FUNDAMENTOS DE COMPUTADORES II – FINAL LAB PROJECT

Name & Surname – Arda Harman

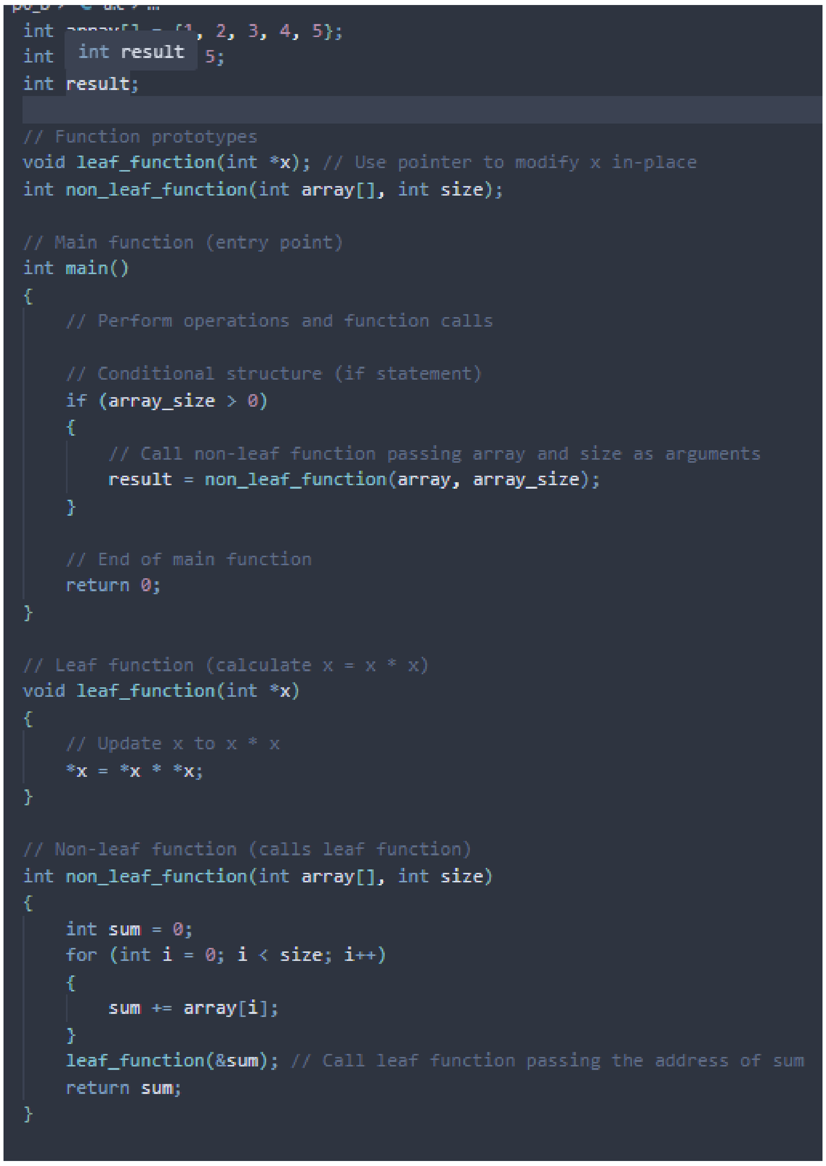
- General description

- Program Structure

General description:

My code is only coded in assembly and I haven’t coded anything in C (since implementing C is optional).

I will put out the C representation of my algorithm to describe the assembly code easier…



```c
int array[] = {1, 2, 3, 4, 5};
int  int result  5;
int result;

// Function prototypes
void leaf_function(int *x); // Use pointer to modify x in-place
int non_leaf_function(int array[], int size);

// Main function (entry point)
int main()
{
    // Perform operations and function calls

    // Conditional structure (if statement)
    if (array_size > 0)
    {
        // Call non-leaf function passing array and size as arguments
        result = non_leaf_function(array, array_size);
    }

    // End of main function
    return 0;
}

// Leaf function (calculate x = x * x)
void leaf_function(int *x)
{
    // Update x to x * x
    *x = *x * *x;
}

// Non-leaf function (calls leaf function)
int non_leaf_function(int array[], int size)
{
    int sum = 0;
    for (int i = 0; i < size; i++)
    {
        sum += array[i];
    }
    leaf_function(&sum); // Call leaf function passing the address of sum
    return sum;
}
```

My algorithm, checka the size of the array with an in condition block

After that, it first “sums up all the elements inside an array” and secondly “takes the square of the sum” and finally stores the obtained result in memory

Project structure:

The project has 3 parts:

- main function
    - Entry point of the code is here and execution starts at main
    - Other functions (“non-leaf function” and “leaf function”) are called from there

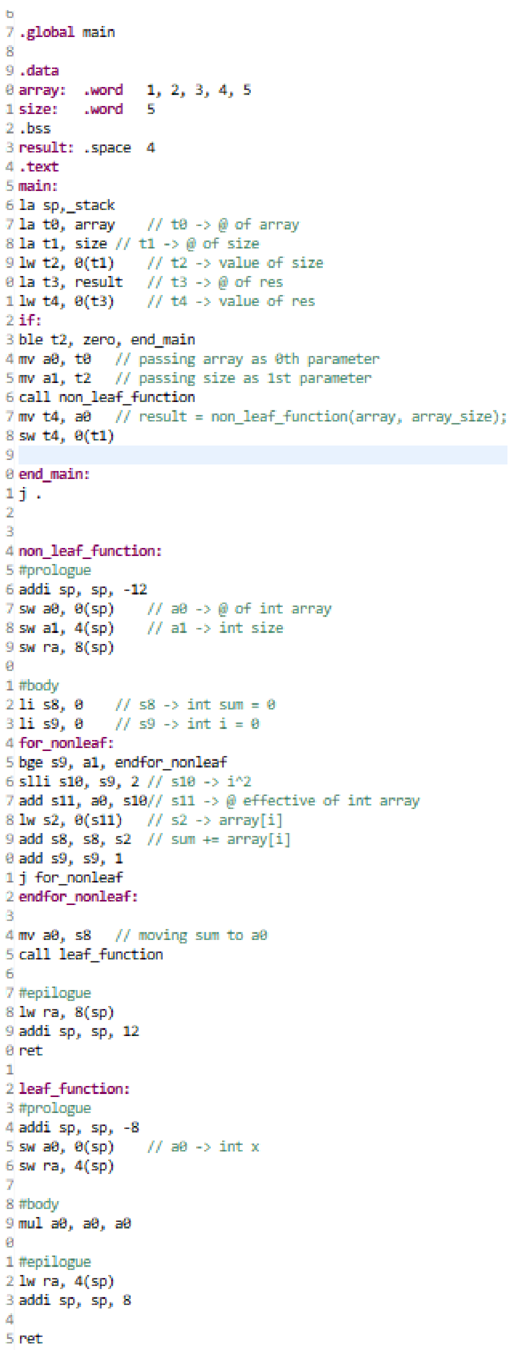
- non-leaf function
    - It sums up all the elements inside the array
        - For this purpose, it inputs “array” and “the size of the array”
    - All the parameters are passed using “ ’a’ registers”
        - 1st parameter -> a0
        - 2nd parameter -> a1
    - This function calls “leaf function” inside of itself

- leaf function
    - It takes the square of the given input
    - It is called by “non-leaf function”
    - Also, this function uses “ ‘a’ registers”

Also, I have 3 variables:

- array -> holds the elements which will be summed up
- size -> value of the length of the array
- result -> the obtained value after the operations (initially it doesn’t have an assigned value)

Here is the assembly code:



```
.global main

.data
array:  .word   1, 2, 3, 4, 5
size:   .word   5
.bss
result: .space  4
.text
main:
la sp,_stack
la t0, array    // t0 -> @ of array
la t1, size // t1 -> @ of size
lw t2, 0(t1)    // t2 -> value of size
la t3, result   // t3 -> @ of res
lw t4, 0(t3)    // t4 -> value of res
if:
ble t2, zero, end_main
mv a0, t0   // passing array as 0th parameter
mv a1, t2   // passing size as 1st parameter
call non_leaf_function
mv t4, a0   // result = non_leaf_function(array, array_size);
sw t4, 0(t1)

end_main:
j .


non_leaf_function:
#prologue
addi sp, sp, -12
sw a0, 0(sp)    // a0 -> @ of int array
sw a1, 4(sp)    // a1 -> int size
sw ra, 8(sp)

#body
li s8, 0    // s8 -> int sum = 0
li s9, 0    // s9 -> int i = 0
for_nonleaf:
bge s9, a1, endfor_nonleaf
slli s10, s9, 2 // s10 -> i^2
add s11, a0, s10// s11 -> @ effective of int array
lw s2, 0(s11)   // s2 -> array[i]
add s8, s8, s2  // sum += array[i]
add s9, s9, 1
j for_nonleaf
endfor_nonleaf:

mv a0, s8   // moving sum to a0
call leaf_function

#epilogue
lw ra, 8(sp)
addi sp, sp, 12
ret

leaf_function:
#prologue
addi sp, sp, -8
sw a0, 0(sp)    // a0 -> int x
sw ra, 4(sp)

#body
mul a0, a0, a0

#epilogue
lw ra, 4(sp)
addi sp, sp, 8

ret
```