CSE 3113: Microprocessor and Assembly Language

1. *Install Arm Fixed Virtual Platforms (FVPs): a complete simulation of ARM systems*

   https://www.keil.com/mdk5/simulation

2. *Assembly Language Syntax*

   label

   opcode operand1, operand2, ... ; Comment

3. *(i) label:*

   - *Label is an optional first field of an assembly statement.*
   - *Labels are alphanumeric names used to define the starting location of a block of statements.*
   - *When creating the executable file the assembler will replace the label with the assigned value.*

   (ii) Opcode (Mnemonics) :

   - *Opcode is the second field in assembly language instruction.*
   - *Assembly language consists of mnemonics, each corresponding to a machine instruction.*
   - *Assembler must translate each mnemonic opcode into their binary equivalent.*

   (iii) Operands:

   - *Next to the opcode is the operand field which might contain different number of operands.*
   - *Normally, the first operand is the destination of the operation.*

   (iv) Comments:
   Comments are messages intended only for human consumption.

4. ***A Sample ARM Assembly Program***

```
          AREA test, CODE, READONLY
          ENTRY ; starting point of the code execution
          EXPORT main ; the declaration of identifier main
 main ; address of the main function
     ; User code starts from the next line
          MOV r0, #4 ; store some arbitrary numbers
          MOV r1, #5
          ADD r2, r0, r1 ;  add the values in r0 and r1 and store the  result in r2
STOP  B Stop ; Endless loop
          END ; End of the program, matched with ENTRY keyword
```

- ; indicates user- supplied comment.
- AREA test, CODE, READONLY is an assembler directive and is required to setup the program.
- AREA refers to the segment code, test is the name I have defined,
- CODE means executable code rather that data, and
- READONLY indicates that it cannot be modified at runtime.
- Anything used in column1 is a label that is used to label that line.
- Stop B Stop means " Branch to line labeled Stop" , used to create an infinite loop. This is a way to end the program.
- Last line END tells the assembler that there is no more code to execute.

5. Assembler Directives:
    - Keil has an ARM assembler which can compile and build ARM assembly language programs.
    - To drive the assembly and linking process, we need to use directives, which are interpreted by the assembler.
    - Assembler directives are commands to the assembler that direct the assembly process.
    - They are executed by the assembler at assembly time not by the processor at run time.
    - Machine code is not generated for assembler directives as they are not directly translated to machine language.
6. AREA Directive
    - AREA directive allows the programmer to specify the memory location to store code and data.
    - A name must be specified for an area directive.
7. ENTRY and END Directives
    - The first instruction to be executed within an application is marked by the ENTRY directive.
    - Entry point must be specified for every assembly language program.

- *This directive causes the assembler to stop processing the current source file.*
- *Every assembly language source module must therefore finish with this directive.*

8. *EXPORT Directives*

   - *A project may contain multiple source files. You may need to use a symbol in a source file that is defined in another source file.*
   - *In order for a symbol to be found by a different program file, we need to declare that symbol name as a global variable.*
   - *The EXPORT directive declares a symbol that can be used in different program files.*

9. *The EQUATE Directive*

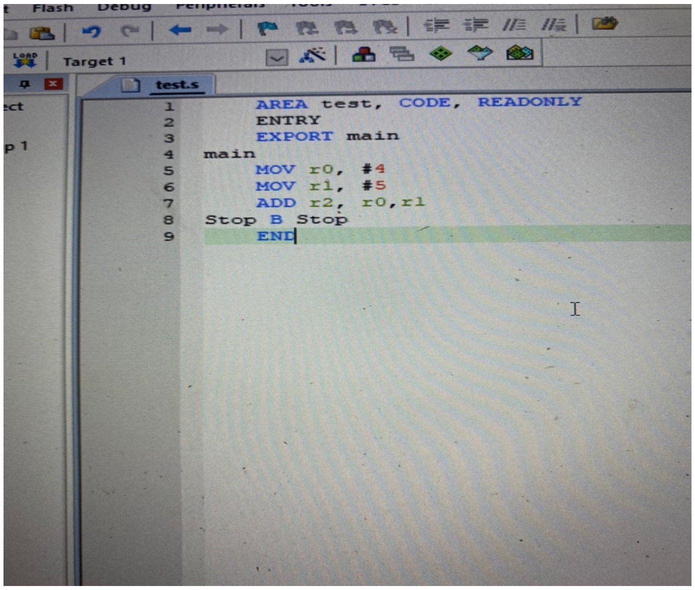
   - *The EQUATE directive allows the programmer to equate names with addresses or data.*
   - *This pseudo-operation is almost always given the mnemonic EQU.*
   - *The names may refer to device addresses, numeric data, starting addresses, fixed addresses, etc.*

10. *READONLY as the name indicates protects this area from being overwritten by the program code.*

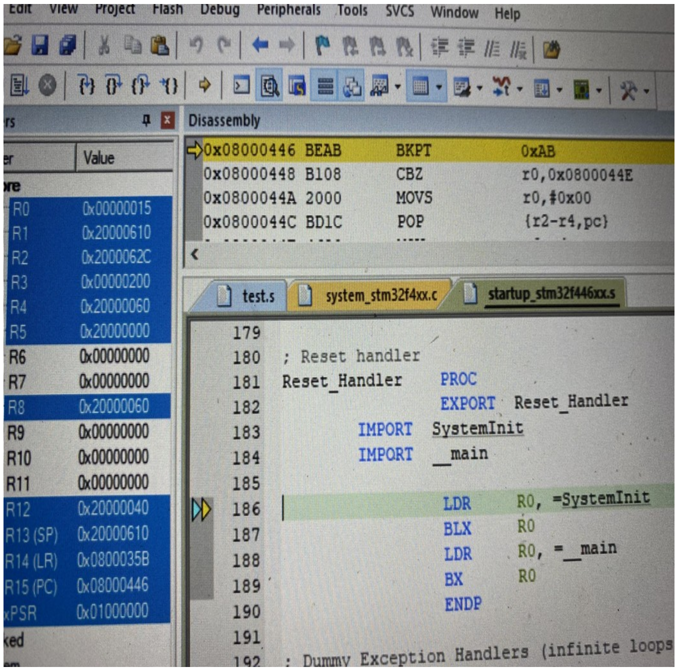
11. ***Some Basic Instruction***

    - *Data Processing Instructions*
    - *Arithmetic operations: – ADD, SUB, MUL*
    - *Bit-wise logical operations: – AND, EOR, ORR, BIC*
    - *Register movement operations: – MOV*
    - *Comparison operations: – TST, TEQ, CMP, CMN*
    - *LDR : Load Word from memory to register*
    - *STR: Store Word from register to memory*

12. ***Debug Scenario of the Sample Program***



```
        AREA test, CODE, READONLY
        ENTRY
        EXPORT main
main
        MOV r0, #4
        MOV r1, #5
        ADD r2, r0,r1
Stop B Stop
        END
```



er	Value
ore	
R0	0x00000015
R1	0x20000610
R2	0x2000062C
R3	0x00000200
R4	0x20000060
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x20000060
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000610
R14 (LR)	0x0800035B
R15 (PC)	0x08000446
xPSR	0x01000000
ked	

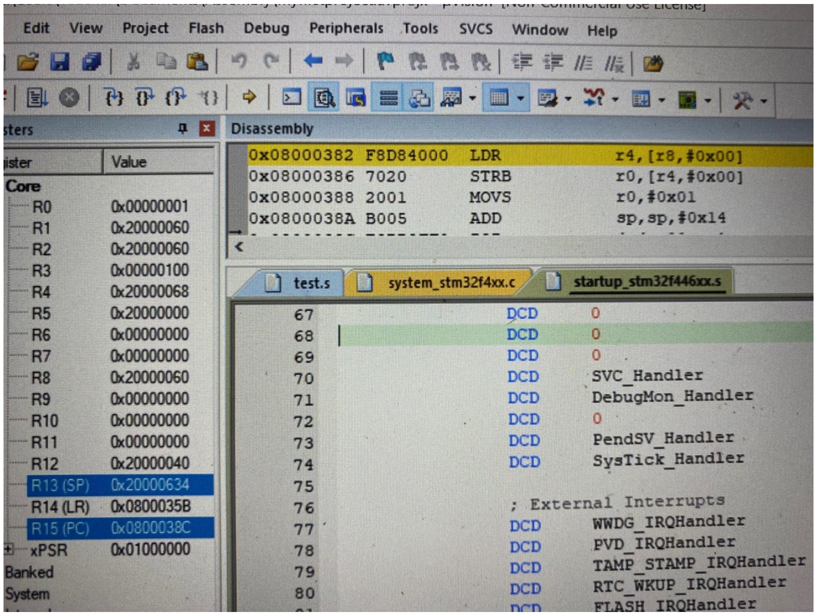
```
0x08000446 BEAB     BKPT       0xAB
0x08000448 B108     CBZ        r0,0x0800044E
0x0800044A 2000     MOVS       r0,#0x00
0x0800044C BD1C     POP        {r2-r4,pc}
```

```

; Reset handler
Reset_Handler    PROC
                 EXPORT  Reset_Handler
         IMPORT  SystemInit
         IMPORT  __main

                 LDR     R0, =SystemInit
                 BLX     R0
                 LDR     R0, =__main
                 BX      R0
                 ENDP

; Dummy Exception Handlers (infinite loops
```

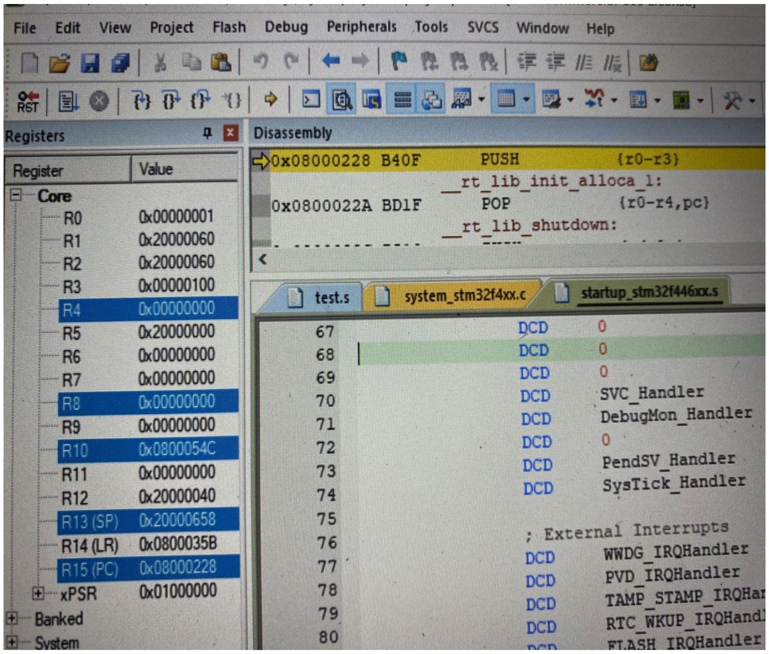


ister	Value
Core	
R0	0x00000001
R1	0x20000060
R2	0x20000060
R3	0x00000100
R4	0x20000068
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x20000060
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000634
R14 (LR)	0x0800035B
R15 (PC)	0x0800038C
xPSR	0x01000000
Banked	
System	

```
0x08000382 F8D84000  LDR          r4,[r8,#0x00]
0x08000386 7020      STRB         r0,[r4,#0x00]
0x08000388 2001      MOVS         r0,#0x01
0x0800038A B005      ADD          sp,sp,#0x14
```

```
                DCD     0
                DCD     0
                DCD     0
                DCD     SVC_Handler
                DCD     DebugMon_Handler
                DCD     0
                DCD     PendSV_Handler
                DCD     SysTick_Handler

                ; External Interrupts
                DCD     WWDG_IRQHandler
                DCD     PVD_IRQHandler
                DCD     TAMP_STAMP_IRQHandler
                DCD     RTC_WKUP_IRQHandler
                DCD     FLASH_IRQHandler
```

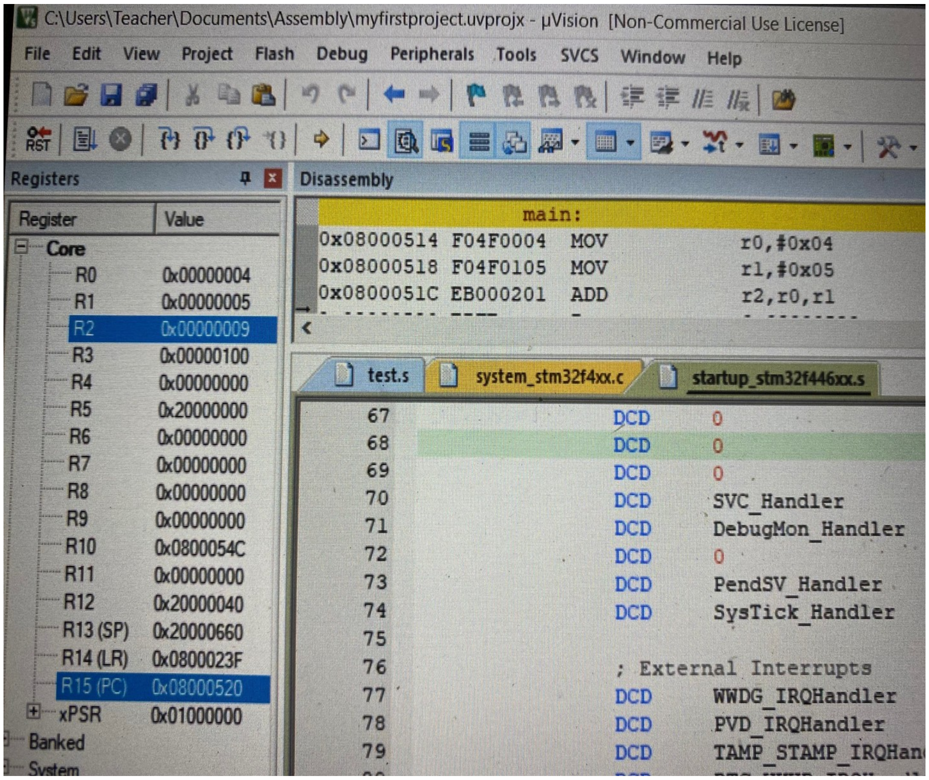


Register	Value
Core	
R0	0x00000001
R1	0x20000060
R2	0x20000060
R3	0x00000100
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x0800054C
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000658
R14 (LR)	0x0800035B
R15 (PC)	0x08000228
xPSR	0x01000000
Banked	
System	

```
0x08000228 B40F      PUSH         {r0-r3}
                __rt_lib_init_alloca_1:
0x0800022A BD1F      POP          {r0-r4,pc}
                __rt_lib_shutdown:
```

```
                DCD     0
                DCD     0
                DCD     0
                DCD     SVC_Handler
                DCD     DebugMon_Handler
                DCD     0
                DCD     PendSV_Handler
                DCD     SysTick_Handler

                ; External Interrupts
                DCD     WWDG_IRQHandler
                DCD     PVD_IRQHandler
                DCD     TAMP_STAMP_IRQHar
                DCD     RTC_WKUP_IRQHandl
                DCD     FLASH_IRQHandler
```



Register	Value
Core	
R0	0x00000004
R1	0x00000005
R2	0x00000009
R3	0x00000100
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x0800054C
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000660
R14 (LR)	0x0800023F
R15 (PC)	0x08000520
xPSR	0x01000000
Banked	
System	

```
                    main:
0x08000514 F04F0004 MOV          r0,#0x04
0x08000518 F04F0105 MOV          r1,#0x05
0x0800051C EB000201 ADD          r2,r0,r1
```

```
        DCD     0
        DCD     0
        DCD     0
        DCD     SVC_Handler
        DCD     DebugMon_Handler
        DCD     0
        DCD     PendSV_Handler
        DCD     SysTick_Handler

        ; External Interrupts
        DCD     WWDG_IRQHandler
        DCD     PVD_IRQHandler
        DCD     TAMP_STAMP_IRQHan
```

13. *Your Task:*

(i) This problem is same as the sample problem. W = X + Y + Z Once again, let X = 9, Y = 8, Z = 5 and we assume that r4 = X, r3 = Y, r2 = Z. In this case, you will put the data in memory in the form of constants before the program runs.
(ii) Repeat the previous problem once again is W = X + Y + Z Once again, let X = 9, Y = 8, Z = 5 and we assume that r4 = X, r3 = Y, r2 = Z.. In this case, you will put the data in memory as constants before the program runs. But you first use the load register, LDR r4,X instruction to load register r4 with the contents of memory location r4.
(iii) Find the addition of two 16 bit variables v1 and v2.
(iv) Find the smaller of two integer numbers.

14. *Submission Guideline:*

1. Your Assembly code with proper comments. (.s file)*
2. A document (.tex file) that contains:*
a. Detail explanation of the code
b. Screenshot that shows the state of the system after the code has been loaded.
c. Screenshot that shows the situation after the code has been executed.
3. Submit as a .zip file. Example: your classroll_lab#.zip (12_lab2.zip)

Thank You !!!!