

EXAMEN DE PRÁCTICAS
Convocatoria extraordinaria
ARQUITECTURA DE COMPUTADORES

3o, IST, ITT, IT, 4o IT-ADE y IT-AEROESPACIAL, URJC

Fuenlabrada (Examen remoto), 7 de Julio de 2020

AVISO: Asegúrate que tus programas **cumplen** con los siguientes criterios. Si no se cumple alguno de ellos la **nota máxima** de tu examen será de **2 puntos**

- **Cumplimiento de especificaciones.** Se deben cumplir las especificaciones indicadas en el enunciado: nombres de funciones, nombres de archivos, funcionalidad, etc. Compruébalo antes de entregar el examen
- **Respetar el convenio.** Resuelve las preguntas **sin violar** el convenio del uso de registros (ABI del RISC-V)
- **Sin errores en tiempo de ejecución** (Runtime errors). Tus programas no deben generar excepciones al ejecutarse
- **Sin errores al ensamblar.** Los ficheros entregados NO deben dar errores al ensamblarlos. Si una función la has dejado a medio hacer, asegúrate que al menos se ensambla sin errores

El sistema de comunicaciones de un satélite dado está controlado por un procesador RISC-V. Por temas de ahorro de consumo este procesador **NO** incorpora la unidad hardware para hacer **divisiones** entre números enteros, por lo que **tenemos que hacerlo por software**. En esta fase inicial del proyecto no se está interesado en la optimización de velocidad, sino en lograr la **funcionalidad**. En fases posteriores se realizarán las optimizaciones oportunas

Nuestro jefe de proyecto nos ha asignado la **implementación** de las siguientes funciones, cuya **especificación** es la siguiente:

- *int,int division(a, b)*: Calcular la **división** de los números enteros no negativos a y b (a / b), así como el **resto** de esta división. La función tiene **dos parámetros de entrada**: el número a es el **dividendo** ($a \geq 0$) y el número b es el **divisor** ($b > 0$). La función tiene **dos parámetros de salida**: El primero es el resultado de la división, y el segundo el resto. Por ejemplo, al invoca a *division(7,3)* se obtendrá como resultado el par de números 2 y 1. El primero es el **cociente** y el segundo el **resto** ($2 * 3 + 1 = 7$). Ambos son números enteros no negativos

Esta división se implementará con el **algoritmo iterativo** de restas sucesivas expresado en este pseudo-código:

```

division (a, b):
    cociente = 0
    resto = 0
    while (a >= b):
        resto = a - b
        a = resto
        cociente = cociente + 1

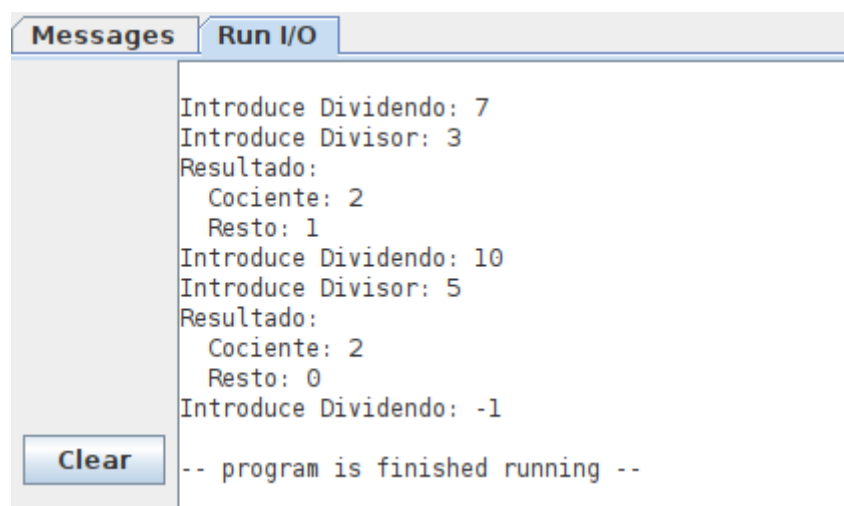
    return cociente, resto

```

- *int divisible(a, b)*: Determinar si el número *b* es divisible entre *a*, donde $a \geq b$, $a \geq 0$ y $b > 0$. La función tiene **dos parámetros de entrada**: El número principal (*a*) y el divisor (*b*). Tiene **un parámetro de salida**, cuyo valor es **1** si **a es divisible entre b** y **0** en caso de **no** serlo. Para implementar esta función se debe llamar a **division()**, que previamente hemos implementado. Sabremos que **a es divisible entre b** si el **resto** de su división es **0**. Supondremos que siempre se va a llamar a esta función con los valores correctos de *a* y *b*. NO hay que implementar comprobación de errores

Además, se definen los siguientes **programas principales** de prueba, para comprobar que las funciones anteriores funcionan correctamente

- *test-division.s*: **Programa** para probar la función de **división**. Se pedirá al usuario que introduzca el dividendo y luego el divisor, y se mostrará en la consola tanto el cociente resultante como el resto. Esta operación se repetirá hasta que el usuario introduzca el valor -1 en el dividendo, con lo que termina el programa. Este es un ejemplo de la salida del programa. El usuario ha realizado las divisiones 7 / 3 y 10 / 5 y luego ha salido introduciendo el -1

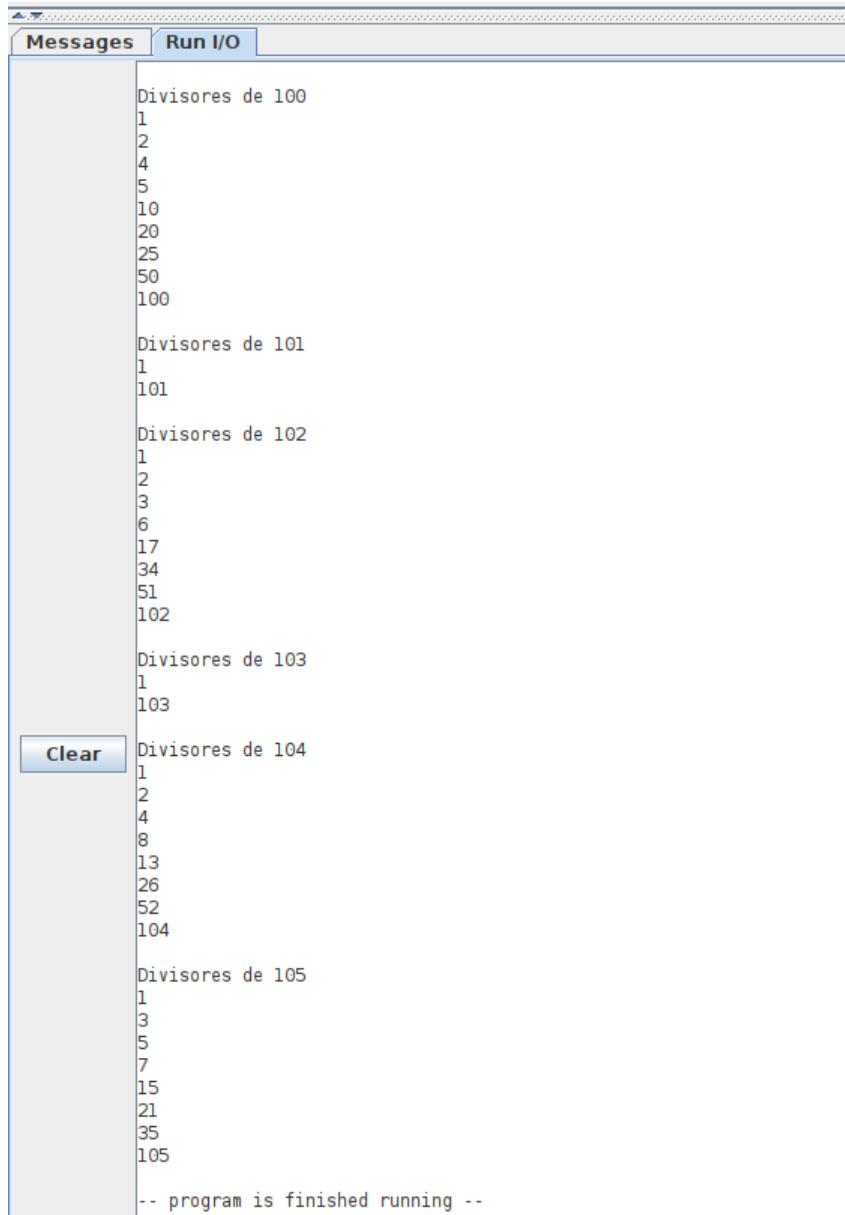


```

Messages Run I/O
Introduce Dividendo: 7
Introduce Divisor: 3
Resultado:
  Cociente: 2
  Resto: 1
Introduce Dividendo: 10
Introduce Divisor: 5
Resultado:
  Cociente: 2
  Resto: 0
Introduce Dividendo: -1
-- program is finished running --
Clear

```

- *test-divisible.s*: **Programa** para probar la función divisible(). Debe calcular TODOS los divisores de los números comprendidos entre el 100 y el 105, ambos incluidos. Para comprobar los divisores de un número n, se utilizará un contador que vaya desde 1 hasta n (algoritmo iterativo), imprimiendo en la consola un mensaje en caso de que n sea divisible entre este contador. La salida de este programa de test **debe ser** la siguiente:



```
Divisores de 100
1
2
4
5
10
20
25
50
100

Divisores de 101
1
101

Divisores de 102
1
2
3
6
17
34
51
102

Divisores de 103
1
103

Divisores de 104
1
2
4
8
13
26
52
104

Divisores de 105
1
3
5
7
15
21
35
105

-- program is finished running --
```

Se pide:

1. Implementar la función *division()* en el fichero **division.s** (2 puntos)
2. Implementar el programa de prueba **test-division.s** (3 puntos)
3. Implementar la función *divisible()* en el fichero **divisible.s** (2 puntos)
4. Implementar el programa de prueba **test-divisible.s** (3 puntos)

Supondremos que el usuario **siempre** introducirá valores correctos, por lo que las funciones y los programas de pruebas **no hace falta** que realicen comprobaciones (por ejemplo, no se comprueba que el divisor sea mayor que 0 para evitar division por cero. Se supondrá que ese caso no va a ocurrir nunca)

Asegúrate de que tus programas sólo funcionan con los 4 ficheros pedidos (no podrás incorporar ningún fichero adicional). **Define todas las constantes que necesites dentro de esos ficheros**