Práctica 4

Organización y Arquitectura de Computadoras

Docente: Garcia Rocha Jose Isabel

Alumno:

- Chaparro Herrera Hugo Giovanni - 2200357

Tijuana, Baja California a 18 de septiembre de 2025

Dificultades durante el desarrollo

Durante el desarrollo de la práctica, la principal dificultad encontrada fue el cambio de lógica requerido al programar en lenguaje ensamblador. A diferencia de lenguajes de alto nivel, donde las operaciones sobre variables son más abstractas, en ensamblador se trabaja directamente con registros de propósito general, lo que exige una comprensión precisa de su estructura y comportamiento.

En particular, fue común cometer errores al manipular las versiones de 8 bits de los registros, como AL, BL, CL y DL. Estos registros al poder ser manipulada cierta sección del registro a conveniencia del programador, comúnmente el error recurrente era que al no setear el registro apropiadamente, el contenido esperado siempre se modificaba.

Desarrollo

1. Captura y conversión de datos

El programa inicia mostrando un mensaje al usuario para ingresar un número entre 0 y 9. Utiliza call getch para capturar el carácter, lo almacena en la variable numero, y lo imprime en formato hexadecimal con printHex. Posteriormente, convierte el carácter ASCII a su valor numérico restando '0' y lo vuelve a imprimir con su valor convertido. Este mismo proceso se repite para el segundo número, almacenado en numero2.

```
; se muestra el mensaje de captura del primer numero
mov edx, msg
call puts
call salto

; se captura el primer numero y se guarda
mov eax, 0
call getch
mov ebx, numero
mov [ebx], al
mov esi, cad
call printHex
call salto

; se convierte a numero
mov ebx, numero
sub byte[ebx], '0'

; se muestra el mensaje de conversion
mov edx, msg2
call puts
call salto
```

2. Multiplicación por sumas sucesivas

Se muestra el mensaje de multiplicación y se inicializa la variable resultado en cero. Luego, se carga el segundo número en DL y el primero en CL, que se usa como contador. El ciclo mult suma el valor de DL al byte apuntado por resultado en cada iteración. Se imprime el resultado parcial en cada paso utilizando printHex. Una vez finalizada la operación, se imprime el resultado final con printHex

```
    ; se muestra el mensaje de multiplicacion
    mov edx, multi
    call puts
    call salto

    ; se setean los registros para comenzar la multiplicacion
    mov ebx, resultado      ; *EBX = &resultado
    mov byte [ebx], 0       ; *EBX = 0 ==> resultado = 0
    mov ebx, numero2        ; *EBX = &numero2
    mov dl, [ebx]           ; DL = *EBX ==> DL = *numero2

    mov ebx, numero         ; *EBX = &numero
     mov cl, [ebx]           ; se usa el primer numero como contador de iteraciones para la
realizacion de la multiplicacion

    ; se genera la multiplicacion
    mult:
        mov ebx, resultado
        add byte[ebx], dl

        ; se imprime la suma
        mov ebx, resultado
        mov eax, [ebx]
        mov esi, cad
        call printHex
        call salto
    loop mult
    call salto

    ; se imprime la multiplicacion
    mov eax, 0
    mov ebx, resultado
    mov al, [ebx]
    mov esi, cad
    call printHex
    call salto
    call salto
```

3. División

Se muestra el mensaje de división y se prepara el divisor (DL) y el dividendo (AL) con el valor de resultado. Se inicializa CL en cero para contar cuántas veces se puede restar el divisor del dividendo. El ciclo div realiza restas sucesivas, incrementando CL hasta que AL llegue a cero. Se imprime el resultado final de la división.

```
    ; se muestra el mensaje de division
    mov edx, divi
    call puts
    call salto

    ; se setean los registros para comenzar la division
    mov ebx, numero          ; *EBX = &numero = 3
    mov dl, [ebx]            ; DL = *EBX ==> DL = *numero = 3 (Divisor)

     mov cl, 0               ; se usa el primer numero como contador de iteraciones para la
realizacion de la division

    mov ebx, resultado       ; resultado 3 x 4 = 12
    mov al, [ebx]            ; Dividendo

    ; se genera el ciclo de la division
    div:
         sub al, dl            ; se resta al (el resultado de la multiplicacion) con dl (el
numero1)
         add cl, 1             ; se le suma 1 a cl para ver cuantas veces se puede dividir
(restar) dicho numero

         cmp al, 0             ; se compara al con 0 para ver si ya llegamos al maximo de
restas
        je fin              ; si al es igual a 0, entonces, se rompe el ciclo de division

        ; se imprime la como se va restando
        mov esi, cad
        call printHex
        call salto
        jmp div

    fin:
    call salto

    ; se imprime la division
    mov eax, 0
    mov al, cl
    mov esi, cad
    call printHex
    call salto
    call salto
```

4. Contador de 1 al 100

Se muestra el mensaje correspondiente y se inicializa la variable contador en cero. Se establece CL en 100 para controlar el número de iteraciones. En cada ciclo, se incrementa en 1 el contador y se imprime su valor en hexadecimal.

```
; se muestra el mensaje del contador
mov edx, cont
call puts
call salto

; se setea contador = 0
mov ebx, contador
mov byte[ebx], 0

; cl = 100 (numero de iteraciones)
mov cl, 100
ciclo:
    ; suma 1 a ebx -> contador
    add byte[ebx], 1

    ; se imprime el contenido de contador
    mov eax, 0
    mov ebx, contador
    mov eax, [ebx]
    mov esi, cad
    call printHex
    call salto
loop ciclo
call salto
```

5. Contador del 1 al 100 (de 2 en 2)

Se muestra el mensaje del segundo contador y se inicializa contador en cero. Se utiliza el registro AL como bandera de paridad, comenzando en 0. En cada iteración del ciclo ciclo2, se incrementa contador y se llama a la subrutina par, que decide si se imprime el valor actual dependiendo del estado de la bandera. La impresión se realiza solo cuando AL es igual a 1, alternando entre valores pares.

```
    ; mostrar mensaje del contador
    mov edx, cont2
    call puts
    call salto

    ; setear contador = 0
    mov ebx, contador
    mov byte [ebx], 0

    ; registro que se utiliza como bandera de paridad
    ; se coloca en 0 porque la primer iteracion (cont = 1) es impar
    mov al, 0

    ; cl = 100 (numero de iteraciones)
    mov cl, 100
    ciclo2:
        add byte [ebx], 1       ; sumar 1 al contador

        call par                ; se manda a llamar la subrutina para verificar la paridad
        loop ciclo2

par:
     cmp al, 1             ; se compara con 1 al, si es igual, se imprime el contenido de
contador
    je imprimir

     mov al, 1            ; se enciende la bandera para la siguiente iteracion y se regresa
al ciclo prindipal
    ret

imprimir:
    mov al, 0           ; se apaga la bandera

    ; se imprime el contenido de ebx (cont)
    mov eax, [ebx]
    mov esi, cad
    call printHex
    call salto
    ret
```

Código

```
%include "../Libreria/pc_io.inc"  ; se incluye la libreria
section .text
    global _start:

_start:
; ========================= CAPTURA DE DATOS ========================

    ; se muestra el mensaje de captura del primer numero
    mov edx, msg
    call puts
    call salto

    ; se captura el primer numero y se guarda
    mov eax, 0          ; puede estar comentado
    call getch
    mov ebx, numero
    mov [ebx], al
    mov esi, cad        ; puede estar comentado
    call printHex       ; puede estar comentado
    call salto          ; puede estar comentado

    ; se convierte a numero
    mov ebx, numero
    sub byte[ebx], '0'

    ; se muestra el mensaje de conversion
    mov edx, msg2       ; puede estar comentado
    call puts           ; puede estar comentado
    call salto          ; puede estar comentado

    ; se imprime el numero convertido
    mov eax, 0
    mov ebx, numero
    mov al, [ebx]
    mov esi, cad
    call printHex
    call salto
    call salto
```

```
    ; se muestra el mensaje de captura n2
    mov edx, msg
    call puts
    call salto

    ; se captura el segundo numero y se guarda
    mov eax, 0          ; puede estar comentado
    call getch
    mov ebx, numero2
    mov [ebx], al
    mov esi, cad        ; puede estar comentado
    call printHex       ; puede estar comentado
    call salto          ; puede estar comentado

    ; se convierte a numero
    mov ebx, numero2
    sub byte [ebx], '0'

    ; se muestra el mensaje de conversion
    mov edx, msg2       ; puede estar comentado
    call puts           ; puede estar comentado
    call salto          ; puede estar comentado

    ; se imprime el numero convertido
    mov eax, 0
    mov ebx, numero2
    mov al, [ebx]
    mov esi, cad
    call printHex
    call salto
    call salto

; ========================================== MULTIPLICACION

    ; se muestra el mensaje de multiplicacion
    mov edx, multi
    call puts
    call salto
```

```
    ; se setean los registros para comenzar la multiplicacion
    mov ebx, resultado      ; *EBX = &resultado
    mov byte [ebx], 0       ; *EBX = 0 ==> resultado = 0
    mov ebx, numero2        ; *EBX = &numero2
    mov dl, [ebx]           ; DL = *EBX ==> DL = *numero2

    mov ebx, numero         ; *EBX = &numero
    mov cl, [ebx]           ; se usa el primer numero como contador de
iteraciones para la realizacion de la multiplicacion

    mult:
            mov  ebx,  resultado              ; Carga la dirección de
'resultado' en EBX
          add byte [ebx], dl            ; Suma el valor de DL al byte
apuntado por EBX (resultado)

        ; se imprime la suma
         mov ebx, resultado          ; Vuelve a cargar la dirección de
'resultado' en EBX
        mov eax, [ebx]              ; Carga el valor de 'resultado' en
EAX
         mov esi, cad                ; Carga la dirección de la cadena
'cad' en ESI
         call printHex                 ; Llama a la rutina que imprime
EAX en formato hexadecimal
        call salto                  ; Llama a la rutina que imprime un
salto de línea
    loop mult
    call salto

    ; se imprime la multiplicacion
    mov eax, 0
    mov ebx, resultado
    mov al, [ebx]
    mov esi, cad
    call printHex
    call salto
    call salto

; ========================================== DIVISION
```

```
    ; se muestra el mensaje de division
    mov edx, divi
    call puts
    call salto

    ; se imprime la multiplicacion
    mov eax, 0
    mov ebx, resultado
    mov al, [ebx]
    mov esi, cad
    call printHex
    call salto


    ; se setean los registros para comenzar la division
    mov ebx, numero          ; *EBX = &numero = 3
    mov dl, [ebx]            ; DL = *EBX ==> DL = *numero = 3

    mov cl, 0                ; se usa el primer numero como contador de
iteraciones para la realizacion de la division

    mov ebx, resultado       ; resultado 3 x 4 = 12
    mov al, [ebx]

    ; se genera el ciclo de la division
    div:
          sub al, dl              ; se resta al (el resultado de la
multiplicacion) con dl (el numero1)
        add cl, 1            ; se le suma 1 a cl para ver cuantas veces
se puede dividir (restar) dicho numero

         cmp al, 0             ; se compara al con 0 para ver si ya
llegamos al maximo de restas
        je fin                ; si al es igual a 0, entonces, se rompe
el ciclo de division

        ; se imprime la como se va restando
        mov esi, cad
        call printHex
```

```
        call salto
        jmp div

    fin:
    call salto

    ; se imprime la division
    mov eax, 0
    mov al, cl
    mov esi, cad
    call printHex
    call salto
    call salto

; ==================================== contador del 1 al 100 (64h)

    ; se muestra el mensaje del contador
    mov edx, cont
    call puts
    call salto

    ; se setea contador = 0
    mov ebx, contador
    mov byte[ebx], 0

    ; cl = 100 (numero de iteraciones)
    mov cl, 100
    ciclo:
        ; suma 1 a ebx -> contador
        add byte[ebx], 1

        ; se imprime el contenido de contador
        mov eax, 0
        mov ebx, contador
        mov eax, [ebx]
        mov esi, cad
        call printHex
        call salto
    loop ciclo
    call salto
```

```
; ======================================= 1 - 100 (2 en 2)

    ; mostrar mensaje del contador
    mov edx, cont2
    call puts
    call salto

    ; setear contador = 0
    mov ebx, contador
    mov byte [ebx], 0

    ; registro que se utiliza como bandera de paridad
    ; se coloca en 0 porque la primer iteracion (cont = 1) es impar
    mov al, 0

    ; cl = 100 (numero de iteraciones)
    mov cl, 100
    ciclo2:
        add byte [ebx], 1       ; sumar 1 al contador

         call par               ; se manda a llamar la subrutina para
verificar la paridad
        loop ciclo2

    ; SYS_EXIT
    mov eax, 1
    mov ebx, 0
    int 80h

par:
    cmp al, 1           ; se compara con 1 al, si es igual, se imprime
el contenido de contador
    je imprimir

     mov al, 1          ; se enciende la bandera para la siguiente
iteracion y se regresa al ciclo prindipal
    ret

imprimir:
```

```
    mov al, 0           ; se apaga la bandera

    ; se imprime el contenido de ebx (cont)
    mov eax, [ebx]
    mov esi, cad
    call printHex
    call salto
    ret

; ======================================

salto:
    pushad
    mov al, 13
    call putchar

    mov al, 10
    call putchar
    popad
    ret

printHex:
    pushad
    mov edx, eax
    mov ebx, 0fh
    mov cl, 28
.nxt: shr eax,cl
.msk: and eax,ebx
    cmp al, 9
    jbe .menor
    add al,7
.menor:add al,'0'
    mov byte [esi],al
    inc esi
    mov eax, edx
    cmp cl, 0
    je .print
    sub cl, 4
    cmp cl, 0
    ja .nxt
```

```
    je .msk
.print: mov eax, 4
    mov ebx, 1
    sub esi, 8
    mov ecx, esi
    mov edx, 8
    int 80h
    popad
    ret

section .data
    msg: db "Ingresa un numero (0-9)", 0x0
    len: equ $-msg

    msg2: db "Numero convertido: ", 0x0
    len2: equ $-msg2

    multi: db "Multiplicacion", 0x0
    len3: equ $-multi

    divi: db "Division", 0x0
    len4: equ $-divi

    cont: db "Contador del 1 al 100", 0x0
    len5: equ $-cont

    cont2: db "Contador del 1 al 100 (de 2 en 2)", 0x0
    len6: equ $-cont2

section .bss
    numero resb 1
    numero2 resb 1
    cad resb 12
    resultado resb 10
    contador resb 10
```

Referencias Bibliográficas

- 02. NASMx86: partes del código fuente
- Guide to x86 Assembly
- Using as - Assembler Directives
- NASM Assembly Language Tutorials - asmtutor.com
- Assembly - System Calls