ENCS2380

Computer Organization Project #2 Report

جامعة بيرزيت

BIRZEIT UNIVERSITY

Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Mona Dweikat-1200277

Hazar Michael-1201838

Dr. Abualseoud Hanani

Asst. Raha Zabade

20.02.2023

Abstract

This Assembly language programming project aims to implement three procedures that demonstrate the manipulation of strings in ARM Assembly. The project helps in understanding the instructions used, and seeing how registers work in storing operands.

Table of Content

Theory

The Keil software allows for coding in ARM Assembly, and provides the output in registers and a memory viewer to keep track of the code. The first procedure used, named "ConvertString" converts capital letters to small letters in two input strings and stores the result in two separate arrays. The second procedure named “Common”, compares each character of converted string1 to each character in converted string2, and counts the number of common letters and stores that value into a register. The third procedure named “EncryptString”, XORs the ASCII code of each string’s character that means complementing each bit (i.e., changing 0s to 1s and 1s to 0s) to obtain the encrypted value.

Question 1

string1

DCB "AsseMbly", 0

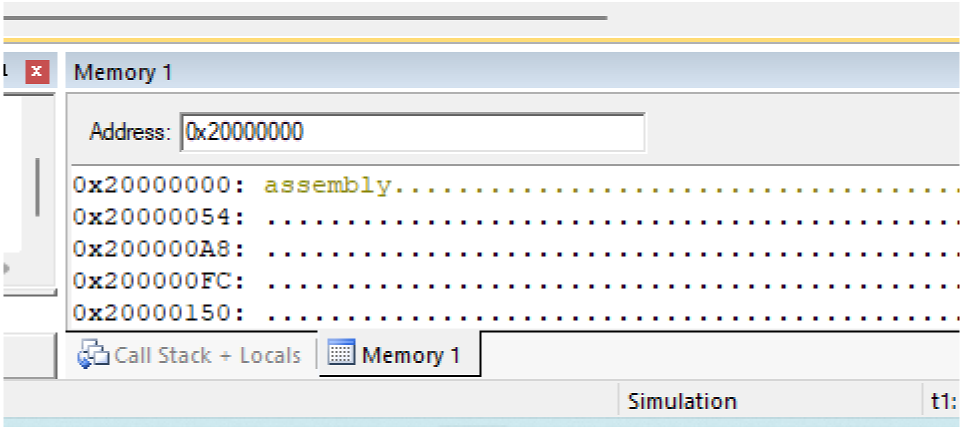
string2

DCB "PrograMMing", 0

Question 2

String 1 = “AsseMbly” , String 2 = “PrograMMing ”

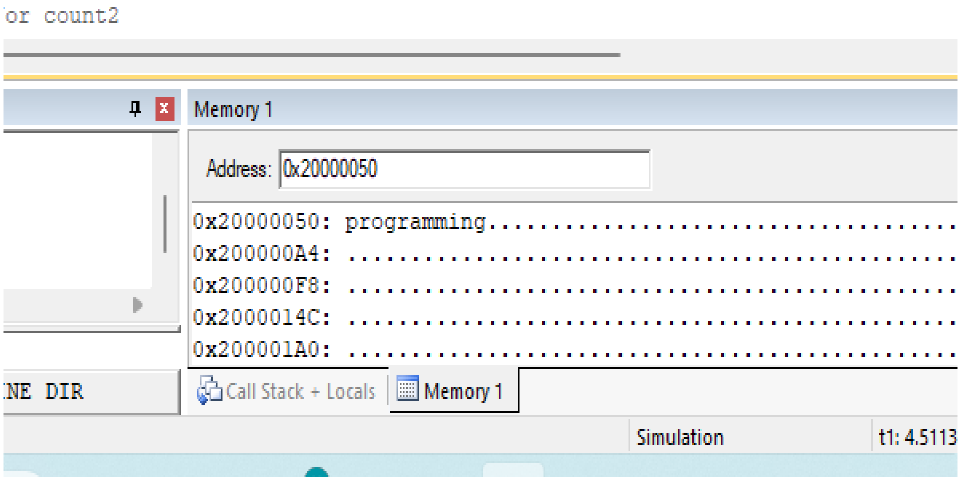
TXT1 is Stored in R0 = assembly , as shown in the memory viewer



Address: 0x20000000

```
0x20000000: assembly................................
0x20000054: ....................................
0x200000A8: ....................................
0x200000FC: ....................................
0x20000150: ....................................
```

TXT2 is Stored in R2 = programming



Address: 0x20000050

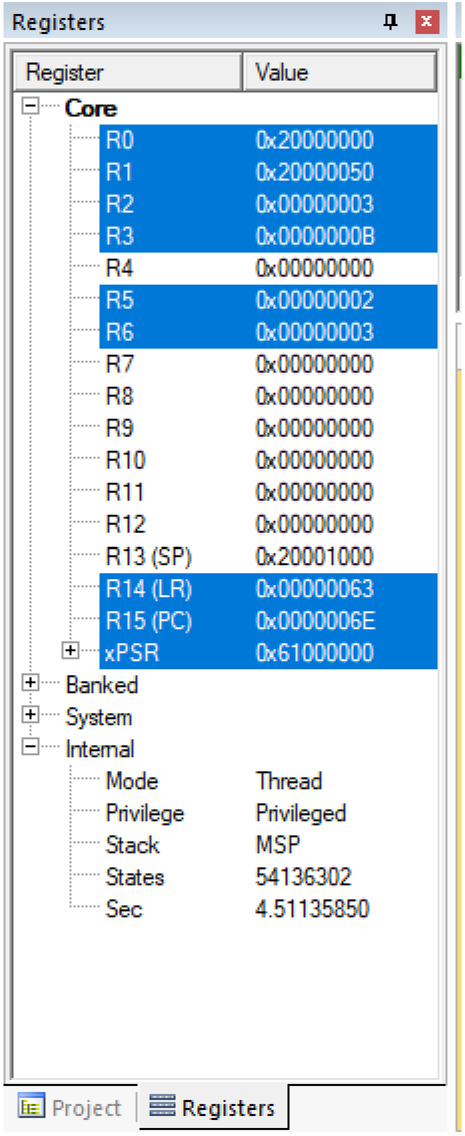
```
0x20000050: programming...............................
0x200000A4: ....................................
0x200000F8: ....................................
0x2000014C: ....................................
0x200001A0: ....................................
```

Count1 is stored in R5 which holds the number of converted letters from String 1, = 2 (A,M)

Count2 is stored in R6 which holds the number of converted letters from String 2, =3 (P,M,M)

```
ConvertString PROC

  PUSH {R4, LR}

  MOV R2, #0 ; initialize count to 0

  MOV R3, #0 ; initialize index to 0

ConvertLoop

  LDRB R4, [R0], #1 ; load a byte into the register and increment by 1

  CMP R4, #0 ; check for end of string

  BEQ ConvertDone

  CMP R4, #'A' ; check for capital letter

  BLT ConvertNext

  CMP R4, #'Z'

  BGT ConvertNext

  ADD R4, R4, #'a'-'A' ; convert to small letter

  STRB R4, [R1, R3] ; store the converted letter into an array

  ADD R2, R2, #1 ; increment count

ConvertNext

  STRB R4, [R1, R3]
```

```
    ADD R3, R3, #1 ; increment index

    B ConvertLoop

ConvertDone

    STR R2, [R1, #32] ; store the count of converted letters to register R2

    MOV R0, R1 ; return address of converted string

    POP {R4, PC}

    ENDP
```



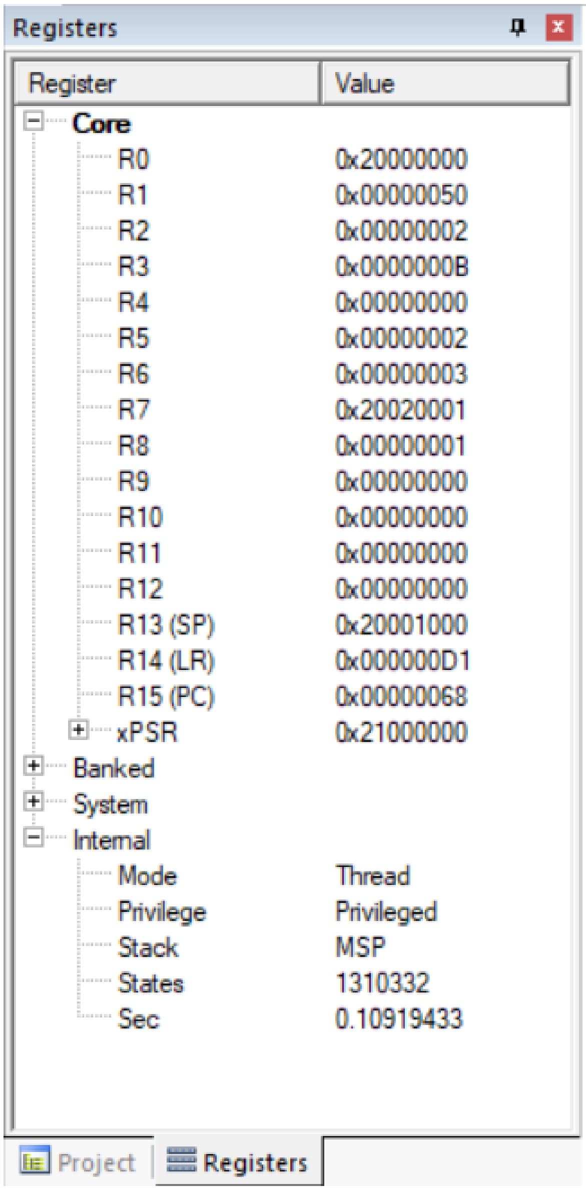
```
Registers
Register        Value
⊟ Core
    R0          0x20000000
    R1          0x20000050
    R2          0x00000003
    R3          0x0000000B
    R4          0x00000000
    R5          0x00000002
    R6          0x00000003
    R7          0x00000000
    R8          0x00000000
    R9          0x00000000
    R10         0x00000000
    R11         0x00000000
    R12         0x00000000
    R13 (SP)    0x20001000
    R14 (LR)    0x00000063
    R15 (PC)    0x0000006E
  ⊞ xPSR        0x61000000
⊞ Banked
⊞ System
⊟ Internal
    Mode        Thread
    Privilege   Privileged
    Stack       MSP
    States      54136302
    Sec         4.51135850
```

Question 3

For the second procedure, it compares each letter of TXT1 to TXT2 and counts the number of common letters between the two strings, and stores it in a variable called Common.

```
Common PROC
   LDRB R3, [R0], #1 ; load a byte from TXT1, increment address by 1
   CMP R3, #0 ; check for end of string
   BEQ CommonDone
   MOV R7, R1 ; save the address of TXT2 to R7
   MOV R1, R5 ; set the address of count2 to R1
   LDR R5, [R5] ; load the count of TXT2 to R5
   MOV R8, #0 ; initialize flag to 0
InnerLoop
   LDRB R4, [R7], #1 ; load a byte from TXT2, increment address by 1
   CMP R4, #0 ; check for end of string
   BEQ NextChar
   CMP R3, R4 ; compare the two bytes
   BNE NextChar
   ADD R2, R2, #1 ; increment count
   CMP R8, #0 ; check if the letter has been found before
   BNE NextChar
```

```
    MOV R8, #1 ; set flag to 1
NextChar
    SUBS R5, R5, #1 ; decrement count of TXT2
    BNE InnerLoop
    MOV R5, R1 ; restore the address of count2 to R5
    MOV R1, R7 ; restore the address of TXT2 to R1
    CMP R2, #0 ; check if any common letter was found
    BEQ CommonNext
    LDR R7, [R6] ; load the current common value from memory
    ADD R2, R2, R7 ; add the new count to the common value
    STR R2, [R6] ; store the updated common value to memory
CommonNext
    LDRB R3, [R0], #1 ; load a byte from TXT1, increment address by 1
    B Common
CommonDone
    MOV R0, #0
    POP {R4, PC} ; return
    ENDP
```



Register	Value
Core	
R0	0x20000000
R1	0x00000050
R2	0x00000002
R3	0x0000000B
R4	0x00000000
R5	0x00000002
R6	0x00000003
R7	0x20020001
R8	0x00000001
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20001000
R14 (LR)	0x000000D1
R15 (PC)	0x00000068
xPSR	0x21000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	1310332
Sec	0.10919433

Question 4

For the third procedure, it goes over all the characters into a string, and encrypts the ASCII code of each, (i.e it complements each bit for every ASCII character). The encrypted value of TXT1 is stored in ENCRYPT1, and TXT2 is stored in ENCRYPT2.

```
EncryptString PROC
   PUSH {R4, LR}
   MOV R3, #0 ; initialize index to 0
EncryptLoop
   LDRB R4, [R0], #1 ; load a byte from input string, and increment address by 1
   CMP R4, #0 ; check for end of string
   BEQ EncryptDone
   MVN R4, R4 ; invert all the bits of the byte
   STRB R4, [R1, R3] ; store the encrypted byte to the output array
   ADD R3, R3, #1 ; increment index
   B EncryptLoop
EncryptDone
   POP {R4, PC}
   ENDP
```

- For full program code, check Appendix A.

Appendices

Appendix A

```
        PRESERVE8

        THUMB



        AREA RESET, DATA, READONLY

        EXPORT __Vectors

__Vectors

  DCD 0x20001000

  DCD Reset_Handler

  ALIGN



string1

  DCB "AsseMbly", 0

string2

  DCB "PrograMMing", 0



TXT1

  SPACE 36
```

```
count1
   DCB 4


TXT2 EQU 0x20000050
   SPACE 36
count2
   DCB 4
COMMON
   DCB 4
ENCRYPT1
  SPACE 32
ENCRYPT2
  SPACE 32
         AREA MYCODE, CODE, READONLY
         ENTRY
         EXPORT Reset_Handler


ConvertString PROC
   PUSH {R4, LR}
   MOV R2, #0 ; initialize count to 0
```

```
    MOV R3, #0 ; initialize index to 0
ConvertLoop
    LDRB R4, [R0], #1 ; load a byte into the register and increment by 1
    CMP R4, #0 ; check for end of string
    BEQ ConvertDone
    CMP R4, #'A' ; check for capital letter
    BLT ConvertNext
    CMP R4, #'Z'
    BGT ConvertNext
    ADD R4, R4, #'a'-'A' ; convert to small letter
    STRB R4, [R1, R3] ; store the converted letter into an array
    ADD R2, R2, #1 ; increment count
ConvertNext
    STRB R4, [R1, R3]
    ADD R3, R3, #1 ; increment index
    B ConvertLoop
ConvertDone
    STR R2, [R1, #32] ; store the count of converted letters to register R2
    MOV R0, R1 ; return address of converted string
    POP {R4, PC}
    ENDP
Common PROC
    LDRB R3, [R0], #1 ; load a byte from TXT1, increment address by 1
    CMP R3, #0 ; check for end of string
    BEQ CommonDone
    MOV R7, R1 ; save the address of TXT2 to R7
    MOV R1, R5 ; set the address of count2 to R1
    LDR R5, [R5] ; load the count of TXT2 to R5
    MOV R8, #0 ; initialize flag to 0
InnerLoop
    LDRB R4, [R7], #1 ; load a byte from TXT2, increment address by 1
    CMP R4, #0 ; check for end of string
    BEQ NextChar
    CMP R3, R4 ; compare the two bytes
    BNE NextChar
    ADD R2, R2, #1 ; increment count
    CMP R8, #0 ; check if the letter has been found before
    BNE NextChar
    MOV R8, #1 ; set flag to 1
NextChar
```

```
    SUBS R5, R5, #1 ; decrement count of TXT2

    BNE InnerLoop

    MOV R5, R1 ; restore the address of count2 to R5

    MOV R1, R7 ; restore the address of TXT2 to R1

    CMP R2, #0 ; check if any common letter was found

    BEQ CommonNext

    LDR R7, [R6] ; load the current common value from memory

    ADD R2, R2, R7 ; add the new count to the common value

    STR R2, [R6] ; store the updated common value to memory

CommonNext

    LDRB R3, [R0], #1 ; load a byte from TXT1, increment address by 1

    B Common

CommonDone

    MOV R0, #0

    POP {R4, PC} ; return

    ENDP

EncryptString PROC

    PUSH {R4, LR}

    MOV R3, #0 ; initialize index to 0

EncryptLoop
```

```
    LDRB R4, [R0], #1 ; load a byte from input string, and increment address by 1

    CMP R4, #0 ; check for end of string

    BEQ EncryptDone

    MVN R4, R4 ; invert all the bits of the byte

    STRB R4, [R1, R3] ; store the encrypted byte to the output array

    ADD R3, R3, #1 ; increment index

    B EncryptLoop

EncryptDone

    POP {R4, PC}

    ENDP



Reset_Handler

    ; Load the address of string1 to R0 and the address of TXT1 to R1

    LDR R0, =string1

    LDR R1, =TXT1



    ; Call the ConvertString procedure to convert string1 and store it in TXT1

    BL ConvertString



    ; Load the address of string2 to R0 and the address of TXT2 to R1
```

```
        LDR R0, =string2
        LDR R1, =TXT2

        ; Call the ConvertString procedure to convert string2 and store it in TXT2
        BL ConvertString

        ; Load the addresses of TXT1 and TXT2 to R0 and R1
        LDR R0, =TXT1
        LDR R1, =TXT2

        ; Call the CompareStrings procedure to compare the two strings and store the count
and common value
        BL Common

        ; Load the count and common value from their memory locations into R3 and R4
        LDR R3, =count1
        LDR R3, [R3]
        LDR R4, =count2
        LDR R4, [R4]
          LDR R2, =COMMON
```

```
            LDR R2, [R2] ; load the value of common from the memory location pointed to by R2

            LDR R0, =TXT1

        LDR R1, =ENCRYPT1

        BL EncryptString ; encrypt TXT1 and store in ENCRYPT1

        LDR R0, =TXT2

        LDR R1, =ENCRYPT2

        BL EncryptString ; encrypt TXT2 and store in ENCRYPT2


      STOP

        B STOP


        END
```