Actividad 6

Christian Geovany Muñoz Rodríguez

Ingeniería en computación

Código: 221350605

Seminario de Solución de Problemas Traductores de lenguajes I – D04 (Lunes y Miércoles de 1 a 3)

Maestro: José Juan Meza Espinosa

Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

6 de marzo del 2023

Código:

```
org 100h
jmp inicio
result DD ?
u1 db 0
d1 db 0
c1 db 0
m1 db 0
n1 dw 0
u2 db 0
d2 db 0
c2 db 0
m2 db 0
n2 dw 0
msg1 db "Ingrese un numero: $"
msg2 db "El resultado de la multipliacion es: $"

inicio:
  MOV dx, offset msg1
  MOV ah, 09h
  INT 21h

  MOV ah, 01h
  INT 21h
  SUB al, 30h
  MOV m1, al  ;Obtenemos el digito de las unidades de millar de n1
```

```
MOV ah, 01h
INT 21h
SUB al, 30h
MOV c1, al  ; Obtenemos el digito de las centenas de n1

MOV ah, 01h
INT 21h
SUB al, 30h
MOV d1, al  ; Obtenemos el digito de las decenas de n1

MOV ah, 01h
INT 21h
SUB al, 30h
MOV u1, al  ; Obtenemos el digito de las unidades de n1

XOR ax, ax

MOV bx, 1000
MOV al, m1  ; m1 * 10000
MUL bx      ; bx porque es 16 bits y cubre a 1000
MOV n1, ax

XOR bx, bx  ; limpiamos todo bx, para no dejar nada en bx

MOV bl, 100
MOV al, c1
MUL bl      ; c1 * 100
```

```
ADD n1, ax ; sumamos las centenas a n1

MOV bl, 10
MOV al, d1 ;c1 * 100 + dl * 10
MUL bl
ADD n1, ax  ;Sumamos las decenas a n1

MOV bl, u1  ; n1 = c1 * 100 + d1 * 10 + u1
ADD n1, bx  ; guardamos el numero completo en n1

call newline
MOV dx, offset msg1 ; pedimos el segundo numero
MOV ah, 09h
INT 21h

MOV ah, 01h
INT 21h
SUB al, 30h
MOV m2, al  ;capta las unidades de millar de n2

MOV ah, 01h
INT 21h
SUB al, 30h
MOV c2, al  ;capta las centenas de n2

MOV ah, 01h
INT 21h
SUB al, 30h
```

```
MOV d2, al  ;capta las decenas del n2

MOV ah, 01h
INT 21h
SUB al, 30h
MOV u2, al  ;capta las unidades del n2

XOR ax, ax

MOV bx, 1000
MOV al, m2  ; m1 * 1000
MUL bx      ; bx porque es 16 bits y cubre a 1000
MOV n2, ax ;guardamos las unidades de millar en n2

XOR bx, bx  ; limpiamos todo bx, para no dejar nada en bx

MOV bl, 100
MOV al, c2
MUL bl      ; c1 * 100
ADD n2, ax ; sumamos las centenas a n2

MOV bl, 10
MOV al, d2
MUL bl
ADD n2, ax  ;Sumamos las decenas a n2
call newline

MOV bl, u2  ; n1 = c1 * 100 + d1 * 10 + u1
```

```
  ADD n2, bx  ; guardamos el numero completo en n1


  MOV dx, offset msg2 ; Imprimimos el mensaje de resultado
  MOV ah, 09h
  INT 21h


  MOV AX, n1 ; multiplicando numero 1
  MOV BX, n2 ; multiplicando numero 2
  MOV CX, 0000H ; contador inicializado a 0
  MOV DX, 0000H ; inicializar el registro de la suma a 0


  LOOP_START:
    TEST BX, 0001H ; comprobar si el ultimo bit de multiplicando
numero 2 esta establecido
    JZ SKIP_ADDITION ; saltar si el ultimo bit esta en 0
    ADD DX, AX ; sumar multiplicando numero 1 al registro de la
suma
  SKIP_ADDITION:
    SHL AX, 1 ; multiplicando numero 1 se desplaza un bit a la
izquierda
    SHR BX, 1 ; multiplicando numero 2 se desplaza un bit a la
derecha
    INC CX ; incrementar el contador
    CMP CX, 16 ; comprobar si se han realizado 16 iteraciones
    JL LOOP_START ; saltar al inicio del bucle si no se han
realizado todas las iteraciones


  MOV result, DX ; almacenar el resultado en la variable result


  ; Imprimir el resultado
```

```
; Imprimir el primer digito
MOV BX, 10000
MOV AX, result
XOR DX, DX
DIV BX
MUL BX
SUB result,AX
DIV BX
ADD AL, 30H
MOV AH, 2
MOV DL, AL
INT 21h

; Imprimir el segundo digito
MOV BX, 1000
MOV AX, result
XOR DX, DX
DIV BX
MUL BX
SUB result,AX
DIV BX
ADD AL, 30H
MOV AH, 2
MOV DL, AL
INT 21h

; Imprimir el tercer digito
```

```
MOV BX, 100
MOV AX, result
XOR DX, DX
DIV BX
MUL BX
SUB result,AX
DIV BX
ADD AL, 30H
MOV AH, 2
MOV DL, AL
INT 21h

; Imprimir el cuarto digito
MOV BX, 10
MOV AX, result
XOR DX, DX
DIV BX
MUL BX
SUB result,AX
DIV BX
ADD AL, 30H
MOV AH, 2
MOV DL, AL
INT 21h

; Imprimir el quinto digito
MOV BX, 1
MOV AX, result
```

```
  XOR DX, DX
  DIV BX
  MUL BX
  SUB result,AX
  DIV BX
  ADD AL, 30H
  MOV AH, 2
  MOV DL, AL
  INT 21h


newline:
    MOV al, 10
    MOV ah,0eh
    INT 10h

    MOV al, 13
    MOV ah,0eh
    INT 10h
    ret

END
```

Desarrollo:

Este programa recibe dos números ingresados por el usuario y los multiplica utilizando el algoritmo de multiplicación binaria. El resultado se almacena en una variable y se muestra en pantalla.

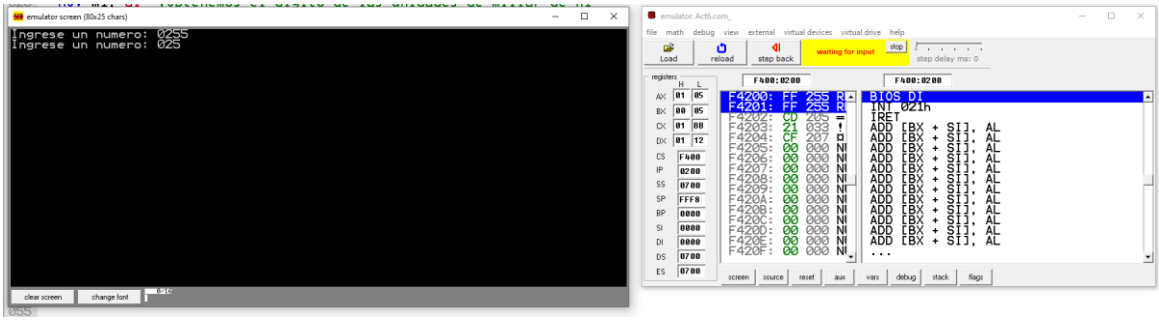
El código comienza con la declaración de las variables: "result" que almacena el resultado de la multiplicación y "n1" y "n2" que almacenan los números ingresados por el usuario en formato numérico. Las variables "u1", "d1", "c1" y "m1" son utilizadas para almacenar los dígitos que conforman el primer número ingresado y

"u2", "d2", "c2" y "m2" para almacenar los dígitos que conforman el segundo número ingresado. Por último, se definen las cadenas de caracteres que se utilizarán para mostrar los mensajes en pantalla.

```
brg 100h
jmp inicio
result DD ?
u1 db 0
d1 db 0
c1 db 0
m1 db 0
n1 dw 0
u2 db 0
d2 db 0
c2 db 0
m2 db 0
n2 dw 0
msg1 db "Ingrese un numero: $"
msg2 db "El resultado de la multipliacion es: $"
```

El código utiliza la interrupción INT 21h para mostrar los mensajes en pantalla y para obtener los dígitos que conforman cada número ingresado por el usuario.

```
inicio:
  MOV dx, offset msg1
  MOV ah, 09h
  INT 21h

  MOV ah, 01h
  INT 21h
  SUB al, 30h
  MOV m1, al  ;Obtenemos el digito de las unidades de millar d

  MOV ah, 01h
  INT 21h
  SUB al, 30h
  MOV c1, al  ; Obtenemos el digito de las centenas de n1

  MOV ah, 01h
  INT 21h
  SUB al, 30h
  MOV d1, al  ; Obtenemos el digito de las decenas de n1

  MOV ah, 01h
  INT 21h
  SUB al, 30h
  MOV u1, al  ; Obtenemos el digito de las unidades de n1

  XOR ax, ax

  MOV bx, 1000
  MOV al, m1  ; m1 * 10000
  MUL bx      ; bx porque es 16 bits y cubre a 1000
  MOV n1, ax

  XOR bx, bx  ; limpiamos todo bx, para no dejar nada en bx

  MOV bl, 100
  MOV al, c1
  MUL bl      ; c1 * 100
  ADD n1, ax ; sumamos las centenas a n1

  MOV bl, 10
  MOV al, d1 ;c1 * 100 + dl * 10
  MUL bl
  ADD n1, ax  ;Sumamos las decenas a n1

  MOV bl, u1  ; n1 = c1 * 100 + d1 * 10 + u1
  ADD n1, bx  ; guardamos el numero completo en n1

  call newline
  MOV dx, offset msg1 ; pedimos el segundo numero
  MOV ah, 09h
  INT 21h

  MOV ah, 01h
  INT 21h
  SUB al, 30h
  MOV m2, al  ;capta las unidades de millar de n2
```



Una vez que se han obtenido los dígitos, se utiliza una serie de operaciones aritméticas para convertir los números ingresados en su representación numérica completa. Es decir, se combina cada dígito en un solo número.

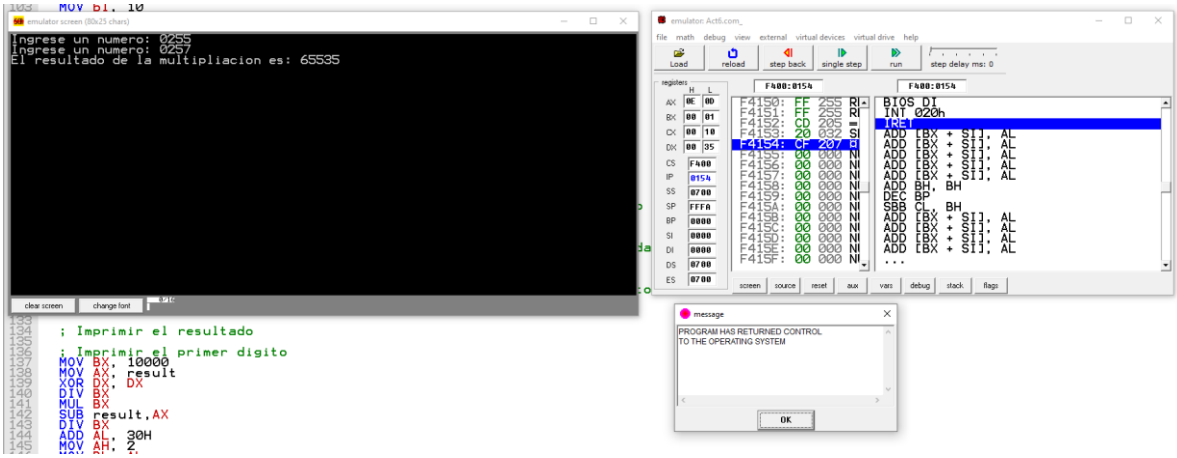
```
MOV bx, 1000
MOV al, m2   ; m1 * 1000
MUL bx       ; bx porque es 16 bits y cubre a 1000
MOV n2, ax ;guardamos las unidades de millar en n2

XOR bx, bx   ; limpiamos todo bx, para no dejar nada en bx

MOV bl, 100
MOV al, c2
MUL bl       ; c1 * 100
ADD n2, ax ; sumamos las centenas a n2

MOV bl, 10
MOV al, d2
MUL bl
ADD n2, ax   ;Sumamos las decenas a n2
call newline

MOV bl, u2   ; n1 = c1 * 100 + d1 * 10 + u1
ADD n2, bx   ; guardamos el numero completo en n2
```

Una vez que se han obtenido los dos números en su representación numérica completa, el código procede a utilizar el algoritmo de multiplicación binaria para obtener el resultado.

```
MOV AX, n1 ; multiplicando numero 1
MOV BX, n2 ; multiplicando numero 2
MOV CX, 0000H ; contador inicializado a 0
MOV DX, 0000H ; inicializar el registro de la suma a 0

LOOP_START:
  TEST BX, 0001H ; comprobar si el ultimo bit de multiplicando numero 2 esta establecido
  JZ SKIP_ADDITION ; saltar si el ultimo bit esta en 0
  ADD DX, AX ; sumar multiplicando numero 1 al registro de la suma
SKIP_ADDITION:
  SHL AX, 1 ; multiplicando numero 1 se desplaza un bit a la izquierda
  SHR BX, 1 ; multiplicando numero 2 se desplaza un bit a la derecha
  INC CX ; incrementar el contador
  CMP CX, 16 ; comprobar si se han realizado 16 iteraciones
  JL LOOP_START ; saltar al inicio del bucle si no se han realizado todas las iteraciones

MOV result, DX ; almacenar el resultado en la variable result
```

Este algoritmo consiste en realizar un desplazamiento a la izquierda del primer número (multiplicando número 1) y un desplazamiento a la derecha del segundo número (multiplicando número 2). Si el último bit del segundo número está establecido, se suma el multiplicando número 1 al registro de la suma. Este proceso se repite 16 veces (para números de 16 bits).

Una vez que se ha completado la multiplicación, el resultado se almacena en la variable "result" y se muestra en pantalla utilizando la interrupción INT 21h.



Conclusiones:

La realización de este programa puede ser vista como un ejercicio interesante para aprender a utilizar el lenguaje de ensamblador y la arquitectura x86. En particular, el algoritmo de multiplicación binaria implementado en este programa es un algoritmo eficiente para realizar la multiplicación de dos números en binario.

El algoritmo de multiplicación binaria es un buen ejemplo de cómo los algoritmos eficientes pueden ser implementados utilizando el lenguaje de ensamblador y aprovechando las características de la arquitectura de la computadora. Además, también es interesante notar cómo el lenguaje de ensamblador nos permite trabajar a un nivel muy bajo de abstracción, permitiéndonos tener un mayor control sobre los recursos de la computadora.

Bibliografía:

Brey, B. B. (2006). *Microprocesadores Intel : 8086/8088, 80186/80188, 80286, 80386 y 80486, Pentium, procesador Pentium Pro, Pentium II, Pentium III y Pentium 4: arquitectura, programación e interfaces*. Pearson Educación.