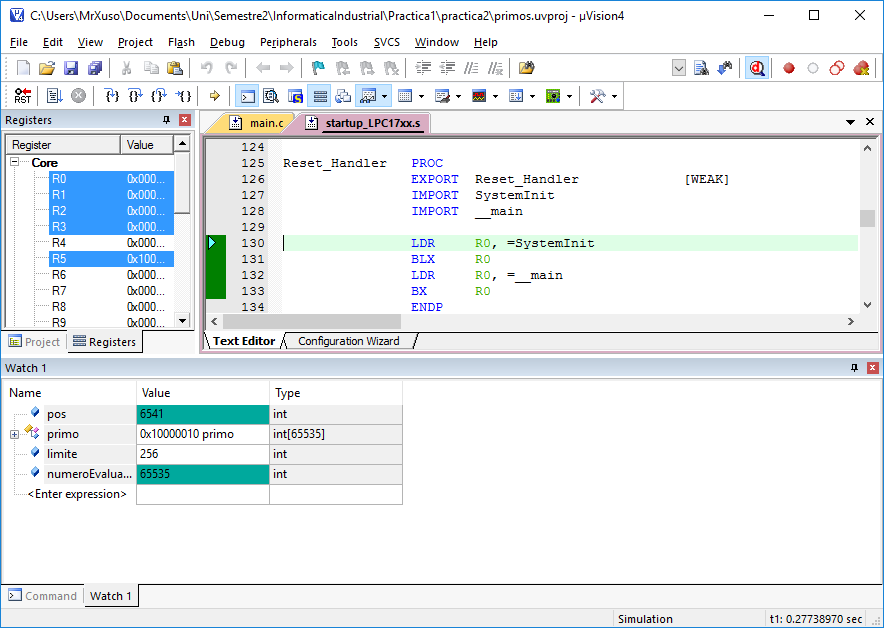
Práctica 2: Entorno de programación

# Introducción:

En el desarrollo de la práctica, se nos pedía calcular todos los números primos comprendidos entre el 2 y el 65535.

Hemos trabajado en el entorno de desarrollo KEIL, simulando en el mismo programa cómo sería la ejecución del programa sobre el LPC1768.

Hemos resuelto el ejercicio de distintas formas, cada una más eficaz que la anterior, hasta que hemos conseguido mejorar el algoritmo para que se ejecute en 0,27738970 segundos.



# Algoritmo:

## Variables:

* int numeroEvaluado -> Número que vamos a evaluar.
* int limite -> Máximo posible entero divisor de un número.
* int primo[] -> Array en el que guardamos los primos.
* int pos ->Variable con la que controlamos la posición del array en la que guardar el primo.
* int esPrimo ->Variable con la que controlamos si el número es primo o no
* int i -> Índice usado en el bucle for para recorrer el array de primos

## Constantes

* NUM\_MAX -> Número máximo hasta el que comprobamos si es primo.c

## Funcionamiento:

El algoritmo final realiza las siguientes operaciones:

1. El dos directamente es el primer número primo del array primo[] y numeroEvaluado empieza en 3.
2. Halla el entero limite con la raíz cuadrada de numeroEvaluado y le sumo 1.
3. Comprueba si hasta el límite hay algún número primo de los obtenido anteriormente que sea divisor de numeroEvaluado.
4. En el caso de que haya algún primo que pueda dividir a numeroEvaluado, entonces éste no es primo. En el caso de que sí lo sea guardo el número en primo[pos] e incremento pos en uno.
5. Incremento numeroEvaluado en 2 para, de esta manera no evaluar número pares, claramente no primos y vuelvo al paso **2** hasta que numeroEvaluado es igual a NUM\_MAX.

## Proceso de obtención del algoritmo óptimo:

Todos los algoritmos de cálculo de primos que hemos ido probando, se pueden divir en dos partes, la parte que evalúa si es primo, y la parte que guarda el primo en un array.

Nuestra primera opción fue la de dividir cada número entre todos los anteriores hasta llegar a \*, pero no era eficiente y calculaba divisiones innecesarias, como puede ser x/6 (ya que si no era divisible entre 2 ni entre 3, no lo sería entre 6).

A raiz de lo anterior, nos dimos cuenta que podíamos dividir sólo entre los primos hallados anteriormente hasta \*.

Por último, en vez de hasta \*, calculamos hasta \*, ya que es el máximo número que puede dividir a nuestro número a evaluar (n).

\*Tratando los resultados de estas operaciones como enteros, truncando la parte decimal del número.

## Otras optimizaciones:

Al obtener limite fuera del bucle for, ahorramos mucho tiempo al no calcular innecesariamente limite sin que haya variado numeroEvaluado.

Aumentar numeroEvaluado de dos en dos para no intentar ver si un número par es divisible entre dos (ya que eso llevaría más tiempo de cálculo que sumar 2 simplemente).