Universidad
Carlos III de Madrid

Computer Structure

DEGREE IN COMPUTER ENGINEERING

Assignment 1

Introduction to assembly programming (optional part)

Course 2023/2024

Salvador Ayala Iglesias | 100495832 | 100495832@alumnos.uc3m.es | g89

Inés Fuai Guillén Peña | 100495752 | 100495752@alumnos.uc3m.es | g89

Universidad
Carlos III de Madrid

Table of contents

Factorial function

```
factorial:
	li t0, 1 # total
	beq a0, zero, end_factorial
	mv t1, t0 # count
loop_factorial:
	beq t1, a0, end_factorial
	addi t1, t1, 1
	mul t0, t0, t1
	j loop_factorial
end_factorial:
	mv a0, t0
	jr ra
```

Factorial Matrix

```
factorialMatrix:
	# The function receives 4 parameters: a0 = addressA ; a1 = addressB ; a2 = rows ; a3 = columns
	addi sp, sp, -16 # allocate memory
	sw ra, 0(sp) # push
	sw s0, 4(sp) # push
	sw s1, 8(sp) # push
	sw s2, 12(sp) # push
	mv s0, a0 # store value of A address to manipulate it
	mv s1, a1 # store value of B address to manipulate it
	mul s2, a2, a3 # get the total number of terms of the matrix by multipliying (and store it in s2, will be used as a count)
```

```
loop_SinMatrix:
    ble s2, zero, end_SinMatrix # if count <= zero -> jump to
end_SinMatrix
    lw a0, 0(s0) # load term of the A matrix into a0
    jal ra, factorial # execute sin of the term
    sw a0, 0(s1) # save result of the sin(term)
    addi s0, s0, 4 # move to the next term
    addi s1, s1, 4 # move to the next term
    addi s2, s2, -1 # decrease counter
    j loop_SinMatrix # repeat
end_SinMatrix:
    lw s2, 12(sp) # pop
    lw s1, 8(sp) # pop
    lw s0, 4(sp) # pop
    lw ra, 0(sp) # pop
    addi sp, sp, 16 # free memory
    jr ra # return
```

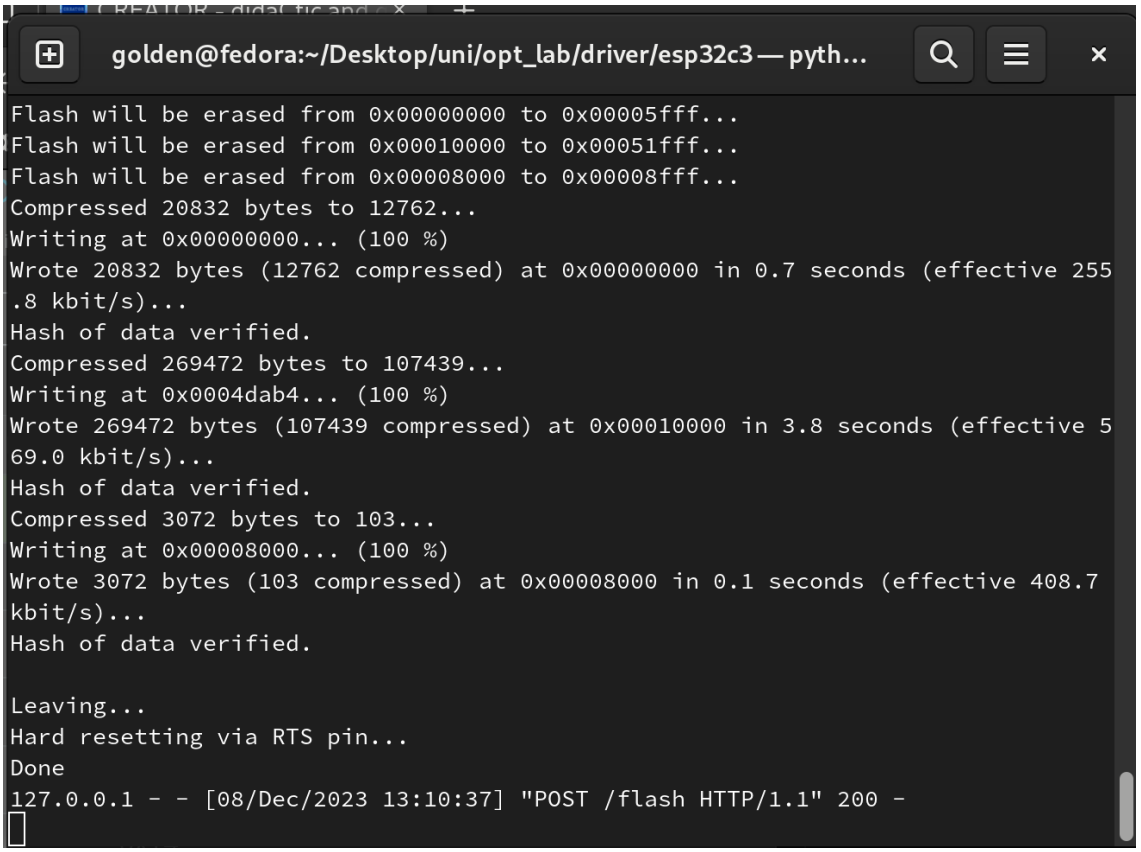
Main developed and test data

```
.data
    A: .word 1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4
    B: .word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    N: .word 4 # Number of rows
    M: .word 4 # Number of columns
```

```
.text
main:
      addi sp, sp, -12 # allocate stack
      sw ra, 0(sp) # push
      sw s0, 4(sp) # push
      sw s1, 8 (sp) # push
      la a0, A # load addresses
      la a1, B # load addresses
      la t0, N # load addresses
      lw a2, 0(t0) # load rows
      la t0, M # load addresses
      lw a3, 0(t0) # load columns

      # the function is executed for the first time
      rdcycle s0
      jal ra, factorialMatrix
      rdcycle s1
      sub s1, s1, s0

      # print the number of cycles (s1)
      mv a0, s1
      li a7, 1
      ecall

      # print "1234" as a quick way of spacing 1st and 2nd
iteration cycles
      li a0, 1234
      li a7, 1
    ecall
```

```
# the function is executed a second time
la a0, A
la a1, B
la t0, N
lw a2, 0(t0)
la t0, M
lw a3, 0(t0)
rdcycle s0
jal ra, factorialMatrix
rdcycle s1
sub s1, s1, s0

# print the number of cycles (s1)
mv a0, s1
li a7, 1
ecall
lw ra, 0(sp) # pop
lw s0, 4(sp) # pop
lw s1, 8 (sp) # pop
addi sp, sp, 12 # restore stack
jr ra
```

Screenshot of the flash operation



Screenshot of the monitor operation

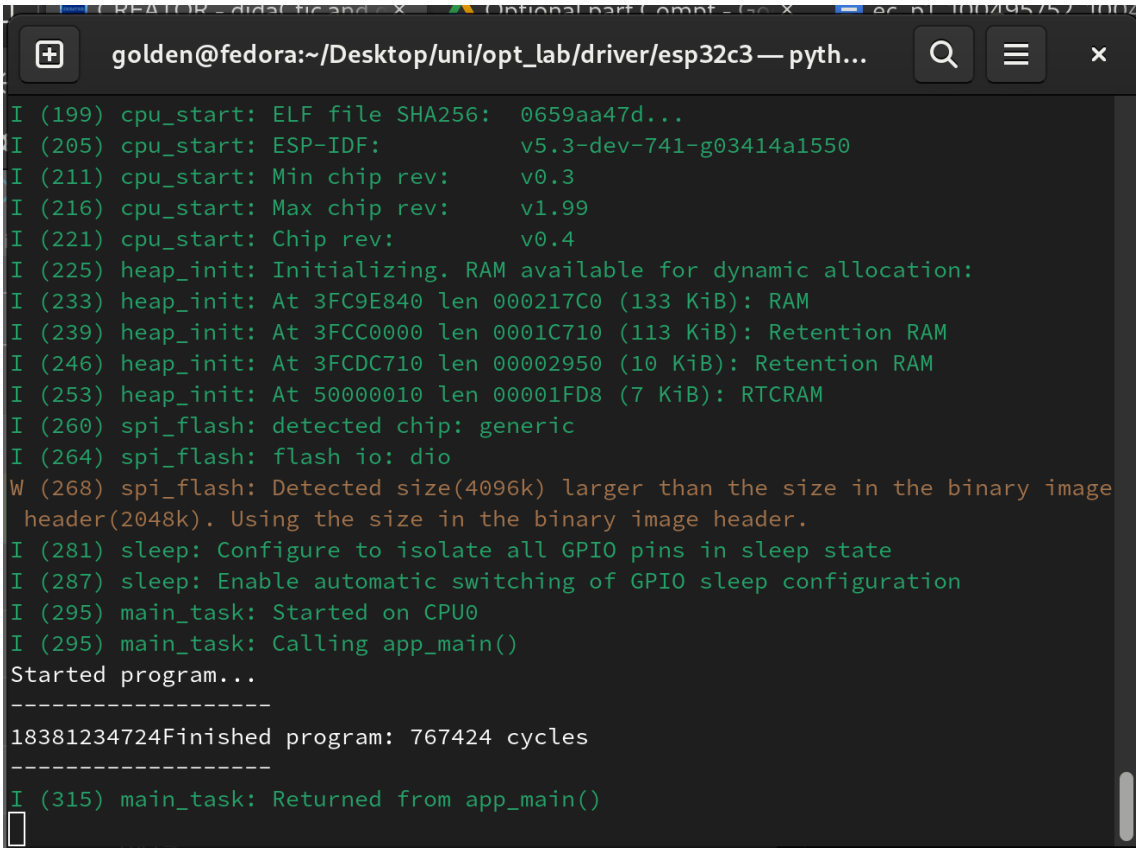


Table with results and analysis

Matrix size	Cycles in Creator (1st time)	Cycles in Creator (2nd time)	Cycles in Esp32-c3 (1st time)	Cycles in Esp32-c3 (2nd time)
4x4	465	465	1838	724
8x8*	1809	1809	3800	2684
16x16	7185	7185	11000	9884
32x32	28689	28689	39972	38622

*For each increase in size, the data segment values stored in the first matrix (A) were duplicated until the desired size was reached. At the same time, the allocation of space for the second matrix was increased to reach the same size.

We can appreciate a significant difference between the number of cycles in Creator with respect to the number of cycles in the ESP32-C3. Not only is the execution in Creator always shorter, but the number of cycles on both iterations are exactly the same for any data input.

The reason for this difference is that Creator does not take into account the cycles needed to access memory for each instruction, while the ESP32-C3 needs to access the memory to take each value of each position of the matrix, calculate the factorial of it and then access the memory position of the other matrix to save it there, needing significantly more cycles than the virtual simulation that Creator uses.

We can also see that the relation between the cycles taken in the first and the second execution, in the ESP32-C3, becomes lower as the size of the matrix increases. The comparison (4x4 matrix) takes more than half of the cycles for the second execution to finish, while on the last one (32x32) the relationship between executions is not even a 10% faster.

The reason for this difference is that the ESP32-C3 has an instruction cache, but not a data cache. Therefore, during the first iteration, the cache is filled with the instructions used, avoiding unnecessary fetches on the second execution. We can appreciate the lack of a data cache by looking at the relation between executions, inversely proportional with the increase in size of the matrices. The cycles it takes to access a lot of memory positions without the cache optimization reflects the importance of a memory cache.

Conclusions

The development of this practice helped us to understand the importance of the cache in a computer structure and the difference between a data cache and an instruction cache. Also, being able to test it with a real physical device helped us to take the abstract concepts learned in class to a more practical and physical understanding of them. Seeing the structure of a microcontroller with our own eyes was such a rewarding experience.

In parallel to this knowledge gain in computer structure, the need of using Linux for the practice helped us to approach this OS we weren't used to working with. This will help us in future courses of our studies and in our professional career.

In conclusion, this practice taught us a lot and should totally be repeated in future years as a way of reinforcing and connecting the knowledge gained along the course.