Implementación de un sistema operativo en modo real

Asignatura “Sistemas Operativos”

Murcia, marzo de 2004

Universidad de Murcia

índice

1. Modo real de los procesadores Intel i-8086
2. ROM BIOS. Detalles de la tabla de vectores y de las instrucciones *int* e *iret*
3. Mapa de memoria de un PC
4. Proceso de arranque de un PC
5. Estructura de un disco 3 ½ 1.44 Mb
   con sistema de archivo FAT 12. Gestión de sectores mediante servicio int 13h de BIOS
6. Detalles del código generado por el compilador TCC
7. Interrupciones del timer y del teclado
8. Ficheros .COM de MS-DOS
9. Elaboración de un sector de arranque

1. Modo real de los procesadores Intel i-8086

DiTEC
Universidad de Murcia

Los registros de segmento se fijan al principio del programa (16 bits)

SSSS

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

BP

SI

DI

SP ← DIRECCIÓN DE PILA (SE COMBINA CON SS)

IP ← DIRECCIÓN DE CÓDIGO (SE COMBINA CON CS)

← 16 →

flags

				O	D	I	T	S	Z		A		P		C

DIRECCIONES DE 16 BITS

CS

DS

SS

ES

CUANDO LA DIRECCIÓN ES DE DATOS SE COMBINA CON DS[1]

$$\left(\text{DIRECCION DE DATOS}\right) \equiv \left[\left\{\begin{matrix}BX\\BP\end{matrix}\right\} + \left\{\begin{matrix}SI\\DI\end{matrix}\right\} + \{Desp\}\right]$$

DIRECCIONES DE 20 BITS

$$\begin{array}{r} S\,S\,S\,S\,0 \\ +\quad D\,D\,D\,D \\ \hline X\,X\,X\,X\,X \end{array}$$

Fuera de la cpu es un procesador con un bus de direcciones de 20 bits

Dirección real de memoria

Dentro de la cpu las direcciones generadas (de código, de datos y de pila) son de 16 bits

DDDD

CPU i8086

2^{20} palabras de 8 bits

[1] Si aparece el registro BP se combina con SS

2. ROM BIOS

Detalles de la tabla de vectores y de las instrucciones *int* e *iret*

Ideas claves

- Memoria ROM

  Es necesaria ya que al encender el ordenador ha de haber algún código en memoria
- Disco

  Lo único que se puede hacer con un disco es traer un sector a memoria o llevar 512 bytes de memoria a un sector. Nada más
- Arranque de ordenador

  El código en ROM traerá bytes del disco a una posición de ram para ejecutarlos a continuación

Código en ROM

- ROM de arranque
- ROM BIOS

ROM de arranque

Rutina introducida por el fabricante a partir de la dirección de memoria FFFF:0000

Todos los procesadores Intel-86 se inician en modo real y ejecutan la instrucción que se encuentra en la dirección FFFF:0000. Ahí se encuentra (en memoria ROM) la primera instrucción que ejecuta todo PC. El código en ROM traerá bytes del disco a una posición de RAM para ejecutarlo a continuación.

ROM BIOS

BIOS = Basic Input/Output System

- Rutinas introducidas por el fabricante en los últimos 64 kb de memoria (memoria ROM).

- Evita al programador la manipulación directa del periférico

- Especificación única, la implementación depende del hardware del fabricante

- La incorporación del BIOS permitió la compatibilidad del software entre los ordenadores PC – Intel 8086 – y definió el término PC compatible

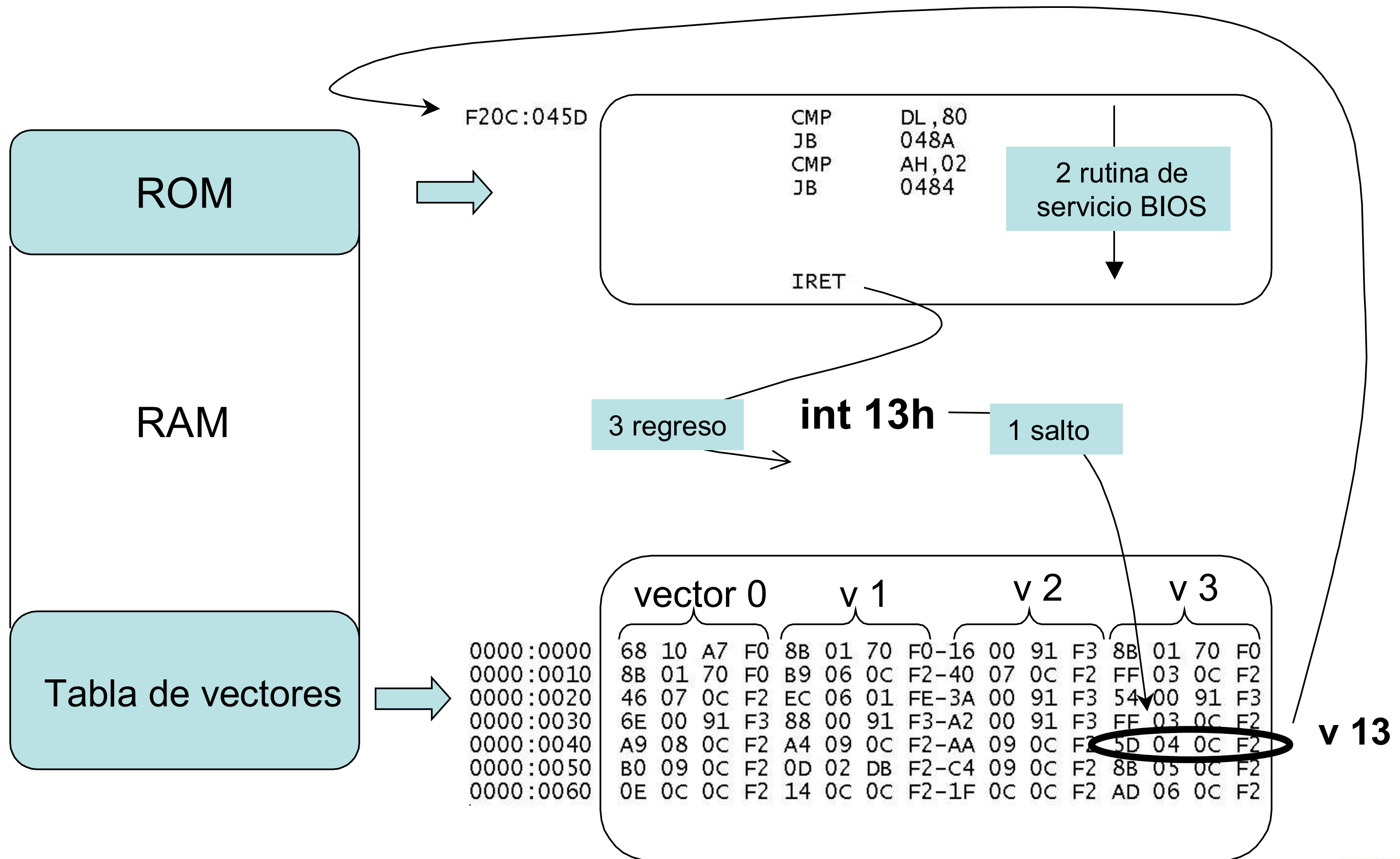
La NO utilización del BIOS por parte del sistema operativo:

- Permite mayor control de los periféricos
- Permite el control completo del código
- Evita el uso del “modo real” del procesador
- Exige programación de periféricos “a medida”

Llamada a una rutina del BIOS (Instrucción de salto indirecto “*int n*”)

Ejemplo

Tras la instrucción int 13 (en RAM) se ejecuta cmp dl,80 (en ROM). Antes del salto almacena en la pila la dirección de retorno y los flags



Detalles de la instrucción *int x*

- Apila flags
- Apila CS,IP
- Inhibe interrupciones (IF=0)
- Salta a la dirección apuntada por el vector x

Nota

La instrucción *int x* siempre se ejecuta, independientemente del valor que tenga IF

DITEC Universidad de Murcia

Detalles de la instrucción *iret*

Retorno de subrutina de atención a la interrupción

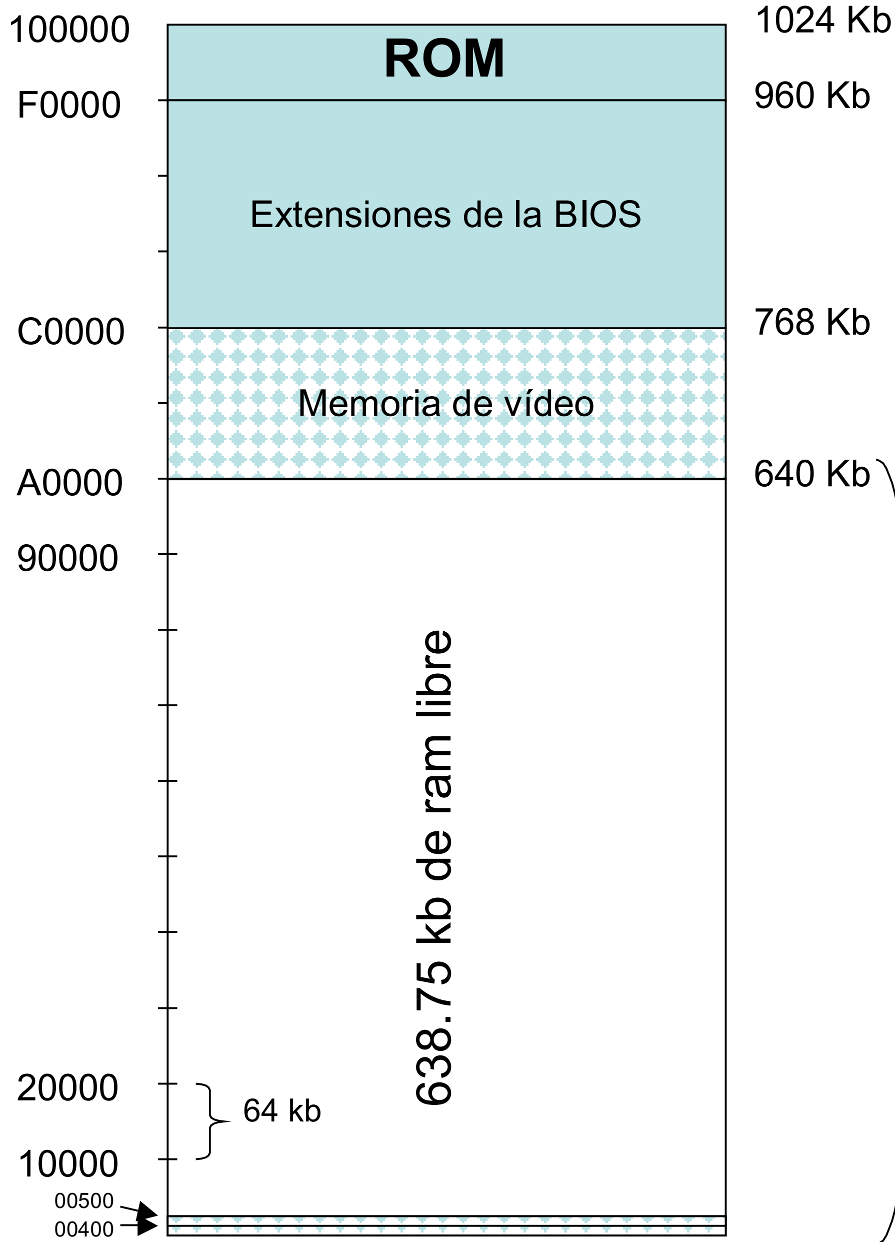
- Desapila IP,CS
- Desapila flags

Nota

La desapilación de los flags hace que IF se cargue con el valor congelado en pila. Así, si las interrupciones estaban permitidas antes de saltar a la SAI éstas vuelven a ser permitidas al retornar

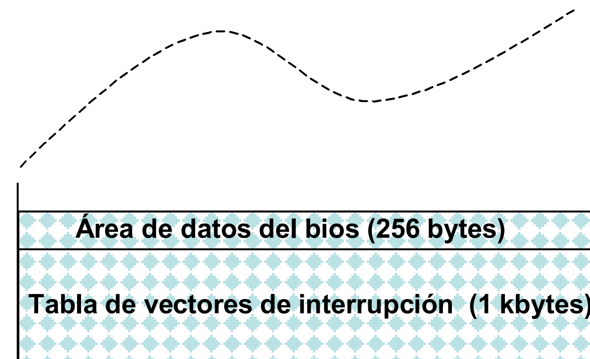
3. Mapa de memoria de un PC

Mapa de memoria de un ordenador compatible en modo real (1 Mb)



- Código de arranque FFFF0 (FFFF:0000)
- Código de los servicios bios

- El código de arranque inicializa la tabla de vectores (RAM)
- Todo código en ROM necesita zona RAM para variables

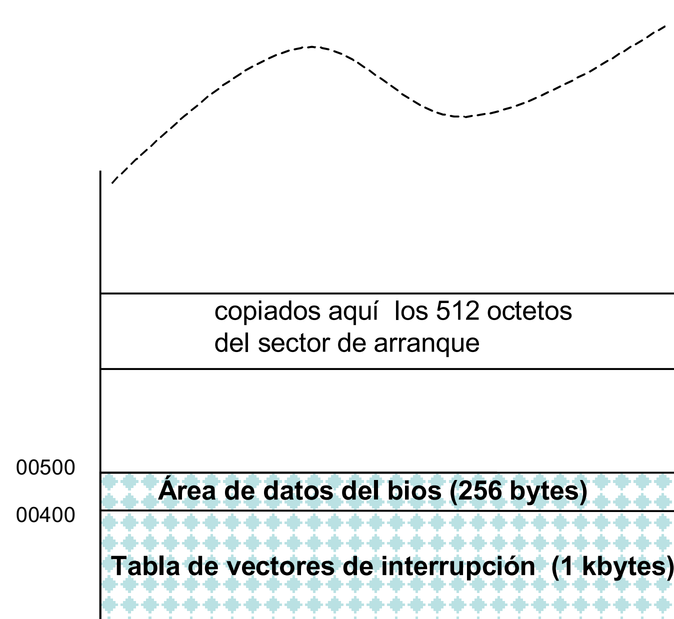


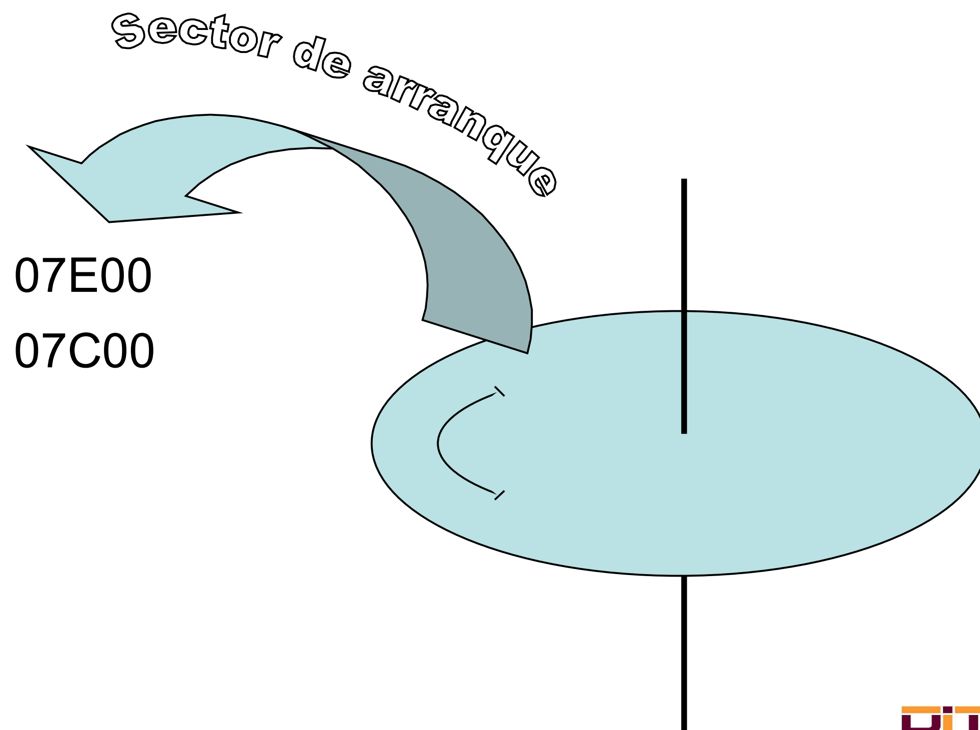
4. Proceso de arranque de un PC

Código de arranque (ROM de arranque) :

La primera instrucción de la ROM de arranque (la primera que ejecuta el ordenador) es la situada en la dirección FFFF:0000 (FFFF0). A continuación el código de arranque:

- inicializa la tabla de vectores
- copia el sector de arranque (primer sector) a partir de la dirección 0000:7c00 (07C00)
- ejecuta la instrucción jmp 0000:7c00 (ejecuta los 512 octetos copiados en memoria)





5. Estructura de un disco 3 ½ 1.44 Mb con sistema de archivo FAT 12 .

Gestión de sectores mediante servicio int 13h de BIOS

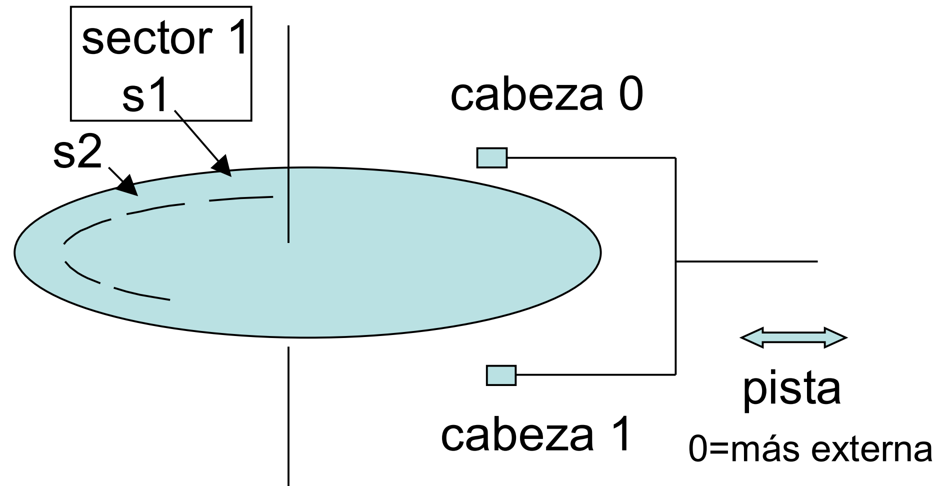
Ideas claves

- Acceso a Disco
  - El acceso a disco es atómico (sectores completos de 512 bytes)
  - La única manera de modificar un bit en un disco es:
    - Traer a memoria el sector que lo contiene,
    - Modificar en la copia de memoria el octeto en el que se encuentra el bit
    - Actualizar el sector del disco con la copia de memoria
- Editor de Disco (editor de sectores)
  - Se pide al usuario el sector a editar
  - Se trae el sector a un array de bytes
  - Se edita el array de bytes
  - Se actualiza el sector

Lectura de sector mediante BIOS

- INT 13h (AH = 02)
  - DL = nº de unidad (0,1,2,3)
  - DH = nº de cabeza (0,1)
  - CH = nº de pista (0,79)
  - CL = nº de sector (1,18)
  - AL = 1 (nº de sectores a leer)

  Deposita el sector a partir de la posición ES:BX (Si CF=1 error)



Numeración DOS: Sector 0 __ s17,s18 __s35, __s2879

Numeración BIOS: P0C0S1 P0C0S18 P0C1S1 P0C1S18 P79C1S18

S0 : sector de arranque
S1.. S18 : FAT y copia

S19..S32 : tabla directorio
S33.. S2879 : datos

cluster 2 cluster 2848

1 cluster =1 sector de datos

cluster 0 y cluster 1 no existen

Lectura de sector mediante BIOS

```
unsigned int leersector (unsigned char far *buffer, char pista, char cabeza,
                         char sector, char unidad)
{
    int i;

    for(i = 0; i < 3; i++){
       asm {
         les   bx, dword ptr buffer
         mov   al , 1      /* 1 sector */
         mov   ch , pista
         mov   cl , sector
         mov   dh , cabeza
         mov   dl , unidad
         mov   ah , 02     /* De disco a memoria. */
         int   13h
         mov   ax , 0      /* Devolvemos 0 indicando que la operación tuvo éxito. */
         jnc   noError     /* Si no se produce error salimos del bucle. */
         mov   ax , -1     /* Devolvemos -1 indicando que la operación falló. */
       }
    }
    noError:
    if (unidad == 0 || unidad == 1)
         pararmotor = 36; /* Unos dos segundos */
}
```

DITEC Universidad de Murcia

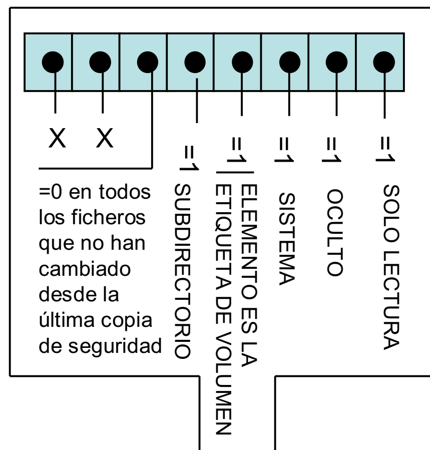
Directorio raiz

Tabla de 224 elementos de 32 bytes (14 sectores)
Cada elemento almacena:

- Nombre del fichero : octeto 0 .. octeto7 (8)

00h: este elemento está sin ocupar E5: fichero borrado (elemento libre) 2E: subdirectorio . 2E 2E : subdirectorio . .

- Extensión: o8 .. o10 (3)
- Atributo: o11 (1)



- Sin uso: o12 .. o21 (10)
- Hora: o22,o23 (2)

  Tiempo = Hora * 2048 + Minutos * 32 + Segundos / 2

- Fecha: o24,o25 (2)

  Fecha = (año-1980) * 512 + mes * 32 + día

- Nº de cluster de comienzo o26,o27 (2)

  = 000 fichero sin espacio

- Tamaño o28..031 (4)

Ejemplo. Primer elemento de la tabla directorio raiz (primeros 32 bytes del sector 19)

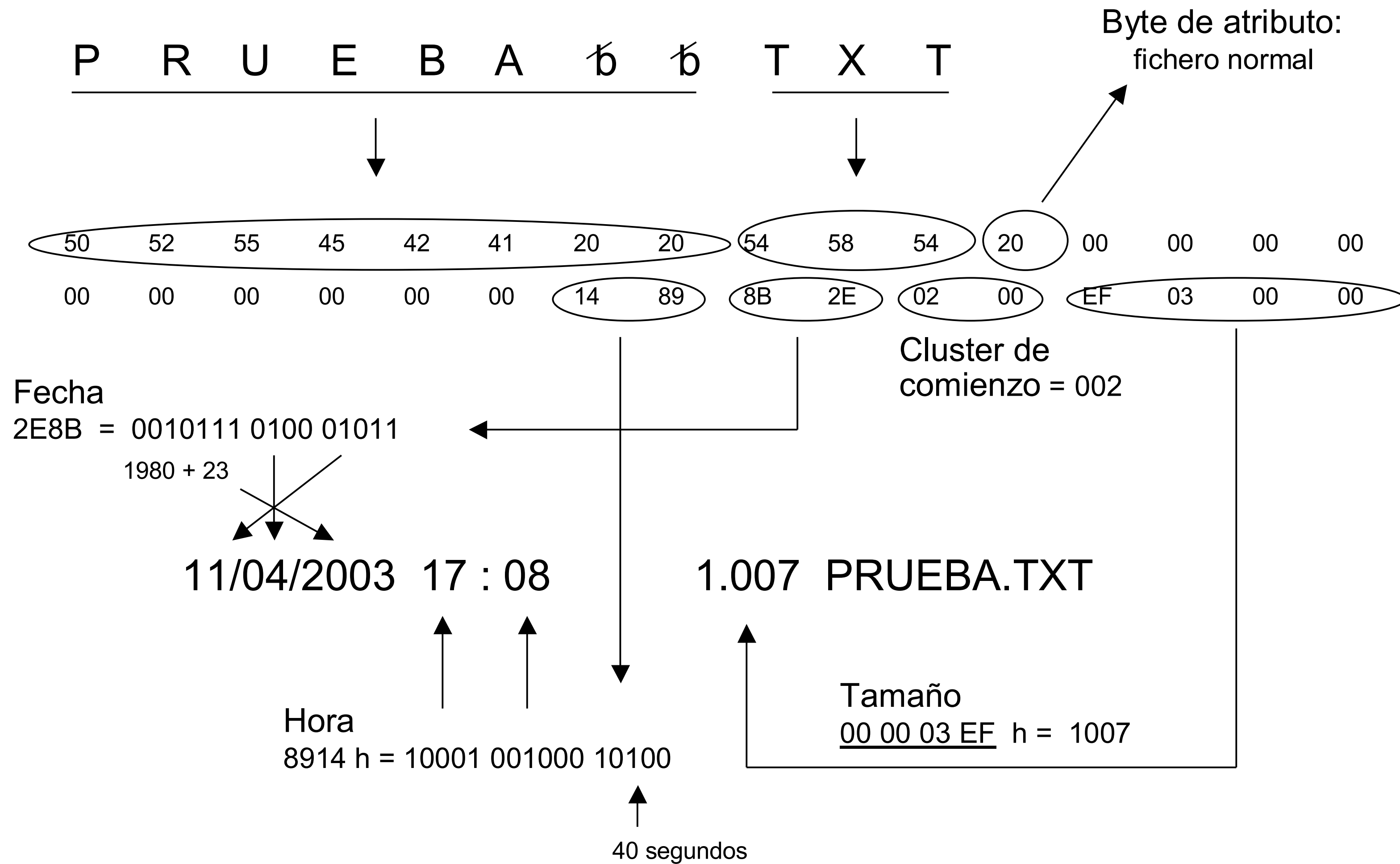


TABLA DE LOCALIZACIÓN DE FICHEROS
(FAT: File Allocation Table)

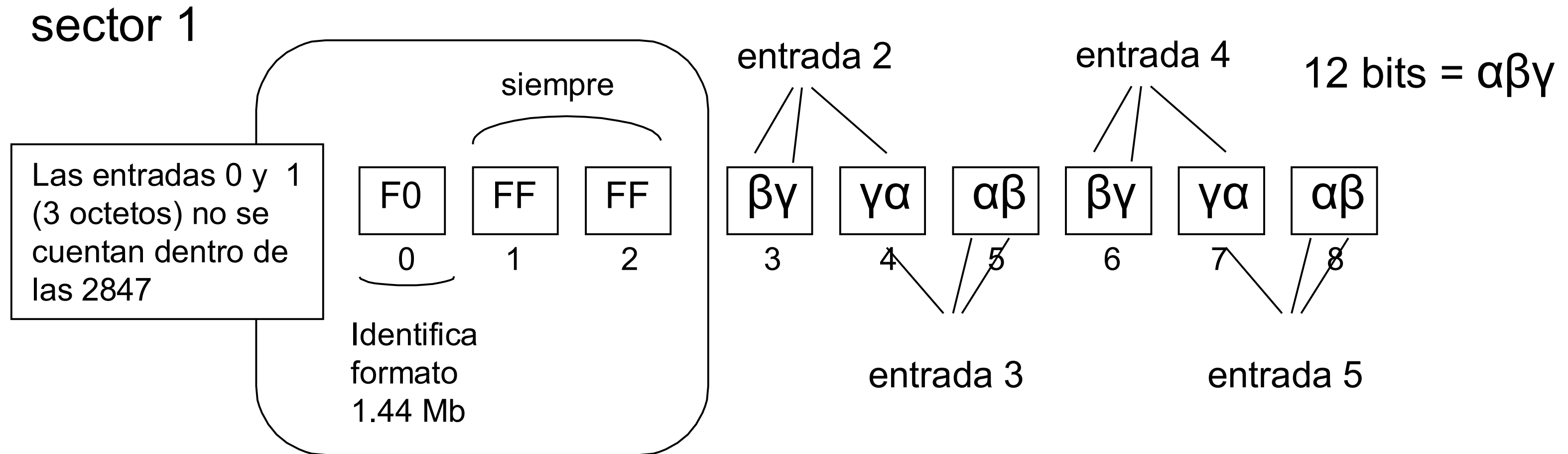
Tabla de 2847 elementos (1 por cluster). El elemento j de la tabla almacena el nº del cluster que le sigue al cluster j en el fichero.

AHORA BIEN:

Si el elemento j vale (cada elemento ocupa 1'5 octetos):

FFF	el cluster j	es el último cluster del fichero
000	" " "	está libre
FF7	" " "	está declarado como inutilizable por un error de formateo

Estructura de la FAT (9 sectores)



Algoritmo de lectura de la FAT

entrada par

$4 * 1.5 \rightarrow 6$

```
  0000 0000 0100
+ 0000 0000 0010
----------------
  0000 0000 0110
```

leer WORD (0000 0000 0110)

γ α β γ

QUITAR

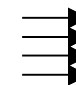
entrada impar

$5 * 1.5 \rightarrow 7$

```
  0000 0000 0101
+ 0000 0000 0010
----------------
  0000 0000 0111
```

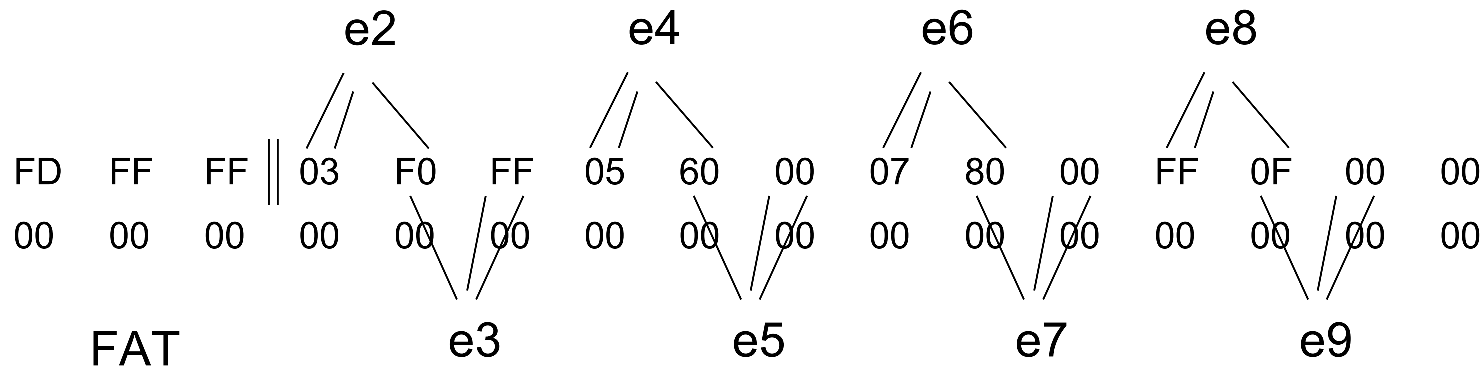
leer WORD (0000 0000 0111)

α β γ α



4 desplazamientos

Ejemplo. Primeras entradas de la FAT (sector 1)



FAT

e 002	003
e 003	FFF
e 004	005
e 005	006
e 006	007
e 007	008
e 008	FFF
e 009	000
e 00A	000
e 00B	000
e 00C	000
e 00D	000

El fichero

11/04/2003 17:08 1.007 PRUEBA.TXT

vimos que comenzaba en el cluster 2. La FAT nos indica que sigue en el 3 y que éste es el último (no aprovecha el total de los 1024 bytes de los 2 clusters)

Hay otro fichero cuyos datos están almacenados en los clusters c4, c5, c6, c7, c8

No hay dispersión de clusters en estos 2 ficheros. Sólo se han gastado 7 clusters del disco

6. Detalles del código generado por el compilador TCC

Ideas claves

- Lenguaje C

  A partir de un fichero de texto c (.c) el compilador crea un fichero de texto ensamblador (.asm) en donde las líneas de lenguaje C aparecen en forma de comentario

- Lenguaje asm

  Los ficheros de texto ensamblador (.asm) son los que realmente necesitamos. Pueden ser creados:

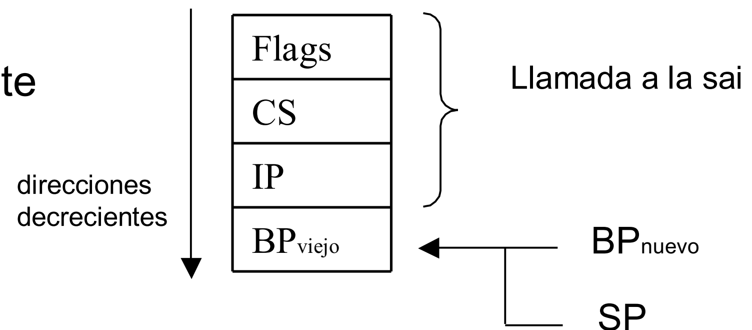
  - desde cero, directamente en ensamblador
  - modificando el texto del fichero .asm generado por el compilador de C
  - generando el fichero asm directamente a partir de un fichero C

DITEC Universidad de Murcia

Detalles del código generado por el compilador TCC

`void sai(void)`
`{`
`   asm { mov ax , bx }`
`}`
	`_sai:`
	`    push bp`
	`    mov bp,sp`
	`        mov ax , bx`
	`    pop bp`
	`    ret`

Estado de la pila a la altura de la instrucción "*mov ax, bx*" si hemos llegado a la dirección "*_sai*" mediante la instrucción de salto "*int x*"



Nota: si dentro de la sai se utilizan los registros si o di el compilador los apila (y desapila) a continuación automáticamente

7. Interrupciones del timer y del teclado

Interrupciones timer y teclado

- Timer, idea clave

  18.2 veces por segundo la CPU abandona el código que esté ejecutando para dar una pasada a una subrutina preparada por el programador

- Teclado, idea clave

  Cada vez que oprimimos una tecla o la soltamos se envía un mensaje de tecla oprimida o tecla soltada (scan code) a la tarjeta de teclado y ésta a su vez activa el hilo de petición de interrupción de teclado. En consecuencia, la CPU abandona el código que esté ejecutando para dar una pasada a la subrutina. La subrutina recoge del hardware del teclado el *scan code* y almacena en memoria (buffer de teclado) el código ASCII asociado. No hace nada más. Los programas no acceden al hardware de teclado, sólo se dedican a esperar a que haya algo en el buffer (si ya hay algo, evidentemente no esperan)

Interrupciones timer y teclado

- Cambios de flujo por activación eléctrica externa de cables
- No surten efecto si IF=0
  - la instrucción *sti* pone IF=1 (habilita interrup.)
  - *cli* pone IF=0 (inhabilita interrupciones)
- Las SAIs (subrutinas de atención a interrupción) antes de finalizar deben de indicarlo al hardware de interrupciones:

```
mov al, 20h
out 20h,al
```

} mensaje EOI: End of Interruption

DITEC Universidad de Murcia

Interrupciones Timer y teclado

- Hilo de petición de interrupción del timer
  - Activado eléctricamente 18,2 veces/segundo
  - Se ejecuta la SAI apuntada por el vector 8 (mayor prioridad que la del vector 9)
- Hilo de petición de interrupción de teclado
  - Activado cada vez que llega un código de opresión o soltado de tecla a la tarjeta de teclado
  - Se ejecuta la SAI apuntada por el vector 9

Interrupciones Timer y teclado

- Detalles del mecanismo de interrupción

  Cada vez que se activa un hilo de interrupción y las interrupciones están permitidas (IF=1), se apilan flags, CS, IP, se inhiben interrupciones (IF=0) y se salta a la dirección indicada por el vector

- Diferencia entre instrucción *int x /* interrupción

  El origen del salto en un caso es una instrucción y en el otro la activación eléctrica de un cable por un periférico que puede ocurrir en cualquier momento (antes de realizarse el salto se completa la instrucción en curso). El resto de detalles es idéntico

DITEC Universidad de Murcia

8. Fichero .COM

Fichero ejecutable .com de ms-dos

- Contiene la imagen binaria “tal cual” del código
- Tamaño < 64 KB
- El código (el fichero .com, completo y sin modificar) se carga en una dirección xxxx:0100 h
- La primera instrucción al ejecutarse se debe encontrar con el siguiente entorno
  - CS=DS=ES=SS=xxxx (código, datos y pila en un solo segmento de 64KB)
  - IP=0100 h
  - SP=inicializado normalmente a FFFE h
  - Los primeros 100h bytes del segmento - direcciones xxxx:0000 a xxxx:00FF- los inicializa el sistema operativo MS-DOS para dar a conocer al proceso diferente información (línea de comandos MS-DOS, dirección del bloque de entorno MS-DOS, etc…). Se denomina PSP (los dejaremos libres en principio)

9. Elaboración de un sector de arranque

Universidad de Murcia

Ubicación del código de arranque en el sector de arranque

Ejemplo: sector de arranque de MS-DOS 6.22

```
struct boot_sector {
        char    salto[3];                 /* 00h */
        char    identificacion[8];        /* 03h */
        short   bytes_por_sector;         /* 0Bh */
        char    sectores_por_cluster;     /* 0Dh */
        short   sectores_reservados;      /* 0Eh */
        char    copias_fat;               /* 10h */
        short   entradas_raiz;            /* 11h */
        short   total_sectores;           /* 13h */
        char    formato_disco;            /* 15h */
        short   sectores_por_fat;         /* 16h */
        short   sectores_por_pista;       /* 18h */
        short   cabezas;                  /* 1Ah */
        short   sectores_ocultos;         /* 1Ch */
        short   pad;
        int     total_sectores_long;      /* 20h */
        char    unidad_hd;                /* 24h */
        char    reservado;                /* 25h */
        char    marca;                    /* 26h */
        int     numero_serie;             /* 27h */
        char    etiqueta[11];             /* 2Bh */
        char    reservado_dos[8];         /* 36h */
        char    cargador[0x1BE−0x3E];     /* 3Eh */
        char    particiones[512−0x1BE];   /* 1BEh */
};
```

Instrucción “*jmp +003E*”

000	EB	3C	90	4D	53	44	4F	53-35	2E	30	00	02	01	01	00
010	02	E0	00	40	0B	F0	09	00-12	00	02	00	00	00	00	00
020	00	00	00	00	00	00	29	F0-19	36	2B	4E	4F	20	4E	41
030	4D	45	20	20	20	20	46	41-54	31	32	20	20	20	FA	33
040	C0	8E	D0	BC	00	7C	16	07-BB	78	00	36	C5	37	1E	56
050	16	53	BF	3E	7C	B9	0B	00-FC	F3	A4	06	1F	C6	45	FE
060	0F	8B	0E	18	7C	88	4D	F9-89	47	02	C7	07	3E	7C	FB
070	CD	13	72	79	33	C0	39	06-13	7C	74	08	8B	0E	13	7C
080	89	0E	20	7C	A0	10	7C	F7-26	16	7C	03	06	1C	7C	13
090	16	1E	7C	03	06	0E	7C	83-D2	00	A3	50	7C	89	16	52
0A0	7C	A3	49	7C	89	16	4B	7C-B8	20	00	F7	26	11	7C	8B
0B0	1E	0B												[illegible]	7C
0C0	00	BB												[illegible]	1D
0D0	B0	01												[illegible]	F3
0E0	A6	75												[illegible]	7D
0F0	E8	5F												[illegible]	19
100	58	58												[illegible]	FF
110	F7	E3												[illegible]	00
120	50	52	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	58
130	72	BB	05	01	00	83	D2	00-03	1E	0B	7C	E2	E2	8A	2E
140	15	7C	8A	16	24	7C	8B	1E-49	7C	A1	4B	7C	EA	00	00
150	70	00	AC	0A	C0	74	29	B4-0E	BB	07	00	CD	10	EB	F2
160	3B	16	18	7C	73	19	F7	36-18	7C	FE	C2	88	16	4F	7C
170	33	D2	F7	36	1A	7C	88	16-25	7C	A3	4D	7C	F8	C3	F9
180	C3	B4	02	8B	16	4D	7C	B1-06	D2	E6	0A	36	4F	7C	8B
190	CA	86	E9	8A	16	24	7C	8A-36	25	7C	CD	13	C3	0D	0A
1A0	4E	6F	6E	2D	53	79	73	74-65	6D	20	64	69	73	6B	20
1B0	6F	72	20	64	69	73	6B	20-65	72	72	6F	72	0D	0A	52
1C0	65	70	6C	61	63	65	20	61-6E	64	20	70	72	65	73	73
1D0	20	61	6E	79	20	6B	65	79-20	77	68	65	6E	20	72	65
1E0	61	64	79	0D	0A	00	49	4F-20	20	20	20	20	20	53	59
1F0	53	4D	53	44	4F	53	20	20-20	53	59	53	00	00	55	AA

código que se ejecuta después de la instrucción *jmp +003E* (cargador)

Modificación del código de arranque

- Copiaremos el sector de arranque a un array de 512 bytes (o bien a una variable de tipo *boot_sector*)
- Cambiaremos los bytes que se encuentran a partir de la posición 3Eh del array por los bytes de nuestro programa cargador (cambiaremos el campo cargador de la variable con los bytes de nuestro cargador)
- Finalmente copiaremos el array de bytes en el sector de arranque (copiaremos la variable completa en el sector de arranque)

Hemos de tener en cuenta, a la hora de preparar el código de nuestro cargador, que cuando la ROM de arranque cargue el sector de arranque en la dirección 0000:7C00 nuestro cargador estará ubicado a partir de la dirección 0000:7C3E

DITEC Universidad de Murcia

Bibliografía

- PC INTERNO 2

  Michael Tischer, MARCOMBO 1995
- Sistema Operativo DOS 4

  Jaime de Yraolagoitia, PARANINFO 1992
- Lenguaje Ensamblador y Programación para PC IBM y Compatibles

  Peter Abel, PRENTICE HALL 1996