EE306 - Microprocessors

Lab 1 \\ February 17, 2020

*** buraya konu gelecek ***

Erdal Sidal Dogan
#041702023

Alp Gokcek
#31

1 Longest Alternating Strings

Our approach for this problem was to utilize the *XOR* operation. We XORed the input string with another string that consist of alternating 1's and 0's. Given that XOR of two different bits always results in 1, when we XOR the input with the alternating string, longest sequence of 1's in the output should give the longest sequence of alternating bits in the input also. The problem is that we didn't knew wheter the first bit of longest alternating string was 1 or 0. This was a problem because the answer was either the length of longest sequence of 0's or 1's inthe output string.

$$\begin{array}{r} 101101010001 \\ \text{XOR } 101010101010 \\ \hline 000111111011 \end{array} \qquad \begin{array}{r} 101101010001 \\ \text{XOR } 010101010101 \\ \hline 111000000100 \end{array}$$

Above you can see the input string 101101010001 is being XORed with two alternating strings, one of them starts with 1 and other with 0. Using two alternating strings is to address the problem mentioned in previous paragraph, unknown start bits of the longest alternating string. What we did in order to solve this problem is, we used XOR operation for two different alternating strings. We know that the lenght of the longest sequence of 1's in the result of XOR operation is equal to the lenght of longest alternating bits in the input.

Listing 1: Assembly Code for finding longest string of alternating 1's and 0's

```
.global _start
_start:
        LDR R1, TEST_NUM // load the data word into R1
        LDR R3, CONSECUTIVE_ALTERNATING // load the alternating word into R3
        MOV R0, #0 // R0 will hold the result
        EOR R1, R1, R3
LOOP:
        CMP R1, #0 // loop until the data contains no more 1's
        MOV R4, R0
        BEQ LOOP1
        LSR R2, R1, #1 // perform SHIFT, followed by AND
        AND R1, R1, R2
        ADD R0, #1 // count the string lengths so far
        B LOOP

        LDR R1, TEST_NUM // load the data word into R3
        LDR R3, CONSECUTIVE_ALTERNATING1 // load the alternating word into R3
        MOV R0, #0 // R0 will hold the result
LOOP1:
        CMP R1, #0 // loop until the data contains no more 1's
        BEQ END
        LSR R2, R1, #1 // perform SHIFT, followed by AND
        AND R1, R1, R2
        ADD R0, #1 // count the string lengths so far
        B LOOP1

COMPARE_RESULTS:
        CMP R0, R4
        MOVLT R0, R4
        MOV R4, #0
END:
        B END
```

```
TEST_NUM: .word 0x103fe05f
CONSECUTIVE_ALTERNATING: .word 0xAAAAAAAA
CONSECUTIVE_ALTERNATING1: .word 0x55555555
.end
```

2 Parity Bit

Listing 2: caption

```
.global _start
_start:
        LDR R8, TEST_NUM // load the data word into R1
        LDR R10, AND_NUM1
        LDR R11, OR_NUM1
        LDR R12, OR_NUM2
        MOV R0, #0 // R0 will hold the result
        MOV R2, R8
LOOP:
        CMP R2, #0 // loop until the data contains no more 1's
        BEQ CONT
        AND R3, R2, #1
        LSR R2, R2, #1 // perform SHIFT, followed by AND
        ADD R0, R0, R3 // count the string lengths so far
        B LOOP
EVEN:
        AND R8, R8, R10
        ORR R8, R8, R11
        B END
CONT:
        ANDS R3, R0, #1
        BEQ EVEN
        AND R8, R8, R10
        ORR R8, R8, R12
END:
        B END

TEST_NUM: .word 0xAAAAAAAB
AND_NUM1: .word 0x7FFFFFFE
OR_NUM1: .word 0x80000000
OR_NUM2: .word 0x00000001
.end
```