Computer Architectures	Delivery date: 27 November 2024
Laboratory 8 REVALOR RICCARDO s339423	Expected delivery of lab_08.zip must include: - zipped project folders for Exercise1, Exercise2 - this lab track completed and converted to pdf format.

Exercise 1) Experiment the SVC instruction.

- Download the **template project** for Keil µVision "**01_SVC**" from the course material.

Write, compile, and execute a code that invokes an `SVC` instruction in the main (in main.c) as following:

```
call_svc();
```

The `call_svc` is a naked function (LINK) written in assembly (i.e., no save and restore of registers) declared into a code, read only AREA called svc_code. It calls the svc with a fixed SVC number.

Q1: Where is the code of the function when the AREA is declared as READONLY?
Q2: Where is the code of the function when the AREA is declared as READWRITE?

You must set the control to `user mode(unprivileged)` before leaving the Reset_Handler function.

By means of invoking a SuperVisor Call, implement an error-correction method to enable a fault-tolerant data transmission by leveraging **Repetition Coding**. The same bit is transmitted multiple times to increase reliability in data transmission, particularly in noisy communication channels. Instead of sending each data bit once, the bit is repeated several times, allowing the receiver to determine the original bit based on majority voting among the received bits.

For example, suppose that we send the message «10101» with a **repetition code (3,1)**. It will be encoded in the following sequence of data «111000111000111», where each transmitted bit is repeated three times.

Correct Transmission	Received Message
111000111000111	10101
Faulty Transmission	**Received Message**
111010111000101000	10101

It can correct a message when one of the transmitted bits is flipped - a single bit error - meaning that the **correcting ability of this code is 1 bit.**

You are required to implement the following ASM code.

- Use a SuperVisor Call (SVC) to request encoding of a binary message.
- The message to be encoded is stored in the first 8 bits of the SVC instruction (bit 0 through 7).
- In SVC handler, implement the **repetition code (3,1)** on the input message.
- For each bit in the 8-bit message:
  - Repeat each bit **three times** to create an encoded output.
  - This repetition ensures each bit is represented as three identical bits (e.g., "1" becomes "111" and "0" becomes "000").
- Return the encoded message using the stack.

Example:

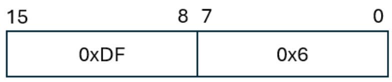


Figure 1: SVC format

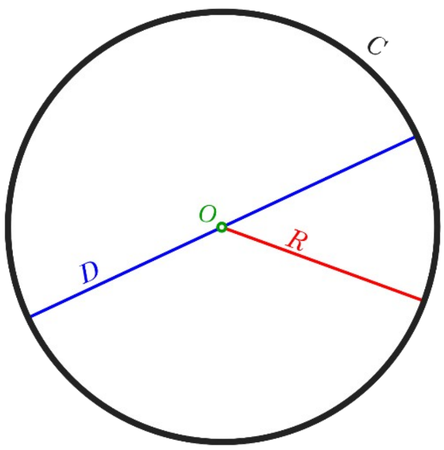
```
SVC  0x5; 2_0101 binary value of the SVC number
```

Returning from the SVC, the process ToS (Top of the stack) must contain the value “000111000111”.

Q3: Describe how the stack structure is used by your project. Where is the return value? Please provide the addresses for the stack structures.

Exercise 2) – Compute the value of π using the circle method.

- <u>Download the template project for Keil µVision “*ABI*” from the course material.</u>



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:

$$x^2+y^2 \leq r^2 (Eq.4)$$

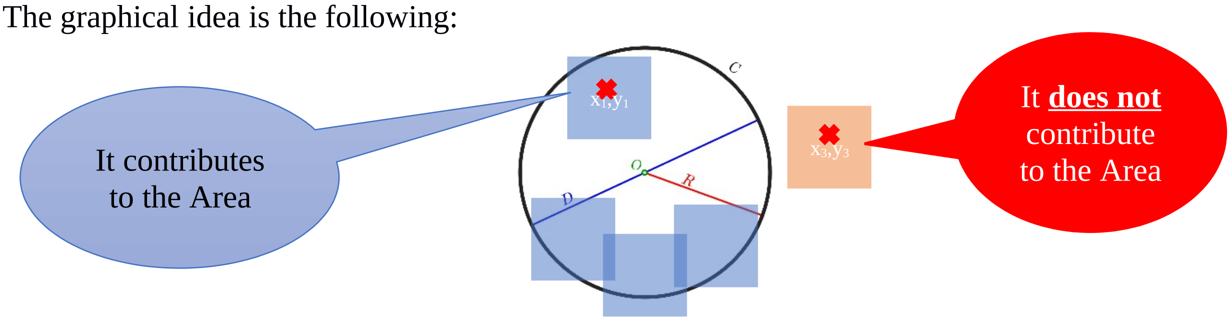
The number of points that satisfy the above condition (Eq. 4) approximates the area of the circle.

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

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

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

The graphical idea is the following:



Declare the coordinates as couples of `(x,y)` (<u>signed integer</u>) 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
```

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;
```

Also remember that any assembly data structure that you want to use in C must be `EXPORT`ed!

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:

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

1.1) Moreover, the Arm Cortex-M3 does not provide hardware floating point support. 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 */
```

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

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	10183
3	29	3.222	10273
5	81	3.240	10549

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