Navegando Linux beta

Horacio Goetendía Bonilla. HGB

Copyright

Comuniquese

La presente tentativa del libro no se salva de errores simples o garrafales. Si encuentras alguno por favor comuniquese a Horacio_Goetendia@Hotmail.com ó goetendia@linuxmail.org que gustoso lo atenderé para corregir el error.

Si alguien tiene alguna crítica constructiva o destructiva sobre este Libro, por favor que la envíe a Horacio_Goetendia@Hotmail.com ó goetendia@linuxmail.org que trataré de hacer un esfuerzo para corregirla y así seguir aprendiendo todos.

Sitio oficial

La tentativa de Libro puede ser descargada de la página oficial:

http://LinuxHGB.linuxeros.org

Nota Introductoria

La presente tentativa de libro está en la fase inicial de desarrollo y paulatinamente serán siendo añadidas nuevas secciones y capítulos si eres alguien a quien le interesaría participar en la elaboración y desarrollo de este Libro por favor contactese a Horacio_Goetendia@Hotmail.com ó goetendia@linuxmail.org para que compartas tus conocimientos y mantener una política similar a la de Linux el de ser un sistema que englobe a desarrolladores de todo el mundo.

La distribución Linux que se usó para este propósito es *Red Hat 7.3* y por omisión el código fuente del sistema se operativo se encuentra en la carpeta `/usr/src/linux-2.4` por lo cual omito la ruta base y escribo b asado en la ruta relativa del código fuente, salvo indicaciones de rutas absolutas.

El modo de explicación que se emplea tiene un énfasis en la ilustración de las acciones del sistema operativo Linux para un mayor aprendizaje. Empezando desde el *booteo* del sistema.

Por favor cualquier error u omisión comuniquese a Horacio_Goetendia@Hotmail.com ó goetendia@linuxmail.org. Herrar es humano ¡¡¡¡Perdón!!!! Errar es humano :-).

El Libro está siendo desarrollado en LaTeX 2_ε.

Capítulo 1

Construcción de la Imagen del Kernel de Linux

1.1. El *booteo* en diskette

Un diskette esta estructurado de la manera en que se muestra en la Figura 1.1:

El sector de *booteo* es cargado a memoria y ejecutado, en esta pieza de código debe existir un pequeño programa que cargue la imagen del kernel que está en el disco a la memoria y ejecutarlo, a este programa encargado de cargar la imagen del kernel se le denomina *bootstrap*.

Para comenzar con la exploración del código fuente de Linux y observar su funcionamiento desde el inicio comencemos explorando archivo `arch/i386/boot/bootsect.S` encargado del *booteo* de linux. Pero antes mencionemos que en todo sistema:

- La BIOS cargará el *bootstrap* a la dirección de memoria `07C00h`.
- El bootstrap debe ser compilado como un archivo binario plano.
- El tamaño del archivo binario no debe exceder los 512 bytes.
- El archivo debe culminar con el *boot signature* AA55h.

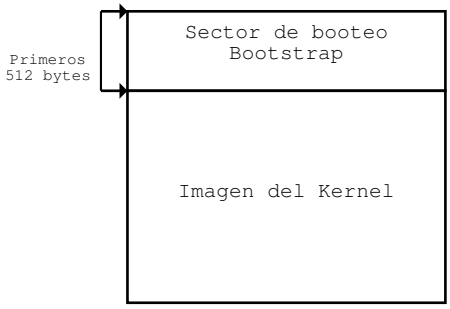


Figura 1.1: Estructura de un diskette.

Como visión general de lo que es el *booteo* de un diskette y tener una referencia de lo que hace y de que queremos entender empleemos el siguiente ejemplo práctico:

Compilemos el archivo *bootsect.S* como un archivo binario plano como se muestra a continuación [1].

```
$ gcc -c bootsect.S
$ ld -o bootsect -Ttext 0x0 -e main bootsect.o
$ objcopy -R .note -R .comment -S -O binary bootsect bootsect.bin
```

Una vez compilado obtenemos el archivo `bootstrap.bin`, como dijimos anteriormente este archivo debe ser de 512 bytes y lo volcamos a un un diskette con el comando `dd` de Linux.

```
$ dd if=bootsect.bin of=/dev/fd0
```

Si se tiene otra computadora a la mano introducir el diskette y encienda la computadora y observe lo que sucede.

1.2. La estructura del programa `bootsect.S`

1.2.1. Paso1: Las definiciones de segmentos de memoria

```
#include <asm/boot.h>

SETUPSECTS              =   4               /* default nr of setup-sectors*/
BOOTSEG                 =   0x07C0          /* original address of boot-sector*/
INITSEG                 =   DEF_INITSEG     /* we move boot here-out of the way*/
SETUPSEG                =   DEF_SETUPSEG    /* setup starts here */
SYSSEG                  =   DEF_SYSSEG      /* system loaded at 0x10000 (65536)*/
SYSSIZE                 =   DEF_SYSSIZE     /* system size: # of 16-byte clicks*/
                                            /* to be load*/
```

Sabemos que en assembler las direcciones de inicio de segmento son múltiplos de 16 (en base decimal) por eso se obvia el último cero en una dirección de segmento (en base hexadecimal). Por lo tanto las direcciones que se encuentran en el archivo cabecera `asm/boot.h` que se muestra a continuación:

```
#define   DEF_INITSEG    0x9000
#define   DEF_SYSSEG     0x1000
#define   DEF_SETUPSEG   0x9020
#define   DEF_SYSSIZE    0x7F00
```

Tienen sus equivalencias en direcciones lineales de la siguiente manera:

SIMBOLO	SIGNIFICADO	Dirección de segmento	Dirección lineal
Especificado en bootsect.S			
BOOTSEG	Segmento de booteo	0x07C0	0x007C00
Especificado en asm/boot.h			
INITSEG	Segmento de Inicio	0x9000	0x090000
SYSSEG		0x1000	0x010000
SETUPSEG		0x9020	0x090200
SYSSIZE		0x7F00	0x07F000

[1] Para el ejemplo es necesario tener instalado el paquete de los compiladores del C

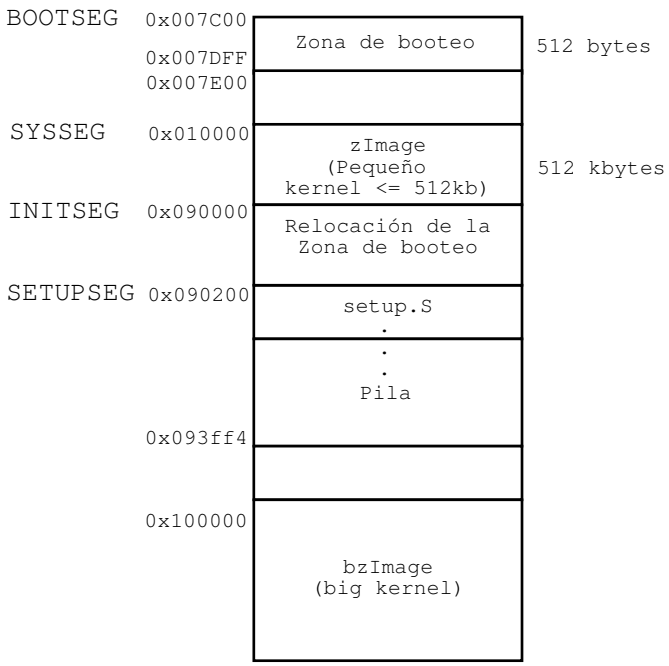


Figura 1.2: Disposición de la memoria

La posición de memoria en la que la BIOS cargará la zona de *booteo* es la 0x07C0 definido por BOOTSEG, estas definiciones de posiciones de segmentos sirven para establecer de que manera va a estar estructurada la memoria, especificándola como se muestra en la Figura 1.2.1

Como una nota aclaratoria antes de revisar a fondo del código en ensamblador del `bootsect.S` menciono que: El modo de integrar lenguaje ensamblador en código de C o C++ compilado con GCC no es en base a la sintaxis de intel sino es basado en la sintaxis AT&T donde:

1. Cambia el orden de los operandos

Sintaxis intel	operador	destino, origen
Sintaxis AT&T	operador	origen, destino

2. Se agrega $ a los valores inmediatos y % al nombre de los registros

3. Se añade el sufijo b,w,l, al *opcode* de operación para indicar byte, word, long bits respectivamente

$$\begin{array}{llcll} \text{mov} & \text{ah,5} & \text{——} & \text{movb} & \$5, \%\text{ah} \\ \text{mov} & \text{ax,15} & \text{——} & \text{movw} & \$15, \%\text{ax} \end{array}$$

Estas no son la únicas diferencia entre la notación Intel y la de AT&T para más información por favor revisar la documentación pertinente.

1.2.2. Paso 2: Salto al segmento de inicio

```
1   .code16
2
3   .text
4   .global _start
5   _start:
6
7   # First things.   Move ourself from   0x7C00 ->0x90000 and jump there.
8
9   movw              $BOOTSEG,%ax
10  movw              %ax,%ds             #%ds = BOOTSEG
11  movw              $INITSEG,%ax
12  movw              %ax,%es             #%ax =%es = INITSEG
13  movw              $256,%cx
14  subw              %si,%si
15  subw              %di,%di
16  cld
17  rep
18  movsw
19  ljmp              $INITSEG, $go
20
```

En la línea 9 y 10 se asigna al segmento de datos (ds) la posición del segmento de booteo BOOTSEG=07C00, en la línea 11 y 12 se asigna al segmento extra (es) ls posición de INITSEG=0x900000 se ubica 256 en el registro cx para obtener 512 bytes por ser 256 por 2 bytes de una palabra nos dan los 512 bytes de la zona de *booteo* las líneas 14 y 15 son encargadas de inicializar a 0 los registros origen y destino (si,di) las líneas 16, 17 y 18 son encargadas de copiar los 256*2 bytes de la zona de booteo a la nueva zona que inicia en INITSEG=0x090000, la línea 19 es un salto largo (long jump) a la zona de base INITSEG + el desplazamiento *go* esto es para saltar a la nueva zona de relocación y no volver a ejecutar el código que hasta ahora ejecutamos. De una manera más explicativa lo ilustramos en en la Figura 1.3

1.2.3. Paso 3: Determinación de la pila y el establecimiento de parámetros

Como se vio en la Figura 1.2.1 se determina también una pila para almacenar datos de forma segura manteniendo las direcciones de retorno en las llamadas a procedimiento e interrupciones así como guardar los parámetros pasados a estos a estos procedimientos, esta pila es de tamaño arbitrario .

```
# bde - changed 0xff00 to 0x4000 to use debugger at 0x6400 up (bde). We
# wouldn't have to worry about this if we checked the top of memory. Also
# my BIOS can be configured to put the wini drive tables in high memory
# instead of in the vector table. The old stack might have clobbered the
# drive table.
```

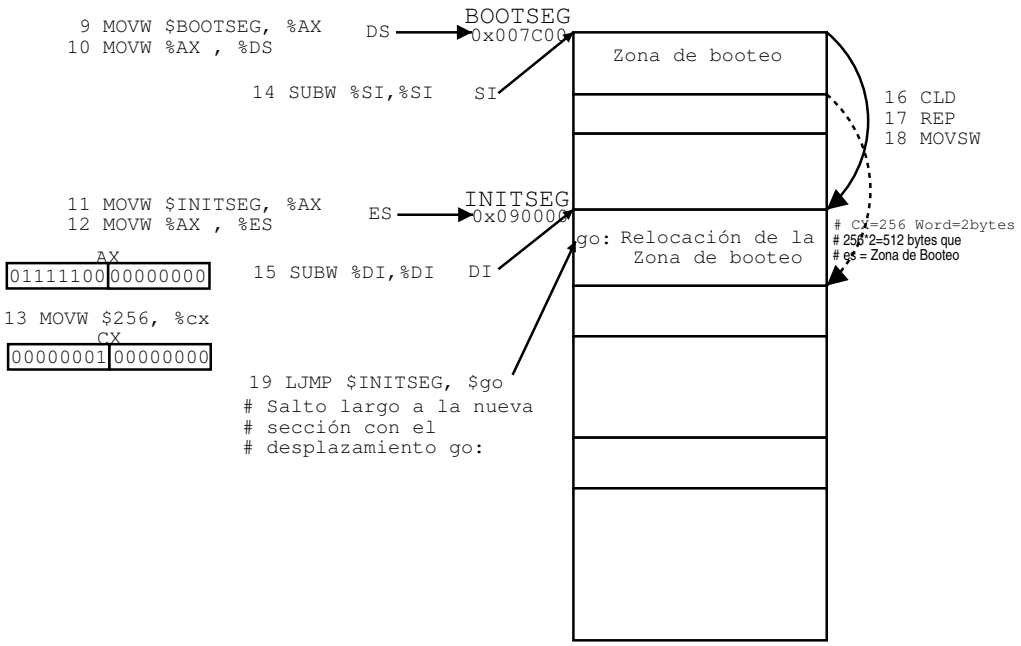


Figura 1.3: Relocación de la zona de booteo y el salto a él.

```
21   go:   movw   $0x4000-12,%di   # 0x4000 is an arbitrary value >=
22                                 # length of bootsect + length of
23                                 # setup + room for stack;
24                                 # 12 is disk parm size.
25         movw   %ax,%ds          #%ax and%es already contain INITSEG
26         movw   %ax,%ss
27         movw   %di,%sp          # put stack at INITSEG:0x4000-12.
```

En la línea 21 se aprovecha la base del registro ES que tiene por referencia el registro DI para hacer un desplazamiento de 0x4000-12=0x3FF4 ya que queremos determinar una pila de base en el segmento 0x090000 que es el INITSEG=0x90000 y un tope de 0x093FF4.

El tamaño de la pila es arbitrario pero mayor o igual a los 512 bytes de la relocación del segmento de *booteo* más la longitud de setup.S que es otro programa que más adelante explicaré. Los 12 bytes que se le restan es para hacer una reserva para la tabla de parámetros del disco, esto es para relocar esta tabla de parámetros, la razón de la relocación de la tabla es para evitar que el diskette fuese leído enteramente, cada sector separadamente. Esto lo haría muy lento. Gracias a esto solo se cambian los parámetros en los discos (nombrando el máximo número de sectores) y se puede leer múltiples sectores a la vez. Otra ventaja es que se puede determinar el número máximo de sectores del disco, de forma ilustrativa en la Figura 1.2.3 se muestra el procedimiento de establecimiento de la pila.

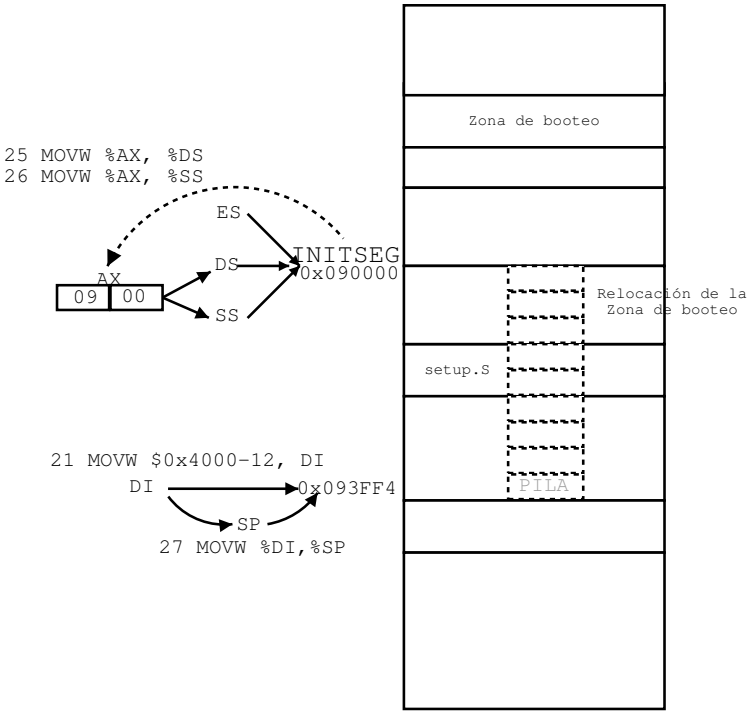


Figura 1.4: Establecimiento de la pila.

```
# Many BIOS's default disk parameter tables will not recognize
# multi-sector reads beyond the maximum sector number specified
# in the default diskette parameter tables - this may mean 7
# sectors in some cases.
#
# Since single sector reads are slow and out of the question,
# we must take care of this by creating new parameter tables
# (for the first disk) in RAM. We will set the maximum sector
# count to 36 - the most we will encounter on an ED 2.88.
#
# High doesn't hurt. Low does.
# Segments are as follows: %cs = %ds = %es = %ss = INITSEG, %fs = 0,
# and %gs is unused.
28   movw    %cx, %fs          # %fs = 0
29   movw    $0x78, %bx        # %fs:%bx is parameter table address
30   pushw   %ds
31   ldsw    %fs:(%bx), %si    # %ds:%si is source
32   movb    $6, %cl           # copy 12 bytes
33   pushw   %di               # %di = 0x4000-12.
34   rep                       # don't worry about cld
35   movsw                     # already done above
36   popw    %di
37   popw    %ds
38   movb    $36, 0x4(%di)     # patch sector count
39   movw    %di, %fs:(%bx)
40   movw    %es, %fs:2(%bx)
```

La determinación de los parametros de disco viene dado por la copia de estos de la manera indicada en las líneas del 28 al 30 que ilustramos en la Figura 1.5

Una vez copiado los parametros del disco devolvemos las direcciones almacenadas en la pila a su estado normal con esto culminamos las líneas del 36-40 del código fuente ilustrándolo en la Figura 1.2.3

1.2.4. Paso 4: Determinación de la cantidad de sectores del disco y exhibición del mensaje Loading

Ahora con los datos de los parametros de disco relocados, procederemos a leer los sectores del disco. Pero la BIOS no tiene una llama para obtener el número de sectores del disco por lo tanto intentaremos a leer el sector máximo de cada tipo de disco de acuerdo al siguiente cuadro.

Probaremos leyendo en 36avo sector si no se puede probaremos con el 18avo sector y así sucesivamente para llegar a ultima instancia al 9no sector.

```
# Get disk drive parameters, specifically number of sectors/track.

# It seems that there is no BIOS call to get the number of sectors.
# Guess 36 sectors if sector 36 can be read, 18 sectors if sector 18
# can be read, 15 if sector 15 can be read. Otherwise guess 9.
# Note that
```

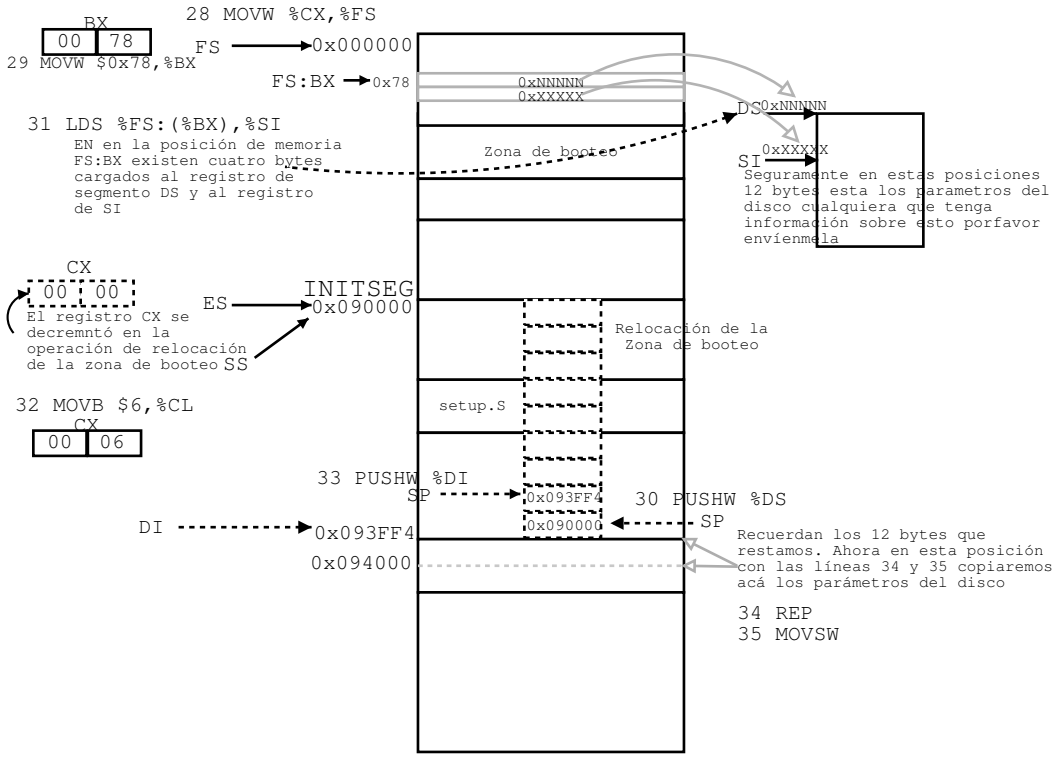


Figura 1.5: Copiado de los parametros del disco.

Capacidad	Pistas por Lado (Cilindros)	Sectores por pista	Bytes por sector	Total 2 lados	Sectores por Grupo
5.25¨ – 360KB	40	9	512	368.640	2
5.25¨ – 1.2MB	80	15	512	1,228.800	1
3.5¨ – 720KB	80	9	512	737.280	2
3.5¨ – 1.44MB	80	18	512	1,474.560	1
3.5¨ – 2.88MB	80	36	512	2,949.120	–

Cuadro 1.1: Capacidades de almacenamiento en discos flexibles.

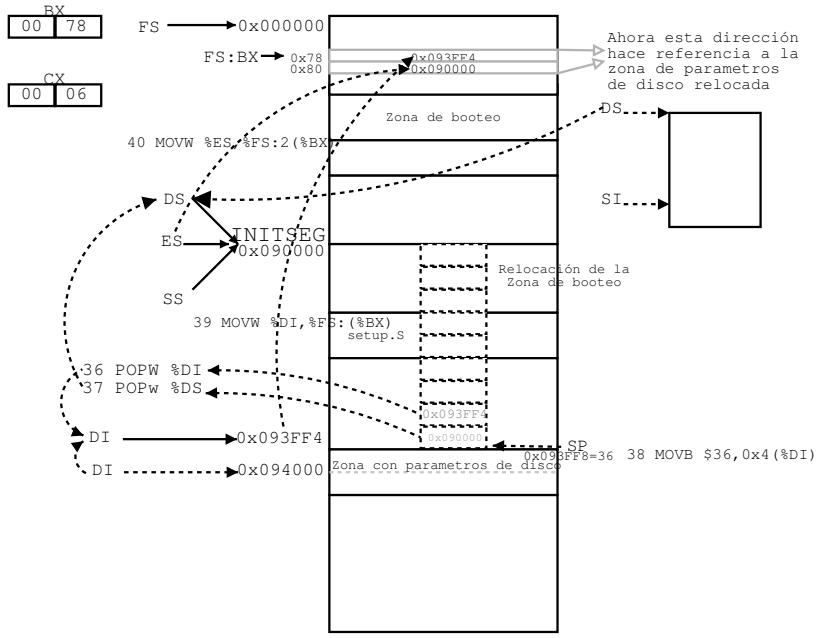


Figura 1.6: Reposición de datos y parchado de la zona de parametros

```
41              movw    $disksizes,%si      #table of sizes to try
42  probe_loop:
43              lodsb
44              cbtw                        #extend to word
45              movw    %ax, sectors
46              cmpw    $disksizes+4,%si
47              jae     got_sectors         #If all else fails, try 9
48
49              xchgw   %cx,%ax             #%cx = track and sector
50              xorw    %dx,%dx             #drive 0, head 0
51              movw    $0x0200,%bx         #address=512,in INITSEG(%es=%cs)
52              movw    $0x0201,%ax         #service 2, 1 sector
53              int     $0x13
54              jc      probe_loop          # try next value
```

```
55  got_sectors:
56                  movb   $0x03,%ah     # read cursor pos
57                  xorb   %bh,%bh
58                  int    $0x10
59                  movw   $9,%cx
60                  movb   $0x07,%bl     # page 0, attribute 7 (normal)
61                                       #%bh is set above; int10 doesn't
62                                       # modify it
63                  movw   $msg1,%bp
64                  movw   $0x1301,%ax   # write string, move cursor
65                  int    $0x10         # tell the user we're loading..
```

En la linea 41 se inicializa la dirección de SI al vector de datos *disksizes*, este vector esta declarado e inicializado con los respectivos sectores ya vistos en el cuadro 1.2.4. Si quiere ubica la declaración está casi al final del código fuente de la siguiente manera.

```
disksizes:   .byte   36, 18, 15, 9
```

Lo mismo sucede con la referencia a la variable *sectors* de la linea 45 y con *msg1* de la linea 63 ambos están declarados en la parte final del código fuente de *bootsect.S* de la siguiente manera.

```
sectors:   .word    0
msg1:      .byte    13, 10
           .ascii   "Loading"
```

La ilustración del código para la prueba de sectores se muestra en la Figura 1.2.4

Una vez que se han probado los sectores y se ha logrado éxito se procede a mostrar el mensaje LOADING para indicar que se esta cargando. El código lo tenemos en la Figura 1.2.4.

1.2.5. Paso 6: Carga de los sectores de setup.S

```
# Load the setup-sectors directly after the moved bootblock (at 0x90200).
# We should know the drive geometry to do it, as setup may exceed first
# cylinder (for 9-sector 360K and 720K floppies).
66      movw   $0x0001,%ax   # set sread (sector-to-read)
67      movw   $sread,%si    # to 1 as the boot sector has
68      movw   %ax, (%si)    # already been read
69
70      xorw   %ax,%ax       # reset FDC
71      xorb   %dl,%dl
72      int    $0x13
73      movw   $0x0200,%bx   # address = 512, in INITSEG
```

En el disco el sector 1 fue ocupado por la instalación del *bootsect* [2] y existe una variable llamada *sread* donde se colocan los sectores leídos y como el sector de booteo del disco ya ha sido leído se coloca 1 la variable *sread*, esta variable está declarada varias líneas abajo de la siguiente manera:

```
sread:   .word   0   # sectors read of current track
```

[2] Recuerdan el dd if=bootsect.bin of=/dev/fd0

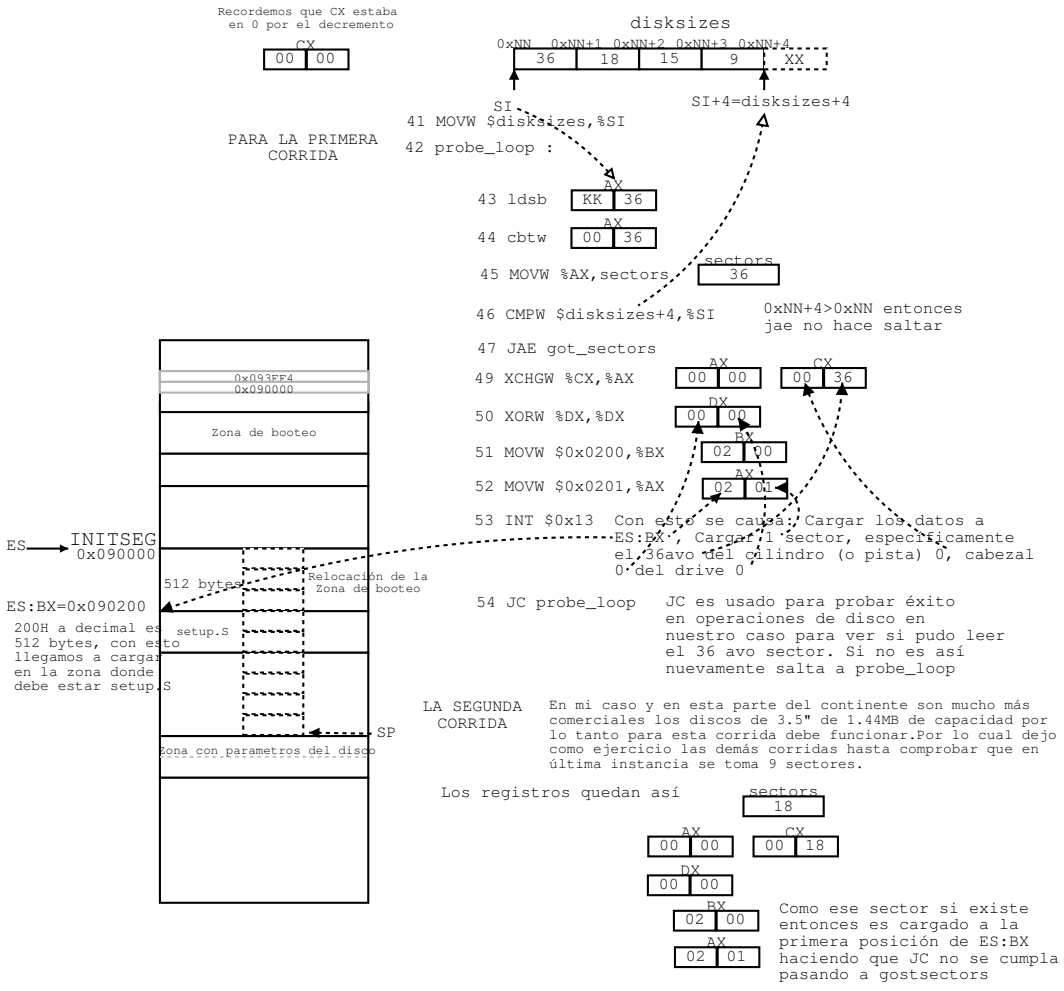


Figura 1.7: Prueba de los diferentes tamaños de sectores de disco

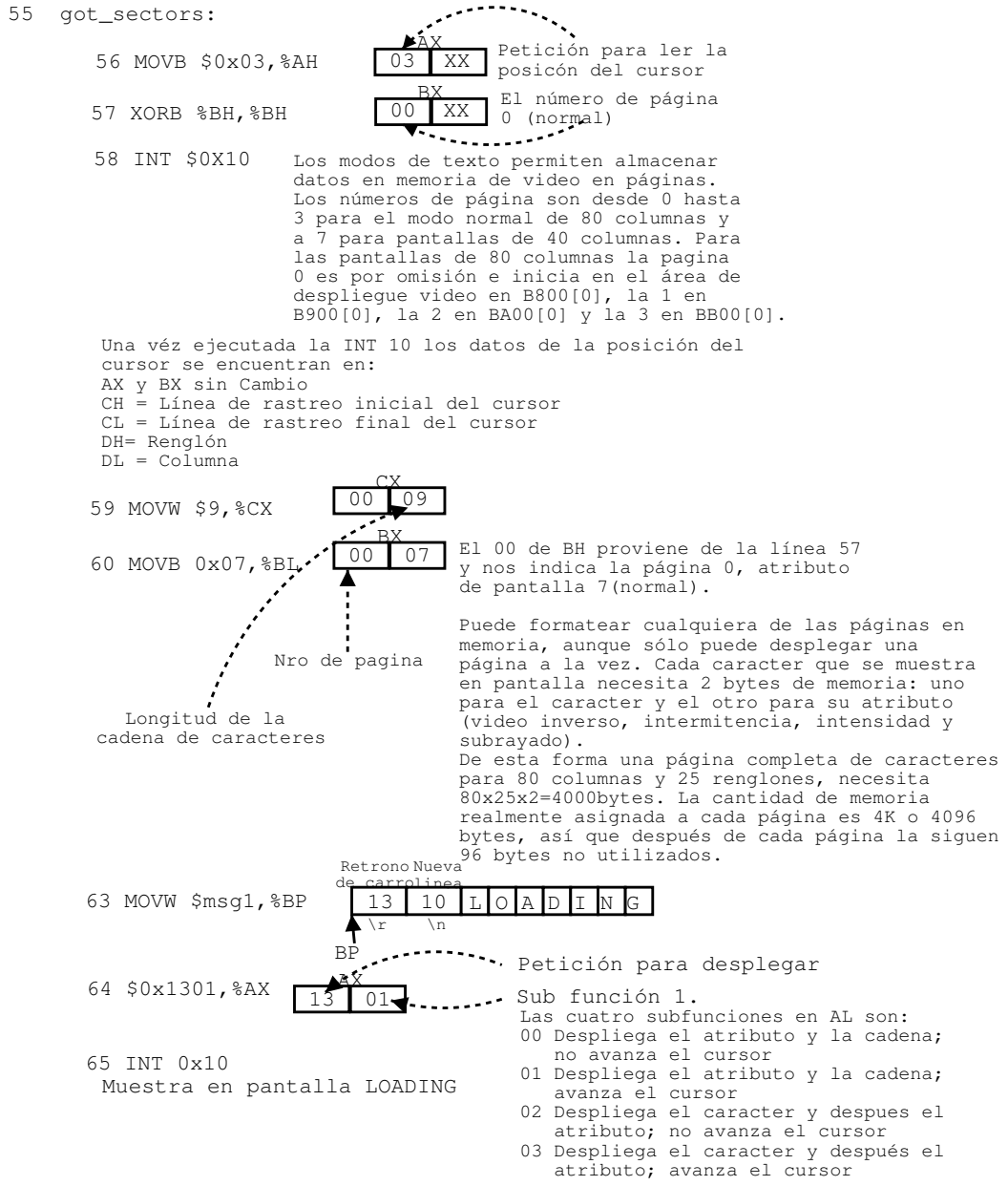


Figura 1.8: Mostrando Loading.

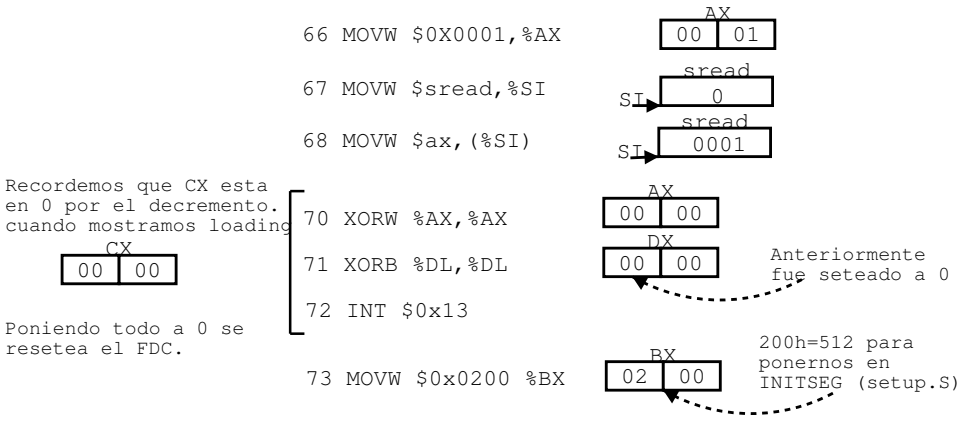


Figura 1.9: Poniendo 1 como sector leído y reseteando el FDC.

Luego se procede a resetear el floppy disk controller (FDC). Estos pasos se ilustran en la figura 1.2.5

```
74  next_step:
75                      movb    setup_sects,%al
76                      movw    sectors,%cx
77                      subw    (%si),%cx         # (%si) = sread
78                      cmpb    %cl,%al
79                      jbe     no_cyl_crossing
80                      movw    sectors,%ax
81                      subw    (%si),%ax         # (%si) = sread
82  no_cyl_crossing:
83                      call    read_track
84                      pushw   %ax               #save it
85                      call    set_next          #set %bx properly;it uses %ax %cx %dx
86                      popw    %ax               #restore
87                      subb    %al, setup_sects  #rest - for next step
88                      jnz     next_step
89
90                      pushw   $SYSSEG
91                      popw    %es               #%es = SYSSEG
92                      call    read_it
93                      call    kill_motor
94                      call    print_nl
```

A continuación se procede a cargar los sectores de *setup.S* definidos en SETUPSECTS a la posición de memoria definida por SETUPSEG. Esto se ilustra en la Figura 1.2.5.

La llamada a set_next se muestra en la figura 1.2.5

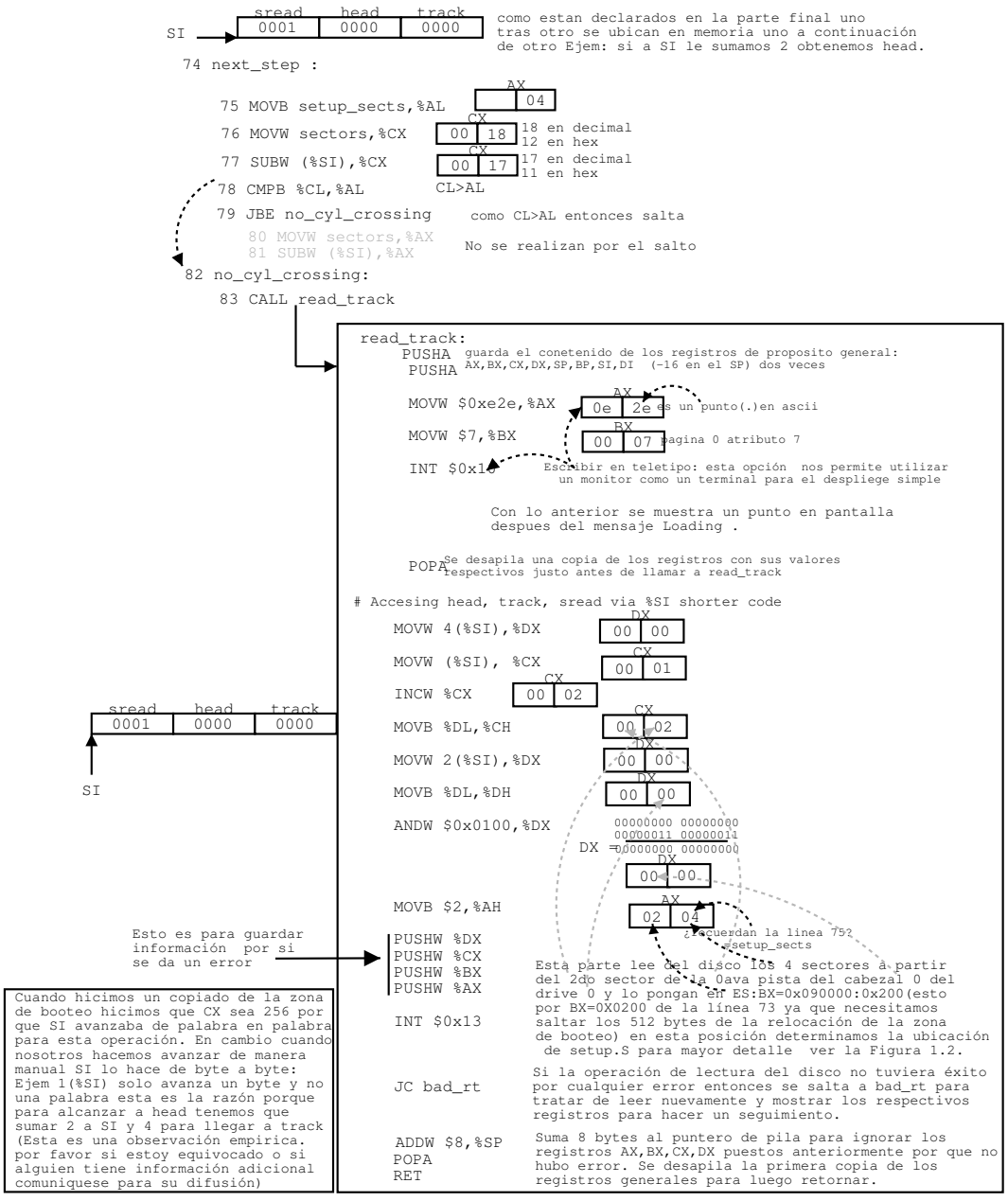
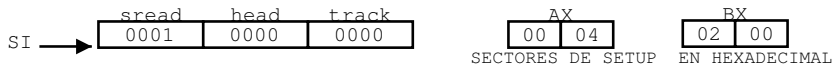
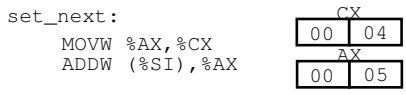


Figura 1.10: Cargando los 4 sectores de setup.S definidos por SETUPSEG.

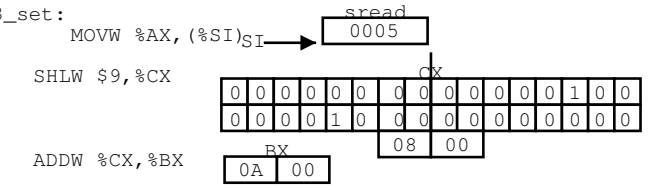


Con el POPA de la fihura anterior recuperamos el estado de
todos los registros de estado

```
84 PUSHW %AX          almacenamos el valor de AX
85 CALL set_next
```

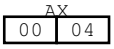



```
CMP sectors,%AX      18>05
JNE ok3_set          Salta por que no son iguales
```

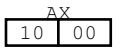



```
JNC            Como no hubo acarreo entonces salta
                        a set_next_fin.

      set_next_fin:
                                    ret     Sin hacer nada retorna.
```

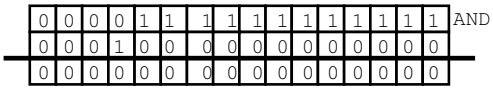
```
86 POPW %AX     Recupera el valor anterior de AX
```




```
87 SUBB %AL,setup_secta          setup_secta=0

88 JNZ NEXT_STEP      Como la operación anterior dio 0 entonces no salta

89 PUSHW $SYSSEG
90 POPW  %ES          Asigna a ES el valor de SYSSEG    ES=0x1000

91 CALL read_it

          read_it:    Esta rutina carga el sistema en la dirección dada por SYSSEG,
                      asegurandose que no sea traspasado el límite de los 64kb. Trataremos
                      de cargarlo tan rápido como sea posible, cargando enteramente las
                      pistas siempre en cuando se pueda.
                      (Traducción del comentarios del código fuente de esta sección)

                      MOVW %ES,%AX
```




```
                      TESTW $0x0FFF,%AX
```




```
           die : JNE  die       Si la operación anterior nos ubiese dado otro
                 XORW %BX,%BX   resultado se ubiera caido en este bucle.
```

Figura 1.11: set_next (Continua en la figura **??**)

Declinación de responsabilidad

Lamento mucho que alguna de las imprecisiones que habrás encontrado te haya causado problemas, pero sencillamente no puedes culparme. Legalmente hablando, no me responsabilizo de una sola de las palabras de esta tentativa de Libro.