I. TEMA: PROCESAMIENTO DE ARREGLOS EN ENSAMBLADOR

II. OBJETIVOS DE LA PRACTICA

Al finalizar la presente práctica, el estudiante:

1. Explica las particularidades del almacenamiento de arreglos en los procesadores con arquitectura x86
2. Implementa aplicaciones para el procesamiento de arreglos bidimensionales utilizando el lenguaje de programación ensamblador para computadores con arquitectura x86

III. TRABAJO PREPARATORIO.

Para obtener mejores resultados, es necesario que el estudiante:

1. Conozca los conceptos básicos del modelo de memoria de los computadores x86
2. Conozca los modos de direccionamiento para el acceso a elementos de arreglos en procesadores con arquitectura x86.

IV. MATERIALES.

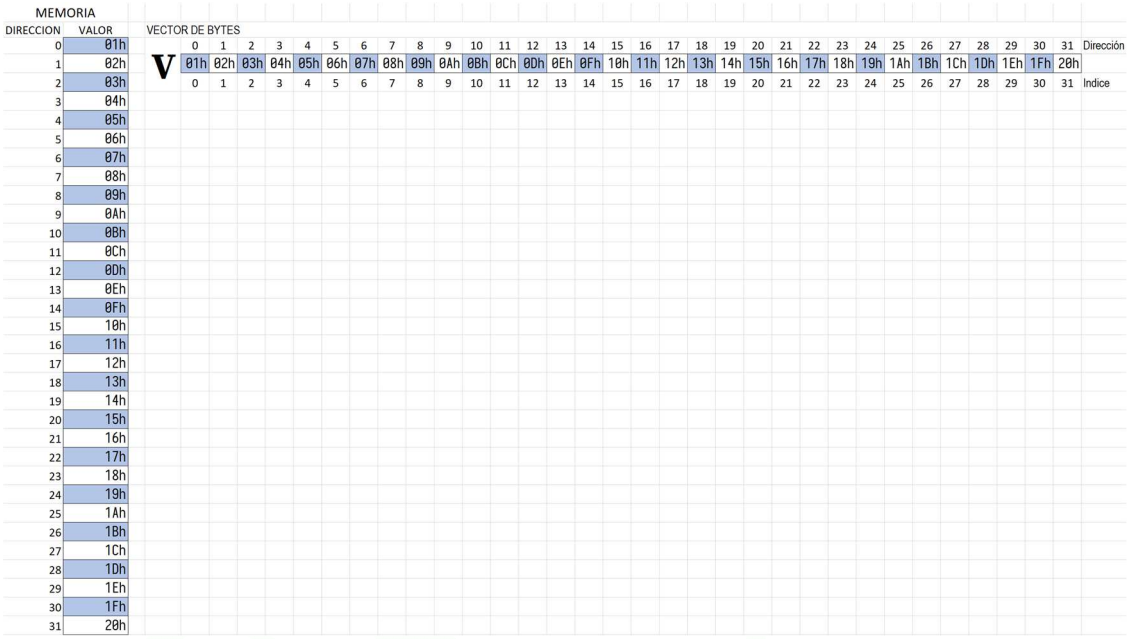
1. Sistema operativo Linux
2. Compilador NASM
3. Librería io.mac

V. MARCO TEORICO

ALMACENAMIENTO DE ARREGLOS EN MEMORIA

La memoria de un computador es una secuencia de bytes accesibles a través de una dirección. Esta puede representarse mediante una matriz unidimensional (vector) con facilidad.

Por ejemplo, un arreglo de bytes se puede almacenar y representar de la siguiente manera:

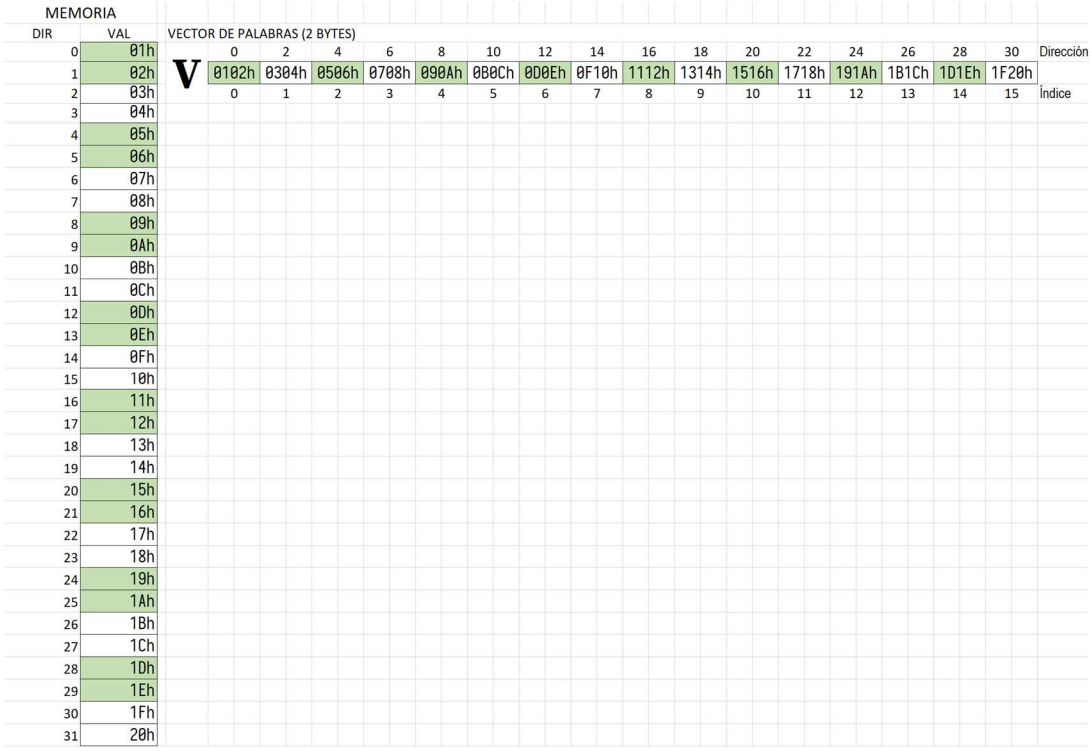


MEMORIA	
DIRECCION	VALOR
0	01h
1	02h
2	03h
3	04h
4	05h
5	06h
6	07h
7	08h
8	09h
9	0Ah
10	0Bh
11	0Ch
12	0Dh
13	0Eh
14	0Fh
15	10h
16	11h
17	12h
18	13h
19	14h
20	15h
21	16h
22	17h
23	18h
24	19h
25	1Ah
26	1Bh
27	1Ch
28	1Dh
29	1Eh
30	1Fh
31	20h

VECTOR DE BYTES

V	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	Dirección
	01h	02h	03h	04h	05h	06h	07h	08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh	10h	11h	12h	13h	14h	15h	16h	17h	18h	19h	1Ah	1Bh	1Ch	1Dh	1Eh	1Fh	20h	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	Indice

un arreglo de palabras se puede representar de la siguiente manera:



MEMORIA	
DIR	VAL
0	01h
1	02h
2	03h
3	04h
4	05h
5	06h
6	07h
7	08h
8	09h
9	0Ah
10	0Bh
11	0Ch
12	0Dh
13	0Eh
14	0Fh
15	10h
16	11h
17	12h
18	13h
19	14h
20	15h
21	16h
22	17h
23	18h
24	19h
25	1Ah
26	1Bh
27	1Ch
28	1Dh
29	1Eh
30	1Fh
31	20h

VECTOR DE PALABRAS (2 BYTES)

V	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	Dirección
	0102h	0304h	0506h	0708h	090Ah	0B0Ch	0D0Eh	0F10h	1112h	1314h	1516h	1718h	191Ah	1B1Ch	1D1Eh	1F20h	
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Índice

Un arreglo de palabras se puede representar de la siguiente manera:

MEMORIA		VECTOR DE PALABRA DOBLE (4 BYTES)								
DIR	VAL		0	4	8	12	16	20	24	28	Dirección
0	01h	V	01020304h	05060708h	090A0B0Ch	0D0E0F10h	11121314h	15161718h	191A1B1Ch	1D1E1F20h	
1	02h		0	1	2	3	4	5	6	7	Índice
2	03h										
3	04h										
4	05h										
5	06h										
6	07h										
7	08h										
8	09h										
9	0Ah										
10	0Bh										
11	0Ch										
12	0Dh										
13	0Eh										
14	0Fh										
15	10h										
16	11h										
17	12h										
18	13h										
19	14h										
20	15h										
21	16h										
22	17h										
23	18h										
24	19h										
25	1Ah										
26	1Bh										
27	1Ch										
28	1Dh										
29	1Eh										
30	1Fh										
31	20h										

En cuanto a matrices multidimensionales, la representación, interpretación y procesamiento es responsabilidad del lenguaje de programación.

En el caso del lenguaje ensamblador, el programador debe decidir estos detalles.

Por ejemplo, un arreglo bidimensional puede representarse de la siguiente manera, si los datos se almacenan en memoria por filas:

MEMORIA		MATRIZ DE 2x4 DE PALABRA DOBLE ALMACENANDO POR FILAS				
DIR	VAL		0	1	2	3	
0	01h	M	01020304h	05060708h	090A0B0Ch	0D0E0F10h	0
1	02h		11121314h	15161718h	191A1B1Ch	1D1E1F20h	1
2	03h						
3	04h						
4	05h						
5	06h						
6	07h						
7	08h						
8	09h						
9	0Ah						
10	0Bh						
11	0Ch						
12	0Dh						
13	0Eh						
14	0Fh						
15	10h						
16	11h						
17	12h						
18	13h						
19	14h						
20	15h						
21	16h						
22	17h						
23	18h						
24	19h						
25	1Ah						
26	1Bh						
27	1Ch						
28	1Dh						
29	1Eh						
30	1Fh						
31	20h						

El mismo arreglo, pero almacenando los datos por columnas:

MEMORIA	
DIR	VAL
0	01h
1	02h
2	03h
3	04h
4	05h
5	06h
6	07h
7	08h
8	09h
9	0Ah
10	0Bh
11	0Ch
12	0Dh
13	0Eh
14	0Fh
15	10h
16	11h
17	12h
18	13h
19	14h
20	15h
21	16h
22	17h
23	18h
24	19h
25	1Ah
26	1Bh
27	1Ch
28	1Dh
29	1Eh
30	1Fh
31	20h

MATRIZ DE 2x4 DE PALABRA DOBLE ALMACENANDO POR COLUMNAS

M	0	1	2	3	
	01020304h	090A0B0Ch	11121314h	191A1B1Ch	0
	05060708h	0D0E0F10h	15161718h	1D1E1F20h	1

VI. TRABAJO DE LABORATORIO.

1. Escriba un programa que permita crear una matriz bidimensional M x N de enteros de 16 bits. M y N deben ser definidos por el usuario en tiempo de ejecución. Los datos deben ingresarse en tiempo de ejecución y finalmente, debe mostrarse la matriz

Solución

```
;Nombre     :    matrizBi.asm
;Proposito  :    muestra el manejo de matrices bidimensionales
;Autor      :    Edwin Carrasco
;FCreacion  :    29/12/2021
;FModific.  :    ---
;Compilar   :
;           nasm -f elf matrizBi.asm -Wall
;           ld -m elf_i386 -s -o matrizBi matrizBi.o io.o
;           ./matrizBi

%include "io.mac"


section .bss
   matriz: resw 200 ; Reservar 200 palabras de 2 bytes
   m: resw 1    ; Numero de filas
   n: resw 1    ; numero de columnas
   i: resw 1    ; Indice de filas
   j: resw 1    ; Indice de columnas


section .data
   msgFilas: db "Ingrese el nro de filas de la matriz : ",0
   msgColumnas: db "Ingrese el nro de columnas de la matriz : ",0
   msgElementos:  db "Ingrese los elementos de uno en uno (fila
                        por fila) : ",10,0
   tab: db  "",9,0 ;Tabulacion entre columnas


section .text
  global _start

_start:
   ;Leer nro de filas
   PutStr msgFilas
   GetInt word [m]

   ;Leer nro de columnas
   PutStr msgColumnas
   GetInt word [n]

   ;Pedir elementos de la matriz
   PutStr msgElementos

   ;Leer cada elemento de la matriz...
   mov eax, 0
```

```
    mov ebx, matriz  ; Direccion base de la matriz

    mov word[i], 0
    mov word[j], 0

bucle_filas:
    mov word[j], 0

bucle_columnas:
    GetInt dx

    ;eax es el indice de la matriz y cada elemento es de 2 bytes
    (1 palabra)
    mov  word[ebx + 2 * eax], dx
    inc eax      ;Actualizar indice

    inc word[j]
    mov cx, word[j]
    cmp cx, word[n]
    jb bucle_columnas

    inc word[i]
    mov cx, word[i]
    cmp cx, word[m]
    jb bucle_filas

    ;Leer cada elemento de la matriz y almacenarlos por filas
    mov eax, 0
    mov ebx, matriz

    mov word[i], 0
    mov word[j], 0
    nwln

    ;Mostrar matriz
bucle_filas2:
    mov word[j], 0

bucle_columnas2:
    ;eax es el indice de la matriz y cada elemento es de 2 bytes
    PutInt word[ebx+2*eax]

    PutStr tab   ;Imprime un espacio despues de cada elemento
    inc eax

    inc word[j]
    mov cx, word[j]
    cmp cx, word[n]
    jb bucle_columnas2
    nwln

    inc word[i]
    mov cx, word[i]
    cmp cx, word[m]
    jb bucle_filas2
    nwln
```

```
    ;Salir del programa
exit:
    mov eax, 1
    mov ebx, 0
    int 80h
```

PRUEBA DE FUNCIONAMIENTO:

```
ecp@ecp-OAC:~/codigo_ASM/arreglos$ #Pruebas de funcionamiento
ecp@ecp-OAC:~/codigo_ASM/arreglos$ nasm -f elf matrizBi.asm -Wall
ecp@ecp-OAC:~/codigo_ASM/arreglos$ ld -m elf_i386 -s -o matrizBi matrizBi.o io.o
ecp@ecp-OAC:~/codigo_ASM/arreglos$ ./matrizBi
Ingrese el nro de filas de la matriz : 2
Ingrese el nro de columnas de la matriz : 4
Ingrese los elementos de uno en uno (fila por fila) :
1
2
3
4
5
6
7
8

1       2       3       4
5       6       7       8
```

VII. PRACTICAS DE LABORATORIO

1. Escriba un programa que sume dos matrices bidimensionales MxN y muestre tanto las matrices de entrada, como la matriz de salida. M y N se definen en tiempo de ejecución, al igual que los datos de las matrices.

2. Escriba un programa que indique la fila y columna en la que se encuentra el primer elemento múltiplo de 17, en una matriz de MxN. M, N y los elementos de la matriz deben ser ingresados por el usuario en tiempo de ejecución.

3. Escriba un programa que multiplique dos matrices A y B. las dimensiones de A son M x N y de B son N x P. M, N, P y los datos de las matrices se definen en tiempo de ejecución. El programa debe mostrar A, B y la matriz resultante.

VIII. EVALUACION

La evaluación de las actividades realizadas en la presente guía de práctica se hará en función de la siguiente tabla:

ACTIVIDAD	Procedimental	
	Sesión 01	Sesión 02
Resolución del ejercicio propuesto 01	--	06
Resolución del ejercicio propuesto 02	--	06
Resolución del ejercicio propuesto 03	--	08
TOTAL	**--**	**20**

IX. BIBLIOGRAFIA

1. Anvin, P. http://nasm.sourceforge.net/. Sitio web del compilador NASM.
2. Bartlett, J. "*Programming From The Ground Up*". Bartlett Publishing, 2003
3. Brey, B. "*Los Microprocesadores Intel. Arquitectura, Programación e Interfaces*". Prentice Hall 3Ed.
4. Carter, P. "*PC Assembly Language*" 2005
5. Dandamudi. "*Guide To Assembly Language Programming In Linux*". Springer 2005
6. Hyde, R. "*Art of Assembly Language Programming*". Nostarch Press 1Ed.
7. Leto J. "*Writing a useful Program With NASM*". http://leto.net/writing/nasm.php.
8. Smith B. E., Johnson M. T. "*Programming the Intel 80386*" Editorial Scott, Foresman And Company, 1987.
9. Toal R. *"NASM Tutorial"*. https://cs.lmu.edu/~ray/notes/nasmtutorial/.
10. Unknown. "*Matrix in NASM*". https://yazary.blogspot.com/2013/01/matrix-in-nasm.html.