EJERCICIOS

2- Indique cuáles de los siguientes direccionamientos son correctos:

Instrucción	¿Correcto?
`MOV esi, [esi+1]`	SÍ
`MOV esi, esi+1`	NO
`SUB [esi], [edi + ebx]`	NO
`ADD esi, [edi + edx]`	SÍ
`MOV ax, var + 1`	SÍ
`MOV esi, [esi + 2] - esi`	NO
`OR ebx, [ebp - 2]`	SÍ
`MOV [esi], 4`	SÍ
`MOV 4, [esi]`	NO
`MUL [esi + ebx + ebp]`	NO
`AND ax, [esi - ebx]`	NO

5- Dadas las declaraciones mostradas, escriba en ensamblador la evaluación de las siguientes expresiones (trate de optimizar los cálculos):

```
int   a, b, c, d, e, f;
char  x, y;
```

Expresión	

`(a + b)*(a + c)*(a + d)`	`mov eax, a`
	`mov ebx, eax`
	`add ebx, b`
	`mov ecx, eax`
	`add ecx, c`
	`imul ebx, ecx`
	`add eax, d`
	`imul eax, ebx`

`(a + b)*(c - d) + (a + b)* (e + f)`	`mov eax, a`
	`add eax, b`
	`mov ebx, c`
	`sub ebx, d`
	`imul ebx, eax`
	`mov ecx, e`
	`add ecx, f`
	`imul ecx, eax`
	`add ebx, ecx`

`(a + b)*(c - d) + (a + b)* (c + d)`	`mov eax, a`
	`add eax, b`
	`mov ecx, c`
	`mov edx, d`
	`mov ebx, ecx`
	`sub ebx, edx`
	`add ecx, edx`
	`imul ebx, eax`
	`imul ecx, eax`
	`add ebx, ecx`

`a - - b`	`mov eax, a` `add eax, b`
`x + y`	`mov al, x`

	add al, y
x – 'a'	mov al, x sub al, 'a'

x * y	movsx ax, x
	movsx bx, y
	imul ax, bx

x + a	movsx eax, x add eax, a

6- En el registro `al` se tiene un dígito- carácter ('0' a '9') codificado en ASCII. Calcule su valor numérico (0 a 9) y déjelo en `al` mismo.

```
and al, 0FH
ó
sub al, '0'
```

7- Dadas las declaraciones mostradas, analice las siguientes instrucciones de C; diga si compilan, y, si es así, tradúzcalas a ensamblador.

```
int   a, b;
char  x;
char * p = &x, *q;
```

Expresión	¿Compila?

a = b++;	mov eax, b
	mov a, eax
	inc eax
	mov b, eax

a = a++; Resultado indefinido. Se puede hacer de varias maneras.	inc a ó nada!
p+1 = q + 1;	NO

*(p+1) = *(q+1);	mov eax, p
	mov ebx, q
	inc eax
	inc ebx
	mov cl, [ebx]
	mov [eax], cl

q = p++;	mov eax, p
	mov q, eax
	inc eax
	mov p, eax

*p++ =5;	mov eax, p
	mov [eax], 5
	inc eax
	mov p, eax

(*p)++ = 'a';	NO

(*(++q))++;	mov eax, q
	inc eax
	mov q, eax
	inc [eax]

q = *p++;	NO
p++ = q;	NO

8- Dadas las declaraciones mostradas, analice las siguientes instrucciones de C; diga si compilan, y, si es así, tradúzcalas a ensamblador.

```
int   a, i, j;
```

```
int   v[100];
int  * p, *q;
```

Expresión	¿Compila?

`v[6] = v[i];`	`mov esi, i`
	`mov eax, v[4*esi]`
	`mov v+24, eax`

`v[i] += a;`	`mov eax, a`
	`mov esi, i`
	`add v[4*esi], eax`

`p = v + i;`	`mov esi, i`
	`lea eax, v[4*esi]`
	`mov p, eax`

`v = p + i;`	NO

`*p = *(v + i);`	`mov esi, i`
	`mov eax, v[4*esi]`
	`mov ebx, p`
	`mov [ebx], eax`

`*v = *(p + i);`	`mov esi, i`
	`mov ebx, p`
	`mov eax, [ebx+4*esi]`
	`mov v, eax`

`p[i] = a;`	`mov esi, i`
	`mov ebx, p`
	`mov eax, a`
	`mov [ebx+4*esi], eax`

`v[j] = a*(v[i] + v[j]);`	`mov esi, i`
	`mov eax, v[4*esi]`
	`mov esi, j`
	`add eax, v[4*esi]`
	`imul eax, a`
	`mov v[4*esi], eax`

`a = v[v[i]];`	`mov esi, i`
	`mov esi, v[4*esi]`
	`mov esi, v[4*esi]`
	`mov a, esi`

`p = 3*q;`	NO

9- Se tiene la siguiente declaración para un vector:

```
v   byte  100 dup (?)
```

a- Escriba código en ensamblador para generar un apuntador a `v[i]`; `i` es una variable entera (de tipo `dword`). El apuntador se debe dejar en `eax`.

```
mov esi, i
lea eax, v[esi]
```

b- Repita el problema anterior con un vector de dobles palabras (`dword`).

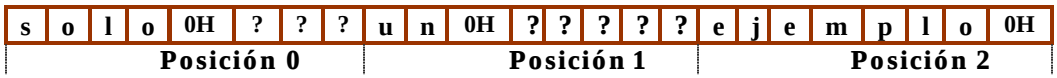
```
mov esi, i
lea eax, v[4*esi]
```

c- Repita el problema anterior con un vector de elementos de tamaño *t*.

```
mov esi, i
imul esi, t
lea eax, v[esi]
```

11- Se tienen cadena de caracteres representadas con convenciones de C.

a- Una forma de representar un vector de cadenas de caracteres es reservar un espacio máximo para cada cadena. Por ejemplo, el vector que se muestra a continuación, tiene tres cadenas cada una con 8 bytes disponibles (pero no necesariamente usados):



Las posiciones marcadas con "?" no están ocupadas.

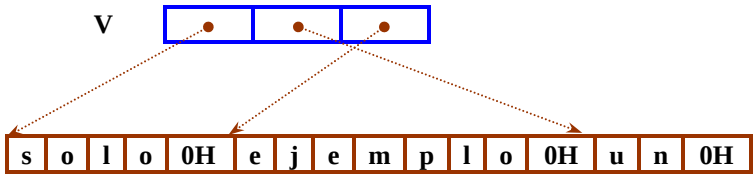
- Se tiene un apuntador `p` al comienzo del vector, escriba código en ensamblador para que quede apuntando al *i*-ésimo elemento (`i` es una variable entera).

```
mov esi, i
lea eax, v[8*esi]
```

- Escriba código en ensamblador para poner un blanco en la posición `j` de la *i*-ésima cadena (`i` y `j` son variables enteras).

```
mov esi, i
lea eax, v[8*esi]
mov esi, j
mov [eax+esi], ' '
```

b- Otra forma de representar vectores de cadenas es por medio de un vector de apuntadores a las cadenas. En este caso, solo se reserva el mínimo espacio requerido por cada cadena:



Repita el punto anterior con esta representación.

```
mov esi, i
mov eax, v[4*esi]
mov esi, j
mov [eax+esi], ' '
```

21- Se quiere manejar números de 3 bytes. Los números se almacenan en variables del estilo:

```
n     byte  3 dup(?)
```

El formato de los números es *little endian* —byte menos significativo "a la izquierda"—.

a- Desarrolle código en ensamblador para leer un número de 3 bytes en `eax`.

```
mov eax, n
and eax, 00FFFFFFH
```

b- Desarrolle código en ensamblador para escribir un número de 3 bytes que se encuentra en `eax` en una variable de 3 bytes en memoria.

```
mov n, ax
shr eax, 16
mov n+2, al
```