations and comparison logic in assembly language. The programs successfully:

- Added two 16-bit variables
- Performed addition, subtraction, and multiplication on two variables
- Found the smaller of two integer numbers

These implementations illustrate fundamental assembly language concepts including memory access, register operations, and conditional branching.