Universidad Nacional Autónoma de México
Facultad de Ingeniería

Proyecto Final: Convertidor

Estructura y Programación de Computadoras

Ingeniería en computación

Grupo 04

Núñez Quintana Luis Axel

Leyva Mercado Christian Alejandro

Ciudad de México 14 de diciembre, 2021

Índice

1. Credenciales

Figura 1: Credencial alumno Núñez Quintana Luis Axel

Figura 2: Credencial alumno Leyva Mercado Christian Alejandro

2. Criterios de Diseño y Estructura del Convertidor

El programa se divide en varias etapas siguiendo el siguiente diagrama de flujo.

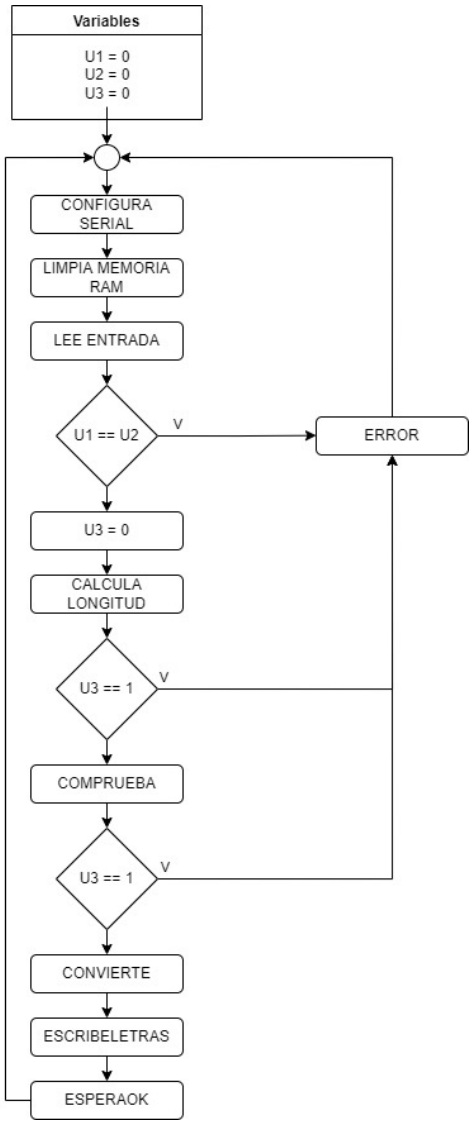

Figura 3: Diagrama de flujo Main

El programa inicia configurando el puerto serial para permitir la entrada de datos vía puerto serial asíncrono, se configura a una velocidad de 9600 baudios considerando se tiene un cristal de cuarzo de 8MHz.

Posterior a este proceso se hace uso de una función elaborada por nosotros llamada limpia memoria RAM, esta subrutina inicializará todas las localidades de RAM entre $0000 y $009D, debido a que estas localidades comprenden las variables que se utilizan en el programa así como aquellas localidades que almacenan caracteres ASCII en el

caso de convertir el número que utilice la mayor cantidad de memoria.

Después se entra a la función lee entrada en la cual por medio de la interrupción por el puerto serial se obtiene un carácter ASCII y se opera siguiendo el siguiente diagrama de flujo.

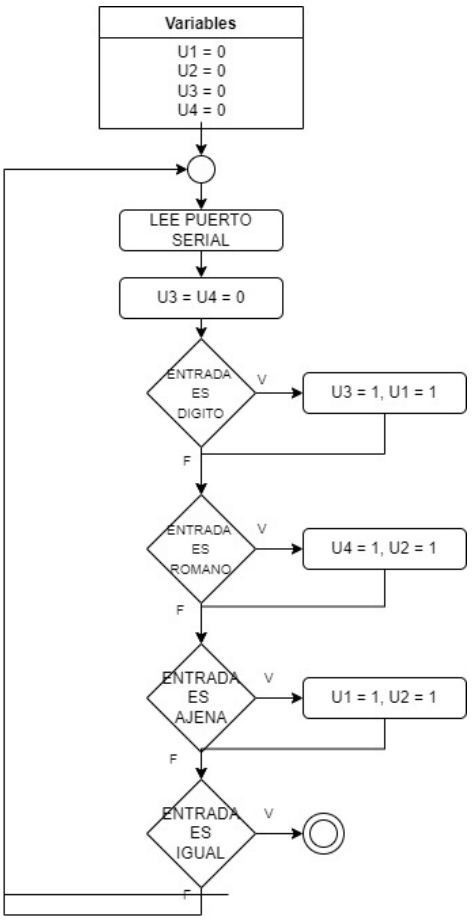

Figura 4: Diagrama de flujo Lee Entrada

Cada carácter se almacenará a partir de la localidad $0050 siguiendo los requerimientos del proyecto hasta almacenar un carácter "=", no obstante cada carácter prenderá una de las banderas U1 o U2 dependiendo de si es un dígito o un número romano. Caracteres ajenos también se almacenan pero prenderán ambas banderas, esto se hace utilizando las banderas auxiliares U3 y U4. Una vez almacenada la cadena si ambas banderas se encuentran prendidas o apagadas se tiene una entrada errónea debido a que solo contiene el carácter "=", es un número que mezcla caracteres romanos y dígitos o se trata de una cadena que contiene caracteres ajenos.

Posterior a este proceso se utiliza la función calcula longitud para capturar errores correspondientes al intervalo que acepta el programa, en caso de números se rechazan

todos aquellos que se encuentren formados por más de 4 dígitos mientras que para números romanos se rechaza toda cadena superior a 22 caracteres.

Después de esto solo queda comprobar el orden de los dígitos o caracteres romanos para poder comenzar a operar con ellos, en el caso de los dígitos se rechaza toda cadena que comience con el carácter 0 y los números romanos se comprueban en el proceso de conversión.

Una vez comprobado los caracteres de un número formado por dígitos o la entrada de una cadena de números romanos correcta, comienza el proceso de conversión.

A continuación se muestra el diagrama de flujo que sigue la conversión de los caracteres:

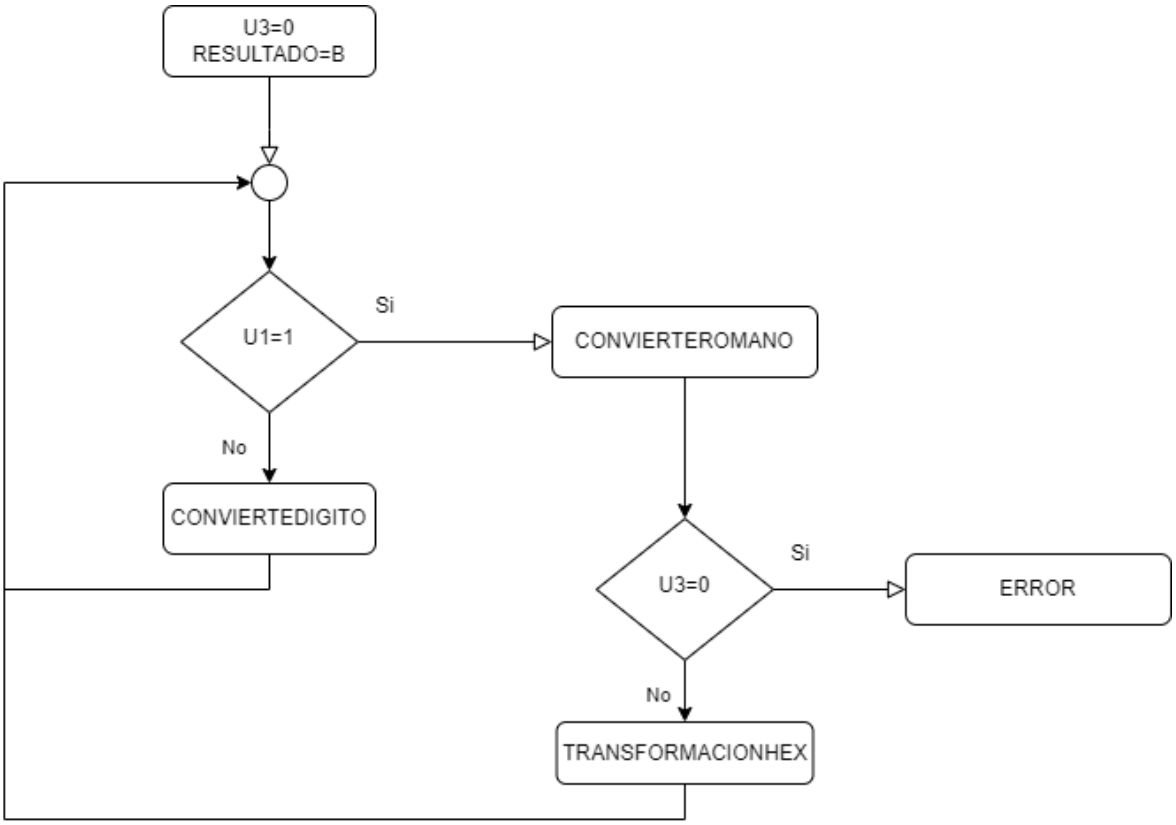

Figura 5: Diagrama de flujo Convierte

Cuando se ejecuta el algoritmo de conversión en la función principal, este primero valida si la bandera **U1** esta encendida, si esta encendida esto significa que el numero a convertir esta escrito en forma romana, por lo tanto se ejecuta la función **CONVIERTEROMANO**. El diagrama de flujo de la función CONVIERTEROMANO es el siguiente:

Como se puede ver en el diagrama (Figura 6 y haciendo zoom), este algoritmo hace la

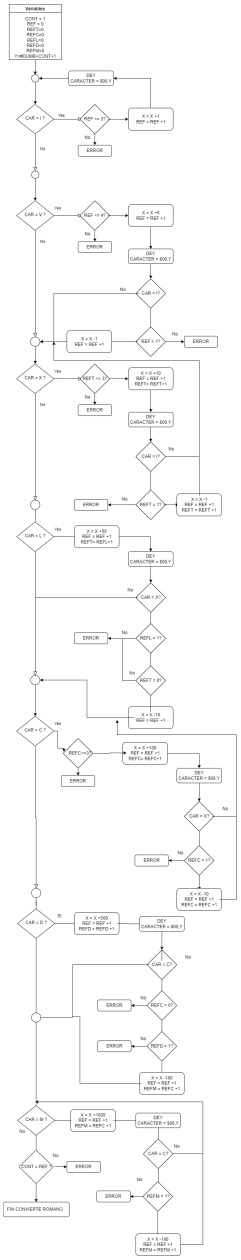

Figura 6: Diagrama de flujo Convierte Romano

lectura de la cadena a convertir de forma inversa (Del final al inicio de la misma), por lo tanto busca que el carácter de la misma sea de I al M (1 a 1000), para hacer esto se carga en Y la dirección DUMB y a esta se le suma el numero de caracteres que se ingresaron mediante el puerto serial (este numero se encuentra almacenado en la variable **CONT**), y mediante avanza el programa se utiliza la operación **DEY** para disminuir Y, al punto que llegue al la ultima dirección de memoria. El numero en decimal se encuentra en un inicio en X ya que dependiendo el numero romano que se lee en memoria, se le suma su

valor a la variable X, por lo tanto al final del programa el numero final se encuentra en X en **Decimal**, sin embargo para convertirlo en carácter ASCII y escribirlo delante del carácter '=', se utiliza la función **TRANSFORMACIONHEX**, la cual convierte el numero en X en carácter ASCII y lo escribe a partir de la dirección de memoria delante del símbolo '='.

A su vez, esta función para convertir números romanos a decimal también valida que el numero este escrito **correctamente** en romano, esto se logra mediante las variables **REF, REFT, REFC, REFL, REFD y REFM**, ya que estas se encargan de verificar que los caracteres no superen el numero permitido ej. solo se pueden poner 3 I's seguidas, el cual equivale a un 3 decimal. Por lo tanto IIII **es incorrecto**. También la misma lógica de la función descarta cadenas de números romanos mal escritos ya que por ejemplo al leer que se encuentra un carácter **V** el programa descarta automáticamente cualquier cadena que incluya posteriormente un carácter **V**.

Ahora, cuando se tiene un numero decimal y se quiere convertir a romano, se utiliza la función **CONVDIGITO** , la cual realiza esta operación. Su diagrama de flujo es el siguiente:

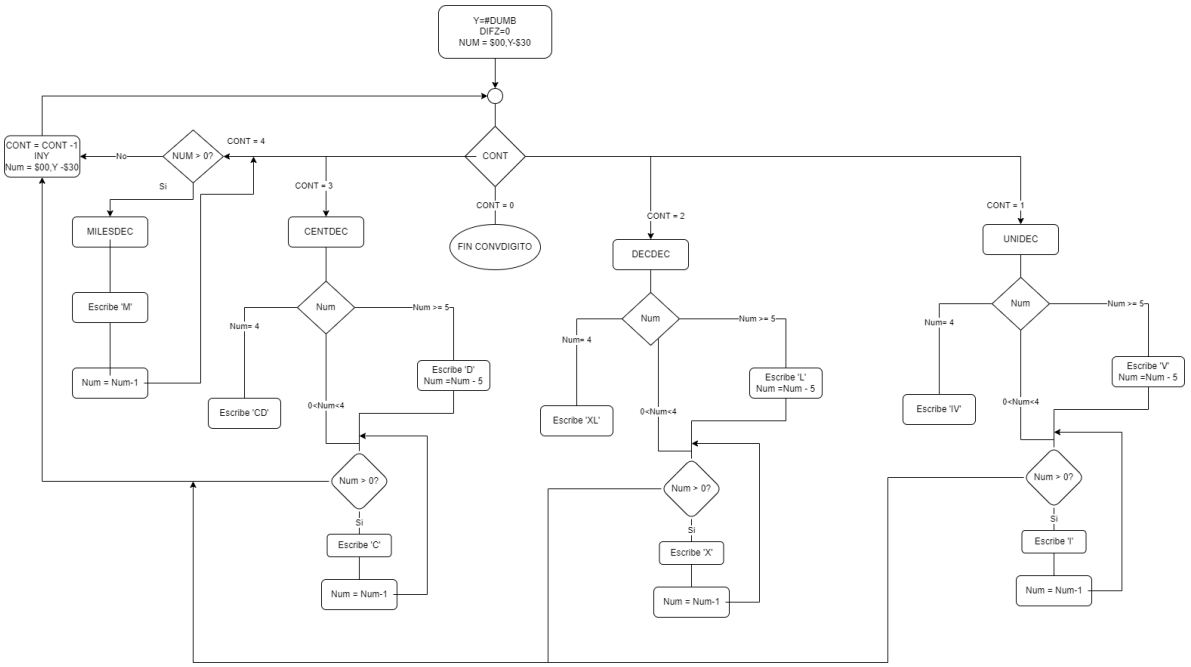

Figura 7: Diagrama de flujo Convierte Decimal

Como se aprecia en el diagrama de flujo, la función **CONVDIGITO** se basa en el numero de caracteres que tiene el decimal a convertir, ya que a partir de ello se empezara a iterar para escribir el numero de izquierda a derecha, es decir desde los

miles a las unidades. La función utiliza la variable CONT la cual contiene el número de caracteres de la cadena a convertir, a esta se le restara 1 por cada iteración, de forma que cuando llegue a 0 se detenga la función. Dependiendo del valor de a variable cont es como se comenzara a escribir el caracter o los caracteres en ASCII, por ejemplo si CONT = 3 y NUM (Carácter en la posicion 4 - CONT del numero a convertir) es 3, se escribirá 'CCC' ya que esta en el lugar de las centenas.

Posterior a la conversión empieza el proceso de transformación a palabras siguiendo la gramática del español. Antes de comenzar el proceso se sitúa un lector en el primer dígito del número en decimal y a la vez se calcula su longitud. Después se lee cada carácter y dependiendo de la longitud y utilizando banderas se obtiene la palabra a escribir a partir de la localidad $0070. El funcionamiento de esta subrutina se muestra en el siguiente diagrama de flujo.

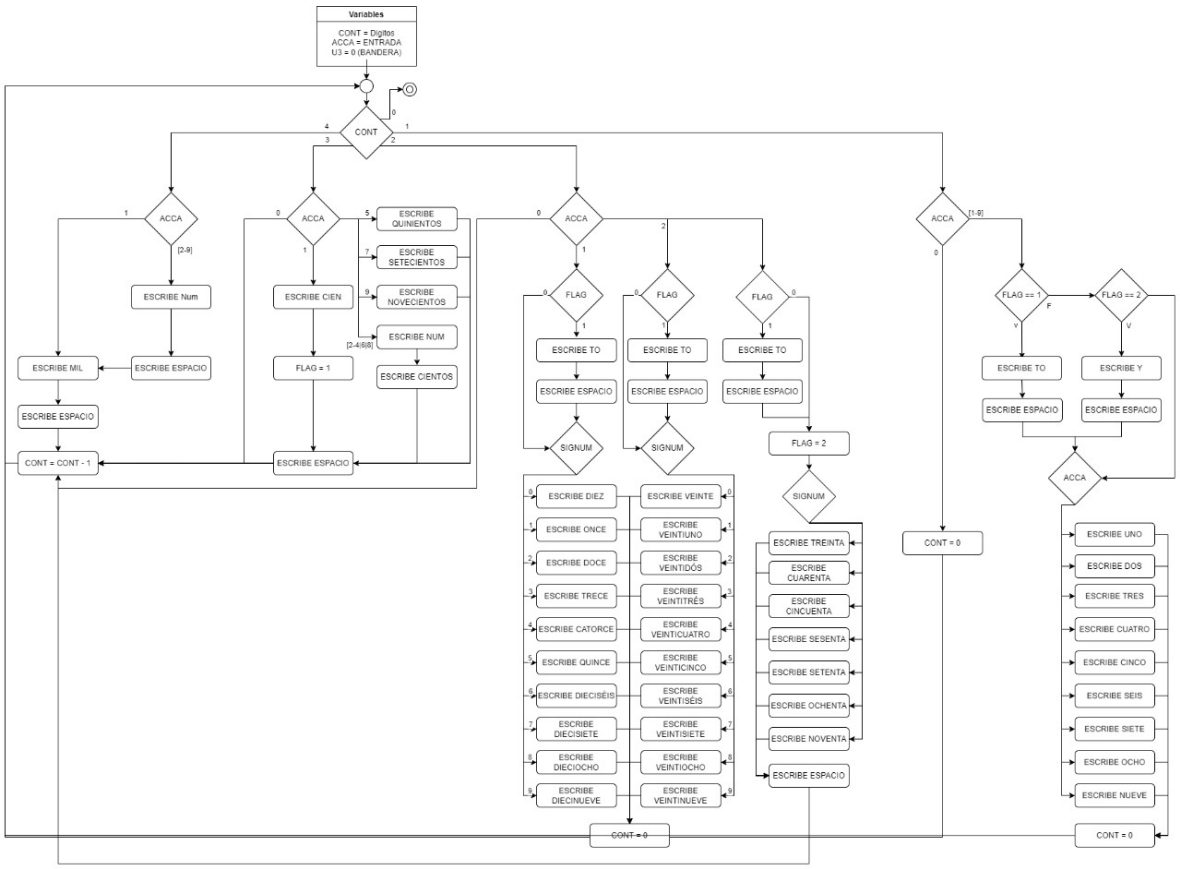

Figura 8: Diagrama de flujo Escribe

Una vez transformado el número, se espera la llegada de la cadena "OK" para repetir el proceso iniciando con la limpieza de memoria RAM. Cabe destacar que en caso de ocurrir un error, se limpian las localidades de memoria RAM a partir de la localidad de

memoria siguiente donde se encuentra almacenado el carácter “=” , después se escribe la cadena de caracteres “ERROR” y finalmente a la conversión exitosa, se espera la llegada de la cadena “OK” para reiniciar el proceso.

3. Evidencias de los puntos planteados en el proyecto

3.1. Rechazo de entradas inválidas

Antes de comprobar la correcta escritura de los números decimales y romanos, el programa revisa los casos en dónde se tengan mezclas de caracteres asociados a números romanos y dígitos, cadenas que contengan caracteres ajenos al programa como la letra "A" y que la longitud de cadenas de dígitos no sea mayor a 4, que la cadena de números romanos no supere los 22 caracteres y la cadena vacía. A continuación se muestra el despliegue de error al obtener cualquiera de los casos antes mencionados.

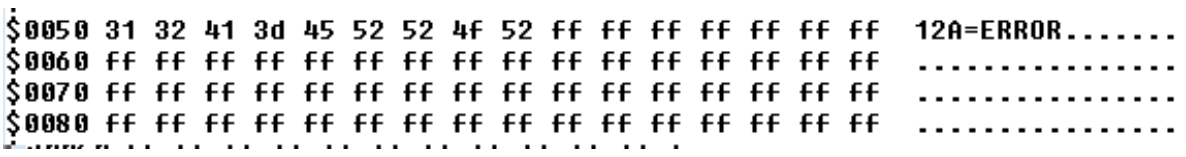

Figura 9: Entrada con caracteres inválidos

Figura 10: Entrada con caracteres mezclados

Figura 11: Longitud errónea número decimal

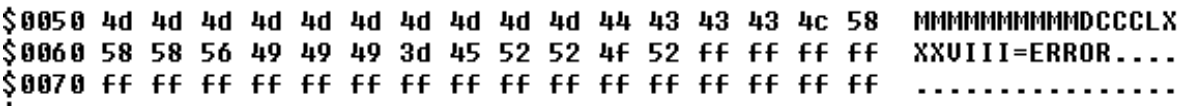

Figura 12: Longitud errónea número romano

```
$0050 3d 45 52 52 4f 52 ff ff ff ff ff ff ff ff ff ff  =ERROR..........
$0060 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
$0070 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
```

Figura 13: Cadena vacía

3.2. Validación de números decimales

Una vez pasada la etapa de comprobación de longitud y caracteres los únicos números que consideramos inválidos son aquellos que contengan el dígito 0 como carácter inicial, esto nos ayuda a rechazar este caso especial y además la entrada del número 0 ya que se encuentra fuera del intervalo.

```
$0050 30 31 32 3d 45 52 52 4f 52 ff ff ff ff ff ff ff  012=ERROR.......
$0060 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
$0070 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
```

Figura 14: Número con 0 inicial

```
$0050 30 3d 45 52 52 4f 52 ff ff ff ff ff ff ff ff ff  0=ERROR.........
$0060 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
$0070 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  ................
```

Figura 15: Número 0

3.3. Validación de números romanos

Como se menciono anteriormente, al momento de convertir un numero romano a decimal en la función **CONVIERTEROMANO** se valida que el numero a convertir este escrito de forma correcta, para comprobar su funcionamiento se probara con la cadena 'XVVIX' la cual tiene una 'X' y una 'V' en lugares que no les corresponden.

En este caso sucede que la función detecta que antes de introducir las 'V', ya se había introducido una 'X' por lo tanto no puede haber ni una sola V después de leer una 'X'. Al detectar una error en la escritura de un numero romano, se envía un mensaje de error y el programa espera a que el usuario ingrese la cadena 'OK' para continuar ingresando mas números e limpiar el memory dump.

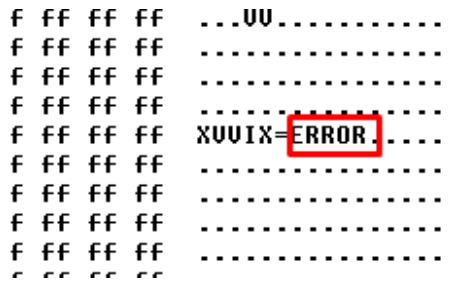

Figura 16: Mensaje de error al escribir un numero romano de forma incorrecta

3.4. Conversión de números decimales a romanos

Como se explico anteriormente, al ingresar un numero decimal y el signo de '=', el programa devuelve el equivalente del mismo en números romanos, esto lo hace en la función **CONVDIGITO**, se comprobara el correcto funcionamiento del mismo ingresando el numero '8654':

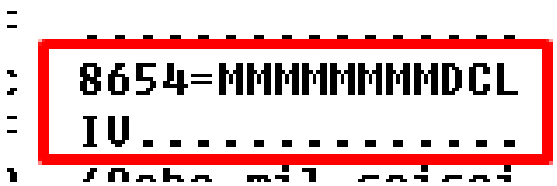

Figura 17: Conversión del numero decimal 8654 a números romanos

Como se puede ver en la imagen, el numero resultante es correcto ya que se tienen 8 M's que representan al 8000, se tiene la cadena 'DC' que representa al 600, la cadena 'L' representando al 50 y finalmente 'IV' representando al 4. Lo cual se logro cuando el programa itero en los 4 caracteres de la cadena, y comprobó la forma de escribir cada carácter en su equivalente romano dependiendo de su valor y posición en la cadena.

3.5. Conversión de números romanos a decimales

Como se comento antes, la función **CONVIERTEROMANO** es la encargada de convertir los números romanos ingresados mediante el puerto serial con el carácter '=' al final del mismo. Para comprobar su funcionamiento se utilizara la cadena 'MMCXV' la cual equivale a '2115':

Como se puede apreciar en la imagen, el numero devuelto por el programa es correcto. Esto sucedió debido a que el programa itero entre los caracteres ingresados (de derecha a izquierda, es decir, desde el carácter 'V' hasta el carácter 'M'). Por lo

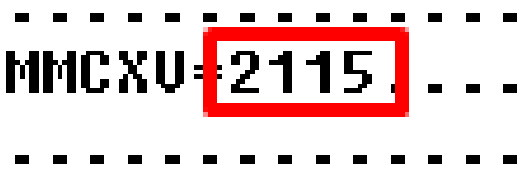

Figura 18: Conversión del numero romano MMCXV a números decimales

tanto mediante el algoritmo explicado anteriormente, fue incrementando el valor de X sumando el valor de cada simbolo con su equivalente en decimal, donde finalmente fue escrito delante del simbolo '=' mediante la función **TRANSFORMACIONHEX**.

3.6. Despliegue de las conversiones a su equivalente en texto

A continuación se mostrará la impresión de un caso general como lo es el número 9233 y posteriormente casos específicos que involucren la diferencia entre la cadena de caracteres cien o ciento, números con acento, números que contengan al número 2 en la posición de decenas o el 5 en la posición de centenas. También para comprobar que no importa el orden de entrada se mostrará el resultado al ingresar las cifras como números romanos y decimales.

```
$0050 39 32 33 33 3d 4d 4d 4d 4d 4d 4d 4d 4d 4d 43 43  9233=MMMMMMMMMCC
$0060 58 58 58 49 49 49 ff ff ff ff ff ff ff ff ff ff  XXXIII..........
$0070 28 4e 75 65 76 65 20 6d 69 6c 20 64 6f 73 63 69  (Nueve mil dosci
$0080 65 6e 74 6f 73 20 74 72 65 69 6e 74 61 20 79 20  entos treinta y
$0090 74 72 65 73 29 ff ff ff ff ff ff ff ff ff ff ff  tres)...........
```

Figura 19: Salida número 9233

Como puede observarse, el número se encuentra encerrado entre paréntesis, separa correctamente las palabras por medio de espacios en blanco, pone como mayúscula la primera letra del número y lo almacena a partir de la dirección de memoria $0070.

```
$0050 39 31 34 30 3d 4d 4d 4d 4d 4d 4d 4d 4d 4d 43 58  9140=MMMMMMMMMCX
$0060 4c ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff  L...............
$0070 28 4e 75 65 76 65 20 6d 69 6c 20 63 69 65 6e 74  (Nueve mil cient
$0080 6f 20 63 75 61 72 65 6e 74 61 29 ff ff ff ff ff  o cuarenta).....
```

Figura 20: Salida número 9140

En este ejemplo se comprueba la correcta escritura de las centenas ya que agrega la cadena "to" en caso de que sigan cifras diferentes de 0 en las decenas y unidades.

En el siguiente ejemplo se valida la correcta escritura de los números que contienen la cadena quinientos, veinti, acento y la entrada como número romano.

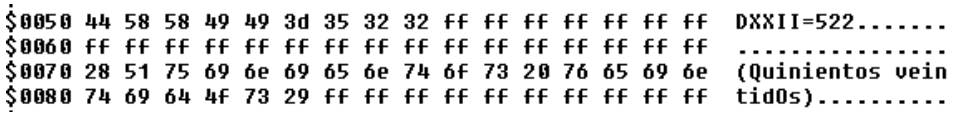

Figura 21: Salida número DXXII

Finalmente se prueba la correcta impresión del número $1100 para comprobar que no agrega la cadena "to".

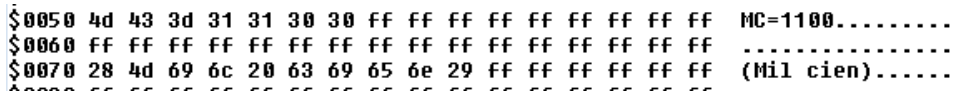

Figura 22: Salida número MC

3.7. Empleo del puerto serial para introducir un número romano

Esta sección puede comprobarse al observar las capturas de los puntos anteriores y posteriores debido a que se observa que la entrada es a partir del serial transmitter.

3.8. Despliegue de mensaje de error

Este punto puede verificarse en la sección "Validación de números romanos y decimales" ya que en las capturas de pantalla se observa la limpieza de memoria y escritura de cadena ERROR después del carácter "=".

3.9. Empleo de OK para dinámica de siguiente número a procesar

Después de mostrar el resultado en pantalla, el programa espera la cadena de caracteres "OK" para reiniciar el programa y esperar el siguiente número a procesar. A continuación se muestra el mapa de memoria así como el puerto serial para observar que no limpia la memoria y espera el siguiente número hasta recibir la cadena antes mencionada.

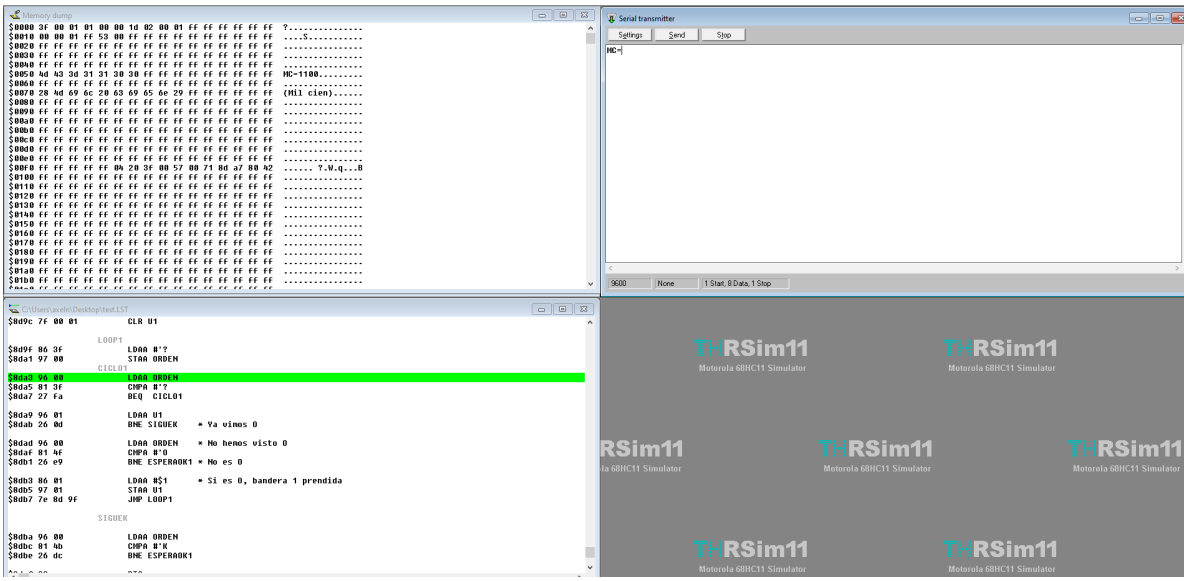

Figura 23: Espera de cadena OK

En la captura mostrada se puede observar cómo el programa se encuentra ciclado por la etiqueta CICLO1, esto forma parte de la subrutina de espera de cadena OK.

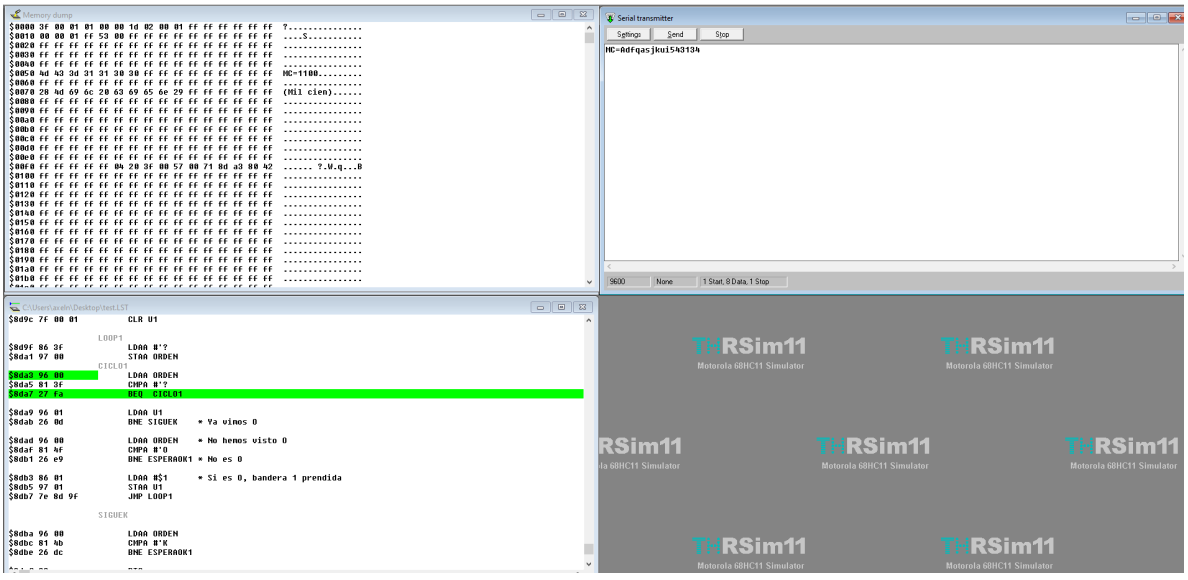

Figura 24: Entrada diferente de OK

En la captura anterior se observa cómo a pesar de recibir la cadena “Adfqasjkui543134” el programa sigue ciclado en la misma subrutina.

Finalmente se observa cómo al recibir la cadena “OK” se realiza una limpieza de la memoria RAM, y el programa ahora se encuentra ciclado en otra sección del programa,

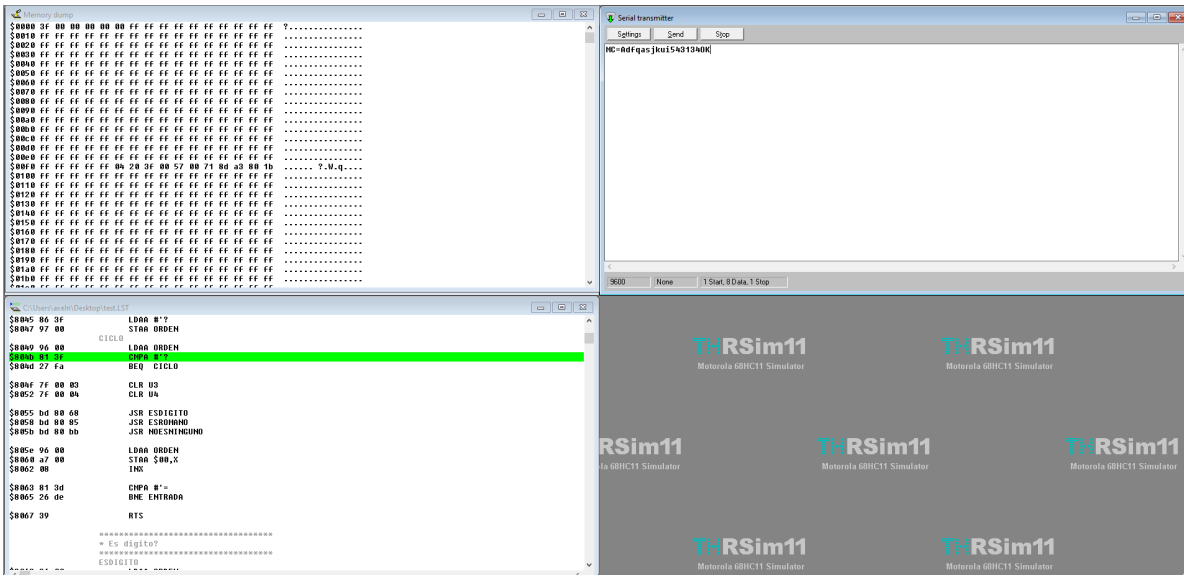

Figura 25: Entrada OK

esto puede observarse al ver la ventana del archivo test.LST ya que ahora se encuentra en una subrutina con etiqueta CICLO la cual forma parte de la subrutina de entrada de números romanos o decimales.