2.- Conversor Decimal a Binario

Contreras R Orlando[1*]

[3]Departamento de Sistemas Electrónicos, UAA, Av. Universidad 940, Aguascalientes, 20131, Aguascalientes, México.

Corresponding author(s). E-mail(s): al348390@edu.uaa.mx;

Abstract

El objetivo de este documento es implementar un conversor de decimal a binario en un ARM, para esto implementaremos el divisor que se realizó en la practica pasada.

Keywords: ASM, Banderas, Carry

1 Introducción

El funcionamiento codigo es el siguiente, se realizará la conversión mediante divisiones sucesivas entre 2, el residuo se irá guardando en una dirección de memoria especificada, cabe hacer mención que cada bit se guardará en un espacio de 8 bits, es decir la dirección podrá contener "00000000" o "00000001".

2 Metodología

2.0.1 Divisor de 2

Para poder convertir un número de decimal a binario es importante considerar las divisiones sucesivas, hay múltiples formas de convertirlo, una de las más populares es dividir el número a convertir en entero entre 2 y considerar el residuo.

Podemos observar que si es un número par, obtendremos de residuo 0, justo como se desea ya que significa que es múltiplo de 2, siendo el 2 la base del sistema binario. En caso de que el número sea potencia de 2 es decir 2^n siendo $n \in \mathbb{N}$, se obtendrán residuos ceros hasta llegar al valor 1. Esto se puede observar en el siguiente ejemplo.

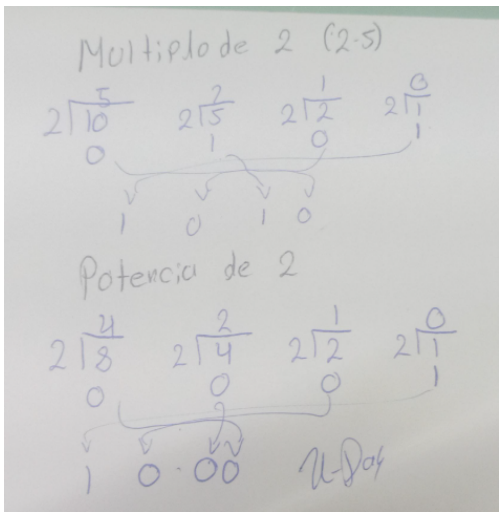

Fig. 1 División binaria

2.0.2 Lógica y análisis

Tomando como base la práctica pasada (Divisor con residuo), se agregó la parte de actualizar el valor a dividir tras cada resta, además de ir guardando cada residuo. En este caso la parte que pusimos de control fue configurando el limite de divisiones posibles, siendo el número de bits 32, lo que se hizo para poder visualizarlo de forma correcta en la memoria fue ir guardándolos desde la última posición a la primera, siendo que cuando nuestro contador que esta iniciado en 32 llegue a 0 saltar directamente al bucle infinito. Dicho comportamiento se puede visualizar mejor en el siguiente diagrama de flujo.

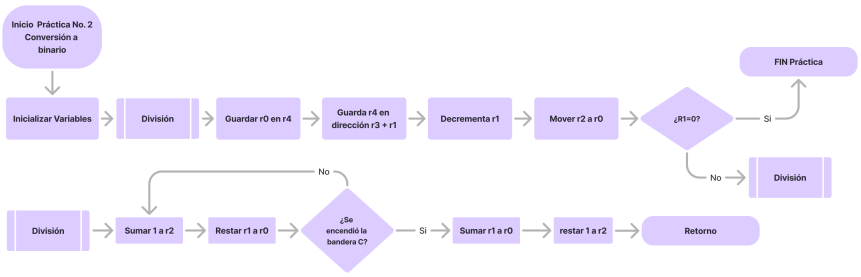

Fig. 2 Diagrama de flujo

Cabe hacer mención que en el código la etiqueta "Brinquitos" fue implementada como etiqueta, y no como función, la diferencia radica en que como función se agrega el program counter actual al Stack Pointer para que cuando éste haga un retorno o blx regrese a la dirección donde se encontraba, de la otra forma tendría que regresarse por medio de saltos.

```
1  brinquitos
2      add     r2,r2,#1
3      subs    r0,r0,#2
4      bpl     brinquitos
5      add     r0,r0,#2
6      subs    r2,r2, #1
7      mov r4, r0
8
9      strb r4, [r3,r1]
10     sub r1,r1,#1
11     mov r0,r2
12     eor r2,r2
13     cmp r1, 0
14     bne brinquitos
```