Universidad de los Andes

Ingeniería de Sistemas y Computación
ISIS1304 – Fundamentos de Infraestructura Tecnológica
Banco – Ensamblador control

Capacidades evaluadas:

- *Manejo correcto de direccionamiento (registros, memoria, índices, etc.)*
- *Manejo correcto de condiciones y saltos (instrucciones condicionales, ciclos)*
- *Algorítmica correcta*

Se tiene un vector `v` de apuntadores a cadenas de caracteres, y `s` un apuntador a una cadena de caracteres. El tamaño del vector está en la variable `n`.

Escriba código en ensamblador para buscar si la cadena apuntada por `s` es igual a alguna de las cadenas apuntadas por `v`. Si alguna de las cadenas es igual, se debe retornar el índice de `v` donde se encontró; si no, se debe retornar -1.

```
char *v[TAMMAX];
char *s
int n;
...
__asm  {
mov eax, 0
recorrerVector:
cmp eax, n
je noEsta
  mov esi, s
  mov edi, v[4*eax]
  compararChar:
    mov bl, [esi]
    cmp bl, [edi]
    jne siguienteCadena
    cmp bl, 0
    je esta
      inc esi
      inc edi
  jmp compararChar
  siguienteCadena:
  inc eax
jmp recorrerVector
noEsta:
mov eax, -1
esta:
```

Se tiene un vector `v` de apuntadores a cadenas de caracteres, y un vector `longitud` de enteros.

Se quiere calcular la longitud de cada una de las cadenas de `v`, y dejar el resultado en la posición correspondiente de longitud. Es decir, la poscondición es:

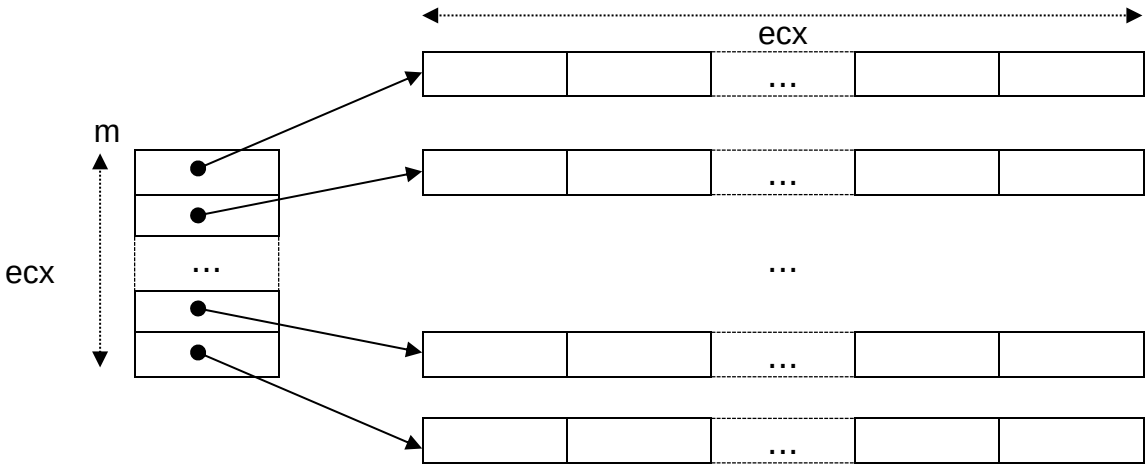
longitud[i] = longitud de la cadena v[i]

El tamaño de los dos vectores está en la variable `n` (el número de posiciones efectivamente usadas; `TAMMAX` es el máximo posible).

Escriba en ensamblador el código correspondiente.

```
char *v[TAMMAX];    //TAMMAX es una constante
int longitud[TAMMAX]
int n;
...
__asm  {
```

Se tiene una matriz representada por medio de un vector `m` de apuntadores a vectores de enteros.



En `ecx` se tiene la dimensión de la matriz (es cuadrada), y en `ebx` se tiene un valor que se quiere buscar en la matriz.

Escriba un programa en ensamblador para buscar la primera ocurrencia de `ebx` en la matriz. Si lo encuentra, el programa debe retornar en `eax` un apuntador a dicho elemento; si no, debe retornar el apuntador nulo.

```
mov esi, 0
recorrerFilas:
cmp esi, ecx
je finFilas
  mov eax, m[esi*4]
  mov edi, 0
  recorrerColumnas:
```

```
  cmp edi, ecx
  je finColumnas
    cmp ebx, [eax]
    je encontrado
    add eax, 4
    inc edi
  jmp recorrerColumnas
  finColumnas:
  inc esi
jmp recorrerFilas
finFilas:
  mov eax, 0
encontrado:
```

En `esi` se tiene un apuntador a un vector, en `ecx` se tiene el número de elementos del vector y en `ebx` se tiene un cierto valor.

Escriba un programa en ensamblador para contar cuántas veces aparece el valor `ebx` en el vector apuntado por `esi`.

En `esi` y `edi` se tienen apuntadores a dos matrices de tamaño N x N. Escriba un programa en ensamblador para sumar la matriz apuntada por `esi` con la matriz apuntada por `edi` dejando el resultado en la primera (en la matriz apuntada por `esi`).

Suponga que está declarada una variable entera `N` que especifica el tamaño de la matriz (N x N).

En `esi` y `edi` se tienen apuntadores a dos cadenas de caracteres. Escriba un programa en ensamblador que busca en la cadena apuntada por `esi` la primera ocurrencia de un carácter que aparezca también en la cadena apuntada por `edi`. `esi` debe quedar apuntando al carácter en cuestión; si ningún carácter de la cadena apuntada por esi aparece en la otra cadena, `esi` debe quedar en el apuntador nulo (es decir, cero).

Por ejemplo, si `esi` apunta a `"abcde"` y `edi` apunta a `"12db3"`, `esi` quedaría apuntando a la `'b'` (`"a`b`cde"`).

```
recorrer1:
mov bl, [esi]
cmp bl, 0
je noHay
  mov eax, edi
  recorrer2:
  mov bh, [eax]
  cmp bh, 0
  je siguienteChar
    cmp bl, bh
    je fin
    inc eax
  jmp recorrer2
  siguienteChar:
  inc esi
```

```
jmp recorrer1
noHay:
mov esi, 0
fin:
```

Dada una matriz de enteros, de `n` filas y `m` columnas (A_{nxm}), se quiere saber cuántas filas están en cero (todos los elementos de la fila valen 0). Escriba en ensamblador el código correspondiente.

```
int filasEnCero (int *A, int n, int m)
{
	int cont = 0;
	__asm
	{
		mov ebx, A
		mov esi, 0		// i = 0
	ini1:	cmp esi, n		// MQ i < n
		jz fin1
		mov ecx, 0		// sumaParcial = 0
		mov edi, 0		// j = 0
	ini2:	cmp edi, m		// MQ j < m
		jz fin2
		mov eax, esi // eax --> pos = i*m +j
		imul eax, m
		add eax, edi
		cmp [ebx + eax * 4], 0	// mat [pos] == 0
		jnz cont1
		inc edi			// j++
		jmp ini2
	fin2:	inc cont
	cont1:	inc esi			// i++
		jmp ini1
	fin1:
	}
	return cont;
}
```

Desarrolle un programa en ensamblador para verificar si una cadena es prefijo de otra; por ejemplo, "casa" es prefijo de "casados".

En concreto, `esi` y `edi` apuntan a sendas cadenas de caracteres. Se debe verificar si la cadena apuntada por `esi` es prefijo de la cadena apuntada por `edi`.

Si es prefijo, debe poner `eax` en 1; y en 0, si no.

Desarrolle un programa en ensamblador para eliminar un cierto número de caracteres en una cadena a partir de una posición. El programa debe tener la posición inicial y el número de caracteres que debe borrar. Las posiciones se numeran desde cero, por ejemplo, si hay que borrar 3 caracteres desde la posición 1 en la cadena "mañana", queda "mna".

En concreto, `esi` apunta a una cadena de caracteres, `ebx` indica el carácter a partir del cual se debe empezar a eliminar y `ecx` indica cuántos caracteres se deben eliminar.

```
mov edi, esi
add edi, ebx
iterar:
 mov al, [edi+ecx]
 mov [edi], al
 inc edi
 cmp al, 0
 jne iterar
```

Se dispone de un vector `v` de apuntadores a vectores de enteros.

En `ecx` se tiene el tamaño (número de elementos) del vector `v`; en `edx` se tiene el tamaño (número de elementos) de cada uno de los vectores de enteros.

Escriba un programa en ensamblador que recorre todos los vectores de enteros modificando cada posición de la siguiente manera: si el elemento es menor que cero, le asigna `-1`; si es mayor que cero, le asigna `1`; si es cero no lo modifica.

```
mov esi, 0
siguienteVector:
cmp esi, ecx
je fin
  mov edi, v[4*esi]
  mov ebx, 0
  siguienteElemento:
  cmp ebx, edx
  je finVector
    mov eax, [edi+4*ebx]
    cmp eax, 0
    jl esNegativo
    jg esPositivo
    jmp incrementar
    esNegativo:
      mov [edi+4*ebx], -1
      jmp incrementar
    espositivo:
      mov [edi+4*ebx], 1
    incrementar:
    inc ebx
  jmp siguienteElemento
  finVector:
  inc esi
jmp siguienteVector
fin:
```

Se dispone de un vector `v` de apuntadores a cadenas de caracteres, y en `al` se tiene un carácter.

Escriba un programa que busca si el carácter en `al` se encuentra en alguna de las cadenas apuntadas por el vector `v`. Si se encuentra, `esi` debe quedar apuntando al carácter en cuestión; si no, `esi` debe quedar con el apuntador nulo.

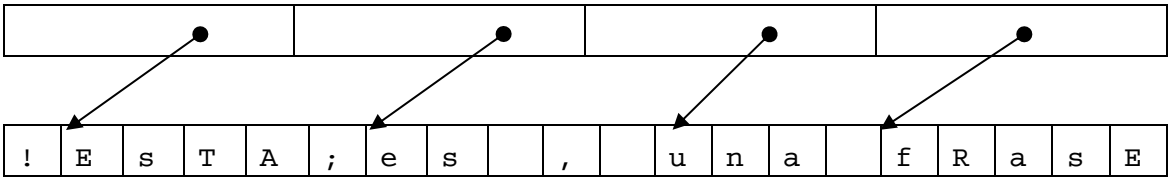
```
mov ebx, 0
recorrerVector:
cmp ebx, ecx
je noEsta
  mov esi, v[4*ebx]
  recorrerCadena:
    cmp [esi], 0
    je fincadena
      cmp [esi], al
      je finVector
        inc esi
  jmp recorrerCadena
  fincadena:
  inc ebx
jmp recorrerVector
noEsta:
  mov esi, 0
finVector:
```

En `esi` se tiene un apuntador a una cadena de caracteres. Además, se tiene un vector `v` de apuntadores.

Escriba un programa en ensamblador que recorre la cadena apuntada por `esi` identificando las palabras que la componen; cada vez que identifica el comienzo de una palabra, guarda en `v` un apuntador a ese sitio.

Una palabra es una secuencia de caracteres alfabéticos (mayúsculas y minúsculas). Todos los demás caracteres sirven de separadores de palabras y no se tienen en cuenta.

Ejemplo: dada la cadena "`!EsTA;es , una fRasE`", el vector de apuntadores debería quedar así:



Suponga que v tiene el tamaño suficiente para almacenar todos los apuntadores que sea necesario.

Esta es una posible solución; hay otras. En particular, se podría hacer más eficiente, pero esta solución no supone ningún conocimiento sobre el valor de los códigos ASCII ni sobre el orden relativo entre mayúsculas y minúsculas.

```
mov edi, 0
mov al, [esi]
examinarSiguiente:
cmp al, 0          ;¿Fin de cadena?
je terminar
  cmp al, 'a'      ;Si es menor que 'a', no es una minúscula, pero
  jl verMayuscula  ;puede ser una mayúscula.
  cmp al, 'z'      ;Si está entre 'a' y 'z', es una letra.
  jle esLetra
  verMayuscula:
  cmp al, 'A'      ;Dado que no es minúscula, y que es menor que 'A',
  jl esSimbolo     ;no es letra, luego es símbolo o nulo.
  cmp al, 'Z'      ;Si está entre 'A' y 'Z', es una letra.
  jg esSimbolo
```

```
  esLetra:        ;Primera letra que aparece, comienzo de palabra,
    mov v[4*edi], esi   ;luego se guarda el apuntador.
    inc edi
    siguienteCaracter:
      inc esi
      mov al, [esi]     ;Se toma el siguiente carácter.
    cmp al, 'a'
    jl verMayuscula2    ;Se continúa con este recorrido interno
mientras
    cmp al, 'z'         ;que sean letras (para eliminar el resto de la
    jle siguienteCaracter     ; palabra).
    verMayuscula2:      ;Si no es letra, se va al ciclo externo para
    cmp al, 'A'         ;revisar el fin de cadena y para saltarse otros
    jl examinarSiguiente      ;símbolos que pueda haber.
    cmp al, 'Z'
    jg examinarSiguiente
    jmp siguienteCaracter
  esSimbolo:            ;Si es símbolo, se pasa al siguiente carácter.
    inc esi
    mov al, [esi]
jmp examinarSiguiente
terminar:
```

En `esi` se tiene un apuntador a una cadena de caracteres. Escriba un programa en ensamblador que recorre la cadena apuntada por `esi` cambiando los caracteres de minúsculas a mayúsculas y viceversa. Note que puede haber caracteres no alfabéticos. Además, se sabe que el espacio reservado para la cadena es 100 bytes; por ende, el procesamiento termina cuando se acabe la cadena o cuando se llegue al límite del espacio reservado.

Por ejemplo, si `esi` apunta a “Abc - De”, la cadena se convertiría en (“aBC - dE”).

Hay diversas formas de hacerlo; la solución presentada no es la mejor, pero tampoco supone conocimiento sobre el orden en ASCII de las mayúsculas y las minúsculas.

```
mov ecx, 0
recorrer:
mov al, [esi+ecx]
cmp al, 0
je fin
cmp ecx, 100
je fin
  cmp al, 'A'
  jl revisarMin
   cmp al, 'Z'
  jg revisarMin
    add al, 'a'-'A'
    mov [esi+ecx], al
    inc ecx
  jmp recorrer
  revisarMin:
  cmp al, 'a'
  jl otro
   cmp al, 'z'
  jg otro
    add al, 'A'-'a'
    mov [esi+ecx], al
  otro:
    inc ecx
```

```
jmp recorrer
```

En `esi` se tiene un apuntador a una cadena de caracteres; la cadena puede estar compuesta de cualquier carácter (letra, símbolo, dígito, etc.).

Se quiere contar cuántas ocurrencias de cada letra hay (sin importar si es mayúscula o minúscula). Para esto se dispone de un vector de enteros `v` de 26 posiciones:

```
int v[26];
```

En la posición `v[0]` se cuentan las `'a'` (o `'A'`), en `v[1]` las `'b'` (o `'B'`), y así hasta `v[25]` que cuenta las `'z'` o `'Z'`.

Escriba un programa en ensamblador que realice esta tarea. Puede suponer que el vector v está inicializado en ceros.

En `esi` se tiene un apuntador a una cadena de caracteres (siguiendo la convención C); la cadena está compuesta solamente por caracteres alfabéticos (mayúsculas y minúsculas). Además, se tiene un vector de enteros (`cuenta`) de 26 posiciones.

Escriba un programa que cuente el número de ocurrencias de cada carácter en la cadena apuntada por `esi`. El conteo se lleva en el vector `cuenta` (las ocurrencias de 'a' se llevan en `cuenta[0]`, las de 'b' en `cuenta[1]`, etc.) El conteo se realiza sin importar si la letra es mayúscula o minúscula; por ejemplo, tanto las ocurrencias de 'a' y como las de 'A' se acumulan en `cuenta[0]`.

Dos versiones entre muchas posibles.

Versión 1:

```
mov eax, 0
while:
mov al, [esi]
cmp al, 0  ; ¿[esi] != '\0'? (¿se acabó la cadena?)
je finWhile    ; Es igual; terminar  (se acabó la cadena)
               ; Es diferente, ejecutar el while
  cmp al, 'a'  ; ¿[esi] >= 'a'?
  jae esMinuscula    ; es mayor o igual, luego es minúscula
                     ; es menor, luego es mayúscula
    add al, 32    ; Convertir a minúscula (¿Por qué funciona?)
  esMinuscula:
  sub eax, 'a'  ; Calcular posición en el alfabeto (¿Por qué?)
  inc cuenta[4*eax]
  inc esi
  jmp while
finWhile:
```

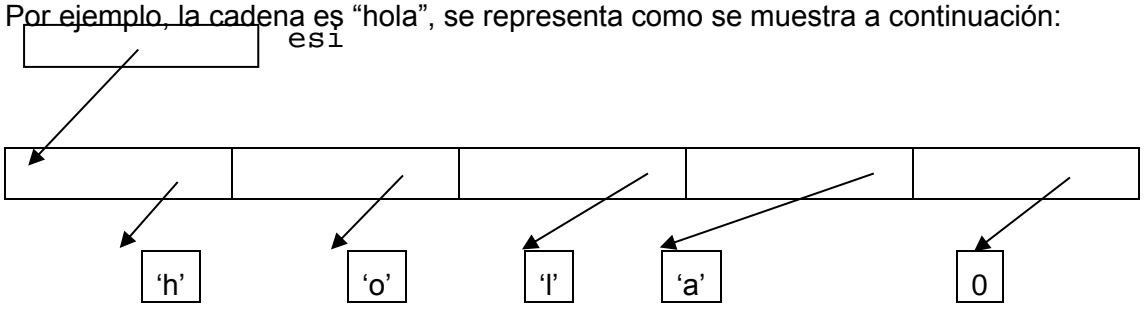
Versión 2:

```
while:
mov al, [esi]
cmp al, 0  ; ¿[esi] != '\0'? (¿se acabó la cadena?)
je finWhile    ; Es igual; terminar  (se acabó la cadena)
               ; Es diferente, ejecutar el while
  dec al
  and eax, 11111B   ; Calcular posición en el alfabeto (¿Por
qué?)
  inc cuenta[4*eax]
  inc esi
```

```
  jmp while
finWhile:
```

Se tiene la siguiente representación de cadena de caracteres: se dispone de un apuntador a un vector de apuntadores; los apuntadores en cada una de las posiciones del vector apuntan a un carácter; la cadena de caracteres está conformada por estos caracteres.

Por ejemplo, la cadena es "hola", se representa como se muestra a continuación:



En `esi` y `edi` se tienen apuntadores a dos cadenas representadas de esta manera. En eax se tiene un apuntador a un vector cuyas posiciones están sin inicializar (pero la idea es almacenar en él una cadena con la misma representación).

Escriba un programa en ensamblador que concatena las dos cadenas de caracteres apuntadas por `esi` y `edi` dejando el resultado en el vector apuntado por `eax`. Suponga que el vector apuntado por `eax` tiene el tamaño suficiente.

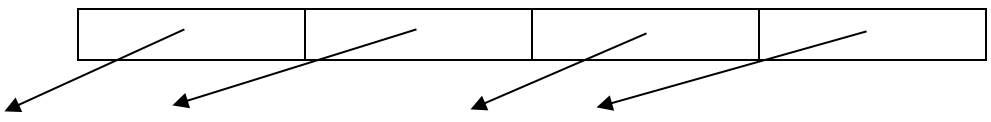
```
copiar1:
  mov ebx, [esi]            ;Traer apuntador del vector.
  cmp BYTE PTR [ebx], 0     ;Ver si apunta al carácter nulo.
  je finCopiar1
    mov [eax], ebx
    add eax, 4  ;eax y esi apuntan a vectores de apuntadores.
    add esi, 4
  jmp copiar1
finCopiar1:

copiar2:
    mov ebx, [edi]   ;Traer apuntador del vector.
    mov [eax], ebx
    add eax, 4    ;eax y edi apuntan a vectores de apuntadores.
    add edi, 4
  cmp BYTE PTR [ebx], 0     ;Ver si apunta al carácter nulo.
  jne copiar2
```

Se tiene una cadena de caracteres (`s`) compuesta de palabras separadas por un blanco y escritas solo con minúsculas; la cadena `s` puede estar vacía.

Por otro lado, se tiene un vector (`palabra`) de apuntadores, el cual se encuentra vacío inicialmente.

Escriba un programa en ensamblador que recorre la cadena de caracteres, buscando las palabras que la componen, y dejando en cada posición del vector `palabra` un apuntador a cada una de dicha palabras.

Por ejemplo, si la cadena es "una cadena de caracteres", el vector `palabra` debe quedar como se muestra a continuación:

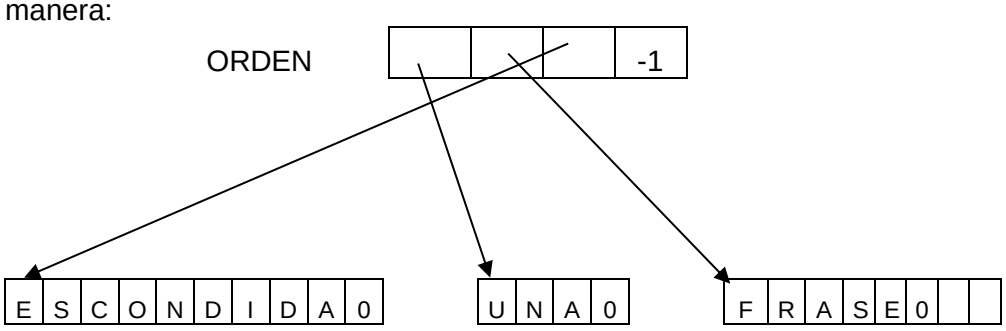


u	n	a		c	a	d	e	n	a		d	e		c	a	r	a	c	t	e	r	e	s	0

```
mov esi, offset s
mov edi, offset palabra
cmp [esi], 0
je finRecorrer
  guardarPalabra:
  mov [edi], esi
  add edi, 4
  recorrer:
  cmp [esi], 0
  je finRecorrer
  cmp [esi], ' '
  je encontroPalabra
    inc esi
  jmp recorrer
  encontroPalabra:
    inc esi
    jmp guardarPalabra
finRecorrer:
```

Se tiene una frase representada por medio de un vector de apuntadores a cadenas de caracteres en formato C. El vector, llamado `orden`, apunta a las diferentes palabras que componen la frase e indica el orden en que dichas palabras se acomodan para componer la frase. Así, la primera posición del vector apunta a la primera palabra de la frase, la segunda posición apunta a la segunda palabra de la frase y así sucesivamente. La última posición del vector tiene un valor de -1, señalando el final de la frase.

Por ejemplo, la frase “Una frase Escondida” se puede representar de la siguiente manera:



ESI apunta a una zona de memoria en la que se debe dejar la frase representada en formato C. Haga un programa en assembler que a partir del vector orden, arme en la zona apuntada por ESI la frase representada. Suponga que la zona de memoria apuntada por ESI tiene espacio suficiente para almacenar la frase completa. No olvide que al armar la frase cada palabra debe separarse de la siguiente con un espacio..

```
mov edi, 0
siguienteCadena:
mov eax, orden[4*edi]
cmp eax, -1
je finVector
  siguienteCaracter:
  mov cl, [eax]
  cmp cl, 0
```

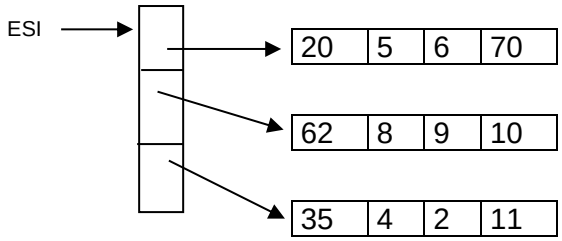
```
  je finCadena
    mov [esi], cl
    inc esi
    inc eax
    jmp siguienteCaracter
  finCadena:
  mov [esi], ' '
  inc esi
  inc edi
jmp siguienteCadena
finVector:
dec esi
mov [esi], 0
Esta solución supone que la frase tiene por lo menos una
palabra.
```

Una matriz de enteros puede representarse por medio de un vector de apuntadores, en el que, en cada posición i, se tiene un apuntador a un vector de enteros que representa los elementos de la fila i de la matriz.

Por ejemplo, la siguiente matriz:

20	5	6	70
62	8	9	10
35	4	2	11

Puede representarse así:



En este ejemplo, ESI apunta a la matriz que estamos representando. En la posición 0 del vector de apuntadores tenemos un apuntador al vector de enteros 20, 5, 6, 70 que corresponden a los elementos de la fila 0 de nuestra matriz. De igual manera se tiene para los elementos de la filas 1 y 2 de la matriz.

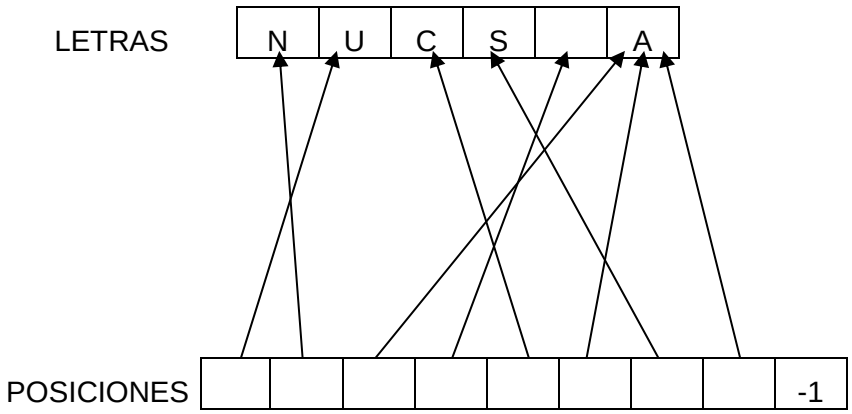
Escriba un programa en ensamblador que sume dos matrices en esta representación. ESI y EDI apuntan a las matrices que se van a sumar. El resultado debe quedar en la matriz apuntada por ESI.

`filas` y `columnas` son dos variables de tipo int que indican el número de filas y columnas que tiene la matriz.

```
  mov edx, filas
  siguienteFila:
  dec edx
  cmp edx, 0
  jl finSumar
    mov ebx,[esi+edx*4]
    mov ebp,[edi+edx*4]
    mov ecx, columnas
    siguienteColumna:
      mov eax, [ebp+4*ecx-4]
```

```
      add [ebx+4*ecx-4], eax
    loop siguienteColumna
  jmp siguienteFila
  finSumar:
```

Se tiene una frase representada por medio de dos vectores. El primer vector, llamado `letras`, contiene todas las letras que componen la frase. El segundo vector, `posiciones`, indica la posición que ocupa cada letra dentro de la frase. Para ello, la i-ésima posición del vector `posiciones` apunta a la letra que ocupa la posición i dentro de la frase final. Por ejemplo, una posible representación para la frase UNA CASA, es la siguiente:



El vector `posiciones` es la guía para reconstruir la frase original. En el ejemplo anterior, en la posición 0 del vector `posiciones` hay un apuntador a la letra U, lo que indica que esta es la primera letra de la frase. La posición 1 del vector `posiciones` apunta a la letra N, lo que indica que esta es la siguiente letra de la frase, y así sucesivamente hasta encontrar el número –1 en el vector `posiciones`, el cual señala el final de la frase.

Escriba un programa que, a partir del vector `letras` y del vector `posiciones`, arme una cadena de caracteres tipo C, sobre una variable que llamaremos `cadena`. Suponga declarada la variable `cadena` con el espacio suficiente para almacenar la frase.

Solución usando indexamiento y escalamiento:

```
mov esi, 0
mov edi, posiciones
siguienteCaracter:
cmp edi, -1
je fin
  mov al, [edi]
  mov cadena[esi], al
  inc esi
  mov edi, posiciones[4*esi]
jmp  siguienteCaracter
fin:
mov  cadena[esi], 0
```

Otra (usando apuntadores):

```
mov esi, offset cadena
mov edi, offset posiciones
mov ebx, [edi]
```

```
siguienteCaracter:
cmp ebx, -1
je fin
  mov al, [ebx]
  mov [esi], al
  inc esi
  add edi, 4
  mov ebx, [edi]
jmp  siguienteCaracter
fin:
mov  BYTE PTR [esi], 0
```

Para este punto, recuerde que una imagen se representa con un vector de pixels, donde cada pixel está compuesto de 3 bytes que representan la magnitud de sus componentes roja, verde y azul.

La superposición es una operación que se realiza entre dos imágenes, y permite hacer efectos como los "montajes" de las películas. Para esto, se parte de dos imágenes: una es el fondo (el escenario), y sobre ella se superpone la otra; la segunda, tiene la escena en sí (los actores y demás) que se filman con un fondo de un color especial (usualmente un cierto tono de azul o verde), a este color lo llamaremos "transparente".

La operación de superposición se realiza así: los pixels de la escena se van copiando en las posiciones correspondientes del fondo, pero con la siguiente condición: si el pixel es igual al color transparente, no se copia (se deja el pixel original del fondo); si es diferente, se copia sobre el pixel original. Es decir:

Por ejemplo, si se tiene los vectores (donde T representa el color transparente):

fondo	A	B	C	D	E	F

escena	G	T	T	H	I	T

El resultado será:

fondo	G	B	C	H	I	F

Escriba un programa que haga la superposición de dos imágenes que se encuentran en los vectores "`fondo`" y "`escena`". Hay una variable de tipo DWORD, `numeroPixels`, donde se tiene el tamaño de los vectores.

Se tiene un vector de palabras (WORD) de 27 posiciones. Escriba un programa que reporta en EAX el número de elementos del vector que son positivos y divisibles por 4.