

Manual de ASM

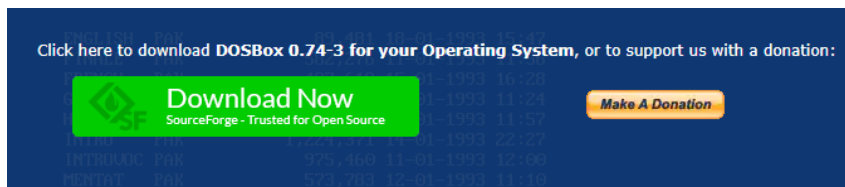
Contenidos

1. Instalación.....	2
2. Recomendaciones generales.....	4
3. Segmentos de un código ASM.....	6
1) Segmento de datos	6
2) Segmento de pila	7
3) Segmento de código	8
4. Código.....	10
1) Etiquetas	10
2) Instrucciones.....	10
3) Registros y variables	16
4) Procedimientos.....	16
5. Ejecución.....	18
6. Debugger	19
7. Ejemplos de programas.....	22

1. Instalación

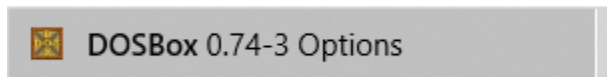
Para ejecutar un programa ASM, es decir en ensamblador, en el curso se utiliza el formato Tasm y una máquina virtual llamada DOSBox. Para instalarla siga los siguientes pasos:

- 1) Ingrese al siguiente enlace: <https://www.dosbox.com/download.php?main=1>
- 2) Descargue la máquina virtual en el botón verde como se muestra en la imagen y siga los pasos que le indiquen.



Puede omitir los pasos 3 y 4 pero tendrá que indicar la dirección de donde está cada archivo que desee ejecutar.

- 3) Busque el archivo de texto que se descargó llamado Options. Se ve algo así:



- 4) Abra el archivo y diríjase al final y escriba lo siguiente:

```
mount k: c:\
mount T: c:\
t:
cd
```

- Después de cd debe escribir el nombre de la carpeta en donde estarán los programas.
- En mount T escriba la carpeta que contiene la de cd.
- En mount K escriba la carpeta que contiene la de mount T.

- 5) Ahora puede comenzar a escribir su programa, guárdelo en la carpeta indicada en cd. Los programas pueden ser escritos en la aplicación de archivos de texto, pero deben ser guardados como .ASM.

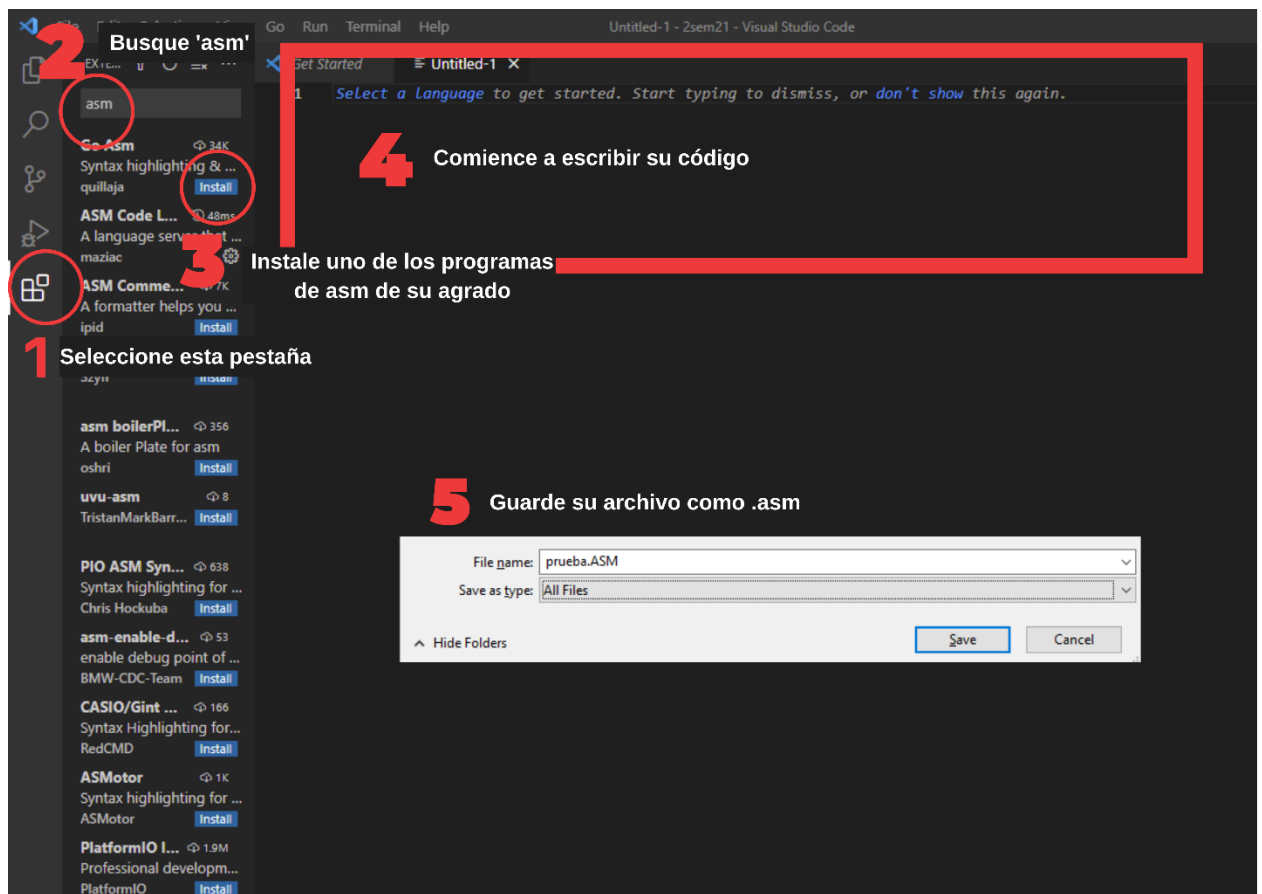
2. Recomendaciones generales

- El programa .ASM se puede escribir en un archivo de texto como fue mencionado anteriormente, sin embargo, existen otras aplicaciones que permiten hacerlo. Se recomienda usar alguna aplicación como Visual Studio Code ya que utiliza colores, numera las líneas, permite ocultar código y solo ver los nombres de los procedimientos, para navegar por él más fácilmente, entre otras funcionalidades.

Esta es la aplicación mencionada:



Una vez se encuentre dentro de la aplicación, siga los pasos de la siguiente imagen:



- Si se encuentra atascado en la resolución de algún problema o en la escritura de un código y no encuentra el error haga uso del Debugger (esto será explicado a profundidad en el apartado #6). De igual forma, es bueno tomar un descanso para despejar la mente y después poder volver con nuevas ideas.

3. Segmentos de un código ASM

El código ASM cuenta con 3 segmentos distintos que serán explicados a continuación:

1) Segmento de datos

En este segmento se declaran los datos, es decir las variables.

Se debe escribir:

```
'nombre' segment
... aquí las variables
'nombre' ends
```

Un ejemplo es el siguiente:

```
datos segment

    elString db "hola$"
    elString2 db "adios",0
    tam dw ?
    numero dw 3
    numBin db 0101b
    numHex dw 78h
    auxiliar dw 256 dup (?)

170 references
datos ends
```

Para las variables se encuentran distintas opciones al declararlas:

- Primero se escribe el **nombre** deseado (los que están en celeste en la imagen).
- En segundo lugar, viene se escribe si es **db** (define byte, es decir 8 bits o 1 byte) o **dw** (define word, es decir, 16 bits o 2 bytes). Cuando se trabaja con lo que se conoce con strings o caracteres, se debe usar db.
- Por último, se debe colocar lo que se desea en la variable, puede ser texto que iría en comillas.

Si es **ASCII** se debe escribir con un \$ al final para indicar que es el final cuando se quiere desplegar en pantalla. (Ejemplo: variable elString en la imagen).

Si es **ASCIIZ** es porque se escribe un ,0 después de las comillas. (Ejemplo: variable elString2 en la imagen).

Se puede no especificar y poner un signo de pregunta (?) ya que será modificada la variable en el programa por un valor desconocido. (Ejemplo: variable tam en la imagen).

Se pueden declarar también **números en decimal** (ejemplo: variable numero en la imagen), en **binario** con una b al final (ejemplo: variable numBin en la imagen) o en **hexadecimal** con una h al final (ejemplo: variable numHex en la imagen).

También está la opción de usar **dup** que es para repetir lo declarado la cantidad de veces que dice entre el paréntesis (esto último también puede estar sin especificar con el signo de pregunta). (Ejemplo: variable auxiliar en la imagen).

- Por último, está la opción de no utilizar un nombre para la variable y acceder a ella por medio de su dirección en memoria, por ejemplo, para utilizar matrices en donde se suele nombrar solo al primer valor. Ejemplo:

```
tablero dw 0,0,0,0,0,0,0,0,0
         dw 0,0,0,0,0,0,0,0,0
         dw 0,0,0,0,0,0,0,0,0
         dw 0,0,0,0,0,0,0,0,0
         dw 0,0,0,0,0,0,0,0,0
         dw 0,0,0,0,0,0,0,0,0
         dw 0,0,0,0,0,0,0,0,0
         dw 0,0,0,0,0,0,0,0,0
```

2) Segmento de pila

En este segmento se guarda espacio para la pila. Sólo se puede acceder a la pila por el top, es decir por último elemento enviado a ella. Para enviar a la pila un elemento, se hace push (lo coloca arriba de la pila). Para sacar un elemento se hace pop (da el que está arriba de la pila).

Para declarar este segmento se escribe:

```
‘nombre’ segment stack ‘stack’
... aquí la declaración del tamaño
```

‘nombre’ ends

Ejemplo:

```
pila segment stack 'stack'
    dw 256 dup (?)
171 references
pila ends
```

3) Segmento de código

En este segmento se escribe propiamente el código. Al inicio de este segmento se deben especificar los nombres dados a los diferentes segmentos.

Se debe escribir:

‘nombre’ segment

Asume cs: ‘nombre del segmento de código’, ds: ‘nombre del segmento de datos’,
ss: ‘nombre del segmento de pila’

‘nombre del programa principal’ : (no debe estar al inicio necesariamente)

... aquí el código

‘nombre’ ends

end ‘nombre programa principal’

Un ejemplo de un código pequeño y sencillo es el siguiente:

```
120 references
codigo segment
    assume cs:codigo , ds:datos , ss:pila

50 references
main: mov ax, pila
      mov ss, ax

      mov ax, datos
      mov ds, ax

      mov ah, 09h
      lea dx, rotulo

      int 21h

      mov ax, 4C00h

      int 21h

130 references
codigo ends

39 references
end main
```


Este segmento puede estar dividido en procedimientos diferentes, que es lo recomendado para un código grande, esto evita escribir muchas veces las mismas líneas, sino que es posible llamar al procedimiento en diferentes ocasiones. Sin embargo, siempre tiene que haber una parte en el código que haga las llamadas y demás, esto es lo que se ejecutará a la hora de cargar el programa, normalmente ese le conoce como 'main' o 'inicio'. Más especificaciones y recomendaciones para la escritura del código serán detalladas en los siguientes apartados.

4. Código

1) Etiquetas

Son los nombres que se le dan a partes del código y van seguidos de dos puntos (:).

Ejemplo:

```
desplegar:
    mov ah, 09h
    lea dx, rotulo
```

2) Instrucciones

En la presente sección, se detallarán diferentes instrucciones que puede utilizar y serán agrupadas por tipo, con su debida explicación y forma de uso.

Generales

- **MOV** destino, origen

Instrucción que mueve el valor del origen al destino.

Ejemplos:

```
mov ax, rotulo
4206 references
mov bl, dl
4204 references
mov num, 9
```

- **RET**

Instrucción para retornar después de que una rutina fue llamada.

- **CALL**

Instrucción para llamar a una rutina.

Ejemplo: call desplegar

- **PUSH**

Instrucción para meter un registro a la pila. (Deben ser de 16 bits)

Ejemplo: push ax

- **POP**
Instrucción para sacar un registro de la pila. (Deben ser de 16 bits)
Ejemplo: pop cx
- **LEA** destino, origen
Instrucción para obtener la dirección de memoria de un origen y colocarla en el destino.
Ejemplo: lea ax, rotulo
- **CMP** destino, fuente
Instrucción para comparar el destino con la fuente, restándosela.
Ejemplo: cmp ax, 10

Aritméticos

- **ADD** destino, n
Suma ambos y deja el resultado en destino.
Ejemplo: add ax, bx
- **SUB** destino, n
Le resta a destino n.
Ejemplo: sub ax, 10
- **INC** n
Suma 1.
Ejemplo: inc cx
- **DEC** n
Resta 1.
Ejemplo: dec cx
- **MUL** n
Hace una multiplicación. Hay dos opciones dependiendo del tamaño de n:

- a) Si es de 8 bits: multiplica n por lo que hay en AH y deja el resultado en AX.
- b) Si es de 16 bits: multiplica n por lo que hay en AX y deja el resultado en DX:AX.

Ejemplo:

Mov ah, 8

Mov ch, 9

Mult ch

- **DIV** n

Hace una división. Hay dos opciones dependiendo del tamaño de n:

- a) Si es de 8 bits: divide lo que hay en AX por n y deja el cociente en AL y el residuo en AH.
- b) Si es de 16 bits: multiplica lo que hay en DX:AX y deja el cociente en AX y el residuo en DX.

Ejemplo:

Mov ax, 32

Mov ch, 2

Div ch

Lógicos

- **AND** destino, origen

Realiza la operación 'y' , deja el resultado en destino.

Ejemplo: and ax, bx

- **NOT** destino

Calcula la negación de cada bit.

Ejemplo: not ax

- **NEG** destino
Calcula el complemento a 2 en destino, y lo almacena allí.
Ejemplo: neg ax
- **OR** destino, origen
Realiza la operación 'o' y deja el resultado en destino.
Ejemplo: or ax, bx
- **XOR** destino, origen
Realiza la operación OR exclusivo y deja el resultado en destino. Suele usarse con el destino igual al origen, para dejarlo en 0.
Ejemplo: xor ax, ax

Saltos

Todos deben ir seguidos de una etiqueta.

- **JMP**
Salta siempre

Los siguientes se utilizan después de una comparación y deben ir seguidos de una etiqueta:

- **JA (JNBE)**
Salta si es mayor (si no es menor o igual).
- **JAE (JNB)**
Salta si es mayor o igual (si no es menor).
- **JB (JNAE)**
Salta si es menor (si no es mayor o igual).
- **JBE (JNA)**
Salta si es menor o igual (si no es mayor).
- **JE (JZ)**
Salta si es igual.

- **JNE (JNZ)**

Salta si no es igual.

- **JG (JNLE)**

Salta si es mayor (si no es menor o igual). Toma en cuenta el signo.

- **JGE (JNL)**

Salta si es mayor o igual (si no es menor). Toma en cuenta el signo.

- **JL (JNGE)**

Salta si es menor (si no es mayor o igual). Toma en cuenta el signo.

- **JLE (JNG)**

Salta si es menor o igual (si no es mayor). Toma en cuenta el signo.

- **JC**

Salta si hay acarreo, es decir si la bandera Carry está en 1.

- **JNC**

Salta si no hay acarreo.

- **LOOP**

Salta siempre que cx no sea 0 y lo decrementa en 1.

Un ejemplo de código que utilice saltos es el de la imagen, que compara ax con 10 y despliega si es mayor, igual o menor (para ello las variables que contienen los rótulos deben estar declaradas previamente).

```
cmp ax, 10
ja esMayor
jb esMenor
lea dx, rotEsIgual

esMayor:
lea dx, rotEsMayor
jmp desplegar

esMenor:
lea dx, rotEsMenor
jmp desplegar

desplegar:
mov ah, 09h
int 21h

mov ax, 4C00h
int 21h
```

Strings y caracteres

Desplegar a la salida estándar:

Para el manejo de strings, la instrucción más conocida es la de desplegar en pantalla. Para esto se mueve la dirección del string a desplegar al registro dx y se ejecutan las siguientes instrucciones:

```
lea dx, rotulo
mov ah, 09h
int 21h
```

Si es solo un carácter, se mueve el carácter al registro dl y de allí se ejecuta lo siguiente:

```
mov dl, '3'
mov ah, 02h
int 21h
```

Recibir por la entrada estándar:

Para obtener lo que fue escrito en la entrada estándar se utiliza a SI, que es un registro que sirve como puntero para acceder a la memoria. En CL queda el byte al que apunta SI, apunta al primero de la entrada por lo que deber ir incrementándolo si desea acceder a los siguientes.

```
mov si, 80h
mov cl, byte ptr es:[si]
xor ch, ch
```

Terminar programa

Para finalizar el programa y cerrarlo de la manera correcta se utilizan las siguientes instrucciones:

```
mov ax, 4C00h
```

```
int 21h
```

Es de suma importancia utilizarlas ya que de lo contrario el programa no se cierra adecuadamente y quedaría abierto que lleven a diversos errores.

3) Registros y variables

Los registros se pueden utilizar a lo largo del programa y en Tasm hay tanto de 16 bits (1 word) como de 8 bits (1 byte).

Los principales de 16 bits son AX, BX, CX y DX. Estos tienen una parte alta (high) y una baja (low), de 8 bits cada uno. Por lo que el AX se divide por ejemplo en AH y en AL.

También existen otros registros que son utilizados como punteros normalmente. Estos son el SI y el DI, ambos de 16 bits cada uno.

Un ejemplo en donde se podrían usar como punteros es si se tiene la siguiente variable:

```
Rotulo db "hola"
```

Y se quiere mover el tercer caracter al registro BL. Esto se podría hacer así.

```
MOV SI, 2
```

```
MOV DL, rotulo[SI]
```

4) Procedimientos

Los procedimientos son subrutinas que tienen como fin no repetir código, sino que puedan ser llamadas las veces que sean necesarias.

Hay dos tipos, procedimientos far (lejanos) o near (ceranos). Su principal diferencia es que los near se usan dentro del mismo segmento y los far sí pueden ser llamados desde otros segmentos.

Se declaran de la siguiente manera:

Desplegar proc near

... código

ret

Desplegar endP

Desplegar proc far

... código

ret

Desplegar endP



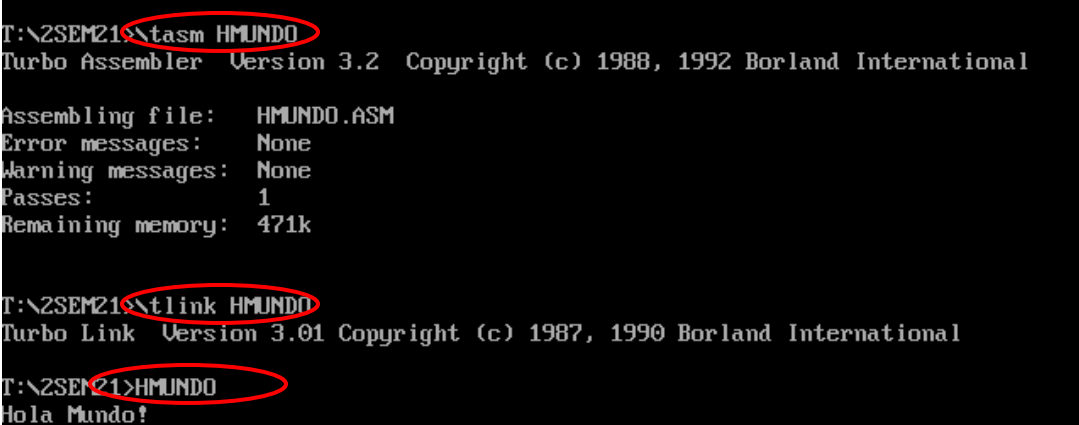
Croacia

5. Ejecución

Para ejecutar su programa abra la aplicación del DOSBox y siga los siguientes pasos:

- Escriba: `\tasm nombreDelPrograma` y dé enter.
Si hay errores de compilación, aparecerán en este momento y deberá corregirlos para que el programa pueda correr.
- Escriba: `\tlink nombreDelPrograma` y dé enter.
- Escriba: `nombreDelPrograma` y dé enter.
Su programa debería de correr en este momento.

Ejemplo:



```
T:\2SEM21>\tasm HMUNDO
Turbo Assembler Version 3.2 Copyright (c) 1988, 1992 Borland International

Assembling file:   HMUNDO.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  471k

T:\2SEM21>\tlink HMUNDO
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International

T:\2SEM21>HMUNDO
Hola Mundo!
```

6. Debugger

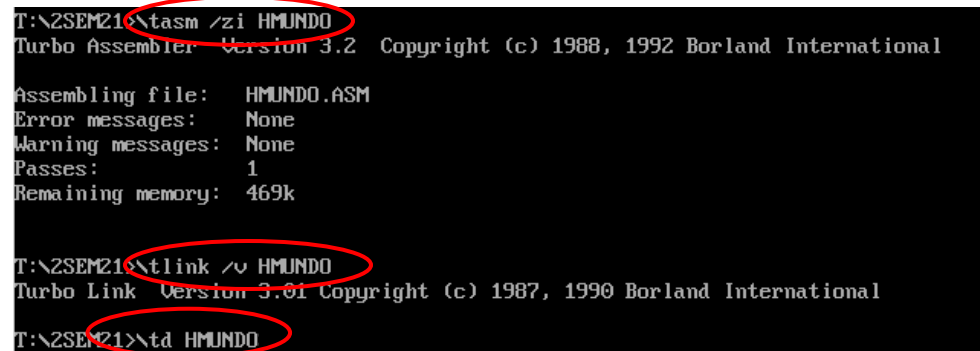
El Debugger es una excelente herramienta para ir siguiendo la ejecución de su programa. Esto es de gran ayuda especialmente cuando el programa está presentando alguna falla (no de compilación) y no se sabe en dónde o por qué.

Para ejecutarlo siga los siguientes pasos tras abrir la aplicación del DOSBox:

- Escriba: `\tasm /zi nombreDelPrograma` y dé enter.
- Escriba: `\tlink /v nombreDelPrograma` y dé enter.
- Escriba: `\td nombreDelPrograma` y dé enter.

El Debugger debería abrirse en este momento.

Ejemplo:



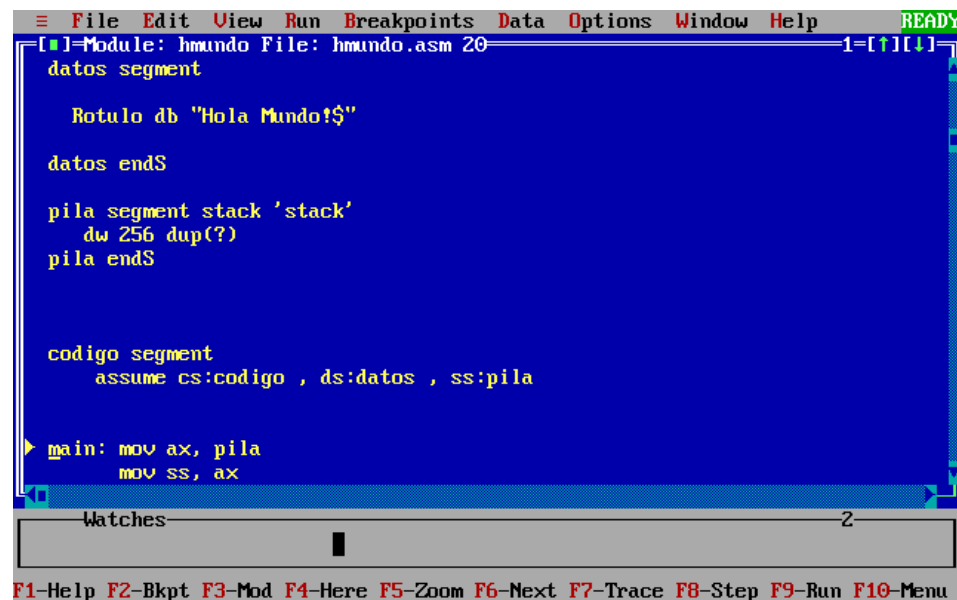
```
T:\2SEM21>\tasm /zi HMUNDO
Turbo Assembler Version 3.2 Copyright (c) 1988, 1992 Borland International

Assembling file:  HMUNDO.ASM
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 469k

T:\2SEM21>\tlink /v HMUNDO
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International

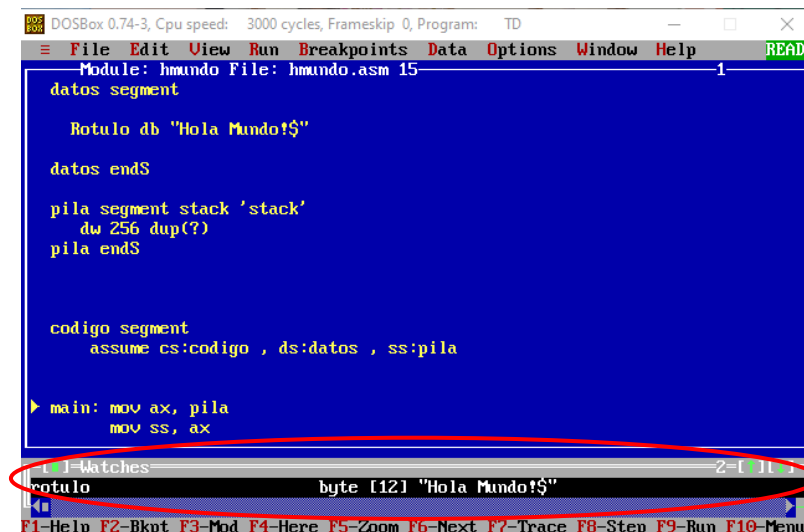
T:\2SEM21>\td HMUNDO
```

La pestaña del Debugger se ve así:

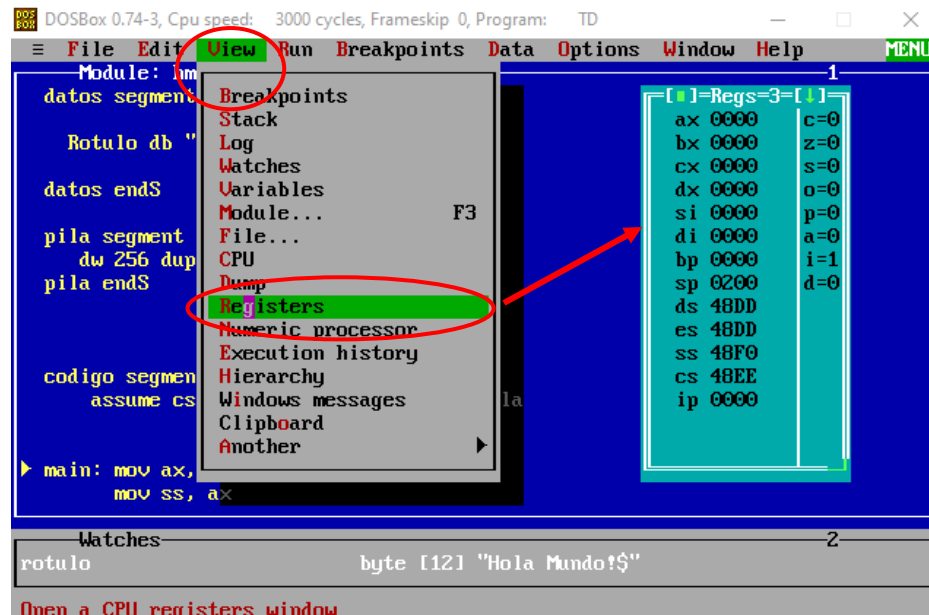


Estas son algunas de las principales opciones que se pueden utilizar en el Debugger y que son de gran utilidad:

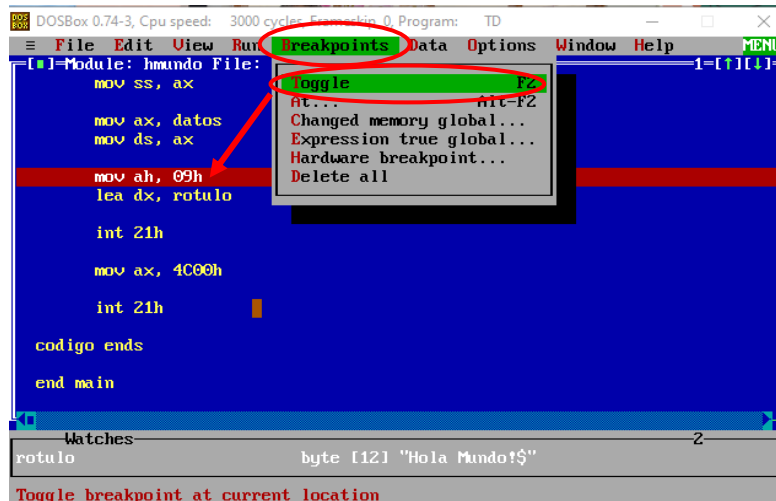
- ✓ Escribir el nombre de las variables que desea ver en el recuadro inferior llamado 'Watches' como se puede ver en la imagen. Esto con el fin de ver lo que contienen y cómo van cambiando al correr el programa.



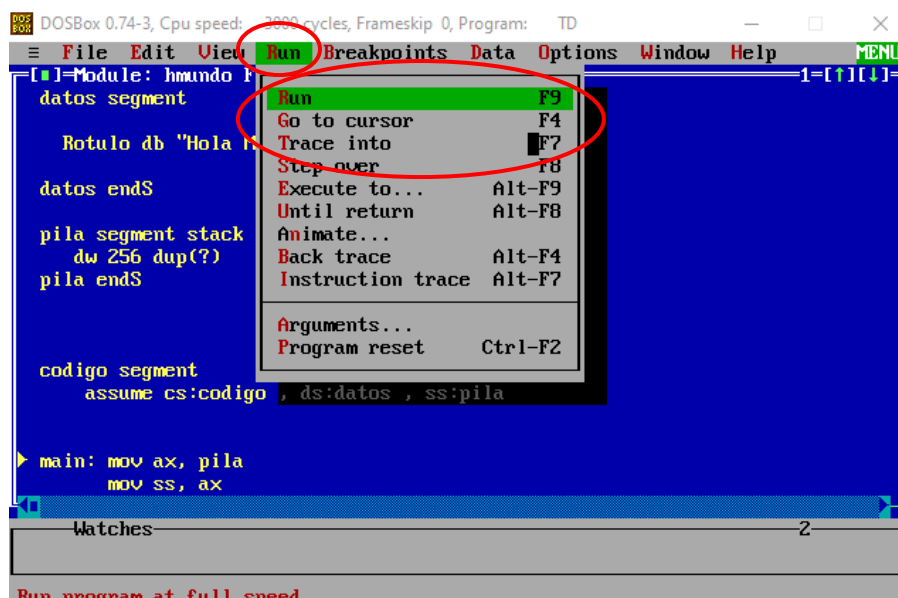
- ✓ En la pestaña 'View' seleccionar 'Registers' para ver los registros y banderas y lo que hay en ellos, como se puede ver en la imagen.



- ✓ Colocar el cursor en donde quiere ir y en la pestaña 'Breakpoints' seleccionar la opción 'Toggle'. Esto hará que cuando se corra el programa, avance hasta esa posición, señalada en rojo.



- ✓ Hay diferentes formas de correr el programa. Tres de las principales son las siguientes, y se encuentran en la pestaña 'Run' (o presionando las teclas correspondientes que aparecen al lado de sus nombres):
 - 'Run': corre el programa completo.
 - 'Go to cursor': corre el programa hasta donde se encuentre el cursor.
 - 'Trace into': corre la siguiente instrucción.



7. Ejemplos de programas

a) Machote para un programa .ASM:

```
datos segment

    ; inserte aqui las variables

datos endS

pila segment stack 'stack'

    dw 256 dup(?)

pila endS

codigo segment
    assume cs:codigo , ds:datos , ss:pila

main: mov ax, pila
      mov ss, ax

      mov ax, datos
      mov ds, ax

      ; inserte aqui el codigo

      mov ax, 4C00h          ; protocolo de salida del programa
      int 21h

codigo ends

end main
```

- b) Programa que recibe en la entrada estándar una serie de caracteres y cambia las vocales por determinados símbolos:
(Puede copiarlo y correrlo para hacer la prueba)

```
datos segment

    rotulo db 128 dup ( ? )

datos ends

pila segment stack 'stack'

    dw 256 dup(?)

pila ends

codigo segment
    assume cs:codigo , ds:datos , ss:pila

verVocales proc near
; rutina que recibe un caracter en al y si es vocal minúscula lo cambia por un
; símbolo
    cmp al, 'a'           ; compara lo que hay en al con 'a'
    jne v1                ; salta a v1 si no es igual
    mov al, '@'           ; mueve '@' a al
    jmp finVocales        ; salta a finVocales

v1:
    cmp al, 'i'           ; compara lo que hay en al con 'i'
    jne v2                ; salta a v2 si no es igual
    mov al, '|'           ; mueve '|' a al
    jmp finVocales        ; salta a finVocales

v2:
    cmp al, 'e'           ; compara lo que hay en al con 'e'
    jne v3                ; salta a v3 si no es igual
    mov al, '*'           ; mueve '*' a al
    jmp finVocales        ; salta a finVocales

v3:
    cmp al, 'o'           ; compara lo que hay en al con 'o'
    jne v4                ; salta a v4 si no es igual
    mov al, '.'           ; mueve '.' a al
```

```

    jmp finVocales          ; salta a finVocales

v4:
    cmp al, 'u'             ; compara lo que hay en al con 'u'
    jne finVocales         ; salta a v1finVocales si no es igual
    mov al, '^'             ; mueve '^' a al
    jmp finVocales         ; salta a finVocales

finVocales:
    ret                    ; retorna a donde fue llamado

verVocales endP

main:
    mov ax, ds
    mov es, ax

    mov ax, pila
    mov ss, ax

    mov ax, datos
    mov ds, ax

    mov si, 80h             ; el SI apunta a la entrada estándar
    mov cl, byte ptr es:[si] ; el cl tiene el tamaño de la entrada
    xor ch, ch

    dec cx                 ; le resta 1 para descontar el espacio inicial
;entre el nombre del programa y la entrada
    inc si                 ; le suma 1 para saltarse dicho espacio

    xor di, di

ciclo:
    inc si                 ; para pasar al siguiente caracter
    mov al, byte ptr es:[si] ; mueve a al el caracter al que apunta si

    call verVocales        ; llama al procedimiento verVocales

    mov byte ptr rotulo[di], al ; mueve a la posición di de la variable
;rotulo, lo que hay en al
    inc di                 ; le suma 1 a di para pasar a la siguiente
;posición

```



```

    loop ciclo                ; repite el ciclo y le resta 1 a cx hasta que
;cx sea 0.

    mov byte ptr rotulo[di], "$"    ; le mueve un $ para indicar su final

    ; el 13 y el 10 juntos imprimen un cambio de línea
    mov dl, 13
    mov ah, 02h
    int 21h
    mov dl, 10
    int 21h

    ; llama a la instrucción que despliega lo que apunta dx
    mov ah, 09h
    lea dx, rotulo
    int 21h

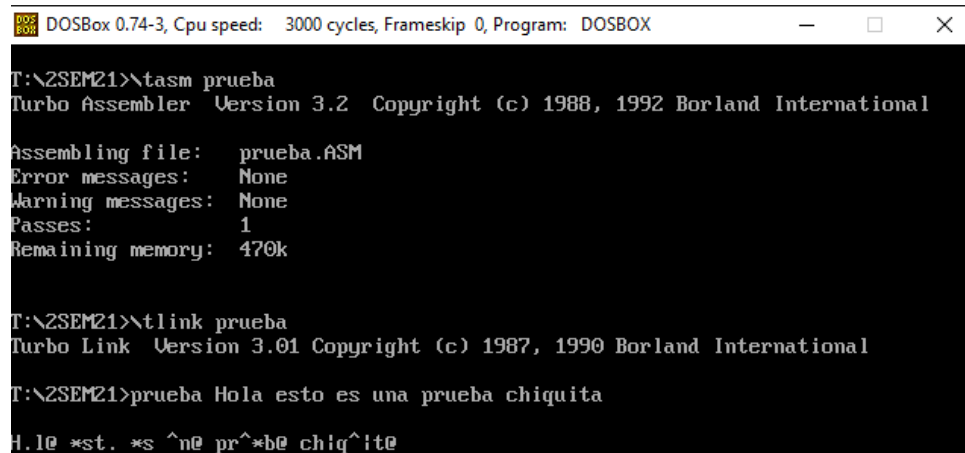
    mov ax, 4C00h              ; protocolo de salida del programa
    int 21h

codigo ends

end main

```

El resultado de ejecutar el programa anterior es:



```

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX
T:\2SEM21>tasm prueba
Turbo Assembler Version 3.2 Copyright (c) 1988, 1992 Borland International

Assembling file:   prueba.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  470k

T:\2SEM21>tlink prueba
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International

T:\2SEM21>prueba Hola esto es una prueba chiquita

H.le *st. *s ^ne pr^*be chiq^ite

```

- c) Programa que recibe en la entrada estándar un número del 0 al 9 y despliega lo que se escriba después esa cantidad de veces.

Asume que el primer caracter es un número entre 0 y 9.

(Puede copiarlo y correrlo para hacer la prueba)

```
datos segment

    rotulo db 128 dup ( ? )          ; aquí va la entrada recibida
    N dw ?                          ; aquí va el número que recibido

datos endS

pila segment stack 'stack'

    dw 256 dup(?)

pila endS

codigo segment
    assume cs:codigo , ds:datos , ss:pila

main:
    mov ax, ds
    mov es, ax

    mov ax, pila
    mov ss, ax

    mov ax, datos
    mov ds, ax

    mov si, 80h                      ; el SI apunta a la entrada estándar
    mov cl, byte ptr es:[si]         ; el cl tiene el tamaño de la entrada
    xor ch, ch

    dec cx                          ; le resta 1 para descontar el espacio inicial
;entre el nombre del programa y la entrada
    inc si                          ; le suma 1 para saltarse dicho espacio

    xor di, di                      ; pone el di en 0
```

```

    inc si                      ; incrementa SI para llegar al primer
;carácter (que asume que es un numero del 0 al 9)

    ; El numero en realidad no es un numero sino el caracter de dicho número,
;según los códigos de la tabla ASCII
    ; al restarle 30h obtenemos el número real

    mov al, byte ptr es:[si]    ; movemos el caracter a al
    xor ah, ah                  ; dejamos ah en 0
    sub al, 30h                 ; le restamos a al 30h

    mov N, ax                   ; movemos a N el número que hay en ax que
;sería el resultado de la resta anterior
    dec cx                      ; le restamos 1 a cx

ciclo:
    inc si                      ; para pasar al siguiente caracter
    mov al, byte ptr es:[si]    ; mueve a al el caracter al que apunta si

    mov byte ptr rotulo[di], al ; mueve a la posición di de la variable
;rótulo, lo que hay en al
    inc di                      ; le suma 1 a di para pasar a la siguiente
;posición

    loop ciclo                  ; repite el ciclo y le resta 1 a cx hasta que
;cx sea 0.

    mov byte ptr rotulo[di], "$" ; le mueve un $ para indicar su final

    mov cx, N
desplegar:
    ; el 13 y el 10 juntos imprimen un cambio de línea
    mov dl, 13
    mov ah, 02h
    int 21h
    mov dl, 10
    int 21h

    ; llama a la instrucción que despliega lo que apunta dx
    mov ah, 09h
    lea dx, rotulo
    int 21h

    loop desplegar

```

```

    mov ax, 4C00h           ; protocolo de salida del programa
    int 21h

codigo ends

end main

```

El resultado de ejecutar el programa anterior es:

```

T:\ZSEM21>\tasm prueba
Turbo Assembler Version 3.2 Copyright (c) 1988, 1992 Borland International

Assembling file:  prueba.ASM
Error messages:   None
Warning messages: None
Passes:          1
Remaining memory: 470k

T:\ZSEM21>\tlink prueba
Turbo Link Version 3.01 Copyright (c) 1987, 1990 Borland International

T:\ZSEM21>prueba 5probando 5 veces

probando 5 veces
probando 5 veces
probando 5 veces
probando 5 veces
probando 5 veces
T:\ZSEM21>_

```