CSE 331

COMPUTER ORGANIZATION

HOMEWORK 2

REPORT

Name: Türker

Surname: Tercan

No: 171044032

Outputs:

Test 1- Passed

```
ttwicer@ttwicer-VirtualBox:~/Desktop$ g++ hw2.cpp -o test
ttwicer@ttwicer-VirtualBox:~/Desktop$ ./test
8
129
3
11
22
33
73
64
41
11
Not Possible!
```

Clear

```
8
129
3
11
22
33
73
64
41
11
Not Possible!
-- program is finished running --
```

Test 2- Passed

```
ttwicer@ttwicer-VirtualBox:~/Desktop$ ./test
8
129
92
82
21
16
18
95
47
26
Possible!
```

```
8
129
92
82
21
16
18
95
47
26
Possible!
-- program is finished running --
```

Test 3- Passed

```
ttwicer@ttwicer-VirtualBox:~/Desktop$ ./test
8 129
95 42 27 36 91 4 2 53
Possible!
```

```
8
129
95
42
27
36
91
4
2
53
Possible!
-- program is finished running --
```

Test 4- Passed

```
ttwicer@ttwicer-VirtualBox:~/Desktop$ ./test
8 129
41 67 34 0 69 24 78 58
Not Possible!
```

Mars Messages | Run I/O

```
8
129
41
67
34
0
69
24
78
58
Not Possible!
-- program is finished running --
```

Test 5- Passed

```
ttwicer@ttwicer-VirtualBox:~/Desktop$ ./test
8 129
62 64 5 45 81 27 61 91
Not Possible!
```

Clear

```
8
129
62
64
5
45
81
27
61
91
Not Possible!
-- program is finished running --
```

Test 6- Passed

```
ttwicer@ttwicer-VirtualBox:~/Desktop$ ./test
8 129
71 38 69 12 67 99 35 94
Possible!
```

```
8
129
71
38
69
12
67
99
35
94
Possible!
-- program is finished running --
```

Code Explanations and Optimizations:

1-

```
1       .data
2   fail:
3       .asciiz "Not Possible!"
4   success:
5       .asciiz "Possible!"
6   array:
7       .space 400      #1000 elements of an integer array should be sufficient
8   maxSize:
9       .word 100       #MAX_SIZE of the array
10
11      .text
12      .globl main
13  main:
14      li $v0, 5
15      syscall
16      move $a3, $v0           #$a3 -> arraySize
17
18      li $v0, 5
19      syscall
20      move $a1, $v0           #$a1 -> num
```

```
int main()
{
        int arraySize;
        int arr[MAX_SIZE];
        int num;
        int returnVal;

        cin >> arraySize;
        cin >> num;
```

- Define “Not Possible!” and “Possible!” strings
- Allocate memory for 100 sized integer array
  which is MAX_SIZE(100) * 4 = 400 bytes
- Take inputs from terminal for arraySize and target number

2 –

```
24  load_array:
25      slt $t3, $t0, $a3       #if (i < arraySize)
26      bne $t3, 1, exit_load_array
27
28      li $v0, 5
29      syscall
30      move $t1, $v0           #The value that is read stored in $t1
31
32      sll $t2, $t0, 2         # Multiply by 4
33      add $t2, $t2, $a2
34      sw $t1, 0($t2)          # cin >> arr[i]
35
36      addi $t0, $t0, 1
37      j load_array
38  exit_load_array:
39      jal first
40      j print_fail
```

```
for (int i = 0; i < arraySize; ++i) {
        cin >> arr[i];
}

returnVal = CheckSumPossibility(num, arr, arraySize);
```

- Array address loaded to $a2 register and $t0 initialized as 0 for loop
- If arraySize is not reached, take integer input from terminal
- Save the integer input to relevant memory space
- Function call to CheckSumPossibility and save the next program counter

3-

```
first:
        addi $sp, $sp, -12      #save the arguments at stack
        sw $ra, 0($sp)          #0(sp) => return address
        sw $a1, 4($sp)          #$4(sp) => target number
        sw $a3, 8($sp)          #$8(sp) => array size
        j checkSumPossibility
```

- Save the arguments to global pointer to use them recursively

4-

```
checkSumPossibility:
        addi $s0, $s0, 1
        lw $ra, 0($sp)          #Load Return Adress
        lw $a1, 4($sp)          #Load Target Number
        lw $a3, 8($sp)          #Load arraySize

        beq $a1, $zero, print_success  #if target is zero program is succ

        addi $t0, $a3, -1       # i = arraySize - 1
        move $t1, $zero         #total = 0
if_not_possible:
        slt $t3, $t0, $zero     #if i < 0
        bne $t3, 0, exit_if_not_possible       # break
        sll $t3, $t0, 2         # $t3 = 4 * i
        add $t3, $t3, $a2       # 4i + arr*
        lw $t4, 0($t3)          # $t4 = arr[i]
        add $t1, $t1, $t4       # total += arr[i]
        addi $t0, $t0, -1       # i--
        j if_not_possible

exit_if_not_possible:
        bgt $a1, $t1, return
```

```
int CheckSumPossibility(int num, int* arr, int arraySize) {
        int total = 0;
        if (num == 0) {
                return 1;
        }
        if (arraySize == 0) {
                return 0;
        }
        for (int i = arraySize - 1; i >= 0; i--) {
                total += arr[i];
        }
        if (num > total) {
                return 0;
        }
```

- Load arguments from global pointer if target num equals to zero, program is successful.
- Otherwise, sum up all elements remaining in the array and check if it is target number is greater than the sum, return. **(Optimization)**

5-

```
        addi $t0, $a3, -1       #arraySize - 1
        sll $t1, $t0, 2         # x4
        add $t1, $t1, $a2       # *arr + [arraySize - 1] * 4
        lw $t2, 0($t1)          #arr[arraySize - 1]

        blt $a1, $t2, save1
```

```
if (num < arr[arraySize - 1]) {
        return CheckSumPossibility(num, arr, arraySize - 1);
}
```

- If the integer at the index arraySize – 1 is greater than target num, do not subtract it from target num. **(Optimization)**

6-

```
save1:                          #CheckSumPossibility)(num, array, arraySize - 1)
        addi $sp, $sp, -12      #push stack to save arguments
        sw $ra, 0($sp)          #return address
        sw $a1, 4($sp)          #target num
        addi $t0, $a3, -1       #arraySize - 1
        sw $t0, 8($sp)
        j checkSumPossibility
```

```
return CheckSumPossibility(num, arr, arraySize - 1);
```

- Before the recursive call save the arguments

7-

```
save2:                          #CheckSumPosibility(num - array[arraySize - 1], array, arraySize - 1)
        addi $t0, $a3, -1       #$t0 = arraySize  - 1
        sll $t1, $t0, 2         #$t1 = 4 * [arraySize - 1]
        add $t1, $t1, $a2       # * arr + $t1
        lw $t2, 0($t1)          #$t2 = array[arraySize - 1]
        sub $t3, $a1, $t2       #$t3 = num - array[arraySiz
        addi $sp, $sp, -12      #push stack to save argumen
        sw $ra, 0($sp)
        sw $t3, 4($sp)
        sw $t0, 8($sp)
        j checkSumPossibility
```

```
int retVal1 = CheckSumPossibility(num - arr[arraySize - 1], arr, arraySize - 1);
int retVal2 = CheckSumPossibility(num, arr, arraySize - 1);
return retVal1 || retVal2;
```

```
return:
        lw $ra, 0($sp)          #load adress from stack
        addi $sp, $sp, 12       #go back to retrieve the arguments
        jr $ra                  #jump to its adress
```

- Save arguments for first recursive call.
- When the a function is finished, return its backwards position with upper function's arguments.

8-

```
print_success:                  #if program is succesfull
        li $v0, 4
        la $a0, success
        syscall
        j terminate

print_fail:                     #if program fails
        li $v0, 4
        la $a0, fail
        syscall
        j terminate

terminate:                      #end the program
        li $v0, 10
        syscall
```

- If target is found, print "Possible!"
- Otherwise, print "Not Possible!"
- Terminate the program

Result:

Optimized for two situations:

- If current element is greater than current target number.
- Sum of all the remaining elements in the array is less than target number, result cannot be found.

Missing Parts:

- I did not find what elements sum up to target number.