Estructura del archivo .ASM
Forma de declarar variables
Condiciones con AND y OR
Para indexar (o acceder) a un array
Para operar con strings
Imprimir strings
Espera de tecla
Terminar la ejecución

Estructura del archivo .ASM
El archivo .ASM es un archivo de texto que posee algunas indicaciones para el compilador. Las mismas son:

```
.MODEL      LARGE   ; tipo del modelo de memoria usado.
.386
.STACK 200h   ; bytes en el stack

.DATA ; comienzo de la zona de datos
;
declaración de todas las variables.
;
;
.CODE       ; comienzo de la zona de codigo

mov AX,@DATA ; inicializa el segmento de datos
mov DS,AX ;
;
código assembler resultante de compilar el programa fuente
;
mov ax, 4C00h ; termina la ejecución.
int 21h

END ; final del archivo.
```

Constantes con nombres:
Los compiladores de assembler poseen una operación para definir constantes numéricas con nombres similar al #define del lenguaje **C**
La operación es *equ* y se utiliza de la siguiente manera:

nombre *equ* valor

Declaración de variables:

El formato para declarar variables es **<Nombre> <Tipo> <Elementos>**

donde:

<Nombre>	nombre de la variable (no debe comenzar con digitos o con ciertos caracteres especiales)
<Tipo>	valores en distintas precisiones

(DB) definen elementos de 1 byte
(DD) definen enteros con signo (-32768 a +32767) **2 bytes**,
(DD) definen enteros con signo (-2.147.483.648 a +2.147.483.647), y valores de punto flotante con precisión simple. **4 bytes**,
(DQ) definen valores en el formato de punto flotante de doble precisión.(solo coprocesador) **8 bytes**,

<Elementos> lista separada por coma de los valores.

Ejemplo:

l programa fuente contiene las siguientes declaraciones de variables:

byte a; // variable sin valor inicial.
array b[10]; // los arrays numéricos se inicializan a 0
int c=23; // es una variable con valor inicial 23.
float _cte1= 5.3
long d;
string e,f="cadena de caracteres"; // esta cadena ocupa 20 caracteres
la declaración en assembler de estas variables es:
MAXTEXTSIZE equ 50

```
a db ?                          ; como no tiene valor inicial se agrega "?"
b dd 10 dup (0)                 ; esto genera 10 words seguidos con el valor 0.
c dd 23                         ; valor inicial 23
_cte_1 dd 5.300000
d dd ?                          ; sin valor inicial
e db MAXTEXTSIZE dup (?),'$' ; las variables string sin inicializar tienen una long de 50.
f db "cadena de caracteres", '$', 30 dup (?)
```

Nota sobre strings:
Cada lenguaje de alto nivel tiene su forma de manejar los strings. En assembler no hay una forma definida, por lo tanto se asumirá la siguiente forma:

- todos los strings tienen una capacidad máxima de 50 caracteres + carácter terminador, y su declaración incluye el espacio para esos 50 caracteres, aunque la cantidad inicial sea menor.
- cada carácter ocupa un byte
- las variables que tienen valor inicial se declaran como la cadena de valor inicial , luego el carácter separador y el resto de caracteres hasta completar los 50 sin valor (declaración de la variable **f** del ejemplo).
- en DOS el carácter de fin de cadena es '**$**'. Para el TP deben terminar de la misma manera de forma tal que puedan imprimirse.

Condiciones con AND y OR
Siempre que ocurre una condición compuesta sus 2 operandos son de tipo booleano. En el caso (a<b) AND (c<>d) a,b,c y d son de tipo escalar, pero la operación (a<b) devuelve un tipo booleano. Es recomendable guardar el resultado de cada sub-operación en una variable auxiliar, por lo que la operación anterior puede haber generado 2 variables auxiliares, una donde se guardo el resultado de a<b (que llamaremos _aux1) y otra que guardara el resultado de c<>d (_aux2 por ejemplo)

La condición compuesta queda internamente resuelta como _aux1 AND _aux2.
En assembler existe la instrucción para hacer el AND entre 2 variables

mov ax,_aux1
and ax,_aux2

y el resultado queda en AX. Luego se pregunta si el valor de AX es 0 (FALSE) o si es 1 (TRUE) para saltar a donde corresponda.

Jz Not_And (Si es cero el and es falso)

Para el OR existe la instrucción OR entre dos variables que opera de igual manera que el and.

Para indexar (o acceder) a un array:
Un array puede entenderse como una sucesión de variables del mismo tipo. La declaración:

array b[10]; // *asumimos que los arrays son solamente de tipos enteros.*

termina definiendo en assembler 10 espacios contiguos de 2 bytes (1 word) cada uno.

Desde este punto de vista un cada elemento del array puede utilizarse en los mismo lugares de la gramática donde se utiliza un ID. La asignación tiene del lado izquierdo un ID puede tener también un array indexado (elemento del array). Las expresiones con ID del lado derecho pueden aceptar también arrays indexados.
Para acceder a un elemento particular dentro del array (ya sea para asignarle un valor o para extraerlo) se debe hacer lo siguiente:

```
b dw 10 dup (0)                 ; declaración del array
```

el nombre **b** dentro del ámbito de este programa en assembler indicara la posición de memoria en donde comienza el array, o sea, la primer posición del array. Para acceder a una posición particular hay que sumarle a la posición de inicio la cantidad de espacios de memoria hasta la dirección en que se halla el elemento deseado.

ejemplo 1: **var1=b[3];**

```
        mov ax, 3               ; valor del índice

        cmp ax,#LIMITE          ; aquí debería ir la verificación de limites
        jle _LIMOK              ; (no se pide este año)
        call ERROR_LIMITE
        jmp TERMINAR

LIMOK:
        mov bx,2                ; cada elemento ocupa 2 bytes
        mul bx                  ; cantidad de espacios de memoria desde el ppio del array
        mov bx,ax               ; el resultado queda en AX, pero se necesita en BX para
                                ; indexar. Se mueve de AX a BX.
        add bx, OFFSET b        ; suma la dirección de comienzo del array. BX ahora
                                ; apunta al elemento en cuestión.
        mov ax, word ptr [bx]   ; copia el word apuntado por BX al registro AX.
        mov var1,ax             ; se guarda en var1.
```

ejemplo 2: **b[4]=3+2;**

```
                                ; la expresión 3+2 esta resuelta mas arriba y suponemos
                                ; que su resultado se guardo en la variable _aux7

        mov ax, 4 ; valor del indice
        cmp ax,#LIMITE          ; aquí debería ir la verificación de limites
        jle _LIMOK              ; (no se pide este año)
        call ERROR_LIMITE
        jmp TERMINAR

LIMOK:
        mov bx,2                ; cada elemento ocupa 2 bytes
        mul bx                  ; cantidad de espacios de memoria desde el ppio del array
        mov bx,ax               ; el resultado queda en AX, pero se necesita en BX para
                                ; indexar. Se mueve de AX a BX.
        add bx, OFFSET b        ; suma la dirección de comienzo del array. BX ahora
                                ; apunta al elemento en cuestión.
        mov ax,_aux7            ; copia el dato a guardar en AX
        mov word ptr[BX],ax     ; copia AX a la posición de memoria apuntada por BX.
```

ejemplo 3: **b[17 * (var1+7)]=0;**

```
                                ; la expresión 17*(var1+7) esta resuelta mas arriba
                                ; y suponemos que se guardo en la variable _aux13
        mov ax,_aux13           ; valor del índice
        cmp ax,#LIMITE          ; aquí debería ir la verificación de limites (no se pide
        jle _LIMOK              ; este año)
        call ERROR_LIMITE
        jmp TERMINAR

LIMOK:
        mov bx,2                ; cada elemento ocupa 2 bytes
        mul bx                  ; cantidad de espacios de memoria desde el ppio. del array
        mov bx,ax               ; el resultado queda en AX, pero se necesita en BX para
                                ; indexar. Se mueve de AX a BX.
        add bx, OFFSET b        ; suma la dirección de comienzo del array. BX ahora
                                ; apunta al elemento en cuestión.
        mov ax,0                ; copia el dato a guardar en AX
        mov word ptr[BX],ax     ; copia AX a la posición de memoria apuntada pos BX.
```

Para operar con strings

Como se vio en el punto ***Forma de declarar variables*** un string es una sucesión de caracteres en memoria, donde cada carácter ocupa 1 byte. El microprocesador posee instrucciones especificas que facilitan el manejo de strings; y para el TP se usa solamente la instrucción que copia datos de un lugar a otro para concatenar strings o para realizar las asignaciones.

La forma en que trabaja el micro es la siguiente:

apuntar DS:SI al comienzo del string source (origen de los datos)

apuntar ES:DI al punto donde se quiere comenzar a copiar

cargar CX con la cantidad de bytes a copiar.

Veamos un ejemplo:

```
string cad1=”primer cadena”;
string cad2=”segunda cadena”;
string cad3;                          // cadena sin valor inicial

cad3=cad1+cad2;                       // operación de concatenación y asignación
```

Habíamos dicho que todas las cadenas (tengan valor inicial o no) tendrán un tamaño de 50 caracteres mas el terminador $. La declaración en assembler de este fragmento de código es el que sigue:

```
.DATA
MAXTEXTSIZE equ 50
cad1 db “primer cadena”,’$’, 37 dup (?)
cad2 db “segunda cadena”,’$’, 36 dup (?)
cad3 db MAXTEXTSIZE dup (?), ‘$’
_aux1 db MAXTEXTSIZE dup (?), ‘$’
```

En los dos primeros casos la declaración incluye el valor inicial, el terminador, y los caracteres restantes para formar 50 bytes de caracteres. El tercer y cuarto caso, al no tener valor inicial, se declara simplemente como los 50 caracteres + terminador.

La línea cad3=cad1+cad2; contiene 2 operaciones con strings: concatenar y asignar. Veamos la primera:

Para concatenar utilizaremos una variable auxiliar en donde almacenar el resultado. Dicha variable será del tipo string y contendrá también 50 caracteres + terminador. La concatenación se realiza como 2 copias sucesivas

copiar la 1er cadena al inicio de la cadena auxiliar
copiar la 2da cadena a la cadena auxiliar, desde el punto donde terminó la primera.

```
mov ax,@DATA                    ; Esto prepara los registros
mov d                           ; y flags para copiar.
mov es,ax                       ;
mov si,OFFSET cad1              ; apunta el origen a la primer cadena
mov di,OFFSET aux1              ; apunta el destino a la variable auxiliar
call COPIAR                     ; realiza la copia.

mov si,OFFSET cad2              ; apunta el origen a la segunda cadena
mov di,OFFSET aux1              ; apunta el destino a la variable auxiliar
call CONCAT                     ; concatena la segunda cadena al auxiliar
```

Por ultimo está la asignación de la variable auxiliar a cad3.

```
mov si,OFFSET aux1              ; apunta el origen al auxiliar
mov di,OFFSET cad3              ;apunta el destino a la cadena 3
call COPIAR                     ; copia los strings
```

RUTINAS UTILIZADAS:

```
;*********************************************************
```

```
; devuelve en BX la cantidad de caracteres que tiene un string
; DS:SI apunta al string.
;
STRLEN PROC
        mov bx,0

STRL01:
        cmp BYTE PTR [SI+BX],'$'
        je STREND
        inc BX
        jmp STRL01
STREND:
        ret

STRLEN ENDP


;*****************************************************************8
; copia DS:SI a ES:DI; busca la cantidad de caracteres
;
COPIAR PROC
        call STRLEN                             ; busco la cantidad de caracteres
        cmp bx,MAXTEXTSIZE
        jle COPIARSIZEOK

        mov bx,MAXTEXTSIZE

COPIARSIZEOK:
        mov cx,bx                               ; la copia se hace de 'CX' caracteres
        cld                                     ; cld es para que la copia se realice
                                                ; hacia adelante"
        rep movsb                               ; copia la cadea
        mov al,'$'                              ; carácter terminador
        mov BYTE PTR [DI],al                    ; el registro DI quedo apuntando al
                                                ; final
        ret

COPIAR ENDP


;*****************************************************
; concatena DS:SI al final de ES:DI.
;
; busco el size del primer string
; sumo el size del segundo string
; si la suma excede MAXTEXTSIZE, copio solamente MAXTEXTSIZE caracteres
; si la suma NO excede MAXTEXTSIZE, copio el total de caracteres que tiene el segundo string
;
CONCAT PROC
        push ds
        push si
        call STRLEN                             ; busco la cantidad de caracteres del
                                                ;2do string
        mov dx,bx                               ; guardo en DX la cantidad de caracteres
                                                ;en el origen.
        mov si,di
        push es
        pop ds
        call STRLEN                             ; tama#o del 1er string
        add di,bx                               ; DI ya queda apuntando al final del
                                                ;primer string
        add bx,dx                               ; tama#o total
        cmp bx,MAXTEXTSIZE                      ; excede el tama#o maximo?
        jg CONCATSIZEMAL

CONCATSIZEOK:                                   ; La suma no excede el maximo, copio
                                                ; todos
        mov cx,dx                               ; los caracteres del segundo string.
```

```
        jmp CONCATSIGO

CONCATSIZEMAL:                          ; La suma de caracteres de los 2 strings
                                        ; exceden el maximo
        sub bx,MAXTEXTSIZE
        sub dx,bx
        mov cx,dx                       ; copio lo maximo permitido el resto
                                        ;se pierde.
CONCATSIGO:

        push ds
        pop es
        pop si
        pop ds
        cld                             ; cld es para que la copia se realice
                                        ;hacia adelante
        rep movsb                       ; copia la cadea
        mov al,'$'                      ; carácter terminador
        mov BYTE PTR [DI],al            ; el registro DI quedo apuntando al
                                        ;final
        ret

CONCAT ENDP
```

Imprimir strings:

Como se dijo con anterioridad los strings se definen en assembler de la siguiente manera:

cadena db 0Dh, 0Ah, “esto es un ejemplo de string para imprimir ”,$

para imprimir se utiliza el servicio de impresión de DOS

```
mov DX, OFFSET cadena           ;DX apunta al ppio de la cadena
mov ah, 9                       ; AH=9 es el servicio de impresion
int 21h                         ; imprime
```

Los valores 0Dh,0Ah generan un retorno de carro y una nueva línea, de manera que el string se imprima en la línea de abajo.

Espera de tecla:

La siguiente rutina se queda esperando que se presione una tecla. El resultado (codigo ASCII de la tecla presionada) queda en el registro AL.

```
;DEFINICION DE VARIABLES

        msgPRESIONE db 0DH,0AH,"Presione una tecla para continuar...",'$'
        _NEWLINE db 0Dh,0Ah,'$'

;FIN DEFINICION DE VARIABLES

;CODIGO

        mov DX, OFFSET _NEWLINE         ;agrega newline
        mov ah,09
        int 21h

        mov dx,OFFSET msgPRESIONE       ;imprimir mensaje de espera
        mov ah,09
```

```
        int 21h

        mov ah, 1                   ; pausa, espera que oprima una tecla
        int 21h                     ; AH=1 es el servicio de lectura
```

Terminar la ejecución:

```
TERMINAR:
        mov ax, 4C00h               ; Indica que debe terminar la ejecución
        int 21h                     ; Int de servicios del Sistema Operativo (DOS)
```