CSE 331

Computer Organization

Fall 2020– 2021

Homework 2 Report

Caner KARAKAŞ

131044061

Part 1 CPP

Explanations

- I define the sets beginning of .cpp file. I'm keeping an array called returnArray. This sequence allows us to keep the subset we will print after. Our static variable named index keeps the index of the array we will print. Our variable named “with” marks whether the number we control in the reversible algorithm is included in this set or not.
- Recursive backtracking algorithm has been applied as required. The target number is scanned in the array starting from the last digit of the given array. scanning is done by comparing the target number with the number to be controlled. If the target number is less than the number we are in, this return passes to the next number without adding to our array. If it is large, the number is marked, thrown into the return sequence, and the function is called again, updating the target number and our check number. If the return value of this function and the sign we use in number marking returns negative, it means that we do not add our number to our return array. In such a case, the added number is subtracted. the target number is restored and our control number is updated and recalled. If the return value of this function does not give the number we want, our sign is updated and our function returns the return value.
- The return points of the recursive function, ie the end points; our target number is zero, which means a positive return, the other is that we don't have a control number, which means a negative return.

BONUS PARTS

- The desired algorithm has been implemented successfully.
- It is written without an auxiliary function and without a loop.
- The requested return sequence has been successfully suppressed.
- The photos were uploaded at the end of the report in comparison with the asm file.

PART 2 ASM

Explanations

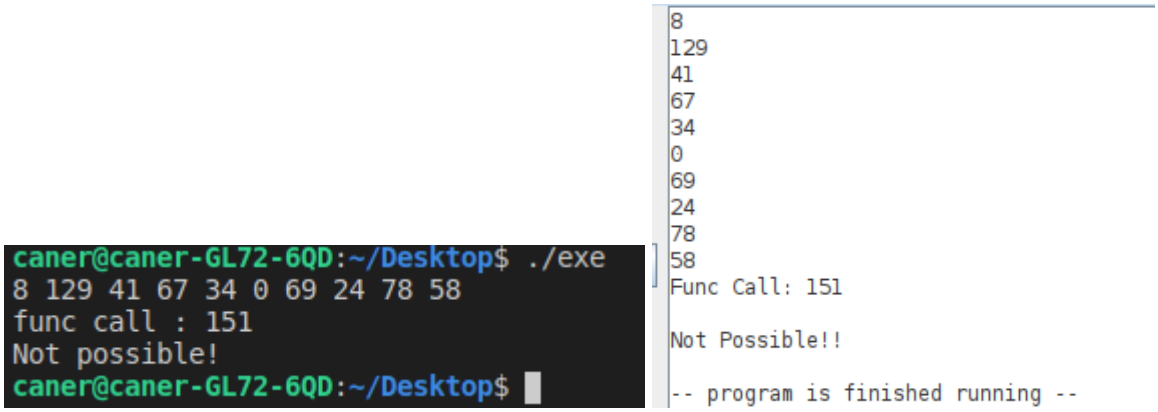
- I define the sets beginning of .asm file. A .space named input_array was created to hold the array to be retrieved from the user. Defined result_array to hold the result array. The required word index was kept in the result_array array. A “with” parameter that marks numbers has been retained as in the cpp code. Word_bytes constant value of 4 was kept to navigate through arrays. The func_call parameter is kept to keep the number of times the recursive function has been called. In addition to these, the fixed strings that we use to provide information to the user are kept.
- **The main function** has been implemented to be the same as the main function in the cpp file. A **read_array** procedure is used that reads values from the user. This procedure takes values from the user. Nested procedure is called here. In this way, the use of stack pointer has been learned. Next, the **CheckSumPossibility** function was called.
- **CheckSumPossibility** function is written to be exactly the same code found in the cpp file. It is simply not implemented in this code, since the values made to -1 in the return sequence are not required. Again, as in the cpp file, it is a recursive function. The return conditions of the function are thought to be exactly the same. helpful skips have been made here. these auxiliary skips are not auxiliary functions. they just give back. These fallback hops are **return, positive_return, negative_return** , **delete_last_element** and finally **first_call**. It does something unrelated to the name **first_call**. It is required to implement the first condition in the cpp code. This condition is compared to our target number and control number. As I said at the beginning, it is exactly the same as the cpp code. there is nothing different or extra here. The return value is stored in the v0 register. Since the target number thought from the user is in register a2 and the size of the array is in register a1, it has continued to be used here.
- After the function is completed, the user is informed of how long the function has been called. Then the results and if a sub-link is found, the sub-click information is given.

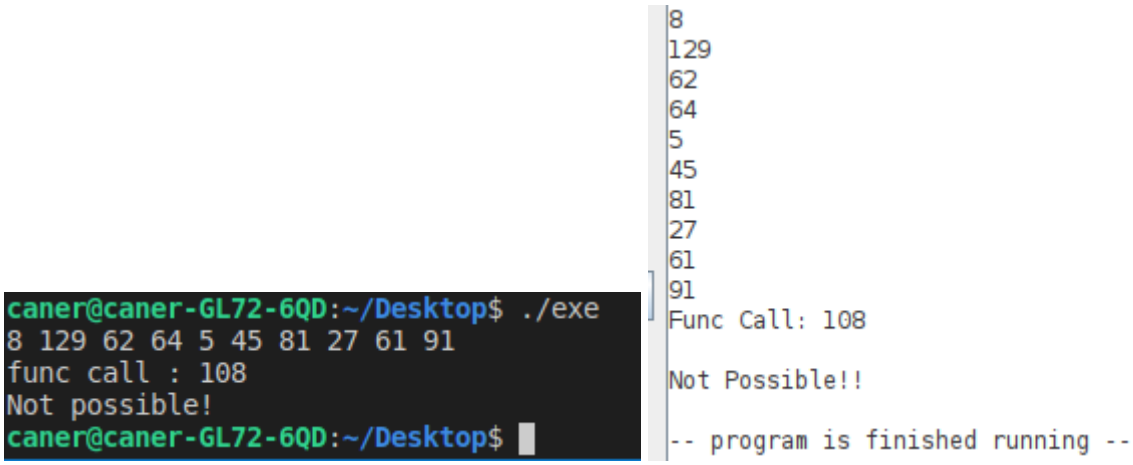
BONUS PARTS

- The requested return sequence has been successfully suppressed.
- Cpp code is exactly the same.
- Code optimization is partially done. Sometimes these situations give better results than sample files and sometimes worse. But it is completely different from non-optimized output. In fact, this situation did not occur in the code I wrote while browsing when all subsets were not found in those outputs.

Missing : Some outputs found no results. Sometimes in the asm code the index got lost and the output could not be obtained.

OUTPUTS (1- cpp output 2- asm output)





```
caner@caner-GL72-6QD:~/Desktop$ ./exe
8 129 3 11 22 33 73 64 41 11
func call : 48
Not possible!
caner@caner-GL72-6QD:~/Desktop$
```

```
8
129
3
11
22
33
73
64
41
11
Func Call: 48

Not Possible!!

-- program is finished running --
```

```
caner@caner-GL72-6QD:~/Desktop$ ./exe
8 129 95 42 27 36 91 4 2 53
Possible!
func call : 121
2 91 36
caner@caner-GL72-6QD:~/Desktop$
```

```
8
129
95
42
27
36
91
4
2
53
Func Call: 121

Possible!!
36
-- program is finished running --
```

```
caner@caner-GL72-6QD:~/Desktop$ ./exe
8 129 92 82 21 16 18 95 47 26
Possible!
func call : 72
26 21 82
caner@caner-GL72-6QD:~/Desktop$
```

```
8
129
92
82
21
16
18
95
47
26
Func Call: 72

Possible!!
82 21 26
-- program is finished running --
```

```
caner@caner-GL72-6QD:~/Desktop$ ./exe
8 129 71 38 69 12 67 99 35 94
Possible!
func call : 3
94 35
caner@caner-GL72-6QD:~/Desktop$
```

```
8
129
71
38
69
12
67
99
35
94
Func Call: 3

Possible!!
35 94
-- program is finished running --
```

```
caner@caner-GL72-6QD:~/Desktop$ ./exe
10 2 9 21 32 2 30 17 28 22 2 12
Possible!
func call : 3
2
caner@caner-GL72-6QD:~/Desktop$
```

```
10
2
9
21
32
2
30
17
28
22
2
12
Func Call: 3

Possible!!
2
-- program is finished running --
```

```
caner@caner-GL72-6QD:~/Desktop$ ./exe
10 12 30 30 17 31 29 30 26 30 10 25
func call : 20
Not possible!
caner@caner-GL72-6QD:~/Desktop$
```

```
10
12
30
30
17
31
29
30
26
30
10
25
Func Call: 20

Not Possible!!

-- program is finished running --
```

```
caner@caner-GL72-6QD:~/Desktop$ ./exe
10 22 19 19 9 22 29 8 31 6 18 30
Possible!
func call : 43
22
caner@caner-GL72-6QD:~/Desktop$
```

```
10
22
19
19
9
22
29
8
31
6
18
30
Func Call: 43

Possible!!
22
-- program is finished running --
```

```
caner@caner-GL72-6QD:~/Desktop$ ./exe
10 42 27 19 6 7 19 12 28 23 6 5
Possible!
func call : 29
5 6 12 19
caner@caner-GL72-6QD:~/Desktop$
```

```
10
42
27
19
6
7
19
12
28
23
6
5
Func Call: 29

Possible!!
19 12 6 5
-- program is finished running --
```

```
caner@caner-GL72-6QD:~/Desktop$ ./exe
10 242 33 24 8 24 6 21 16 20 17 28
func call : 11
Not possible!
caner@caner-GL72-6QD:~/Desktop$
```

```
10
242
33
24
8
24
6
21
16
20
17
28
Func Call: 11

Not Possible!!

-- program is finished running --
```

```
10
172
29
15
3
32
10
31
25
1
1
32
Func Call: 162

Not Possible!!

-- program is finished running --
```

```
caner@caner-GL72-6QD:~/Desktop$ ./exe
10 172 29 15 3 32 10 31 25 1 1 32
func call : 162
Not possible!
caner@caner-GL72-6QD:~/Desktop$
```

```
caner@caner-GL72-6QD:~/Desktop$ ./exe
10 152 21 32 18 29 13 30 10 21 19 6
Possible!
func call : 23
6 19 21 10 30 13 32 21
caner@caner-GL72-6QD:~/Desktop$
```

```
The array is as follows: 21 32 18 29 13 30 10 21 19 6
The target number is: 152
The sequence giving the target number: 21 32 29 13 30 21 6
Possible!
Number of function calls: 85
```