Facultad de Ingeniería

Estructura y Programación de Computadoras

Grupo 2

Proyecto Nº2

Martínez Baeza José Alfonso
Santana Sánchez María Yvette

03 de junio de 2020

Índice

1. Introducción

1.1. Objetivo

Se requiere desarrollar un programa en lenguaje ensamblador para arquitectura Intel x86 que muestre una calculadora de manera gráfica en pantalla.

1.2. Descripción del problema

Para programar una calculadora con gráficos en lenguaje ensamblador es necesario utilizar algunas interrupciones para el modo en el que visualizamos los programas en Dosbox, también se hará uso de algunas macros y procedimientos del proyecto anterior para acelerar el desarrollo de esta calculadora, la dificultad de este proyecto radica en la división de la pantalla para saber en donde se ubica el cursor y donde efectúa el cliqueo, esto implica tener claro cuanto miden los botones utilizados y los espacios que tenemos dentro de la pantalla.

Utilizaremos un código base distinto al proporcionado por el profesor, el código fuente base puede encontrarse en el siguiente respositorio Github:

`https://github.com/Raufoon/Simple-Calculator-Assembly-Language-.git`

1.3. Planteamiento del problema

Desarrollar un programa en lenguaje ensamblador para arquitectura Intel x86 que muestre una calculadora de manera gráfica en pantalla.

- Números del 0 al 9
- Botón suma (+)
- Botón resta (-)
- Botón multiplicación (*)
- Botón cociente (/)
- Botón residuo (%)
- Botón resultado (=)
- Botón AC
- Botón C

Consideraciones:

- Los números deben estar en sistema decimal (dígitos de 0-9).
- Los números deberán ser enteros sin signo.
- Cada número introducido por el usuario puede ser de hasta 4 dígitos. El programa debe restringir que el usuario introduzca más números.

2. Desarrollo

Para la realización de este proyecto nos apoyaremos del uso de macros y procedimientos dentro del lenguaje ensamblador para poder reducir el número de líneas de código, explicaremos a grandes rasgos el funcionamiento de dos interrupciones.

2.1. Interrupciones

Las interrupciones son un método del que disponen los dispositivos e incluso los procesos para hacer notar a la CPU la aparición de alguna circunstancia que requiera su intervención, esto con el fin de que los dispositivos puedan provocar que la CPU deje por el momento o la tarea que estaba realizando y haga la interrupción que se requiere.

Las interrupciones que veremos son tres, aunque la gama de interrupciones es mucho más amplia.

2.1.1. int 21h

La interrupción 21h es una de las mas usadas e importantes para trabajar con el sistema operativo ya que llama o invoca todos los servicios de llamada función DOS, este compuesto por un grupo amplio de funciones, por ello solo mencionaremos la más usadas.

00h: Terminación de Programa.
01h: Entrada de caracteres con eco.
02h: Salida de caracteres.
05h: Salida de impresora.
06h: E/S directa de consola.
08h: Entrada de caracteres sin eco.
09h: Salida de una cadena de caracteres.

No todas las funciones mencionadas se utilizarán en este proyecto, aunque es importante señalar que las funciones como 08h y 01h, es decir, lectura sin eco y con eco respectivamente, se utilizaron en el proyecto anterior.

2.1.2. int 33h

Esta es posiblemente una de las interrupciones que mas usaremos dentro de este proyecto, pues esta la que implica el manejo del ratón, dentro de esta misma interrupción tenemos las siguientes funciones de mayor uso.

00h: Inicializa el ratón.
01h: Muestra el apuntador del ratón.
02h: Oculta el apuntador del ratón.
03h: Obtiene el estado del botón y la posición del ratón.
04h: Establece la posición del apuntador.
05h: Obtiene información del botón presionado del ratón.
06h: Obtiene la información acerca de la liberación del botón
07h: Fija limites horizontales para el apuntador.
08h: Fija limites verticales para el apuntador.
09h: Establece el tipo de apuntador gráfico.
0Ah: Establece el tipo de apuntador en texto.

Las funciones señaladas son las que deberemos tener en cuenta para el siguiente trabajo, como curiosidad esta interrupción trabaja directamente con el registro AX y no el Ah que se usan en las operaciones.

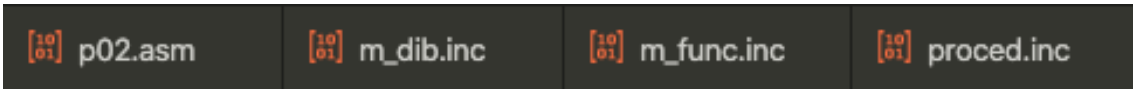
2.1.3. int 10h

Esta interrupción selecciona y activa el modo de video.

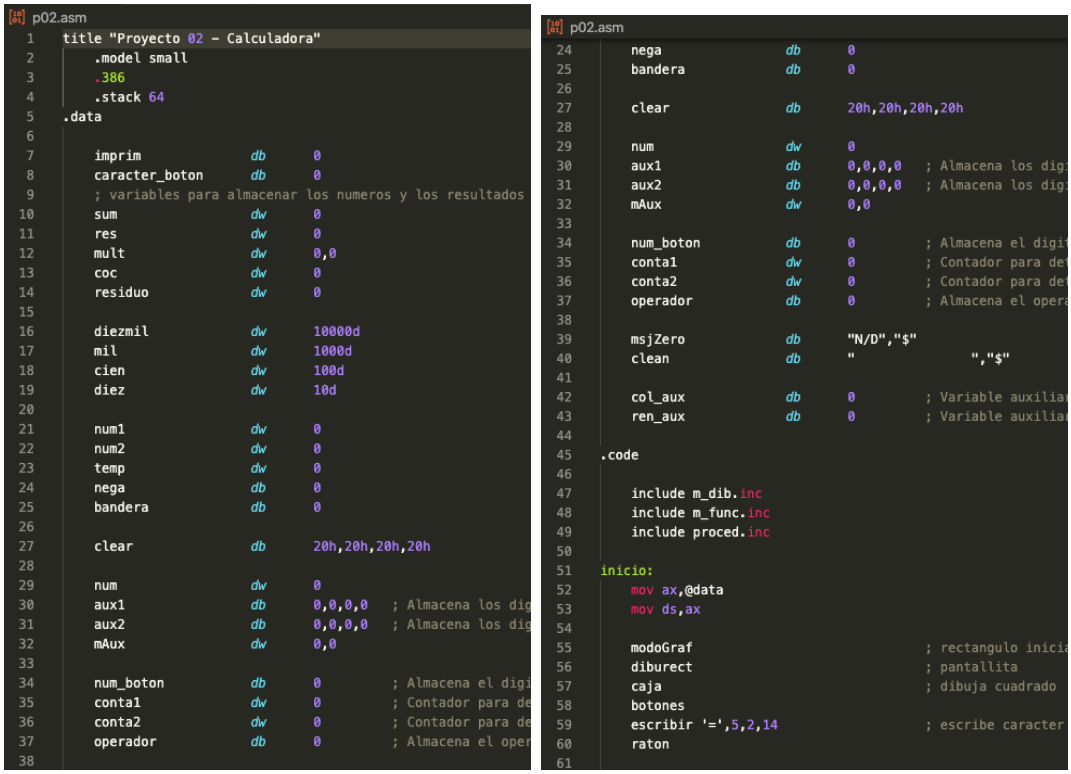
00h: Asignar modo de video
01h: Asignar tipo de cursor
02h: Situar posición del cursor
03h: Leer posición del cursor
08h: Obtener atributo y carácter en el cursor
09h: Escribir atributo y carácter en el cursor
0Ah: Escribir únicamente carácter en el cursor
0Bh: Asignar paleta de colores
0Ch: Mostrar pixel grafico

En este caso se trabaja con el registro AH, como vemos ya tenemos señalado las funciones que se mostraran mas adelante en el proyecto.

2.2. Código



Como podemos notar en la imagen anterior tenemos el proyecto divido en tres archivos: p02, m_dib, m_func y proced, esto con el fin de separar el código para una cómoda lectura no solo de un tercero sino de nosotros mismos para descubrir posibles errores de manera más rápida.



Tenemos las directivas conocidas en el proyecto anterior, en .data se ubican las variables que vamos a utilizar a lo largo del código, aux1 y aux2 será la variable que guarde los 4 dígitos

que se van a introducir por medio del ratón, en .code podemos ver como incluimos dentro del archivo p02.asm los otros componentes (o podemos verlos como bibliotecas locales semejante a C), donde m_dib y m_func contienen las respectivas macros utilizadas dentro del código para hacer el tema grafico mientras que proced contiene procedimientos que acumulan los cuatro dígitos en una variable para luego imprimirla (adaptación del proyecto anterior).

```
inicio:
    mov ax,@data
    mov ds,ax

    modoGraf                          ; rectangulo inicial
    diburect                          ; pantallita
    caja                              ; dibuja cuadrado
    botones
    escribir '=',5,2,14               ; escribe caracter en pantalla
    raton
```

Dentro de la etiqueta inicio, que pertenece a la directiva .code, podemos ver como comenzamos el programa, con su respectiva reserva de segmento, posteriormente se hace una llamada a las macros en este caso modoGraf,diburect, caja, botones y ratón.

A continuación, analizaremos dichas macros (iremos por ello al apartado m_dib.inc) al bloque de código perteneciente a modoGraf.

```
modoGraf macro
    mov ah,0
    mov al,13H
    int 10H
endm
```

Imaginemos que solo tenemos el código anterior ejecutado en el inicio, entonces al ejecutarlo en DOSBox encontraremos los siguiente:

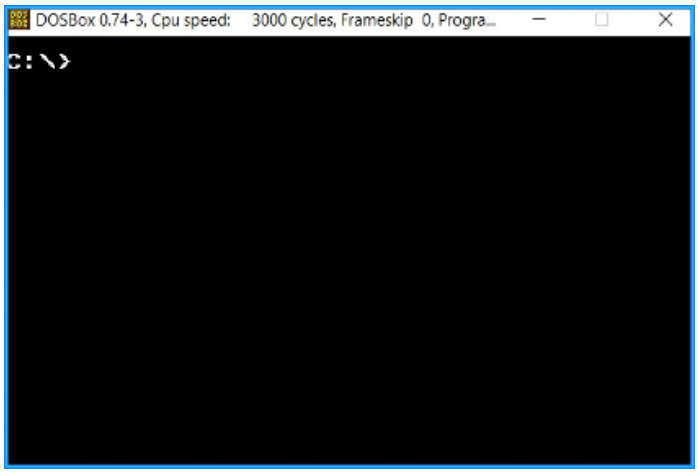


Figura 1: Entra en modo gráfico

Con este apartado de código iniciamos el modo grafico con la interrupción 13h.

En la siguiente imagen veremos las macros de diburect, caja.

2.2.1. Interfaz gráfica

```
; Dibujamos el rectangulo de afuera
diburect macro
    limpiar
    lineahor 4,254,4,1                ; Primero alarga la linea horizontal (bajo) corto (alto)
    limpiar
    lineahor 4,255,194,1
    limpiar
    lineaver 4,194,4,1
    limpiar
    lineaver 4,194,254,1
    limpiar
endm

; Dibuja caja
caja macro
    limpiar
    lineahor 9,249,9,7                ; El ultimo valor es el color
    limpiar
    lineahor 9,250,69,7               ; El primero es hacia arriba
    limpiar
    lineaver 9,69,9,7                 ; El segundo hacia abajo
    limpiar
    lineaver 9,69,249,7               ; La tercera linea divisioria de altura
    limpiar
endm
```

En diburect, primero limpia la pantalla, posteriormente genera dos líneas horizontales y otras dos líneas verticales.

Nuevamente para ejemplificar mejor el desarrollo de este programa, mostraremos paso a paso como se desarrolla la transformación gráfica.

En la macro tenemos la siguiente línea de código:

- lineahor 4,254,4,1

  Parámetro 1: La extensión de la línea horizontal hacia la izquierda.
  Parámetro 2: La extensión de la línea horizontal hacia la derecha.
  Parámetro 3: La posición de la línea horizontal más arriba o abajo.
  Parámetro 4: Determina el color.

- lineaver 4,254,4,1

  Parámetro 1: La extensión de la línea vertical hacia la izquierda.
  Parámetro 2: La extensión de la línea vertical hacia la derecha.
  Parámetro 3: La posición de la línea vertical más arriba o abajo.
  Parámetro 4: Determina el color.

Estos códigos llaman a su vez a otra macro que determina el dibujo del rectángulo.

```
lineahor macro X,Y,Z,C
    LOCAL L1

    mov cx,X
    mov dx,Z

L1:
    mov ah,0CH
    mov al,C
    int 10H
    inc CX
    cmp CX,Y
    jnz L1
endm
```

```
lineaver macro X,Y,Z,C
    LOCAL L2

    mov cx,Z
    mov dx,X

L2:
    mov ah,0CH
    mov al,C
    int 10H
    inc DX
    cmp DX,Y
    jnz L2
endm
```

Figura 2: Dibuja marco de la calculadora

Al agregar la macro caja al código dibujaremos nuevamente otro rectángulo valiéndonos de las macros utilizadas anteriormente líneahor y lineaver.

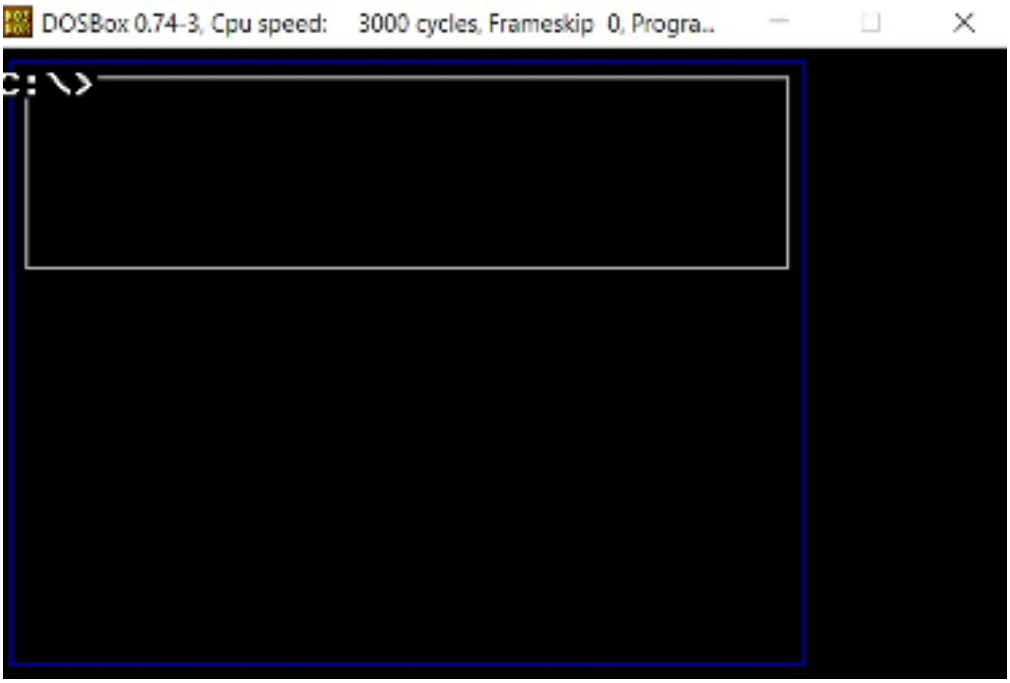


Figura 3: Dibuja caja donde imprimirá las operaciones

Un detalle importante al dibujar los rectángulos que deseamos debemos siempre tener en cuenta que ese código debe ir después de ejecutar el modo gráfico, sino el dibujo no se realizará.

```
botones macro
    ; botones izquierdos
    dibujarboton 74,9,15
    dibujarboton 104,9,15
    dibujarboton 134,9,15
    dibujarboton 164,9,40

    ; botones de en medio
    dibujarboton 74,59,15
    dibujarboton 104,59,15
    dibujarboton 134,59,15
    dibujarboton 164,59,15
    dibujarboton 164,109,40

    dibujarboton 74,109,15
    dibujarboton 104,109,15
    dibujarboton 134,109,15

    dibujarboton 74,159,100
    dibujarboton 104,159,100
    dibujarboton 134,159,100
    dibujarboton 164,159,100

    ; botones derechos
    dibujarboton 74,209,100
    botontipo 104,209,10
```

```
    escribir '0',22,9,15
    escribir '1',18,3,15
    escribir '2',18,9,31
    escribir '3',18,16,15

    escribir '4',14,3,31
    escribir '5',14,9,31

    escribir '6',14,16,31

    escribir '7',10,3,31
    escribir '8',10,9,31
    escribir '9',10,16,31

    escribir 'x',10,22,100
    escribir '/',14,22,100
    escribir '-',18,22,100
    escribir '+',22,22,100

    escribir '%',10,28,100

    escribir '=',18,28,10

    escribir 'A',22,3,40
    escribir 'C',22,4,40

    escribir 'D',22,15,40
    escribir 'E',22,16,40
    escribir 'L',22,17,40

    escribir 'X',0,30,40

endm
```

En esta línea de código se invocan dos macros:

- escribir ‘0’,22,9,15

  Parámetro 1: Se envía el carácter que deseamos imprimir dentro del modo gráfico.
  Parámetro 2: Posiciona el carácter hacia arriba o abajo.
  Parámetro 3: Posiciona el carácter hacia la derecha o izquierda
  Parámetro 4: Determina el color del caracter.

- dibujarboton 74,9,15

  Parámetro 1: Posiciona el botón hacia arriba o abajo.
  Parámetro 2: Posiciona el botón hacia la derecha o hacia la izquierda.
  Parámetro 3: Determina el color del botón.

La imagen siguiente muestra como se ve lo anteriormente mencionado para una mejor visualización de lo explicado.

Figura 4: Dibuja botones

En la macro de botones debemos analizar la macro dibujarboton, botontipo y escribir, ya que esta dibuja las letras dentro de los botones.

```
botontipo macro X, Y, C
    limpiar
    lineahor Y,(Y+39),X,C
    limpiar
    lineahor Y,(Y+39),(X+35+25+25),C
    limpiar
    lineaver X,(X+35+25+25),Y,C
    limpiar
    lineaver X,(X+35+25+25),(Y+39),C
    limpiar
endm


dibujarboton macro X, Y, C
    limpiar
    lineahor Y,(Y+39),X,C
    limpiar
    lineahor Y,(Y+39),(X+24),C
    limpiar
    lineaver X,(X+24),Y,C
    limpiar
    lineaver X,(X+25),(Y+39),C
    limpiar
endm
```

```
escribir macro CHAR,R,C,COL
    mov ah,0Bh
    mov bh,0h
    mov bl,3
    int 10h
    mov ah,2
    mov bh,0
    mov dh,R
    mov dl,C
    int 10h
    mov ah,9
    mov al,CHAR
    mov bl,COL
    mov cx,1
    int 10h
endm
```

Estas macros anteriores son las encargadas de dibujar el botón y los caracteres que los representa para el usuario.

Otra de las macros importante es la ratón es la encargada de inicializar el mouse y mostrarlo.

```
raton macro
    mov ax,00h
    int 33h

    mov ax,01h
    int 33h
endm
```

2.2.2. Funcionamiento lógico

Una vez que se tiene la interfaz gráfica dibujada es necesario programar el código necesario para hacer funcionar cada uno de los botones, para esto es necesario ir delimitando la zona de cada botón en donde permitirá dar clic.

Al iniciar se verifica si se presiona el boton izquierdo del mouse, una vez que se comprueba esto se compara con el número de columna, si es menor o igual a 48 es probable que se haya presionado sobre los botones más a la izquierda (7,4,1 o AC); si es menor o igual a 98 es probable que se haya presionado sobre los números del centro (8,5,2 o 0); si es menor o igual a 148 es probable que se haya presionado sobre los números más a la derecha (9,6,3 o DEL); si es menor o igual a 198 es probable que se haya presionado sobre los botones de operaciones (X,/,− o +) y si es menor o igual a 198 es probable que se haya presionado sobre los botones más a la derecha (% o =)

```
mouse:
    revisaClicIzquierdo
    ; Al final de la macro anterior, obtenemos las coordenadas del mouse
    ; DX es el numero del renglon, que puede ser desde 0 hasta 200
    ; CX es el numero de la columna, que puede ser desde 0 hasta 320

    ; Revisar si el mouse se presiono en X para cerrar
    ; ;En res 320x200, X inicia en (240,0) y termina en (247,8)
    cmp dx,8
    jle botonX

    ; Rango de columnas en donde esta la primera columna de botones (botones izquierdos)
    ; minimo 9, ya se hizo la comparacion por ese valor anteriormente
    ; maximo 48
    cmp cx,48
    jle botones_AC_1_4_7

    cmp cx,98
    jle botones_0_2_5_8

    cmp cx,148
    jle botones_C_3_6_9

    cmp cx,198
    jle botones_operaciones

    cmp cx,248
    jle botones_operaciones2

    jmp mouse
```

El código base proporcionado por el profesor sugería una triple comprobación para verificar que estuviera dentro del rango de botones, dentro de ciertas columnas y dentro de ciertos renglones, parecía una solución viable, sin embargo al avanzar con el desarrollo del proyecto nos dimos cuenta de que el código comenzaba a extenderse demasiado, por lo que decidimos reducir estas comprobaciones con una macro llamada $isButton$ la cual recibe 4 parámetros: $xmin$, $xmax$, $ymin$ y $ymax$, estos hacen referencia a las cuatro esquinas que conforman un boton y a su vez su funcionamiento puede extenderse para verificar una zona completa.

```
isButton macro xmin,xmax,ymin,ymax
    ; Comparar que se haga clic dentro del boton
    cmp dx,ymin
    jl mouse                            ; Si DX < ymin, regresa a etiqueta 'mouse'
    cmp dx,ymax
    ja mouse                            ; Si DX > ymax, regresa a etiqueta 'mouse'
    cmp cx,xmin
    jl mouse                            ; Si CX < xmin, regresa a etiqueta 'mouse'
    cmp cx,xmax
    ja mouse                            ; Si CX > xmax, regresa a etiqueta 'mouse'
endm
```

Una vez que se comprueba que se hace clic sobre un botón se realizan distintas operaciones dependiendo del tipo de botón del cual se trate.

Si es un **número** se almacena el valor correspondiente en la variable *num_boton* y da el salto a la etiqueta *lee_num1*

```
boton8:
    isButton 59,99,74,98
    mov [num_boton],8
    jmp lee_num1
boton5:
    isButton 59,99,104,128
    mov [num_boton],5
    jmp lee_num1
boton2:
    isButton 59,99,134,158
    mov [num_boton],2
    jmp lee_num1
boton0:
    isButton 59,99,164,188
    mov [num_boton],0
    jmp lee_num1
```

```
no_lee_num:
    jmp mouse
lee_num1:
    cmp [bandera],0
    jne mouse
    cmp [operador],0                    ; compara el valor del operador que puede ser
    0, '+', '-', '*', '/', '%'
    jne lee_num2                        ; Si el operador es diferente de 0, entonces
    lee el segundo numero
    cmp [conta1],4                      ; compara si el contador para num1 llego al
    maximo
    jae no_lee_num                      ; si conta1 es mayor o igual a 4, entonces se
    ha alcanzado el numero de digitos
                                        ; y no hace nada
    mov al,num_boton                    ; valor del boton presionado en AL
    mov di,[conta1]                     ; copia el valor de conta1 en registro indice
    DI
    mov [aux1+di],al                    ; num1 es un arreglo de tipo byte
                                        ; se utiliza di para acceder el elemento
                                        di-esimo del arreglo num1
                                        ; se guarda el valor del boton presionado en
                                        el arreglo
    inc [conta1]                        ; incrementa conta1 por numero correctamente
    leido
prepara_num1:
    ; Se imprime el numero del arreglo num1 de acuerdo a conta1
    xor di,di                           ; limpia DI para utilizarlo
    mov cx,[conta1]                     ; prepara CX para loop de acuerdo al numero de
    digitos introducidos
    mov [ren_aux],2                     ; variable ren_aux para poner cursor en
    pantalla
                                        ; ren_aux se mantiene fijo a lo largo del
                                        siguiente loop
                                        ; Loop para imprimir numero
```

```
imprime_num1:
    push cx                            ; guarda el valor de CX en la pila
    cmp conta1,0
    je mouse
    mov [col_aux],30                   ; variable col_aux para mover cursor en
    pantalla
    sub [col_aux],cl                   ; Para calcular la columna en donde comienza a
    imprimir en pantalla de acuerdo a CX
    mov cl,[aux1+di]                   ; copia el digito en CL
    add cl,30h                         ; Pasa valor a ASCII
    escribir cl,[ren_aux],[col_aux],15 ; escribe caracter en pantalla
    inc di                             ; incrementa DI para recorrer el arreglo num1
    pop cx                             ; recupera el valor de CX al inicio del loop
    loop imprime_num1
    ; Fin loop
    jmp mouse
```

Esta parte del código se encarga de almacenar el dígito almacenado en la variable *num_boton* en un arreglo de 4 elementos tipo Byte e imprimirlos en pantalla recorriendo el arreglo con ayuda del registro *di*. Tambíen se encarga de comprobar si ingresamos por lo menos un dígito para no permitir al usuario efectuar operaciones sin números, comprueba si estamos ingresando dígitos para el primer o segundo número o si el resultado ya se muestra en pantalla para no cambiar los números sin limpiar antes a pantalla.

Figura 5: Lee e imprime digitos del primer número.

Si el botón fuera de operación entonces se verifica que se haya ingresado por lo menos un número comprobando *conta1* con 0, si es igual entonces no permite ingresar ningún símbolo y si es distinto de 0 (o en este caso mayor), guarda el respectivo símbolo en la variable *operador* y lo pinta en pantalla con un color amarillo un renglón abajo del primer número y completamente a la izquierda.

```
botonMultiplicacion:
    isButton 159,198,74,98
    cmp [conta1],0
    je mouse
    mov [operador],'x'
    escribir [operador],3,2,14
botonDivisionCociente:
    isButton 159,198,104,128
    cmp [conta1],0
    je mouse
    mov [operador],'/'
    escribir [operador],3,2,14
botonResta:
    isButton 159,198,134,158
    cmp [conta1],0
    je mouse
    mov [operador],'-'
    escribir [operador],3,2,14
botonSuma:
    isButton 159,198,164,188
    cmp [conta1],0
    je mouse
    mov [operador],'+'
    escribir [operador],3,2,14
```

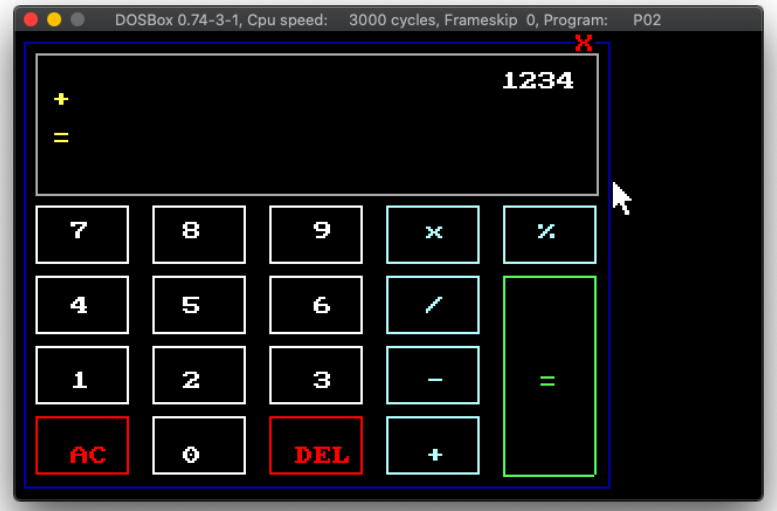


Figura 6: Imprime símbolo de la operación aritmética a realizar.

Para el segundo número se comprueba si la variable *operador* es distinto de cero, para este punto esta comprobación será verdadera por lo que comenzaremos a almacenar dígitos en un segundo arreglo para posteriormente imprimir el segundo número 2 renglones más abajo del primero.

Fuera de esto las operaciones realizadas son las mismas que para el primer número.

Cabe aclarar que para ambos números el programa deja de leer dígitos cuando el respectivo contador (*conta1* o *conta2*) llegue a 4.

Figura 7: Lee e imprime dígitos del segundo número.

El siguiente botón es el de resultado, este botón podría decirse que contiene la lógica más importante del programa debido a que es aquí donde se procesan los dígitos previamente almacenados en arreglo y a su vez hace llamado de las procedimientos y macros que se encargan de realizar e imprimir las operaciones.

```
botonResultado:
    isButton 209,248,104,188
    cmp [conta2],0
    je mouse
    call procesarNumero1
    mov num1,ax
    call procesarNumero2
    mov num2,ax
    escribir '',5,4,14
    operacion [operador]
    mov bandera,1
```

Los procedimientos *procesarNum1* y *procesarNum2* fueron reutilizados del proyecto anterior pero adaptados de tal manera que utilizaran los valores almacenados en un arreglo a diferencia de las variables independientes dedicadas a almacenar unidades, decenas, centenas o millares.

```
procesarNumero1 proc
    xor ah,ah           ; limpia la parte alta del registro AX
    mov al,[aux1]       ; AL = um
    mov bx,1000         ; BX = 1000
    mul bx              ; DX:AX = AX * BX
    mov num,ax          ; num = AX

    xor ah,ah           ; limpia la parte alta del registro AX
    mov al,[aux1+1]     ; AL = c
    mov bl,100          ; BL = 100
    mul bl              ; AX = AL * BL
    add num,ax          ; num1 = num1 + AX

    xor ah,ah           ; limpia la parte alta del registro AX
    mov al,[aux1+2]     ; AL = d
    mov bl,10           ; BL = 10
    mul bl              ; AX = AL * BL
    add num,ax          ; num1 = num1 + AX

    xor ah,ah           ; limpia la parte alta del registro AX
    mov al,[aux1+3]     ; AL = u
    add num,ax          ; num1 = num1 + AX

    mov cx,conta1
    xor ch,ch
    cmp cl,1            ; Compara CL con 1
    je i1               ; Si Cl == 1 entonces salta a i1
    cmp cl,2            ; Compara CL con 2
    je i2               ; Si Cl == 2 entonces salta a i2
    cmp cl,3            ; Compara CL con 3
    je i3               ; Si Cl == 3 entonces salta a i3
    cmp cl,4            ; Compara CL con 4
    je flujo            ; Si Cl == 4 entonces salta a flujo3
i1:
    mov ax,num          ; AX = num1
    mov bx,mil          ; BX = 1000
    xor dx,dx
    div bx              ; DX:AX = AX / BX
    mov num,ax          ; num = AX
    jmp flujo           ; Salta a flujo3
i2:
    mov ax,num          ; AX = num1
    mov bx,cien         ; BX = 100
    xor dx,dx
    div bx              ; DX:AX = AX / BX
    mov num,ax          ; num = AX
    jmp flujo           ; Salta a flujo3
i3:
    mov ax,num          ; AX = num1
    mov bx,diez         ; BX = 10
    xor dx,dx
    div bx              ; DX:AX = AX / BX
    mov num,ax          ; num = AX
    jmp flujo           ; Salta a flujo3
flujo:
    mov ax,num
    ret
endp procesarNumero1
```

Posteriormente es necesario posicionar el cursor al final del signo =, esto para poder imprimir el resultado reutilizando el código del proyecto anterior utilizando la interrupción $21h$ descrita con anterioridad.

Para realizar la operación deseada hacemos uso de la macro *operacion* pasando como parámetro a *operador*. Una diferencia importante con respecto al proyecto anterior es que esta calculadora no requiere de realizar todos las operaciones por cada ejecución, por lo que implementamos un salto al final de cada operación para salir de este procedimiento.

Independientemente a los cambios mencionados y junto con la usencia de mensajes en pantalla, la implementación de las operaciones aritméticas del proyecto anterior es la misma.

Al final se cambia el valor de la variable *bandera* a 1 para indicar que el resultado ya se muestra en pantalla, esto nos servirá para cambiar ligeramente el funcionamiento del botón *DEL* para que se comporte igual que *AC* los cuales se explicarán más adelante.

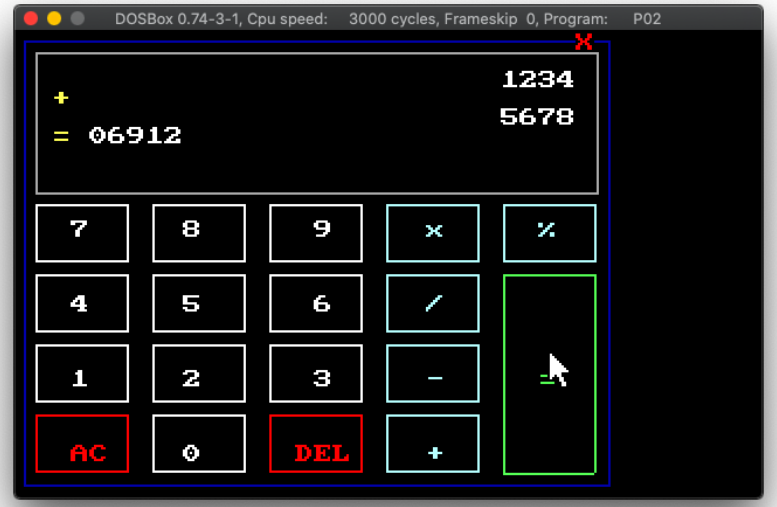


Figura 8: Imprime el resultado de la operación realizada.

Otro tipo de botones, como hemos podido ver, son los diseñados para borrar, ya sea dígito por dígito (*DEL*) o toda la pantalla al mismo tiempo (*AC*).

La implementación de estos botones parecía sencilla durante el planteamiento, sin embargo, fue una de la que más nos costó trabajo al momento de afinar detalles pues en nuestras primeras pruebas los dígitos parecían borrarse pero dejaba remanentes en pantalla al continuar escribiendo, es decir, si por ejemplo escribíamos 1234 y borrábamos dos dígitos (12__) porque realmente queríamos escribir 128 había un error de impresión por el cual se pintaba en pantalla el número 1128, sin embargo el resultado se imprimía correctamente utilizando como operandos los números deseados, que para este ejemplo es 128.

Así mismo, una primera solución para el botón AC era dar un salto a la etiqueta *inicio* no sin antes inicializar las variables en 0's, pero estéticamente no era de nuestro agrado, pues generaba un parpadeo en la pantalla, es por esto que decidimos aplicar un método un poco "*arcaico*" (por decirlo de alguna manera) que consiste en declarar un mensaje de 8 espacios en blanco e imprimirlos en la misma posición donde estaba la impresión del resultado y un arreglo de 4 espacios para sobreescribir los operandos.

```
botonAC:
    isButton 9,49,164,188
borrar:
    cmp conta1,0
    je salto
    borra conta1,2
    cmp conta2,0
    je salto
    borra conta2,4
salto:
    escribir '',3,2,14
    borraResultado
    mov aux1,0
    mov aux2,0
    mov conta1,0
    mov conta2,0
    mov ren_aux,0
    mov col_aux,0
    mov operador,0
    mov num1,0
    mov num2,0
    mov bandera,0
    jmp mouse
```

```
borra macro cont,n
    local borra_num

    xor di,di                          ; limpia DI para utilizarlo
    mov cx,[cont]                      ; prepara CX para loop de acuerdo al numero de
    digitos introducidos
    mov [ren_aux],n                    ; variable ren_aux para poner cursor en pantalla
                                       ; ren_aux se mantiene fijo a lo largo del
                                       siguiente loop
borra_num:                             ; Loop para imprimir numero
    push cx                            ; guarda el valor de CX en la pila
    mov [col_aux],30                   ; variable col_aux para mover cursor en pantalla
    sub [col_aux],cl                   ; Para calcular la columna en donde comienza a
    imprimir en pantalla de acuerdo a CX
    mov cl,[clear+di]                  ; copia el digito en CL
    escribir cl,[ren_aux],[col_aux],15 ; escribe caracter en pantalla
    inc di                             ; incrementa DI para recorrer el arreglo num1
    pop cx                             ; recupera el valor de CX al inicio del loop
    loop borra_num
endm
```

Una vez que se detecta el botón, se compara si el contador del primero número es 0, si es asi entonces no borra nada y solo inicializa variables en caso de ser mayor a 0 entonces imprime el arreglo de espacios segun la cantidad de dígitos que haya guardado, todo esto con ayuda del registro *di*. Lo mismo sucede para el segundo número. Y para el resultado imprime un mensaje de 8 espacios debido a que es la cantidad máxima de dígitos que puede mostrar el resultado.

```
clear               db      20h,20h,20h,20h          ; Arreglo de espacios
clean               db      "               ","$"    ; Mensaje de espacios
```

```
botonC:
    isButton 109,149,164,188
    cmp [bandera],0
    jne borrar
    cmp [operador],0
    jne botonC2
    cmp conta1,0
    je mouse
    borra conta1,2
    dec conta1
    jmp prepara_num1
botonC2:
    cmp conta2,0
    je mouse
    borra conta2,4
    dec conta2
    cmp conta2,0
    jne prepara_num2
    escribir '',3,2,14
    mov [operador],0
    mov [ren_aux],2
    mov [col_aux],29
    mov aux2,0
```

Para el botón *DEL*, primero se compara la *bandera* con 0, pues de ser diferente de 0 el funcionamiento de este botón es idéntico al de *AC*, es decir, borra la pantalla.

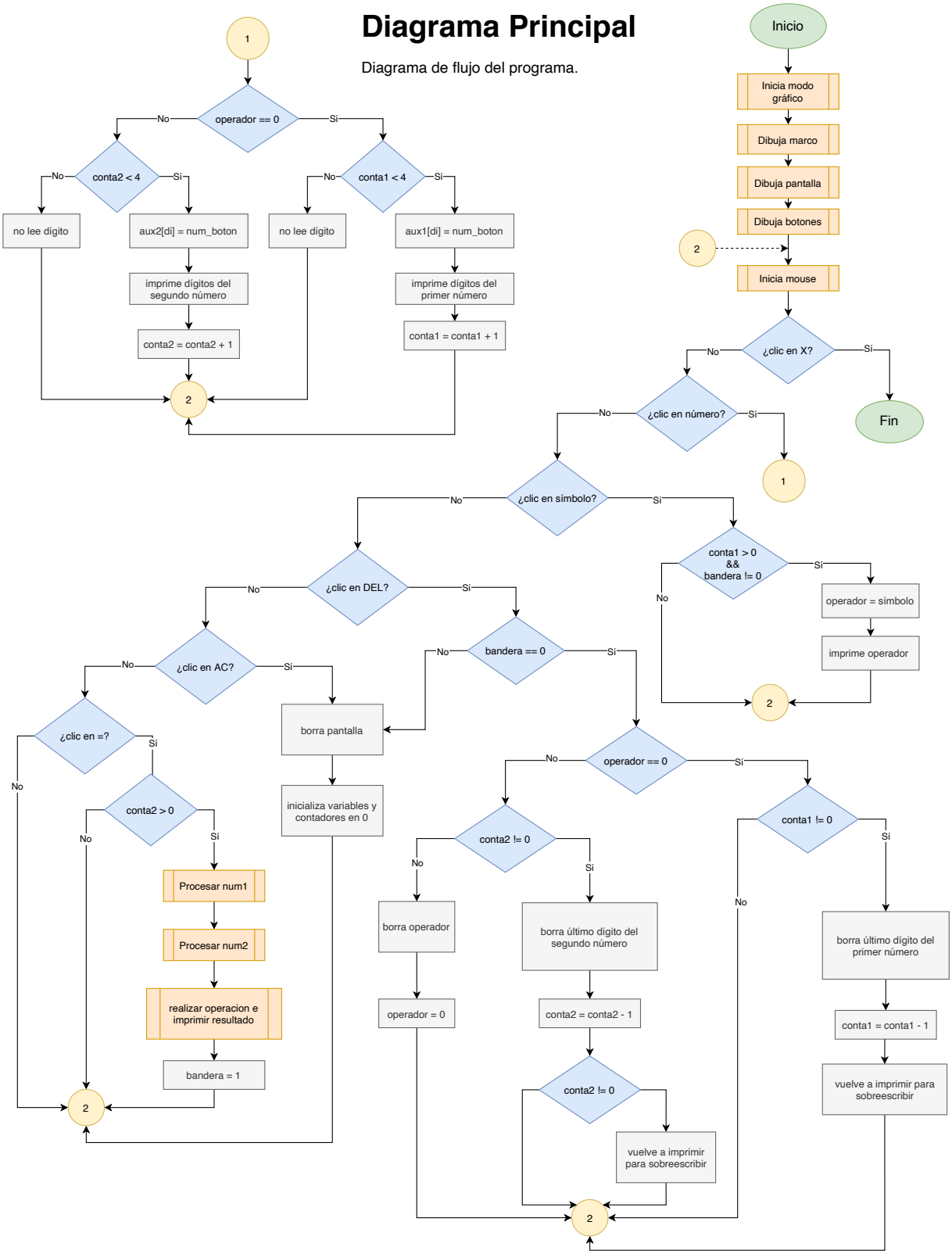
De lo contrario, se compara ahora el estado del *operador*, ya que este nos indica cuál de los dos número es el que se está modificando.

Si resulta ser 0 el valor de ambas comparaciones entonces se procede a eliminar el último dígito del primer número.

Si en la segunda comparación *operador* contiene un valor diferente de 0, entonces se elimina el último dígito del segundo número.

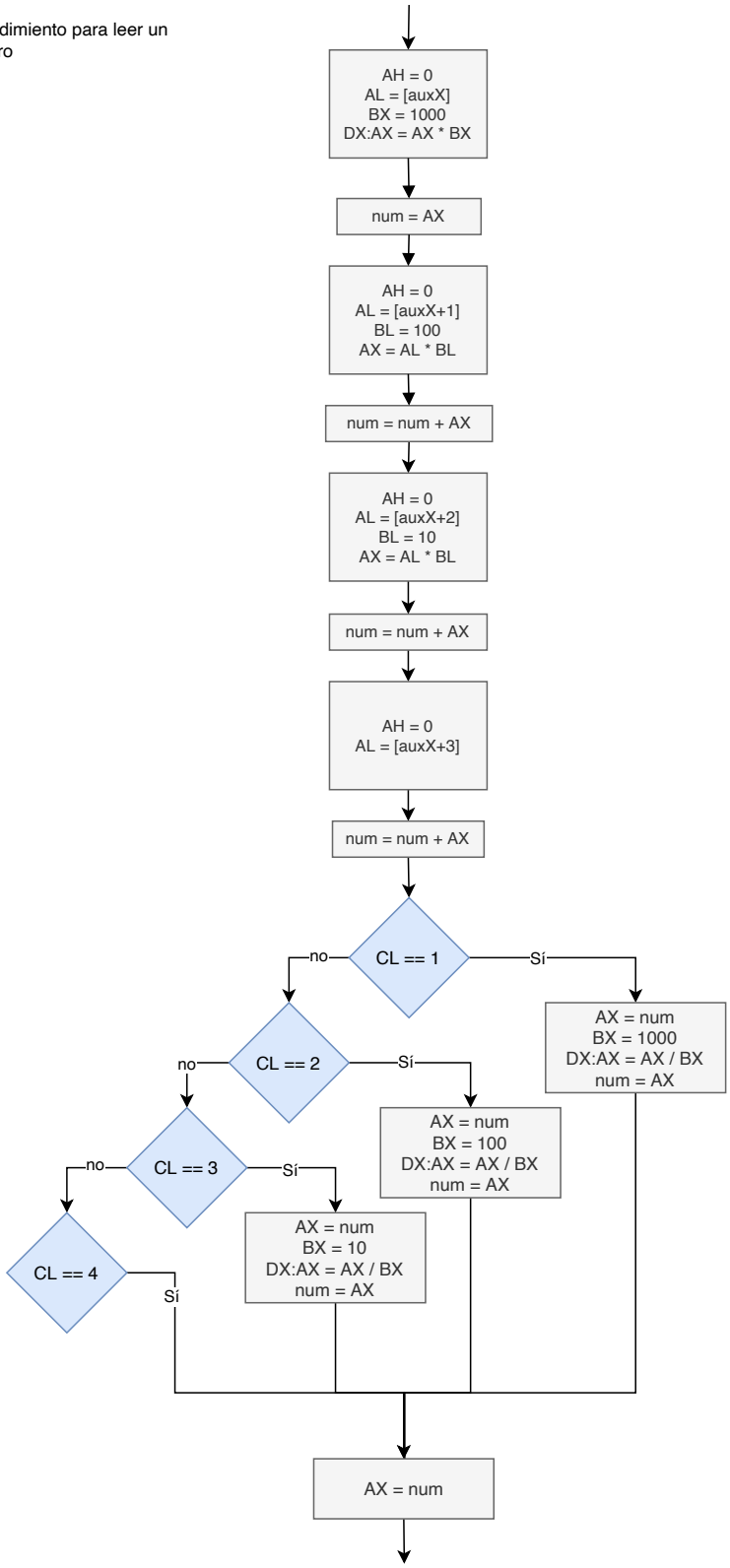
En el proceso de borrado de dígitos del segundo número, cuando el contador llega a 0 también de borra de la pantalla el símbolo del operando.

3. Diagramas de flujo



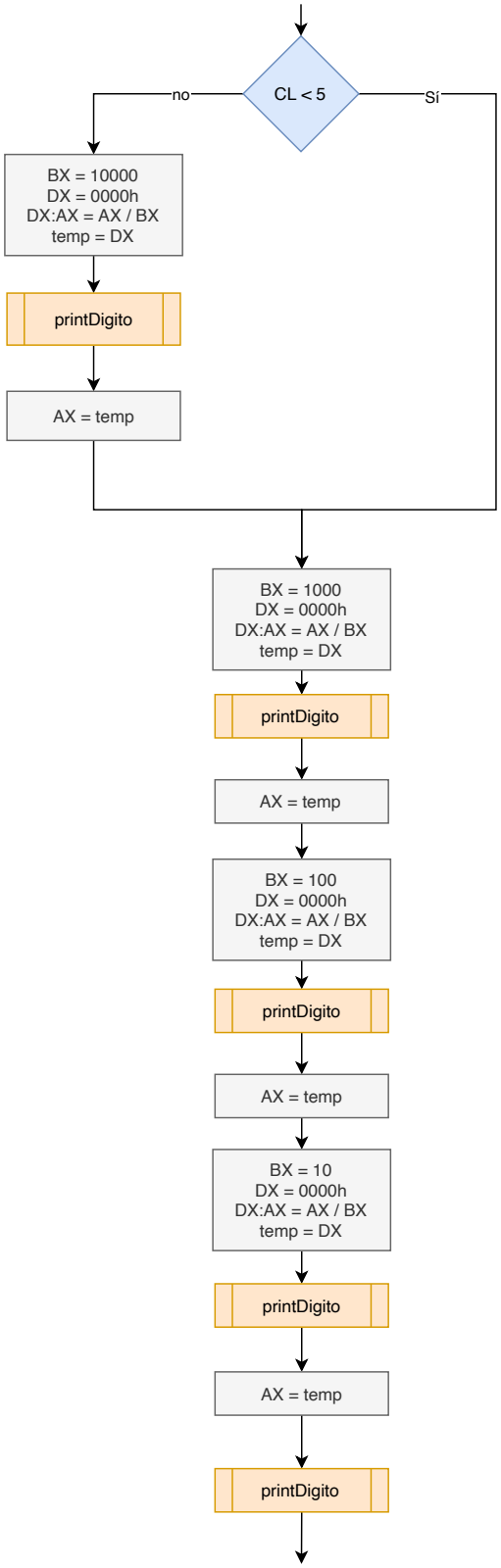
ProcesarNumX

Procedimiento para leer un número



PrintNum

Procedimiento para imprimir un número de 4 o 5 dígitos.



4. Pruebas de escritorio

5. Conclusiones