EE2016: Microprocessors Lab
Experiment # 6: ARM Assembly 2 - Computations in ARM

Amizhthni PRK, EE21B015
Nachiket Dighe, EE21B093

November 2, 2022

Contents

List of Figures

1 To check Even-odd Parity

1.1 Aim of this experiment:

Given a 32-bit number, generate even parity bit for that (32-bit) word.

1.2 Algorithm and Approach

The method we are using is to count number of 1's and output 1 if number of 1's are odd and output 0 if number of 1's are even.

- R0 receives the input of the 32 bit number
- Loading R1 with 1 which is used to check each bit of R0
- Loading R2 with 20 which is a counter for 32 bits
- Coding a for loop which will run until R2 becomes 0
- Loading R3 with 0 which will serve as a counter to check no of 1's.
- **AND**ing R0 and R1 to get the first bit of R0 and storing in R10.
- If R10 = 0, left shift R1 by 1 bit, else increment R3 and left shift R1 by 1 bit.
- Decrement R2 and compare with 0(if true then exit the loop otherwise goto **Loop**).
- Check for even parity and store result in R4.
- End

1.3 Code

The code for checking even/odd parity of a given number is given in listing 1.

```
AREA program,CODE,READONLY
        ENTRY

        LDR R0,= 0x12341234; Taking 32 bit number whose even parity is to be checked
        LDR R1,= 0x01; Used to check each bit of R0
        LDR R2,= 0x20; Counter for 32 bits
    LDR R3,= 0x00; Counter

Loop
        AND R10, R0, R1; AND the first Bit of R0 with R1 and store the result in R10
        CMP R10, #0 ;Check if it is zero
        BEQ label;
        ADD R3, R3, #1; Count increases to 1 if we get 1 in input
label LSL R1, #1; Left shift of R1 to check for next input bit
        SUB R2, R2, #1; Since it is 32 bit number, we need to run loop 32 times
        CMP R2, #0; Exit from loop if R2 = 0
        BNE Loop
```

```

        AND R4, R3, #1; Check for even parity and storing result in R4

        END
```

Listing 1: Code for checking EVEN/ODD Parity

1.4 Inference

- **Input** taken as **0x12341234** in R0.
- Total number of **1's** in R0 is **0x0A** which is stored in R3
- Since the result obtained is even, EVEN parity will be zero
- Thus, **0x00** is stored in R4 as **output**.

1.5 Outputs

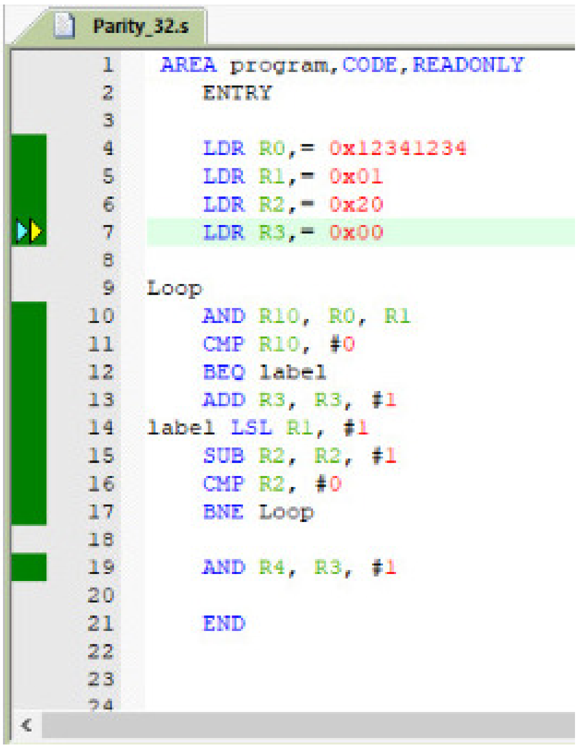

Figure 1: Code

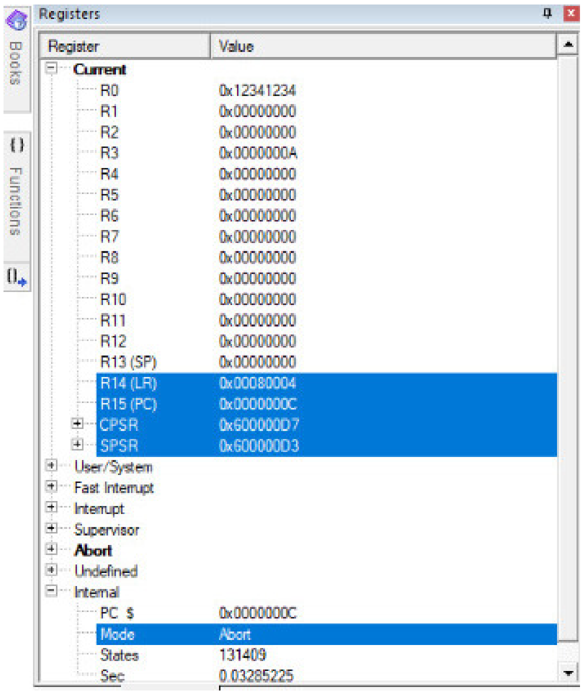

Figure 2: Output(R4 = 0x00) for input (R1 = 0x12341234)

2 Total Number of Characters in a given String

2.1 Aim of this experiment:

- Determine the length of an ASCII message. All characters are 7-bit ASCII with MSB = 0.
- The string of characters in which the message is embedded has a starting address which is contained in the START variable.
- The message itself starts with an ASCII STX (Start of Text) character (0x02) and ends with ETX (End of Text) character (0x03).
- Save the length of the message, the number of characters between the STX and the ETX markers (but not including the markers) in the LENGTH variable .

2.2 Algorithm

- START
- Store the contents of the given **LIST** in a Register **R0**.
- Initialize **R1** to 0 for using it to count the number of characters in the string.
- Load the *first* element of the LIST in **R3** and move the pointer of R0 to next element.
- Check if the element is 0x02 (**STX**) by subtracting R3 by 2 (if not equal to zero then repeat the loop **findSTX**).
- Load the next element in R3 and increment R1 counter by 1.
- Check if the element is 0x03 (**ETX**) by subtracting R3 by 0x03 (if not equal to zero then repeat the loop **findETX**).

- Store the given Result(**R1**) in **length**.
- END the program.

2.3 Code

The code used for Finding the number of characters in a string is given in listing 2.

```
;TTL wordCount
                AREA Program, CODE, READONLY
                ENTRY
Main
                LDR R0, Message;
                EOR R1, R1, R1; Initialize R1 to 0 to count the number of characters

findSTX
                LDR R3, [R0], #4; Move the first element of LIST in R3 and move the pointer of R0 to next element
                SUBS R3, R3, #2; Check for the starting text(0x02)
                BNE findSTX; If it is not equal to zero run the loop again
findETX
                LDR R3, [R0], #4; Move the element of LIST in R3 and move the pointer of R0 to next element
                ADD R1, #1; Increment the counter to count the total words
                SUBS R3, R3, #3; Check for the ending text(0x03)
                BNE findETX; If it is not equal to zero run the loop again
Done
                SUB R1, #1; to get the elements in between start and end text
                STR R1, length; store the result in length
Stop
                B Stop;

LIST
                        DCD &5C;
                        DCD &02; Start of Text
                        DCD &2D;
                        DCD &04;
                        DCD &2D;
                        DCD &1A;
                        DCD &0A;
                        DCD &03; End of Text
                        ALIGN

Message DCD LIST;

length DCW 0;
           ALIGN

           END;
```

Listing 2: Code for finding the Length of a string

2.4 Inference

- LIST contains some elements of which we check for 0x02.
- After that increment counter of number of characters (R1) by 1 for each character.
- Characters are : **0x2D, 0x04, 0x2D, 0x1A, 0x0A, 0x03**
- Result obtained is **5** which is stored in **R1** and the program ends after the first occurence of 0x03.

2.5 Output figures

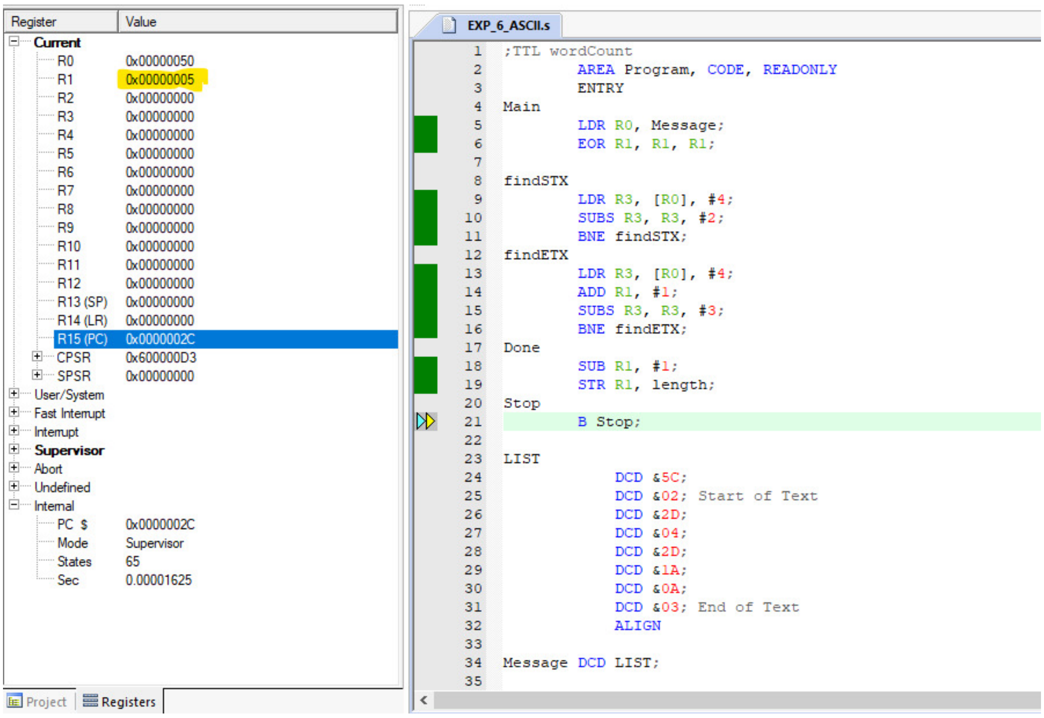

Figure 3: Output(R1 = 0x05) for input given in LIST

3 To check For a given Sequence

3.1 Aim of this experiment:

- Given a sequence of 32-bit words (sequentially arranged) of length 8 (32 bytes or 256 bits), identify and track specialbit patterns of 01111110 in the sequence (if at all appears in the sequence).
- This special bit sequence is called 'framing bits', which corresponds to HDLC protocol.
- Note that this special bit pattern may start at any bit, not neccessarily at byte boundaries.
- Framing bits, allow the digital receiver to identify the start of the frame (from the stream of bits received).

3.2 Approach and Code Algorithm

A list of eight 32-bit words with the 01111110 series is provided to us. We run a programme to count the instances of this series in the list we are provided.

- Designate R3 as the register that stores the quantity of 01111110 occurrences.
- Load the first byte into R7 and run a loop to check each bit.
- Count the right bit occurrences in the sequence and record them. R4 register
- Increment R3 and clear if the 8 bits in the sequence were successfully received. R4.
- Continue until all 256 bits have been examined. So, R3 stores the quantity of occurrences.

3.3 Code

The code used for finding whether the given number is odd/even 3.

```
AREA Program, CODE, READONLY
     ENTRY
Main
     LDR R0, =Bytes
     LDR R1, =BytesEnd
     ADD R1, R1, #1
     LDR R3, =0   ;number of sequences
     LDR R4, =0    ;progress in sequence
     LDR R5, =0   ;amount shifted
     LDRB R7, [R0], #1      ;current byte
     LSL R7, #24       ;put this byte at the left
load_byte
     LDRB R8, [R0], #1
     LSL R8, #16
     ORR R7, R8
     LDR R5, =0

shift
         MOV R9, R7, LSR #31
```

```
        CMP R4, #0
        BEQ startseq
        CMP R4, #7
        BEQ endseq
        CMP R9, #1
        BEQ incstate
        LDR R4, =1
        B noresetstate
startseq
     CMP R9, #0
        BEQ incstate
     B resetstate
endseq
     CMP R9, #0
     BNE resetstate
     ADD R3, R3, #1
     B resetstate
incstate
     ADD R4, R4, #1
        CMP R4, #8
     BNE noresetstate
resetstate
     LDR R4, =0
noresetstate
     LSL R7, #1
     ADD R5, #1
     CMP R5, #8
     BNE shift
     CMP R0, R1
     BNE load_byte
     LDR R2, =Result
     STR R3, [R2]
        SWI &11
Bytes
     DCD &107e0010
     DCD &007e7e01
     DCD &7e000000
     DCD &7e310000
     DCD &1207e000
     DCD &20010010
     DCD &11000000
     DCD &10000700
BytesEnd DCD 0

Length DCD (BytesEnd - Bytes)

     AREA DataRAM, DATA, READWRITE
Result DCD  0
        align
        END
```

Listing 3: Number of occurences of 0x7E

3.4 inputs and outputs

Eight 32-bit words are included in our input list: 0x107e0010,0x007e7e01, 0x7e000000, 0x7e310000, 0x1207e000, 0x20010010, 0x11000000, 0x10000700. Hexadecimal 7e is identical to 01111110. In the aforementioned list, 7e appears six times, hence the outcome is recorded in register R3.

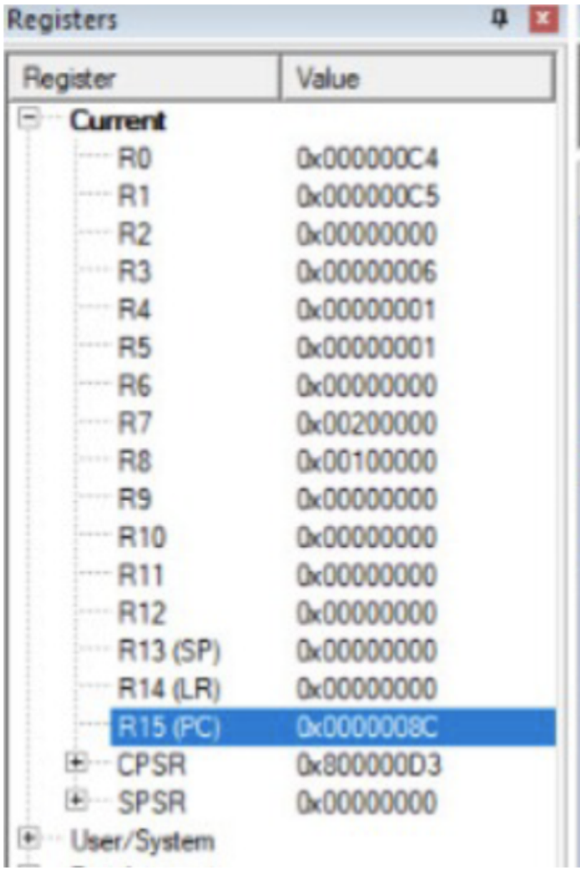

Figure 4: Output