UNIDAD 4 – Microprocesadores y microcontroladores

a - Procesadores: Arquitecturas. Operación a nivel registros.

Introducción:

Hemos estudiado en la Unidad 3 diversos módulos digitales combinacionales y secuenciales, entre ellos:

- Multiplexor digital, decodificador.
- Sumador/restador
- Buffer de tercer estado (TRI-STATE)
- Registros paralelo/paralelo, paralelo/serie, serie/paralelo.
- De mayor complejidad estudiamos las memorias ROM y RAM, esta última formada por decodificadores, registros y buffers TRI-STATE.

Con estos elementos es posible construir un **Sistema de Cómputo Programable**. Éste es un sistema digital capaz de procesar datos en función de listas de instrucciones previamente almacenadas, denominadas **programas**. En otros términos, es un sistema que ejecuta operaciones (aritméticas, lógicas, de lectura y escritura) con datos que ingresan y otros que están alojados en memoria RAM y ROM, y que entrega resultados. Estos resultados pueden ser visualizados (display, monitor) comunicados (modem, etc) o almacenados (discos, memorias).
Es decir podemos identificar tres tareas básicas:

1. **Recibir** datos a través de **Entradas**
2. Procesarlos de acuerdo con un programa y datos previamente cargados.
3. **Presentar** y/o almacenar resultados (datos procesados, órdenes) a través de **Salidas**

como se esquematiza en la **fig 4-1**.

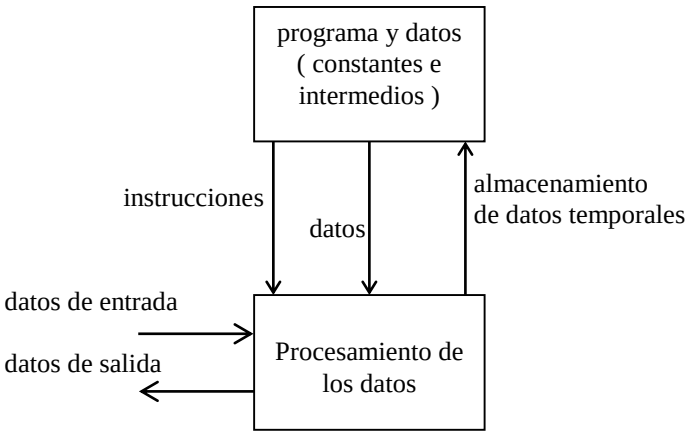


Figura 4-1: Esquema básico de un Sistema de Cómputo Programable

Los datos de entrada/salida pueden intercambiarse con un usuario mediante una interfaz adecuada (teclado, monitor, mouse, placa de sonido, etc), o con otro sistema electrónico mediante puertos de comunicaciones serie o paralelo (modem, impresora etc).

¿Cómo se puede implementar este sistema programable mediante circuitos digitales?

Existen básicamente dos arquitecturas de sistemas programables, la arquitectura **Von Neumann** y la arquitectura **Harvard**.
La Von Neumann es la más difundida por utilizarse en los microprocesadores (µP) de los PC, aunque la Harvard es muy utilizada en algunos microcontroladores y procesadores de señales digitales (DSPs).

Los sistemas que se describirán son genéricos, aunque orientados a lo que se aplica en PC.

Arquitectura Von Neumann

Está formado por 4 bloques funcionales: CPU, ROM, RAM y E/S.

Decimos que son funcionales porque la división en bloques mostrada no necesariamente es física sino funcional: por ejemplo hay chips microprocesadores que incluyen habitualmente - además de la CPU (Unidad Central de Procesamiento ó Unidad de Control y Procesamiento) - segmentos de RAM y ROM, mientras que los denominados microcontroladores (µC) también incluyen dispositivos de E/S tales como puertos serie y paralelo. Por otra parte un bloque de memoria RAM puede estar formado varios chips (banco de memoria).

Estos bloques funcionales están conectados por tres grupos de líneas de datos digitales, denominadas **Bus de Datos**, **Bus de Direcciones** y **Bus de Control**.

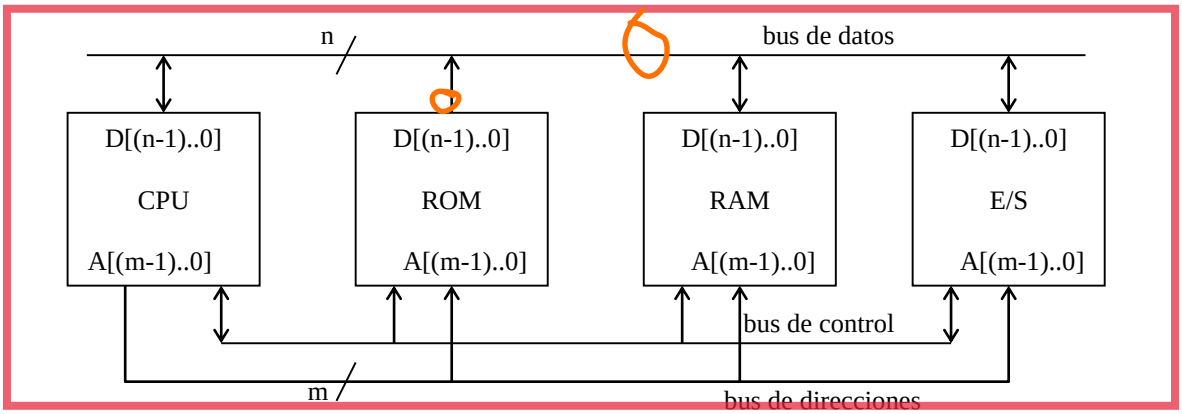


Figura 4-2: Arquitectura Von Neumann para realizar un Sistema de Cómputo Programable

Bus de Direcciones

Ancho de Bus

- El **bus de direcciones** es un conjunto de **m** conductores paralelos por el que la CPU presenta a los módulos RAM/ROM/E-S la dirección del operando a ser leído o escrito.

  Decimos que el **Ancho del Bus** es **m**. La cantidad de memoria que puede manejar se denomina **capacidad de direccionamiento** del sistema y es $\mathbf{2^m}$, es decir depende exponencialmente del ancho del bus.

  Este bus es **unidireccional**, tiene el sentido CPU→ RAM|ROM|E/S. Esto es así porque la CPU es la que normalmente gestiona el tráfico de datos. Hay excepciones en sistemas que utilizan DMA (Acceso Directo a Memoria), que se verá más adelante.

Bus de Datos

- El **bus de datos** es un conjunto homogéneo de **n** conductores por el que se transfieren datos e instrucciones, como se indica:
  RAM|ROM → CPU: las instrucciones de programas (ver nota sobre instrucción).
  RAM|E/S ←→CPU ó ROM →CPU: los datos propiamente dichos (operandos).

  Este bus es bidireccional, es decir los datos se leen y escriben por las mismas líneas. Para esto se utilizan compuertas TRI-STATE en antiparalelo que permiten que el dato circule en uno u otro sentido, según sea la señal R/-W (lectura/escritura) del bus de control,.

  Se dice que el ancho del bus es de n bits, con n igual a 8, 16, 32 etc. Hay que distinguir entre el ancho del bus interno del µP –es decir la cantidad de bits de sus registros internos- y el ancho del bus externo. Por ejemplo el intel 80386 tiene un bus de datos interno de 32 bits y un bus de datos externo de 16 bits (los datos de 32 bits se transfieren en dos etapas), mientras que todos los Pentium tienen un bus de datos interno también de 32 bits pero un bus de datos externo de 64 bits (puede transferir en una sola etapa 2 datos de 32 bits).

Bus de Control

- El **bus de control** es un conjunto *heterogéneo* de conductores, por eso sería más correcto hablar de *líneas de control*. Su función básica es coordinar la transferencia de datos. La señal más importante es la R/-W (lectura/escritura), que como se explicó controla el sentido de circulación del bus de datos. Otras señales son RESET (inicializa registros), IRQ (petición de interrupción desde un dispositivo de E/S), DRQ (petición de acceso directo a memoria desde un dispositivo de E/S) etc.

Formato de instrucciones

Un programa es un conjunto de instrucciones. Puede estar escrito en un Lenguaje de Alto Nivel, tal como BASIC, C, PASCAL, que tienen una sintaxis similar al lenguaje natural, pero para que pueda ser ejecutado por un microprocesador debe ser traducido a un Lenguaje de Bajo Nivel, denominado Lenguaje de Máquina.
Veremos como ejemplo una instrucción de asignación:

a - Una instrucción tal como

y = 4 + x

significa

- Sumar el valor 4 más el valor contenido en la variable **x**
- El resultado guardarlo en la variable **y**

se podría descomponer en las siguientes instrucciones que debería ejecutar un µP:

```
lee     [x]   ; guarda x en un registro interno la CPU (acumulador)
suma    4     ; le suma el valor constante 4
escribe [y]   ; el resultado lo escribe en [y]
...
```

Nota: Los corchetes indican que el valor se *lee de* o *escribe en* una posición de memoria (RAM|ROM o E/S) como el caso de [x] e [y]. En cambio el dato "4" es una *constante*.

En este ejemplo las instrucciones operan con **un** valor u operando (instrucciones *unarias*).
Los µP más potentes pueden realizar operaciones con **dos** o más operandos, por ejemplo

con 2 operandos podría ser

```
suma    [x]   4   ; guarda la suma en un registro de la CPU
escribe [y]       ; el resultado anterior lo escribe en [y]
```

con 3 operandos podría ser

```
suma    [x]   4   [y]   ; dir. dato |constante |dir. destino
```

Los programas para ix86 compatibles con todas las PCs trabajan en general con instrucciones *unarias*, mientras que los programas para procesadores de 32 bits pueden trabajar con instrucciones *binarias* y *ternarias*, lo que mejora el rendimiento. En adelante nos referiremos al proceso de instrucciones *unarias*.

b - Un programa tal como y=4 + x - y se podría descomponer en las siguientes instrucciones:

```
lee     [x]
suma    4
resta   [y]   ; valor previo de y
escribe [y]   ; nuevo valor de y
...
```

Obsérvese que se lee y escribe en la misma posición de memoria [y].

c - Un programa de un termostato:

"Si **Tentrada** < **Treferencia** → **Salida=1**, sino **Salida = 0**)"
con Treferencia *constante* podría realizarse con las siguientes instrucciones:

```
ciclo   lee                 [Tentrada]
        resta               Treferencia
        si resultado<0 ir a encender
```

```
                lee             0
                escribe         [Salida]        ; esto es apagar
                ir a            ciclo           ; vuelve a ciclo de control
encender        lee             1
                escribe         [Salida]
                ir a            ciclo           ; vuelve a ciclo de control
```

d - Un programa de termostato similar al anterior pero con Treferencia *variable* podría realizarse con las siguientes instrucciones:

```
ciclo           lee             [Tentrada]
                resta           [Treferencia]
                si resultado<0 ir a   encender
                lee             0
                escribe         [Salida]
                ir a            ciclo
encender        leer            1
                escribe         [Salida]
                ir a            ciclo
```

Vemos que en este caso tenemos el valor de referencia está en una variable (distinguir por los corchetes).

Las instrucciones básicas con las que debe contar un µP son:

- de transferencia: **leer|escribir** leer: **µP←RAM|ROM|E/S** escribir: **µP→RAM|E/S**
- suma aritmética
- complemento a '1', o inversión: cambia 'ceros' por 'unos' y viceversa. Necesaria para la resta
- operación lógica (AND u OR o ambas)
- de salto: **ir a**
- de salto condicional: **si** *condición* **ir a** (siendo *condición* resultado de una operación previa)

A estas se agregan otras sencillas como rotación a izquierda o derecha, XOR, manejo de subrutinas, manipulación de bit, manejo de *pilas de memoria*, hasta instrucciones muy complejas como las presentes en µPs con tecnología MMX.

Microprocesador elemental

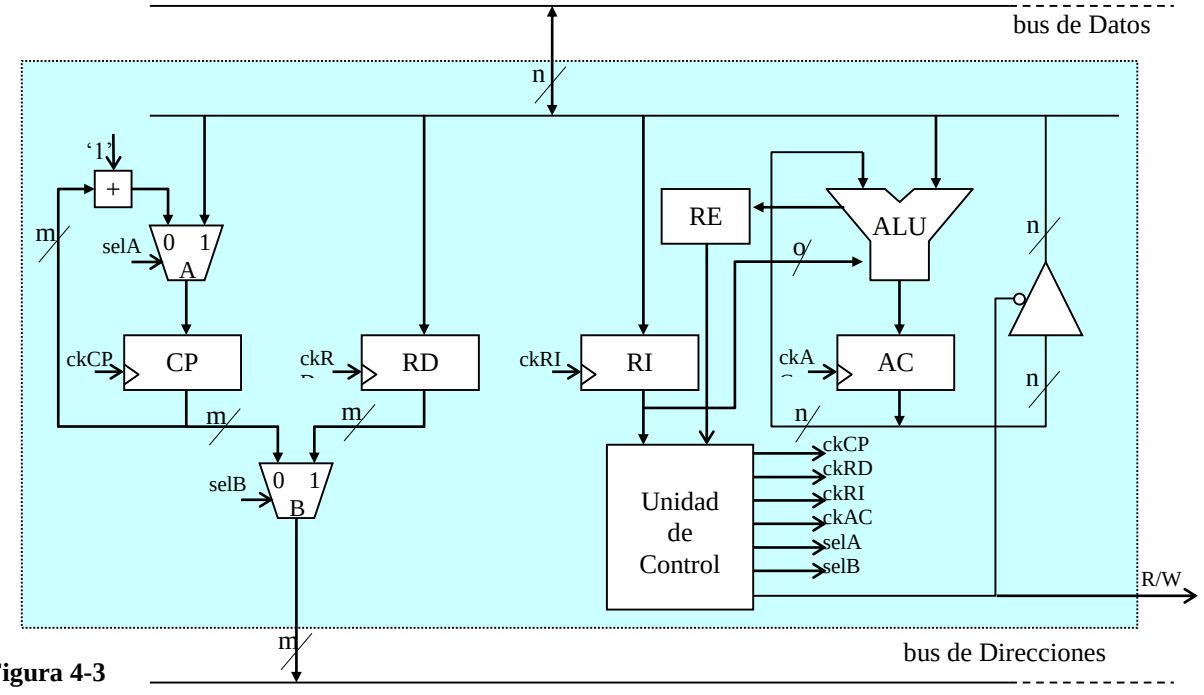


Figura 4-3

Para comprender los principios de funcionamiento utilizaremos un µP ficticio con una arquitectura Von Neumann básica y un repertorio elemental de instrucciones.
El microprocesador está constituido por registros tipo D (*Contador de Programa* CP, *Registro de Direcciones* RD, *Registro de Instrucciones* RI, *Registro Acumulador* AC), multiplexores que actúan como llaves selectoras (A,B), una Unidad Aritmético-Lógica(ALU), y un buffer TRI-STATE de n bits, todos manejados por un sistema combinacional/secuencial denominado Unidad de Control (UC). Hay además un *Registro de Estado* (RE) que cambia según el resultado de la ALU y un sumador de m bits que tiene como sumandos el valor de CP y un valor fijo '1', de modo tal que en su salida tiene el valor de CP incrementado en 1. Todo el "misterio" del funcionamiento del microprocesador está en el orden en el que la UC active las señales ckCP, ckRD, ckRI, ckAC, selA, selB y R/W,. La UC es básicamente un *generador de secuencias* que cambia según lo que reciba del registro de instrucciones RI y del registro de estado RE. La UC de nuestro microprocesador elemental realiza sólo cuatro tipos de secuencia: de lectura inmediata, de lectura direccionada, de escritura y de salto.

Repertorio de instrucciones (*instruction set*) de un microprocesador elemental

Vamos a definir para este µP un repertorio de instrucciones **ficticio**, **no debe procurar aprenderlo porque se trata de un ejercicio teórico**, codificaremos cada instrucción con un número hexadecimal y le colocaremos un mnemónico.

Código	Mnemónico	Nro de bytes	Sintaxis	Explicación
00	LEEC	2	LEEC k	Carga constante k en el acumulador AC. El segundo byte es k. **AC←k**
01	ANDC	2	ANDC k	AND bit a bit entre el contenido de AC y la constante k. El segundo byte es k. El resultado se carga en AC. **AC←AC and k**
02	ORC	2	ORC k	Realiza una OR bit a bit entre el contenido de AC y la constante k. El segundo byte es k. **AC←AC or k**
03	RESTAC	2	RESTAC k	Resta al contenido de AC la constante k. El segundo byte es k. **AC←AC – k**
04	SUMAC	2	SUMAC k	Suma aritmética entre el contenido de AC y la constante k. El segundo byte es k. **AC←AC + k**
05	SALTA	2	SALTA D	Salto incondicional del programa. D es la dirección donde continúa la ejecución del programa. **CP← D**
06	SALTAZ	2	SALTAZ D	Salto de programa si el resultado de la última operación de la ALU es cero (bit Z del RE activado). D es la dirección donde continúa la ejecución del programa. **CP← D si Z='1'**
07	SALTAN	2	SALTAN D	Salto de programa si el resultado de la última operación de la ALU es negativo (bit N del RE activado). D es la dirección donde continúa la ejecución del programa. **CP← D si N='1'**
08	LEE	2	LEE [N]	Carga variable N en el acumulador AC. El segundo byte es la dirección de N. **AC ←[N]**
09	AND	2	AND [N]	AND bit a bit entre el contenido de AC y la variable N. El segundo byte es la dirección de N. El resultado se carga en AC. **AC←AC and [N]**
0A	OR	2	OR [N]	OR bit a bit entre el contenido de AC y la variable N. El segundo byte es la dirección de N. El resultado se carga en AC. **AC←AC or [N]**
0B	RESTA	2	XOR [N]	Resta al contenido de AC la variable N.El segundo byte es la dirección de N. El resultado se carga en AC. **AC←AC - [N]**
0C	SUMA	2	SUMA	Suma aritmética entre el contenido de AC y la variable N. El segundo byte es la dirección de N. El resultado queda en AC. **AC←AC + [N]**
0D	NOT	1	NOT	Complementa a '1' el acumulador **AC←not AC**
0E	ESCRIBE	2	ESCRIBE [N]	Escribe en variable N el valor de AC. **[N] ←AC**
0F	NOP	1	NOP	No realiza operación aritmético-lógica. Sólo se incrementa CP

Tabla 4-4

Este conjunto de instrucciones , de 0 a F, podría ser codificado con sólo 4 bits, pero para simplificar el análisis adoptaremos un formato de 8 bits tanto para instrucciones como para direcciones y datos. Analicemos qué secuencias de activación de señales debe realizar la UC para ejecutar estas instrucciones. Para eso damos un ejemplo.

Operación de un sistema con RAM, ROM y E/S

Ejemplo de ejecución de un programa elemental

El µP está conectado a una memoria RAM|ROM con el programa correspondiente a la operación c=a+b en las direcciones 00 a 05 (zona de instrucciones), y las variables a, b c en las direcciones 81, 82, 83 (zona de datos).

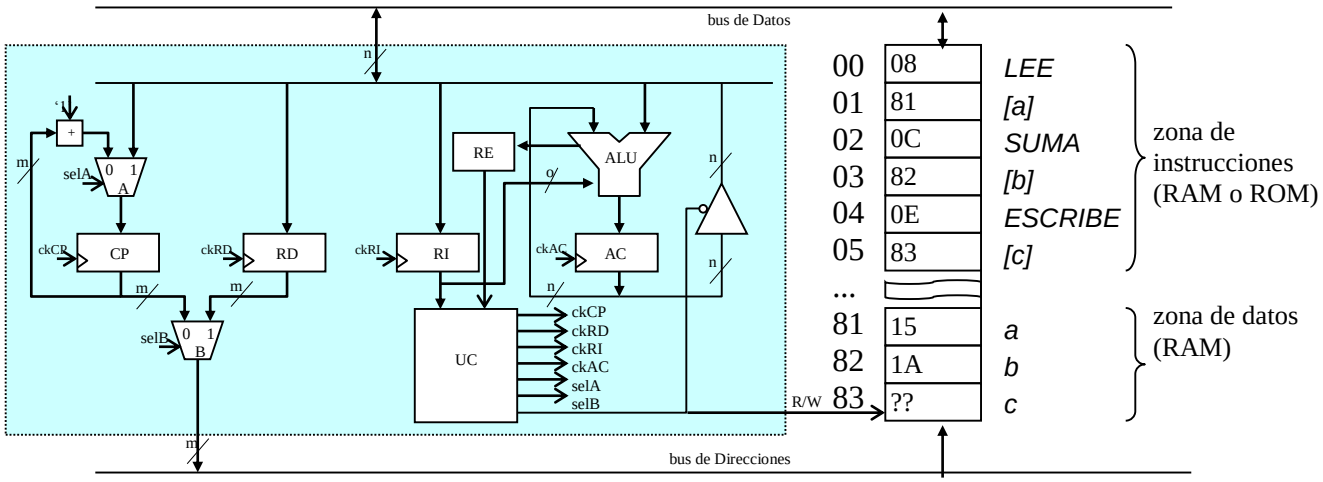


Figura 4-5

1. Al inicio (reset del µP) CP=00. selA=0, selB=0, R/W=1.
   Como selB=0 sale al bus de direcciones el valor de CP, que es 00. Como R/W=1 la memoria presenta el número 08 al bus de datos.
2. Este valor es el codigo de instrucción (*LEE*), debe ingresar al RI para ser analizado por la UC. Entonces la primera señal a activar es clkRI, para cargar el código en RI. Este código '08' se presenta tanto a la UC como a la ALU. Con el código '08' la ALU deja pasar sin modificar el dato presente en su rama derecha. Por otra parte la UC realizará la secuencia de *lectura direccionada*. Corresponde "buscar" la dirección del operando que debe leer.
3. Para esto activa clkCP, lo que provoca que CP se cargue con la suma CP+1, es decir pasa a valer 01 (se incrementa).
4. Sale al bus de direcciones el valor de CP, que es 01, Como R/W=1 la memoria presenta el número 81 al bus de datos.
5. Este valor es la dirección del operando **a**, y debe ser utilizado para acceder a dicho operando. Para eso la UC activa la señal clkRD (carga RD con el 81).
6. Cuando la UC pone selB='1' sale al bus de direcciones el valor de RD, que es 81. Como R/W=1 la memoria presenta el número '15' (valor de **a**) al bus de datos.
7. Este es el dato a cargar en el acumulador. Para eso la UC activa clkAC.
8. Una vez cargado el acumulador, debe avanzar a la próxima instrucción (en la posición 02 de memoria) , para lo cual debe volver a presentar CP al bus de direcciones (haciendo selB=0) e incrementar el valor de CP (activando clkCP).

En síntesis, para la instrucción LEE [**a**] la UC realiza la secuencia

clkRI, clkCP, clkRD, selB=1, clkAC, selB=0,clkCP

Obsérvese que para ejecutar la instrucción SUMA [**b**] la UC realiza la misma secuencia:

clkRI, clkCP, clkRD, selB=1, clkAC, selB=0,clkCP

(comprobar). La diferencia radica en que la ALU no estará en modo "transparente" sino en modo sumador. De esta forma, cuando la UC activa clkAC, el acumulador va a cargar el resultado de sumar su propio contenido ('15') con el dato que esté presentado en el bus de datos en ese momento, esto es el valor de **b** ('1A'). Es decir AC pasa a valer '2F' (15+1A=2F).

Para realizar la instrucción ESCRIBE [C] la UC realiza la siguiente secuencia

14. Sale al bus de direcciones el valor de CP, que es '04'. Como R/W=1 la memoria presenta el número '0E' al bus de datos.
15. Este valor es el codigo de instrucción (*ESCRIBE*) que debe ingresar al RI para ser analizado por la UC. Entonces se activa clkRI. Este código '0E' se presenta tanto a la UC como a la ALU, sin embargo ahora no importa en qué modo esté la ALU puesto que no se transferirá información a AC. La UC realizará una secuencia de *escritura*. Corresponde "buscar" la dirección del operando donde debe escribir.
16. Para esto activa clkCP, y CP pasa a valer '05'. Sale al bus de direcciones este el valor. Como R/W=1 la memoria presenta el número '83' (dirección de **c**) al bus de datos.
17. Este valor es la dirección del operando **c**, y debe ser utilizado para acceder a dicho operando. Para eso la UC activa la señal clkRD (carga RD con el '83').
18. Al poner selB='1' sale al bus de direcciones el valor de RD, que es '83'. Obsérvese que mientras R/W=1 se está leyendo la memoria y se tiene el valor x (valor *previo* de **c**) en el bus de datos.
19. **Ahora no se activa la señal clkAC, sino R/W**. Esto hace que al mismo tiempo que la RAM pasa a modo escritura (toma dato), el buffer TRI-STATE del µP tome el control del bus de datos, presentando así el valor del registro AC (0C). Es decir se transfiere el valor 0C a la posición 83 de RAM.
20. Antes de cambiar la dirección del registro accedido debe volverse la memoria a modo lectura, es decir R/W=1

Es decir, para la instrucción ESCRIBE [**c**] la UC realiza la secuencia

clkRI, clkCP, clkRD, selB=1, R/W=0, R/W=1, selB=0,clkCP

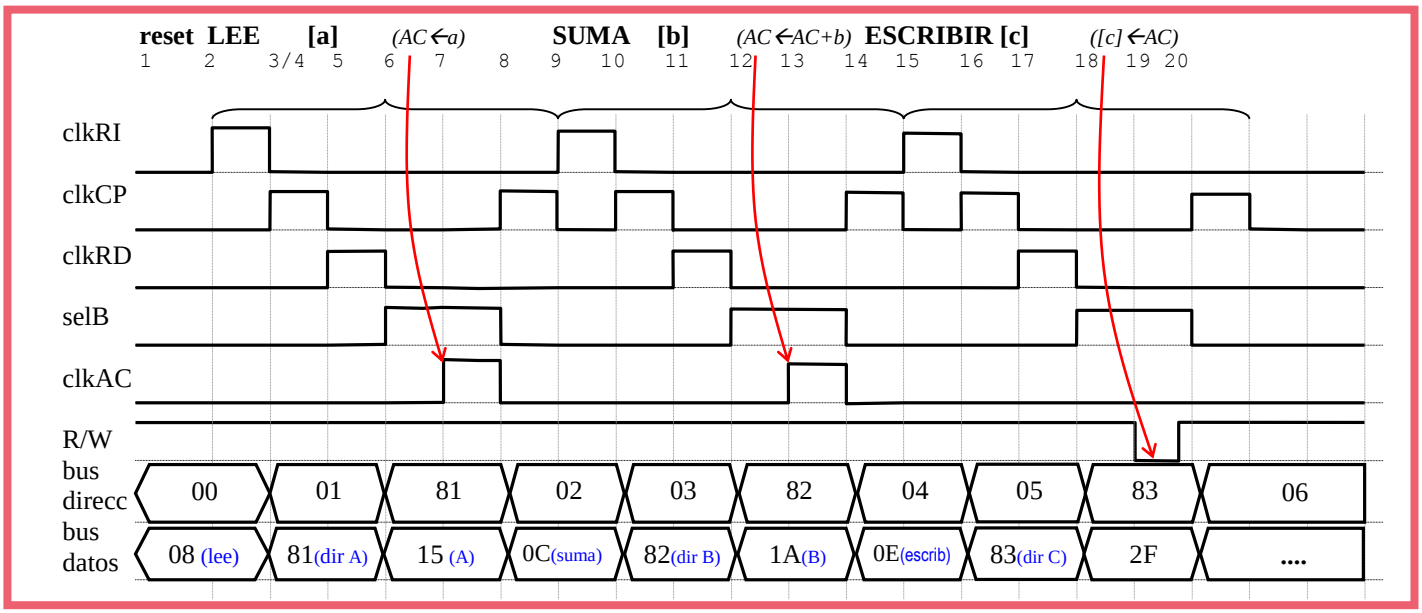


Figura 4-6

En el diagrama de tiempos de la figura 4-6 se aprecian las secuencias realizadas por la UC (suponemos flanco activo el flanco de subida), y los valores presentes en los buses de direcciones y datos.

Lectura de constante

Cuando se pasa a lenguaje de máquina una operación tal como: c=4+a, la codificación es

LEEC	4
SUMA	[a]
ESCRIBE	[c]

ó

```
LEE        [a]
SUMAC      4
ESCRIBE    [c]
```

Lo que sigue a los códigos de instrucción *LEEC* ó *SUMAC (o cualquiera con operando constante)* no es la dirección del operando sino el valor del operando. La secuencia de la UC será en estos casos mucho más sencilla

clkRI, clkCP, clkAC, clkCP

Saltos de programa

Dos de las cualidades más importantes de un sistema de cómputo son la capacidad de realizar un cálculo de manera reiterativa y la capacidad de tomar decisiones, esto es realizar acciones diferentes según las condiciones que se evalúen. Los ejemplos c y d (control de temperatura) muestran ambas capacidades, el controlador realiza de manera reiterativa la comparación entre un valor leído de E/S y un valor de referencia, y según sea el resultado de esta comparación realiza una acción diferente (encender o apagar). Reescribimos el ejemplo:

```
ciclo       lee                  [Tentrada]
            resta                Treferencia
            si resultado<0 ir a  encender
            lee                  0
            escribe              [Salida]      ; esto es apagar
            ir a                 ciclo
encender    lee                  1
            escribe              [Salida]
            ir a                 ciclo         ; vuelve a ciclo de
      control
```

Codificado según el *set* de instrucciones de la tabla 4-4 será (suponiendo que se inicia en la dirección 00)

Dirección de memoria	rótulo	instrucción
00	***ciclo***	LEE
01		[Tentrada]
02		RESTAC
03		Treferencia
04		SALTAN
05		***encender***
06		LEEC
06		0
07		ESCRIBE
08		[Salida]
09		SALTA
0A		***ciclo***
0B	***encender***	LEEC
0C		1
0D		ESCRIBE
0E		[Salida]
0F		SALTA
10		***ciclo***

Saltar significa que el programa deja su curso normal (a través de direcciones consecutivas de memoria, apuntadas por el contador de programa CP) para pasar a otra dirección anterior o posterior. En el ejemplo tenemos dos puntos del programa a los que se salta: ***ciclo*** (dirección 00) y ***encender (***dirección 0B). Esto se consigue simplemente cargando en el CP un valor correspondiente a la dirección a donde se quiere saltar. Obsérvese que en una instrucción tal como SALTA ***ciclo*** el "operando" que sigue al código de instrucción SALTA es la dirección a la que debe saltarse, esto es el valor que debe tomar el CP. Para cargar este valor (que viene por el bus de datos), se coloca selA=1 (lo que conecta el bus de datos a la entrada de CP) y se activa clkCP,

con lo cual en vez de incrementarse CP se transferirá el valor correspondiente a la dirección destino. A partir de allí, volviendo selA=0, el programa continúa su normal ejecución. La secuencia de salto es entonces la siguiente:

clkRI, clkCP, selA=1, clkCP, selA=0...

Esta secuencia es la misma para salto incondicional que para salto condicional. El salto condicional se produce según la instrucción utilizada (SALTAN, SALTAZ) y el estado de los correspondientes bits de Cero y Negativo (Z y N) del registro de estado RE , que acusan si la última operación realizada en la ALU dio resultado negativo o cero respectivamente.

Nota: En el ejemplo del termostato hemos tratado los valores de *entrada* y *salida* como variables en memoria, hablamos de dispositivos de E/S *mapeados en memoria*. Sin embargo los microprocesadores Intel x86 tratan de manera distinta los registros de RAM de los registros de dispositivos de E/S, contando con instrucciones distintas. A las primeras se las codifica como MOV, a las segundas como INP (leer Entrada) y OUT (escribir Salida). (Luego trataremos los dispositivos de E/S).

Llamado a subrutina

A lo largo de un programa se suele realizar de forma repetida un mismo conjunto de instrucciones (llamado *segmento de código*), por ejemplo cada vez que se realiza una operación de división. Para ahorrar memoria, en vez de repetir el mismo segmento de código éste se escribe una sola vez y se *salta* a él cada vez que se requiera. A tal segmento se lo denomina *subrutina*. Para poder volver luego de ser invocado desde distintos puntos del programa existe un par de nuevas instrucciones: LLAMA (call) y RETORNA (return).
Tomemos como ejemplo una subrutina de división. Aquí la subrutina utiliza los valores *dividendo* y *divisor* que han sido establecidos en el programa principal justo antes de invocarla. Se dice que estos valores son los *argumentos* que se le pasan a la subrutina de división (los µPs suelen contar con registros auxiliares donde se escriben estos argumentos en vez de utilizar la RAM).
Al terminar de ejecutarse debe retornarse a la instrucción siguiente a la instrucción desde donde fue llamado. Para que esto sea posible, antes de saltar a la subrutina deberá resguardarse el valor del contador de programa CP, pues de otro modo no sería posible retornar al punto de programa desde donde fue invocada la subrutina. Para esto el µP deberá tener un hardware más complejo que el visto, como contar con un registro auxiliar que guarde el valor de CP antes del salto, y luego se recargue en el CP.

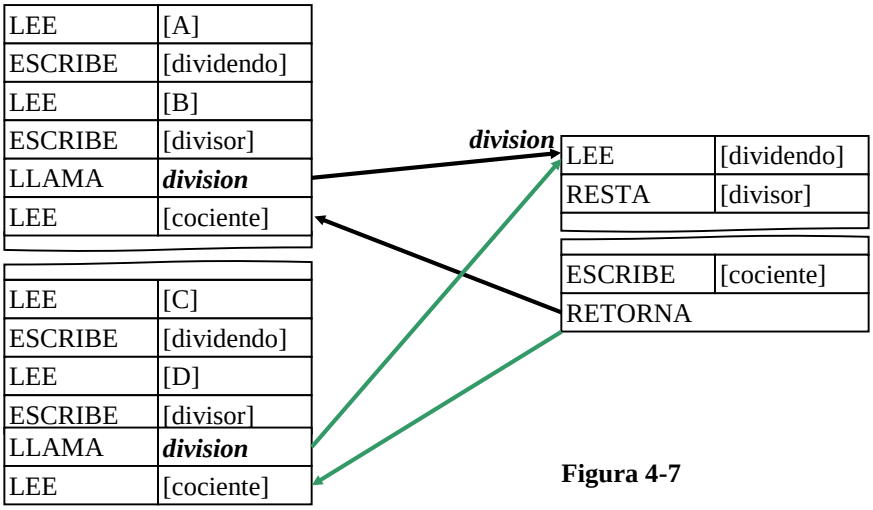


Figura 4-7

Una subrutina compleja puede a su vez requerir otras subrutinas, dando lugar a un esquema de *subrutinas anidadas*, es decir subrutinas dentro de subrutinas Para hacer posible este esquema con múltiples niveles de subrutina se reserva un área de la memoria RAM donde se van resguardando los valores de CP *apilándolos* a medida que se adentra un nivel, de modo tal que cuando se retorna de subrutina se vuelve al último valor guardado de CP y éste se quita del tope de la *pila*. En realidad los µP cuentan con un registro auxiliar llamado *puntero de pila (SP stack pointer*) que se incrementa (o decrementa) cada vez que se entra un nivel de subrutina, y se

decrementa (o incrementa) cuando se sube un nivel. El conocido mensaje de error "*stack overflow*" o "desborde de pila", se debe a que por un fallo de software (por ejemplo la falta de una instrucción *return*) se excede el área de memoria reservada para la pila.
En algunos µPs (por ejemplo en la familia x86) existe la posibilidad de guardar en la pila no sólo el valor de CP sino el de los otros registros, (como el acumulador y demás registros auxiliares), mediante una instrucción específica (por ejemplo "*push AX"*: resguarda el registro acumulador AX en pila) de modo tal que al retornar se recupera ("*pop AX*": recupera valor del tope de la pila y lo guarda en el registro acumulador AX) . Este proceso permite preservar el estado del µP (esto es el valor de todos sus registros) previo a la ejecución de la subrutina. Para no tener que escribir este proceso en cada llamado a subrutina, el resguardo conviene hacerlo directamente dentro de la misma subrutina. Las instrucciones ***push*** irán al comienzo de la subrutina y las instrucciones ***pop*** al final, **en el orden inverso** y justo antes de la instrucción de retorno. (En el orden inverso porque si he guardado los valores de registros A, B, C debo rescatar luego C, B A.)
Resumen:

- Un sistema de cómputo programable está constituido por una unidad de Procesamiento, una unidad de Memoria y una interfaz de E/S.
- El modo en que se implementa este sistema da lugar a las arquitecturas Harvard o Von Neumann. Esta última es la que se ha analizado porque es la más utilizada.
- Un sistema con arquitectura Von Neumann tiene una única memoria para instrucciones y datos, una unidad de procesamiento y una unidad de E/S, conectados por 3 buses: de datos, de direcciones y de control. El bus de datos transfiere tanto instrucciones de programa como datos.
- Un µP es básicamente un conjunto de registros, dispositivos de selección, una ALU y una Unidad de Control UC, que es un circuito generador de secuencias, que se denomina también Máquina de Estados.
- La UC de un µP elemental con un reducido repertorio de instrucciones –16 en nuestro ejemplo- debe realizar para cada una de ellas alguno de estos cuatro tipos de secuencia: de lectura inmediata (LEEC ANDC ..) , de lectura direccionada (LEE AND SUMA ...), de escritura o de salto.
- Una instrucción típica está compuesta por dos campos, un campo código de operación (qué hay que hacer) y un campo operando (dato para ejecutar dicha operación). El campo operando puede ser el valor con el cual operar (instrucciones de lectura inmediata), o la dirección de memoria de datos donde se encuentra el valor a operar (instrucciones de lectura direccionada o de escritura), o la dirección de memoria donde debe continuar el programa (instrucciones de salto).
- El uso de *llamado a subrutinas (y el correspondiente retorno)* permite escribir programas más cortos, escribiendo una sola vez las rutinas más utilizadas. Para el retorno se requiere resguardar el contador de programa. En un esquema de subrutinas anidadas es necesario contar con una pila y un puntero de pila.

4.b Interfaces de Entrada / Salida

Analizaremos cómo se organiza la interfaz de E/S para que el µP pueda intercambiar datos con los periféricos (monitor, mouse, teclado, impresora, modem, sonido, webcam) y unidades de almacenamiento (disco rígido, CDROM, disquetera). Cada uno de estos dispositivos tiene sus propios requerimientos, que son atendidos por un hardware de interfaz, por ejemplo:

Disco rígido, CDROM:	Controladora (IDE, SCSI, etc)
Disquetera:	Controladora de floppy
Monitor:	Controladora de video
Impresora:	Puerto paralelo o puerto USB
Modem, mouse:	Puerto serie o puerto USB
Webcam	Puerto USB
Sonido:	Placa multimedia

Existen otros dispositivos menos habituales, como filmadoras, placas de adquisición de datos, que pueden ser conectados a hardware de interfaz estándar o contar con su propio hardware de interfaz.
Cada interfaz cuenta con registros que permiten interactuar con el periférico, esto es:
intercambiar información: registros de Datos, de E/S

configurarlo: registros de Control
verificar su estado: registros de Estado

Estos registros pueden ser leídos o escritos a través de líneas de dirección, dato y control de manera similar a los registros de una RAM. Por otra parte los registros tendrán conexión a circuitos específicos de interacción con el periférico.

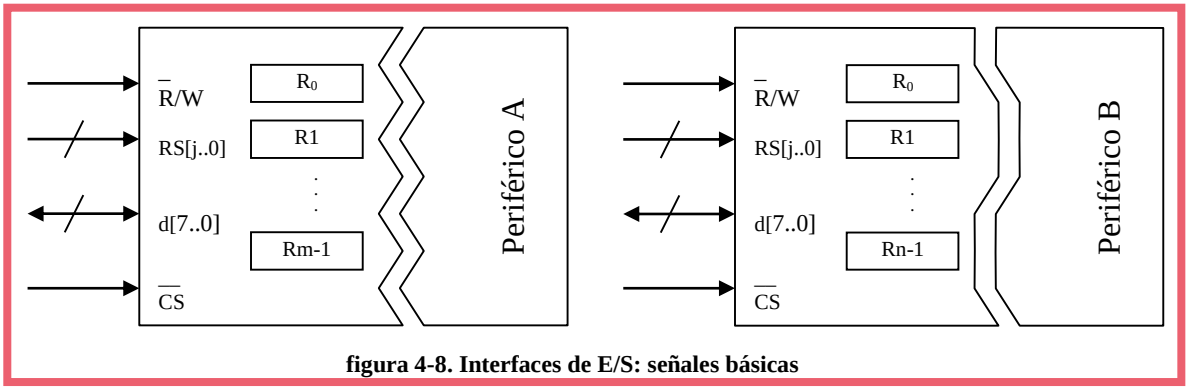


figura 4-8. Interfaces de E/S: señales básicas

En la **figura 4-8** se representan interfaces con sus señales básicas (costado izquierdo), y su vinculación con el periférico (a la derecha). Esta vinculación será mediante un conjunto de señales con un formato específico para el periférico (serie, paralelo, analógico, digital etc).

Obsérvese que las señales del costado izquierdo son similares a las de una memoria RAM. Los registros R_0 a Rm-1 son direccionados mediante las líneas RS[j..0] (RS:*register select*), y escritos o leídos a través de d[7..0] (hemos supuesto un bus de datos de 8 bits) según sea la señal R/W. La señal CS (*chip select*) habilita estos intercambios.

La característica que impuso a la PC de IBM fue su arquitectura abierta, que permitió incorporar a su sistema de E/S dispositivos de otros fabricantes (placas de video, controladoras de disco, placas de adquisición de datos etc). IBM reserva direcciones específicas para los dispositivos de E/S estándar como puertos paralelos de impresora (LPT1, LPT2, LPT3), puertos de comunicaciones serie (COM1, COM2...), controladores de interrupciones. Todos los periféricos están mapeados en el rango de direcciones desde la 000h a la FFFFh (en decimal 0 a 65535). El bus de direcciones para E/S es entonces de 16 bits (a[15..0]).

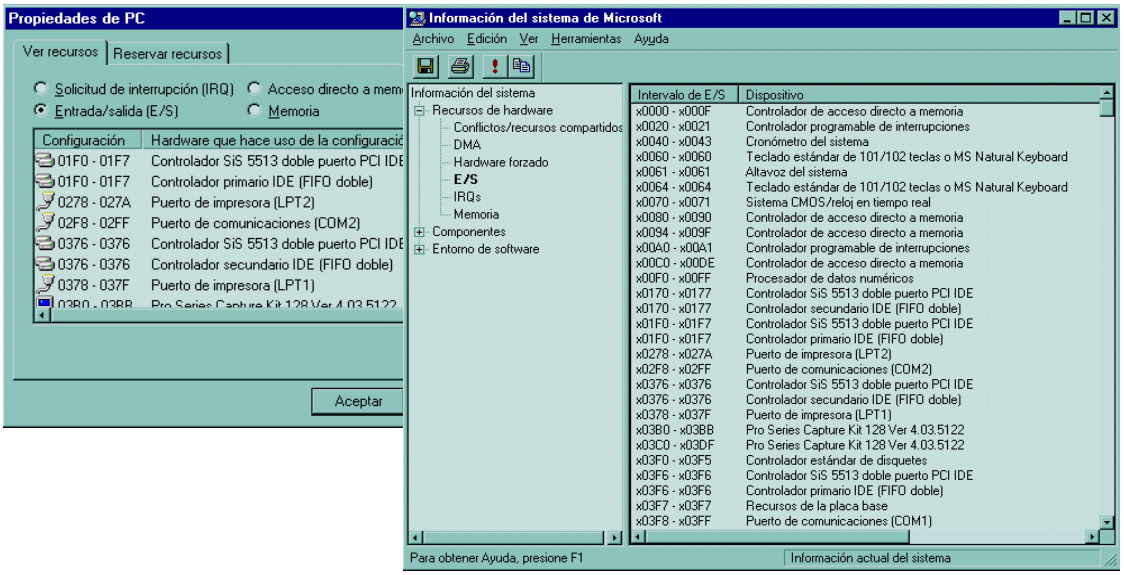


figura 4-9. Reportes de E/S en Windows

En la **figura 4-9** se muestran reportes típicos del sistema desde Windows donde se aprecia que cada dispositivo tiene reservado un intervalo de E/S. (En las PCs actuales, especialmente en la notebooks, muchos de estos dispositivos han caído en desuso).

Por ejemplo el altavoz del sistema tiene reservada la dirección 061h, el puerto paralelo LPT1 tiene reservado el intervalo 378h-37Fh, el puerto paralelo LPT2 el intervalo 278h-27Ah etc.
¿Cómo hacer para que cada dispositivo con una estructura similar a la de la figura 4-8 responda al intervalo que se le ha asignado?
Se puede utilizar una estrategia que ilustraremos con el puerto paralelo LPT2 (intervalo 278h-27Ah). En este esquema hemos omitido las líneas de IRQ que se verán después.

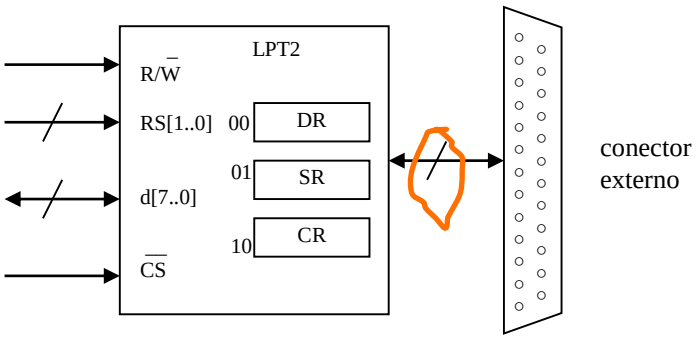


figura 4-10. Puerto paralelo, señales básicas

- Del bus de direcciones asignado a dispositivos de E/S (a[15..0]) se toman las líneas necesarias para direccionar todos los registros de la interfaz. La interfaz paralelo estándar tiene tres registros (Datos, Estado y Control) que se direccionan con las líneas RS1 RS0 (combinaciones 00, 01 y 10, la 11 no se utiliza), entonces conectamos a[1]→RS1, a[0] →RS0
- El resto de las líneas a[9..2] se utilizan para producir la habilitación del chip (que se da con -CS=0) cuando coincidan con el intervalo asignado. En nuestro caso el intervalo es 278h-27Ah. (el 27Bh no se utiliza)

		a[9..0] (bus de direcciones)			
		a[15..8]	a[7..4]	a[3..0]	registro
otro	277h	00000010	0111	**01** 11	
LPT2	278h	00000010	0111	10 00	DR
	279h	00000010	0111	10 01	SR
	27Ah	00000010	0111	10 10	CR
	27Bh	00000010	0111	10 11	--
otro	27Ch	00000010	0111	**11** 00	

Coincidencia con la dirección asignada, -CS=0

a[9..2] al comparador binario a[1..0] a RS1-RS0

Nota: Como ya se mencionó, los microprocesadores Intel x86 utilizan instrucciones distintas para acceder a los registros de RAM (instrucción MOV) y a los registros de dispositivos de E/S (IN y OUT). Una línea del bus de control, AEN (del µP a los dispositivos de E/S) permite diferenciar el acceso a E/S del acceso a RAM/ROM. Esta señal se combina con la salida del comparador de direcciones para controlar el CS del dispositivo y así evitar conflictos con las memorias, ya que en la PC también se utilizan las direcciones de RAM/ROM de 000 a 3FFh.

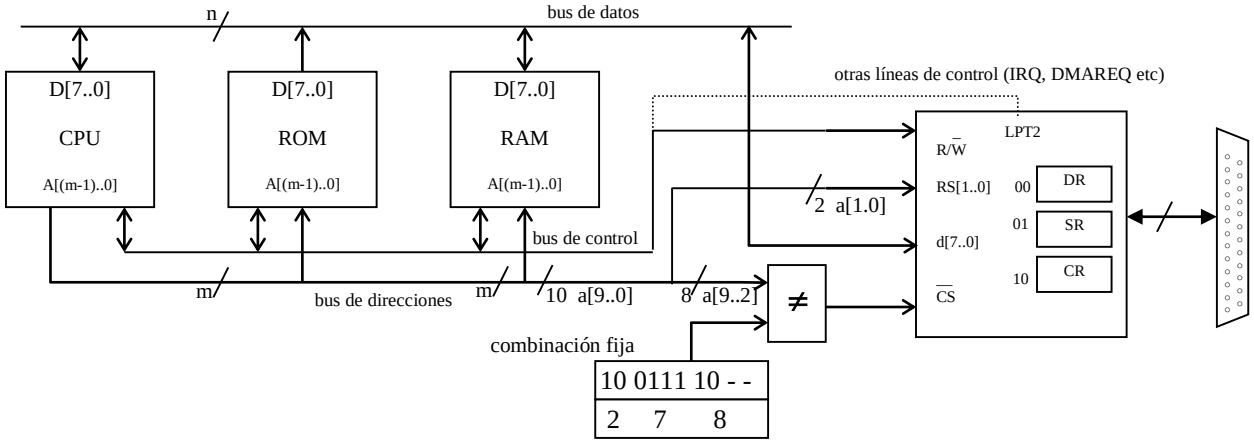


Figura 4-11. Modo de integración de una interfaz de E/S al sistema de buses de la PC

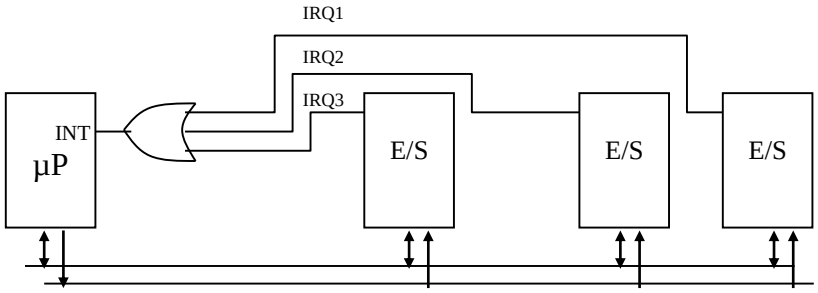
4.c Interrupciones

Existen tres tipos de interrupciones, considerando su origen: Interrupciones de hardware, interrupciones de software e interrupciones internas (éstas últimas llamadas también excepciones).

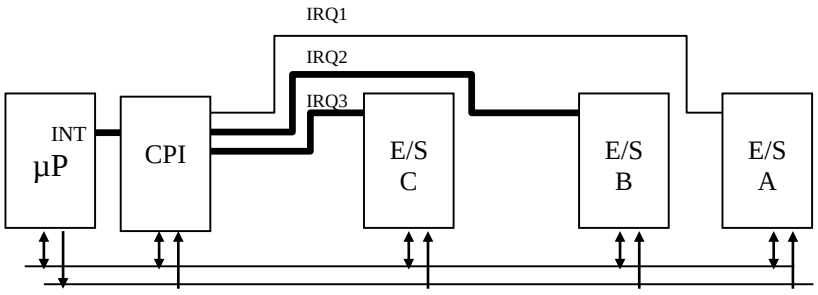
Interrupciones de hardware

Un µP que interactúa con un usuario o sistema externo debe responder a las demandas que éste realice a través de la interfaz de E/S. Una forma es mediante el *polling*, esto es consultar en forma periódica cada dispositivo de E/S (teclado, mouse, modem etc) para verificar si requiere atención (ej. lectura de datos para procesar o escritura de datos). La necesidad de atención de un dispositivo se indicará en un bit en un registro de estado de dicho dispositivo.
El mecanismo de *polling* es poco eficiente pues si el µP consulta muy frecuentemente los dispositivos de E/S, pierde tiempo que podría aprovechar en procesamiento, y si por el contrario los consulta muy espaciadamente puede tardar demasiado en atender un requerimiento externo. Un mecanismo más eficiente es el de *interrupción*; el µP se dedica al cómputo y sólo atiende al dispositivo externo cuando éste lo solicita mediante una señal denominada *interrupción externa*.
Los µP más sencillos cuentan habitualmente con una sola línea de interrupción externa, que es activada cuando cualquiera de los dispositivos requiere atención. Esto se puede observar en la **figura 4-12a** resuelto con una compuerta OR. Para saber cuál fue el dispositivo solicitante el µP debe consultar a cada uno de ellos, es decir en el peor caso se deberá cumplir un ciclo de consulta completo cada vez que alguno de los dispositivos requiere atención, y un criterio de optimización podría ser consultar en primer lugar los dispositivos que más frecuentemente interrumpen, otro criterio puede ser consultar primero los de mayor prioridad.

(a)



(b)



Si se activan IRQ2 e IRQ3, se activa INT. El µP consulta al CPI y éste le informa "IRQ2", que tiene mayor prioridad que IRQ3. Una vez atendido "B", la IRQ2 se desactiva y se "ve" la IRQ3

Figura 4-12

Para agilizar la identificación de la fuente de interrupción y administrar las prioridades de atención existen los denominados *controladores programables de interrupciones (CPI ó PIC, no confundir con los microcontroladores PIC)*, que se ha utilizado en el esquema de la figura 4-12b.
El CPI es en sí mismo un dispositivo de E/S basado en el esquema de la figura 4-13. Cuenta con un *codificador con prioridad* que codifica la fuente de interrupción. El µP, al ser interrumpido, consulta directamente al CPI, el cual entrega dicho código.
Una vez identificado el dispositivo que solicita atención, debe ejecutarse lo que se denomina la *rutina de servicio*, por ejemplo si se trata de un puerto serie la rutina de servicio deberá leer el dato recibido y almacenarlo en un lugar de memoria. El mecanismo para "saltar a" y "volver de" estas rutinas de servicio es similar al visto para subrutinas (se hace uso de la pila etc), pero en este caso el salto es provocado por una señal de interrupción externa y no por una "llamada a subrutina".

Prioridad de interrupciones – máscara de interrupciones.

Los dispositivos externos podrán requerir atención con mayor o menor urgencia según sus características, por lo que habrá que administrar prioridades. Supongamos dos dispositivos externos A y B, siendo A un dispositivo con mayor prioridad de atención que B. Si A y B solicitan atención al mismo tiempo deberá atenderse primero al de mayor prioridad, esto es A. Si mientras se está atendiendo a A, B solicita atención, se lo ignorará. Si en cambio se está atendiendo a B y A solicita atención, en general se suspende el servicio de B para atender a A. Si el servicio de B una vez iniciado no puede suspenderse, deberá recurrirse a inhibir las demás interrupciones hasta terminar la rutina de servicio de B. La inhibición se realiza a través de una instrucción genérica o de manera selectiva mediante un registro de máscara de interrupciones. Esta lógica se ilustra en la figura 4-13

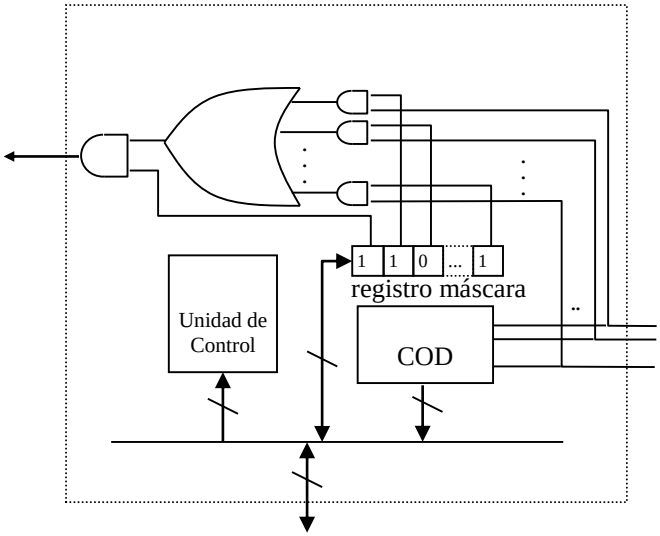


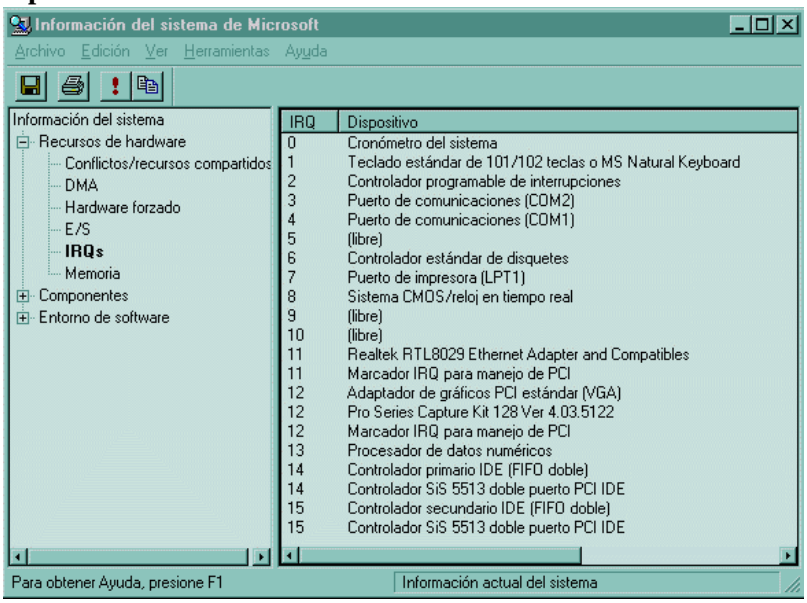
figura 4-13. Controlador de interrupciones

Los micropcocesadores de las PC cuentan además con una entrada de interrupción no enmascarable (NMI), para errores graves (paridad de memoria o error de canal de E/S).
En la PC el sistema de interrupciones de hardware tiene 16 niveles, conseguido con 2 controladores de interrupciones tipo 8259A más la entrada NMI.. El sistema posibilita el enmascaramiento (inhibición) de todas las posibles fuentes deinterrupción, incluyendo la NMI también.
Los niveles de interrupción en orden descendente de prioridad son:

Prioridad		**Función**
NMI – Interrupción no enmascarable del µP		paridad, error de canal de E/S
IRQ0		canal 0 del temporizador
IRQ1		teclado
IRQ2	IRQ8	interrupción del reloj de tiempo real
	IRQ9	redirección de software a la INT0Ah
	IRQ10, IRQ11, IRQ12	reservado
	IRQ13	coprocesador matemático
	IRQ14	controlador de disco duro
	IRQ15	reservado
IRQ3		puertos serie 1 y 3 (COM2-COM4)
IRQ4		puertos serie 2 y 4 (COM1-COM3)
IRQ5		puerto paralelo 2 (LPT2)
IRQ6		controladora de disquete
IRQ7		puerto paralelo 1 (LPT1)

En la IRQ2 del primer 8259 se conecta la salida del segundo 8259, que toma las IRQs 8 a 15.

Interrupciones por software:



Las denominadas *interrupciones por software*, disponibles por ejemplo en los µP Intel x86 (en las PCs) son equivalentes a *llamadas a subrutina*, pero que hacen uso de rutinas estándar del BIOS y del Sistema Operativo del computador en cuestión. Los µP de la familia Intel x86 disponen de una instrucción específica **INT** (INT *n*) donde *n* es el número de interrupción. Esto equivale a llamar a una rutina de servicio estándar *n*, por ejemplo de lectura/escritura en disco, de impresión en pantalla etc.

Interrupciones internas o excepciones: Las genera automáticamente la CPU ante una situación anormal o usos especiales.
INT 0: error de división, cuando el cociente no cabe en el registro o el divisor es cero. Se puede producir ante una instrucción DIV o IDIV.
INT 1: paso a paso, se produce tras cada instrucción cuando el procesador está en modo traza (utilizada en depuración de programas).
INT 2: interrupción no enmascarable (*NMI*)
INT 3: utilizada para poner puntos de ruptura (*breakpoints*) en la depuración de programas.
INT 4: provocada intencionalmente mediante una instrucción INTO, si existía desbordamiento.
INT 5: rango excedido en la instrucción BOUND.
INT 6: código de operación inválido. Se produce al ejecutar una instrucción indefinida, en la pila se almacena la dirección (CS:IP) de la instrucción ilegal.
INT 7: dispositivo no disponible.

Concepto de interrupción vectorizada:

Cada interrupción tiene su propia rutina de servicio, cuyo comienzo estará en una dirección RAM/ROM. En la PC dichas direcciones están escritas en una tabla o *vector* (000-3FFh de RAM). Según qué interrupción se produzca será el punto de entrada (INT) a la tabla (número entre 00 y FF). La dirección de comienzo de la rutina de servicio estará escrita en 4 bytes consecutivos. (Ej. para la interrupción de teclado el punto de entrada es 09, la dirección de la rutina de servicio se encuentra en los bytes 24h a 27h.

INT	Dirección	Función
00	00 - 03	división por cero
01	04 - 07	paso a paso
02	08 - 0B	NMI
03	0C - 0F	punto de ruptura
04	10 - 13	desbordamiento
05	14 - 17	PRINT SCREEN
06	18 - 1B	reservado
07	1C - 1F	reservado
08	20 - 23	tiempo
09	24 - 27	teclado
0A	28 - 2B	reservado
0B	2C - 2F	comunicaciones
0C	30 - 33	comunicaciones

0D	34 - 37	impresor secundario
0E	38 - 3B	floppy
0F	3C - 3F	impresor
10	40 - 43	video
11	44 - 47	chequeo de la configuración
12	48 - 4B	memoria
13	4C - 4F	floppy/disco duro
14	50 - 53	comunicaciones
15	54 - 57	
16	58 - 5B	teclado
17	5C - 5F	impresor
18	60 - 63	
19	64 - 67	carga del sistema
1ª	68 - 6B	hora
1B	6C - 6F	tecla BREAK
1C	70 – 73	tick del temporizador
1D	74 – 77	inicialización de video
1E	78 - 7B	parámetros del floppy
1F	7C - 7F	caracteres gráficos de video
20	80 – 83	terminación de programas (DOS)
21	84 - 87	llamado a las funciones (DOS)
22	88 - 8B	terminación de dirección (DOS)
23	8C - 8F	dirección de salida del CTRL BREAK (DOS)
24	90 – 93	vector de error fatal (DOS)
25	94 - 97	lectura absoluta del disco (DOS)
26	98 - 9B	escritura absoluta del disco (DOS)
27	9C - 9F	terminación (DOS)
28 - 3F	A0 - FF	reservado para el DOS
40 - 5F	100 – 17F	reservado
60 -67	180 – 19F	reservado para los usuarios
68 - 6F	1A0 - 1BF	
70	1C0 - 1C3	IRQ 8 (interrupción del reloj)
71	1C4 - 1C7	IRQ 9
72	1C8 - 1CB	IRQ 10
73	1CC - 1CF	IRQ 11
74	1D0 - 1D3	IRQ 12
75	1D4 - 1D7	IRQ 13 (dirección del NMI)
76	1D8 - 1DB	IRQ 14
77	1DC - 1DF	IRQ 15
78 - 7F	1E0 - 1FF	no se utiliza
80 - 85	200 – 217	
86 - F0	218 - 3C3	
F1 - FF	3C4 - 3FF	

Transferencia masiva de datos mediante Acceso Directo a Memoria (DMA)

La forma habitual de transferir un dato de un dispositivo de E/S a la memoria, es que la CPU lea el dispositivo de E/S y escriba el valor leído en la posición de memoria. Sin embargo cuando deben transferirse grandes bloques de información (por ejemplo cargar en memoria un archivo que está un disquete) se utiliza la técnica de DMA que libera a la CPU de esa tarea. El control del DMA lo realiza un procesador específico (en la PC el 8237) al que se le indica el rango de direcciones de origen y destino para realizar la transferencia, tarea que ejecuta sin intervención de la CPU, e incluso aprovechando las latencias dentro de los ciclos de la CPU (intervalos en los que la CPU no controla los buses. Esta técnica se denomina "robo de ciclo"). En el sistema pueden utilizarse 7 canales de DMA, que pueden ser utilizados por los distintos dispositivos de E/S para transferir bloques de información a la RAM (disquetera, puertos paralelos, placas de adquisición de datos etc). Incluso se pueden hacer transferencias de memoria a memoria.

Resumen

- El µP se comunica y controla los dispositivos periféricos a través de interfaces de E/S, que cuentan con registros de Datos, Estado y Control específicos. Mediante un circuito externo (comparador de direcciones etc) los registros quedan ordenados de manera similar a los registros de una RAM, ocupando cada periférico un rango de E/S dentro del denominado Mapa de E/S.
- Para liberar al µP de la tarea de consulta permanente los perifericos se utilizan las interrupciones de hardware, con un dispositivo de E/S especial llamado Controlador de Interrupciones Programable. En la PC hay dos PIC conectados en cascada para atender un total de 15 IRQs (interrupt requests).

- Las interrupciones de software son invocadas por programa, para acceder a las rutinas de servicio estándar del Sistema Operativo y del BIOS.
- La interrupciones internas o excepciones son provocadas por errores y condiciones anormales.
- Los tres tipos de interrupciones referencian a sus rutinas de servicio a través de una tabla, en un esquema llamado interrupciones vectorizadas, que permite redireccionar el servicio a rutinas propias.
- El DMA es una técnica para la transferencia masiva de información de E/S a RAM sin intervención del μP.