CAPÍTULO 2

PROGRAMACIÓN EN AMBIENTE DE PC

2.1 EL SISTEMA OPERATIVO

El sistema operativo (SO) provee el ambiente para que otros programas se ejecuten. En las computadoras que tienen múltiples usuarios, todos compiten por los recursos de la máquina. El sistema operativo es el árbitro que decide que determinado usuario acceda a determinado recurso y en que momento. El sistema operativo previene que un usuario interfiera con otro. El sistema operativo también ofrece un conjunto de servicios que libera al usuario de lidiar con las dificultades de hardware. Esos servicios mantienen al usuario advertido si pretende realizar operaciones que podrían destruir datos o programas de otros usuarios. Las necesidades de la mayoría de los usuarios reducen las libertades individuales de los programas.

2.1.1 Clasificación

Podemos clasificar a los sistemas operativos en multi-usuarios y mono-usuarios. Los sistemas operativos multi-usuarios deben ejecutarse en computadoras que contengan el hardware que soporte los niveles de privilegio y la protección de memoria de manera que se mantenga la aislación adecuada entre los distintos usuarios. Como ejemplos podemos citar el Windows (2000, NT, XP, Vista, 7 y posteriores), el UNIX y el Linux.

También los podemos considerar como sistemas en tiempo real o no. Los sistemas en tiempo real son los que proveen un servicio en un lapso adecuado a la necesidad del proceso que lo requiere. Tal es el caso de las computadoras de control de procesos. Como ejemplo podemos citar el QNX.

Otra distinción puede ser la interfaz. Están los de acceso por línea de comandos, o texto (DOS) y los de interfaz visual o gráfica (Windows, Linux). En algunos casos la interfaz visual está por encima de un sistema de línea de comandos pero pueden separarse claramente las funciones de cada uno (Linux, Windows 95).

Por último, tenemos los de código abierto (open source) y los privativos. En los primeros está disponible al público el programa fuente, lo que permite revisiones y mejoras soportadas por una comunidad de programadores (Linux), a diferencia de los otros, donde las actualizaciones las hacen las grandes empresas de software (Microsoft).

2.1.2 Programación del SO

Los primeros sistemas operativos se programaron en lenguaje assembly, ya que era lo natural para una interfaz entre el procesador (hardware) y el usuario. Cada instrucción en lenguaje assembly se corresponde con una instrucción del lenguaje de máquina. Esto tenía la ventaja de la eficiencia en el uso de los recursos del harware pero la desventaja de tener que re-escribir todo el código si cambiaba el hardware.

El lenguaje de programación C fue diseñado e implementado en ambiente UNIX, con el objetivo de achicar la "brecha semántica" entre el assembly y el usuario, pero tratando de mantener expresiones breves. Con el tiempo sobrevivió a una cantidad de competidores, siendo hoy uno de los mas usados para crear aplicaciones eficientes y la base de los sistemas UNIX y sus derivados, como Linux. se dice que estos sistemas tienen un 95% de su código en C y sólo el 5% restante en assembly, lo que facilita su adaptación al harware en constante evolución.

2.1.3 Sistema operativo DOS

Para las pequeñas computadoras, como fue en sus comienzos la IBM PC, un sistema operativo mono-usuario era suficiente. Las PC eran utilizadas por una sola persona a la vez. A nivel de programa, no hay prohibición en contra de realizar cualquier secuencia de instrucciones. El propósito de sistema operativo DOS de las PC era proveer un ambiente para operar y un conjunto de servicios al usuario.

El usuario puede ser un programador o puede ser algún otro programa de aplicación. Por ejemplo, cuando usted se sienta frente al teclado y comienza a trabajar con el sistema, usted es un usuario del DOS. Pero cuando el editor salva un archivo en diskette, el editor usa el DOS. El editor está usando los servicios del DOS para salvar el archivo, en lugar de incluir los programas para hacer tal trabajo en el editor.

El servicio principal del DOS en una PC es proveer un sistema de archivos y un ambiente de ejecución para los programas. El sistema de archivos es el método por el cual son salvados y recuperados los datos en diskettes o discos rígidos. Si todos los programas de aplicación usan el DOS para salvar información, entonces ellos pueden compartir dicha información.

2.1.4 Sistema de archivos

Es un conjunto de datos relacionados de alguna manera. El creador del archivo le asigna un nombre. El archivo está compuesto de registros (records). El registro es un conjunto de bytes. El programador define el sentido de los bytes en el registro.

Un programa en assembly puede almacenarse como un archivo con un nombre (PROGRAMA.ASM). El archivo está compuesto de registros, donde cada registro es una única sentencia en lenguaje assembly. Cada registro tiene una composición que tiene significado para el programador y no para el DOS. Las diferentes áreas en cada registro son los campos. Al DOS no le interesa como se dividen los registros en campos; este trabajo se deja para la aplicación, en este caso el ensamblador.

Nombres de archivos

El nombre puede tener de uno a ocho caracteres (NOMBRE.EXT). La extensión, de uno a tres caracteres, indica el tipo de archivo. En el caso del ensamblador, existe un archivo de entrada, con extensión .ASM y tres archivos de salida con extensiones: .OBJ para el objeto, .LST para el listado y .CRF para las referencias cruzadas.

Directorios o carpetas

El sistema operativo maneja el método para almacenar múltiples archivos en el mismo disco. El directorio es una tabla de contenidos, que además del nombre del archivo contiene punteros para localizar la posición física de los archivos en el disco y también la fecha de creación. Si hay mas de un disco, el archivo se nombra agregando la identificación de la unidad de disco donde está instalado (ej.: A:NOMBRE.EXT).

2.1.5 Procesador de comandos

El DOS provee un ambiente para ejecutar los programas de aplicación. La primera parte del DOS que el usuario ve es el procesador de comandos, que responde a los comandos del usuario y comienza la ejecución de los programas de aplicación.

Cuando la PC se enciende, se ejecuta una rutina en ROM que prueba los componentes del sistema (Test de encendido o POST) e inicializa los dispositivos de entrada/salida. El resto del sistema básico de entrada/salida (BIOS) que está en la ROM provee al programador en assembly

un conjunto de servicios para acceder a los dispositivos sin especificar la implementación específica del hardware.

Luego de ejecutar el POST, se carga desde el disco una rutina que será luego la encargada de cargar el resto del sistema operativo (BOOT). Cuando este proceso termina, aparece en pantalla el título y nivel de revisión del DOS y luego el aviso (prompt) A> que indica que el procesador de comandos, el sistema de archivos y otros utilitarios han sido cargados y están listos para correr.

El aviso A> tiene dos significados: el > indica que el intérprete de comandos está esperando un comando y la A indica la unidad de discos que se toma por omisión (default drive). Usualmente A y B se asignan a unidades de diskettes y C a un disco rígido.

Hay comandos incluidos o residentes (built-in) que están siempre disponibles y otros que requieren la existencia de un archivo en disco para su ejecución. El usuario teclea el comando, por ejemplo DIR y el intérprete de comandos pasa el control a la rutina correspondiente del DOS, que realiza la función solicitada y devuelve el control al DOS.

Si el usuario ingresa un comando no residente, el intérprete de comandos intenta cargar la rutina correspondiente desde el disco; en este caso el intérprete de comandos actúa como cargador de programas. El intérprete asume que el nombre del comando es el nombre de un archivo, que buscará en el disco con la extensión .COM o .EXE. Estos son archivos de comando y archivos ejecutables, que están en lenguaje de máquina.

También se pueden invocar para su ejecución los archivos de comandos indirectos (batch), que tienen una extensión .BAT y son archivos de texto que contienen comandos que interpreta el procesador de comandos como si ingresaran por teclado. Hay un archivo de comando especial, llamado AUTOEXEC.BAT que se ejecuta, si existe, inmediatamente luego que el DOS es cargado. Se utiliza para arrancar la computadora de alguna manera particular con programas de aplicación.

2.1.6 Funciones del DOS

A diferencia de los comandos, las funciones son llamadas desde los programas de aplicación, en lenguaje assembly. Un programa ejecuta una función del DOS a través de una interrupción de software, sin saber en qué posición de memoria se encuentra la rutina de atención de esa función. Los vectores de interrupción son cargados por el DOS durante la inicialización del sistema. Una misma interrupción puede realizar distintas funciones, cargando distintos valores en el registro AH antes de ejecutar la instrucción INT.

2.2 CREACIÓN DE UN PROGRAMA EN LENGUAJE ASSEMBLY

2.2.1 Editor

En primer lugar, el EDITOR crea un programa fuente en formato texto. Éste puede ser en lenguaje assembly o lenguaje C. Este programa se almacena en un archivo, generalmente con la extensión .ASM o .C.

Se puede usar cualquier editor de texto, dependiendo de la cantidad de memoria y espacio de almacenamiento disponible. Los programas más grandes, en general brindan prestaciones extra, como indentación automática, diferentes colores para instrucciones y nombres de variables, plantillas, integración con otros programas, etc.

El archivo fuente debe respetar la sintaxis normalizada, tanto para las directivas del ensamblador (ver apéndice al final del apunte) como para las instrucciones del procesador donde será ejecutado el programa. Lo mismo aplica para el compilador C.

La figura es un diagrama de flujo que muestra las actividades involucradas cuando se trabaja en lenguaje assembly y C en una computadora personal compatible. Los hexágonos representan programas utilitarios, los círculos indican archivos en disco, generados por el usuario y por último las flechas guían las actividades.

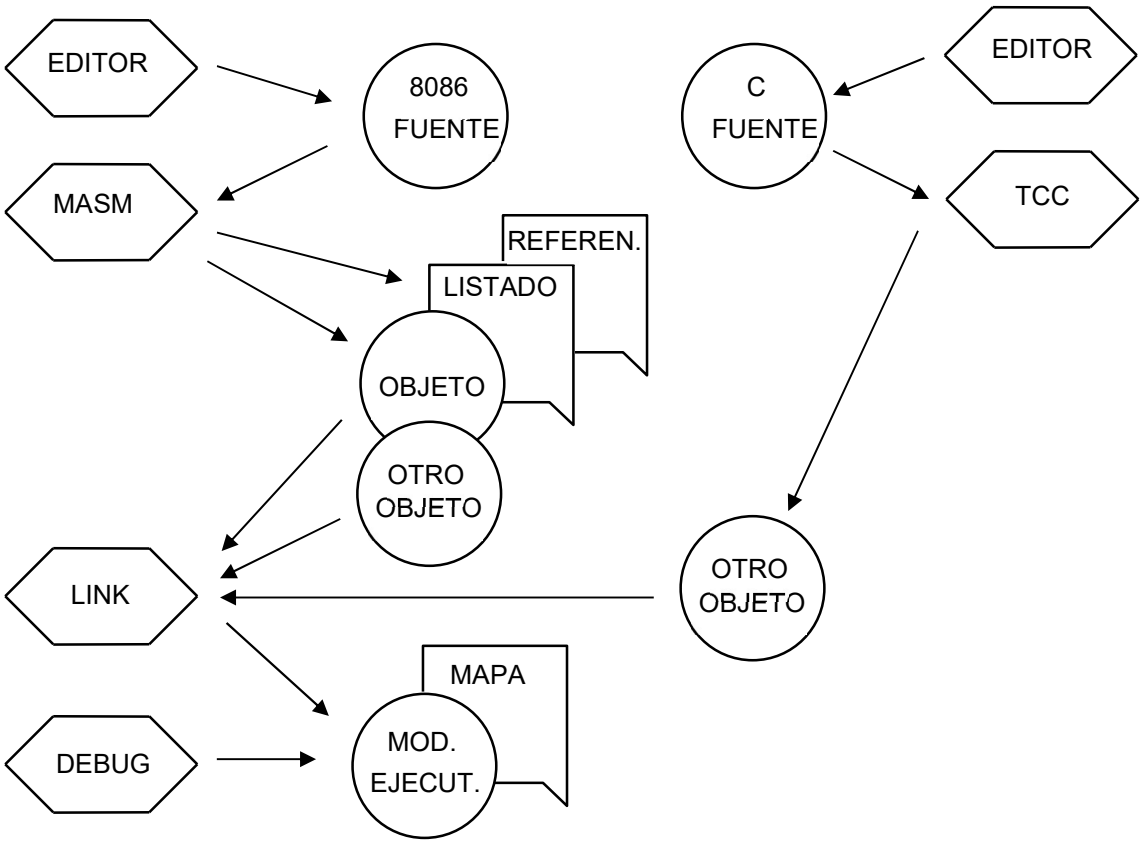

Figura 2.2.1 – Programación en ambiente de PC

2.2.2 Macro ensamblador y turbo ensamblador

Hay dos versiones del ensamblador: el MASM.EXE (macro ensamblador) y el TASM.EXE (turbo ensamblador). Ambos cumplen la misma función, por lo que hablaremos simplemente de ensamblador, estudiándose en la práctica las particularidades de cada uno de ellos.

La entrada al ensamblador es el archivo fuente creado por el editor de textos y está en código ASCII. El ensamblador produce hasta tres archivos de salida. El archivo objeto es la versión en lenguaje de máquina del programa, aunque aún no está lista para ser ejecutada.

El archivo listado es un archivo de texto en ASCII que contiene el tanto la información fuente como la generada por el ensamblador. El archivo de referencias cruzadas, que no está en lenguaje de máquina ni es un texto, contiene información acerca del uso de símbolos y rótulos en

el programa. Al igual que el archivo objeto, el archivo de referencias cruzadas requiere un procesamiento adicional antes de que pueda ser usado.

El ensamblador se activa con un comando DOS:

A>MASM

Luego el ensamblador consulta los nombres de los archivos que intervendrán en el ensamblado. El nombre por omisión del archivo para el programa fuente es .ASM, de manera que no es necesario ingresar la extensión. Si no se ingresa otro nombre, el archivo objeto se denominará igual que el archivo fuente, pero con la extensión .OBJ. Para el listado y las referencias cruzadas, en caso de responder RETURN (o ENTER) se utilizará el archivo NUL, que no puede ser leído, como valor por omisión. Luego del ensamblado se indica si hubo errores de advertencia (warning errors) o errores graves (severe errors) tanto en pantalla como en el listado.

Otro método para invocar al ensamblador es:

A>MASM PROG,,,;

donde PROG es el nombre del archivo fuente y las comas indican que se deben tomar las asignaciones por omisión para los nombres de los otros archivos.

El archivo listado se puede ver con el comando DOS:

A>TYPE PROG.LST

Oprimiendo CONTROL y PrtScrn antes del comando TYPE, se puede enviar el listado a la impresora.

Tabla de símbolos

El archivo listado contiene la tabla de símbolos, siguiendo al programa. La misma muestra los segmentos y símbolos usados en el programa. Los símbolos pueden ser variables, rótulos o números y van acompañados por sus atributos, tipo y valor.

Referencias cruzadas

El archivo de referencias cruzadas producido por el ensamblador no está listo para ser usado. Se debe ejecutar el comando CREF para cambiar el archivo .CRF en un archivo de texto. El comando DOS:

A>CREF PROG,

crea el archivo PROG.REF, que puede ser mostrado con el comando TYPE.

La columna de la izquierda contiene los nombres de las variables y símbolos definidos en el programa. A la derecha siguen un grupo de números enteros que indican la línea de programa donde el símbolo aparece. Si el número es seguido de "#" indica que el símbolo fue definido en esa línea. Esto es útil cuando se sigue la pista de una variable cuyo valor es incorrecto: la tabla proporciona los números de línea que hacen referencia a esa variable. También en el caso en que se modifique una subrutina, la tabla indica las porciones de programa que efectúan llamados a dicha subrutina, permitiendo evaluar los efectos de la modificación en cada tramo del programa.

2.2.3 Vinculación

El archivo PROG.OBJ se transforma en PROG.EXE mediante el comando DOS:

A>LINK PROG.OBJ, PROG.EXE, PROG.MAP,,

2.2.4 Compilación

El archivo PROG_C.C se transforma en PROG_C.OBJ y luego en PROG_C.EXE

A>TCC –PROG_C.EXE PROG_C.C *PROG.OBJ (opcional, para incluir .OBJ desde Assembly)*

2.2.5 Depuración

Para ejecutar los programas en ambiente de prueba se utilizan DEBUG y TD.

A>TD

En la práctica se utilizará el ambiente Turbo, que consta de los programas AE (editor), TASM (ensamblador), TLINK (vinculador), TD (depurador) y TREF (referencias). Este editor fue concebido especialmente para trabajar en un ambiente de desarrollo en assembly.

Como alternativa, si se trabaja en ambiente Windows, se puede usar el Notepad y los ambientes de Microsoft o Borland. Existen también IDE (Integrated Development Environment) como Negatory Assembly Studio que están diseñados para Windows y Linux y son gratuitos.

2.3 EJEMPLOS DE PROGRAMACION EN ASSEMBLY

A continuación, se muestra un ejemplo de programa y subrutina, que luego serán ensamblados y vinculados en la PC.

Los archivos que se mencionan están incluidos en el CD de Técnicas Digitales III y en la página web de la materia (http://www.frsn.utn.edu.ar/tecnicas3).

2.3.1 Contenido del archivo ENSAYO.ASM

```
                              ;     PROGRAMA ENSAYO
                              ;
                              ; Este programa crea una tabla de
                              ; datos y llama la subrutina ORDEN_DESC
                              ; pasando la direccion del primer dato
                              ; y la cantidad de datos por la pila.
                              ;
DATOS SEGMENT                 ; Define un segmento llamado DATOS
N     DW  5                   ; Define la variable N con el valor 5
TABLA DW 1,2,3,4,5            ; Define la variable TABLA con el
                              ; valor 1 y asigna valores a cuatro
                              ; palabras mas
CARTEL1 DB 'TECNICAS DIGITALES III$'
CARTEL2 DB ' - LOS DATOS HAN SIDO ORDENADOS$'
DATOS       ENDS              ; Fin del segmento DATOS
                              ;
PILA SEGMENT STACK            ; Define un segmento PILA de tipo STACK
DW  128 DUP (?)               ; Reserva 128 palabras
CAB_PILA LABEL WORD           ; Nombra CAB_PILA al tope del stack
PILA  ENDS                    ; Fin del segmento PILA
                              ;
PROG SEGMENT PUBLIC           ; Define un segmento PROG de tipo PUBLIC
EXTRN ORDEN_DESC:NEAR         ; Define subrutina externa y proxima
ASSUME CS:PROG,DS:DATOS,SS:PILA; Asocia CS al segmento PROG , DS al
                              ; segmento DATOS y SS al segmento PILA
                              ;
COMENZAR:   MOV AX,DATOS      ; Inicializa el registro DS con el
            MOV DS,AX         ; valor DATOS, obtenido por el MASM
            MOV AX,PILA       ; Inicializa el registro SS con el
            MOV SS,AX         ; valor PILA.
```

```
            LEA SP,CAB_PILA   ; Inicializa SP con el valor CAB_PILA
                              ;
            LEA DX,CARTEL1    ; Carga DX con la direccion de comienzo
            MOV AH,9          ; Selecciona la funcion imprimir texto
            INT 21H           ; Llama a la funcion imprimir texto
                              ;
            LEA AX,TABLA      ; Carga AX con la direccion de TABLA
            PUSH AX           ; Salva la direccion en la pila
            MOV AX,N          ; Carga AX con la cant. de datos (N)
            PUSH AX           ; Salva la cant. de datos en la pila
            CALL ORDEN_DESC   ; Llama a la sub. que ordena los datos
                              ;
            LEA DX,CARTEL2    ; Carga DX con la direccion de comienzo
            MOV AH,9          ; Selecciona la funcion imprimir texto
            INT 21H           ; Llama a la funcion imprimir texto
                              ;
            MOV AH,4CH        ; Inicializa la funcion Retorno al DOS
            INT 21H           ; Llama a la funcion Retorno al DOS
                              ;
PROG  ENDS                    ; Fin del segmento PROG
END   COMENZAR                ; Direccion de transferencia luego de
                              ; cargar el programa en memoria.
```

2.3.2 Contenido del archivo ORDENDES.ASM

```
                              ; SUBRUTINA ORDEN_DESC
                              ;
                              ; Esta subrutina recibe en la pila la
                              ; direccion de una tabla de datos y
                              ; la extension de la misma. Los datos se
                              ; ordenan de mayor a menor, quedando
                              ; la tabla en su posicion original en
                              ; memoria. Los registros utilizados
                              ; por la subrutina se salvan en la pila
                              ; para luego recuperarse.
                              ;
PROG  SEGMENT  PUBLIC         ; Define un segmento llamado PROG
ASSUME CS:PROG                ; Asocia CS al segmento PROG
PUBLIC ORDEN_DESC             ; Define el rotulo ORDEN_DESC publico
                              ;
ORDEN_DESC PROC NEAR          ; Comienza proced. llamado ORDEN_DESC
            PUSH BP           ; Salva el registro BP en la pila
            MOV BP,SP         ; Carga BP con el puntero SP actual
            PUSH AX           ; Salva el registro AX en la pila
            PUSH BX           ; Salva el registro BX en la pila
            PUSH CX           ; Salva el registro CX en la pila
            PUSH SI           ; Salva el registro SI en la pila
            PUSH DI           ; Salva el registro DI en la pila
            MOV BX,[BP+6]     ; Mueve TABLA al índice BX
            MOV CX,[BP+4]     ; Mueve N al contador CX
            DEC CX            ; Decrementa el contador
LOOP1:      MOV DI,CX         ; Salva CX en DI p/ la prox. iteracion
            MOV SI,0          ; Limpia SI
LOOP2:      MOV AX,[BX][SI]   ; Mueve un dato de la tabla a AX
            CMP AX,[BX][SI+2] ; Compara con el dato sig. en la tabla
            JGE SEGUIR        ; Si el orden es correcto seguir
            XCHG AX,[BX][SI+2]; Intercambia datos p/ lograr el orden
            MOV [BX][SI],AX   ;
SEGUIR:     ADD SI,2          ; Increm. indice p/ la prox. iteracion
            LOOP LOOP2        ; Retorna a LOOP2 hasta que CX sea cero
            MOV CX,DI         ; Restaura CX
```

```
            LOOP LOOP1        ; Retorna a LOOP1 hasta CX (DI)sea cero
            POP DI            ; Restaura el registro DI
            POP SI            ; Restaura el registro SI
            POP CX            ; Restaura el registro CX
            POP BX            ; Restaura el registro BX
            POP AX            ; Restaura el registro AX
            POP BP            ; Restaura el registro BP
            RET 4             ; Retorna al programa principal e
                              ; incrementa en 4 el registro SP
                              ;
ORDEN_DESC        ENDP        ; Fin del procedimiento ORDEN_DESC
PROG              ENDS        ; Fin del segmento PROG
END                           ; Fin del modulo a ensamblar
```

2.3.3 Proceso de ensamblado y vinculación

>path = %path%;d:\Programs\MASM51 (agrega los directorios de MASM51)

>masm ensayo.asm,ensayo.obj,ensayo.lst,ensayo.crf

```
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988.  All rights reserved.

  46474 + 436643 Bytes symbol space free
      0 Warning Errors
      0 Severe  Errors
```

crea ensayo.obj, ensayo.lst y ensayo.crf

>cref ensayo.crf,ensayo.ref

```
Microsoft (R) Cross-Reference Utility  Version 5.10
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

12 Symbols
```

crea ensayo.ref

>masm ordendes.asm,ordendes.obj,ordendes.lst,ordendes.crf

```
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988.  All rights reserved.

  46274 + 436843 Bytes symbol space free
      0 Warning Errors
      0 Severe  Errors
```

crea ordendes.obj, ordendes.lst y ordendes.crf

>cref ordendes.crf,ordendes.ref

```
Microsoft (R) Cross-Reference Utility  Version 5.10
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

7 Symbols
```

crea ordendes.ref

>link ensayo.obj+ordendes.obj,ensayo.exe,ensayo.map,,

```
Microsoft (R) Overlay Linker  Version 3.64
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.
```

crea ensayo.exe y ensayo.map

2.3.4 Ejecución del programa en la línea de comando DOS

```
>ensayo

>TECNICAS DIGITALES III - LOS DATOS HAN SIDO ORDENADOS
```

Antes de ordenar los datos, con el programa TD se puede observar la tabla de datos, que contiene los 5 valores cargados por defecto.

```
ds:0100 05 00 01 00 02 00 03 00 ♣ ☺ ☻ ♥
ds:0108 04 00 05 00 54 45 43 4E ♦ ♣ TECN
```

La tabla de datos ordenada luego de la ejecución queda así:

```
447C:0100 05 00 05 00 04 00 03 00 ♣ ♣ ♦ ♥
447C:0108 02 00 01 00 54 45 43 4E ☻ ☺ TECN
```

Observe que para la ejecución se asignó un valor a DS.

2.3.5 Contenido del archivo ENSAYO.LST

```
Microsoft (R) Macro Assembler Version 5.10

1                               ; PROGRAMA ENSAYO
2                               ;
3                               ; Este programa crea una tabla de
4                               ; datos y llama la subrutina ORDEN_DESC
5                               ; pasando la direccion del primer dato
6                               ; y la cantidad de datos por la pila.
7                               ;
8 0000                          DATOS SEGMENT
                                ; Define un segmento llamado DATOS
9 0000  0005                    N     DW  5
                                ; Define la variable N con el valor 5
10 0002  0001 0002 0003 0004    TABLA DW 1,2,3,4,5
                                ; Define la variable TABLA con el valor 1
11       0005
12                              ; y asigna valores a cuatro palabras mas
13 000C  54 45 43 4E 49 43      CARTEL1 DB 'TECNICAS DIGITALES III$'
14       41 53 20 44 49 47
15       49 54 41 4C 45 53
16       20 49 49 49 24
17 0023  20 2D 20 4C 4F 53      CARTEL2 DB ' - LOS DATOS HAN SIDO ORDEN
                                ADOS$'
18       20 44 41 54 4F 53
19       20 48 41 4E 20 53
20       49 44 4F 20 4F 52
21       44 45 4E 41 44 4F
22       53 24
23 0043                         DATOS ENDS
                                ; Fin del segmento DATOS
24                              ;
25 0000                         PILA SEGMENT STACK
                                ; Define un segmento PILA de tipo STACK
26 0000  0080[                        DW  128 DUP (?)
                                ; Reserva 128 palabras
27       ????
28                 ]
29
30 0100                         CAB_PILA LABEL WORD
```

```
                                ; Nombra CAB_PILA al tope del stack
31 0100                         PILA ENDS
                                ; Fin del segmento PILA
32                              ;
33 0000                         PROG SEGMENT PUBLIC
                                ; Define un segmento PROG de tipo PUBLIC
34                              EXTRN ORDEN_DESC:NEAR
                                ; Define subrutina externa y proxima
35                              ASSUME CS:PROG,DS:DATOS,SS:PILA
                                ; Asocia CS al segmento PROG ,DS al seg.
36                              ; DATOS y SS al seg. PILA
37                              ;
38 0000  B8 ---- R              COMENZAR: MOV AX,DATOS
                                ; Inicializa el registro DS con el valor
                                ; DATOS, obtenido por el ensamblador
39 0003  8E D8                           MOV DS,AX
40 0005  B8 ---- R                       MOV AX,PILA
                                ; Inicializa el registro SS con el valor
                                ; PILA , obtenido por el ensamblador
41 0008  8E D0                           MOV SS,AX
42 000A  8D 26 0100 R                    LEA SP,CAB_PILA
                                ; Inicializa SP con el valor CAB_PILA
43                              ;
44 000E  8D 16 000C R                    LEA DX,CARTEL1
                                ; Carga DX con la direccion de comienzo
45 0012  B4 09                           MOV AH,9
                                ; Selecciona la funcion imprimir texto
46 0014  CD 21                           INT 21H
                                ; Llama a la funcion imprimir texto
47                              ;
48 0016  8D 06 0002 R                    LEA AX,TABLA
                                ; Carga AX con la direccion de TABLA
49 001A  50                              PUSH AX
                                ; Salva la direccion en la pila
50 001B  A1 0000 R                       MOV AX,N
                                ; Carga AX con la cantidad de datos (N)
51 001E  50                              PUSH AX
                                ; Salva la cantidad de datos en la pila
52 001F  E8 0000 E                       CALL ORDEN_DESC
                                ; Llama a la subr. que ordena los datos
53                              ;
54 0022  8D 16 0023 R                    LEA DX,CARTEL2
                                ; Carga DX con la direccion de comienzo
55 0026  B4 09                           MOV AH,9
                                ; Selecciona la funcion imprimir texto
56 0028  CD 21                           INT 21H
                                ; Llama a la funcion imprimir texto
57                              ;
58 002A  B4 4C                           MOV AH,4CH
                                ; Inicializa la funcion Retorno al DOS
59 002C  CD 21                         INT 21H
                                ; Llama a la funcion Retorno al DOS
60                              ;
61 002E                         PROG ENDS
                                ; Fin del segmento PROG
62                              END COMENZAR
                                ; Direccion de transf. luego de cargar

Symbols-1
Segments and Groups:
```

```
                N a m e             Length       Align      Combine Class

DATOS  . . . . . . . . . . . . . .  0043  PARA  NONE
PILA . . . . . . . . . . . . . . .  0100  PARA  STACK
PROG . . . . . . . . . . . . . . .  002E  PARA  PUBLIC

Symbols:
                N a m e             Type   Value       Attr

CAB_PILA . . . . . . . . . . . . .  L WORD      0100  PILA
CARTEL1  . . . . . . . . . . . . .  L BYTE      000C  DATOS
CARTEL2  . . . . . . . . . . . . .  L BYTE      0023  DATOS
COMENZAR . . . . . . . . . . . . .  L NEAR      0000  PROG

N  . . . . . . . . . . . . . . . .  L WORD      0000  DATOS

ORDEN_DESC . . . . . . . . . . . .  L NEAR      0000  PROG  External

TABLA  . . . . . . . . . . . . . .  L WORD      0002  DATOS

@CPU . . . . . . . . . . . . . . .  TEXT  0101h
@FILENAME  . . . . . . . . . . . .  TEXT  ensayo
@VERSION . . . . . . . . . . . . .  TEXT  510

     50 Source  Lines
     50 Total   Lines
     15 Symbols

  46474 + 436643 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```

2.3.6 Contenido del archivo ENSAYO.REF

```
Microsoft Cross-Reference  Version 5.10
Symbol Cross-Reference      (# definition, + modification)  Cref-1

@CPU . . . . . . . . . . . . . . .    1#
@VERSION . . . . . . . . . . . . .    1#

CAB_PILA . . . . . . . . . . . . .   30#    42
CARTEL1. . . . . . . . . . . . . .   13#    44
CARTEL2. . . . . . . . . . . . . .   17#    54
COMENZAR . . . . . . . . . . . . .   38#    62

DATOS. . . . . . . . . . . . . . .    8#    23     35     38

N. . . . . . . . . . . . . . . . .    9#    50

ORDEN_DESC . . . . . . . . . . . .   34#    52

PILA . . . . . . . . . . . . . . .   25#    31     35     40
PROG . . . . . . . . . . . . . . .   33#    35     61

TABLA. . . . . . . . . . . . . . .   10#    48

 12 Symbols
```

2.3.7 Contenido del archivo ORDENDES.LST

```
Microsoft (R) Macro Assembler Version 5.10

1                              ; SUBRUTINA ORDEN_DESC
2                              ;
3                              ; Esta subrutina recibe en la pila la
4                              ; direccion de una tabla de datos y
5                              ; la extension de la misma. Los datos se
6                              ; ordenan de mayor a menor, quedando
7                              ; la tabla en su posicion original en
8                              ; memoria. Los registros utilizados
9                              ; por la subrutina se salvan en la pila
10                             ; para luego recuperarse.
11                             ;
12 0000                        PROG  SEGMENT  PUBLIC
                               ; Define un segmento llamado PROG
13                             ASSUME CS:PROG
                               ; Asocia CS al segmento PROG
14                             PUBLIC ORDEN_DESC
                               ; Define rotulo ORDEN_DESC publico
15                             ;
16 0000                        ORDEN_DESC  PROC  NEAR
                               ; Comienza el procedimiento ORDEN_DESC
17 0000  55                          PUSH BP
                               ; Salva el registro BP en la pila
18 0001  8B EC                       MOV BP,SP
                               ; Carga indice BP con puntero SP actual
19 0003  50                          PUSH AX
                               ; Salva el registro AX en la pila
20 0004  53                          PUSH BX
                               ; Salva el registro BX en la pila
21 0005  51                          PUSH CX
                               ; Salva el registro CX en la pila
22 0006  56                          PUSH SI
                               ; Salva el registro SI en la pila
23 0007  57                          PUSH DI
                               ; Salva el registro DI en la pila
24 0008  8B 5E 06                    MOV BX,[BP+6]
                               ; Mueve TABLA al indice BX
25 000B  8B 4E 04                    MOV CX,[BP+4]
                               ; Mueve N al contador CX
26 000E  49                          DEC CX
                               ; Decrementa el contador
27 000F  8B F9                 LOOP1:MOV DI,CX
                               ; Salva CX en DI para proxima iteracion
28 0011  BE 0000                     MOV SI,0
                               ; Limpia SI
29 0014  8B 00                 LOOP2:MOV AX,[BX][SI]
                               ; Mueve un dato de la tabla a registro AX
30 0016  3B 40 02                    CMP AX,[BX][SI+2]
                               ; Compara con dato siguiente en la tabla
31 0019  7D 05                       JGE SEGUIR
                               ; Si el orden es correcto seguir
32 001B  87 40 02                    XCHG AX,[BX][SI+2]
                               ; Intercambia datos para lograr el orden
33 001E  89 00                       MOV [BX][SI],AX
                               ;
34 0020  83 C6 02              SEGUIR:ADD SI,2
                               ; Increm. indice para proxima iteracion
35 0023  E2 EF                       LOOP LOOP2
```

```
                                ; Retorna a LOOP2 hasta que CX sea cero
36 0025  8B CF                        MOV CX,DI
                                ; Restaura CX
37 0027  E2 E6                        LOOP LOOP1
                                ; Retorna a LOOP1 hasta CX (DI) sea cero
38 0029  5F                           POP DI
                                ; Restaura el registro DI
39 002A  5E                           POP SI
                                ; Restaura el registro SI
40 002B  59                           POP CX
                                ; Restaura el registro CX
41 002C  5B                           POP BX
                                ; Restaura el registro BX
42 002D  58                           POP AX
                                ; Restaura el registro AX
43 002E  5D                           POP BP
                                ; Restaura el registro BP
44 002F  C2 0004                      RET 4
                                ; Retorna al prog. principal e incrementa
45                              ; en 4 el registro SP
46                              ;
47 0032                         ORDEN_DESC  ENDP
                                ; Fin del procedimiento ORDEN_DESC
48 0032                         PROG        ENDS
                                ; Fin del segmento PROG
49                                    END
                                ; Fin del modulo a ensamblar

Symbols-1
Segments and Groups:

                N a m e                 Length       Align      Combine Class

PROG . . . . . .                0032  PARA  PUBLIC

Symbols:
                N a m e                 Type   Value       Attr

LOOP1  . . . . . .              L NEAR      000F  PROG
LOOP2  . . . . . .              L NEAR      0014  PROG

ORDEN_DESC . . . . . .          N PROC      0000  PROG  Global      Length
= 0032

SEGUIR . . . . . .              L NEAR      0020  PROG

@CPU . . . . . .                TEXT  0101h
@FILENAME  . . . . . .          TEXT  ordendes
@VERSION . . . . . .            TEXT  510

     49 Source  Lines
     49 Total   Lines
     10 Symbols

  46274 + 436843 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```

2.3.8 Contenido del archivo ORDENDES.REF

```
Microsoft Cross-Reference  Version 5.10

Symbol Cross-Reference      (# definition, + modification)  Cref-1

@CPU . . . . . .    1#
@VERSION . . . . . .    1#

LOOP1. . . . . .   27#    37
LOOP2. . . . . .   29#    35

ORDEN_DESC . . . . . .   14     16#    47

PROG . . . . . .   12#    13     48

SEGUIR . . . . . .   31     34#
```

2.3.9 Contenido del archivo ENSAYO.MAP

```
 Start  Stop   Length Name                   Class
 00000H 00042H 00043H DATOS
 00050H 0014FH 00100H PILA
 00150H 001B1H 00062H PROG

Program entry point at 0015:0000
```

2.4 EJEMPLOS DE PROGRAMACION EN C Y ASSEMBLY

A continuación se muestra un ejemplo de programa en C y una subrutina en assembly, que luego serán compilados, ensamblados y vinculados en la PC.

Los archivos que se mencionan están incluidos en el CD de Técnicas Digitales III y en la página web de la materia (http://www.frsn.utn.edu.ar/tecnicas3).

2.4.1 Contenido del archivo PRIMOS.C

```
/* devuelve los numeros primos hasta el 1000 */
#include <stdio.h>
extern int primo(int n);  /* definimos el prototipo de la funcion */
main()
{
  int i,n;
  n=1000;
  for (i=2;i<=n;i++)
    if (primo(i))         /* llamamos a la funcion externa en asm */
      printf("%d ",i);
  return 0;
}
```

2.4.2 Contenido del archivo CALCULA.ASM

```
; Decide si un numero que se le pasa como parametro es primo o no
; Devuelve false (0) o true (!0) al programa en C
; Se invoca desde C: primo(i) siendo i de tipo int (16 bits)
;
.MODEL SMALL                ;Usamos el modelo SMALL y debe coincidir
                            ;con el usado en C
.CODE                       ;Solo usamos el segmento de codigo
```

```
PUBLIC      _primo            ;Se declara el procedimiento como publico
                              ;para que pueda ser llamado desde otro modulo
_primo      PROC NEAR         ;
            PUSH BP           ;Guardamos BP
            MOV BP, SP        ;Recuperamos en BP el puntero de la pila
            MOV SI, [BP+4]    ;Recuperamos en SI el parámetro que le hemos
                              ;pasado [BP+4] (2 del BP + 2 de la direccion
                              ;de retorno)
            MOV BX, 2         ;BX contiene el divisor (de 2 a SI)
BUCLE:      XOR DX, DX        ;Dividendo DX&AX
            MOV AX, SI        ;Divisor BX
            CMP AX, BX        ;Si ya se ha dividido por todos
            JE  PRIMO         ;indica que es primo
            DIV BX            ;Dividimos DX&AX / BX
            OR DX, DX         ;DX contiene el resto de la division
            JZ NOPRIMO        ;si es 0 el numero no es primo
            INC BX            ;Incrementamos el divisor
            JMP BUCLE         ;volvemos a dividir
PRIMO:      MOV AX, 1         ;Colocamos el valor de retorno a 1 (true en C)
            JMP FIN           ;
NOPRIMO:    XOR AX, AX        ;Colocamos el valor de retorno a 0 (falso en C)
FIN:        POP BP            ;Quitamos el valor de BP que guardamos
                              ;al principio
            RET               ;Retornamos
_primo      ENDP              ;
            END               ;
```

2.4.3 Proceso de ensamblado, compilación y vinculación

>path = %path%;d:\Programs\TC201 (agrega los directorios de TC201)

>tasm /ml calcula.asm

```
Turbo Assembler  Version 1.0  Copyright (c) 1988 by Borland International

Assembling file:   CALCULA.ASM
Error messages:    None
Warning messages:  None
Remaining memory:  483k
```

crea calcula.obj

>tcc –Eprimos.exe primos.c calcula.obj

```
Turbo C  Version 2.01  Copyright (c) 1987, 1988 Borland International
primos.c:
```

crea primos.obj

```
Turbo Link  Version 2.0  Copyright (c) 1987, 1988 Borland International
         Available memory 424408
```

crea primos.exe

2.4.4 Ejecución del programa en la línea de comando DOS

>**primos**

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101
103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197
```

```
199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311
313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431
433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557
563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661
673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809
811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937
941 947 953 967 971 977 983 991 997
```