| Architetture dei Sistemi Di Elaborazione	Delivery date: ==9th December 2022==

Laboratory
8
	Expected delivery of ==lab_08.zip== must include:
	  - zipped project folder
	  - this lab track completed and converted to pdf format.

Download the template project for Keil µVision "*ABI_C+ASM*" from the course material.
==NB: Deliver a single Keil project including Exercise 1 and Exercise 2.==

Exercise 1) – Compute the value of π using the circle method.
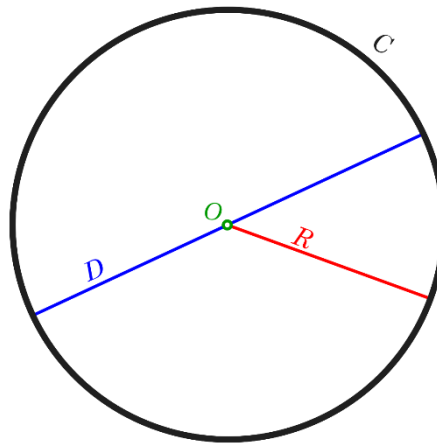
Pythagoreans called a set of points equally spaced from a given origin *Monad* (from Ancient Greek μονάς *(monas)* 'unity', and μόνος *(monos)* 'alone'). As originally conceived by the Pythagoreans, the *Monad* is the Supreme Being, divinity, the totality of all things, or the unreachable perfection by human beings (but we can at least try 😊). The *Monad* (aka circle) in geometry is strongly intertwined with π (a mathematical transcendental irrational constant), commonly defined as the ratio between a circle's circumference and diameter.

$$\pi = \frac{C}{D}$$

An irrational number <u>cannot</u> be expressed <u>exactly</u> as a ratio of two integers.
Consequently, its decimal representation never ends!

3.14159265358979323846264338327950288419716939937510…

However, **mathematical operations in computers are finite**! In the literature, some interesting methods and algorithms to compute an approximated value of π have been developed.

One of the most intuitive methods is the circle method (not the best in terms of performance). It is based on the following observations.
 - The area of the circle is:
$$A = \pi r^2 (Eq.1)$$
 - Where π can be computed as:
$$\pi = \frac{A}{r^2} (Eq.2)$$
 - The assumption is to have a circle of radius **r centered in the origin (0,0).**

Therefore, the Pythagoras theorem states that the distance from the origin is:
$$d^2 = x^2 + y^2 (Eq.3)$$

The cartesian plane can be built thinking of unitary squares centered in every (x, y) point, where x and y are the integers between $-r$ and r.

Squares whose center belongs within or on the circumference contribute to the final area. They must satisfy the following:
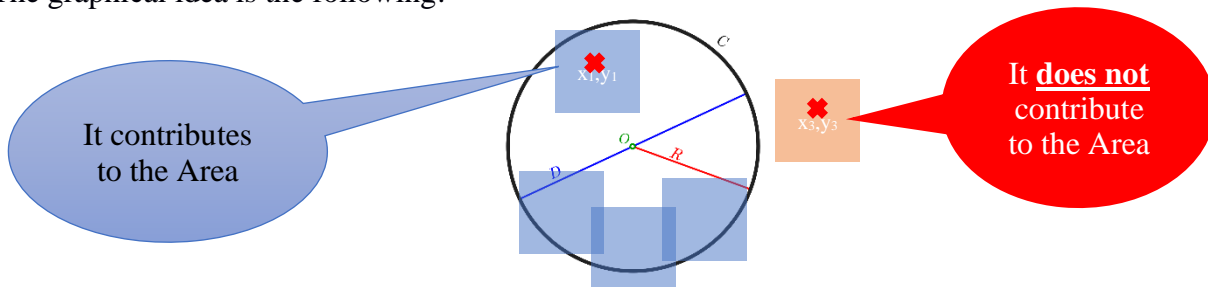
$$x^2 + y^2 \leq r^2 \, (Eq. 4)$$

The number of points that satisfy the above condition (Eq. 4) approximates the area of the circle.

Therefore, the final formula is (where the double sums are the Area):

$$\pi \approx \frac{1}{r^2} \sum_{x=-r}^{r} \sum_{y=-r}^{r} \check{}square(x, y, r) \, (Eq. 5)$$

$$where\,\check{}square(x, y, r) = \begin{cases} 1, \wedge\, x^2 + y^2 \leq r^2 \\ 0, \wedge\, x^2 + y^2 > r^2 \end{cases}$$

The graphical idea is the following:

Declare the coordinates as couples of (x, y) (signed integer) into a read-only memory region 2-byte aligned (into an assembly file) as follows:

```
Matrix_Coordinates  DCD -5,5,-4,5,-3,5,-2,5,-1,5,0,5,1,5,2,5,3,5,4,5,5,5
                    DCD -5,4,-4,4,-3,4,-2,4,-1,4,0,4,1,4,2,4,3,4,4,4,5,4
                    DCD -5,3,-4,3,-3,3,-2,3,-1,3,0,3,1,3,2,3,3,3,4,3,5,3
                    DCD -5,2,-4,2,-3,2,-2,2,-1,2,0,2,1,2,2,2,3,2,4,2,5,2
                    DCD -5,1,-4,1,-3,1,-2,1,-1,1,0,1,1,1,2,1,3,1,4,1,5,1
                    DCD -5,0,-4,0,-3,0,-2,0,-1,0,0,0,1,0,2,0,3,0,4,0,5,0
                    DCD -5,-1,-4,-1,-3,-1,-2,-1, -1,-1 ,0,-1,1,-1,2,-1,3,-1,4,-1,5,-1
                    DCD -5,-2,-4,-2,-3,-2,-2,-2,-1,-2,0,-2,1,-2,2,-2,3,-2,4,-2,5,-2
                    DCD -5,-3,-4,-3,-3,-3,-2,-3,-1,-3,0,-3,1,-3,2,-3,3,-3,4,-3,5,-3
                    DCD -5,-4,-4,-4,-3,-4,-2,-4,-1,-4,0,-4,1,-4,2,-4,3,-4,4,-4,5,-4
                    DCD -5,-5,-4,-5,-3,-5,-2,-5,-1,-5,0,-5,1,-5,2,-5,3,-5,4,-5,5,-5
ROWS                DCB 11
COLUMNS             DCB 22
```

The parsing of `Matrix_Coordinates` must be done in C. Remember that the `extern` keyword must be used for referencing assembly data structures:

```
extern <datatype>  _Matrix_Coordinates;
extern <datatype>  _ROWS;
extern <datatype>  _COLUMNS;
```

In the loop body of Eq. 4, `check_square(x,y,r)` is called using an assembly function with the following prototype:

```
                int check_square(int x, int y, int r)
```

which implements:

$$\check{}square(x, y, r) = \begin{cases} 1, \wedge\, x^2 + y^2 \leq r^2 \\ 0, \wedge\, x^2 + y^2 > r^2 \end{cases}$$

1.1)   Moreover, the Arm Cortex-M3 <u>does not provide hardware floating point support.</u>
Therefore, we can resort to software emulated floating-point algorithms, including the type `float`.
As an example, Arm FPlib has the following EABI-compliant function for float division:

```
float __aeabi_fdiv (float a ,float b)     /* return a/b */
```

<u>https://developer.arm.com/documentation/dui0475/m/floating-point-support/the-software-floating-point-library--fplib/calling-fplib-routines?lang=en</u>

You are required to compute the division with r^2 in (Eq. 5) by calling a second assembly function with the following prototype:

```
float my_division(float* a, float* b)
```

The function body to implement is:
```
my_division:
    /*save R4,R5,R6,R7,LR,PC*/
    /*obtain value of a and b and prepare for next function call*/
    /*call __eabi_fdiv*/
    /*results has to be returned!*/
    /*restore R4,R5,R6,R7,LR,PC*/
```

1.2)   Compute the value of π using a radius of 2,3,5 and store it into a variable.

Radius (r)	Area	Approximated value of π (3 decimal units)	Clock Cycle (xtal=18 MHz)
2	13	3.250	980.75 ns => 17.65 cc
3	29	3.222	988.17 ns => 17.79 cc
5	81	3.240	1011.17 ns => 18.20 cc

Converter from hex to FP (and viceversa): <u>https://gregstoll.com/~gregstoll/floattohex/</u>

Exercise 2) – SuperVisorCall (SVC)
Enhance the program developed in Exercise 1 by using the **SVC** instruction after the computation of π.
Copy part of the template project for Keil µVision "*SVC*".
<mark>You must set the control to `user mode(unprivileged)` and use the Process Stack Pointer.</mark>

2.1) Starting from Exercise 1, write a code to compute a **signature** by calling the SVC with *0xCA*.

In safety-critical domains, such as the automotive one, functional test programs based on CPU instructions are developed to verify the correct in-field behavior of microprocessor-based systems. This safety process is also known as online testing. A possible way to assess if a functional test program has executed its computations correctly is to introduce a signature computation to be checked at the end of the test program.

<u>A signature typically accumulates into one register the others' values by means of an arithmetic operation.</u>
For example, suppose you have `R0, R1, R2, R3, R4, R5`, and you want to compute the signature into `R0`.
```
XOR R0, R0, R1
XOR R0, R0, R2
XOR R0, R0, R3
XOR R0, R0, R4
XOR R0, R0, R5
```

Implement an SVC call to compute the signature between *Registers from 0 to 12, the LR and XPSR* with the XOR instruction.
This must be done by calling the SVC at the end of Exercise 1.

```
__asm__ ("svc 0xca");
```

The computed signature value must be accumulated into R0 and returned from the SVC through the PSP stack (as shown in the figure below).

Stack

	result_here
+28	xPSR
+24	PC
+20	LR
+16	R12
+12	R3
+8	R2
+4	R1
SP→	R0
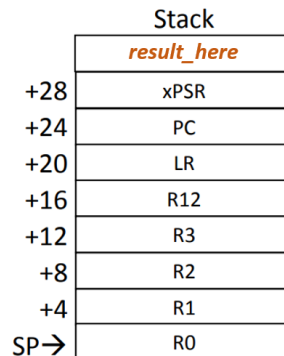
2.2) Starting from Exercise 1, write a code for **memory compaction** by calling the SVC with *0xFE*.

Compute the memory footprint of *Matrix_Coordinates* in bytes in Exercise 1.

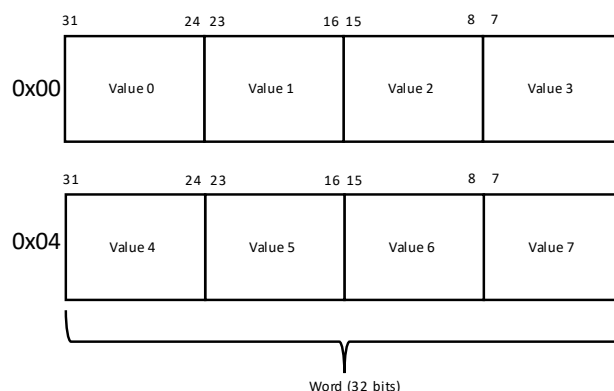| Data	Size [bytes]
Matrix_Coordinates	968 Bytes

In this case, it can be safely assumed that points are always in the range [-128,127].
Therefore, data can be compacted (4 values for each word).

Implement a SVC call to compact *Matrix_Coordinates* and store its compacted version into an empty optimized data structure called *Opt_M_Coordinates* (declared into an assembly file as Read/Write region), you have to calculate the required size!
This must be done by calling the SVC at the end of Exercise 1.

```
__asm__ ("svc 0xfe");
```

The idea is:

31	24 23	16 15	8 7	0
0x00	Value 0	Value 1	Value 2	Value 3

31	24 23	16 15	8 7	0
0x04	Value 4	Value 5	Value 6	Value 7

Word (32 bits)

Then compute the memory footprint.

Data	Size [bytes]	Memory saved [bytes]
Matrix_Coordinates	968 Bytes	0
Opt_M_Coordinates	242 Bytes	726 Bytes