EJERCICIOS

3- Se tiene un vector `v` de enteros de `n` posiciones, y se va realizar la asignación:

```
v[i] = a;
```

Escriba código en ensamblado para hacer esta asignación pero solo si `i` es un subíndice válido (esta entre `0` y `n-1`). Si no lo es, no se hace nada.

```
mov esi, i
cmp esi, 0
jl noAsignar
cmp esi, n
jge noAsignar
  mov eax, a
  mov v[4*esi], eax
noAsignar:
```

5- En el registro `al` se tiene una carácter representado en ASCII. Si el carácter es una minúscula, conviértalo a mayúsculas; si es mayúscula, conviértalo en minúscula; si no es una letra, no le haga nada. El carácter debe quedar en `al` mismo.

```
 cmp al, 'a'
 jl evaluarSegunda
 cmp al, 'z'
 jg fin
   sub al, 20H
   jmp fin
 evaluarSegunda:
 cmp [esi], 'A'
 jl fin
 cmp [esi], 'Z'
 jg fin
   add al, 20H
 fin:
```

6- Un año es bisiesto si es divisible por 4, excepto si es divisible por 100, excepto si es divisible por 400 (es decir, si es divisible por 400, sí es bisiesto). Escriba un programa en ensamblador para determinar si un año es bisiesto. `eax` debe quedar en 1, si lo es; en cero, si no.

```
mov eax, year
mov ebx, eax
mov edx, 0
mov ecx, 400
div ecx
cmp edx, 0
je siEs
mov eax, ebx
mov edx, 0
mov ecx, 4
div ecx
cmp edx, 0
jne noEs
mov eax, ebx
```

```
  mov edx, 0
  mov ecx, 100
  div ecx
  cmp edx, 0
  je noEs
  siEs:
    mov eax, 1
    jmp fin
  noEs:
    mov eax, 0
fin :
```

7- Para cada una de las condiciones que se encuentran a continuación, haga el diagrama de evaluación respectivo (como el de la figura 6.1) y escriba el programa para evaluarla. Suponga que son condiciones para un `if` y que hay etiquetas de `entonces` y `si_no`.

Expresión
(a || b)&&(c || d)
(a < b)||(c > d)
(a < b)&&(c > d)
(a < b)||(c = d)
(a < b)&&(c >= d)
(a < b)||(c != d)
!(a || b)&&(c || d)
!((a || b)&&(c || d))
(a || (b && c))&&(d || e)
(a || b)&&(c || d) || e

8- Traduzca la siguiente expresión de C a ensamblador:

```
x = ( a > b ? a : b );
```

```
mov eax, a
mov ebx, b
cmp eax, ebx
jg asignar
  mov eax, b
asignar:
mov x, eax
```

10- Traduzca el siguiente código de C a ensamblador..

```
char v[100], u[100];
int n, i;
   …
n = 1;
i = 1;
while ( (i < 100) && ( n == 1 ) ) {
  if ( (v[i] & 223) != (u[i] & 223) )
    n = 0;
  i++;
}
```

```
v byte 100 dup(?)
u byte 100 dup(?)
n dword ?
i dword ?
…
```

```
mov eax, 1
mov esi, 1
while:
  cmp esi, 100
  jge finWhile
  cmp eax, 1
  jne finWhile
    mov bl, v[esi]
    and bl, 223
    mov bh, u[esi]
    and bh, 223
      cmp bh, bl
      je finIf
        mov eax, 0
      finIf:
      inc esi
jmp while
finWhile:
mov i, esi
mov n, eax
```

11- Traduzca el siguiente código de C a ensamblador..

```
int year, mes, nDias;
   …
switch (mes) {
  case 4:
  case 6:
  case 9:
  case 11:
    nDias = 30;
    break;
  case 2:
    if ((year % 4 == 0) && (year % 100)) || (year % 400)
      nDias = 29;
    else
      nDias = 28;
    break;
  default: nDias = 31;
}
```

```
mov eax, mes
cmp eax, 4
je caso11
 cmp eax, 6
je caso11
 cmp eax, 9
je caso11
cmp eax, 11
je caso11
cmp eax, 2
je caso2
  mov nDias, 31
  jmp fin
caso11:
  mov nDias, 30
  jmp fin
```

```
caso2:
  mov eax, year
  mov ebx, eax
  mov edx, 0
  mov ecx, 400
  div ecx
  cmp edx, 0
  je entonces
  mov eax, ebx
  mov edx, 0
  mov ecx, 4
  div ecx
  cmp edx, 0
  jne si_no
  mov eax, ebx
  mov edx, 0
  mov ecx, 100
  div ecx
  cmp edx, 0
  je si_no
  entonces:
    mov nDias, 29
    jmp fin
  si_no:
    mov nDias, 28
fin :
```

12- Traduzca el siguiente código de C a ensamblador..

```
char * s;
int n;
   …
n = 0;
while ( *s != '\0' && '0' <= *s && *s <= '9' ) {
  n = 10*n + (*s - '0');
  s++;
}
```

```
s  DWORD  ?          ; s es un apuntador, así que usa 4 bytes.
n  DWORD  ?          ; s es un entero, así que usa 4 bytes.

mov n, 0
evaluarCondicion:
  mov esi, s
  cmp BYTE PTR [esi], 0    ;'\0' es 0 (carácter nulo).
  je finWhile
  cmp BYTE PTR [esi], '0'
  jb finWhile
  cmp BYTE PTR [esi], '9'
  ja finWhile
  imul eax, n, 10
  mov bl, [esi]
  sub bl, '0'
  movzx ebx, bl      ; muchas maneras de hacerlo; lo importante
                     ;es limpiar la parte alta del registro.
  add eax, ebx
  mov n, eax
  inc s
  jmp evaluarCondicion
}
```

```
finWhile:
```

Nota: no es la más eficiente, pero sí muy cercana al programa original.

13- Traduzca el siguiente código de C a ensamblador..

```
char * s;
…
while( *s!='\0' &&
       (('a'<=*s && *s<='z') || ('A'<=*s && *s <='Z')) )
  s++;
```

```
 mov esi, s
 while:
 cmp [esi], 0
 je finWhile
 cmp [esi], 'a'
 jl evaluarSegunda
 cmp [esi], 'z'
 jle ejecutar
 evaluarSegunda:
 cmp [esi], 'A'
 jl finWhile
 cmp [esi], 'Z'
 jg finWhile
   ejecutar:
   inc esi
 jmp while
 finWhile:
 mov s, esi
```

14- Traduzca el siguiente código de C a ensamblador..

```
char * s;
int i;
   …
for ( i = 0; s[i] != '\0'; i++ ) {
   if ( s[i] > 64 && s[i] <= 90 )
     s[i] = s[i] | 32;
}
```

```
s dword ?    ;s es un apuntador
i dword ?
...
mov esi, 0
for:
mov al, s[esi]
cmp al, 0  ; ¿s[i] != '\0'? (¿se acabó la cadena?)
je finFor      ; Es igual; terminar (se acabó la cadena)
               ; Es diferente; ejecutar cuerpo del for
  cmp al, 64  ; ¿s[i] > 64?
  jbe finIf    ; Es menor o igual; saltarse el if
               ; s[i] > 64; evaluar la segunda parte del &&
  cmp al, 90  ; ¿s[i] <= 90?
  ja finIf    ; Es mayo;, saltarse el if
               ; s[i] <= 90; ejecutar cuerpo del if
    or al, 32
    mov s[esi], al
  finIf:
  inc esi
```

```
  jmp for
finFor:
mov i, esi
```

15- Escriba un programa en ensamblador para calcular el producto punto de dos vectores.

```
mov esi, 0
mov eax, 0
for:
cmp esi, n
jge finFor
  mov ebx, a[4*esi]
  mul ebx, b[4*esi]
  add eax, ebx
  inc esi
  jmp for
finFor:
```

18- Manejo de cadenas. Escriba programas para:

a- Concatenar dos cadenas.

```
 mov esi, offset s1
 buscarFinS1:
 cmp [esi], 0
 je finS1
   inc esi
 jmp buscarFinS1
 finS1:
 mov edi, offset s2
 concatS2:
   mov al, [edi]
   mov [esi], al
   inc esi
   inc edi
 cmp al, 0
 jne concatS2
```

b- Comparar dos cadenas.

```
mov esi, offset s1
mov edi, offset s2
mov eax, 1
comparar:
  mov bl, [esi]
  cmp bl, [edi]
  jne diferentes
  cmp bl, 0
  je iguales
  inc esi
  inc edi
 jmp comparar
 diferentes:
 mov eax, 0
 iguales:
```

c- Mirar si una cadena es prefijo de otra; por ejemplo, "casa" es prefijo de "casados". Si es prefijo, debe poner eax en 1; y en 0, si no.

```
mov esi, offset s1
mov edi, offset s2
mov eax, 1
```

```
comparar:
  mov bl, [esi]
  cmp bl, 0
  je prefijo
  cmp bl, [edi]
  jne noEsPrefijo
  inc esi
  inc edi
 jmp comparar
 noEsPrefijo:
 mov eax, 0
 prefijo:
```

d- Mirar si una cadena se encuentra dentro de otra; por ejemplo, "caso" está al interior de "ocasos". Si está, debe poner en eax la posición a partir de la cual se encuentra (1, en el ejemplo); si no está, debe poner eax en -1.

```
mov eax, 1
mov edx, offset s1
mov ecx, offset s2
recorrer:
cmp [ecx], 0
je noEsPrefijo
  mov esi, edx
  mov edi, ecx
  inc ecx
  comparar:
    mov bl, [esi]
    cmp bl, 0
    je subStr
    cmp bl, [edi]
    jne recorrer
    inc esi
    inc edi
   jmp comparar
 noEsPrefijo:
 mov eax, 0
 subStr:
```