Proceso de Arranque del Minix

Fco Javier Arbelo Robaina
Alexis Quintero Jimenez

Índice

1.- Introducción..... 1
2.- Arranque del PC..... 1
3.- Proceso de arranque del Minix..... 2
4.- Explicación del fichero ‘masterboot.s’..... 3
4.1.- Pseudo código..... 3
4.2.- Programa en ensamblador..... 4
5.- Explicación del fichero ‘bootblock.s’..... 8
5.1.- Pseudo código..... 8
5.2.- Programa en ensamblador..... 8

1.- Introducción

A lo largo de esta memoria se explicará todo el proceso que se realiza desde que se enciende el control hasta que se le cede el control al Sistema Operativo (en el caso concreto del Minix, primero se carga el Monitor).

En el primer apartado se explicará que pasos realiza la BIOS para encontrar la unidad de arranque, que estructuras emplea y una breve explicación de las mismas, prosiguiendo con una explicación en pseudo código de los dos programas que intervienen en el arranque, para, finalmente, comentar un poco su código en ensamblador.

2.- Arranque del PC

Una vez encendido el PC, lo primero que realiza la BIOS (*Basic Input Output System*) son una serie de test para almacenar la información de todos los dispositivos instalados (cantidad de memoria RAM, identificación de discos duros o CD-ROM por ejemplo). Una vez hecho esto, la BIOS escoge la primera unidad de arranque (se puede especificar desde que unidad se puede arrancar el Sistema Operativo, este es un parámetro que es configurable en la propia BIOS y como tal se puede cambiar), comprueba que realmente se trata de una unidad de arranque (este proceso se comentará cuando se vea la tabla 1); en el caso que no lo sea, se prosigue con la siguiente unidad definida como arranque en la BIOS y así sucesivamente hasta encontrar una que se pueda arrancar. Si ninguna de las unidades es de arranque, lo más normal es que aparezca un mensaje en pantalla indicando que no se ha encontrado ninguna unidad de arranque, en tal caso se debe resetear la máquina y elegir correctamente la/s unidad/es de arranque.

Suponiendo que se ha encontrado una unidad válida de arranque, el siguiente paso que realiza la BIOS es copiar el primer sector de la unidad (si se trata de un disco duro los parámetros del primer sector de la unidad son cabeza 0, cilindro 0 y sector 0) en la posición absoluta de memoria 0x7C00. La estructura del primer sector físico de una unidad se define en la tabla 1.

	Contenido
0x0000	Código de arranque (masterboot.s)
0x01BE	1ª entrada en la tabla de particiones
0x01CE	2ª entrada en la tabla de particiones
0x01DE	3ª entrada en la tabla de particiones
0x01EE	4ª entrada en la tabla de particiones
0x01FE	Código de arranque: 0xAA55

Tabla 1: Primer sector de un disco duro.

La tabla anterior muestra varios campos interesantes. El primero de ellos es el encargado de alojar el código de arranque del Sistema Operativo en cuestión; el correspondiente aparece en el fichero masterboot.s (se comentará en próximos apartados). Las siguientes cuatro entradas definen las características del disco duro (en

Minix un floppy puede tener esta estructura, esto es, se pueden realizar particiones dentro de un floppy) y la estructura que la forman se muestra en la tabla 2. El último campo indica si la unidad es de arranque o no: si la última palabra de 16 bits contiene 0xAA55, la unidad es de arranque, cualquier otro valor nos dice que no es de arranque.

Dirección	Tipo	Contenido
0x00	1 Byte	Estado de la partición: 0x00: Inactiva 0x80: Activa
0x01	1 Byte	Cabeza lectura/escritura donde comienza la partición
0x02	1 Word	Sector y cilindro con el que comienza la partición
0x04	1 Byte	Tipo de partición
0x05	1 Byte	Cabeza de lectura/escritura con el que termina la partición
0x06	1 Word	Sector y cilindro con el que termina la partición
0x08	1 DWord	Distancia del primer sector de la partición (sector de arranque) del sector de particiones (en sectores)
0x0C	1 DWord	Número de sectores en esta partición

Tabla 2: Estructura de la tabla de particiones.

3.- Proceso de arranque del Minix

Como todo proceso de arranque, consta de varias partes (puede verse de forma simplificada en la figura 1). Una característica de Minix es que no carga directamente el Sistema Operativo, sino un 'boot secundario' (el monitor, desde el cual se puede modificar el comportamiento del arranque del sistema: elegir el arranque entre varios Sistemas Operativos, por ejemplo).

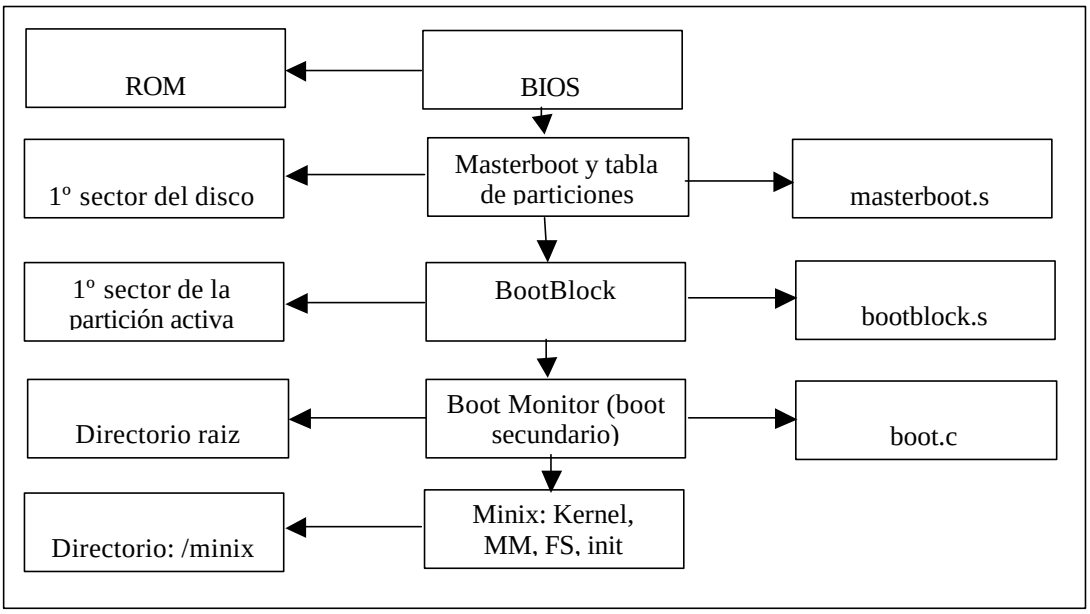

Figura 1: Proceso de arranque del Minix.

Otro punto que hay que tener claro es el por que de dos programas en el arranque (masterboot.s y bootblock.s), la razón es sencilla: el masterboot se encuentra en el primer sector de la partición de arranque (en la tabla 1 se identifica con el código de arranque) y solo tiene sentido cuando se trata de seleccionar cual de las cuatro particiones posibles es la de arranque (una característica del sistema de ficheros Minix es que permite realizar particiones en los floppy's, de esta manera se comporta igual que un disco duro y por tanto tendrá esta estructura). El otro fichero, bootblock.s, se instala en el primer sector de la partición, y es el que propiamente carga el Sistema Operativo (en realidad carga el monitor). En un floppy sin particiones, este código es el que se encuentra en el primer sector.

4.- Explicación del fichero 'masterboot.s'

En un primer apartado se explicará en pseudo código como se comporta el programa *masterboot.s*, en el segundo apartado se verá el código en ensamblador.

4.1.- Pseudo código

Una vez cargado el primer sector en la posición 0x7C00 (la BIOS es quien lo carga), los pasos que realiza el programa son:

1.- Copiar el código cargado de la posición 0x7C00 a la posición 0x0600 y saltar a esta última posición (lo hace por seguridad).

2.- Se verifica si se ha pulsado la tecla ALT. Se da la posibilidad de arrancar desde otra unidad.

2.1.- Si está pulsada la tecla ALT, se espera a que se pulse un número de 0-9, que indica el disco desde el cual se va a proceder la carga del Sistema Operativo.

3.- Una vez se sabe el disco desde el cual se va arrancar (ya sea la unidad por defecto o por que el usuario ha elegido otra), se realizan los siguientes pasos:

3.1.- Se leen las características de la unidad elegida (cabeza, cilindro y sector si se trata de un disco duro).

3.2.- Se carga el primer sector físico de la unidad.

3.2.1.- Si existe error, se reintenta la lectura del primer sector 3 veces. Si falla, se muestra un mensaje en pantalla: 'Error de lectura' y se queda en un bucle infinito.

3.2.2.- Si la lectura tiene éxito, el siguiente paso consiste en comprobar si es de arranque:

a) Si no está la firma de arranque, se muestra un mensaje en pantalla: ‘Unidad no iniciable’ y se queda en un bucle infinito.

b) Si está la firma, se continua con el programa.

4.- Se salta al código cargado (*bootstrap*).

4.2.- Programa en ensamblador

```
BUFFER=0x0600      ; Comienzo de la memoria libre
PART_TABLE=446 ;Localización de la tabla de partición dentro de este código
PENTRYSIZE=16      ;Tamaño de una entrada de la tabla de partición
MAGIC=510       ; Localización del número mágico AA55

; <ibm/partition.h>:
bootind    =        0
sysind     =        4
lowsec     =        8

.define begtext, begdata, begbss, endtext, enddata, endbss, _main
.data
begdata:
.bss
begbss:
.text
begtext:
_main:

;Encontrar (sub)partición activa, cargar su primer sector, ejecutarlo.
master:

       jmp     over
fix:   .data1  0  ;Si  1-9 entonces siempre arrancar ese dispositivo
over:
       xor     ax, ax
       mov     ds, ax
       mov     es, ax
       cli
       mov     ss, ax                ; ds = es = ss = Vector segmento
       mov     sp, #LOADOFF
       sti
       mov     bp, #BUFFER+PART_TABLE; Dirección usada a menudo

; Copia este código por seguridad, luego saltar a él.
       mov     si, sp                ; si = comienzo de este código
       push    si                    ; también su dirección de retorno
       mov     di, #BUFFER           ; Area del buffer
       mov     cx, #512/2            ; Un sector
       cld
       rep
       movs
       jmpf    BUFFER+migrate, 0     ; Por seguridad
migrate:

; ¿ Tecla ALT pulsada para ignorar el arranque del dispositivo activo ?
key:
       movb    ah, #0x02             ;   Desplazamiento   del   estado   del
teclado
       int     0x16
       testb   al, #0x08             ; Bit 3 =  tecla ALT
       jz      noalt                 ; No pulsada la tecla  ALT
       call    print
       .data2  BUFFER+devhd

getkey:xorb    ah, ah                ; Esperar a pulsar una tecla
       int     0x16
       movb    BUFFER+choice, al
       subb    al, #0x30             ; al -= '0'
```

```
        cmpb    al, #10
        jae     getkey          ; La tecla no está en el rango 0 - 9
        push    ax
        call    print                   ; Mostrar la tecla pulsada
        .data2  BUFFER+choice
        pop     ax
        jmp     override

noalt:
        movb    al, BUFFER+fix ;¿Siempre arranca una cierta partición?
        testb   al, al
        jz      findactive              ; No, arranca la partición activa

override:
        cbw                             ; ax = partición elegida
        movb    dl, #5
        divb    dl                      ; al = disco, ah = partición dentro de
disco
        movb    dl, #0x80
        addb    dl, al                  ; dl = disco
        movb    al, ah                  ; al = partición dentro de disco
        push    ax                      ; Grabar partición elegida
        call    load0                   ; Coger el sector 0
        jb      error0                  ; Deshabilitar su lectura
        pop     ax                      ; Restablecer la partición elegida
        subb    al, #1                  ; ¿ Fue 0 modulo 5?
        jl      bootstrap               ; Saltar al master bootstrap
        mov     si, #LOADOFF+PART_TABLE       ; si = nueva tabla de partición
        mov     di, bp                  ; Para area de buffer
        mov     cx, #4*PENTRYSIZE/2
        rep
        movs
        addb    cl, #4          ; Cuatro veces es suficiente para ordenar

sort:   mov     si, bp                  ; Primera entrada de la tabla

bubble: lea     di, PENTRYSIZE(si)      ; Siguiente entrada
        cmpb    sysind(si), ch          ; Tipo de partición, en uso distinta
de cero
        jz      exchg                         ; Entradas no usadas ordenar al
final

inuse:  mov     bx, lowsec+0(di)
        sub     bx, lowsec+0(si)        ; Realizar di->lowsec - si->lowsec
        mov     bx, lowsec+2(di)
        sbb     bx, lowsec+2(si)
        jnb     order                   ; En orden si si->lowsec <= di-
>lowsec

exchg:  movb    ah, (si)
        xchgb   ah, PENTRYSIZE(si)      ; Intercambiar entradas byte a byte
        movb    (si), ah
        inc     si
        cmp     si, di
        jb      exchg

order:  mov     si, di
        cmp     si, #BUFFER+PART_TABLE+3*PENTRYSIZE
        jb      bubble
        loop    sort
        mov     si, bp                  ; si = tabla ordenada
        movb    ah, #PENTRYSIZE
        mulb    ah                      ; ax = al * PENTRYSIZE
        add     si, ax                  ;si = dirección de la entrada de la

        cmpb    sysind(si), #1          ; Debería estar en uso
        jb      error0
        jmp     loadpart                          ; Coger la partición del
bootstrap

; Encontrar la partición activa.
findactive:
        testb   dl, dl
        jge     nextdisk ;No particiones en floppys
        mov     si, bp

find:   cmpb    sysind(si),#0 ;Tipo de partición, en uso distinta de cero
```

```
        jz      nextpart
        testb   bootind(si), #0x80    ; Flag de la partición activa en bit 7
        jz      nextpart              ; No está activa
loadpart:
        call    load                  ; Cargar la partición del bootstrap

error0: jb      error1                ; No supuso fallo

bootstrap:
        ret                           ; Saltar al master bootstrap

nextpart:
        add     si, #PENTRYSIZE
        cmp     si, #BUFFER+PART_TABLE+4*PENTRYSIZE
        jb      find

; No partición activa, dimelo
        call    print
        .data2  BUFFER+noactive

; No hay particiones activas en esta unidad, intentar la próxima unidad.
nextdisk:
        incb    dl                    ;  Incrementar  dl  para  la  próxima
unidad
        testb   dl, dl
        jl      nexthd                ; Disco duro si negativo
        int     0x11                  ; Coger la configuración del equipo
        shl     ax, #1                                ; Número de unidad de
floppy más alto en los bist 6-7
        shl     ax, #1                ; Ahora en los bits 0-1 de ah
        andb    ah, #0x03             ; Extraer bits
        cmpb    dl, ah                ; Debe ser dl <= ah para que la unidad
exista
        ja      nextdisk              ;   En   otro   caso   intentar   hd0
eventualmente
        call    load0                 ; Leer el próximo floppy del bootstrap
        jb      nextdisk              ; Si falló, próximo disco por favor
        ret                           ; Saltar al próximo master bootstrap

nexthd: call    load0                      ;  Leer  el  bootstrap  del  disco
duro

error1: jb      error                 ; ¿ No disco?
        ret

; Cargar sector 0 desde el dispositivo actual. Es o el floppy del bootstrap
o un ;disco duro del master bootstrap.
load0:
        mov     si, bp
        mov     lowsec+0(si), ds      ; Crear una entrada con un lowsec cero
        mov     lowsec+2(si), ds
        !jmp    load

; Cargar el sector lowsec(si) desde el dispositivo actual. Los número de
cabeza, sector y ;cilindro son ignorados para favorecer de manera absoluta
el comienzo de la  particón.
load:
        mov     di, #3         ; Tres reintentos para comprobar si hay floppy

retry:  push    dx             ; Grabar código de la unidad
        push    es
        push    di             ;  La próxima llamada destruye es y di
        movb    ah, #0x08      ; Código para los parámetros de la unidad
        int     0x13
        pop     di
        pop     es
        andb    cl, #0x3F      ; cl = número máximo de sector (1-origen)
        incb    dh             ; dh = 1 + número de cabeza máximo (0-origen)
        movb    al, cl         ; al = cl = sectores por pista
        mulb    dh             ; dh = cabezas, ax = cabezas * sectores
        mov     bx, ax         ;  bx  =  sectores  por  cilindro  =  cabezas  *
sectores
        mov     ax, lowsec+0(si)
        mov     dx, lowsec+2(si); dx:ax = sector dentro de la unidad
        div     bx             ;  ax  =  cilindro,  dx  =  sector  dentro  de
cilindro
```

```
        xchg    ax, dx          ;  ax  =  sector  dentro  de  cilindro,  dx  =
cilindro
        movb    ch, dl          ; ch = 8 bits inferiores de cilindro
        divb    cl              ; al = cabeza, ah = sector (0-origen)
        xorb    dl, dl          ; Desplazamiento de los bits 8-9 del cilindro
en dl
        shr     dx, #1
        shr     dx, #1          ; dl[6..7] = cilindro superior
        orb     dl, ah          ; dl[0..5] = sector (0-origen)
        movb    cl, dl          ;  cl[0..5]  =  sector,  cl[6..7]  =  cilindro
superior
        incb    cl              ; cl[0..5] = sector (1-origen)
        pop     dx              ; Restablecer código de la unidad en dl
        movb    dh, al          ; dh = al = cabeza
        mov     bx, #LOADOFF    ; es:bx = donde se carga el sector
        mov     ax, #0x0201     ; Código para leer, unicamente un sector
        int     0x13            ; Llamar a la BIOS para una lectura
        jnb     ok              ; Lectura satisfactoria
        cmpb    ah, #0x80       ; ¿ Timed out del disco?  (Floppy vacío)
        je      bad
        dec     di
        jl      bad             ; Recuperar contador expirado
        xorb    ah, ah
        int     0x13            ; Resetear
        jnb     retry           ; Intentar de nuevo

bad:    stc                     ; Poner bandera de acarreo
        ret

ok:     cmp     LOADOFF+MAGIC, #0xAA55
        jne     nosig           ; Error si “firma” erronea
        ret                     ;  Retornar con acarreo limpio

nosig:  call    print
        .data2  BUFFER+noboot
        jmp     hang

; Ha ocurrido error en una lectura, reclamar y bucle infinito.
error:
        call    print
        .data2  BUFFER+readerr


; Bucle infinito esperando por CTRL-ALT-DEL
hang:   jmp     hang

print:  pop     si                      ; dirección de retorno
        lods                            ;ax = *si++ = palabra después de'call
print'
        push    si                      ; nueva dirección de retorno
        mov     si, ax

prnext: lodsb                           ; al = *si++ es el carácter a imprimir
        testb   al, al
        jz      prdone                  ; Nulo marca fin
        movb    ah, #14                      ; 14 = imprimir caracter
        mov     bx, #0x0001             ; Página 0, color de primer plano
        int     0x10                         ; Llamar a la  BIOS VIDEO_IO (E/S
de la BIOS)
        jmp     prnext

prdone: ret

.data
devhd:          .ascii  "/dev/hd?\b"
choice:         .ascii  "\0\r\n\0"
noactive:       .ascii  "None active\r\n\0"
readerr:        .ascii  "Read error \0"
noboot:         .ascii  "Not bootable \0"
.text
endtext:
.data
enddata:
.bss
endbss:
```

5.- Explicación del fichero ‘bootblock.s’

5.1.- Pseudo código

1.- Creación del entorno de trabajo: inicializa ds=ss=ax=0, sp=bp=0x7C00.

2.- Se comprueba si se va a cargar de disco duro o de floppy.

> 2.1.- Si es un disco duro, se obtienen los parámetros de la unidad y se salta a cargar el boot secundario.
>
> 2.2.- Si es un floppy, hay que determinar que tipo de unidad se trata. El proceso que se sigue es sencillo, se tiene una variable que contiene los parámetros de las distintas unidades posibles (3.5’’ alta densidad, 3.5’’ baja densidad, 5.25’’ alta densidad y 5.25’’ baja densidad) y se lee el último sector de la primera pista, si falla la prueba se sigue con la siguiente unidad y así sucesivamente. Cuando se determina el tipo de unidad, se salta a cargar el boot secundario (monitor).

3.- El monitor se empieza a cargar en la posición 0x1000:0x0000. Se entra en un proceso iterativo: Mientras queden sectores por leer del monitor:

> 3.1.- Cargar el sector especificado en la posición de memoria es:bx.
>
> 3.2.- Se modifica la próxima posición de memoria: es:bx+512
>
> 3.3.- Si existe error, mostrarlo en pantalla y quedarse en un bucle infinito.

4.- Cuando están todos los sectores del monitor cargados en memoria, se salta a la posición de memoria adecuada (0x1000:0x0000) para ceder el control al monitor.

5.2.- Programa en ensamblador

```
LOADOFF=0x7C00   ; 0x0000:LOADOFF donde se encuentra este código
BOOTSEG=0x1000   ; Segmento de código del boot secundario
BOOTOFF =0x0030   ; Desplazamiento en el boot secundario por encima de la
cabeza
BUFFER= 0x0600    ; Comienzo de la memoria libre
DSKBASE=0x1E     ; Vector de parámetros del floppy
DSKPARSIZE=11    ; 11 bytes de parámetros del floppy
SECTORS=4          ; Desplazamiento en los parámetros de los sectores por
pista
LOWSEC=8        ; Desplazamiento del primer sector lógico en la tabla de
                                          ; particiones


; Variables direccionadas usando el registro bp
device     =         0 ; El dispositivo de arranque
lowsec     =         2 ; Desplazamiento de la partición dentro de la unidad
secpcyl    =         6 ; Sectores por cilindro = cabezas * sectores

.define begtext, begdata, begbss, endtext, enddata, endbss, _main
.data
begdata:
```

```
.bss
begbss:
.text
begtext:
_main:

; Comienzo del procedimiento de arranque
boot:
       xor     ax, ax          ; ax = 0x0000, el vector de segmento
       mov     ds, ax
       cli                     ;   Desactiva   las   interrupciones   mientras
inicializa la pila
       mov     ss, ax          ; ss = ds = vector de segmento
       mov     sp, #LOADOFF    ; Lugar usual para la pila del bootstrap
       sti


            push     ax
       push    ax              ; Pone un cero en lowsec(bp)

       push    dx              ;   Dispositivo   de   arranque   en   dl   será
device(bp)
       mov     bp, sp          ; Usar var(bp) es un byte más económico que
var.

       push    es
       push    si              ; es:si = entrada de la tabla de partición si
disco duro
       mov     di, #LOADOFF+parameters      ; carcáter (*di)[DSKPARSIZE] =
parámetros;

       testb   dl, dl          ; Disco duro si dl >= 0x80
       jge     floppy

winchester:

; Obtiene el desplazamiento del primer sector de la partición de arranque
desde la tabla ;de partición.  La tabla se encuentra en es:si, el parámetro
lowsec en desplazamiento ;LOWSEC.

       eseg
       les     ax, LOWSEC(si)   ; es:ax = LOWSEC+2(si):LOWSEC(si)
       mov     lowsec+0(bp), ax  ; 16 bits inferiores del primer sector de
la partición
       mov     lowsec+2(bp), es  ; 16 bits superiores del primer sector de
la partición


; Obtiene los parámetros de la unidad, el número de sectores es de manera
terminante ;escrito en los parámetros del floppy.

       movb    ah, #0x08       ; Codigo para los parámetros de la unidad
       int     0x13            ; dl aún contiene la unidad
       andb    cl, #0x3F       ; cl = número máximo de sector  (1-origen)
       movb    SECTORS(di), cl       ; Número de sectores por pista
       incb    dh              ; dh = 1 + número máximo de cabeza (0-origen)
       jmp     loadboot

; Floppy:
; Ejecuta 3 tests de lectura para determinar el tipo de unidad. Prueba para
cada tipo de ;floppy mediante la lectura del último sector de la primera
pista. Si esto falla, intenta un ;tipo que tenga menos sectores. Por lo
tanto comenzamos con 1.44M (18 sectores) luego ;con 1.2M (15 sectores) y
finaliza con 720K/360K (ambos con   9 sectores).   (Los ;parámetros del
floppy de los dos últimos son iguales, aparte del tiempo de arranque del
;motor. Esto nos evita el desagradable test “intentar leer pista 41”).

next:  add     di, #DSKPARSIZE       ; Siguiente conjunto de parámetros

floppy: mov    DSKBASE*4+0, di       ;  Carga  el  desplazamiento  de  los
parámetros del disco
       mov     DSKBASE*4+2, ds       ; Carga el segmento de los parámetros
del disco


       xorb    ah, ah          ; Resetea la unidad
```

```
       int    0x13

       movb   cl, SECTORS(di)                ; cl = número   del último
sector de la pista
       cmp    di,  #LOADOFF+dsdd3    ;  No  necesita  hacer  el  último  test
720K/360K
       jz     success


; Intenta leer el último sector de la pista 0

       mov    es, lowsec(bp) ; es = vector segmento (lowsec = 0)
       mov    bx, #BUFFER    ; es:bx buffer = 0x0000:0x0600
       mov    ax, #0x0201    ; Leer sector, número de sectores = 1
       xorb   ch, ch         ; Pista 0, último sector
       xorb   dh, dh         ; Unidad dl, cabeza 0
       int    0x13
       jb     next           ; Error, intentar el próximo tipo de floppy

success:movb   dh, #2         ; Cargar el número de cabezas para multiplicar
                                                    ;   El   número   de
sectores está aún en cl

loadboot:
; Carga el código del boot secundario desde el dispositivo de arranque

       movb   al, cl         ; al = cl = sectores por pista
       mulb   dh             ; dh = cabezas, ax = cabezas * sectores
       mov    secpcyl(bp), ax       ;  Sectores  por  cilindro  =  cabezas  *
sectores

       mov    ax, #BOOTSEG   ; Segmento para cargar dentro el código del
boot secundario       mov    es, ax
       xor    bx, bx         ;   Carga   el   primer   sector   en   es:bx   =
BOOTSEG:0x0000
       mov    si, #LOADOFF+addresses; Comienzo de la dirección del código
del boot
load:
       mov    ax, 1(si)      ; Obtiene el próximo número de sector: 16 bits
inferiores
       movb   dl, 3(si)      ; Bits 16-23 para tu disco de 8GB
       xorb   dh, dh         ; dx:ax = sector dentro de la partición
       add    ax, lowsec+0(bp)

       adc    dx, lowsec+2(bp); dx:ax = sector dentro de la unidad
       div    secpcyl(bp)    ;  ax  =  cilindro,  dx  =  sector  dentro  del
cilindro
       xchg   ax, dx         ;  ax  =  sector  dentro  del  cilindro,  dx  =
cilindro
       movb   ch, dl         ; ch = 8 bits inferiores del cilindro
       divb   SECTORS(di)    ; al = cabeza, ah = sector (0-origen)
       xorb   dl, dl         ; Desplazar los bits 8-9 del  cilindro en dl
       shr    dx, #1
       shr    dx, #1         ; dl[6..7] = cilindro superior
       orb    dl, ah         ; dl[0..5] = sector (0-origen)
       movb   cl, dl         ;  cl[0..5]  =  sector,  cl[6..7]  =  cilindro
superior
       incb   cl             ; cl[0..5] = sector (1-origen)
       movb   dh, al         ; dh = al = cabeza
       movb   dl, device(bp) ; dl = dispositivo para leer
       movb   al, SECTORS(di)       ;  Sectores  por  pista  –  Número  de
sector (0-origen)
       subb   al, ah         ; = Sectores a la izquierda en ésta pista
       cmpb   al, (si)                   ;  Compara  con  el  número  de
sectores a leer
       jbe    read           ;  ¿  No  puede  leer  después  del  final  del
cilindro ?
       movb   al, (si)              ; (si) < sectores a la izquierda en
esta pista
read:  push   ax             ; Guarda  al = sectores a leer
       movb   ah, #2         ; Código para leer disco (¡ todos los registro
se usan ahora ;)
       int    0x13           ; Llamar a la BIOS para una lectura
            pop      cx             ; Restaura al en cl
       jb     error          ; Salta al disco para leer un error
       movb   al, cl         ; Restaura al = sectores leídos
```

```
        addb    bh, al          ; bx += 2 * al * 256 (suma bytes leídos)
        addb    bh, al          ;  es:bx  =donde  debe  ser  leído  el  próximo
sector
        add     1(si), ax                   ;  Actualizar  dirección  mediante

        adcb    3(si), ah                   ;  No  olvidar  los  bits  16-23
(sumar ah = 0)
        subb    (si), al              ;  Decrementar  el  contador  de  sector
por sectores leídos
        jnz     load            ; No todos los sectores han sido leídos
        add     si, #4          ; Siguiente par (dirección , contador)
        cmpb    ah, (si)              ; Hecho cuando no hay sectores a leer
        jnz     load            ; Lee el siguiente trozo del código del boot
secundario

done:

; Llama al boot secundario, asumiendo una cabecera larga a.out (48 bytes).
La  cabecera  ;a.out  es  normalmente  pequeña  (32  bytes),  pero  el  boot
secundario tiene dos puntos de ;entrada: Uno es el desplazamiento 0, para
la  cabecera  larga,  y  el  otro  es  el  ;desplazamiento  16  para  la  cabecera
corta.

; Los parámetros pasados en los registros son:

;       dl      = Dispositivo del boot
;       es:si   =  Entrada de la tabla de partición si es el disco duro.

        pop     si      ;  Restablecer  es:si  =  entrada  en  la  tabla  de
partición
        pop     es      ; dl está aún cargado
        jmpf    BOOTOFF, BOOTSEG    ;Saltar al sector del boot (saltando a la
cabecera).

; Error de lectura: imprimir mensaje, bucle infinito
error:
;mov    si, #LOADOFF+errno+1  ; Sin comentarios esto en tiempo destroso
;prnum: movb    al, ah          ; Número de error en ah
;       andb    al, #0x0F       ; 4 bits inferiores
;       cmpb    al, #10         ; ¿A-F?
;       jb      digit           ; 0-9
;       addb    al, #7          ; 'A' - ':'
;digit: addb    (si), al              ; Modificar '0' en la ristra
;       dec     si
;       movb    cl, #4          ; Próximos 4 bits
;       shrb    ah, cl
;       jnz     prnum           ; De nuevo si digit > 0

        mov     si, #LOADOFF+rderr  ; Ristra a imprimir

print:  lodsb                   ; al = *si++ es el carácter a ser imprimido
        movb    ah, #14 ; 14 = imprimir carácter
        mov     bx, #0x0001     ; Página 0, color de primer plano
        int     0x10            ; Llamar a la  BIOS VIDEO_IO (E/S de la BIOS)
        cmp     si, #LOADOFF+errend  ; ¿  Alcanzado el fin de la ristra ?
        jb      print


; Bucle infinito esperando por CTRL-ALT-DEL
hang:   jmp     hang

.data
rderr:  .ascii  "Read error "
;errno: .ascii  "00 "
errend:

parameters:
; Los parámetros del floppy ordenados de forma decreciente en los sectores
por  pista.  ;(La  longitud  de  los  parámetros  de  los  diskets  de  3.5”  podría
ser erronea, pero eso no ;ocurrirá).

; 1.44M 3.5"
dshd3:  .data1  0xAF, 0x02, 25, 2, 18, 0x1B, 0xFF, 0x54, 0xF6, 15, 8

; 1.2M 5.25"
dshd5:  .data1  0xDF, 0x02, 25, 2, 15, 0x1B, 0xFF, 0x54, 0xF6, 15, 8
```

```
; 720K 3.5", también usado por 360K 5.25"
dsdd3:  .data1  0xDF, 0x02, 25, 2,  9, 0x2A, 0xFF, 0x50, 0xF6, 15, 8

; Unicamente para completar, aquí están los 360K de parámetros reales.
;dsdd5: .data1  0xDF, 0x02, 25, 2,  9, 0x2A, 0xFF, 0x54, 0xF6, 15, 3

.text
endtext:
.data
enddata:
addresses:
;El espacio por debajo de esto es para las direcciones de disco para un
programa de boot ;secundario (el peor de los casos, es decir el fichero
está fragmentado). Debería ser ;suficiente.
.bss
endbss:
```