Estos son los apuntes de clase de Jesús Iván Bethencourt Álvarez, que amablemente los pone a disposición de todos los compañeros. No he revisado el contenido en profundidad, sólo he corregido algunos errores que saltaban a la vista, pero aparentemente es una buena base para completar unos apuntes de esta asignatura. Hay un buen número de figuras, lo más tedioso de hacer sin duda, que indican las horas de trabajo que Jesús Iván le dedicó a estos apuntes. Nuestro agradecimiento y de antemano a todos los que contribuyan en el futuro. He reorganizado el contenido del archivo original que me dejó Jesús Iván y añadido algunos comentarios al margen.

Juan Julian Merino Rubio

ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES IV (ETC IV)

Comentario: El programa de la asignatura actual está en el archivo **TemarioETCIV.doc** He reorganizado este archivo con el orden del temario.

Tema 2

Estructura de un Sistema PC

- **Estructura del PC/PC-XT**

Comentario: Insertar el archivo **ibmpc.doc**, que contiene la figura del diagrama de bloques del PC original.

Las líneas de direcciones y datos del 8088 son compartidas, pero no de forma simultánea. Están multiplexadas en el tiempo, tiempo para datos y tiempo para direcciones.

Transceptores: son amplificadores con salida triestado. Funcionan de izquierda a derecha o de derecha a izquierda. Cuando quiere escribir abre el transceptor de izquierda a derecha y se lo pasa al bus de datos, "deja pasar los datos". Para leer lo abre en el otro sentido para transmitir los datos, el sentido lo controla la CPU.

Lógica de NMI: es un circuito que tuvieron que añadir los de IBM para controlar las interrupciones no enmascarables.

El circuito de lógica de estados de espera se pone a causa de los diferentes tiempos de acceso y por la diferencia de velocidades de los dispositivos.

PIC: controlador de interrupciones, este controla las interrupciones que son enmascarables.

- **Generador de Reloj.**

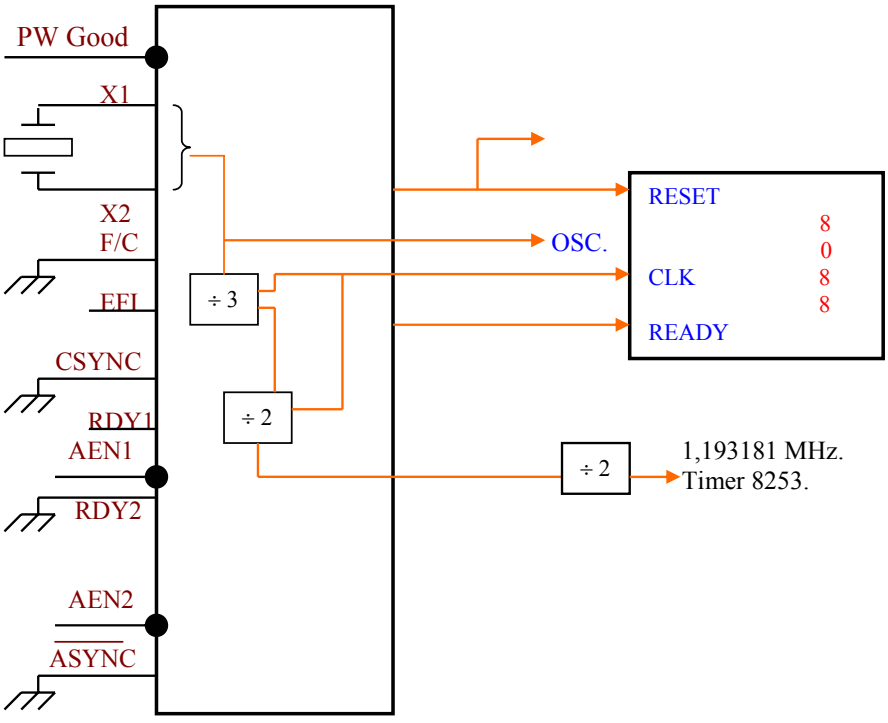
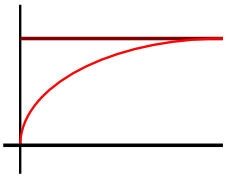


Figura 7. Esquema de un generador de reloj.

Es el pulso READY quien controla a la CPU, diciéndole, más lento o más rápido.

Un período de reloj es igual a 210 nanosegundos (4,772727 MHz), cada instrucción puede durar x periodos. Si tenemos un conjunto de instrucciones sería lógico pensar que si sumamos todos los tiempos tenemos el tiempo total de ejecución pero eso no es así realmente.

Cuando se enciende el equipo van creciendo las tensiones, hasta llegar a la tensión deseada. Una vez que se ha llegado es la propia fuente la que resetea el sistema e inicializa.



La frecuencia inicial es de 14,31818 (por compatibilidad con los televisores norteamericanos) dentro del 8284 se divide por tres y luego por dos por lo que sale del 8284 con frecuencia de 2,3863363334 MHz y luego en placa base se divide por dos con lo que se queda en 1,193181 que va al timer.

• Estructura PC-AT (Procesador 286).

Sigue existiendo la lógica para enmascarar las interrupciones no enmascarables. También sigue teniendo un controlador del bus.

Las diferencias en la placa base con respecto al PC:

- Hay dos controladores de DMA, hay más páginas de DMA.
- El Timer es diferente.
- La arquitectura es compatible con el PC para que los programas que corran en PC también lo hagan en AT.

• Estructura del procesador 386.

Las líneas de direcciones y el bus de datos son de 32 bits. La arquitectura de la placa base es idéntica a la del 286. Tiene el bus ISA, que existe desde el AT en adelante, que se ha mantenido por compatibilidad en los equipos actuales.
A partir del AT se empezaron a hacer circuitos super-integrados.

• Estructura del procesador 486.

El procesador tiene una velocidad de 50 Mhz. La placa base es semejante.

Al tener mayor velocidad se produce un cuello de botella en el bus ISA, para resolver esto empezaron a aparecer otras alternativas al bus ISA, una de esas alternativas es que los equipos que necesitaban velocidad (ej: tarjeta de vídeo) se conectaban directamente al 486 con un bus local.

Con los Pentium nació el bus PCI, pero se mantiene el bus ISA. El bus siempre va detrás de la CPU en cuestión de velocidad.

Comentario: Insertar la figura del archivo **PCIsystem.wmf**, que representa el diagrama de bloques de un sistema Pentium con bus PCI

Tema 3

El Procesador y el Coprocesador matemático

- **Patillaje del 8086.**

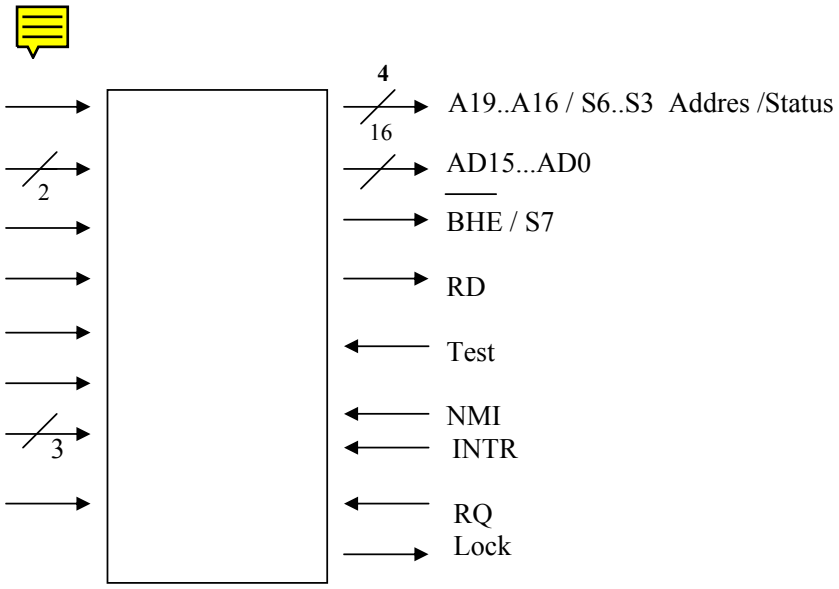


Figura 2. Patillaje del 8086

Los pines más significativos son:

MN/ $\overline{\text{MX}}$: modo mínimo y máximo { 1 mínimo / 0 máximo

$\overline{\text{S9}}$.....$\overline{\text{S2}}$: según sea el modo (máximo o mínimo) cambian algunas de estas líneas, en modo mínimo sirven para control del bus. En la línea de estado está incluida lo que se permite, es decir, si es E/S o si es de memoria.

En modo mínimo las líneas de control (ALE..) están incluidas en las líneas de estado.

En modo máximo se le añade un chip de control del bus.

AD15...AD0: líneas de direcciones y datos multiplexadas, reduce el número de líneas pero no permite la simultaneidad de transmitir datos y direcciones.

READY: reconocimiento de la memoria o de la interfaz de E/S para que la cpu pueda completar el ciclo de bus actual.

RD: indica si va a realizar una lectura en memoria o de E/S.

BHE/S7: si está a cero durante la primera parte del ciclo de bus indica que, al menos, uno de los bytes de transferencia actual va por las patillas AD15...AD8 si está a uno la transferencia se hace por AD7...AD0.

NMI: petición de interrupción no enmascarable (disparable por flanco de subida).

INTR: petición de interrupción enmascarable (disparable por nivel).

Ciclo básico de CPU.

La C.P.U. consta de 4 períodos de reloj:

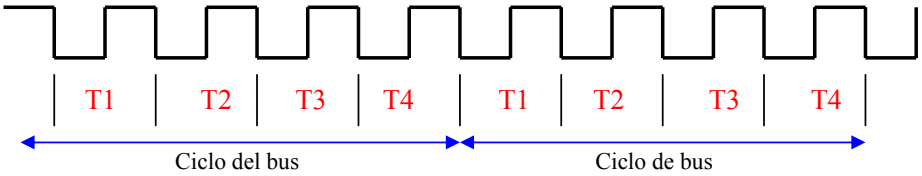


Figura 3. Ciclo del reloj

La dirección se pone en T1 y la quita en T2. Cuando saca las direcciones en T3 se activa la línea de datos, se supones que los datos han sido pasados desde la memoria hasta el bus de datos (atain) leer datos.

La señal de ready la controla la memoria o el periférico. En T3 se mira la línea de ready. Si ready esta bajo duplica el estado T3, es decir, lo que hay en T3 se duplica. Espera indefinidamente hasta que esta se active.

Hay que tener en cuenta que cada 4 pulsos de reloj es un ciclo:

En T1: se mete la dirección a la que se pretende acceder.

En T2: período para cambiar la dirección del bus durante la operación de lectura.

En T3 y T4: transferencia de datos.

La Línea BHE sirve para decidir si esta accediendo a una dirección par o a una dirección impar.

BHE	A0	
0	0	Palabra completa.
0	1	Byte superior, dirección impar.
1	0	Byte inferior, dirección par.
1	1	Nada.

S2	S1	S0	
0	0	0	Reconocimiento de interrupción.
0	0	1	Lectura puerto E/S.
0	1	0	Escritura puerto E/S.
0	1	1	Halt.
1	0	0	Acceso a una instrucción.
1	0	1	Lectura de la memoria.
1	1	0	Escritura en la memoria.
1	1	1	Bus IDLE (No esta ocurriendo nada).

Ciclo de lectura.

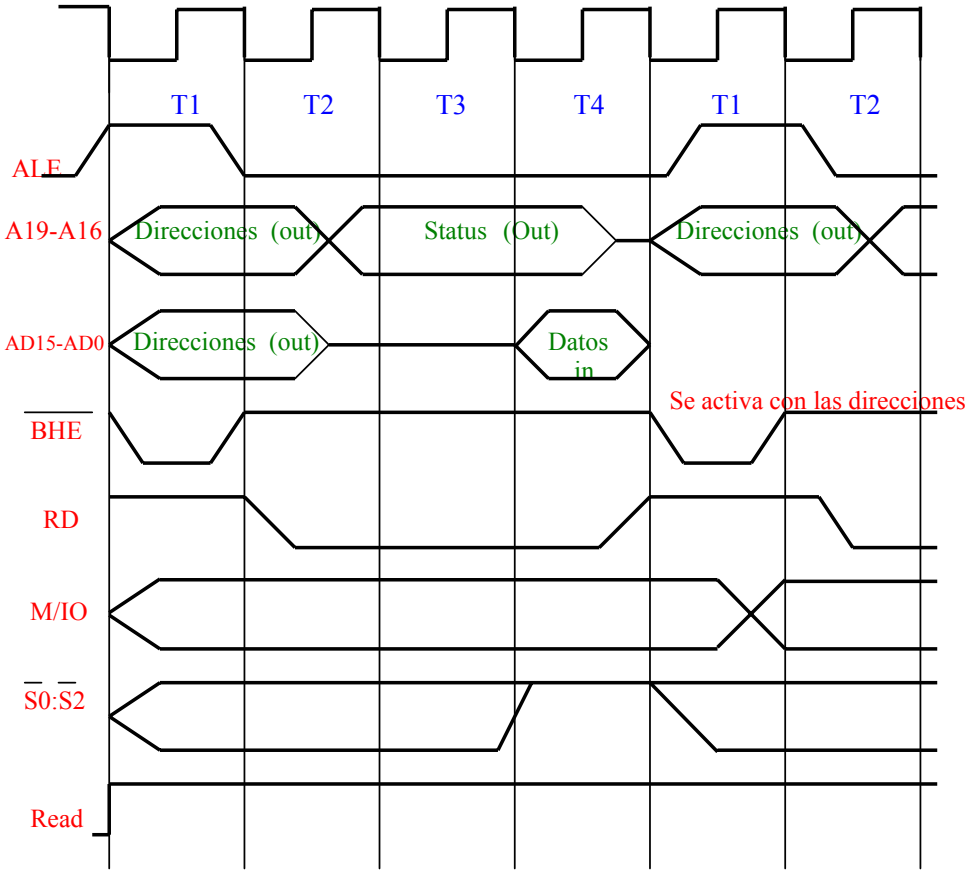


Figura 4. Ciclo de Lectura

Ciclo de lectura con estado de espera.

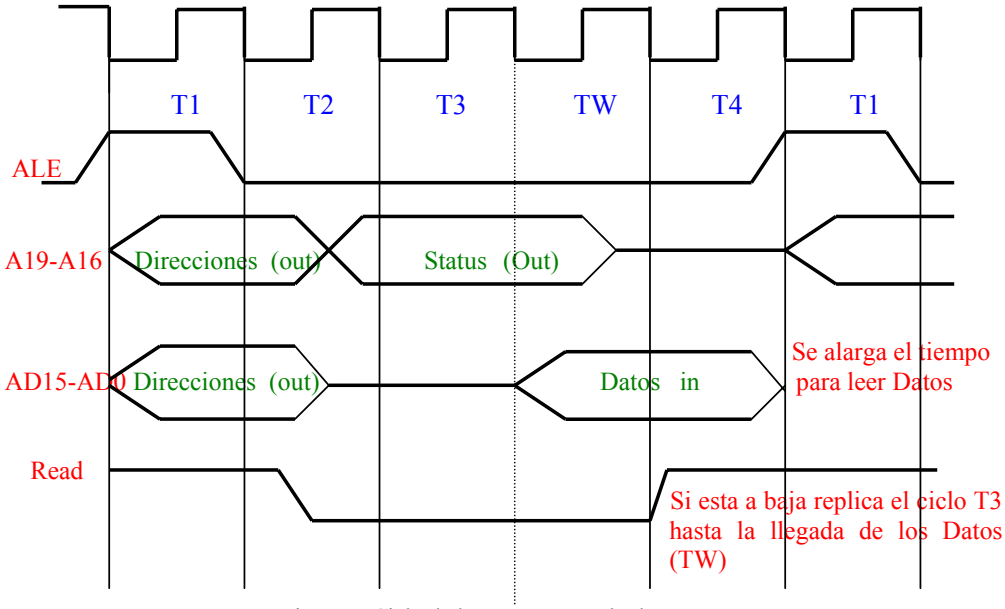


Figura 5. Ciclo de lectura con estado de espera.

Si el periférico entre **T3** y **T4** no puede poner los datos **READY** se pone a baja y replica **T3 (TW)** indefinidamente hasta que halla datos en el bus, esta señal la controla el periférico o la memoria a la que sé esta accediendo.

La línea **ALE** detecta los ciclos, suele ser la que se pincha en el osciloscopio para verla.

Ciclo de escritura.

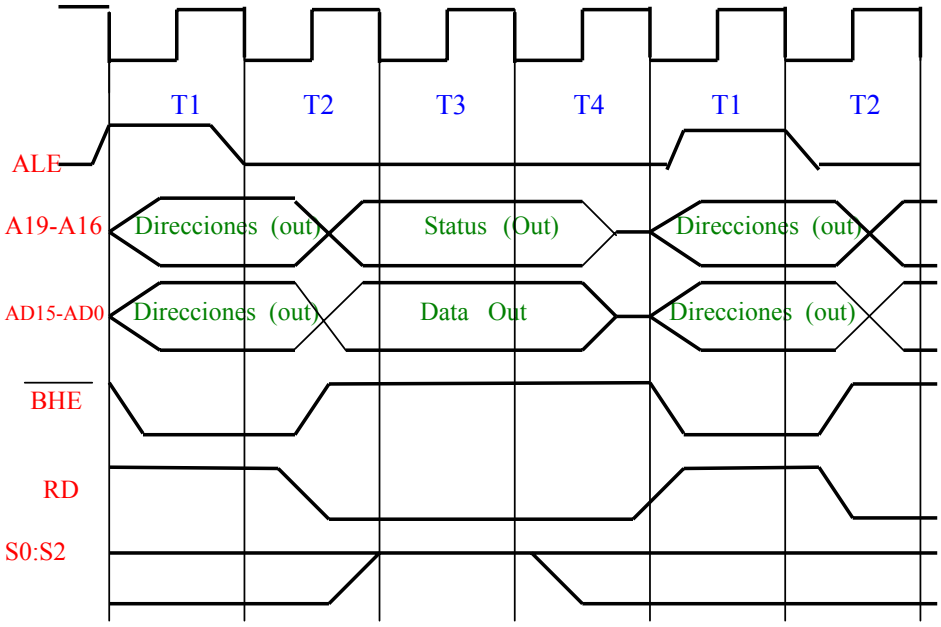


Figura 6. Patillaje del 8086

S4 y S3 nos dicen el registro de segmento que sé esta utilizando para acceder al bus:

(A17) S4	(A18) S3	
0	0	ES.
0	1	SS.
1	0	CS o ninguno.
1	1	DS.

S5 (A18): imagen del flag de interrupción.

S6(A19) { 0 La CPU posee el bus.
1 No lo posee.

Si la CPU reconoce que la quieren resetear (la línea de reset tiene que estar en alta el tiempo suficiente) el registro de flags lo pone todo a cero y también el SS lo pone a cero por tanto habilita las interrupciones. El CS lo pone todo a uno. El PC lo pone todo a FFFFh, los restantes (todos menos CS) los pone a cero. Lanza la ejecución con CS: PC.

Veamos el efecto de la cola de *prefetch* en la ocupación de la unidad de ejecución **EU**:

mov	dl, 4	(a)	B1	04	4
real	bx, 1	(b)	D1	D3	2
mov	al, bl	(c)	8A	C3	2
shr	ax, 1	(d)	D1	28	2
xor	bx, bx	(e)	33	DB	3
shr	ax, 1	(f)	D1	28	2

EU	a			b		c		d		e		f	
BIU	D1	28		33	DB			D1	28				
B1	D1			D1	8A		D1	D1	33		-	D1		
04	D3			D3	C3		28	28	DB		-	28		
D1	8A			8A	D1		-	33	-					
D3	C3			C3	28		-	DB	-					
8A	-			D1	-		-		-					
C3	-			28	-		-		-					

La **BIU** va rellenando la cola. Aprovechando el bus del 8086 se trae un word, pero tarda 4 ciclos de reloj en extraer el siguiente word.

Llega un momento en el que la cola está vacía, aquí la **EU** trabaja permanentemente.

Ahora imaginamos que la cola inicialmente estuviera vacía:

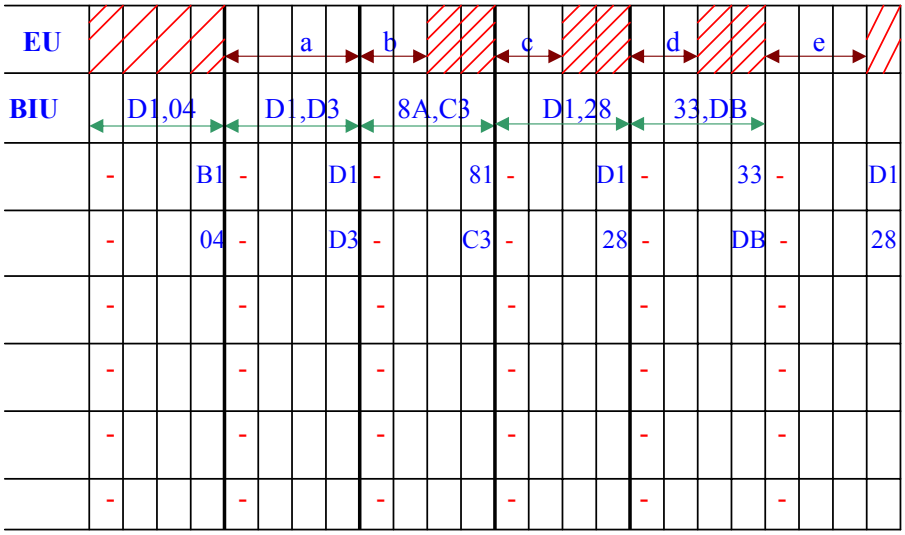
Aquí en muchas ocasiones la **EU** está sin hacer nada, es decir, está ociosa. En el 8088 la situación es aun más grave, la **EU** está más tiempo ociosa.

Mirar el esquema de la página siguiente.

Ejercicio: Hacer lo mismo con el 8088 sabiendo que:

La cola tiene una profundidad de 4. Se trata un byte de la cola cada vez y no un word.

Solución: Está ociosa 19 de los 34 períodos de reloj que tarda. Con la cola vacía en el 8088 el tiempo total es de 50 períodos en el cual 35 períodos ociosa, es decir, el 70% ociosa.



• Ciclo de Reconocimiento de interrupción.

La petición le llega a la CPU en cualquier momento, esta termina el ciclo que está realizando en ese momento y inicia el ciclo de interrupciones compuesto por dos ciclos.

La señal INTA indica el reconocimiento de una petición de interrupción. Consta de dos ciclos de bus consecutivos, en el primero se le indica al dueño de la petición que esta ha sido concedida y en el segundo se le dice que ya puede enviar los datos por el bus y en este ciclo mira el bus y lee la parte baja de este y reconoce al periférico que pide la interrupción (0..255). El bus bloquea el bus con la señal LOCK.

Mirar la figura de la siguiente página.

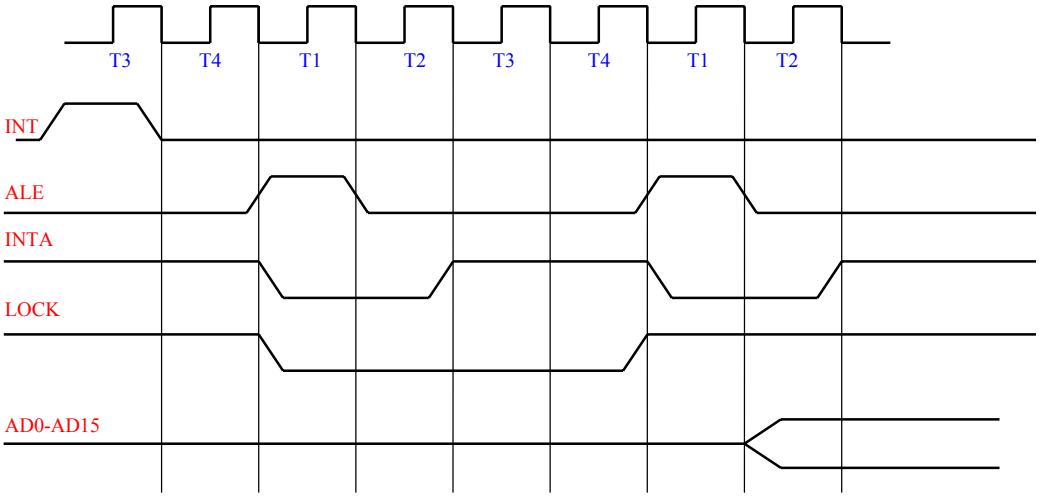


Figura 8. Esquema de reconocimiento de una instrucción

Proceso de petición de interrupción por el periférico.

1.- La C.P.U. le contesta y empieza el reconocimiento.

2.- La C.P.U. bloquea el bus con la señal LOCK

3.- El periférico que pidió la interrupción mira la señal INTA y espera que se active por segunda vez y pone en el bus de datos un número que lo identifique.

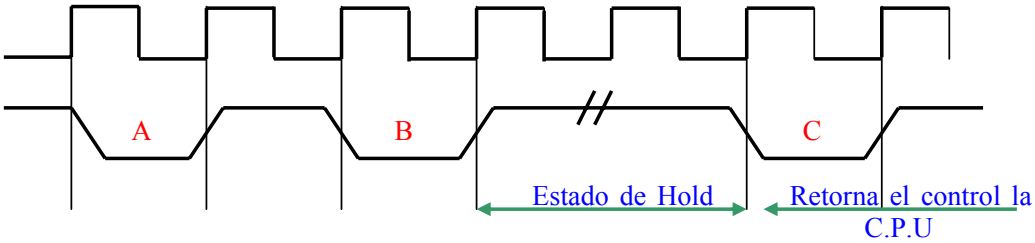


Figura 9. Esquema de petición de interrupción por el periférico.

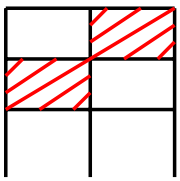
El estado Hold tiene duración indefinida, todo el tiempo que necesita el master del Bus para hacer sus transferencias.

En el estado Hold la C.P.U se desentiende del bus y lo toma el master del periférico que lo esta utilizando.

En la etapa A alguien pide el Bus a la C.P.U. en la etapa B la C.P.U. lo concede, el master se hace dueño del bus y en la etapa C se devuelve el control a la C.P.U.

El estado de Hold lo controla la línea $\overline{RQ}/\overline{GT}$ (0,1) (que según el modo sé dirección) esta línea esta conectada al master del periférico con lo que se aceptan y se envían peticiones, y se dan concesiones del bus.

Nota: Cuando una word está alineada en dirección impar, hace dos accesos ignorando después los bytes a los que no quiere acceder y luego alinea los bytes que interesan:



- **Procesadores que siguieron al 8088 y 8086.**

80286: Dio origen al AT. Trata de eliminar algunos de los problemas del 8086.

- Todos los registros son de 16 bits.
- Tiene 24 líneas de direcciones con un espacio de 16 MByte de memoria.
- Sigue la separación entre ejecución e interfaz del bus.
- Hay 4 unidades que funcionan por su cuenta:
  - Bus de interface.
  - Unidad de codificación.
  - Unidad de ejecución.
  - Unidad de direcciones

Mirar la figura del 80286 CPU block diagram.

Los segmentos pasaron de ser registros a ser descriptores. En el 8086 una posición de memoria podría ser direccionada por más de una combinación de los registros. El 286 se hizo bimodal, el modo real compatible con el 8086 y el modo protegido que admite memoria virtual.

El reloj del sistema es dividido internamente en dos.

La Unidad de Control se considera como una maquina de estados, la cual está formada de cuatro estados, que funciona de la siguiente manera:

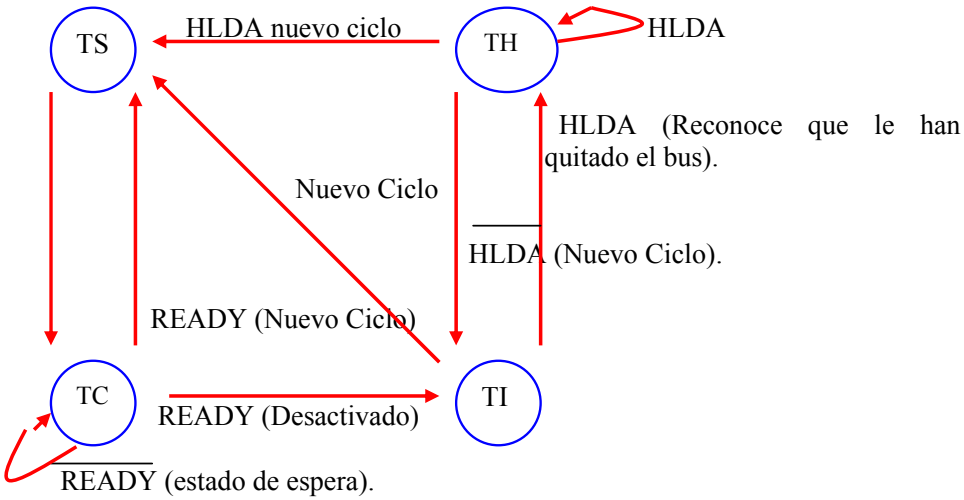


Figura 10. Estados de la Unidad de Control

Los cuatro estados del bus son:

- TS: send status.
- TC: perform command.
- TI: bus idle.
- TH: bus hold.

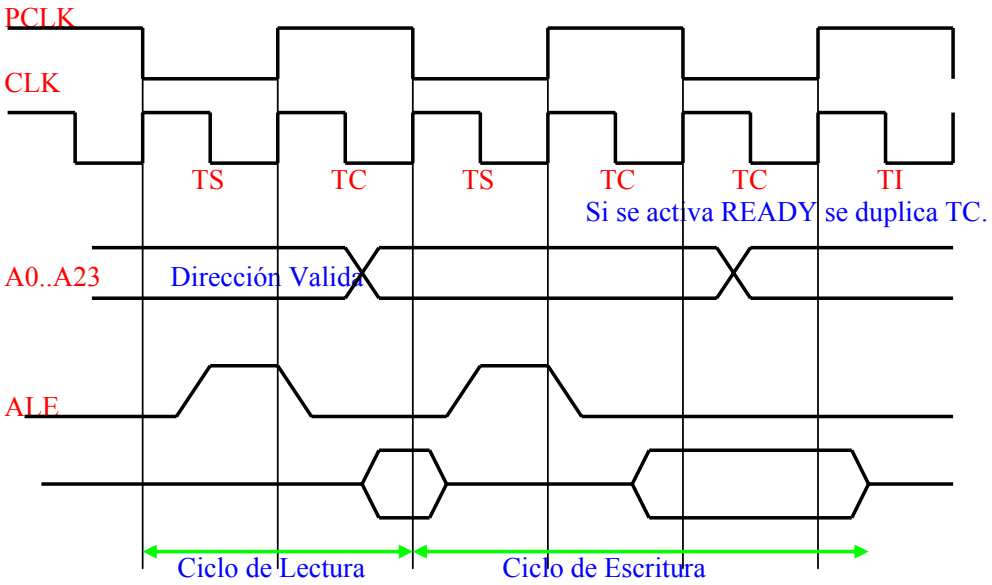


Figura 11. Ciclo de lectura y escritura de los estados de la Unidad de control.

En el AT hay que meter por lo menos un estado de espera en la memoria, porque la memoria no puede ir tan rápido como el procesador (memoria dinámica). Solamente en el XT original no hay estados de espera. Tres períodos de reloj TS, TC, TI con estado de espera.

La arquitectura general de AT se mantiene hasta hoy.

Mirar la figura del 80286 CPU block Diagram.

80386: las características más notables son:

- Permite poner cache externa.
- Acelera multiplicación y la división.

- Salió con 25 MHz y avanzó hasta 83 MHz.

En la **UE** se incorpora un desplazador que es capaz de desplazar 32 bits de un solo ciclo, con el 286 solo se desplazaba un bit.

Antes	Ahora
Shl ax, 1 Mov cl, 3 Shl ax, cl	shl ax, 7

A la A.L.U. se le añade un sumador de 3 estados, está especialmente dedicado al cálculo de las direcciones efectivas, por eso, la instrucción LEA es más rápida.

Otras características importantes son:

- Procesador auténtico de 32 bits.
- 32 Líneas de direcciones.
- 32 Líneas de datos.
- Direccionamiento de 2E32 = 4 GByte.
- Mantiene la unidad de decodificación.
- Cola de instruccuiones decodificadas de profundidad tres.
- Desplazador en tonel (permite desplazar hasta 32 bits en un solo ciclo).

Mirar la figura del 80386 CPU block Diagram.

80486: las características mas notables son:

- La cola de prefetch aumenta hasta profundidad de 32 bytes.
- Por primera vez se añade al procesador memoria caché.
- Posee una unidad de punto flotante que antes era un chip externo opcional , ahora está integrado.

Mirar la figura del 80486 CPU block Diagram.

Pentium: sus características mas notables son:

* Posee dos cache internas { Datos / Código.

- Posee dos colas de prefetch de 64 bytes de profundidad.
- Dos cauces de ejecución (*pipe-line*).
- En cada ciclo se pueden ejecutar dos instrucciones.
- En los saltos se pueden hacer predicción de saltos, por un cauce se decodifica la instrucción siguiente y por el otro las del salto, y ya están decodificadas en ambos casos.
- La unidad de punto flotante ya está dentro del cauce de ejecución, se integra mucho más dentro de la **EU**.

Pentium-pro: las novedades que incorporan son:

- Tabla de áreas de registro.
- Estación de reserva.
- Piscina (de instrucciones ejecutadas).

Nos podemos hacer la siguiente pregunta:

¿Cómo un programa puede diferenciar la CPU?

Hasta el Pentium sólo se diferencian en los flag y a partir de aquí se ejecuta una instrucción y se sabe que CPU es.

Las CPU se diferencian en pequeñas combinaciones que existen entre sus flag.

Lo que hacemos es diseñar un programa que va mirando los flag y cuando uno de ellos falle será la CPU anterior. El programa en ensamblador es el siguiente:

IDENTIFICADOR CPU

Comentario: He introducido aquí mi programa original, más completo y comentado.
La sintaxis corresponde al MASM 6.xx de Microsoft.

```
_DATA   SEGMENT PARA    PUBLIC  'DATA'

infcpuid    DWORD   2 DUP (0)
msgcpuid    BYTE    12 DUP (0)

_DATA   ENDS
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; Una nueva instrucción para el Pentium

CPU_ID   TEXTEQU         <BYTE   00Fh,0A2h>
```

```
BPT     TEXTEQU         <BYTE PTR>
WPT     TEXTEQU         <WORD PTR>
DPT     TEXTEQU         <DWORD PTR>

_TEXT   SEGMENT PARA    PUBLIC  'CODE'
      ASSUME  CS:_TEXT,DS:_DATA

;================================================ IdentificaCPU =====
; El procedimiento IdentificaCPU devuelve un código en el registro AX
; para indicar el tipo de procesador y de coprocesador que está
; instalado en el sistema. Este código se interpreta como sigue:
;
;       AH      tipo de CPU      |       AL      coprocesador
;      -------------------------|-----------------------------------
;       01      8086 u 8088      |       00      ninguno instalado
;       02      80286            |       01      8087
;       03      80386DX u 80386SX|       02      80287
;       04      80486DX u 80486SX|       03      80387DX u 80387SX
;                                |       04      80487DX u 80487SX
;       05      Pentium (586)    |       05      Pentium
;       06      Pentium Pro      |       06      Pentium Pro
;
; Si se devuelve el código 0400 ello indica a un 80486SX sin el
; coprocesador 80487SX.
;-------------------------------------------------------------------------
IdentificaCPU   PROC    NEAR PASCAL USES BX CX DX
      LOCAL   NDP_STATUS: WORD
.8086
      mov     dx,0100h                ;identificador inicial
;---------------------------------
;en el 8086/88 están siempre a 1 los bits 15..12 del registro
;de FLAGS.
;---------------------------------
      pushf                   ;se salva FLAGS
      pop     ax              ;se recuperan en AX
      and     ah,0Fh          ;se borran los 4 bits de más peso
      push    ax              ;se devuelve a
      popf                    ;FLAGS
      pushf                   ;de nuevo se copian en AX
      pop     ax
      and     ah,0F0h         ;se aíslan los 4 bits superiores
      cmp     ah,0F0h         ;si están a 1 la CPU es
      je      CPUID_2         ;un 8086/88
;---------------------------------
;en el 80286, en cambio, los bits 15..12 siempre están
;borrados a 0.
;---------------------------------
      inc     dh              ;aumenta el ID
      pushf                   ;copia los FLAGS
      pop     ax              ;en AX
      or      ah,0F0h         ;pone a 1 los 4 bits altos
      push    ax              ;y los devuelve a FLAGS
      popf
      pushf                   ;se recuperan de nuevo
      pop     ax
      and     ah,0F0h         ;si están borrados debe ser un 80286
      jz      CPUID_2
;---------------------------------
;La CPU debe ser un 80386/486. Para diferenciarlos se comprueba
;el bit de alineamiento (bit 18 de EFLAGS) que sólo está definido
;en el 80486.
```

```
;--------------------------------
.386                              ;habilita el modo de instrucciones
                        ;de 32 bits
      inc     dh                ;aumenta el código

      mov     ecx,esp           ;salva el puntero actual de la pila
      and     esp,not 3         ;lo alinea a doble palabra poniendo
                        ;a cero los dos bits más bajos
      pushfd                    ;copia EFLAGS en
      pop     eax               ;EAX
      mov     ebx,eax           ;lo salva para restaurarlo luego
      xor     eax,00040000h     ;complementa al bit AC
      push    eax               ;lo devuelve
      popfd                     ;a EFLAGS
      pushfd                    ;volvemos a
      pop     eax               ;coger a EFLAGS
      xor     eax,ebx           ;si el bit no ha cambiado
      jz      CPUID_1           ;debe ser un 80386
;por defecto debe ser un 80486 o posterior.
.486
      inc     dh
      push    ebx               ;se restaura el EFLAGS anterior
      popfd
;-----------------------------------
; Comprobación de Pentium !!!
; Se comprueba el bit 21 de EFLAGS, bit ID, que el 486 lo fuerza a 0
;-----------------------------------
      pushfd
      pop     eax
      mov     ebx,eax           ;ebx = EFLAGS
      xor     eax,200000h       ;se bascula el bit 21 (ID)
      push    eax               ;se introduce EAX
      popfd                     ;en EFLAGS
      pushfd                    ;ahora al revés, desde EFLAGS
      pop     eax               ;hacia EAX
      push    ebx               ;se restaura el valor original
      popfd                     ;de EFLAFS
      and     eax,200000h       ;se ponen a cero todos los bits
                                ; menos el 21
      and     ebx,200000h       ;lo mismo
      cmp     eax,ebx           ;¿son iguales?
      je      CPUID_1           ;sí, no es un Pentium
                        ;no, entonces puede ser un Pentium
                        ;o superior, pero también puede ser
                        ;un 486DX con soporte (no documentado)
                        ;de la instrucción CPUID
;.586
      push    dx
      push    ecx
      xor     eax,eax
      CPU_ID
      .IF     (EAX >= 1)
            mov     DPT msgcpuid,ebx
            mov     DPT msgcpuid[4],edx
            mov     DPT msgcpuid[8],ecx
            CPU_ID
            mov     infcpuid,eax
            mov     infcpuid[4],edx
      .ELSE
            pop     ecx
            pop     dx
```

```
            jmp     CPUID_1
      .ENDIF
      pop     ecx
      pop     dx
      and     eax,00000F00h
      .IF     (AX >= 0500h)
            inc     dh
            mov     dl,dh
            mov     ax,dx
            ret
      .ENDIF

CPUID_1:
      mov     esp,ecx         ;se restaura el puntero de stack
original
;-----------------------------------
;ahora se comprueba el tipo de coprocesador
;en principio se debe usar la forma "no-wait" de las instrucciones
;del coprocesador, hasta que se sepa con certeza de su existencia.
;-----------------------------------

.8086

CPUID_2:
      fninit                  ;Reset del coprocesador
;palabra de control = 037Fh (si existe)
      fnstcw  WPT NDP_STATUS   ;escribe la palabra de control
      cmp     BPT NDP_STATUS[1],03h
      jne      CPUID_3

      mov     dl,dh
      cmp     dh,3
      jne     CPUID_3

      fld1
      fldz
      fdiv
      fld     st
      fchs
      fcompp
      fstsw   NDP_STATUS
      mov     ax,NDP_STATUS
      sahf
      jne     CPUID_3
      dec     dl

CPUID_3:
      mov     ax,dx
      ret
IdentificaCPU   ENDP

_TEXT   ENDS

      END
```

En el registro ax se almacena los diferentes tipos, así tenemos:

AL: Copro. Dependiendo del valor puede ser:

0: 8086

1: 8087

2:8287

AH**:** tipo de CPU. Dependiendo del valor puede ser:

1: 8086

2: 286

3: 386

4: 486

5: Pentium

3.2.- El Copro.

Con el 8086 se sacó el coprocesador numérico 8087, que es una unidad auxiliar estrechamente relacionada con la CPU y que amplia el repertorio de instrucciones con algunas posibilidades muy específicas, como operaciones con senos, cosenos y en punto flotante.

Posteriores al 8087 se sacaron el 287, 387, 486. A partir del 486 se cambia la filosofía, integrando el copro dentro de la CPU.

La interacción entre la CPU y el coprocesador cuando este ejecuta una instrucción:

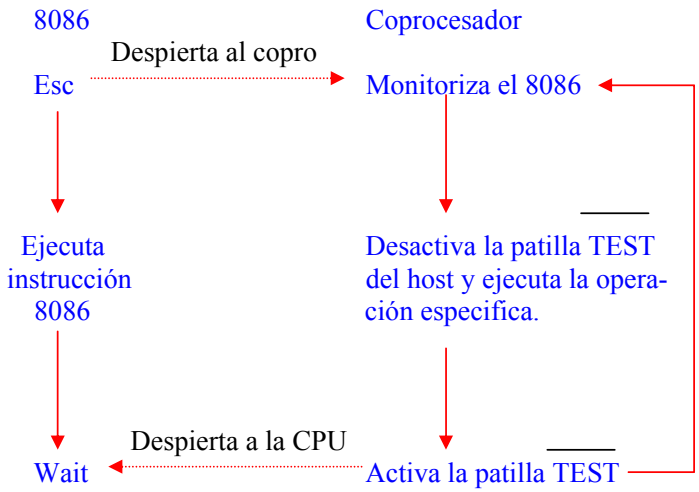


Figura 28. Esquema de interacción entre CPU y coprocesador.

Sólo la CPU puede buscar instrucciones, pero el copro va mirando esas instrucciones hasta que ve una instrucción para el (ESC), esta la decodifica al igual que la CPU, el copro desactiva la patilla test de la CPU y ejecuta la operación, la CPU sigue haciendo lo propio.

Si en este periodo la CPU quiere utilizar otra vez el copro tendrá que ejecutar una instrucción WAIT y esperar hasta que su patilla de test sea activada por el copro.

Tipos de datos del copro.

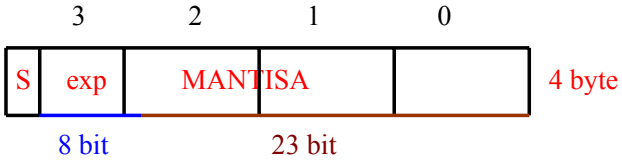
Internamente solo tiene una representación de datos, son registros de 80 bit, pero al leer o escribir en memoria lo hace de diferentes formas:

Enteros
- 16 bits (sword) → -32.768 .. 32.7
- 32 bits (sDword) → -2.147.483.648 .. 2.147.483.647
- 64 bits (sQword) → -2E63.. 2E63 - 1

BCD Empaquetado 80 bit (T byte)

Reales
- Single 32 bit
- Double 64 bit
- Extended 80 bit

Single:



El exponente está en exceso 127.

$V = (-1)^S * 1.m * 2^{e-127}$ si $0 < e < 255$, números normales

El rango es de 10E-38 y 10E38 con 6 dígitos decimales de precisión.

$V = (-1)^S * 0.m * 2^{-126}$ si $e = 0$ y $m <> 0$, número desnormalizado.

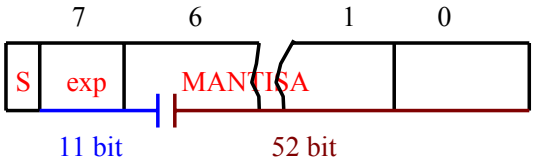
$V = 0.0$ si $e = 0$ y $m = 0$

$V = \infty$ si $e = 255$ y $m = 0$, el número puede ser $\pm\infty$

V = No es un número NaN (Not a Number), si $e = 255$ y $m <> 0$

Las cuatro primeras representaciones las puede devolver el copro como resultado pero nunca dará un NaN, si se da un overflow lo convierte a ∞.

Double:



El exponente en exceso 1023

$V = (-1)^S * 1.m * 2^{e-1023}$ si $0 < e < 2047$, normalizada con 16 cifras decimales de precisión.

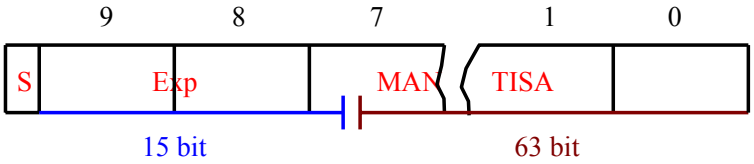
$V = (-1)^S * 0.m * 2^{-1022}$ si $e = 0$ y $m <> 0$, número desnormalizado.

$V = 0.0$ si $e = 0$ y $m = 0$

$V = \infty$ si $e = 2047$ y $m = 0$
$V = NaN$ si $e = 2047$ y $m <> 0$.

Extended:

Es como realmente representa los números el copro en sus registros.



$V = (-1)^S * 1.m * 2^{e-16383}$ si $0 < e < 32767$

$V = (-1)^S * 0.m * 2^{-16382}$ si $e = 0$ y $m <> 0$

$V = 0.0$ si $e = 0$ y $m = 0$

$V = \infty$ si $e = 32.767$ y $m = 0$

$V = NaN$ si $e = 32.767$ y $m <> 0$

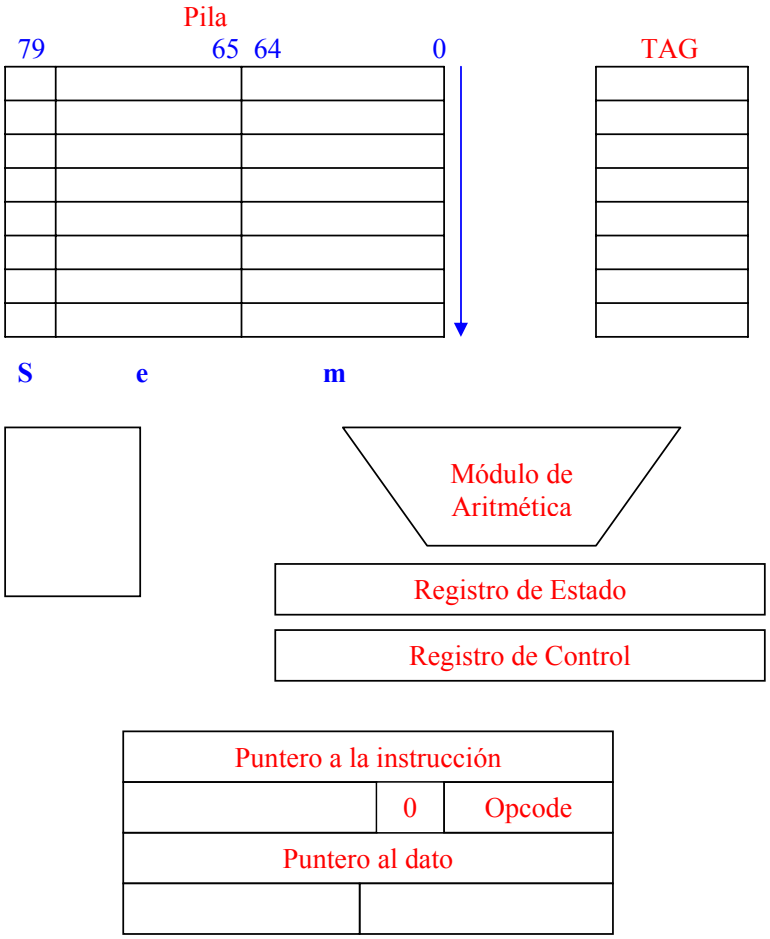
NaNs:

Según la mantisa, podemos distinguir entre:

* QNaN (callados) (11.....1, 10.....0), la mantisa empieza con 1

*SNaN (enteros) (011.....1, 010.....0), la mantisa empieza con 0

Estructura Interna



Los registros de operaciones forman una pila que crece hacia abajo.

Existe un campo de cima de la Pila en el Reg. status que nos determina el top de la pila, esta es dinámica y los operaciones se realizan sobre la cima de la pila.

Palabra de estado.

- Del bit 0 al 8 son los flag de excepción, se ponen a 1 si se produce una:

IE (Operación Invalida): se produce al intentar meter más de 8 valores, también puede saltar con otras operaciones que no se pueden realizar.

DE (Operando normalizado): si uno de los operandos esta desnormalizado se pone este bit a 1.

ZE: división por cero.

OE: overflow en el exponente.

UE: underflow en el exponente.

PE: cuando no se puede dar una precisión coherente con el formato de entrada prefijado.

SF (Stack Fault): fallo de pila, según sea C1 puede ser:

0 overflow
1 underflow

IR: se pone a 1 si alguno de los flag de excepción se pone a 1

C3,C2,C1,C0: códigos de condición, se ven afectados por algunas operaciones del copro.

ST: puntero a la cima de la pila.

B: bit de ocupado, está a 1 si la anterior operación no ha terminado todavía.

fxam: comprueba el valor del registro de la cima de la pila, según este ese valor fija los 4 bit de la cima de la pila (C3, C2, C1, C0), estos valores pueden ser:

0 → el número examinado es positivo y anormal.

1→ el número examinado es positivo y se produce NaN

2 → el número examinado es negativo y no normal

3 → el número examinado es negativo y se produce NaN

4 → el número examinado es positivo y normal

5 → + ∞

6 → el número examinado es negativo y normal

7 → - ∞

8 → + 0.0

9 → Vacío

A→ - 0.0

B → vacío.

C → el número examinado es positivo y denormal

D → vacío.

E → el número examinado es negativo y denormal

F → vacío.

Después de fcom o ftst, quedaría de la siguiente forma:

C3	C2	C1	C0	
0	0		0	st > op. fuente y st > 0
0	0		1	st < op. fuente y st < 0
1	0		0	st = op. fuente y st = 0
1	1		1	número no comparable.

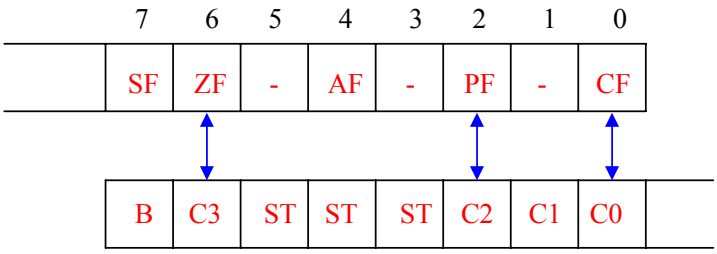
El bit C1 no se mira.

Otra operación que afecta a estos bits es la fprem:

C3	C2	C1	C0	
q1		q0	q2	q2, q1, q0: son los 3 bits menos significativos del cociente que se está reduciendo.

no se utiliza.

La posición caprichosa de estos bits tiene su razón de ser:



Se alinea el byte mas alto del flag del copro con el más bajo de la CPU.

CF { 0 jae | jnb | jnc ; 1 jb | jc | jnae }
PF { 0 jnp | jpo ; 1 jp | jpo }
ZF { 0 jne | jnz ; 1 je | jz }

Los bits provienen del copro pero se meten en los flag de la CPU para poder devolver esos flags y poder producirse saltos, que es la única que puede producirse.

Si CF = 0 y ZF = 0 → ja | jnbe

Si CF = 1 y ZF = 1 → jbe | jna

No existe una instrucción que ponga directamente esos flag en la CPU:

287 {
fstsw: lee el estado del copro y lo pone en memoria
mov ah byte ptr [estado + 1]
sahf copia ah al registro de flag de la CPU
}

387+ {
Las dos primeras instrucciones se agrupan en una.
fstsw ax carga la palabra de estado en cx
sahf
}

FP_STATUS MACRO

push ax
fstsw ax
sahf
pop ax

Esto iría en sitios como

fcom
FP_STATUS
jp no_comparable
jbe

ENDM

Este registro es de solo lectura.

Palabra de Control.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			IC0	RC	RC	PC	PC	IEM		PM	UM	CM	7M	DM	IM

Los bits de 0..5 son flag de excepción, es decir, máscara de flag. Si ponemos un 1 en alguno de ellos enmascara a la excepción correspondiente, es decir, no nos avisaría de esa excepción.

IEM: máscara de la habilitación de la interrupción, a partir del 287 se ignora este bit.

PC (control de precisión): según el valor que toma, puede tener el siguiente significado:

0 0 → 24 bit (Float).

0 1 → reservado.

1 0 → 53 bits (Dfloat).

1 1 → 64 bits (por defecto).

RC (control de redondeo): según el valor que toma, puede tener el siguiente significado:

0 0 → redondeo al mas próximo o par.

0 1 → redondeo inferior.

1 0 → redondeo superior.

1 1 → trunca el valor hacia 0.

IC (control de infinito): si es un 0 es infinito proyectivo y 1 solo un infinito (afin) distingue entre ± ∞.

A partir del 387, solo existe el afin, cualquier número es menor que + ∞ y mayor que - ∞.

Se puede leer de forma especifica cuando se extrae el contenido del copro, existe una instrucción que saca todo el contenido del copro, esto sirve para multiprogramación, pues se vacía el copro, se realiza otra operación y luego se puede volver a realizar esa misma tarea volviendo a cargar el copro.

Instrucciones del copro:

Comentario: El conjunto de instrucciones del coprocesador matemático hay que exponerlo mejor.

Transcendentales

FSIN : calcula seno st(0) ← sen st(0)

FCOS: calcula coseno st(0) ← cos st(0)

FPTAN:

st(0) = x
st(1) = y

} resultado donde tg st(0) = x / y , el ángulo debe estar en st(0)

FPATAN: st(0), arctang (st(1) / st(0)). No hay restricciones en st(0) y st(1).

F2XM1: 2^x - 1. St(0) $\leftarrow$ $2^{st(0)} - 1$.

En el 8087 y 80287 el rango es de $0 \leq st(0) \leq 0.5$

En el 387 y siguientes el rango es de $-1.0 \leq st(0) \leq +1.0$

FYL2X: calcula $\rightarrow$ st(0) = st(1) * Log [st(0)].

FYL2XP1: Calcula $\rightarrow$ st(0) = st(1) * Log [1 + st(0)]. St(0) tiene que ser positivo.

Comparaciones

FCOM OP: compara st(0) con el operando

FCOMP: afecta a los bit C0, C2 y C3, los compara restando a st(0) el operando.

FCOMPP: después de comparar extrae st(0) y st(1).

FICOM OP: es un entero almacenado en memoria, compara integer.

FICOMP: compara un entero (integer) y extrae del alto de la pila.

FUCOM:

FUCOMP:

FUCOMPP

FTST: test real

FXAM: examina un real.

Carga de constantes

FLDZ: carga un 0.0

FLD1: carga un 1.0

FLDPI: carga pi (π).

FLDL2T: carga Log 10 $\approx$ 3.321928…..

FLDL2E: carga Log e $\approx$ 1.442695…….

FLDLG2: carga Log 2 $\approx$ 0.3010299956

FLDLN2: carga Log 2 ≈ 0.09314718……

Instrucciones Aritméticas

FABS: Valor absoluto.
FCHS: Cambia el signo.
FSQRT: raiz cuadrada.
FRNDINT: redondeo a un entero según la forma de redondeo del registro de control. El resultado es un entero y se guarda en ST(0), en formato de punto flotante.

FSCALE: multiplica o divide un número por una potencia de 2, lo que hace es coger el valor de st(1), lo redondea al entero más próximo de menor magnitud, se lo suma a la parte de exponente en st(0).

t = ROUND (st(1), chop) → trunca hacia 0
if t ≠ then
st(0) = st(0) * 2^t

Es más rápido que una multiplicación, no se modifica st(1) ni st(0), aunque el resultado puede ser overflow o underflow del exponente, teniendo que desplazar la mantisa si es underflow o colocar en el exponente ∞ si es overflow.

FXTRACT: separa la parte de mantisa y la parte de exponente de st(0). Después de esta orden es st(0) estará la mantisa y en st(1) el exponente. Si después se realiza un fscale se recupera el número anterior.

FDECSTP: puede afectar a C1 del registro de estado, si hay un underflow de la pila C1 = 0. Se produce una excepción de resultado inexacto (PE, siempre se produce cuando hay redondeo). C1 = 0 si el redondeo es hacia arriba o C1 = 1 si el redondeo es hacia abajo.

FPREM: calculan el resto parcial, de los dos números de la cima de la pila (st0, st1), los resta y lo devuelve almacenando el resultado en ST0.
Si st(0) < | st(1)| * 2^{64}, la reducción es completa y C2 = 0. Si no se cumple C2 = 1 y la reducción es incompleta y se repite de nuevo.

FPREM1: no busca el cociente exacto sino el más aproximado de forma que:

| [cociente – st(0) / st(1)] | < ½

Todas estas instrucciones afectan a las excepciones.

Instrucciones de control del copro.

FSTSW / FNSTSW → por si no se puede ejecutar en copro o no existe
↳ lee la palabra de estado

Las instrucciones de control con la "n" son consideradas como una nop (no-operación) por la CPU si no existe /está desactivado el copro.

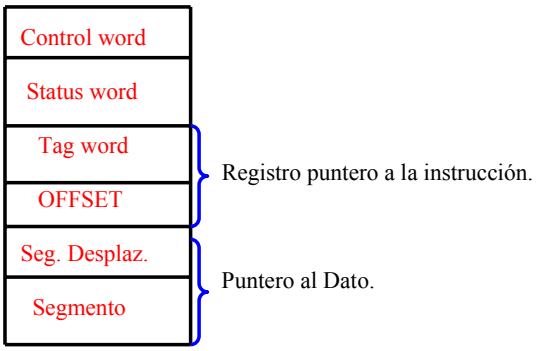
FSTCW AX / FNSTCW AX: lee la palabra de estado y la almacena en ax, en algunos ensambladores se puede poner fstswax.

FLDCW MEM16: escribe en esta memoria el registro de control.

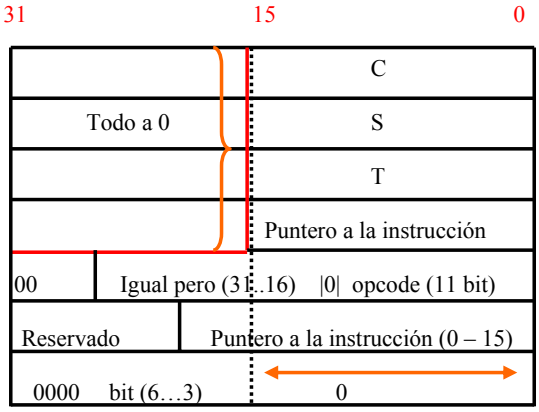
FSTCW / FNSTCW MEM 16: lee la palabra de control de mem 16.

FINIT / FNINIT: inicializa el copro, hace un reset del copro. Solo al principio de un programa, es recomendable no hacer abuso de esta instrucción.

fstenv / fnstenv
fldenv
mem → señala a un área de memoria que depende si trabajos en modo de 16 bits o de 32 bits, donde se guarda el estado del copro de forma abreviada.



En modo de 32 bits son 7 double word:



FSAVE / FNSAVE MEM: (SALVA) guarda todo el contenido del copro en el área apuntada por mem.

FRSTOR MEM: restaura el contenido del copro. La memoria puede ser:

Modo 16 bits → 54 bytes
Modo 32 bits → 108 bytes.

Estas dos instrucciones anteriores están pensadas para entornos multitarea, se salva uno de ellos y luego se ejecuta la otras, …..

Si hay interrupciones del BIOS o del Copro se salva todo el estado del copro.

FCLEX / FNCLEX: borra los flag de excepción del registro de estado y borra el bit busy del registro de estado y el IR5 de petición de interrupción. Pensada para que sea los primero que haga antes de realizar otra tarea después de una excepción.

FDECSTP: decremento
FINCSTP: incrementa
} modulo 8, el puntero de la cima de la pila del registro de estado.

FFREE: st(i), marca como vacío, en los Tag correspondientes al argumento.

FWAIT: sincronizar la CPU y el Copro, para la CPU hasta que el copro termine.

FNOP: no operación, es igual a la instrucción fst st(0), st(0) es igual pero con un pequeña salvedad, si st(0) es igual a vacío, fst da una excepción.

En los sistemas PC cuando se produce una excepción se produce la interrupción Bh.

Si se enmascara las excepciones del copro nunca se nos colgara el copro, (decisiones por defecto), si no se tienen enmascaradas, la rutina de atención a la interrupción decidirá lo que debe hacer.

Rutina para calcular la exponencial de un número.

Comentario: En futuras versiones iré cambiando las rutinas que aquí aparecen por las mías originales.

$y = e^x$, dado x suponemos que entra en la cima del copro y el resultado lo devuelve en st(0)

$y = e^x \rightarrow \text{Log } y = \text{Log } e^x = x * \log e \rightarrow y = 2^{\log y} \rightarrow 2^{x * \log e}$.

```
EXP proc

    call TRUNCA    ;poner en modo truncación
    fldl2e    (coloca log e en la pila)
    fmul    ;  st(0) = z
    fld st(0)   ; se duplica
    frnd int  ; se trunca
    fxch
    fsub  st(0), st(1)
    f2xm1
    fld1
    fadd
    fscale  ;multiplica st(0) * 2 E (st(1))
    fstp st(1)
    ret

EXP endp
```

st(0)	st(1)	st(2)
x		
log e	x	
z		
z	z	
[z]	z	
z	[z]	
{z}	[z]	
2E{z}-1	[z]	
1	2E{z}-1	[z]
2E{z}	[z]	
e	[z]	
y		

Rutina para el cálculo de una expresión racional:

$$R(x) = \frac{a0 + a1 \cdot x + \ldots\ldots + an \cdot x}{b0 + b1 \cdot x + \ldots\ldots + bm \cdot x}$$

El valor de x se da en st(0), donde también se va a devolver el resultado (R(x)), también devolvemos el bit de acarreo del registro de flag.

```
N  EQU  <valor de a>
M EQU <valor de m>
A  DQ  (N + 1) DUP (?)
B  DQ  (M + 1) DUP (?)
……..

Polinomio proc

    ; x  = st(0)
    ; P(x) = st(0)

    mov si, N
    shl si, 3          ; si = n * 8
```

```
            fld qword ptr A[si]

        @_P1:

            sub si, 8
            fmul st(0), st(1)
            fadd qword ptr A[si] * B[si]
            stc fin
            or si, si
            jnz @_P1
            clc
        fin:
            ret

        Polinomio Endp
```

Un esquema puede ser el siguiente:

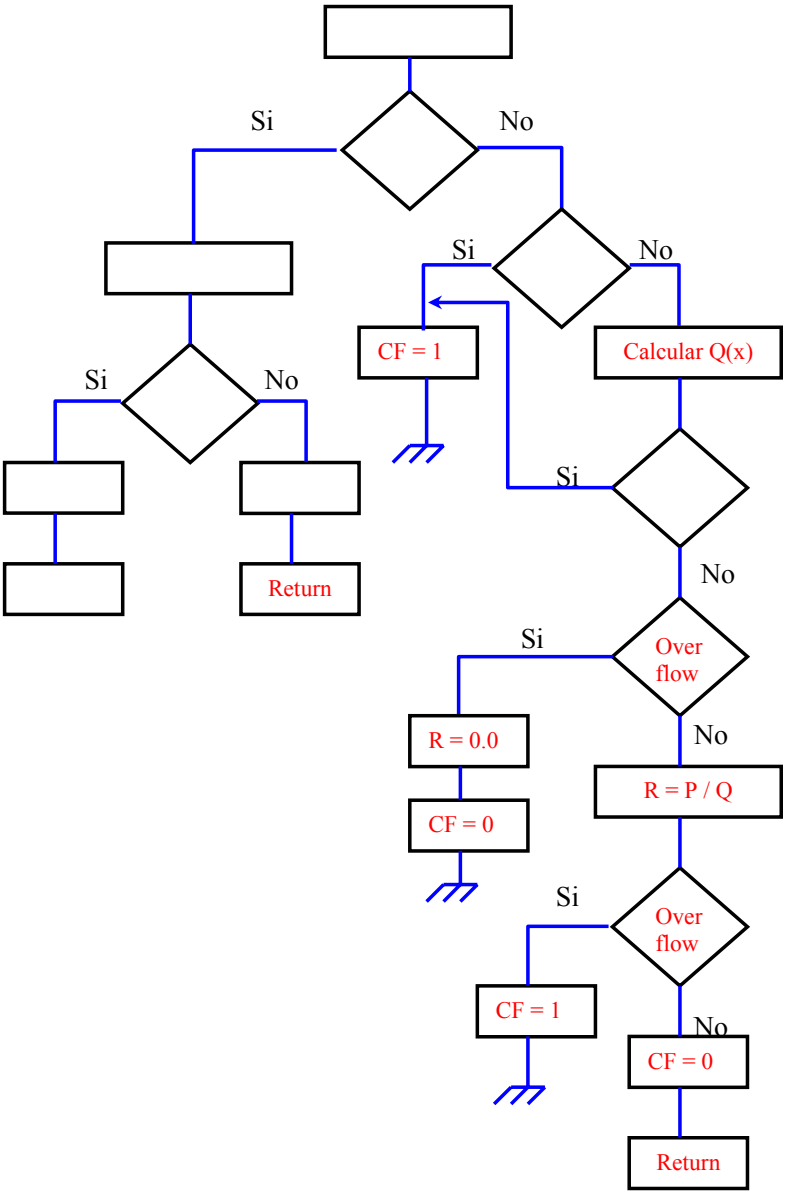


Figura 29. Diagrama de flujo del cálculo de una expresión racional.

Cálculo de un número factorial.

```
Factorial Proc

    ; Entrada AX = n
    ; Salida EAX = n!

    push ebx
    push edx
    cbd    ; convierte la palabra ax en doble palabra en Eax)
    mov ebx, eax

  @_f1:

    dec ebx
    cmp ebx, 1
    je @_f2
    mul ebx
    jmp @_f1

  @_f2:

    pop edx
    pop ebx
    ret

Factorial endp.
```

Calculo de una potencia X (n > 0)

```
Potencia Proc

    ; en st(0) = x
    ; en AX = n

    st(0) = X
    push ax
    mov cx, ax
    fld1

  @_1:

    shr cx, 1
    jc @_3

  @_2:   ; exponente par

    fld st(1)
```

```
        fmulp st(2), st(0)
        jmp @_1
@_3:    ; exponente impar

        fmul st(0), st(1)
        cv  cx, cx
        jnz @_2
        fstp st(1)
        pop cx

        ret

Potencia endp.
```

Al comienzo de un programa:

Comentario: Quiere decir que esto es un esqueleto básico de un programa en ensamblador

```
.model  small
.stack   800h    ; 2K

.data
        Variables que se necesitan
        ……

.code
        assume  CS: @code

inicio:

        mov ax, @data
        mov dx, ax
        ; código del usuario
        ……
        mov al, número   ;error que devuelve el programa.
        mov ah, 4ch
        int 21h

        subrutinas
        ……

        End inicio.
```

Herramientas imprescindibles

Convertir un número entero sin signo a una cadena ascii.

```
Bin_1_Ascii  proc near
```

```
                ;  entrada AL = número a convertir
                ;            DS : DI = puntero a la cadena ascii
                ;  salida DS: DI = puntero a la cadena convertida en ascii.
                ; ax = número de cifras decimales


        LOCAL CifrDec: word

        push bp
        mov bp, sp
        sub sp, 2
        push bx  cx  dx  di  es
        push ds
        pop es
        cld      ; contador hacia adelante
        mov bx, 10
        xor ah, ah
        xor cx, cx

@@_1:

        xor dx, dx
        div bx
        push dx
        inc cx
        or ax, ax
        jnz @@_1
        mov CifrDec, cx

 @@_2:

        pop ax
        or al, 0   ; or con el número 0 se transforma a ascii
        stosb      ; lo guarda en cs:di como un string
        loop @@_2
        xor al, al
        stosb
        mov ax, CifrDec
        pop es  di  dx  ex bx
        mov sp, bp
        pop bp
        ret

 Bin_1_Ascii  endp.



 SBin_2_Ascii

        ; entrada ax = no a convertir
```

```
;          ds: di = puntero a  la cadena
;  salida = ds: di,  ax = no cifras.


        push di
        or ax, ax
        jnz @@_1
        mov byte ptr [di], ‘  ‘
        jmp @@_3

@@_1:

        jl @@_2
        mov byte ptr [di], ‘+’
        jmp @@_3

@@_2:

        mov byte ptr [di], ‘-‘
        neg ax

@@_3:

        inc di
        call Bin_2 Ascii
        inc ax
        pop di

        ret

Sbin_2_ascii endp.

fpt2a  proc near

        ; convierte de punto flotante a ascii
        ; entrada: st(0),  nº a convertir
        ;              ds: di  puntero a la cadena
        ;              ax = nº de cifras decimales en la conversión
        ; salida:    ds: di, puntero a la cadena
        ;              ax: longitud de la cadena

        Local estado: word, dígito: word ,expo10: word,
        CifrDec : word = LOCALSIZE

        push bp
        mov bp, sp
        sub sp,  LOCALSIZE
        push bx, ax, dx, si, di
        mov si, di
        cmp ax, 19
```

```
        jb @@_1
        mov ax, 19

@@_1:

        mov CifrDec, ax
        fxam
        fstsw ax
        mov bl, ah
        shr ah, 3
        and ah, 08h
        and bx, 0007h
        or bl, ah                   ; bx = 0C3C2C1C0
        cmp bl, 4  ; Positivo y normal, devuelto por fxam.
        jne @@_2
        WRITE_CHAR '+'
        jmp  @@_8

@@_2:

        cmp bl, 6    ; si el número es negativo
        jmp @@_3
        WRITE_CHAR '-'
        fchs              ; cambia de signo
        jmp @@_8

@@_3:

        cmp bl, 8
        je @@_4
        cmp bl, 10
        je @@_4
        jmp @@_6

@@_4:

        fdrop
        WRITE_CHAR '   '
        WRITE_CHAR '0'
        WRITE_CHAR '.'
        mov cx, CifrDec

@@_5:

        WRITE_CHAR '0'
        Loop @@5
        WRITE_CHAR   00h
        jmp @@7

@@_6:
```

```
        fdrop
        shl bx, 1
        mov di, word ptr  mal_número[bx]

@@_7:

        mov al, byte ptr [di]
        inc di
        WRITE_CHAR al
        or al, al
        jnz @@_7
        jmp @@_Z

@@_8:

        mov word ptr expo10, 0

@@_9:  ; si es menor que 1 multiplicamos por 10.

        fld 1
        fcomp
        FP_STATUS
        jc @@A
        pushf
        dec word ptr expo10
        fmul qword ptr diez
        popf

@@_A:

        jnc @@9

@@_B:

        fcom qword ptr diez
        FP_STATUS
        jc @@C
        pushf
        inc word ptr expo10
        fdiv qword ptr diez
        popf

@@_C:

        jnc @@B
        fld qword ptr mitad
        mov ax, CifrDec

@@_D:
```

```
        fdiv qword ptr diez
        loop @@_D

@@_E:

        fadd
        fcan diez       ; si nos hemos salido de rango
        FP_STATUS
        jc  @@_F
        inc word ptr expo10
        fdiv qword ptr diez

@@_F:
        call Trunca  ; poner en modo de truncamiento
        EXTRAE_DIGITO  ; extrae el primer dígito
        WRITE_CHAR '.'
        jmp @@_H

@@_G:

        EXTRAE_DIGITO
        dec CifrDec

@@_H:

        cmp CifrDec, 0
        jnz @@_G
        fdrop
        cmp exponente\0, 0
        jz @@_I
        WRITE_CHAR 'E'
        mov ax, expo10
        mov di, si
        call Sbin_2_Ascii
        jmp @@_7

@@_I:

        WRITE_CHAR 00h

@@_Z:

        pop di, si , dx, cx, bx
        mov sp, bp
        pop bp

        ret

fptoc endp
```

```
WRITE_CHAR MACRO char

        mov byte ptr[si], char
        inc si
ENDM.

EXTRAE_DIGITO MACRO

        fdmp
        frnd int
        fsub st(1), st(0)
        fistp word ptr dígito
        fmul qword ptr diez
        mov ax, dígito
        add al, 0
        WRITE_CHAR al

ENDM.


DATA

        diez  DQ   10.0
        mitad DQ  0.5
        Nmn00  DB  "No soportado"
        Nmn01 DB  " + NanN"

        ……

        NmnbF  DB  " ……."
        mal_número  DW  offset Nmm00
                        DW  offset  Nmn01


Dchar proc near    ; muestra el contador de al

        push ax, bx
        xor bh, bh
        mov ah, 0Eh
        int 10h
        pop bx, ax

        ret

Dchar endp
```

```
Dstring proc near

        ; Entrada DS : SI

        push si, ax

@@_1:

        mov al, byte ptr [si]
        or al, al
        jz @@_2
        call Dchar
        inc si
        jmp @@_1

@@_2:

        pop ax, si

        ret

Dstring endp.
```

Tema 4

Los buses del PC

4.1.- El bus ISA.

Se mantiene el conector de 62 contactos, 31 por cada lado. Se añade una extensión de 36 contactos más, 18 por cada lado.

Las 20 primeras líneas son de direcciones en el *slot* largo y las restantes en el de expansión.

Los pines más significativos son:

$\overline{\text{SBHE}}$: se incorporó desde el 286 para indicar el tamaño del acceso, es decir, si el acceso es a un word o a un byte. Si es una word y esta alineada se puede extraer directamente y sino hay que hacer accesos.

$\overline{\text{MEMCS16}}$: se activa en cualquier acceso que vaya a hacer la CPU a memoria o a entrada-salida. Es un pin de extensión del bus.

$\overline{\text{IOCS16}}$: se habilita si es un acceso a entrada-salida. Se inhabilita si es a memoria.

D0-D7: bus de datos de 8 bit.

$\overline{\text{MEMR}}$: se activa si se lee en memoria.

$\overline{\text{MEMW}}$: se activa si se escribe en memoria.

$\overline{\text{IOR}}$: se activa si se lee o en periféricos de entrada-salida.

$\overline{\text{IOW}}$: se activa se escribe en periféricos de entrada-salida.

ALE: (Address Latch Enable): hace que se almacenen las direcciones de los datos de un bus, debido a que existe multiplexación en el tiempo, entre las direcciones y datos.

OSC (14,31818): frecuencia del cristal que se puso originalmente en el PC. Se saca directamente del oscilador. No esta sincronizada con ninguna otra señal y está a disposición del usuario.

CLOCK: señal del reloj del bus. Se especifica a partir del A1, y es de 8 MHz.

$\overline{\text{IOCHCK}}$: señal de entrada hacia el bus. Si la tarjeta o periférico tiene un error de hardware se le hace saber por medio de esta señal.

IOCHRDY: es una señal de estados de espera.

IRQ2—IRQ7: petición de interrupción

IRQ8: se reserva para la placa base.

IRQ13: se reserva para la placa base.

IRQ9: se monta con la dos.

Ciclo de Lectura.

Un ciclo de lectura son dos periodos completos empezando por el flanco de subida.

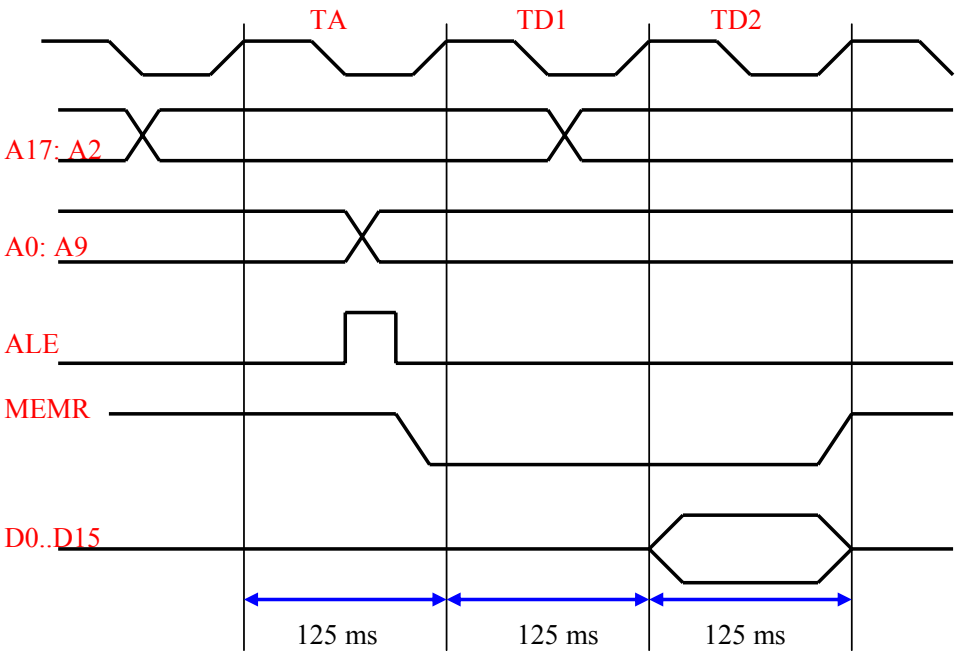


Figura 1. Esquema del Bus ISA.

365 ns es el tiempo mínimo de un acceso con bus ISA.

Tema 6

Periféricos de placa base

6.1.- Decodificación de los periféricos de placa base clásicos

Comentario: Insertar aquí el archivo **decod_pb.wmf** con la figura que detalla la decodificación de los periféricos de la placa base

El decodificador 74138.

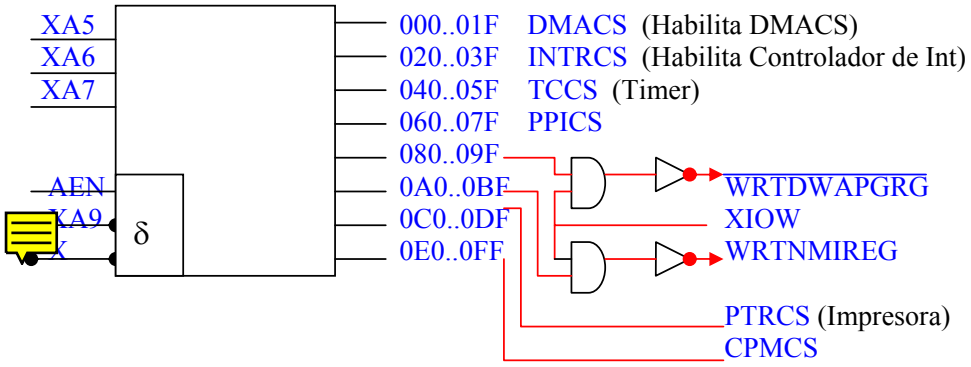


Figura 12. Decodificador 74138.

Circuito de la Señal AEN: (Addres Enable).

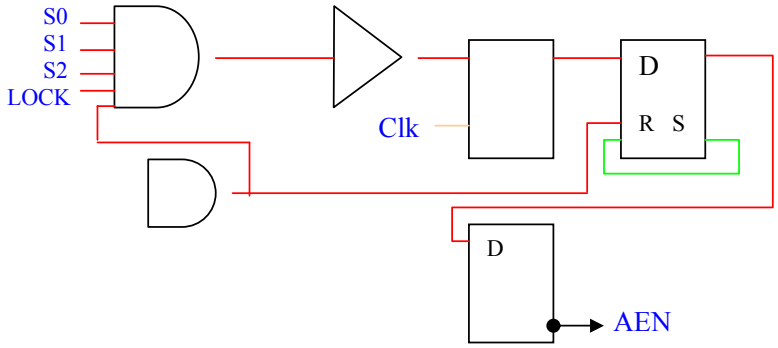


Figura 13. Circuito de la Señal AEN.

Codificación de procesos en placa base.

Tenemos que tener en cuenta que sólo las 10 líneas más bajas intervienen en la decodificación.

10	9	8	7	6	5	4	3	2	1	0	
X	0	0	0	0	0	X	X	X	X	X	0000-001F
X	0	0	0	0	1	X	X	X	X	X	0020-003F1024
X	0	0	0	1	0	X	X	X	X	X	0040-005F
X	0	0	0	1	1	X	X	X	X	X	0060-007F
X	0	0	1	0	0	X	X	X	X	X	0080-009F
X	0	0	1	0	1	X	X	X	X	X	00A0-00BF
X	0	0	1	1	0	X	X	X	X	X	00C0-00DF
X	0	0	1	1	1	X	X	X	X	X	0BE0-00FF

Las 256 primeras están agrupadas en bloques de 32 y a c/u de estos bloques se les asigna un dispositivo de placa base. Casi ningún dispositivo utiliza los 32. El DMA es el que más utiliza, éste utiliza 16 puertos.

Una limitación que tiene todos lo PCs es que se activa independientemente de los valores de la líneas de direcciones de arriba, porque sólo se usa las diez más bajas.

En el AT, por compatibilidad, se siguen decodificando las 10 líneas más bajas. En el AT y sucesivos se adjudicó de la siguiente manera:

* DMA1CS: Toma desde 00-0Fh

* INTR1CS : Controlador de interrupciones 20-21h

* T/CCS: Timer 40-43h

* PPICS: Para la PPI, aunque ya no existe.

* PCREGCS: Registro de páginas 80-8Fh.
* INTR2CS: Segundo controlador de interrupción A0-A1h.

* DMA2C5: Segundo DMA usa: C0-Deh, pero sólo los pares.

* CS287: Copro. F0, F1, F8, FA, FC, posteriormente el copro se ha integrado en el procesador por lo que se ha llevado a otras puertas fuera de este rango.

El rango 0100-03FFh es el rango de puertos que se utilizan para periféricos externos, ejemplo: buses externo, VGA, etc.

Las 256 primeras direcciones están reservadas para periféricos de placa base y el resto hasta 1024 para periféricos externos.

En el AT desaparece la PPICS pero esta línea sirve para ampliar el rango de codificación mediante una ROM (74S288) de 256 bit, que tiene 5 líneas de dirección, de las cuales la más baja (ADA) va la PPICS.

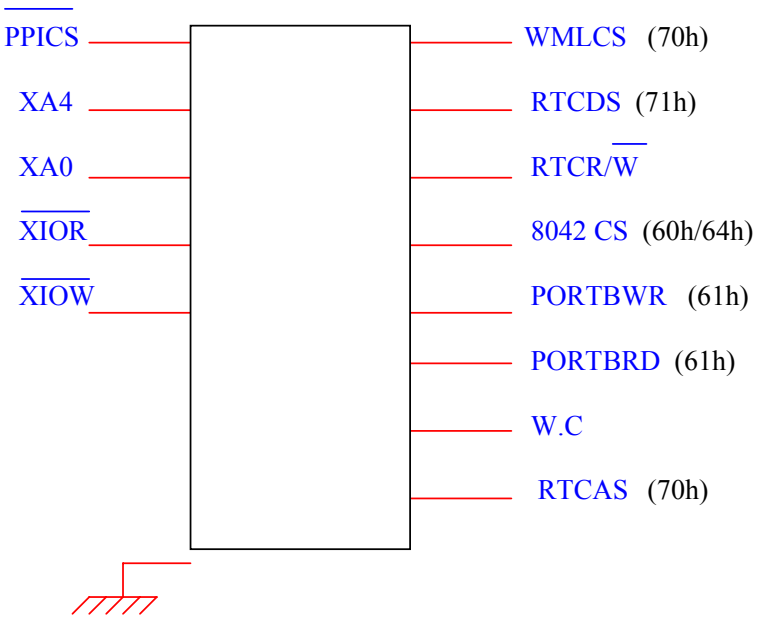


Figura 14. Esquema de un 74S288

Tiene ocho líneas de datos de salida y un habilitador siempre habilitado.

NMICS: habilita la NMI.

RTCDS: Reloj en tiempo real (71h), es de bajo consumo y se mantiene con una pila hasta con el ordenador apagado.

PORTBWR

Simula las funciones de PPICS.

PORTBRD

RTCAS: con esto se mete el controlador de teclado y el reloj.

También se utiliza una PAL para agrupar diversas lógicas que se meten en un chip, que sirve para establecer señales de control del copro del AT.

A partir del AT se incrementa la idea de integrar circuitos en placa base y a partir del 386 todo se mete en *chip-sets* (juego de chips).

6.2.- El controlador de interrupciones.

- PIC Intel 8259A.

Maneja el problema de qué hacer con una petición de interrupción cuando todavía se está atendiendo a otra interrupción anterior.

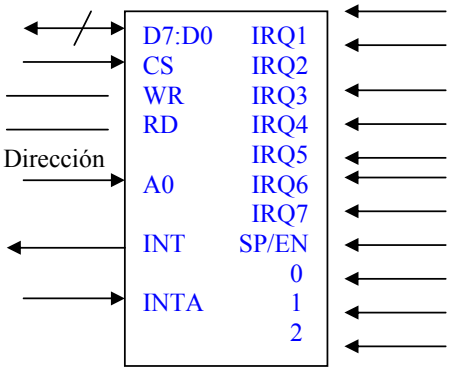


Figura 15. Esquema del PIC Intel 8259A

Es un circuito antiguo de Intel el PIC 8259A, es un chip de 28 pines que tiene 8 líneas de petición de interrupción, 8 de datos, tiene alimentación y masa, línea de habilitación, línea de lectura, etc.

Permite que hasta ocho periféricos le pidan interrupción, él organiza las interrupciones y las pide a la CPU.

Ocupa sólo dos puertos (dos direcciones) distintas, ya que sólo tiene una línea de dirección, A0. La CPU sólo tiene una petición de interrupción (una sola línea), entonces solo hay un periférico o una cadena de margarita.

El PIC tiene ocho líneas para interrupción, permite conectar ocho periféricos y luego los organiza. Se pueden interconectar en cascada o en árbol varios 8259A. Se enlazan mediante las líneas CAS, todas entre sí.

Realizan tres tareas básicas:

1.- Aumenta el número de dispositivos que pueden solicitar interrupción, hasta ocho por PIC, y además identifica al dispositivo y se lo dice a la CPU.

2.- Al ser programable permite que el programador pueda enmascarar algunas peticiones de interrupciones. Puede ser que se quiera que un dispositivo no interrumpa, entonces se enmascara.

3.- Maneja el problema de qué se debe hacer cuando llegan varias peticiones de interrupción simultáneas o mientras se atiende a una.

Registros: aparentemente solo debería tener dos registros pues sólo tiene una línea de Dirección (A0) y habilitada por CS pero no es así.

IRR: Interrupt Request Register. Guarda las peticiones de interrupción que van llegando. Está colocado a la entrada de las líneas de petición de interrupción.

ISR: In-service Register. Dice qué interrupciones se están sirviendo.

IMR: Interrupt Mask Register. Registro de máscara, podemos poner la máscara para enmascarar las líneas de interrupción que no queremos que interrumpan.

ICW1-ICW4: Initialization Command Words (palabras de inicialización de ordenes). Son cuatro registros. Es donde se programa el 8259 para que este funcione como nosotros queramos.

OCW1-OCW3: Operation Control Words (palabras de control).

Son un total de diez registros y solo se acceden por dos direcciones que serán:

BASE (par)

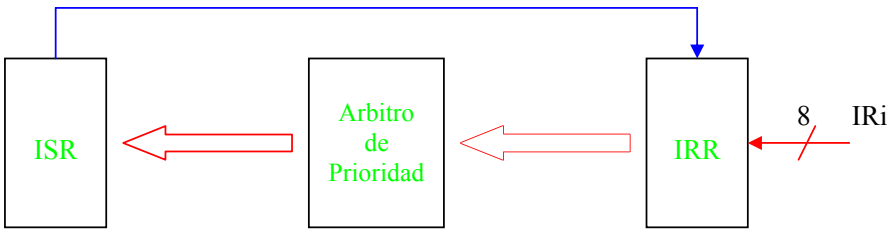
BASE +1 (impar)

Son dos direcciones consecutivas (020h y 021h). Para XT en el AT sería:

PIC1 20h y 21h

PIC2 0A0h y 0A1h

Estructura Interna.



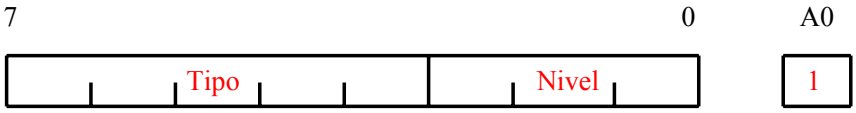
Entran las IRi donde quedan almacenadas y el árbitro las manda a ISR y cuando esta interrupción se pone en servicio entonces se activa la de borrado de petición.

Al ICW1 se accede a través de la dirección par y al ICW2, 3, 4 se accede a través de la dirección impar:

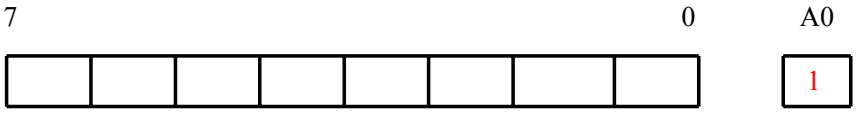
ICW1 (Control de chip):

7							0	A0
			1	LTM	ADI	SNGL		0

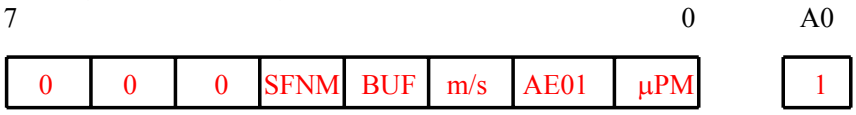
ICW2 (Control del tipo):



ICW3 (Control de Slave):



ICW4 (Control de modo):



1 Modo autofin de interrupción.
0 final de interrupción normal.

								A0
OCW1	IMR							1
OCW2	R	SL	EOI	0	0	L2	L1	L0	0
OCW3	0	ESNM	SNM	0	1	P	RR	RIS	0

Por la dirección PAR (A0 = 0):

Se accede a OCW2
OCW3
ICW1 sólo escritura
ISR } Sólo de escritura
IRR }

Cuando se quiere programar en el PIC se empieza escribiendo en ICW1 y después se escribe en ICW2,3,4.

Por la dirección Impar (A0 = 1):

Se accede a ICW2 }
ICW3 } sólo escritura.
ICW4 }
OCW1 = IMR.

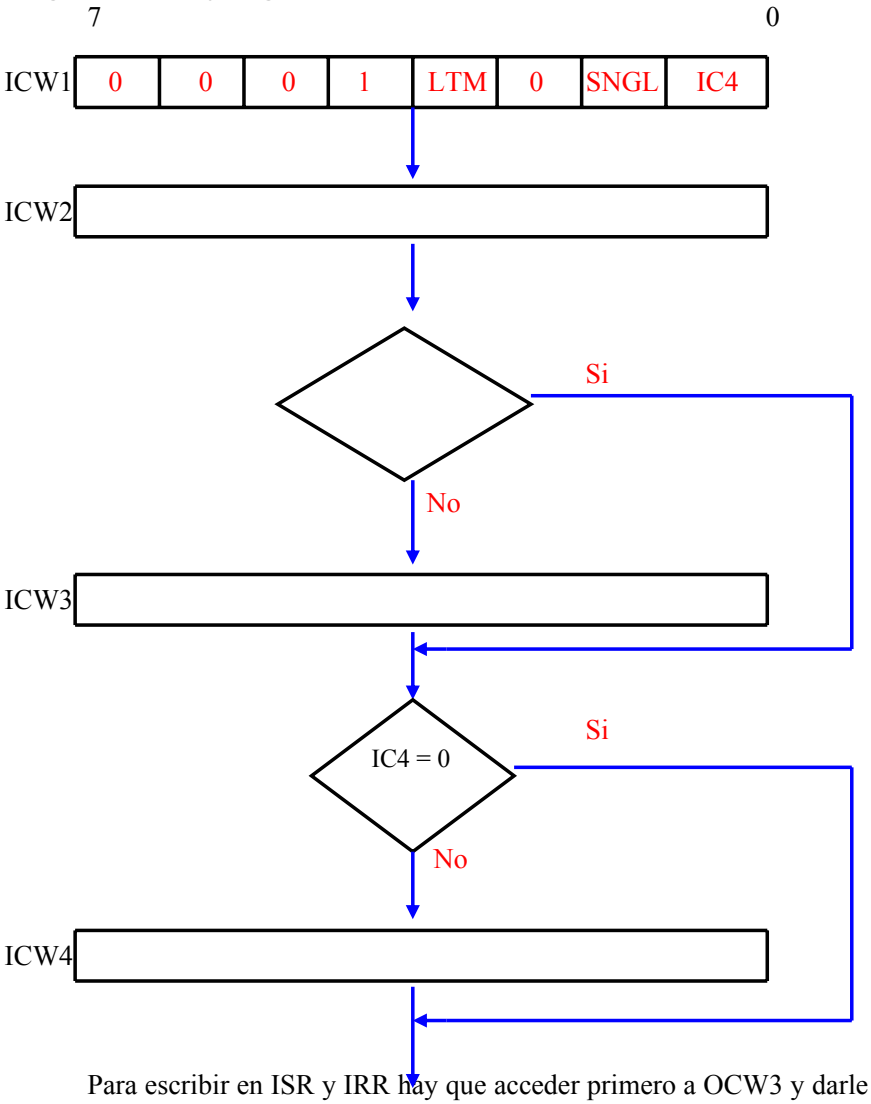
Si queremos escribir en ICW1 forzamos el bit cuatro a uno.

Escribir la dirección par:

- Si el bit 4 = 1 se escribe en ICW1
- Si el bit 4 = 0 se escribe en OCW1 ó OCW3
- Sí el bit 4 = 0
- Si el bit 3 = 0 se escribe en OCW2
- Si el bit 3 = 1 se escribe en OCW3.

Para acceder a las ICW se accede primero al uno y después a los otros. En caso de que no estemos en la programación del chip se accederá al registro de máscara OCW1 = IMR.

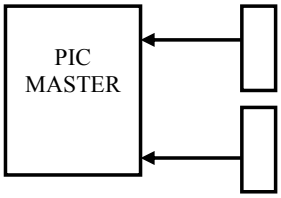
Diagrama de Flujo según los bits de ICW1.



Para escribir en ISR y IRR hay que acceder primero a OCW3 y darle la orden de lectura de ISR y IRR.

Si se quiere leer ISR, se pone 11 en los bits RIS del OCW3, que esta es una orden de lectura.

Puede haber más de un PIC:



Puede haber hasta un máximo de ocho. Hay que programar cada uno de los PIC por separado. Si hay solo un 8259 no se programa el ICW3.

En ICW1: si LT

IM = 0 se reconoce la primera interrupción por flanco de subida.

Si LTIM = 1 se reconoce la primera interrupción por nivel alto ó bajo.

En ICW2: es un vector número que le pasa el PIC a la CPU para decirle que tipo de interrupción está tratando, es programable menos los tres bits bajos que los pone el PIC y es el nivel de la interrupción (línea por la que ha llegado. IRQ0—IRQ7) .

Del bit tres al siete es el tipo de interrupción que es el número que va a la CPU.
En este se indica el tipo de interrupción y determina la dirección del puntero de interrupción.

En ICW3: con un único PIC no se programa y si hay más si se programa diferenciando en que sea master o slave. Se coloca un bit a uno según en qué línea de IRQ se conecte el slave.

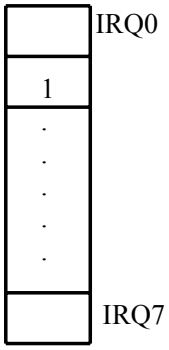
Ejemplo: si por el IRQ1 y IRQ3 están conectados dos slave se coloca un uno en los bit uno y tres del ICW3.

En los slaves, en los tres bits mas bajos 0, 1, 2 se escribe el número de slave que tenga que coincide con la línea del master por la que entra, el resto a cero.

En ICW4: los tres bits superiores a cero y en el bit cero se pone un uno para sistemas de 16 bits. (8086).

Bit 1: **AEOI** (final de interrupción automática), si está a uno indica final de interrupción automática y si está a cero interrupción a mano, esto se debe a que el PIC siempre debe saber cuando ha terminado la interrupción.

En IRR se coló un uno si se solicita una interrupción ya sea por nivel o por flanco.



IRR

Aquí puede haber varios unos simultáneos, a partir de aquí hay un arbitro que decidirá la prioridad de las peticiones de interrupción.

Esta interrupción pasa a la CPU (con INT) y cuando la CPU responda (doble ciclo de lectura) en cada ciclo de lectura activa la INTA del PIC (circuito asíncrono), lo que ve el PIC es dos pulsos de la señal INTA, al llegar la primera sabe que la CPU le ha atendido y en ISR coloca la interrupción más prioritaria y las demás quedan deshabilitadas.

Si el final de interrupción es automático cuando llegue el segundo flanco de subida de INTA hace poner un cero en ISR de final de interrupción, si es manual el final de interrupción lo pondrá el programador.

Bit 2: este bit se ignora si BUF = 0 para sistemas con más de un PIC, en caso contrario sería uno para el master y cero para el slave.

Bit 3: BUF = 1 indica que se va a utilizar SP/EN como salida para desactivar los transceivers (transceptores) del sistema 8086 cuando la CPU acepta datos del 8259A, si estas no existen el BUF debería estar a cero y en sistemas con un solo 8259A debería aplicar un uno a la patilla SP/EN, esto hace que la salida del bus de datos del PIC D7:D0 se queden almacenados en unos *latches*, por si existen conflictos con el bus.

Si no se utiliza un buffer para almacenar D7:D0 (como en los PC) de señal SP/EN se utiliza para determinar si es un slave o master vía hardware, colocando un uno si es master y un cero si es slave.

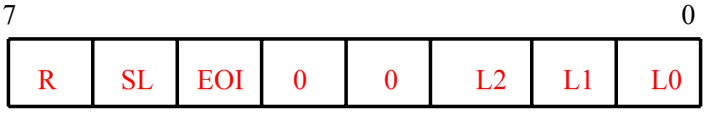
SFNM: bit uno, modo totalmente anidado (tiene sentido si hay más de un 8259, si solamente hay uno no tiene sentido y ni siquiera se mira).

Que esté anidado quiere decir que en ese slave las interrupciones serán menos prioritarias que las que tenga por arriba de donde esté pinchado y más prioritaria que las que tenga por debajo. La preferencia se mira de arriba a bajo.

En el modo no anidado tras atender al slave atiende al master, en vez de al slave.

OCW1: IMR (Registro de Mascara).

OCW2:



Los bit de OCW2 son almacenados por el 8259A solo temporalmente hasta que las acciones especificas se hayan llevado a cabo.

R	SL	EOI	
0	0	1	orden genérica EOI
0	1	1	orden especifica EOI
1	0	1	rotar sobre EOI genérica
1	0	0	establecimiento de la rotación en EOI automática.
0	0	0	borrado de la rotación en EOI automática
1	1	1	rotación sobre EOI especifica
1	1	0	orden de fijar prioridad
0	1	0	no-operación

***EOI*:** instrucción de final de interrupción, si el bit de AEOI de ICW4 está a uno IRS está a uno, que este se coloca de forma automática, pero si AEOI es cero hay que colocarlo de forma manual, el bit cinco de OCW2 puesto a uno realiza esta tarea, esta instrucción se puede realizar de cuatro formas posibles según los bit 7, R (Rotar) y 6 SL (establecer nivel).

Rotación sobre EOI especifica (1 1 1) los da L2-L0 sobre quien rota, la petición normalmente más prioritaria sería IR0.

Bajo la orden genérica EOI (modo normal de prioridad) si ISR está a uno el árbitro de prioridades no reconocerá ninguna petición desde IR7 hacia IR(n+1) pero reconocerá peticiones no enmascaradas desde IR(n-1) hasta IR(6), pudiendo ejecutarse cualquiera de ellas e interrumpiendo la actual.

Si AEOI = 1 el bit de ISR se pone automáticamente a cero al final del segundo pulso de INTA.

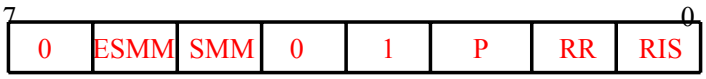
Este modo de prioridad visto anteriormente, OCW2 puede vetar las prioridades mediante la asignación de la prioridad más baja a uno de los niveles IR, el resto seguirán guardando la prioridad con respecto a esta.

Ejemplo: si IR4 más baja, las prioridades quedarían de la siguiente manera:

IR5, IR6, IR7, IR0, IR1, IR2, IR3, IR4
Menor prioridad

Esta rotación se obtiene colocando 10 en los bit R y SL de solo una posición, si se coloca 11, entonces el nivel con menor prioridad es el especificado por L2-L0.

OCW3:



ESMM y SMM: modo especial de mascara, el bit seis lo habilita y se activa o se desactiva con el bit 5.

ESMM	SMM	
0	X	
1	0	Desactivada
1	1	Activada

Este modo sirve para negar los modos de prioridad anteriores si está activada, las peticiones de interrupciones no enmascarables son procesados según orden de llegada y si se desactiva se puede volver a los ordenamientos anteriores (ESMM = 1, SMM = 0).

El bit P: (modo de sondeo) se utiliza para colocar al 8259A en modo sondeo, en este modo asume que la CPU no esta aceptando interrupciones (no activa la líneas de petición de interrupción a la CPU) la CPU debe preguntarle al PIC, esto lo hace accediendo al IRR y explorando las peticiones de interrupción y mirando a IMR (OCW1).

No tiene sentido utilizarlo pues el 8259 esta para que la CPU se despreocupe de esas tareas.

El bit 1 y el bit 0: (RR y RIS) son los que permiten leer los registros de IRR y IRS de la siguiente forma:

RR	RIS	
1	X	
1	0	Orden de lectura en IRR.
1	1	Orden de lectura en ISR.

Para poder leer IRR y IRS primero se deberá colocar las combinaciones anteriores en RR y RIS y en el siguiente pulso de reloj leer esos registros.

En la inicialización 1 0 siempre que se da una dirección par, se leerá IRR.

- **Programación de las interrupciones.**

Los PIC los programa la BIOS en el arranque y no conviene cambiarlo.

Para AT+

Programación del Master

```
mov al, 11h
out 20h, al     } se escribe en la dirección par (20h) en ICW1

jmp $ + 2  ──→  orden de salto, salta 2 bytes hacia delante y vacía
                la cola de prefech y la siguiente instrucción se
                carga otra vez en la cola, lo que se produce un
                pequeño retardo.
mov al, 08h
out 21h, al     } ICW2[00001000]. Dirección impar (21h) y se
                  escribe

mov al, 04h
out 21h, al     } ICW3[ 00000100]. Indica que hay un slave en
                  IRQ2
mov al, 01h
out 21h, al     } ICW4[0000 00 01] ──→ especifica final de
                                       interrupción
```

Programación para el Slave.

```
mov al, 11h
out A0h, al

        (todo igual al anterior)
```

ICW1 = 0001 0001 → Igual al master pasa por ICW4.
ICW2 = 0111 0000 → Int 70h .... int 77h.
ICW3 = 0000 0010 → Sabe que el slave esta conectado por IRQ2.
ICW4 = 0000 0001

Para arrancar se escribe en la dirección impar para enmascarar a todas las Interrupciones, posteriormente se pone a cero, estos valores están programados en el BIOS.

PC / PC-XT.

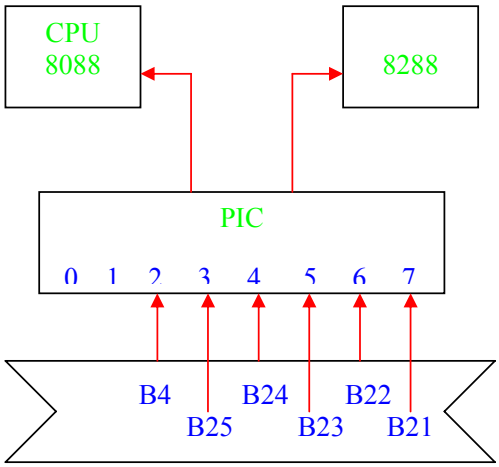


Figura 16. Esquema del PC/PCXT

B4 → No usado → Se usó para conectar el slave en el AT +.
B25 → Com2
B24 → Com1
B23 → HD
B22 → FD
B21 → LPT1

En el AT las interrupciones están intercaladas entre el slave y el master, teniendo este más prioridad que nadie, pero el slave tiene ocho direcciones que van a IRQ2.

más prioritario

0 Timer
1 Teclado
2 0 RTC
1
2
3
4 Ratón
5 Copro (En el PC original se dirigía directamente)
6 (A la NMI del CPU)
7
3 Com2
4 Com1
5 HD
6 FD
7 LPT1

menos prioritario

Tarea → Enmascarar / Desenmascarar

Enmascaramiento de INTS.

```
ICW1_PORT  EQU   21h
IRQ2_ON    EQU   11111011B
IRQ2_OFF   EQU   00000100B
```

Habilitar IRQ2

```
mov dx, OCW1_PORT
mov ah, IRQ2 ON
in al, dx
and al, ah
out dx, al
```

Inhabilitar

```
mov dx, OCW1_PORT
mov ah, IRQ2_OFF
in al, dx
or al, ah
out dx, al
```

Orden de final de interrupción.

```
OCW2_PORT        EQU  20h
EOI              EQU 20h
……
```

R	SL	EOI	0	0	L2	L1	L0
0	0	1	0	0	0	0	0

2 0 h

```
mov dx, OCW2_PORT
mov al, EIO
out dx, al
```

Ver el estado del PIC.

```
OCW3_PORT  EQU  20h
READ_ISR     EQU 00001011B
READ_IRR     EQU 00001010B

……

mov dx, OCW3_PORT  (se cambia por A0h para el slave).
mov al, READ_ISR
out dx, al
in al, dx
mov bh, al
mov al, READ_IRR
out dx, al
in al, dx
mov dl, al
```

Simulación de un programa con interrupciones.

Tenemos un PIC que se ha programado de forma normal, se envía un final de interrupción, no hay ninguna interrupción en pulso y no hay enmascarado. Llegan dos peticiones por IR2 y IR4, más tarde por IR1 y un poco más tarde por IR3.

El diagrama quedaría de la siguiente manera:

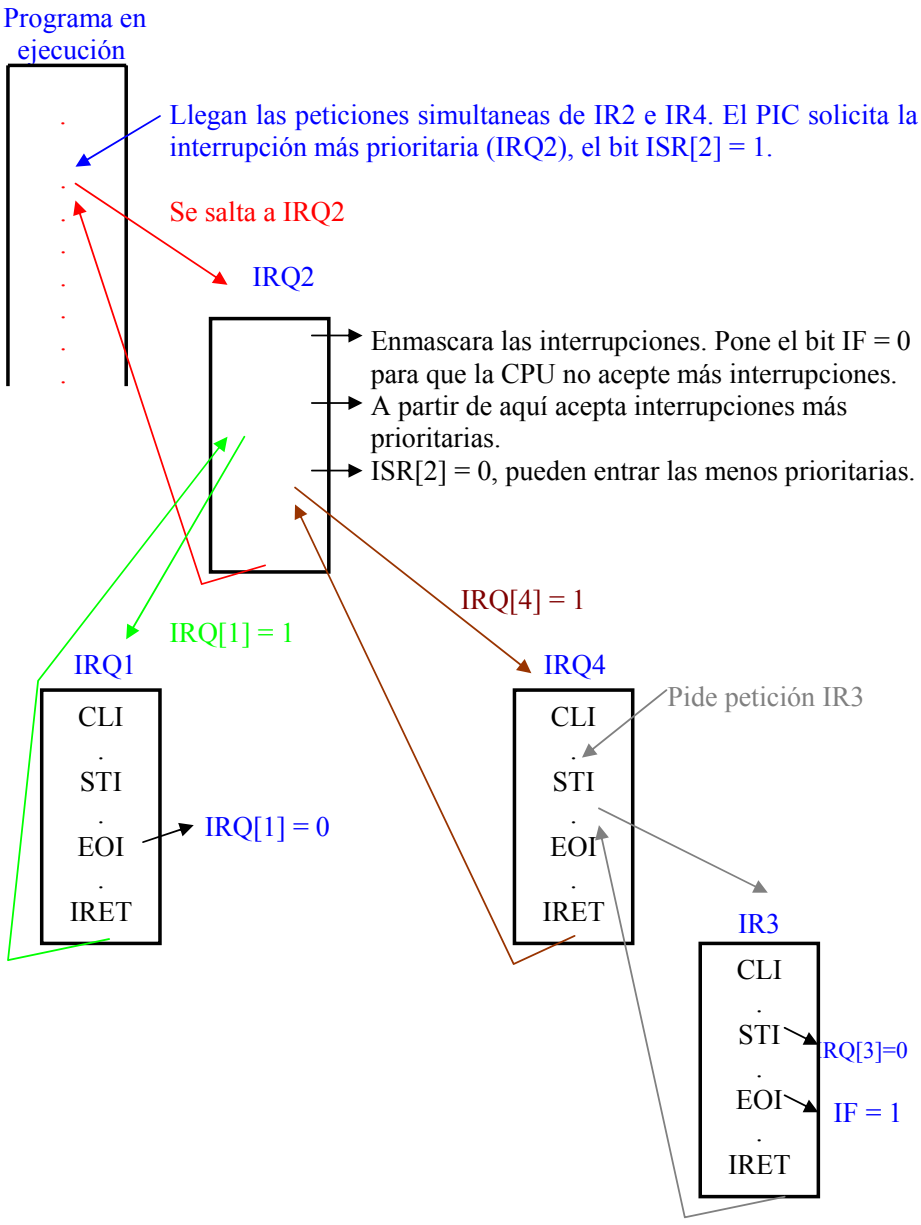


Figura 17. Modo de funcionamiento del PIC.

Tipos de peticiones.

Existen dos tipos de peticiones:

- Hardware
- Software

Las peticiones **Hardware** se dividen en { INT (Enmascarable) / NMI (No enmascarable) } Asíncronas.

En los 8086:

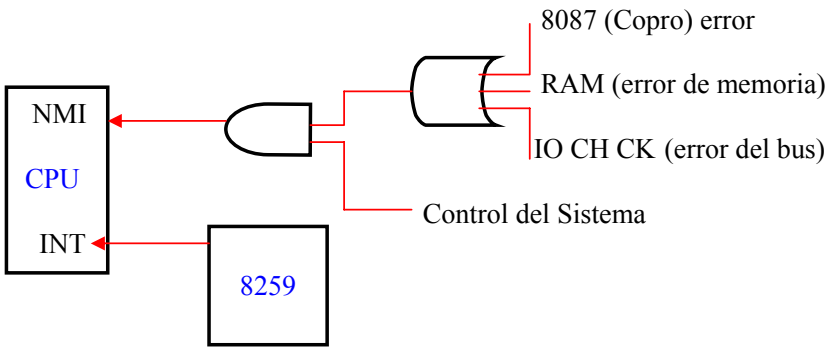


Figura 18. Esquema de peticiones hardware

Se supone que la NMI es no enmascarable pero en realidad sí se puede enmascarar, en este sistema (XT) mediante la señal de control del sistema (AND, si está a cero, enmascarado), en los nuevos es mediante CMOS pero de forma más complicada.

Las peticiones **Software** son a través de las INT → Instrucciones de interrupción.

Las excepciones se producen cuando ocurren determinadas situaciones. Estas son software pero están precedidas al hardware, pues ocurren en la CPU (interna a esta), por lo que no hay protocolo del PIC y estas son síncronas (ej. Dividir por cero) y por lo tanto predecibles, para estas intel se reserva los 16 primeros vectores de interrupción.

Tipos de excepciones.

0	División por cero
1	Ejecución paso a paso
2	NMI
3	Breakpoint (para depuración)
4	INTO overflow
5	BOUND Rango excedido
6	Invalid opcode
7	No coprocessor available
8	IDT limit too small (Double fault in protected mode)
9	Coprocessor segment overflow
Ah	Invalid task state segment
Bh	Segment not present
Ch	Stack Fault
Dh	General protección
Eh	Page Fault
10h	Error del coprocesador
12h	Machine check exception.

6.3.- Los Temporizadores

- **8253 Timer.**

Son tres contadores de 16 bits programables y se pueden programar en varios modos de funcionamiento.

Tiene una entrada de reloj y un habilitador, lo que hace es descontar un número que se mete y cuando llega a 0 da una señal coincidiendo con el último pulso de reloj.

Tiene un registro que guarda el número por si está en modo de repetición.

Permite leer el estado del contador, tiene posibilidad de programarle tres contadores (pulsos, periódicos, programados).

En los PCs está decodificado en placa base en el rango de direcciones es 40h-5Fh, aunque sólo emplea las cuatro primeras. Todos los clk esta conectados al mismo reloj con una frecuencia de 1,19 MHz y si se divide por el número que entra en el out 1 = 18.2 Hz.

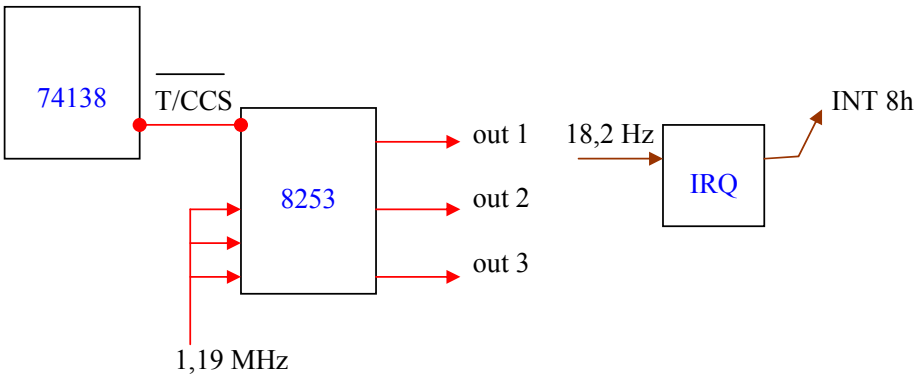
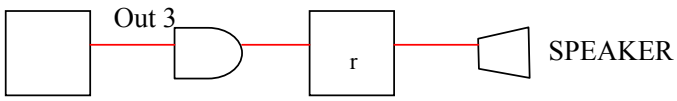


Figura 23. Timer 8253.

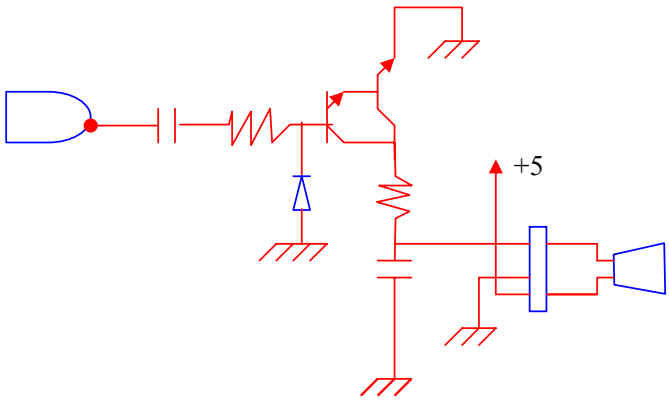
OUT 1: mantener la fecha del sistema, de un pulso cuando llega a 0 empieza en F…..Fh. La salida va al IRQ 0 del contador de interrupciones, con lo que interrumpe al sistema 18.2 veces por segundo y lo mantiene un contador en BIOS que incrementa el contador (INT 8h) TIC del Timer que se utiliza para controlar el tiempo que están encendidas las unidades de disco y controla el apagado de sus motores.

OUT 2: refresco de la memoria dinámica: cada 15 msg se da una orden de lectura de la memoria dinámica para refrescarla, ese pulso coincide cuando out_2 ha llegado a 0. Esta lectura la realiza el DMA y la CPU no se entera.

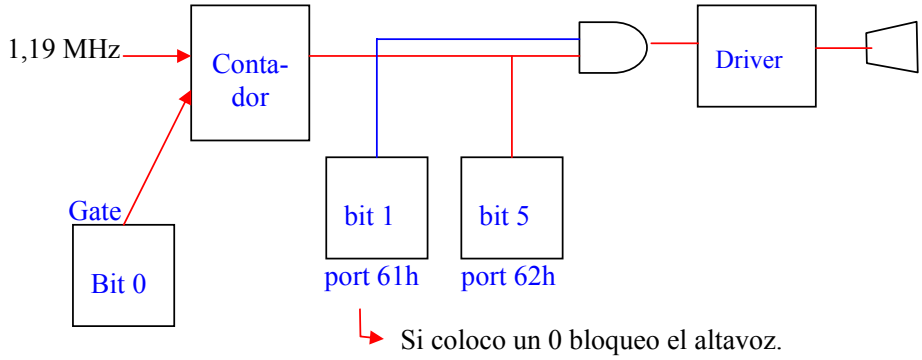
OUT 3: se utiliza para excitar el altavoz del sistema, o bien para contador genérico, desactivando el contador. El BIOS no lo programa.



Esquema habitual del driver.



Esquema con timer:



Para sonar el altavoz lo que hacemos es producir en el contador 2 una frecuencia y habilitar o cortar según el bit 1 del port b (61h)

Las únicas frecuencias son las derivadas del 1,19 MHz, le programamos un número n y la frecuencia de salida nos la dará f = fo/n donde n → 2 ≤ m ≤ 2 E 16. Para una frecuencia conocida donde f → n = fo / f, sería el número a programar en el contador 2.

Ejemplo: fLA = 440 Hz → h = 1,193182/ 440 = 2711 → número que guardamos en el contador.

Para mirar la dirección, se mira los ticks de la Bios empleando el reloj en tiempo real.

Rutina para tocar un La durante 2 seg basado en la primera forma:

```
TOCA_LA PROC

        mov al, B6h    ;10110110  modo de funcionamiento.
        out 43h, al
        mov ax, 2711
        out 40h, al
        mov al, ah
        out 40h, al
        mov bx, 0040h
        mov es, bx
        mov bx, 0060h          ; ES:[BX]  Ticks en bios.
        mov dx, word ptr  es:[bx]  ; lee palabra baja del contador de ticks
        add dx, 36       ; 18 ticks = 1 seg, 36 ticks = 2 seg de duración.
        in al, 61h
        or al, 03h       ; se obliga a 1 los bits bajos.
        out 61h, al
@_1:

        mov cx, es:[bx]
        cmp cx, dx
```

```
        jne @_1
        in al, 61h
        and 0FDh
        out 61h, al
        ret

TOCA_LA ENDP
```

6.3.1.- El contador programable 8254.

A partir del AT+ se ha colocado en vez del timer 8253 el 8254, que es igual al anterior con una pequeña diferencia

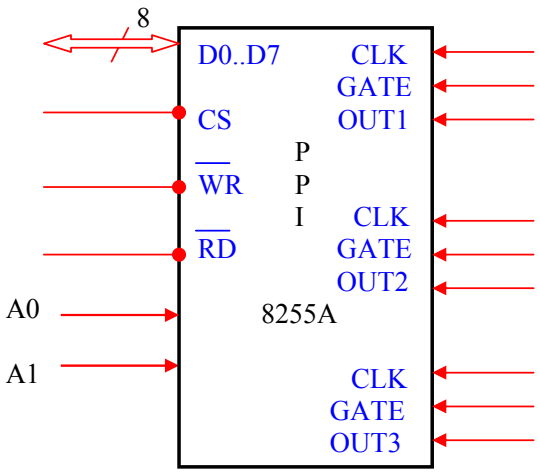


Figura 24. Esquema de un 8253.

Internamente existen 3 contadores y circutería común. (Mirar esquema de la siguiente página).

Para poder cargar el contador se carga el registro contador colocando los datos en CRM y CRL y para leer el contador se lee OLM y OLL, pues el contador nunca debe parar.

Sólo se puede acceder a 4 direcciones internas (A1.A0) de la siguiente forma:

A1	A0	
0	0	Contador 0
0	1	Contador 1
1	0	Contador 2
1	1	Registro de control

Este dispositivo se encuentra en la dirección base 40h. En el registro control se programa cada uno de los contadores y el modo de funcionamiento.

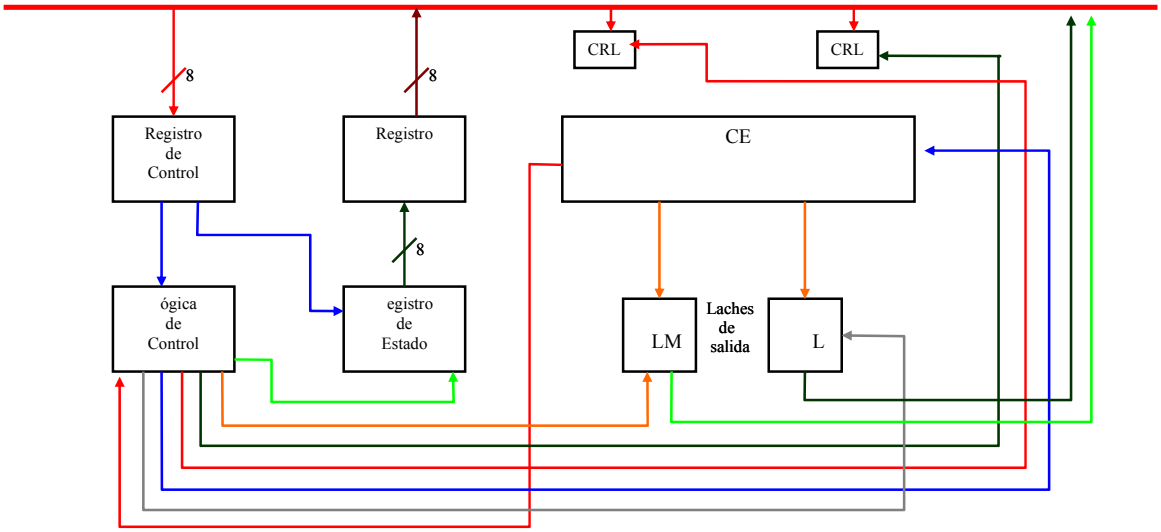


Figura 25. Esquema de controladores.

La estructura interna de este registro de control es la siguiente:

7	6	5	4	3	2	1	0
SCI	SC0	RL1	RL0	M2	M1	M0	BCD

SC1 y SC0 seleccionan el contador al que se quiere acceder (selec counter)

SC1	SC0	
0	0	Contador 0
0	1	Contador 1
1	0	Contador 2
1	1	Poner 11 en el 8253 es ilegal pero en el 8254 se aprovecha el código para la operación de lectura hacia atrás (Read Back Command) que se hecha en falta en el 8253.

RL1 y RL0 selecionan los campos de lectura carga.

RL1	RL0	
0	0	Se lactchean el contador, se cargan los laches de salida, del seleccionado por SC1 y SC0
0	1	Lee o escribe solo el byte menos significativo.
1	0	Lee o escribe solo el byte más significativo por el bus de datos.
1	1	Lee o escribe 2 bytes consecutivos, primero el menos y luego el más significativo.

En M2, M1 y M0 se colocan los posibles modos:

M2	M1	M0	
0	0	0	modo 0
0	0	1	modo 1
X	1	0	modo 2
X	1	1	modo 3
1	0	0	modo 4
1	0	1	modo 5

BCD, si esta 0 contador Binario de 16 bits y si esta 1 cuenta en BCD.

Modo 0 (Interrupción a la terminación de la cuenta): siempre cuentan hacia atrás hasta llegar a 0 en modo 0. La salida del contador se pone a baja y cuando llega a 0 se pone a alta (no pulsos), permanece en alta hasta programar otra nueva cuenta o se reprograme el contador.

La señal de GATE a alta para no parar el contador, si se pone a baja el contador no cuenta.

Modo 1 (Hardware Retriggerable one_shot): la salida inicialmente en alta y el contador esta a punto de dispararse, esa señal le viene por la línea de GATE, que se pone a baja y al poner un pulso en alta en GATE comienza el contador a contar hasta llegar a 0, donde da una salida a alta.

Cada vez que la línea GATE da un pulso a alta se reiniciara el contador.

Modo 2 (Rate Generator): son modos continuos, no se paran, cuando se programa el contador, se empieza a contar y cuando llega a 0 da un pulso en la salida y se reinicia otra vez, hasta reprogramarse. La línea de GATE tiene que estar a alta.

Modo 3: igual al 2, pero se saca una onda cuadrada. Si n es por la salida out a alta n/2 pulsos y los otro n/2 pulsos estará a baja si n es impar (n + 1) / 2 el pulso estará en alta y la salida (n – 1) / 2 será a baja.

La señal GATE se comporta como un habilitador, estará a alta para que funcione. El BIOS programa los contadores 0, 1 en modo 2 y se pone todo a 0 para contar lo máximo y en binario.

Modo 4: es como el modo 1, pero se dispara por software la señal GATE. Es un habilitador.

Modo 5: la señal de GATE es la que activa el contador cada vez que llegue una señal en su flanco de subida.

Se utiliza en cualquier dispositivo que requiera añadir tiempo con cierta precisión.

En el PC el contador 0 es un divisor de frecuencia 18,2 Hz, cada 18 veces por sg da un pulso que va a IRQ0 y existe una interrupción que son las llamadas ticks. Existen unos contadores en el BIOS con cada pulso y se utiliza para medir tiempos.

El 1 se utiliza para refresco de la RAM.

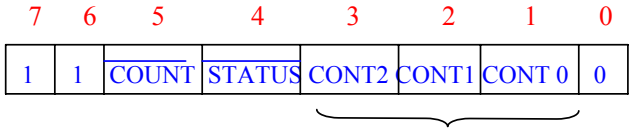
El 2 se utiliza para el altavoz, algunos autores piensan que debería programarse en modo 3, para IRQ0 daría igual al activarse por flanco de subida.

Para leer el contador existen tres métodos:

1.- Se selecciona el contador a leer o escribir y se lee, primero se debería pasarle por la puerta GATE, pero si esta de forma continua no se debería parar.

2.- Se coloca la señal de latches en el contador, se selecciona el contador. A continuación se da la orden de lectura de los registros de salida.

3.- Utilizar el *Read Back Command*, si se activa este método, el resto del registro no tiene el mismo valor.



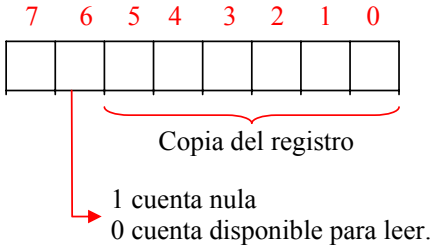
contadores a los que se va a acceder y puede afectar a los 3 de forma simultanea.

$\overline{\text{COUNT}}$: Leeremos el contador si está a 0.

$\overline{\text{STATUS}}$: Leeremos el estado si está a 0.

Si leemos el contador primero leemos el byte menor y luego el mayor del OCM y OLL que están latcheados.

Si leemos el status, es un registro de estado que es una copia del registro de control en los 6 bits más bajos.



Si COUNT y STATUS están a 0 0 primero se lee el contenido del contador.

Con 1,19 MHz de entrada podemos medir en orden de microseg, mirando el contador de ticks, su medición es muy grande, se puede mirar con mayor precisión por donde va. Para ello realizaremos una subrutina para saber la dirección de ejecución de una rutina. Se coge del timer el tiempo al principio y otro al final, y se restan las propias subrutinas.

Primero se mira la memoria del BIOS donde están los ticks y luego se mira el contador y se transforma en microsegundos todo.

La dirección de memoria donde guarda el BIOS la cuenta de ticks es 0040:0066h (Double word) que se aumentan cada 55 miliseg.

En XT se pedía la hora y a partir de AT se mantiene la hora con una pila.

Para ser mas precisos que los ticks se mira el contador de los timer para ver por donde va.

TH	TL

t = (2E16 * TH + TL) * 5,4925 µ * Tfrac.

Tiempo fraccionario: se calcula al leer el contador, pero si lo leemos directamente (decrementa de máximo a 0) no son los pulsos del reloj sino lo que nos de tenemos que restarle el total.

T.frac = (2E16 – contador 0) * 0,8381 µs.

TICKSCOUNT PROC

```
mov al, 0C2h    ; 11000010 para leer el estado y cuenta del contador 0
cli
out 43h, al     ; se escribe en el timer
jmp $ + 2
in al, 40h      ;  se le el estado de la salida
shl al, 1       ; se guarda el estado de out en carry
in al, 40h      ; se lee el estado de la salida (solo nos interesa el bit más
mov ch, al      ; significativo
in al, 40h
mov ch, al
jcxz @_5        ; comprueba si está en 0
rcr cx, 1       ; en modo continuo hace dos pasadas y descuenta de dos en
not cx          ; dos (modo 3) y siempre un número par para sacar la banda
neg cx          ; cuadrada, metemos el bit de acarreo por el más significati-
mov bx, 0040h  ; vo de cx y nos diría el período por el que ha transcurrido
mov es, bx              ; desde el principio.
mov bx, 006Ch           ; ES:[BX] dirección de Ticks
mov ax, TRI1
mul word ptr es:[bx]   ; ax * TL = dx:ax
mov wFrac, ax
mov si, dx
mov ax, TRI1
mul word ptr es:[bx + 2]        ; DX:AX = TH * 16384
mov di, dx
add si, ax
adc di, 0
mov ax, TRI2
mul  word ptr es:[bx]
add si, ax
adc di, dx
mov ax, TRI2
mul word ptr es:[bx + 2]
add di, ax
mov ax, TRI3
shr cx, 4                       ; cx = cx /16
```

```
        mul cx
        add ax, wFrac
        mov ax, si
        adc ax, dx
        mov dx, di
        adc dx, 0
        mov word ptr DWLOWTICKCOUNT[0], ax
        mov word ptr DWLOWTICKCOUNT[2], dx
        sti
        ret
@_5:
        sti
        jump TICKCOUNT

TICKCOUNT ENDP.
```

6.3.2.- Reloj en tiempo real (Real Time Clock RTC) MC 146818A

A partir de los AT+ se coloca este reloj que con una pequeña batería mantiene la fecha y la hora del sistema (tecnología CMOS).

En los AT originales se colocó el MC146818A (RTC Real Time Clock) que dispone de una RAM:

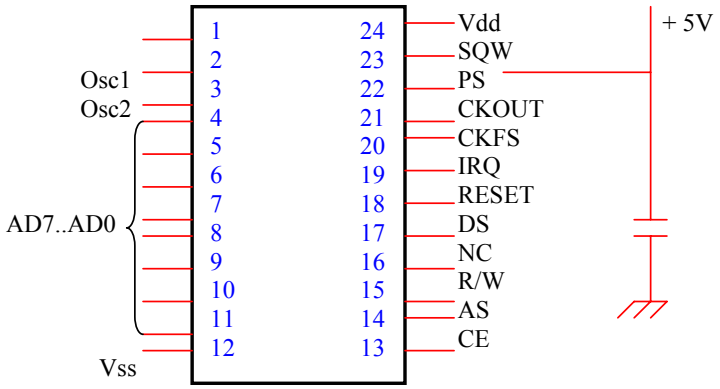


Figura 26. Esquema de un reloj en tiempo real.

OSC1
OSC2 } Entrada de la señal externa de reloj.

4..11: líneas de dirección desde AD7..AD0

Con los circuitos CMOS, Vss (masa) y Vdd (alimentación del ordenador) las frecuencias admitidas de entrada por OCW1 son ondas cuadradas del siguiente tipo:

4,194304 MHz
1, 048576 MHz
32,768 KHz → Frecuencia par los PC

Las direcciones de datos son multiplexados en el tiempo. En la primera parte del ciclo de bus las direcciones y en la segunda parte del ciclo del bus son los datos.

CKOUT: saca la señal de OSC1 bien directamente o bien dividida por 4.

CKFS: decide si se divide por 1 o por 4

- si se pone a Vdd (alta) se divide por 1.
- Si se pone a Vss (baja) se divide por 4.

SQW: sale una onda cuadrada de frecuencia dependiente de uno de los registros internos programables por el software.

AS: equivale a la señal ALE, cuando hay direcciones se activa para pode latchearlos de alguna forma.

R/$\overline{W}$: lectura o escritura.

CE: selección del chip.

Reset: resetea el circuito-

PS (Power Sence): detecta el nivel de alimentación y conmuta a *standby* si se apaga el ordenador (si baja la tensión)

Diagrama de bloques internos.

Mirar esquema de la figura 27.

Cada bloque de 32 tiene 5 salidas y el selector lo utiliza de forma conveniente para sacar 1 Hz que va a los registros básicos A, B, C, D

64 bytes de direcciones internas (0..63) que se ponen en 6 bit de dirección (los 6 menos significativos).

El bit más significativo se utiliza para el enmascaramiento de las interrupciones.

0) Segundos (0..59).
1) Segundos alarma (0..59).
2) Minutos (0..59)
3) Minutos alarma (0..59)
4) Horas
5) Horas alarma (1..12 ó 0..23)
6) Días de la semana (1..7, empezando por el domingo)

7) Día del mes (1..31).

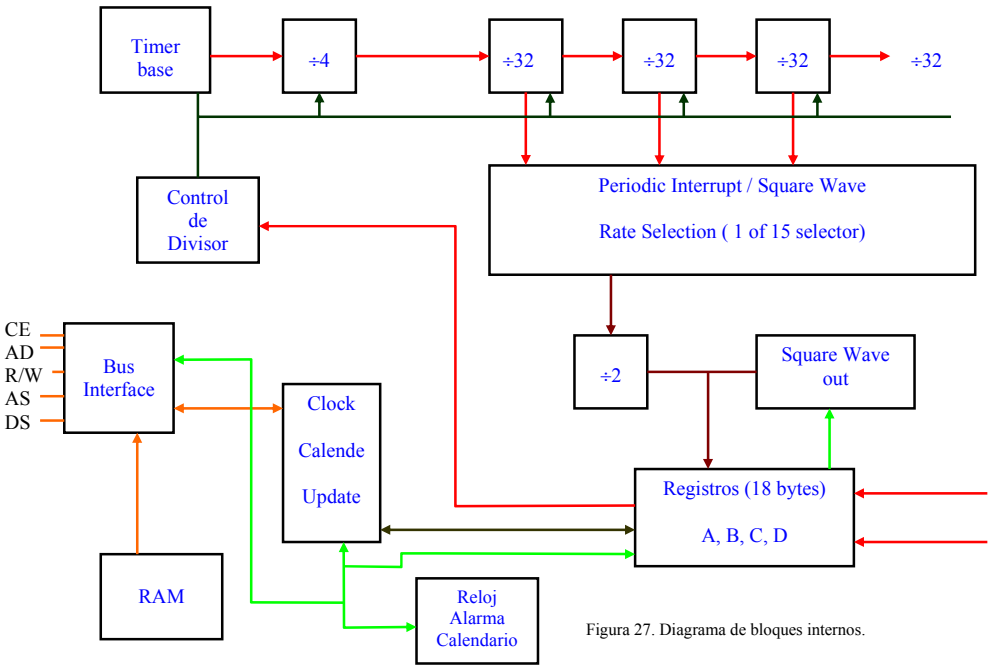


Figura 27. Diagrama de bloques internos.

8) Mes (1..12)
9) Año (0..99)

Registros 10 = ah, 11 = bh, 12 = ch, 13 = dh (programación del RTC):

	7	6	5	4	3	2	1	0
A	UIP	DV2	DV1	DV0	RS3	RS2	RS1	RS0
B	SET	PIE	ALE	VIE	SQWE	DM	24/12	DSE
C	IRQF	PF	AF	VF	0	0	0	0
D	VTR	0	0	0	0	0	0	0

UIP (Upgrade In progres): para saber cuando se está actualizando:

1: está en o va a comenzar un ciclo de actualización.
0: no está actualizándose.

Un ciclo de actualización dura 1984 μs.

DV[0..2]: bit de selección del divisor, permite programar la cadena de divisores para seleccionar la frecuencia base:

DV2	DV1	DV0	
0	0	0	OSC1 = 4 ,194 MHz
0	0	1	OSC2 = 1,048 MHZ
0	1	0	OSC3 = 32768 MHz
1	1	0	Cuando no se pone ningún cristal externo
1	1	1	tiene su propio oscilador y admite estos valores sin ningún cristal externo.

RS[0..3] (Rate Selection): selecciona la frecuencia, selecciona una de las 15 partes del divisor de etapa o bien lo inhabilita.

PF (Periodic Flag): se pone a 1 el ritmo de la frecuencia de salida ó 0 el ritmo de la interrupción (detecta el flanco de subida).

AF (Alarm Plag): se activa a 1 cuando la hora actual coincide con la hora programada para la alarma.

UF (Upgrade flag): se pone a 1 después de cada actualización y se pone a 0 cuando se lee el registro C.

IRQF: tiene formula lógica:

IRQF = PP * PIE + AF * ALE + UF * VIE

Los registros PIE, ALE, VIE, habilitan los que están debajo de él y que saltan con los eventos. El IRQF se pone a 1 cuando salta alguno de los eventos.

La interrupción puede ser por tres causas distintas.

SET: habilitador general de la actualización.

0 permite que el ciclo de actualización se suceda cada segundo.

1 permite cambiar la hora: abortar el ciclo de actualización.

SQWE: habilita el pin de salida SQW

0 onda no sale.

1 sale la onda.

DM: modo de la fecha (binario o BCD).

0 BCD

1 binario

24/12: formato de la hora.

0 12 horas

1 24 horas

DSE (Dailying Saving Enable):

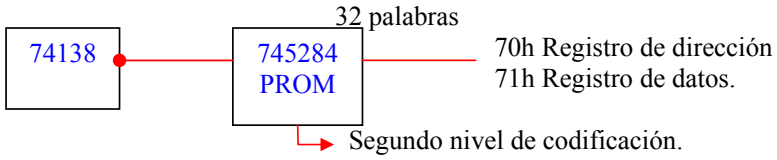
0 no se cambia

1 se habilitan las actualizaciones especiales como p.e: el horario de verano.

VRT: indica la condición del contenido de la memoria RAM que tiene el chip, supuesto que el power sense está bien.

Si PW = 0 $\rightarrow$ VRT = 0

WRT = 1 si se lee el registro D.



En el puerto 70h se escribe el registro al que se quiere acceder y luego se escribe o se lee por el puerto 71h.

Si se accede a los 10 primeros registros hay que mirar URP y si se quiere mirar la hora se pone el set a 1.

```
READ_CMOS PROC

        or al, 80h
        out 70h, al
        IODELAY
        in al, 71h
        ret

READ_CMOS ENDP

WRITE_CMOS PROC

        or al, 80h
        out 70h, al
        IODELAY
        mov al, ah
        out 71h, al
        IODELAY
        Ret             Hay que modificar siempre las rutinas para anidar el VIP.

WRITE_CMOS ENDP.

GETTIME  PROC

        mov  al, 0Ah
        call READ_CMOS
        test al, 80h
        jnz GETTIME
        mov ah, 0
        call READ_CMOS
        mov dh, al
        mov al, 2
        call READ_CMOS
        mov cl, al
        mov al, 4
        call READ_CMOS
        mov ch, al
        ret

GETTIME ENDP.

              ch : cl  : dh
        horas= minutos = segundos
```

Memoria CMOS del reloj.

En el RTC se podría aumentar la memoria a 128 byte pero si se pasa más arriba ya no se direcciona con 7 bits y el bit más significativo de máscara da interrupciones.

Registros:

0Eh → diagnóstico de estado

0Fh →

0Ch → se ha producido un reset del software o inesperado.

01h →

02h → reset después del test de memoria (es bueno).

03h → reset después del test pero no ha habido ningún fallo.

Estos valores los pone el programa de arranque:

04h → reinicio a través del int 14 (reinicio por software).

05h → flush Keyboard and jump vi 40:67h (cuando se resetea del teclado).
Este registro se usa para controlar la fuente del reset.

10h → tipo de floppy

11h → secuencia de repetición del teclado.

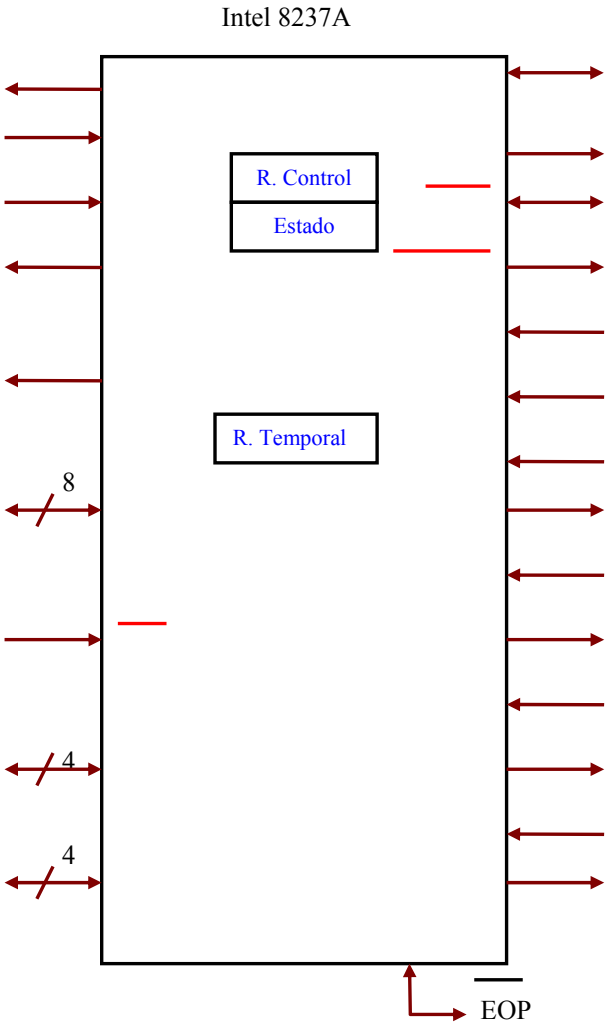
12h → datos del disco duro.

15h → memoria base

16h → memoria base

32h → registro del ciclo.

6.4.- El controlador de DMA y los registros de página



Registro de control: decodifica las ordenes y el modo para saber su funcionamiento.

Decodificador: resuelve los problemas entre los canales.

Tiene un reloj externo que sincroniza el DMA, posteriormente este reloj era ya muy superior y lo que se hace es tener un reloj propio de 8 MHz.

Esta diseñado para trabajar en dos ciclos:

- Activo
- Inactivo

Se puede colocar en siete periodos de estados diferentes.

Estado I (Inactivo) SI.

Estado 0 (Inicio) S0 , estado de inicio del ciclo de transferencia

Estado 1, 2, 3, 4, S1 S2 S3 S4, estados de trabajo.

Estado W, SW estado de espera.

SI entra cuando no tiene peticiones válidas pendientes para transferir, en este estado es cuando se puede programar (por parte de la CPU).

Si se le solicita una petición de transferencia entra S0, aquí el DMA solicita una retención del bus a la CPU (quien le cede el bus) por HRQ, la CPU se desentiende del bus y lo controla el DMA, está en este estado hasta que la CPU no le reconozca la petición, todavía puede ser programado por parte de la CPU, una vez que esta reconoce la petición se lo hace saber al DMA por HLDA. La CPU le cede el bus al DMA, esto lo hace la CPU poniendo a alta impedancia sus líneas de bus.

Después de recibir el DMA la señal de dominio del bus se pasa a los estados de trabajo. Si no se puede transferir todos los datos, se duplica S3 hasta que se termine, siempre vigilando la señal de Ready, si esta a baja puede insertar estos estados de espera.

El DMA puede hacer transferencia entre dispositivos de I/O como a memoria, simultaneamente. Activa simultaneamente memR o memW y I/OR o I/OW.

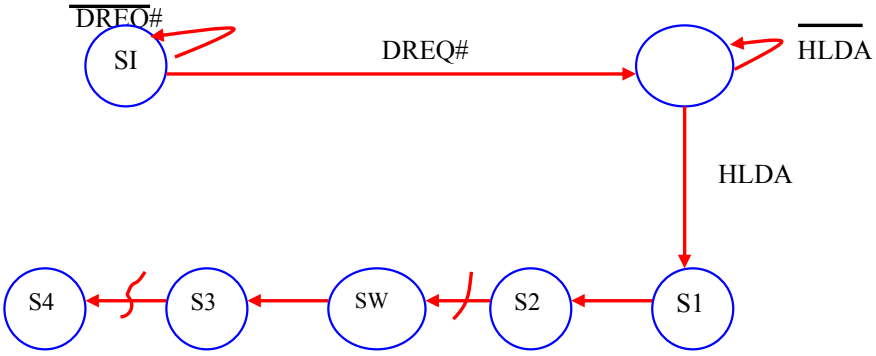
El DMA no tiene registro interno que retenga el dato, solo habilita el bus de datos de forma que indica de donde vienen y a donde van los datos.

Permite hacer transferencias de memoria a memoria, pero, en el bus solo se puede poner una dirección. Lo que se hace es hacer dos transferencias consecutivas y utiliza el registro temporal para almacenar el dato.

Cuando ningún canal pide servicio el DMA entra en el ciclo inactivo SI, que es una secuencia de estados SI que duran un periodo de reloj. Lo que hace el 8237 es muestrear las líneas de DREQ hasta que algún canal solicita transferencia y OCS activa la CPU para programar al DMA.

A0 : A3 determinan a que registro se va a acceder para programar.

Los accesos para programa se hacen durante SI



Si la transferencia es en bloque puede suceder de forma diferente.

Modos de realizar una transferencia.

Existen cuatro modos diferentes de realizar una transferencia:

Modo Simple: esta programado para hacer una sola transferencia dato a dato (ya sea tipo hardware o software), es decir, solo coge una dirección de memoria y se transfiere el dato (con todo el protocolo visto anteriormente). El contador de dirección actual se incrementa o decrementa según la dirección. El contador de byte también se decrementa y si se pasa de 0 a –1 TC (terminal count) avisa que ha llegado a 0 (para indicar un cierto número de transferencias).

Puede ser que se haya programado en modo simple pero DREQ sigue activado para el DMA, desactiva HRQ y el siguiente ciclo lo activa con esto se garantiza que la CPU puede hacer accesos al bus entre HRQ del DMA y HRQ.

De este modo se garantiza que entre acceso y acceso del periférico la CPU utiliza el bus.

Modo bloque: realiza una transferencia tras otra hasta que llegue el TC, entre tanto el DMA es dueño del bus.

En este modo DREG no necesitan estar activados durante todo el tiempo, pues el número de transferencias se realizan según TC y no dato a dato como en el anterior.

EOP, si se activa externamente interrumpe un proceso de bloques, que sea muy largo.

Modo demanda: en este modo se programa para realizar transferencia hasta que TC, EOP o DREQ# se inactive.

Modo cascada: enlazar o encadenar mas de un 8237, para poner más canales de DMA con varios controladores. Solo transmite la petición hacia la CPU pero los datos los pone el DMA correspondiente.

Existen otros tipos de estados:

Transferencia de escritura: lee del periférico y escribe en memoria.

Transferencia de lectura: lee de memoria y escribe en el periférico.

Transferencia de verificación: también denominada falsa transferencia. Opera como si realmente estuviera haciendo una transmisión generando una dirección, etc. Pero las líneas que realmente manejan el bus no se activan, pues lo único que hace es hacer una verificación de que todo va bien.

Transferencia de memoria a memoria: se realiza forzosamente con los canales 0 y el 1, el 0 y DREQ0 es quien inicia la transferencia. En la primera parte del ciclo, lee de 0 y lo pone en el reg. Temporal y en la segunda parte del ciclo lee del temporal y escribe en memoria.

Auto inicialización: se recargan los registros de bytes y base.

Prioridad fija: asigna un orden de prioridad fija.

Prioridad rotatoria: último canal que ha prestado servicio, tiene la prioridad mas baja (cambia la prioridad del resto).

Registro de contado de bytes actual.

Se puede leer o escribir en dos accesos consecutivos para alta o baja.

Si se tiene en autoinicialización se recarga con el valor original, esto ocurre con EOP, tanto puede ser interno o externo.

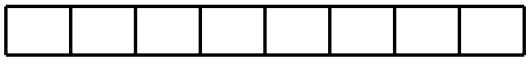
Si no esta en este modo al terminar el contenido será FFFFh. Se accede por BASE + 2 * CANAL + 1.

Registro de Dirección base y Registro de contador de bytes.

Son los que guardan el valor original de los registros de dirección y contador de tal manera que si están en auto inicialización se cargan estos valores.

Los BASE se escriben simultaneamente cuando se escriben los originales al programarlo. Al empezar la transferencia los actuales cambian pero los Bases no.

Registro de control.



Bit 0: habilita la transferencia de memoria a memoria.

1→ habilita el canal 0 al 1.

0 → impide transferencia de memoria a memoria.

Bit 1: bloquea la dirección del canal 0, su valor no importa. Permite grabar un mismo valor en todo el bloque de memoria a través del DMA.

Si se pone un 1, bloque la dirección del canal 0, que queda fijo, y el canal 1 varía, poniendo lo que hay en la dirección del canal 0 en todo el bloque de memoria apuntado por el canal 1.

Bit 2:

0 → funciona normalmente.

1 → bloquea todo el controlador, impide la transferencia.

Bit 3: tiene sentido si el bit 0 = 1, las transferencias se pueden hacer con temporización normal con un 0 ó un 1, que sería ciclo comprimido.

Bit 4: prioridad.

0 → prioridad fija.

1 → prioridad rotatoria.

Bit 5: depende del bit 3, si este esta a 0 no importa el bit 5, pero si esta a 1 y el bit 0 esta también a 1:

0 → hace que el pulso de escritura se retarde, para fijar bien la dirección de memoria.

1→ modo normal.

Bit 6: permite seleccionar el nivel de las líneas de petición de transferencia DREQ:

0 → los DREQ son activa alta

1→ los DREQ son activa baja.

Bit 7: igual pero para el nivel activo de DACK.

0 → baja

1→ alta.

Registro de modos.

Cada canal tiene un registro de modo pero a todos se accede por BASE + 11.

Los dos bit de menos peso deciden el canal al que se va a acceder.

0 0 → canal 0

0 1 → canal 1

1 0 → canal 2

1 1 → canal 3

Se programa cada uno de los canales con las condiciones de funcionamiento que va a tener cada uno.

Bit 3 y 2:

0 0 → verificar

0 1 → write

1 0 → read

1 1 → ilegal

Bit 4: auto inicialización:

0 → no esta en modo de auto-inicialización

1 → esta en modo de auto-inicialización.

Bit 5: dirección de recorrido de la memoria, decide si la memoria se recorre en direcciones crecientes o en direcciones decrecientes.

0 → Registro de dirección actual incremento

1 → Registro de dirección actual decremento

Bit 6 y bit 7: modo de funcionamiento:

0 0 → Demanda

0 1 → simple

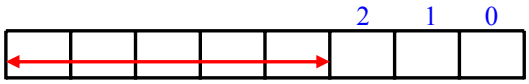
1 0 → bloques

1 1 → cascada

Este es un registro de solo escritura y se escribe por BASE + 0Bh

Registro de peticiones.

Cada canal puede responder a peticiones de transferencia que las puede solicitar el periférico (DREQ) o por software, por lo que cada canal tiene un bit para ver la naturaleza de las peticiones



Don`t care

Bit 0 y bit 1: selecciona el canal al que pertenece.

Bit 2:

0 → resetea el bit de petición.

1 → activa el bit de petición (activa la DREQ).

Para acceder a este registro se accede por BASE + 9h

Registro de mascara.

Existe un bit por cada canal. Las posibilidades de cada bit son las siguientes:

0 → enmascara el canal, impidiendo que le lleguen peticiones de transferencias.

1 → habilita el canal.

Se puede acceder individualmente a cada canal a través de BASE + 0Ah.

Los cuatro bits están agrupados en la dirección BASE + Fh, se acceden a los cuatro de forma simultanea, enmascarando o desenmascarando todos los canales. Las direcciones BASE + 0Ah y BASE 0Fh son de solo escritura.

Registro de estado.

Registro genérico al DMA, le da información a la CP. Siempre está disponible para que la CPU lo lea. Presenta información sobre el estado de los canales en el instante inicial.

Los cuatros bits bajos informan de si el correspondiente canal ha alcanzado el TC, entonces se pone a 1, si se da un EOP externo.

Los cuatro bits altos informan si tiene petición de transferencia por dicho canal, en este caso se pone un 1.

Para acceder a este registro es a través de BASE + 8h, tanto para lectura como para escribir en el reg. Temporal (guarda el dato para traspaso de memoria a memoria).

Borrado del biestable primero / ultimo.

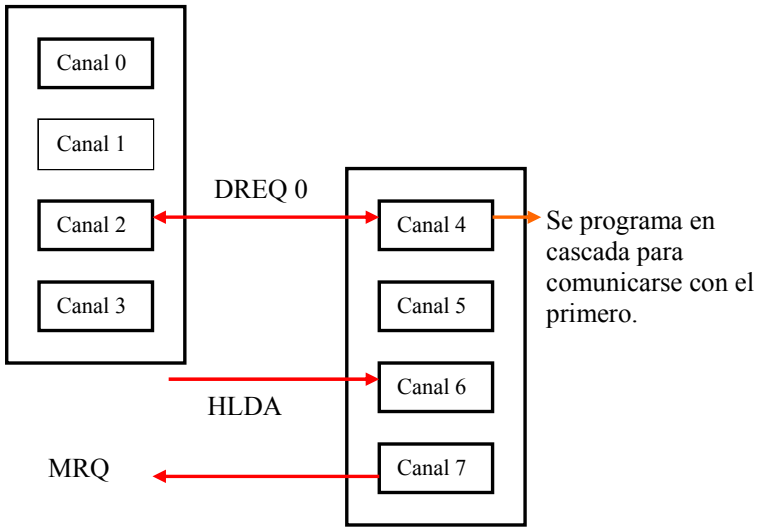
Nos dice si estamos accediendo a la parte alta = 1 o a la parte baja 0.

Esta orden nos la resetea y lo pone a 0. Si se escribe en BASE + Dh, se le da la orden de borrado maestro o borrado general del DMA o reset del DMA, borra el reg. Control, estados de peticiones, reg. Temporal y primero /ultimo los pone a 0. Los de mascara los pone todos a 1 y a continuación entra en el ciclo inactivo.

En BASE + Eh, borra el registro de mascara, los cuatro bits de mascara se ponen a 0 de golpe, sin escrituras mudas (no se escribe nada).

El espacio de direcciones van de BASE + 0 a BASE + F. Sólo hay un 8237 en el PC y PC-XT. Como el 8237 maneja 16 bits de direcciones, lo que le permite acceder solamente a 64 Kbytes, frente al espacio de direcciones de 1 Mbyte del PC, hay que proporcionar las líneas de direcciones que faltan externamente al 8237; Para este cometido se pusieron los registros de páginas donde se fijan las restantes 4 líneas de dirección.

En los equipos AT y posteriores se les ha dotado de dos controladores enlazados.



El controlador BASE esta en 00h y el secundario esta en la Y6 del 74138, en dirección 0C0h 0DFh, sabiendo que A0..A3 bidireccionable y A4…A7 es solo de salida.

En el secundario solo va a las direcciones pares y va de dos en dos, por lo tanto para programarlo se utiliza:

C0h → canal 0 , dirección activa

C2h → canal 0

C4h → canal 1, dirección activa

C6h → canal 1
……
CC
CE
D0 → registro control / estado
D2
D4
D8 → orden de borrado
DC
DE

sigue el orden del primero, pero utilizando solo direcciones pares.

Nótese que se ha pasado de 8 bit a 16 bits de dirección, es decir, de 64 KBytes a 128 KBytes direccionables.

Registros de página del DMA.

Canal 0	87h	
Canal 1	83h	
Canal 2	81h	
Canal 3	82h	
Canal 4	8Fh	memoria de refresco.
Canal 5	8Bh	
Canal 6	89h	
Canal 7	8Ah	

Son todos registros de 8 bits. En el primer DMA la transferencia es de byte a byte.

A0 = 0
BHE = 0 controla desde A1 hasta A16. DMA desde A17… A23 se fija el registro de página.

El segundo DMA la transferencia es de word en word.

Para reservar memoria para el DMA debemos garantizar que no se salga de los valores de página, lo ideal es pedirle memoria al dos.

```
mov bx, nº de párrafos (de 16 bytes)
mov ah, 4Bh
int 21h
```

Secuencia de eventos en una operación de DMA.

Lo primero que tiene que haber son unas rutinas que inicialize y programe el canal correspondiente con dirección BASE de memoria para la primera escritura:

- Página
- Nº de bytes.
- Modo
- Dirección en la que se va a recorrer la memoria.

1) El periférico solicita servicio del DMA a través de la línea DREQ3.

2) El controlador verifica que esa señal de DREG3 no esté enmascarada y la envía a la CPU activando HRQ.

3) La CPU responde cuando puede y activa HLDA reconociendo la petición e indicándole que ya puede utilizar el bus (alta impedancia).

4) Empieza la transferencia, genera una dirección (reg. de direcciones activa).

5) Activa las líneas MEMW y IOR (escritura en memoria, lectura de entrada /salida) y a continuación activa la señal de reconocimiento DACK3 para que el periférico se entere.

   El dato del periférico lo pone en el bus de datos, esto pasa directamente al byte de memoria direccionado. Sin pasar por el DMA. Dependiendo de cómo se haya programado el canal, tendríamos:

   Simple : se desconecta DREQ y la CPU tiene el bus.

   Demanda: mientras DREQ esté activado, realiza un ciclo de memoria tras otro.

   Bloque: un ciclo consecutivo de escritura en memoria hasta que ocurra el TC. Cuando ocurre el TC se desactivan las líneas HRQ y DACK3, se vuelve el control a la CPU y DACK3 para que el periférico lo sepa.

   Cuando el DACK3 se desactiva también DREQ3, en medio puede ocurrir que el periférico no pueda seguir el ritmo de transferencias, él desactiva la línea de DREQ, se termina el ciclo de escritura y se devuelve el bus a la CPU.

Escritura a través del canal 3 del DMA.

Transferencia a demanda con autoinicialización con un buffer de 16k:

```
; cx :  tamaño en bytes del buffer

mov bx, cx
add bx, 15
shr bx, 4        ; redondea a párrafos enteros
```

```
add bx, 1000h  ; añade una página
mov bx, 4Bh
int 21h        ; pide al dos memoria, en ax devuelve el segmento.
mov bx, ax
shl ax, 4
shr bx, 12
mov si, bx
add ax, ax
adc bx, 0
cmp bx, si
je @@_1
; página en bx, despl. 0000
xor ax, ax
```

@@_1:

```
mov al, 07h   ; …..0111h se enmascara el canal 3
out 0Ah, al
DELAY
out 06h , al  ; borra el biestable  primero/último
pop ax
out 06h, al      ; byte bajo
mov al, ah
out 06h, al      ; byte alto
mov ax, ax
dec ax
out 07h, al
mov al, ah
mul 07h, al
mov al, 000101110    ; modo de transferencia
out 0Bh, al
mov al, bl
out 82h, al      ; registro de página
mov al, 03h    ; desenmascara el canal 3
out 0Ah, al
```

Para saber por donde va el DMA:

```
out 0Ch, al
in al, 06h
mov ah, al
in al, 06h
xchg al, ah
```

8.1.- El teclado del PC

Comentario: El teclado aparece en el tema 8 (Periféricos externos) y aquí, como apartado 6.5 sólo entraría el controlador del teclado, el µP 8042, pero esa reorganización se hará en el futuro

Mirar figura 21 correspondiente al esquema del teclado.

Del puerto C del PPI salen unas líneas a unos micro-switches, que sirven para hacer saber al sistema la memoria que tiene, si tiene copro, etc..

El puerto B (de escritura), tiene las siguientes líneas:

- bit 3 y 7 N.L

bit 5 (Enable I/O CHK): habilita el chequeo de las partes de E/S.

- Bit 4 (en RAM PCK): Enable de la prioridad de la RAM.
- Bit 6 (ENDKBCCLK): Inhabilita el reloj de la conexión con el teclado, es decir, inhabilitando al teclado
- Bit 2 (ENCOM): Habilita los puertos serie.
- Bit 1 (SPKRDATA): Datos del altavoz
- Bit 0 (GATOESPK): Máscara del altavoz

} Van al altavoz interno del sistema.

Teclado XT.

La información del teclado en el XT se manda por serie, es una transmisión serie sincronizada (KBC CLK Y KBDDATA).

Cada información se compone de 10 bits, 2 bits a 1 de comienzo + 1 byte, esto entra en un registro de desplazamiento de entrada serie y salida paralelo, el 74LS322. Pero este registro es de 8 bits y entran 10,por lo que los dos primeros bits salen, pero no se pierden, el último será enviado a un biestable que activara el PIC, activando la rutina de atención INT 9h ejecutada por el BIOS, quien tomará el dato del 74LS322 y lo enviará al PPI.

Cuando finalice la interrupción, el bit 6 del puerto B, ENDK-B0 CLK pone a 0 el biestable para así gestionar la siguiente tecla.

En el AT +:

Aquí no hay PPI. Hay un micro controlador para el teclado el 8042. Mirar figura correspondiente.

Mirar figura 22 correspondiente al esquema del teclado de un AT+.

Tiene dos puertos de 5 bits (salida y entrada), dos líneas de test, y por el lado del sistema tiene una línea de dirección (A0), 8 líneas de datos (D7..D0), una línea de Reset activada a baja conectada al reset del sistema, dos entradas de oscilador XTAL Cristal que divide por dos la señal del procesador.

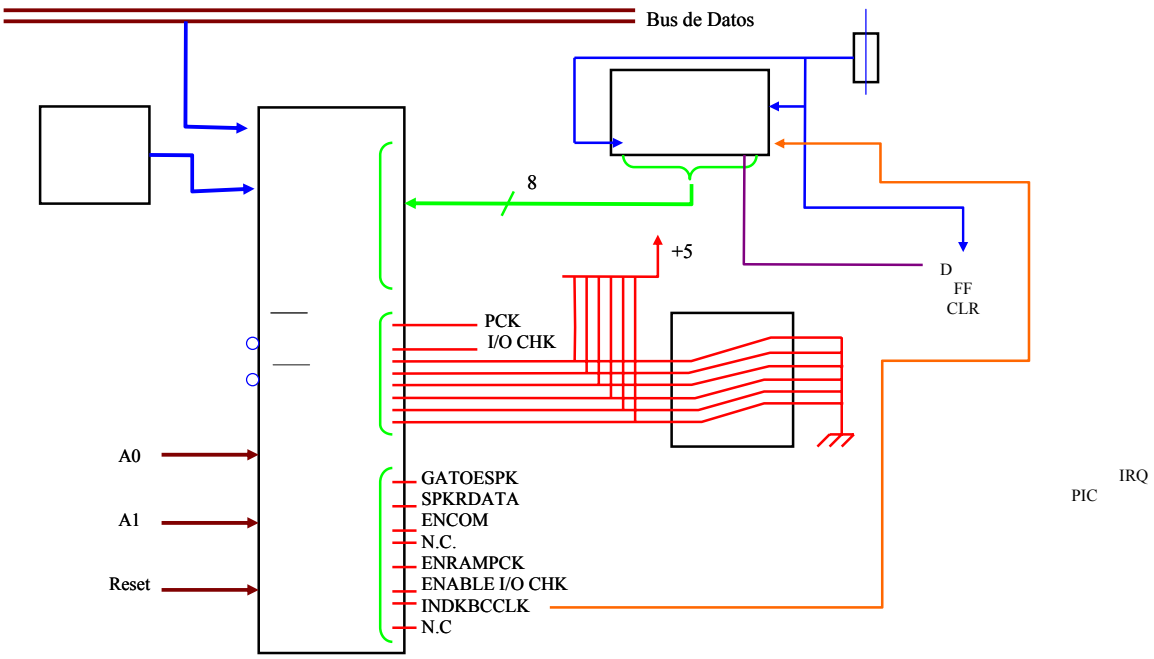


Figura 21. Esquema de un teclado.

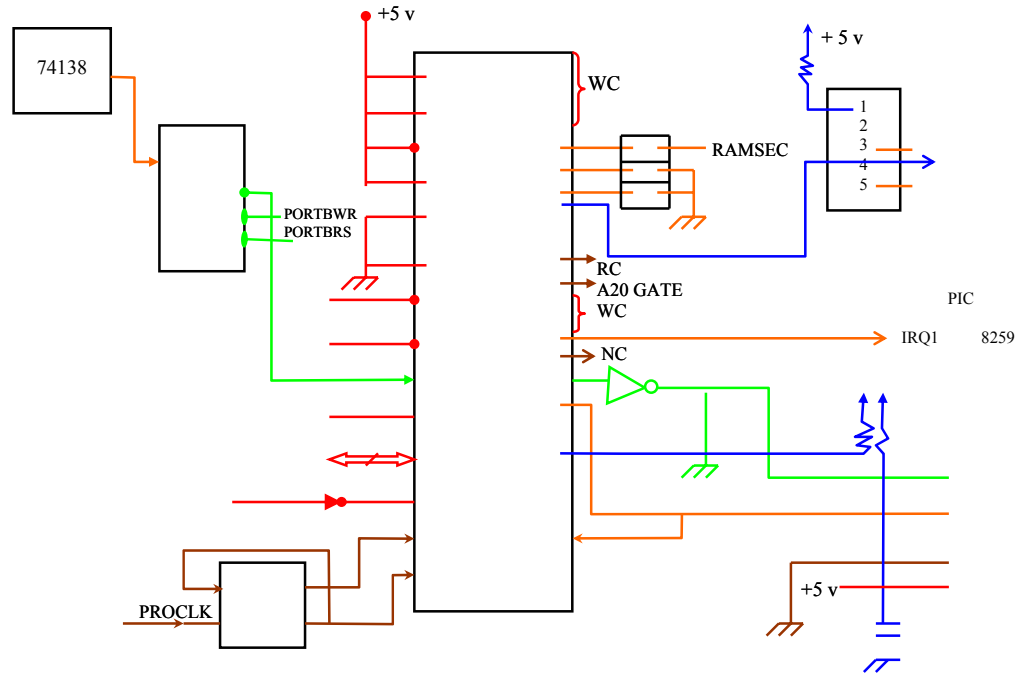


Figura 22. Esquema de un teclado del AT +.

El 74138 habilita la dirección 60h y 64h; sólo hay una línea de dirección (A0) y dos direcciones por donde entra el XA2. Por la línea D5 y D6 del 74S288 salen las líneas que simulan al puerto B, ya que en el AT+ no hay PPI.

El Registro de Estado.

7	6	5	4	3	2	1	0
PE	RTO	TTO	IS	F1	F0	IBF	OBF

BDF (Output Buffer Full): se pone a 1 si el teclado tiene un dato para ser leído por el sistema, por lo que antes de leer el puerto, leemos el registro y si este bit está a 1 se lee el puerto de la dirección 60h y después de ser leído se pone a 0.

IBF (Input Buffer Full): está a uno cuando quiere enviarle un dato al teclado, el sistema mira el bit 1 si está a 1 no manda ningún dato, sino que espera que se ponga a 1 para mandarlo.

F0: reflejo del estado del sistema del flag, control interno del 8042, sirve de chivato. El Reset lo pone a 0.

F1: según esté a 0 o a 1 nos dice si lo último que se ha escrito es una orden (1) o un dato (0).

IS (Inhabilit swich): inhabilita el teclado { 0 bloqueado ; 1 habilitado

TTO (Transmit time out): se pone a 1 si el 8042 no consigue que el teclado tome el dato.

RTO Recive_time out: error en la recepción del dato.

PE: Parity error

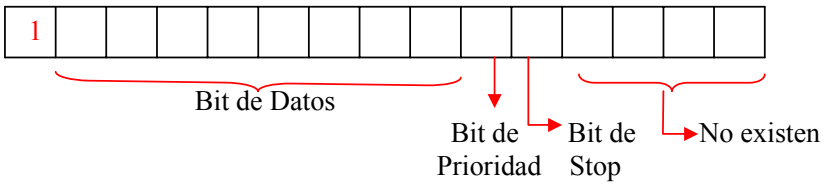
Estos tres últimos controlan el tiempo y si se pasa mucho tiempo se olvidan de la transmisión y dan un error por *time_out.*

Transmisión de la Información.

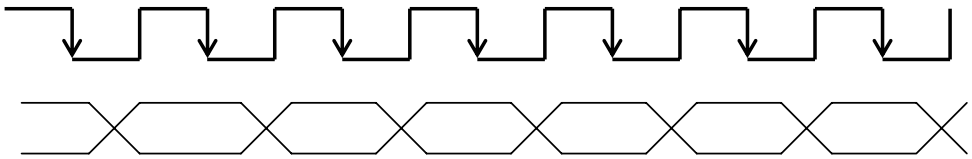
Se puede acceder en cualquier momento y se lee la información.

Para escribir es necesario mirar una serie de bit primero. La conexión entre el teclado y la interfaz es en serie y sincronizada por una línea de reloj que proviene del teclado.

La estructura de esa línea serie es una estructura de 11 bit con un bit de comienzo y después 7 bits, luego un bit de prioridad (impar) y un bit de stop al final (mayor peso).



Por cada byte a enviar lo primero que se comprueba es que las líneas de datos y reloj están en reposo (alta), si es así lo índica (la transmisión) poniendo la línea de datos a baja y el reloj en un momento da un flanco de bajada y sigue con una serie de pulsos, los flancos de bajada tienen que está con el paso medio de los bit.



Los flancos de bajada los utiliza el 8042 para desplazar los bit dentro de un registro de desplazamiento para meter los 11 bits uno a uno.

Después de la señal de Stop se queda las líneas de datos en reposo. Cada bit no puede superar más de 2 mseg; si es más "Timer out". El tiempo de transmitir el byte estará entre 6 msg y 22 msg.

6.5.- El controlador del teclado (µP 8042)

8042 Teclado:

Para que el teclado se entere de que el 8042 quiere algo baja la línea de reloj mucho tiempo hasta que el teclado se entera y luego este la deja en reposo.

La línea de datos se pone a baja y espera que el reloj del 8042 dé un pulso y luego empieza a pasar datos y al final el teclado pone un estado a baja para decirle al 8042 que ha recibido el dato (señal de stop).

Ordenes para el 8042.

20h: se quiere leer el registro de órdenes del teclado.

60h: write Keyboard controller is command byte. Se manda una orden y a continuación un byte de datos que irán por el puerto 60h.

Formato:

7	6	5	4	3	2	1	0
0	1	1				0	

Bit 7 y 1: obligatoriamente a 0.

Bit 6 y 5: obligatoriamente a 1.

Bit 4: inhabilita el teclado forzando a 0 la línea de reloj colocando un 1 en el bit 4

Bit 2 (sistem flag): chivato, para libre uso.

Bit 3: inhabilita el swich de la inhabilitación del teclado: si se pone un 1 no se considera para nada la línea de IS por lo que no se puede bloquear.

Bit 1: activa la INT 09h.

En el puerto 64h se escriben las siguientes ordenes:

AAh: pide que haga un autotest, si todo va bien escribe en el buffer de salida (60h) 55h.

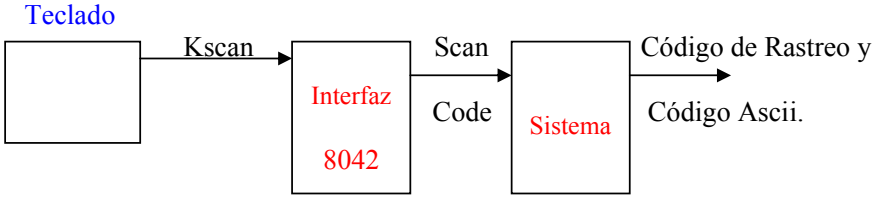
ABh: verifica las líneas de comunicación y si todo va bien devuelve al puerto 60h 00h sino un código de error dependiendo del error.

ACh (Diagnostic Dump): Vacía todo el contenido que serian 19 bits por el puerto de salida

Si recibe un dato con error de prioridad le dice al teclado que vuelva a enviarlo.

Si recibe un error time_out (no responde ante un mandato, tarda mucho tiempo) escribe en el puerto de salida FFh y pone los bit correspondientes en el registro de estado, cuando esto sucede, el BIOS suele mandar una orden de pitido por el altavoz.

Código de rastreo.



Cuando se pulsa una tecla el teclado envía un Kscan (propio de cada tecla (código)) cuando se pulsa (un byte) correspondiente al código de rastreo (MAKE).

Cuando se suelta devuelve un byte F0h y el código anterior (break).

El teclado en sí mantiene una memoria FIFO donde se almacena 16 pulsaciones de teclas y cuando se llena este FIFO devuelve 00h al controlador del teclado y este no acepta más teclas (pitido).

Todas las teclas son de make / break y generan los códigos de make y break, que es el mismo código pero precedido de 0Fh

Todas las teclas son "typematic". El teclado si se mantiene pulsado manda el código repetidas veces, a una frecuencia programable de repetición, (entre unos límites), además es programable un cierto retardo desde la primera pulsación y mandar el make, y luego se envía repetidamente con un cierto periodo hasta que se suelte.

Cuando se repite una tecla solo se guarda el código de break una sola vez.

Tiene dos procedimientos dentro del microcontrolador:

POR (Power_on_reset): cuando se resetea se realiza un chequeo interno (300 msg—9 mseg).

BAT (Basic-Assurance-Test): hace un chequeo de la RAM (2K) y mira si todo esta bien, chequea la RAM, enciende los led del teclado, comprueba los modos de direccionamiento.

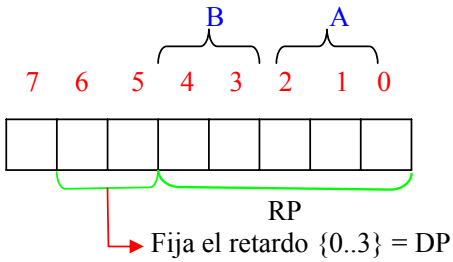
Si todo va bien envía un código al interfaz del teclado (FCh) tarda entre (600-900msg). Esta orden la puede ordenar el interfaces por algún error de paridad u otros.

Las ordenes del sistema al teclado son:

FFh: Reset

FEh: Resend: volver a enviar el último código.

F3h: seguida de 0 byte de datos, en ese byte es donde de fija la frecuencia de repetición del teclado con las teclas (set typematic Rate / Delay)



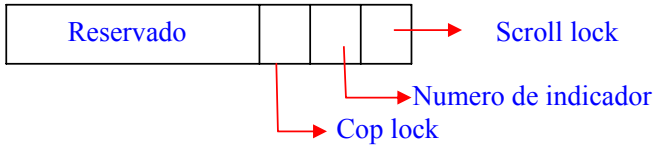
Retardo = 1 + DP * 250 msg (± 20 %)

RP → tiene dos subcampos:

A → {0…7}
B → {0…3}
} Periodo de repetición (β + A) * 2 * 0.00417 seg

Por defecto después de un BAT el retraso es de 500 msg con DP = 1 y el typematic de 10 caracteres/seg.

EDh (seguido de un byte): set /Reset mode Indicater, se puede encender o apagar 1 a 1 los led del teclado.



Estas órdenes se ponen escribiendo en el puerto 60h, primero se deben comprobar los bit de estado, esperar a que esté activada la línea o otra cosa sería enmascarar el bit que habilita el teclado.

Teclados al sistema, son códigos automáticos en los que no tenemos control, suelen ser códigos de error al sistema:

FEh Resend

ACh Reconocimiento, diagnostico (éxito o fallo).

Rutina para escribir un byte en el teclado.

Se lo pasamos AL por el 60h, si algo fallo lo devolverá en AH:

KEYBOARD_WRITE PROC

```
        ; entrada: AL = byte a enviar (por el puerto 60h).
        ; salida: AH = 0       si todo va bien.
        ;         AH = 1       si hay un error.

        push ax
        push dx
        mov dl, al
        xor ax, ax  ┐
                    ├  Si todo va mal se leerá el puerto 64h registro de estado.
@_1:                ┘
        in al, 64h
        IODELAY   ; marco con jmp $+2
        test al, 2
        jz @_2
        loop @_1
        mov ah, 1
        jump @_3

@_2:            ; escribe
        mov al, dl
        out 60h, al
        IODELAY
        xor ah, ah

@_3:
        pop dx
        pop  cx
        ret

KEYBOARD_WRITE ENDP.
```

Tema 7

La comunicación con el exterior

7.1.- Interfaz programable de periféricos, PPI 8255A

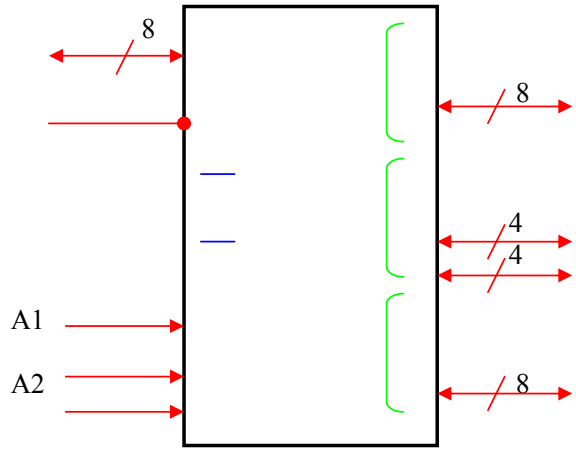


Figura 19. Interfaz programable de periféricos.

Contiene un registro de control y tres puertos direccionables separados, denominados A, B, C.

La señal $\overline{CS}$ determina si se está accediendo al 8255A y las señales $\overline{RD}$ y $\overline{WR}$ determinan la dirección de los accesos.

Las señales aplicadas a las patillas A0 y A1 determinan a su vez a cual de los cuatro registros estamos accediendo (están siendo direccionados).
DIRECCIONAMIENTO

A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	0	1	0	Puerto A al bus de datos.
0	1	0	1	0	Puerto B al bus de datos.
1	0	0	1	0	Puerto C al bus de datos.
0	0	1	0	0	Puerto A lee del bus de datos.
0	1	1	0	0	Puerto B lee del bus de datos.
1	0	1	0	0	Puerto C lee del bus de datos.
X	X	X	X	1	Estado de alta impedancia.
1	1	1	0	0	Se accede al registro de control del bus de datos.

Si D7 = 1 → se transfieren al registro de control.
Si D7 = 0 → la entrada del bus de datos se trata como una interrupción.

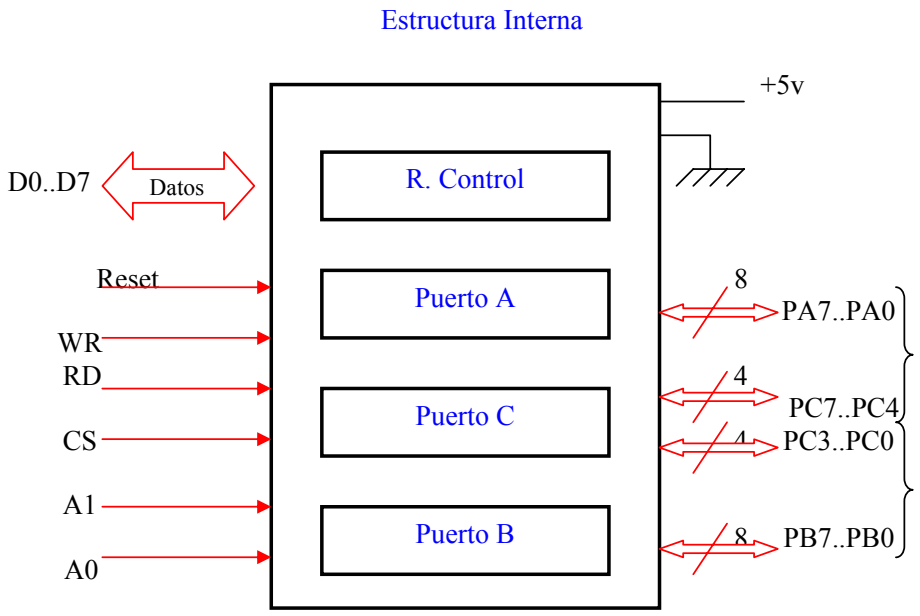
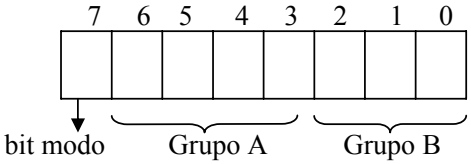


Figura 20. Estructura interna del interfaz programable de periféricos.

Funcionamiento:

Se programa el registro de control que tiene varios campos:



Existen dos modos distintos de funcionamiento, según esté el bit de modo:

0 → Set / Reset bit a bit.

1 → Funcionan los modos, que básicamente son tres.

Para la utilización de los modos los puertos se agrupan en dos grupos:

Grupo A { Puerto A
Parte inferior del C (4 bits)

Grupo B { Puerto B
Parte superior del C (4 bits)

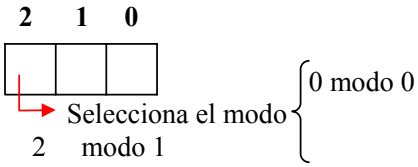
Cada grupo puede tener un modo diferente de programación.

Grupo A:

Se controla con los bit del 6 … 3 de forma que el 6 y el 5 controlan el modo de funcionamiento:

6	5	
0	0	Modo 0
0	1	Modo 1
1	0	} Modo 2
1	1	

Grupo B:



Los otros bits del grupo A se refieren a la parte C del grupo (Cl). El bit 3 y el bit 4 controla el puerto B de la misma forma que el grupo B con el bit 0 controla (Ch) y con el bit 1 controla el puerto B:

Modo 0: El bit 4 y el bit 1 de cada grupo se utiliza en modo 0 para:

0 → Salida

1 → Entrada

Si un grupo está en modo 0 está dividido en dos conjuntos. Para el grupo A este conjunto es el puerto A y los cuatro bits más bajos del puerto C y para el grupo B son el puerto B y los cuatro bit más altos del puerto C.

Cada uno de los conjuntos puede ser utilizado de entrada o de salida pero no de entrada – salida simultáneamente:

3 → Puerto A
4 → Inferior de C
1 → Puerto B
0 → Superior de C

} Disponemos de 16 combinaciones de entrada /salida.

Si uno de estos bit está a cero entonces el conjunto correspondiente se utiliza como salida, si está a uno como entrada.

Las salidas están *latcheadas* y las entradas no.

Modo 1: cuando el grupo A está en este modo se utiliza como entrada o salida según el bit 4 del registro de control y la mitad más alta del puerto C se utiliza como señales de control y protocolo.

Para la entrada los cuatro bit del puerto C se le asignan los siguientes símbolos y definiciones:

Bit 4 ($\overline{STBA}$): un cero en esta patilla hace almacenar PA7..PA0 en este puerto A.

Bit 5 (IBFA): Indica que el buffer de entrada está lleno. Está a uno cuando el puerto A contiene datos que no han sido recogidos por la CPU.

Cuando está a cero el dispositivo puede aceptar un nuevo byte del interfaz.

Bit 6 ($\overline{ACK}$): entrada de señal de reconocimiento, señal del receptor del dato, para decir que ya lo ha recogido (salida).

Para salida:

Bit 4 y bit 5: como bit el 6 y bit el bit descrito anteriormente.

Bit 7 (UBFA): Indica que el buffer de salida está lleno. Envía un 0 al dispositivo cuando el puerto A está enviando nuevos datos que deben ser recogidos por el dispositivo.

Bit 6 ($\overline{ACK}$): el dispositivo pone un cero cuando acepta los datos del puerto A.

En el modo 1, PC3 se denomina INTRA y está asociado al grupo A. Se utiliza como línea de petición de interrupción y se une a una de las líneas IR del bus del sistema.

Para entradas se pone un uno cuando entra un dato nuevo al puerto A (está controlado por PC4), se pone a 0 cuando la CPU recoge el dato.

Para salidas esta patilla se pone a 1 cuando el dispositivo recoge el contenido del puerto A.

Si el grupo B está en modo 1 el puerto B es de entrada o de salida según el bit de D1.

Para entradas desde el puerto B el bit 2 y bit 1 se denominan respectivamente STB e IBF y sirve para el mismo propósito que para el grupo A.

De forma similar las salidas de los bit 2 y bit 1 se denominan PBF y ACK.

Bit 0 será INTRb y se utiliza de forma análoga a la de A.

Grupo A bit 4 3

		0	0	Salida
		0	1	Salida
		1	0	Entrada
		1	1	Entrada

Puerto A

Grupo B	bit	2	1	0	
		1	0	0	Salida
		1	1	X	Entrada

Modo 2: (solo para grupo A). En este modo el puerto A es bidireccional y los cuatro bit del C son:

Bit 4 (STBA): un 0 en esta línea hace que los datos del PA7..PA0 sean almacenados en el puerto A.

Bit 5 (TBFA): se pone a 1 cuando el puerto A se llena con un nuevo dato de las líneas PA7..PA0 y se pone a 0 cuando los datos se toman de la CPU.

Bit 6 (ACKA): indica que el dispositivo esta preparado para aceptar datos de PA7..PA0.

Bit 7 (OBFA): se pone a cero cuando A se llena con un nuevo dato de la CPU, y se pone a 1 cuando el dispositivo toma los datos.

Si el bit 7 del registro de control esta a 0 en los bits 3 2 1 se da el número del pin que se quiere colocar el valor, a continuación en el bit 0 se coloca el valor deseado.

Si este bit (7) esta a uno, los datos al registro de control si esta a 0, estos datos van al puerto C y se colocan según lo dispuesto antes.

Mientras A esta en modo 2 el grupo B puede esta en modo 0 o 1, sin embargo, si el grupo B esta en modo 0 solo se puede utilizar del puerto C los bit 2 1 0 pues el 3 lo tiene A.

En los tres modos el puerto C refleja la señales del PC2..PC0 y puede ser leído en cualquier momento con una instrucción IN.

7.5.- El puerto serie RS-232C

Comunicación serie

Se utiliza para mayores distancias, son encadenamientos de bit agrupados en palabras o caracteres, desarrollando mensajes como dirección, control y datos.

Existe mecanismos para sincronizar los bits de la línea y estos pueden ser a nivel de bit, a nivel de carácter o a nivel de mensaje completo.

Tipos de comunicación serie.

Síncrona: se produce entre dos dispositivos que trabajan a una misma frecuencia, a través del enlace serie el emisor y el receptor ya saben todo. Existe otra línea paralela a esta que lleva la frecuencia de reloj.

Para dispositivos remotos, muy distantes esta no tiene sentido, pues es muy difícil que tengan los tiempos (incluso en paralelo) iguales.

Asíncrona: para grandes distancias. El más extendido es el RS232, por una línea serie se envían caracteres.

Cada carácter empieza con un bit 0, luego se envía el carácter bit a bit empezando por el menos significativo y de 5 a 8 bits de largo.

Es opcional un bit de paridad, par o impar lo que se establezca de mutuo acuerdo.

Se termina con uno o dos bits de cierre.

Lo que dura un bit es lo que determina el reloj local con una frecuencia de 300…600….1200…2400…….19200 es la máxima que soporta el BIOS.

Aunque el BIOS no pase de aquí se puede superar programando el controlador serie, lo que puede llegar hasta 115200

Test port proc

```
; entrada DX = puerto base
; salida  BL = tipo de chip

xor bl, bl
mov dl, dx
cli
plabi
mov ah, 5ah
iorest
jne @@_1
sti
inc bl,
add dx, 7
mov ah, 8Ah
iotest
jne @@_1
mov ah, 0A5h
iotest
jl @@_2
cmp bh, 80h
je @@_6
mov al, 60h
out dx, al
IODELAY
in al, dx
IODELAY
mov al, 0
out dx, al
IODELAY
and bh, 60h
je @@_B

@@_3:

mov dx, di
add dx, 7
mov al, 80h
out dx, al
```

```
@@_1:

        jmp @@_9

@@_2:

        mov dx, di
        cli
        add dx, 2
        in al, dx
        IODELAY
        and al, 0C0h
        cmp al, 0C0h
        jl @@_7
        mov al, 1
        out dx, al
        IODELAY
        in al, dx
        IODELAY
        mov bh, al
        mov al, 0
        out dx, al
        IODEALAY
        and bh, 0C0h
        cmp bh, 0C0h
        je @@_7     ; es un 1655X
        cmp bh, 40h
        je @@_8 ; es un 16550
        DLAB1
        add dx, 2
        mov ah, 7
        IOTEST
        mov bh, al
        mov al, 0
        out dx, al
        cmp bh, 7
        je @@_A
        mov dx, di
        add dx, 4
        mov al, 80h
        out dx, al
        IODELAY
        mov ah, al
        mov al, 0
        out dx, al
        test ah, 80h
        jnz @@_E

@@_8:
        IODEALY
        in al, dx
        IODEALY
        mov ah, al
        mov al, 0
        out dx, al
        test ah, 80h
        jnz @@_5

@@_4:

        ; es un 8250A
        mov bl, 2
        jmp @@_9

@@_5:

; es un 16C450
        mov bl, 3
        jmp @@_9

@@_6:

        mov bl, 4
        jmp @@_9

@@_7:

        mov dx, di
```

```
        ; es un 16550
        mov bl, 5
        jmp @@_9

@@_E:

        mov bl, 6
        jmp @@_9

@@_A:

        ;es un 16552
        mov bl, 7
        jmp @@_9

@@_B:

        mov bl, 8

@@_9:

        mov dx, di
        DLAB0
        sti
        ret

Test_ port endp.
```

Tema 8

Periféricos externos

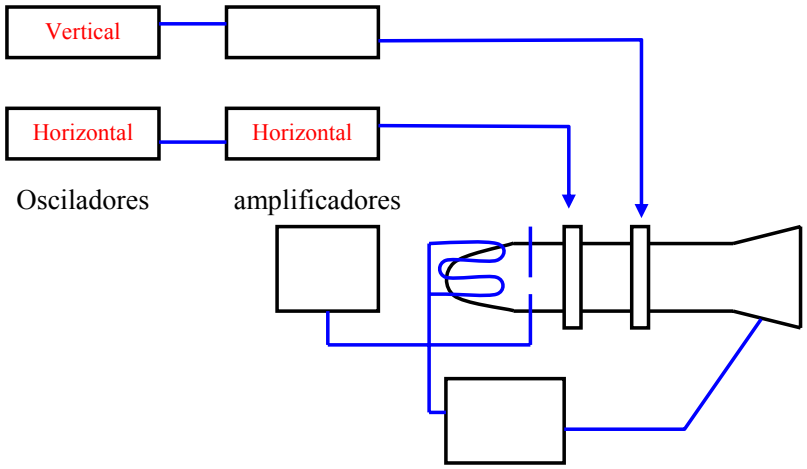
8.3.- El subsistema de vídeo

8.3.1.- La pantalla CRT

El tubo esta recubierto de sales de fósforo, que son fosforescentes (excitan electrones y lo transforman en destellos luminosos).

El haz de electrones se direcciona mediante unas bobinas (deflectoras) que hacen llegar el haz a todos lo puntos de la pantalla; para ello se les hace llegar una señal periódica procedente de los amplificadores y a estas le viene la señal de los osciladores cuyas señales están calculadas de tal forma que se puede poner el haz en cualquier lugar de la pantalla. Está hecho de tal forma que recorre en zig zag toda la pantalla debido a que la señal vertical es de mayor duración que la señal horizontal.

Un monitor tiene el tubo de rayos catódicos y los dispositivos para controlar la dirección del haz y su intensidad.



Barrido Izq.- Der. barrido horizontal.

Barrido Der.- Izq. retorno horizontal.

Barrido Arriba- abajo barrido vertical.

Barrido Abajo – Arriba retorno vertical.

Trama: conjunto de líneas que se crea en la pantalla.

Cuadro: Cada barrido de la trama de la esquina superior izquierda a la otra esquina y volver se denomina cuadro.

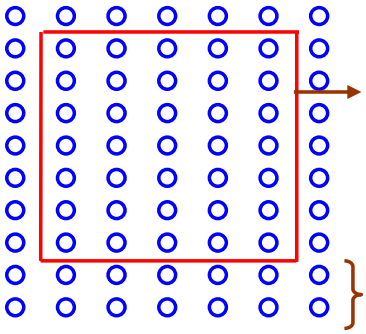
Existen dos señales externas que son:

VSYNC y HSYNC, que sirven para sincronizar las señales verticales y horizontales.

Tiene otra entrada que es la entrada de vídeo, que va al amplificador de vídeo. Lo normal es que lo que llegue es una sola señal y esta la divide.

Las figuras se forman encendiendo y apagando el haz en los momentos adecuados.

Los caracteres se construyen mediante una disposición de una matriz de puntos:

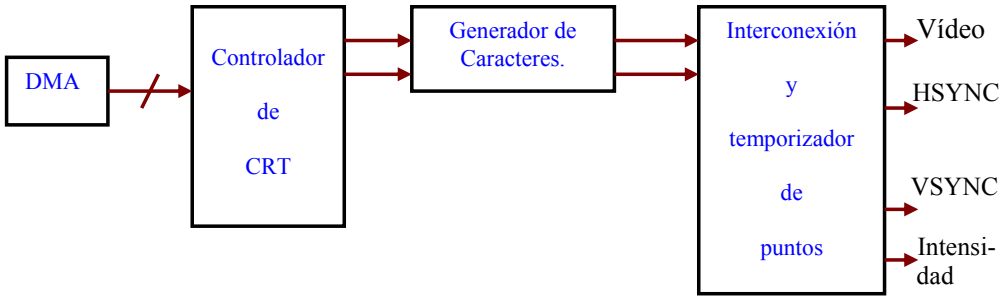


5 * 7 para cada carácter.

para separar las filas puede tener otra línea más para el carácter del subrayado o el cursor.

Cada barrido horizontal barre una línea de puntos y para formar todos los caracteres hará sucesivos barridos para hacer una línea de caracteres (fila).

Debe existir un controlador para controlar todos los dispositivos internos del monitor.



Los tres bloque se integran en un chip o por separado, se puede encontrar de las dos formas.

En los primeros PC eran monocromos, una TV se puede utilizar como monitor.
En los monitores de color tiene el diagrama de antes pero en tres dimensiones:

	NTSC	PAL	SECAM
Nº líneas por frase	525	625	625
free campo	59.94	50	50
Entrelazado/	2.1	2.1	2.1
Free imágenes	29.97	2.5	2.5
Free líneas	19739.964	15.625	15.625
Aspecto (ancho)	4/3	4/3	4/3
Secuencia de barrido	179/der y de arriba / abajo		
Gamma de la señal de imagen	0.45	2.2	2.2
Ancho de banda nominal	4.12	5.5	6.0 , 6.5
Free subpatch de crominancia	3.579545	4.3361875	4.4375

La desviación del haz de rayos catódicos se consigue desviándolo con campos eléctricos o mediante unas bobinas, por lo que al chocar produce un punto luminoso y tarda un cierto tiempo en disiparse.

La duración entre la excitación y el momento en que la emisión de luz ha perdido un 10% de su intensidad se denomina persistencia (en torno a los 10 y 60 µs). Esto permite que se vea y no se mezcle con la siguiente imagen.

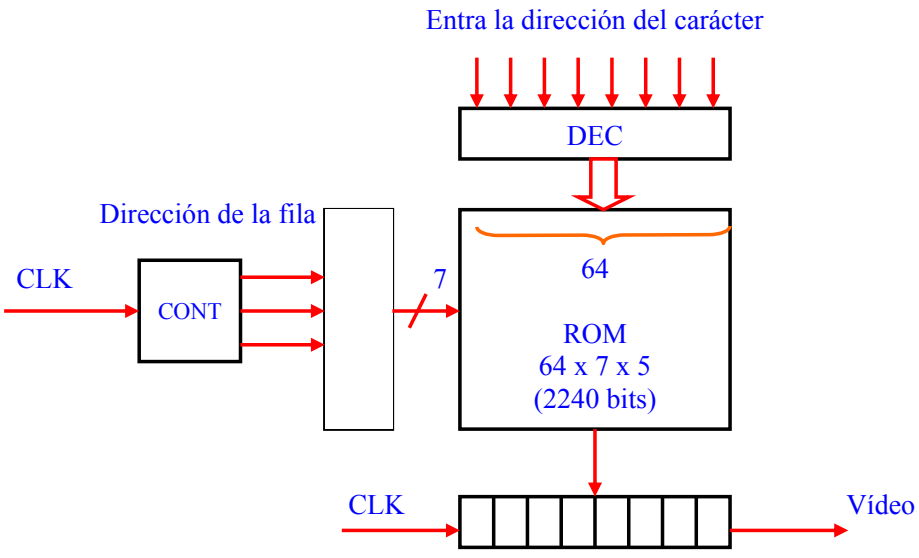
Entrelazado: consiste en que cada ciclo de barrido vertical se refresca solo la mitad de las líneas horizontales. Con esto se consigue que con menos frecuencia de barrido haya más líneas de exploración. Uno de los problemas es que se parte la cadena de barrido, pues la línea de barrido se refresca a la mitad de frecuencia. Para los monitores esta frecuencia sería muy baja pues los caracteres cambian constantemente.

Generación de la señal de vídeo.

Al monitor se envían tres señales, una de vídeo y dos de sincronización. En monocromo la señal de vídeo solo diferencia si esta encendido o apagado, es decir, solo diferencia dos niveles.

Para representar caracteres se borra las líneas de la matriz de puntos pero la líneas barridas deben contener la información de barridos anteriores.

Si tenemos una línea completa de caracteres, tendríamos que poder mostrar toda la línea en cada barrido, para esto se emplea una ROM (64 * 5 * 7 bits).



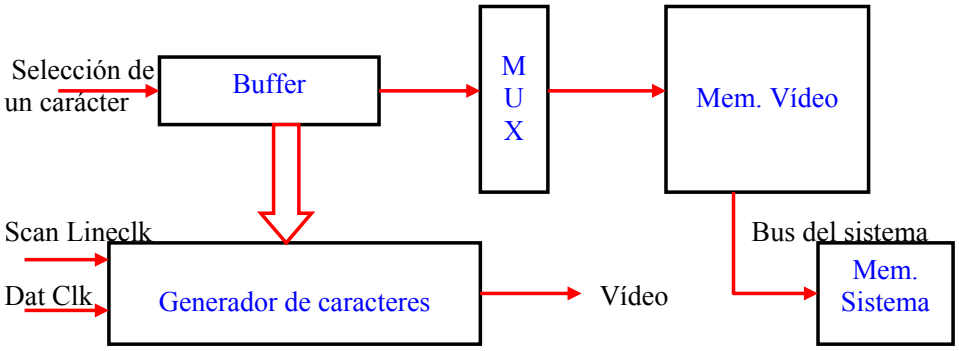
La línea no se modifica hasta mostrar por pantalla los 80 caracteres, luego se pasa a la segunda fila y se muestra toda la líneas y así sucesivamente hasta mostrar todas las líneas pertenecientes al carácter.

Una vez representada la primera línea se vuelve a la siguiente línea y otra vez igual , lo que se hace habitualmente es colocar un contador que cuenta de 0...6 con una frecuencia de barrido horizontal, que es lo que tarda en borrar una línea.

A la entrada de la dirección de carácter se coloca con el decodificador unos latches que ya vienen integrados en un chip, el denominado generador de caracteres).

El tamaño de la ROM depende del juego de caracteres que queramos utilizar.

Al bloque anterior se le añade un buffer de datos de 80 caracteres



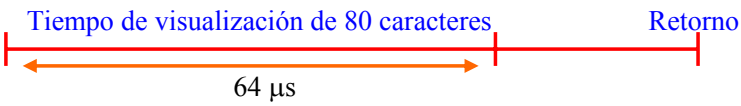
Irá seleccionando cada carácter de la línea, por lo que podemos volver a poner un contador de 0..80, p.e. Pero esto se quedaría corto, lo que realmente sería útil sería tener una memoria que contenga a todos los caracteres de una pantalla (memoria de pantalla), esta memoria es de lectura y escritura, por lo tanto tiene que poder comunicarse con el sistema.

En algunos casos la memoria del sistema se utiliza para la de vídeo, otras la pueden tener separadas pero tiene que haber comunicación o bien a través del bus o mediante un buffer intermedio. Para intercomunicar la memoria de vídeo y la del sistema tiene que tener una circuteria especial para pasar información directamente de una a otra (en algunos equipos es con DMA).

Todo este conjunto de lógica es un controlador de CRT, este se comunica con el sistema a través del bus de datos y de direcciones. Todo lo que es pintar en pantalla no utiliza el bus del sistema para no sobrecargar este bus. La lógica de unión de la memoria esta sincronizada con las señales SYNC.

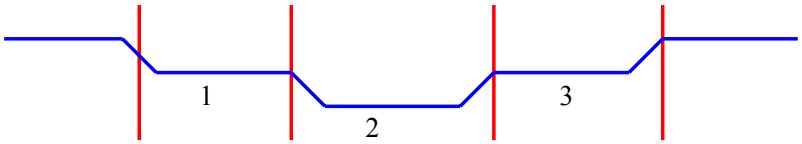
Esquema de sincronización de todos los CLK.

La frecuencia de la señal HSYNC es f = 15,625 y el tiempo de la misma T = 64 μs.



La información en una línea de barrido dura 64 µs por cada línea de barrido, durante este tiempo tiene que ir el tiempo de visualización de los 80 caracteres y cierto tiempo de retorno.

La señal HSYNC da su pulso en el tiempo de retorno, esto es así porque el tiempo de retorno horizontal se puede dividir en tres intervalos:



1.- Retraso (margen de pantalla a la derecha).

2.- Sincronismo horizontal (tiempo en retornar a izquierda).

3.- Retraso de barrido (margen a la izquierda).

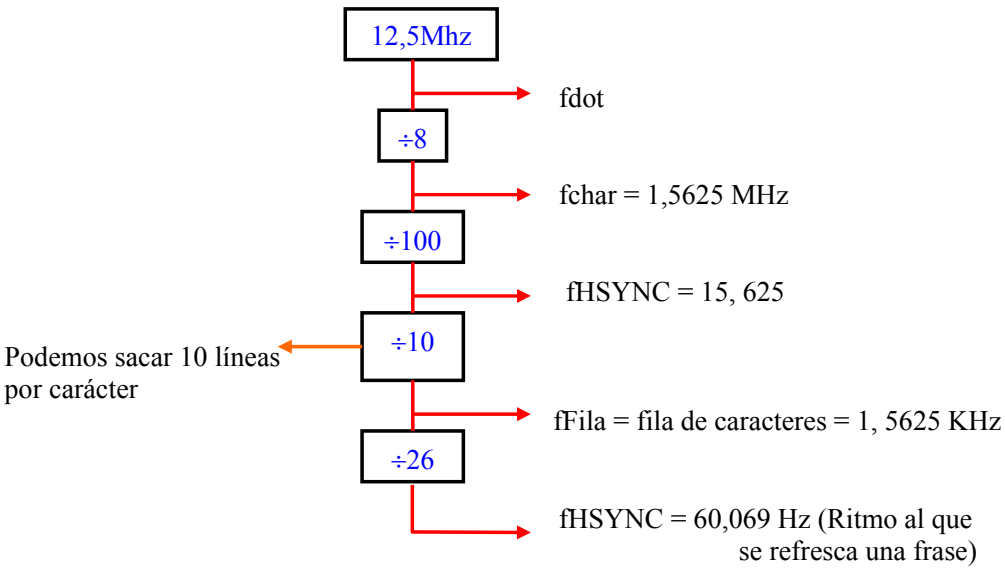
La duración de cada intervalo esta determinado por cada monitor (10.. 12 µs, valores típicos).

El tiempo total de retorno suele ser el 20% del tiempo total de barrido.

Tiempo de Carácter es igual a 0,64 µs, es la centésima parte del tiempo en escribir una línea, a su vez cada carácter tiene 8 puntos por carácter, por lo que Tiempo de Carácter / 8 sería la duración de un punto en pantalla.

Tiempo de punto = 640 nsg / 8 = 80 nsg → fdot = 1 / 80 ns = 12,5 MHz.

Tenemos un reloj de :



Esta cadena de temporización la tienen todos los adaptadores de vídeo, se utilizan divisores de frecuencia para sacar cada frecuencia que realmente hace falta.

Las VGA`s modernas se siguen por tres relojes:

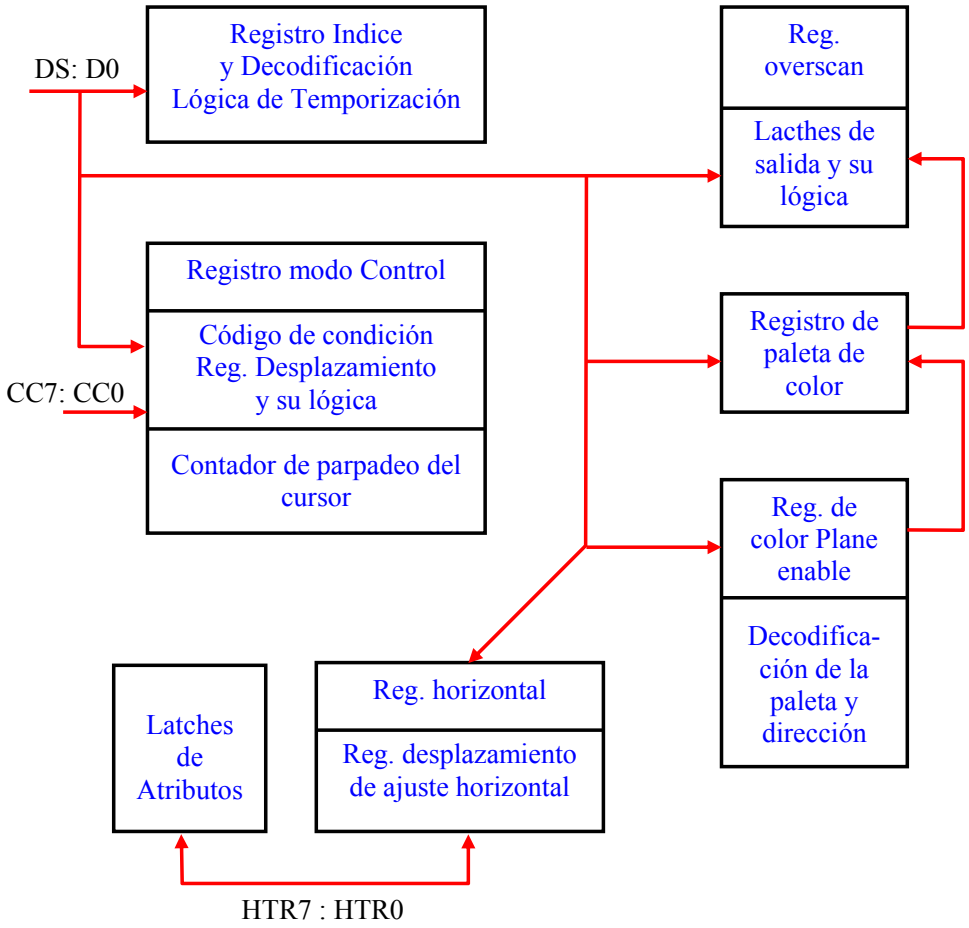
- El de la CPU para comunicarse con el sistema y la memoria.
- El de vídeo para imágenes.
- Reloj interno que lo optimizan para velocidad de memoria RAM de vídeo, es independiente a los dos anteriores.

Controlador de Atributos.

Controla los parpadeos y subrayados en modos alfanuméricos. La capacidad de ajuste fino horizontal es la capacidad de hacer scroll horizontales pixel a pixel.

Posee un total de 22 registros accesibles, incluido el registro índice: TI, T0...T20. El índice es un registro al que se accede de forma distinta. Todos los registros se acceden por :

Indice : 3C0
S : 3C1



Controlador de Gráficos.

Proporciona una línea de acceso de L/E de la memoria de pantalla, este dispositivo es el que maneja los datos, controla los cuatro planos de pantalla. Permite que los datos a escribir en pantalla pueden ser manipulados. Formatea los datos a fin de poder utilizarse con modos antiguos, es decir, para hacerlos compatibles con anteriores placas de vídeo.

Proporcionan comparadores de color, permite acceder a la memoria de pantalla con un ancho de 32 bits, combina la memoria y atributos para la salida de los buses de pixel.

Posee nueve registro programables para gráficas y accede por 03CE y 03CF.

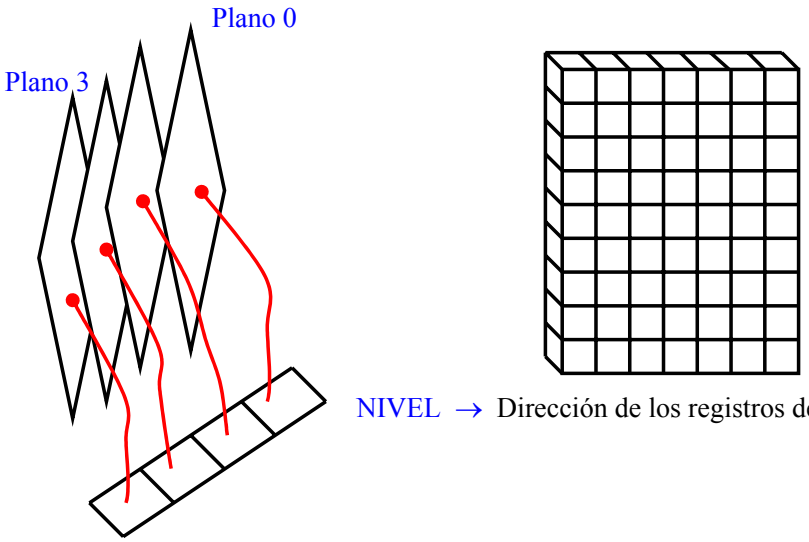
Memoria de VGA.

La memoria esta organizada en cuatro planos, cada uno de los cuales tiene hace de Bit-Map y cada byte controla a 8 pixel consecutivos. Una característica de la VGA es que se distingue entre ancho lógico y ancho real, estos no tienen por que coincidir, por lo que se puede definir una pantalla mayor que la real (4080 pixel de ancho) de igual de alto, hasta lo máximo de la memoria.

Los cuatro planos se direccionan con la misma dirección

A000 : 0000 64 k.
A000 : FFFF

Un byte de este rango permite accederá cuatro bytes de la VGA, por estar en cuatro planos paralelos. Se escriben a la vez los cuatro, pero no necesariamente lo mismo.



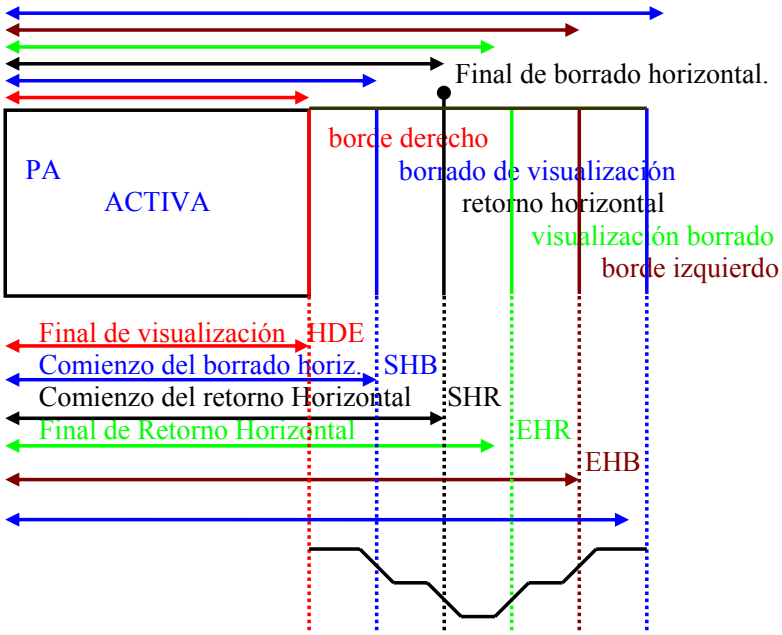
NIVEL → Dirección de los registros de paleta.

Un pixel es la combinación de los bit de cada plano, un pixel puede ser 0..15 y la VGA manejan los cuatro planos en paralelo.

No es la única forma de controlar los colores pero es la más optimizada.

Los modos standard de la VGA vienen programados en la BIOS, en algunos modos no manejan los planos.

Registros de la VGA



Durante el tiempo que transcurre entre el borde derecho y el borde izquierdo se impide mandar información a la pantalla.

HT = R0 + 5 (5... 260 rango)

EHB = está en dos registros:

bit 7 de R5 encadenado con los bit 4...0 de R3
EHB = R5[7] * R3[4..0] = (0..63)

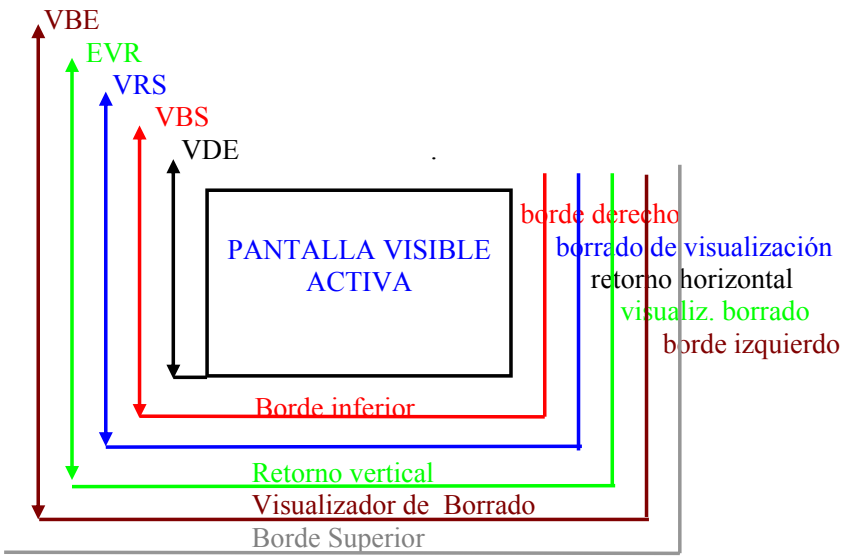
EHR = R5 [4..0] (0..31)

SHR = R4 (0..255)

SHB = R2 (0..255)

HDE = R1 + 1 (1..256)

VT



Algunos de estos registros no miden el tiempo desde el principio sino el ancho de cada fase.

VT = R7[5] * R7[0] * R6 + 2 (2..1025)

VBE = R22[6..0] (0..127)

EVR = R17[3..0] (0..15)

VRS = R7[7] * R7[2] * R16 (0..1023)

VBS = R9[5] * R7[3] * R[21] (0..1023)

VDE = R7[6] * R7[1] * R[18] (0..1023)

Los tiempos de sincronización vertical son tiempos de barrido horizontal.

Hay dos bits que son de protección de acceso CR (R3[7]), activa o desactiva la posibilidad de acceder a R16 y R17 (retorno vertical) el bit 7 de R17 es el bit de protección para R0..R7.

Los bits 5 y 6 del R3 (DES) es el sesgo horizontal, se puede ajustar la sincronización horizontal.

Los bits 6 y 5 de R5 son un campo de sesgo, de ajuste como el anterior.

BW = R17 [6] :

0 → 3 ciclos
1 → 5 ciclos

} número de veces que se refresca la memoria hori-- zontal.

MSL : R9 y R20 se controla con R9[4..0] (número de barrido – 1) de cada fila de caracteres.

HL: R20[4..0] posición del subrayado, a que línea de barrido le corresponde la cuestión del subrayado.

CS : R10[4..0], comienzo del cursor dentro de lo que es la fila de caracteres.

CE: R11[4..0], final del cursor.

C00: R10[5], cursor en off { 0 aparece ; 1 impide que aparezca

CSK (Ajuste): R11[6..5], ajuste del cursor.

CL: R14, R15, posición del cursor (el cursor parpadea a una frecuencia de 1/16).

Cuando barre la pantalla la información, esta sale de la memoria de vídeo, que en principio se sabe en la posición en la que se está, esta se programa en:

SA : R12.R13, comienzo de la memoria de vídeo. Con esto es con lo que se consigue el movimiento de scrool.

En R20 queda el bit 6, DW = R20[6], si tiene un 0 se direcciona de word a word y se tiene un 1 de doble word en doble word.
El bit 5 que es CB4 = R20[5], cuenta con cuatro relaciones con DW:

{ 0 no tiene ninguna misión.
1 activa CB4 { 0 sincronismo normal.
1 divide por cuatro el reloj de caracteres.

2T4: es igual a R9[7], si es un 1 duplica cada línea el barrido dividiendo el reloj de barrido horizontal por dos (baja resolución).

R19: indica el ancho lógico de la zona visualizada correspondiente, el valor guardado es la diferencia entre las direcciones de dos pixel vecinos verticalmente.

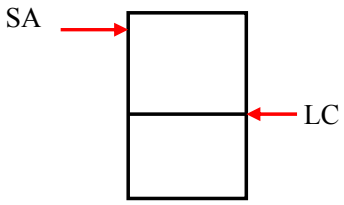
R23[6], si tiene un 0 significa que el modo es un word, si tiene un 1 significa que el modo es de tipo byte.

R23[5]: se usa para emular la EGA.

R23[7]: habilita la temporización, si tiene un 0 sitúa todas las señales en modo suspendido y se hace un Reset. Si tiene un 1 se muestra normalmente.

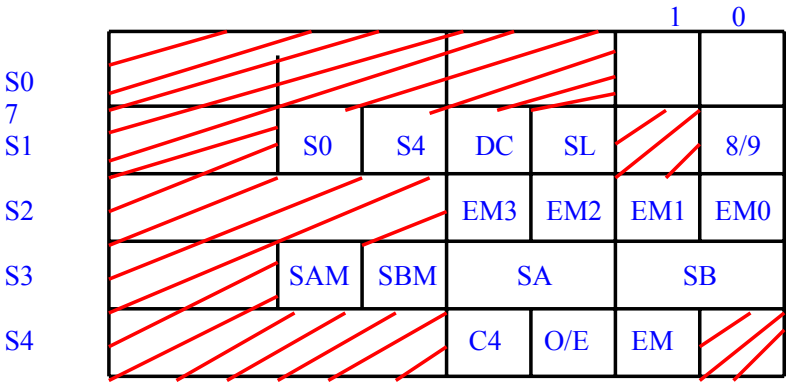
R24: permite dividir la pantalla en dos trozos , pero necesita dos bit adicionales.

R9[6] · R7[4] · R24 = LC, si el número total de líneas visualizadas es menor que LC, la pantalla queda dividida



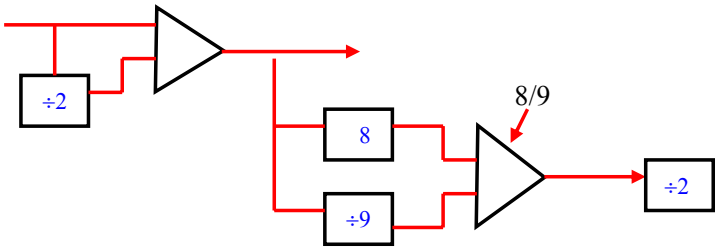
R8[0..4]: fija la fila de barrido dentro de la fila de caracteres, es decir, en que línea de barrido comienza a visualizar.

Registros del SEQU



S0: Reset, si esta a 1 desconecta la pantalla y toda la lógica generadora de la imagen. (screen off).

S1: clocking mode, master clock .



S2: MP Mask, habilita los planos de memoria 0, 1, 2 y 3. Si hay un 0 en alguno de estos bit el plano de la memoria correspondiente quedaría inaccesible.

S3: character map select, selecciona los mapas de caracteres, el B es el primario y el A es el secundario. Los bit SAM y SBM se añaden a la VGA y complementa los respectivos campos para poder seleccionar hasta 8 mapas de caracteres:

S3[4] · S3[1] · S3[0] = (0 .. 7) ocho posibles mapas de caracteres primarios.

Lo que hace es comparar los de la ROM-BIOS y los carga a los dos primeros planos de pantalla. La VGA reserva 64K para cada mapa y puede almacenar hasta 8 al mismo tiempo cada uno de 18K

S4: memory mode, controla la forma de funcionamiento de la memoria de pantalla.

EM: si tiene un 0 memoria de comienzo de 64 K, si tiene un 1 memoria de más de 64 K.

O/E: este bit determina si el procesador cuando accede a la memoria de pantalla lo hace secuencialmente o por paridad. Si tiene un 1 es que accede de forma secuencial. Si tiene un 0 , la paridad de la dirección determina a que plano va.

Este bit tiene que estar combinado con el de gráficos.

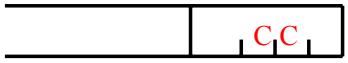
C4: controla la manera en la que se accede a los planos de memoria de pantalla, de tal forma que S2 determina que planos se accede pero en modo de 256 colores solo se accede a un plano cada vez, es por lo que se ha cambiado este bit y se hace un acceso y se actualizan los cuatro planos.

En este modo 13h (320 x 220) 250 colores, los dos bits bajos de la dirección determinan a cual de los cuatro planos se accede, escribiéndolo en direcciones consecutivas de cada plano, dejando huecos en medio.

Controlador Gráficos.

G0 SET / RESET Contiene el color del punto.

G1 Habilitador del SET / RESET.



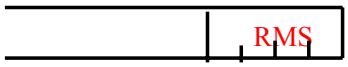
G2

G3 Data Rotate: rotación de datos.

RC: determina cuantos bits se van a rotar a la derecha el dato.

FS: selecciona que operación se hace sobre el dato en la alu:

FS	
0 0	
0 1	AND
1 0	OR
1 1	XOR.



G4 Selecciona la información de qué plano llegará a la CPU, se puede acceder a los cuatro planos simultaneamente.

G5 WM: determina el modo de escritura.

RM: read mode

I/O: pareja del anterior (par / impar)

SR: campo que afecta con el comportamiento de la VGA con controladores antiguos.

G5[6]: Si tiene un 1 habilita el modo de 256 colores, si tiene un 0 habilita el modo de menos colores.

G5[5]: si tiene un 1 significa que tiene compatibilidad con CGA, si tiene un 0 tiene modos VGA y EGA.

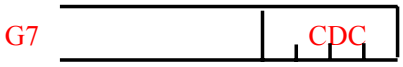
MM: Memory map

0 0 → A0000 - BFFFF 128 K
1 1 → A0000 – AFFFF 64K
1 0 → B0000 – B7FFF 32K
1 1 → B8000 - BFFFF 32 K

COE: Encadena par / impar, se utiliza para emular tarjetas antiguas:

0 → EGA, VGA
1 → antigua MDA.

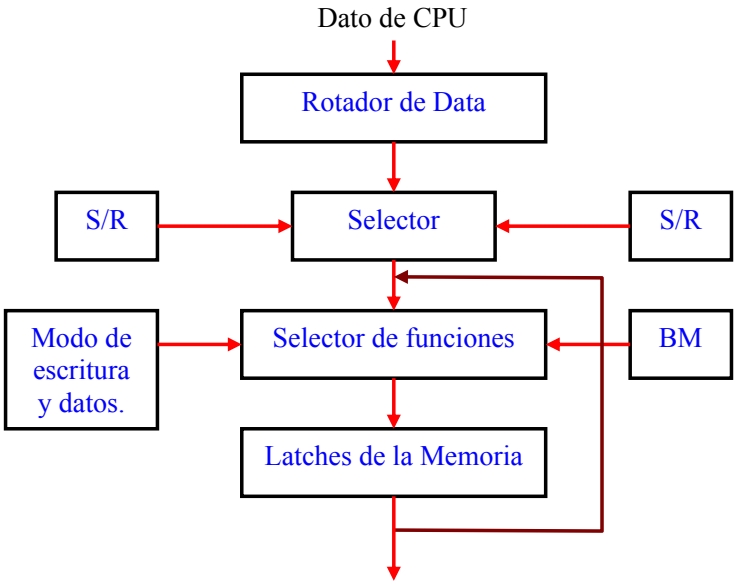
G/A: si tiene un 0 modo alfanumérico, si tiene un 1 modo gráfico.



Color don´t care.

Mascara de bit, controla los accesos a la memoria.

Esquema de escritura.



Modo 0: es el que más posibilidades brinda.

Cuando hacemos una escritura de modo 0 el dato o procede de la CPU o lo hemos puesto en el registro S/R si procede de la CPU.

Si lo que se quiere es pintar una zona , el color se puede poner en el S/R, la orden de escritura es la que decide si este dato llega a memoria de pantalla. En la ALU se puede enmascara con un AND, OR, XOR, pero lo normal es dejarlo pasar. Se enmascara bit a bit y posteriormente se deja pasar.

Modo 1: tiene un objetivo especifico que es el de escribir directamente el contenido de los latches de memoria, pues esta pensado para mover zonas de pantalla de un lado a otro. Es eficiente pues se hace de 32 bit en 32 bits.

Modo 2: no existe rotador, se ignora toda la lógica de S/R, los únicos bits que se toman en cuenta son los 4 bits bajos.

Los 4 bits bajos entran en los expansores, que están controlados por el registro S/R de 4 bits. Debajo lleva unos multiplexadores que seleccionan o el byte del expansor o el del bus interno, esto es controlado por ESR (G1).

Debajo lleva una ALU donde una entrada es la salida del mulltiplexor y otra de los latches de los planos de memoria, en todos los planos se realizará la misma función que será seleccionada con FS (G3). Son cuatro posibles operaciones, de hay el resultado a un multiplexor (8bits), donde también entra el contenido del latches directamente, este mux es controlado por el registro BM (G8), este se selecciona bit a bit, del latch o del ALU, pero no byte a byte. Puede tener el bit 3 y bit 4 del latch pero el resto es de la ALU.

Después pasa a un buffer (free state) de 8 bits vectorial que va a memoria, este buffer es controlado por EMM del S2. Sirve para permitir que entre o no el dato.

Este modo es para escribir un color concreto en un pixel concreto.

Cada pixel estará en un byte de memoria y solo utilizando los 4 bits bajos y despreciando el resto.

El color se puede escribir hasta en 8 pixel a la vez.

Modo 3: el S/R siempre esta habilitado de manera que cada bit de los 4 se expanden y es lo que entra directamente a la ALU, existe otro campo (con respecto al modo 0). El byte de la CPU pasa por el rotador y luego con BM puede enmascarar este byte, esto es lo que entra en los mux antes de entrar en los states.

Esto esta pensado para que el color se fije en S/R y lo que se va escribiendo en la CPU es un patrón de bits cambiantes. Los bit a 0 sean los que enmascaren el dato previo del plano de memoria, y por tanto que siga escribiendo el color que había en pantalla. Y si hay un 1 se escribirá el del S/R.

Pintar un punto en modo 0

```
mov dx, 0A000h      ; para llegar al byte se tiene que leer fila a fila
mov es, dx          ; es : segmento de pantalla
mov ax, coordy
mul ax, ancho_pantalla  ; según resolución  ej: (640 – 80 bytes)
mov bx, coordx
mov cx, bx
shr bx, 3    ;  divide por 8  coordx
add bx, ax      ; bx = dirección del byte donde está el pixel
                ; dentro del byte que hemos seleccionado, tenemos que
                ; seleccionar el bit que queremos
and cx, 07h     ; cx = X mod 8
mov ch, 80h
shr ch, cl
mov dx, 03CEh
mov ah, color
mov al, 0
out dx, ax  ; metemos el color del S/R
mov ax, 0F01
out dx, ax    ; G1 = oFh
mov ax, 0005   ; G5 = 0   ; modo de escritura 0
out dx, ax
mov ah, ch
mov al, 08h
out dx, ax

;  Para garantizar que solo escribimos en ese pixel se hace una lectura
;  para cargar los latches y luego se escribe.

mov al, es:[bx]         ; lectura muda para cargar los latches.
mov es:[bx], al         ; escritura

; estas dos instrucciones anteriores podemos quitarlas y poner or es:[bx],
al
```

Pintar un punto en modo 2.

```
Pixel ( int PX, int Py, char color)

Pixel proc near C

ARG  px: word, py: word, color : byte

push bp
mov bp, sp
push ax, bx, cx, dx
mov bx, py
shl bx, 2       ; bx *= 4
add bx, py      ; bx += py
```

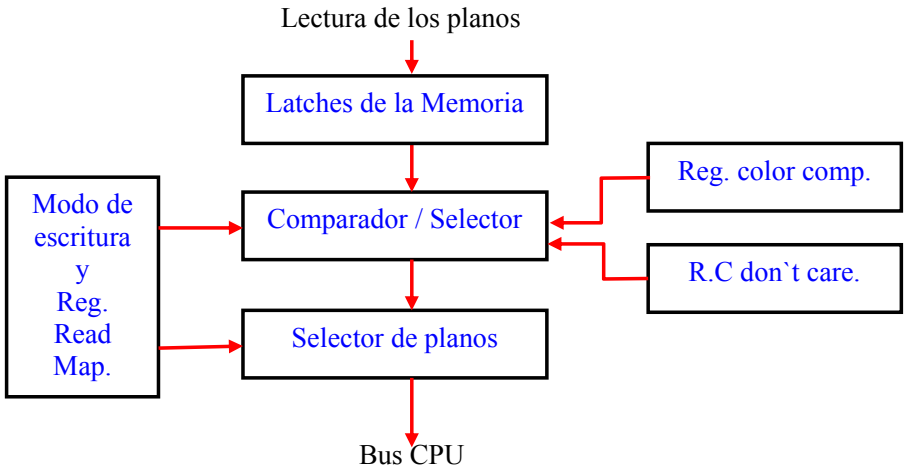
```
shl bx, 4          ; dx *= 4
; hallamos la máscara dentro del byte del pixel apuntado
mov cx, px
mov dx, cx
shr dx,  3   ; dividir por 8
add bx, dx       ; dx = dirección en memoria del byte
and cx, 07h
mov ah, 80h
shr ah, cl         ; ah = máscara del bits
mov dx, 03CEh
mov al,  08h
out dx, ax         ; escribe el BM
mov ax, 0A000h
mov es, ax
mov al, color
mov es:[bx], al
pop dx, cx, bx, ax
mov sp, bp
pop bp

ret
```

Pixel endp

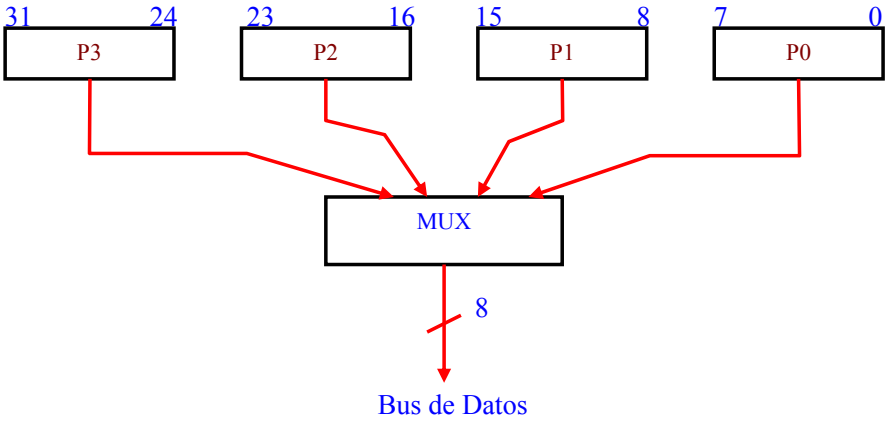
Para pintar líneas con este modo no es eficiente pues el color debería entrar por cada escritura.

Esquema de Lectura.

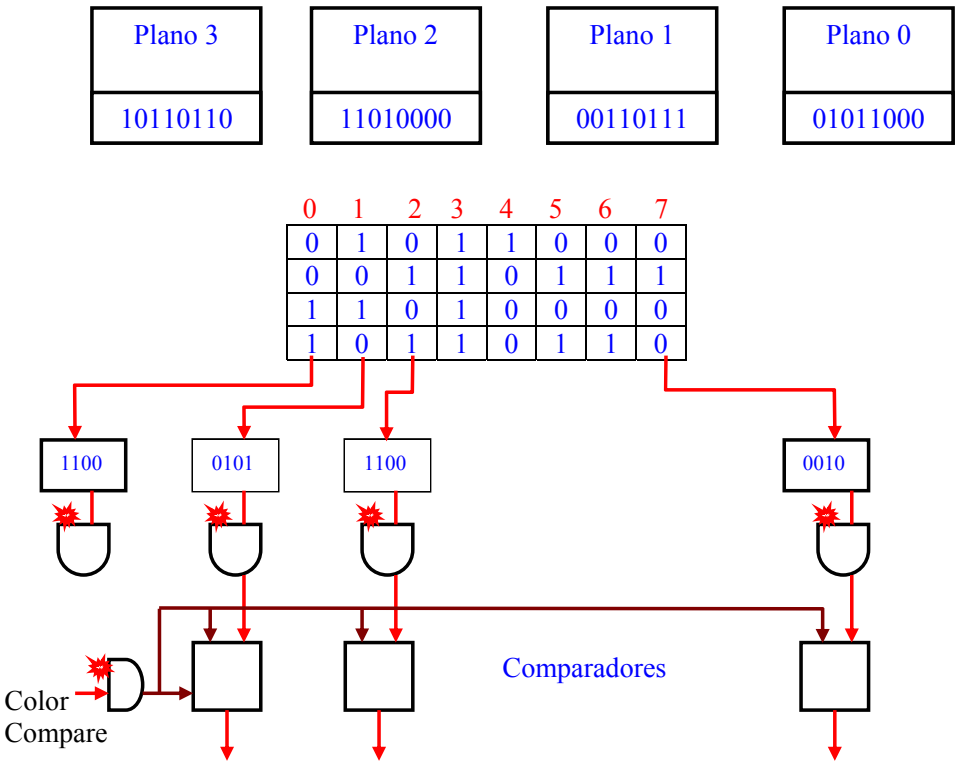


El contacto directo con memoria de pantalla es a través de los latches de la memoria.

Modo 0: al leer de memoria de pantalla, de lo único de que nos preocupamos es de que plano queremos recibir el dato, esto lo relacionamos en RMS.



Modo 1: en este modo lo que se hace es que la información que llega a la CPU es una comparación con algún color en especial que estamos buscando.

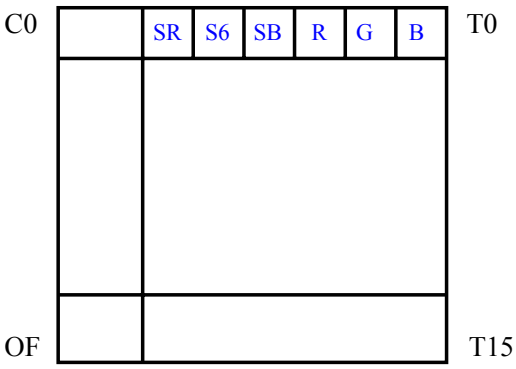


En cada columna tenemos ocho pixel consecutivos, ese se compara con el registro don´t care, con un and, en donde si hay un 0 no importa lo que hay en los planos, solamente se compara los planos que tenga un 1.

En color compare estará el color que queremos comparar, se hace con un and con el don`t care, para ver en que planos tenemos que buscar. Se compara con los comparadores y el resultado es lo que va a la CPU.

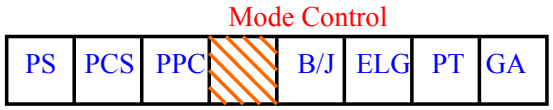
Controlador de Atributos.

Contiene los registros de paleta:



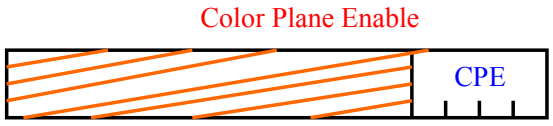
Cada bit es para un color primario y su intensidad.

y algunos registros más:



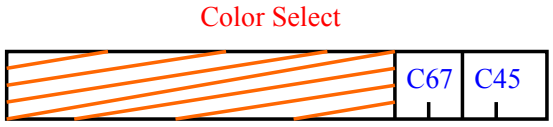
T16

T17 Color para los márgenes.



T18

T19



T20

Registro T16:

GA: si tiene un 1 selecciona el modo gráfico, si tiene un 0 el modo es alfanumérico.

DT: si tiene un 0 es monitor de color, si tiene un 1 es monocromo.

ELG: lo que hace es que los juego de caracteres tienen un ancho de 9 puntos de patrón de bits y la bios es de 8, para conseguir estos 9 lo que se hace es duplicar la ultima columna , esto si existe un 1 en esta posición.

B/J: Si tiene un 0 el bit 7 del atributo es color del fondo, si tiene un 1 el bit 7 es el bit de parpadeo. Solo se permiten ocho colores para el fondo.

PPC: si tiene un 0 impide una comparación de líneas, pero si tiene un 1 habilita la comparación de líneas.

PCS (pixel clocks select): selección de reloj de puntos. Si tiene un 0 el pixel cambia cada ciclo de valor de puntos, si tiene un 1 el reloj cambia cada dos ciclo de puntos.

A estos registros se accede por un puerto el 3C0, que es de solo escritura y este escribe en un bus interno que es el que accede a los 20 registros. A través del puerto 3C1 se lee. Ambos puertos también se puede leer y escribir en su registro índice donde se coloca que registro se acceden. Con el bit de PAS si tiene un 0 es que se accede al índice y si tiene un 1 es que se accede al de datos.

Registro 20:

C67 y C45: en la VGA el color se forma no en la paleta sino que se tiene 256 colores distintos y que están en los registros de color.

256 registros de 18 bits dividido en campos de 6 bits. Se accede con un byte y se desprecian los dos mayores.

Se accede de byte en byte y de forma secuencial hasta el final. Estos 16 bits pasan al DAC y de aquí se generan las señales de vídeo R, G, B, para cada color. Cada color puede tener 2E6 matices, es decir, al mismo tiempo en pantalla, pero disponemos de una gama de 2E16. Para seleccionar tenemos un decodificador para elegir una de estas 256 entradas por lo tanto necesitamos 8 bits para seleccionarlo. Los bits pueden provenir un mux o bien pueden provenir de la siguiente forma: los cuatro inferiores pueden venir del registro de paleta 3, 2, 1, 0, los bit 4 y 5 pueden venir de dos sitios, de la paleta o del C45, y los bits 6 y 7 de C67.

Con cambiar C67 podemos cambiar de paleta, si suponemos que las paletas están divididas en cuatro y además ponemos un uno en IPS tenemos para escoger entre 16 paletas distintas.

Registros Generales.

Son cuatro y cada uno tiene un puerto independiente de acceso.

- Registro salida miscelánea:

Se accede por los puertos:

3C2 para write.

3CC para read.

Dispone de los siguientes campos:

IOA: dirección de entrada / salida del CRT, si contiene un 0 (monocromo), se accede a través de 3B4 / 3B5h, si contiene un 1 (color), se accede a través de 3D4 / 3D5h.

ER: habilita la memoria de pantalla, para que la CPU pueda acceder a esta.

CS: es un campo de dos bits, con el siguiente significado:

0 0 → E/ Reloj de puntos es de 25,180 MHz
0 1 → E/ Fdot es de 28,325 MHz.
1 x → Reservado.

PB: bit de página para los modos impares, este bit afecta al significado del bit menos significativo de la dirección de memoria de pantalla. Si el bit 1 de G6 esta activo o el bit 3 de S4 esta activo, entonces este bit se ignora.

VSP, HSP: dan la polaridad del pulso de sincronismo horizontal o vertical. Si tiene un 0 es flanco de subida, en caso contrario sería un flanco de bajada.

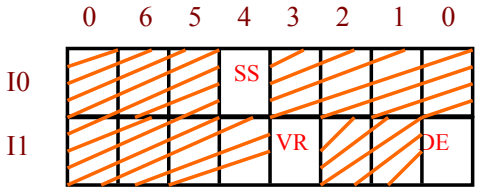
- Feature Control Register:

Este registro se escribe a través de los puertos 3BA para monocromo y 3DA para color. Para la lectura es a través del puerto 3CA

Afecta a la salida de sincronismo vertical, este pulso generado VSYNC pasa por un mux.

VSS: selección del sincronismo vertical.

- Input status 0 y 1.



DE (display enable not): si no está activa la visualización.

VR (vertical retrace): esta a 1 si se está en retorno vertical.

VR	DE	
0	0	Modo visualización
1	1	Retorno vertical.
0	1	Tiempo de retorno horizontal.
1	0	Nunca se da.

Para acceder a los registros del ATRIBC existe un biestable interno que según su contenido se dirige al índice o a los registros. Este biestable conmuta automáticamente cada vez que se escribe. Si se lee de Input Status #1 el biestable se pone a 0 (para acceder al registro índice).

El bit S del registro índice de ATRIBC (PAS) habilita / deshabilita el acceso desde la memoria de vídeo a la paleta. Si está a 0 la pantalla deshabilita el acceso, se deja de refrescar la pantalla.

Código de Salvapantallas.

```
WAIT_KEY macro

        mov ah, 8
        int 21h

mov dx, 3DAh
in al, dx          ; se lee I1 para poner biestable a 0
mov dx, 3C0h
sub al, al
out dx, al
WAIT_KEY
mov dx, 3DAh
in al, dx
mov dx, 3C0h
mov al, 20h
out dx, al
.....
```

El mejor momento para cambiar los registros es en el refresco vertical para evitar que el contenido de los registros desestabilize el contenido de pantalla.

```
        ; espera sincronismo horizontal
        mov dx, 3DAh

WaitNotVync:        ; espera a que termine un retorno vertical
        in al, dx
        and al, 08h
        jnz WaitNotVSync

WaitVSyc:    ; espera al retorno vertical

        in al, dx
        and al, 08h
        jz WaitVsync
```

Leer registros de color, hay que esperar al Vsync y después leerlo.

```
        mov di, SEG paleta          ; es: di   , tabla de 768 bytes (paleta) donde
        mov es, di                  ; se va a guardar la paleta.
        mov es, di
        mov di, 0FFSET paleta
        mov ch, 256
        xor ch, ah

@_1:

        mov dx, 3C7h
        mov al, ah
        cli     ; deshabilita instrucción
        out dx, al
        mov dx, 3C9h
        in al, dx
        stosb   ; mueve el byte de al a es : di
        in al, dx
        stosb
        in al, dx
        stosb
        sti
        inc ah
        loop @_1




. Data
        CRTCParam LABEL WORD
```

```
        DW      00D06h
        DW      03E07h
        DW      04109h
        DW      0EA10h
        DW      0AC11h
        DW      0DF12h
        DW      00014h
        DW      0E715h
        DW      00616h
        DW      0E317h

CRTCParam_ Size EQU  (( $ - CRTCParam) / 2)

SetMode proc
        mov ax, 13h
        int 10h         ; se le dice a la BIOS que ponga el modo 13.
        mov dx, 3C4h
        mov ax, 0604h
        out dx, ax
        mov ax, 0100h
        out dx, ax
        mov dx, 3C2h
        mov al, 0E3h  ; M = E3h, fdot = 25 MHz
        out dx, al
        mov dx, 3C4h
        mov ax, 0300h
        out dx, ax      ; S0 = 03h, activa SR
        mov dx, 3D4h
        mov al, 11h
        out dx, al
        inc dx
        in al, dx       ; al = R17
        and  al, 7Fh    ; se enmascara el bit más significativo
        out dx, al      ; se escribe R17
        dec dx
        cld
        mov si, offset CRTCParam
        mov cx, CRTCParamSize
@_2:
        lodsw
        out dx, ax
        loop @_2
        mov dx, 3C4h
        mov ax, 0F02h
        out dx, ax      ; S2 = 0Fh, habilita acceso a los cuatro planos

SetMode endp
```