



## Informe Tarea 3

### Arquitectura y Organización de Computadores

#### MCD

En primer lugar se dejó al principio en `.data` espacio para modificar inputs para probar máximos común divisores. El problema en sí se resolvió usando el Algoritmo de Euclides, para ello en primer lugar se identificó el menor de ambos números (almacenados en  $a_0$  y  $a_1$ ), intercambiándolos de ser necesario, de modo que el menor de ellos quede en  $a_0$ .

Luego se creó un loop `mcd` que va buscando el resto de la división de ambos para aplicar el Algoritmo, en caso de ser 0 el resto significa que terminó el proceso; de no ser así, se vuelve a iterar usando como nuevos valores el menor de los antes divididos y el resto calculado.

No es necesario hacer caso base, ya que en caso de no encontrar, simplemente el resto llegará a 1, el cual debe entregar como resto 0.

Al ser 0 el resto, el `beq` dentro del loop lo dirige hacia el final, donde para facilitar la visualización se imprime el valor encontrado (que también queda en  $t_0$ ) y se almacena en la salida creada al principio.

#### Lucas

#### Fibonacci

Para comenzar se creó el label enésimo para recibir el término de la serie que se quiere calcular, el cual almacenamos en  $a_0$ .

Para continuar, definimos los dos primeros términos en  $s_0$  y  $s_1$ , respectivamente y se realizan saltos para entregar el resultado en caso de que esos sean los términos que se busquen.

Luego se utilizó  $t_0$  como un contador, el cual usaremos para verificar si se llegó al término enésimo y  $t_1$  como auxiliar para ir calculando la suma de los términos actuales, para poder así actualizar  $s_0$  y  $s_1$  a los siguientes elementos de la serie.

Finalmente al finalizar el loop se imprime el valor, el cual también queda en  $s_0$  y se guarda en la salida definida al principio.

Cabe destacar que como el MIPS trabaja en 32 bits y para enteros, utiliza el primer bit para el signo, el máximo término que puede calcular es cuando  $n = 46$ , ya que es el último que cabe en  $2^{31} - 1$ .



## Factorial

### Extra

Se utilizó hexadecimal para las entradas para tener una mejor representación de los bits que se ingresan (comparado con decimal).

En primer lugar se usa un contador  $s_0$  y se crean dos binarios con el valor dos (0b10), donde en  $s_1$  se va almacenando una potencia de dos y  $s_2$  se utiliza dos como constante. La idea es crear en  $s_1$  un binario con un uno en el MSB seguido de la cantidad de ceros correspondiente a la cantidad bits que se quieren intercambiar.

Con esta potencia guardada en  $s_1$  se dividirán los números, para que en  $LO$  y en  $HI$  (cuociente y resto) queden: la parte que se quiere incluir en el otro número y la parte que se desea guardar del número actual, dependiendo de cuál término estoy dividiendo. Estos resultados se van guardando entre  $t_0$  y  $t_4$ .

A continuación se amplifica por la potencia la parte que corresponde dejar a la izquierda y se suma la parte que se desea dejar a la derecha de cada número, quedando así los resultados en  $a_0$  y  $a_1$  respectivamente. Específicamente, el cuociente obtenido del primer término es lo que se va sumar al segundo término y el resto se mantiene tal cual. Por otro lado, el cuociente obtenido del segundo término se guarda, se amplifica por la potencia y se le suma lo antes ya mencionado; y el resto de éste se amplifica, pero éste es sumado al resto guardado del primero, para obtener así ambos términos requeridos. Dichos valores quedan en  $a_0$  y  $a_1$  y también se almacenan en las salidas creadas al inicio, en los datos.

Es importante ver que al realizar este proceso se pueden producir problemas con los ceros a la izquierda del número, ya que son ignorados cuando se encuentran al principio del resultado, es por esto que analizaremos los casos posibles. Primero vemos que para los LSB no se generan problemas, ya que el número amplificado llena los dígitos para incluir los ceros que falten (como si se estuviera haciendo extensión de 0 hasta tener los dígitos deseados). Por otro lado cuando haya ceros al comienzo del número en los MSB, la rutina simplemente los ignorará y guardará el resultado con menos dígitos de lo deseado. Es por esto que por ejemplo, si el resultado esperado de un número es 0b0000, en cambio se entrega 0b0, lo que corresponde hacer es solamente una extensión de 0 hasta tener el tamaño que se requiere.