USO DEL SET DE INSTRUCCIONES INTEL X86 EN EMU8086

LABORATORIO Nº1

Juan José Niño Toro 201110101, Pedro Pablo Sanabria Paredes 201220728
Ingeniería Electrónica
Universidad Pedagógica y Tecnológica de Colombia
Juanjose.nino@uptc.edu.co
Pedro. Sanabria @uptc.edu.co

Resumen *En este laboratorio se realizan la creación de programas en assembler desde operaciones aritméticas básicas hasta cálculos de determinante para matrices 3x3 y suma y multiplicación de matrices 4x4.*

PALABRAS CLAVE: *microprocesador, stack, segmentos, instrucciones, modos de direccionamiento.*

I. INTRODUCCIÓN

Los procesadores Intel de la familia Intel x86 disponen de una amplia variedad de instrucciones que corresponden a un repertorio tipo CISC (ComplexInstruction Set Computer). En el estudio de los procesadores, es fundamental conocer los diferentes tipos de instrucciones, su forma de uso y los registros y banderas que se afectan durante su ejecución.

En esta guía se abordan 5grupos de instrucciones: las instruccionespara transferencia de datos, instrucciones lógicas y aritméticas, instrucciones para la manipulación de bits, instrucciones para la transferencia de cadenas de datos e instrucciones para el control de programas.

La familia de procesadores Intel x86 pueden sumar restar multiplicar y dividir datos tales como bytes palabras y dobles palabras con signo o sin signo.Además,se pueden modificar datos al nivel de bits mediante las operaciones lógicas.

El lenguaje ensamblador, o assembler (assembly language en inglés), es un lenguaje de programación de bajo nivel.
Los computadores, microprocesadores, microcontrolado res y otros circuitos integrados programables usan lenguaje ensamblador. El lenguaje implementa una representación simbólica de los códigos de máquina binarios y otras constantes necesarias para programar una arquitectura dada de CPU y constituye la representación más directa del código máquina específico para cada arquitectura legible por un programador. Esta representación es usualmente definida por el fabricante de hardware, y está basada en los mnemónicos que simbolizan los pasos de procesamiento (las instrucciones), los registros del procesador, las posiciones de memoria y otras características del lenguaje. Un lenguaje ensamblador es por lo tanto específico de cierta arquitectura de computador física (o virtual). Esto está en contraste con la mayoría de los lenguajes de programación de alto nivel, que idealmente son portátiles.

Fue usado principalmente en los inicios del desarrollo de software, cuando aún no se contaba con potentes lenguajes de alto nivel y los recursos eran limitados. Actualmente se utiliza con frecuencia en ambientes académicos y de investigación, especialmente cuando se requiere la manipulación directa de hardware, alto rendimiento, o un uso de recursos controlado y reducido. También es utilizado en el desarrollo de controladores de dispositivo (en inglés, device drivers) y en el desarrollo de sistemas operativos, debido a la necesidad del acceso directo a las instrucciones de la máquina. Muchos dispositivos programables (como los microcontroladores) aún cuentan con el ensamblador como la única manera de ser manipulados.

II. OBJETIVOS

- Conocer el entorno para el desarrollo de aplicaciones en ensamblador para Intelx86 denominadoEmu8086.
- Identificarlas características de funcionamiento de las instrucciones lógicas y aritméticas para Intelx86.
- Realizar un programa en ensamblador aplicando las instrucciones lógicas y aritméticas para Intel x86, sobre el emulador Emu8086.

III. ELEMENTOS Y EQUIPOS NECESARIOS

- Software *Emu8086*
- Computador Personal

IV. PROCEDIMIENTO

Ejercicios propuestos

1. Diseñe un programa en ensamblador que calcule el resultado de una operación matemática básica haciendo uso de las instrucciones aritméticas. Dicha operación debe tener como mínimo seis operandos y debe ser ingresada por teclado y su resultado debe ser visualizado por pantalla.

Para la solución del programa propuesto se realizaron los siguientes pasos:

Se definió el modelo de memoria "pequeño, .MODEL SMALL (64 KB como máximo)".

A continuación .Data define el inicio del segmento de datos donde se inicia a definir las variables necesarias para el funcionamiento del programa.

Luego se definió el .CODE comienzo del segmento de código, donde se inicializan las variable de los resultados para poder verlos en pantalla gracias a la INT 10 (video screen I/O) y se visualiza el mensaje de bienvenida, se hizo el código para capturar los números del teclado los cuales se convierten en binario y se restringe a máximo 2^16 bits lo cual nos deja hacer operaciones con números menores a esté, realizando comparaciones para saber que numero fue el ingresado, sacando un error si se ingresó un número mayor a 65535, además solo deja ingresar las teclas que corresponden a los números de computador esto se realiza cuando se está convirtiendo a binario donde se compara si es menor a 0 y mayor a 9.

Luego se definen las operaciones a realizar (Num1/Num2) + (Num3*Num4) – (Num5+Num6) este fue la configuración que escogimos.

Cada operación se hizo como una subrutina teniendo en cuenta que la división no está definida cuando el denominador es cero, por lo cual cuando es así retorna un error el cual se muestra en pantalla.

Para todas las operaciones se realiza una conversión de del resultado a ascii para visualizar en la pantalla, el cual se realiza con el proceso imprime el cual utiliza la función 09 de la interrupción Int 21h el cual nos imprime una cadena de caracteres.

Por último se realizó un proceso de los errores para saber que error fue el que ocurrió, y defina qué mensaje debe aparecer en pantalla, tenemos errores por digito no valido en la captura de los términos 0>x<9, error por fuera de rango > 65536, y división por cero.

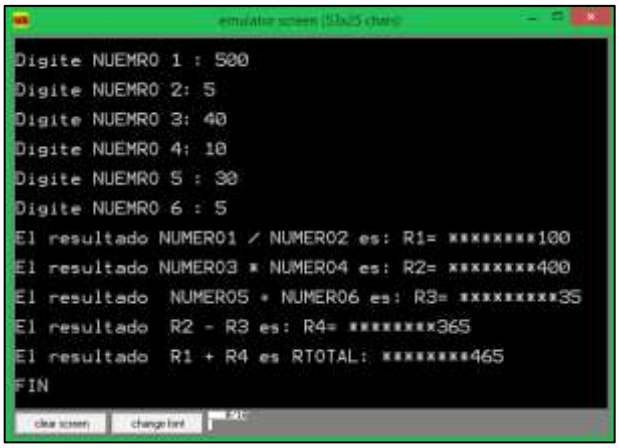

Figura 1. Simulación ejercicio 1.

En la figura 1 se observa la pantalla del ejercicio donde se observa los números ingresados por teclado según nuestra configuración

(Num1/Num2) + (Num3*Num4) – (Num5+Num6)

(500/5) + (40*10) – (30+5) = (R1) + (R2) – (R3)

(100) + (400) – (35) = 465.

2. Diseñe un programa en ensamblador que calcule la suma y la multiplicación de dos matrices 4 x 4. Los datos de entrada deben ser dispuestos en dos arreglos de memoria, así como las matrices de suma y multiplicación. Presente las matrices de entrada y los resultados en pantalla.

```
ARREGLO_1   DB  16  DUP (?)
ARREGLO_2   DB  16  DUP (?)
RESULTADO   DB  16  DUP (?)
MENSAJE_1   DB  'MATRIZ 1','$'
MENSAJE_2   DB  'MATRIZ 2','$'
MENSAJE_3   DB  'RESULTADO SUMA =','$'
MENSAJE_4   DB  'RESULTADO MULTIPLICACION=','$'
var_1 db ?
var_2 db ?
var_3 db ?
var_4 db ?
```

Figura 2. Segmento de datos

Se crean dos arreglos donde serán ingresados por teclado Arreglo_1 matriz A y Arreglo_2 matriz B ambas de tamaño 4x4.

En RESULTADO es donde se guardaran los valores resultantes de suma y multiplicación

entre las dos matrices, las variables creadas en el código anterior son útiles al calcular la multiplicación para guardar datos.

```
      MOV BX,OFFSET ARREGLO_1
      MOV CX,4
      CALL     MENSAJE_M_1
INICIO:

      PUSH     CX
      MOV CX,4
LEER_TECLADO:

      MOV AH, 01H
      INT 21H
      sub AL,30H
      MOV [BX],AL
      INC BX
      LOOP LEER_TECLADO

      POP CX
      CALL     ENTER_RET
      LOOP     INICIO
```

Figura 3. uso interrupción 21H función 1 para guardar los datos de la matriz 4x4

Usando la INT 21h / AH=1 se captan lo datos ingresados por teclado, los cuales son guardados en hexadecimal en el arreglo 1 en dirección apuntada por bx. Mediante direccionamiento indirecto por registro se repite el mismo proceso para la matriz B.

Se hace un process para la suma de las matrices donde por medio del apuntador SI se selecciona cada una de las posiciones de las matrices guardando su resultado en el arreglo RESULTADO

```
suma proc NEAR
    CALL     MENSAJE_M_3
         mov ax,0
         mov bx,0
         mov si,0
         mov cx,0
    suma1:
    mov al,ARREGLO_1[si]
    add al,ARREGLO_2[si]

    mov RESULTADO[si],al

    inc si
    inc bx
    cmp bx,16
    jne suma1
    mov bx,0
    mov si,0
```

Figura 4.process para la operación suma de las matrices 4x4

Usando la librería emu8086.inc visualizamos los datos del arreglo resultado

```
impresionsuma:
print " "
mov al,RESULTADO[si]
call print_num
inc si
inc bx
cmp bx,4
jne impresionsuma
printn " "
jmp impresionsuma1
```

Figura 5.visualización primera fila de la matriz resultado de la suma.

Se realiza este proceso 4 veces para visualizar los 16 elementos resultantes de la suma.

Para el proceso de multiplicación se toma fila de la matriz A con columna de la matriz B de la siguiente manera, variando las posiciones del arreglo.

```
multiplicacion proc
  multiplicacion1:

   mov al,ARREGLO_1[0]
   mov bl,ARREGLO_2[0]
   mul bl
   MOV VAR_1,al
   mov al,ARREGLO_1[1]
   mov bl,ARREGLO_2[4]
   mul bl
   MOV VAR_2,al
   mov al,ARREGLO_1[2]
   mov bl,ARREGLO_2[8]
   mul bl
   MOV VAR_3,al
   mov al,ARREGLO_1[3]
   mov bl,ARREGLO_2[12]
   mul bl
   MOV VAR_4,al
   MOV AX,0
   MOV BX,0
   add al,VAR_1
   add al,VAR_2
   add al,VAR_3
   add al,VAR_4
   mov RESULTADO[0],al
```

Figura 6.proceso de multiplicación fila por columna.

Cada uno de los resultados de la multiplicación son guardados en la matriz REULTADO que posteriormente será visualizada en pantalla.

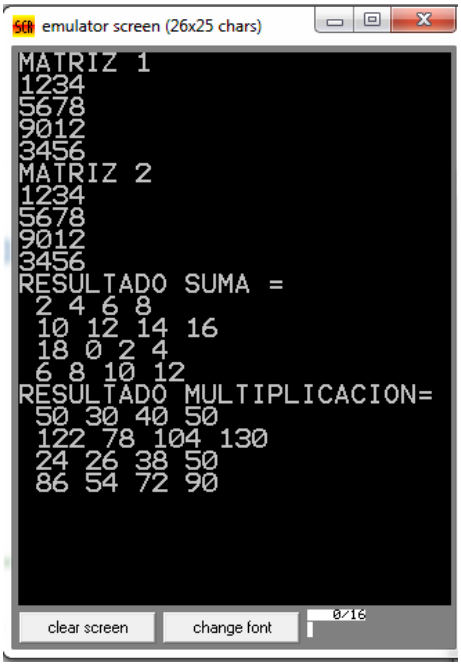

Figura 7. Simulación ejercicio 2.

3. Desarrolle un programa en ensamblador Evaluar el determinante de matrices 3X3. Presente las matrices de entrada y los resultados en pantalla.

Usando la INT 21h / AH=1 se captan lo datos ingresados por teclado, los cuales son guardados en el arreglo 1 en dirección apuntada por bx.mediante direccionamiento indirecto por registro.

```
      MOV BX,OFFSET ARREGLO_1
      MOV CX,3
      CALL     MENSAJE_M_1
INICIO:

      PUSH     CX
      MOV CX,3
LEER_TECLADO:

      MOV AH, 01H
      INT 21H
      MOV [BX],AL
      INC BX
      LOOP LEER_TECLADO

      POP CX
      CALL     ENTER_RET
      LOOP     INICIO
```

Figura 8.uso interrupción 21H función 1 para guardar los datos de la matriz 3x3.

Para el cálculo del determinante se extrae cada dato del arreglo por direccionamiento indirecto por registro usando a bx como apuntador,a este dato se le resta 30h se hacen las operaciones de multiplicación sumas y restas correspondientes obteniendo el resultado en el registro AL se realiza una conversión a bcd con la siguiente rutina donde se obtienen los valores de unidades ,decenas y centenas.

```
BIN_BCD PROC NEAR
        PUSH     CX
        PUSH     DX
        PUSH     BX
        PUSH     AX
        SUB      BX,BX
        SUB      DX,DX

        MOV      BL,AL
        MOV      CX,BX
        SUB      BX,BX

RESTA:
        ADD      BL,01H
        CMP      BL,11
        JAE      RESTA_MAS
        LOOP     RESTA

        CALL     MOSTRAR

        POP      AX
        POP      BX
        POP      DX
        POP      CX
        RET

RESTA_MAS:
        INC      DH
        SUB      BL,BL
        CMP      DH,11
        JAE      RESTA_MAS_MAS
        JMP      RESTA

RESTA_MAS_MAS:
        INC      BH
        SUB      BL,BL
        SUB      DH,DH
        JMP      RESTA_MAS
        RET
BIN_BCD ENDP
```

Figura 9.rutina retorna los valores en bh(centena),dh(decena),bl(unidades)

```
MOSTRAR   PROC NEAR
          PUSH     CX
          PUSH     DX
          PUSH     BX
          PUSH     AX
          MOV      AH,2
          MOV      DL,BH
          ADD      DL,30H
          INT      21H
          MOV      AH,2
          MOV      DL,DH
          ADD      DL,30H
          INT      21H
          MOV      AH,2
          MOV      DL,BL
          ADD      DL,30H
          INT      21H
          MOV      AH,2
          MOV      DL,32
          INT      21H
          POP      AX
          POP      BX
          POP      DX
          POP      CX
          RET
MOSTRAR ENDP
```

Figura 10.rutina para visualizar los datos en pantalla.

Obteniendo el resultado de unidades, decenas y centenas, se suma 30H para obtener el valor en ASCII y por medio de la función 2 de la interrupción 21H

enviamos los datos a visualizar como se observa en la figura 10.

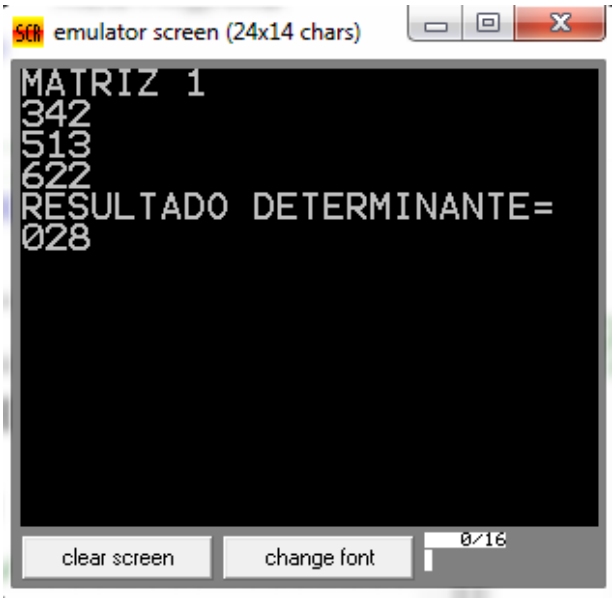

Figura 11.simulación calculo determinante 3x3.

V. CONCLUSIONES

- El emulador Emu8086 permite visualizar con facilidad la ejecución paso a paso de los programas desarrollados.
- Para programas pequeños el simulador Emu8086 es una buena opción, pero cuando los programas tienden a ser más grandes, la velocidad de ejecución lenta, lo convierte en una mala alternativa.
- La inclusión de las instrucciones aritméticas y lógicas permiten una gran ventaja en el desarrollo de los programas.
- La programación en ensamblador es la de más alto nivel, porque se tiene control de casi todo el proceso o programa que se ejecuta.
- Escogiendo de forma adecuada el modo de direccionamiento se logra reducir las líneas de código que suelen ser recurrentes.

VI. BIBLIOGRAFIA

[1] Barry B. Brey. The Intel microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro processor, Pentium II, Pentium III, Pentium 4, and Core2 with 64-bit extensions: architecture, programming, and interfacing. 8th ed.2009.

[2] ABEL, Peter. Lenguaje ensamblador y programación para IBM PC y compatibles. Pearson Educación, 1996.

[3] GODFREY, J. Ferry; TERRY, J. Lenguaje ensamblador para microcomputadoras IBM. Editorial Prentice-Hall, 1991.

[4] Ensamblador 8086/88. Universidad de Sevilla. http://www.cartagena99.com/recursos/programacion/apuntes/Esamblador8086.pdf