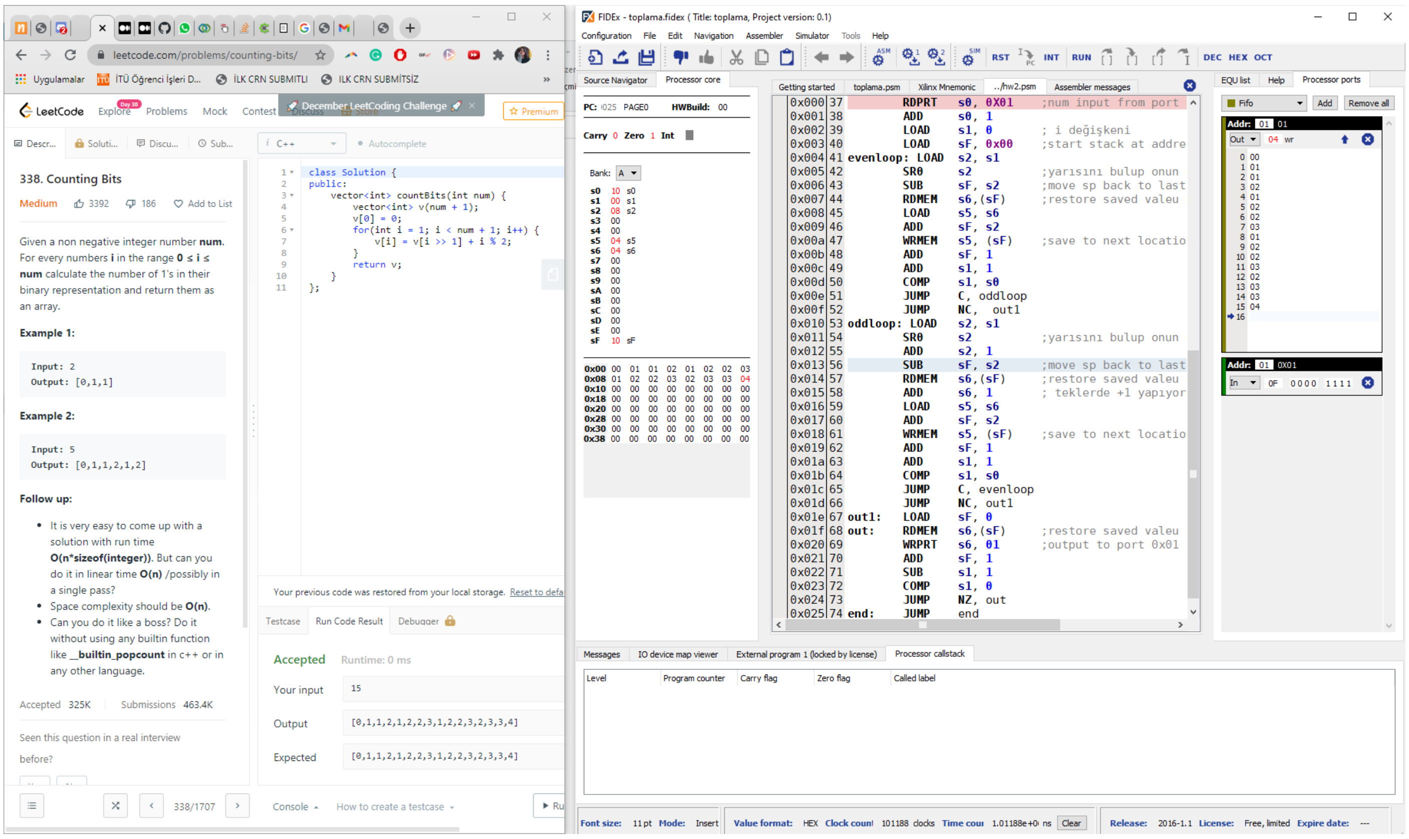
Introduction to Embedded Systems EHB 326E HW#2

ELİF ÖZTÜRK 040170208

SOFTWARE PART:

The problem I have to solve for my software part is: *Given a non negative integer number num. For every numbers i in the range 0 ≤ i ≤ num calculate the number of 1's in their binary representation and return them as an array.*

On the left of Figure 1, you can see the solution with C ++ from the LeetCode site and the answer for the num value of 15. On the right, you can see the version written in assembly language on Fidex. The input and output ports are visible on the right side of the Fidex interface. For input 15, the output "00,01,01,02,01,02,02,03,01,02,02,03,02,03,03,04" is seen, respectively. Also, the registers and scratchpad RAM used are shown on the left side of the interface. As a result, it seems that the outputs match and are correct.



338. Counting Bits

Given a non negative integer number **num**. For every numbers **i** in the range **0 ≤ i ≤ num** calculate the number of 1's in their binary representation and return them as an array.

Example 1:

```
Input: 2
Output: [0,1,1]
```

Example 2:

```
Input: 5
Output: [0,1,1,2,1,2]
```

Follow up:

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**. But can you do it in linear time **O(n)** /possibly in a single pass?
- Space complexity should be **O(n)**.
- Can you do it like a boss? Do it without using any builtin function like **__builtin_popcount** in c++ or in any other language.

```cpp
class Solution {
public:
    vector<int> countBits(int num) {
        vector<int> v(num + 1);
        v[0] = 0;
        for(int i = 1; i < num + 1; i++) {
            v[i] = v[i >> 1] + i % 2;
        }
        return v;
    }
};
```

Accepted Runtime: 0 ms

Your input 15

Output [0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4]

Expected [0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4]

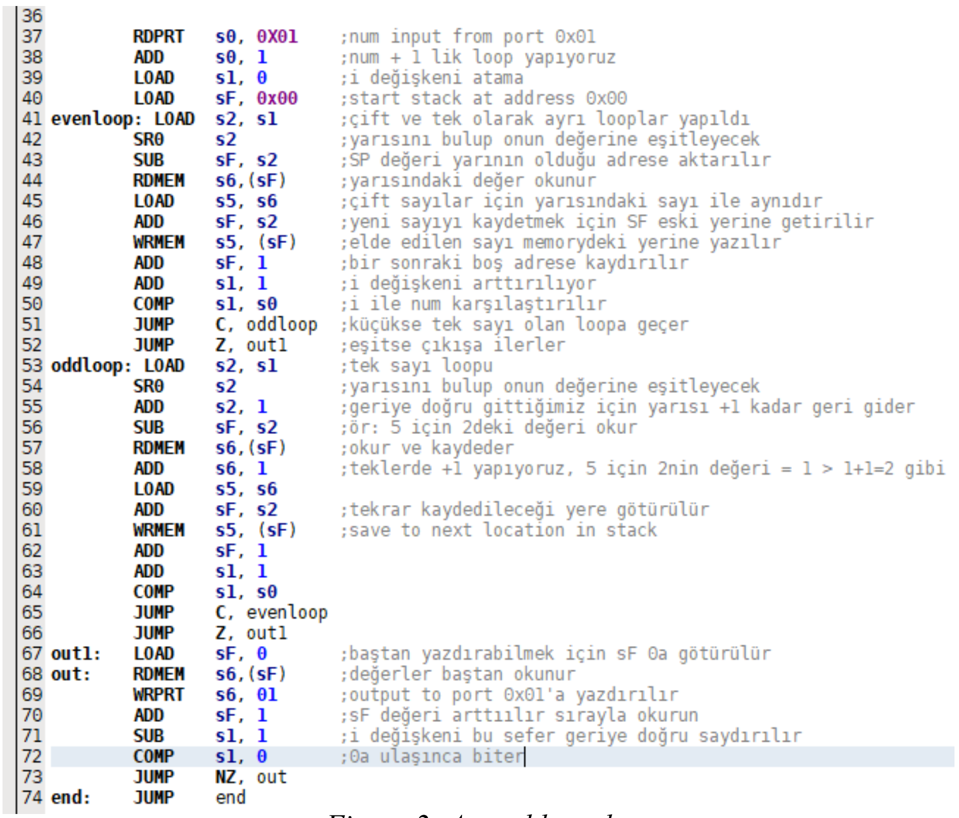
```
0x000 37           RDPRT   s0, 0X01      ;num input from port
0x001 38           ADD     s0, 1
0x002 39           LOAD    s1, 0         ; i değişkeni
0x003 40           LOAD    sF, 0x00      ;start stack at addre
0x004 41 evenloop: LOAD    s2, s1
0x005 42           SR0     s2            ;yarısını bulup onun
0x006 43           SUB     sF, s2        ;move sp back to last
0x007 44           RDMEM   s6,(sF)       ;restore saved valeu
0x008 45           LOAD    s5, s6
0x009 46           ADD     sF, s2
0x00a 47           WRMEM   s5, (sF)      ;save to next locatio
0x00b 48           ADD     sF, 1
0x00c 49           ADD     s1, 1
0x00d 50           COMP    s1, s0
0x00e 51           JUMP    C, oddloop
0x00f 52           JUMP    NC,  out1
0x010 53 oddloop:  LOAD    s2, s1
0x011 54           SR0     s2            ;yarısını bulup onun
0x012 55           ADD     s2, 1
0x013 56           SUB     sF, s2        ;move sp back to last
0x014 57           RDMEM   s6,(sF)       ;restore saved valeu
0x015 58           ADD     s6, 1         ; teklerde +1 yapıyor
0x016 59           LOAD    s5, s6
0x017 60           ADD     sF, s2
0x018 61           WRMEM   s5, (sF)      ;save to next locatio
0x019 62           ADD     sF, 1
0x01a 63           ADD     s1, 1
0x01b 64           COMP    s1, s0
0x01c 65           JUMP    C, evenloop
0x01d 66           JUMP    NC, out1
0x01e 67 out1:     LOAD    sF, 0
0x01f 68 out:      RDMEM   s6,(sF)       ;restore saved valeu
0x020 69           WRPRT   s6, 01        ;output to port 0x01
0x021 70           ADD     sF, 1
0x022 71           SUB     s1, 1
0x023 72           COMP    s1, 0
0x024 73           JUMP    NZ, out
0x025 74 end:      JUMP    end
```

Figure 1: LeetCode and Fidex screen for problem.

In Figure 2, only assembly code is given:

- First of all, "num" variable was named s0, it was requested from port 0x01 with RDPRT. A "< num + 1" loop will be created so that the num value also returns the number 1 it contains.
- The value i is kept at s1 and reset to zero, the sF value is given to 0a as a stack pointer.

- Algorithm works differently for odd and even numbers. The number 1 contained in the binary state of even numbers is equal to what half contains, because multiplying by 2 means adding 0 to the end of the binary number. However, it is different for odd numbers, after finding the half, it is necessary to add by 1. For example, half of the number 7 is 3 species, the number 3 contains 2 numbers 1. Adding 2 with 1, we find the number 1 contained in 7, which is equal to 3.
- Taking the value of the half of the number means to take the SP to the address where half of the number is recorded, therefore first the number i (s1) is transferred to the different register (s2) to avoid confusion, then s2 is shifted to the right and half of the SP goes back half, reads the value here (RDMEM) and loads it to register s5.
- SP is brought back to its old address and saves the value it read halfway to this address. It scrolls to the empty address to go to the new step.
- i (s1) is compared with num (s0). If it is small, Carry (C) is set, it moves to one loop according to this value. If equal, the Zero (Z) flag is set and proceeds to exit.
- There are 2 different operations in a single loop, the first one is the addition in row 55. Since going backwards, 1 more SP is shifted. For example, 5 will be read from 2 but 5-2 = 3 SP is shifted. The other is the addition in row 58, where 1 is added to the value of the half for odd numbers as mentioned above.
- In the out1 branch, SP is taken to 0 so that the values in the ram can be read from beginning to end.
- In the out branch, all values are read one by one from the ram (RDMEM) and given to the exit port (WRPRT) until the value of i (s1) is reset.



```
36
37           RDPRT   s0, 0X01    ;num input from port 0x01
38           ADD     s0, 1       ;num + 1 lik loop yapıyoruz
39           LOAD    s1, 0       ;i değişkeni atama
40           LOAD    sF, 0x00    ;start stack at address 0x00
41 evenloop: LOAD    s2, s1      ;çift ve tek olarak ayrı looplar yapıldı
42           SR0     s2          ;yarısını bulup onun değerine eşitleyecek
43           SUB     sF, s2      ;SP değeri yarının olduğu adrese aktarılır
44           RDMEM   s6,(sF)     ;yarısındaki değer okunur
45           LOAD    s5, s6      ;çift sayılar için yarısındaki sayı ile aynıdır
46           ADD     sF, s2      ;yeni sayıyı kaydetmek için SF eski yerine getirilir
47           WRMEM   s5, (sF)    ;elde edilen sayı memorydeki yerine yazılır
48           ADD     sF, 1       ;bir sonraki boş adrese kaydırılır
49           ADD     s1, 1       ;i değişkeni arttırılıyor
50           COMP    s1, s0      ;i ile num karşılaştırılır
51           JUMP    C, oddloop  ;küçükse tek sayı olan loopa geçer
52           JUMP    Z, out1     ;eşitse çıkışa ilerler
53 oddloop:  LOAD    s2, s1      ;tek sayı loopu
54           SR0     s2          ;yarısını bulup onun değerine eşitleyecek
55           ADD     s2, 1       ;geriye doğru gittiğimiz için yarısı +1 kadar geri gider
56           SUB     sF, s2      ;ör: 5 için 2deki değeri okur
57           RDMEM   s6,(sF)     ;okur ve kaydeder
58           ADD     s6, 1       ;teklerde +1 yapıyoruz, 5 için 2nin değeri = 1 > 1+1=2 gibi
59           LOAD    s5, s6
60           ADD     sF, s2      ;tekrar kaydedileceği yere götürülür
61           WRMEM   s5, (sF)    ;save to next location in stack
62           ADD     sF, 1
63           ADD     s1, 1
64           COMP    s1, s0
65           JUMP    C, evenloop
66           JUMP    Z, out1
67 out1:     LOAD    sF, 0       ;baştan yazdırabilmek için sF 0a götürülür
68 out:      RDMEM   s6,(sF)     ;değerler baştan okunur
69           WRPRT   s6, 01      ;output to port 0x01'a yazdırılır
70           ADD     sF, 1       ;sF değeri arttıılır sırayla okurun
71           SUB     s1, 1       ;i değişkeni bu sefer geriye doğru saydırılır
72           COMP    s1, 0       ;0a ulaşınca biter
73           JUMP    NZ, out
74 end:      JUMP    end
```

Figure 2: Assembly code.