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**.

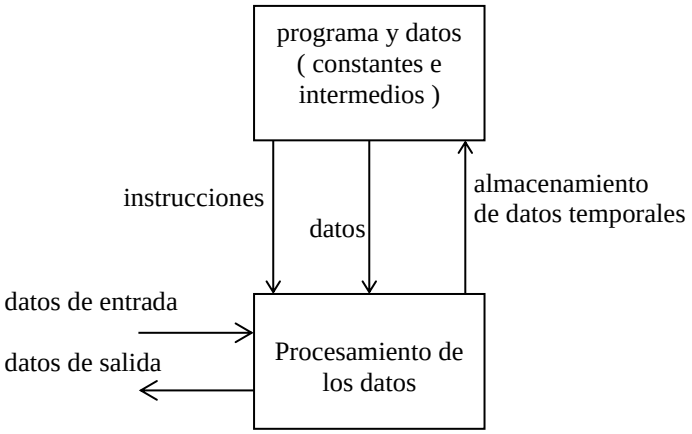


Figura 4-1: Esquema básico de un Sistema de Cómputo Programable

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 (ó **µP**), 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**.

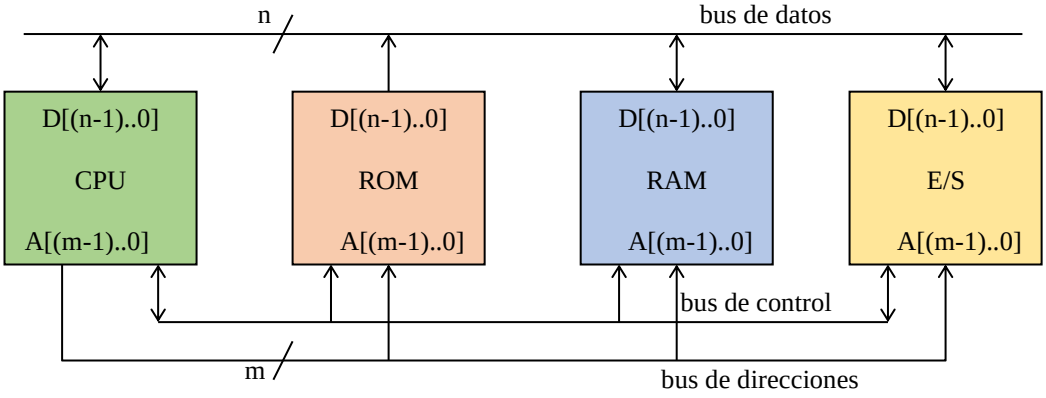


Figura 4-2: Arquitectura Von Neumann para realizar un Sistema de Cómputo Programable

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.
  Por ejemplo con m=16, el bus sería A[15..0] (significa A_{15} a A_0) y permite direccionar 2^{16}=65536 posiciones de memoria.
  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, C++, 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**. El proceso de traducción se denomina **compilación**, aunque puede implicar varios procesos intermedios (preprocesamiento, compilación, enlace).

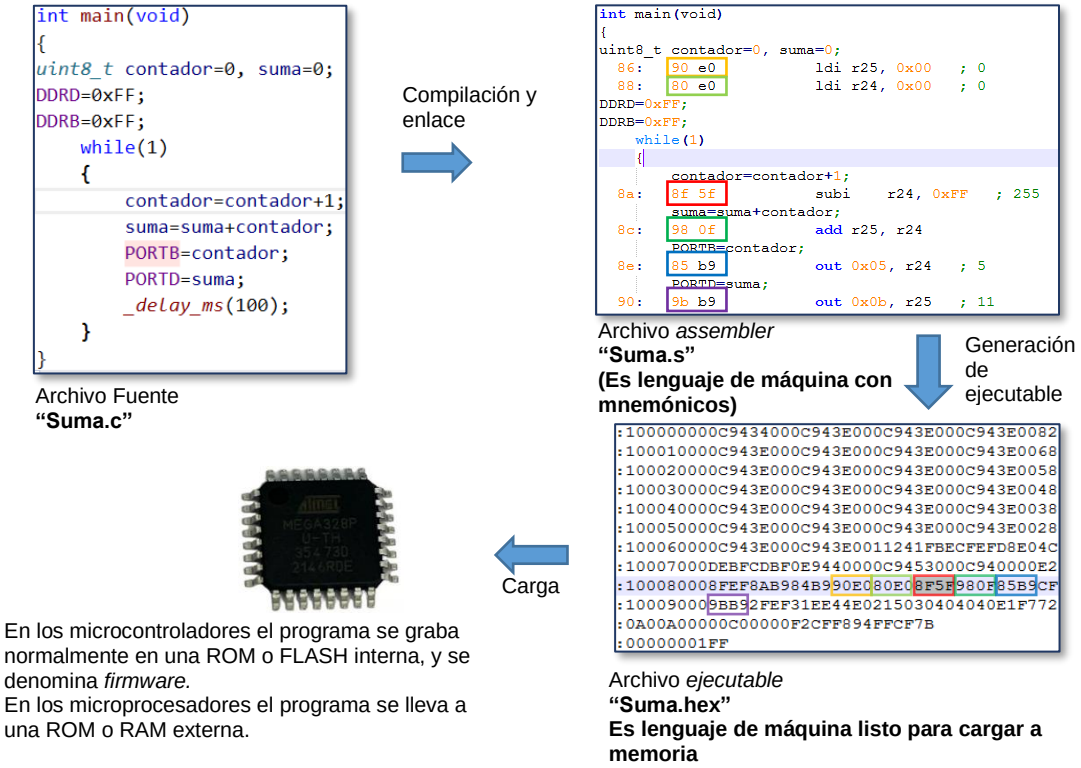


Figura 4-2-b: Etapas de compilación, enlace y carga de un programa en un uC o uP.

Veremos como ejemplo una instrucción de asignación en lenguaje de alto nivel. 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**

Según la complejidad de la arquitectura de la CPU, esta instrucción de alto nivel se realizará en una o más etapas.
Es decir, una CPU que acepte instrucciones **ternarias** (3 operandos) podría resolver esa instrucción de alto nivel con una sola instrucción en lenguaje de máquina, una CPU que acepte instrucciones de 2 operandos requerirá 2 instrucciones, y una CPU muy simple que acepte solo 1 operando por vez (instrucciones **unarias**), requerirá 3 instrucciones:

Por ejemplo

con 3 operandos podría ser

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

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 1 operando podría ser

```
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]
...
```

Nota1: 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*.

Nota 2: Estas instrucciones *lee*, *suma*, *escribe* son típicas de microprocesadores, por ejemplo *LD (lee)*, *ST (escribe), JMP (salta), ADD(suma)*. En la siguiente tabla se han extraído algunas de las **131** instrucciones de un microcontrolador AVR, utilizado en los Arduino Uno, Nano y Mega.

Mnemonics	Operands	Description	Operation	Flags	#Clocks
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP[1]	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL[1]	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1

Tabla 4-2-c: Fragmento del juego de instrucciones de un microcontrolador AVR.

En adelante nos referiremos al proceso de instrucciones *unarias*., para entender cómo trabaja un microprocesador elemental. **No es un microprocesador ni microcontrolador real, sino uno ficticio que nos permite entender los mecanismos básicos.**

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

```
leeC      4      ; lee constante 4
suma      [x]    ; suma variable x
resta     [y]    ; resta variable y
escribe   [y]    ; escribe en variable y el resultado acumulado
...
```

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

b - Un programa de un termostato, que en función de la comparación entre una lectura de un sensor de temperatura (Tentrada) y una consigna dada (Treferencia), activa o desactiva una salida, sería, en lenguaje de alo nivel:

"Si **Tentrada** < **Treferencia** → **Salida=1**, sino **Salida = 0**"

Por ejemplo, en lenguaje C

```
if (Tentrada<Treferencia)
    Salida = 1;
else
    Salida = 0;
```

Si Treferencia fuera una *constante* podría realizarse con las siguientes instrucciones:

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

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

```
ciclo       lee       [Tentrada]
            resta     [Treferencia]
            saltaN    encender      ; si resultado negativo salta
            lee       0
            escribe   [Salida]
            salta     ciclo
encender    leer      1
            escribe   [Salida]
            salta     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 los µPs actuales.
Cabe mencionar que esto es un set **no mínimo** pero básico, teniendo en cuenta que el primer microprocesador de Intel (4004) contaba con 41 instrucciones, y el último Intel Core i9 tiene unas 2000 instrucciones.

Microprocesador elemental (CPU)

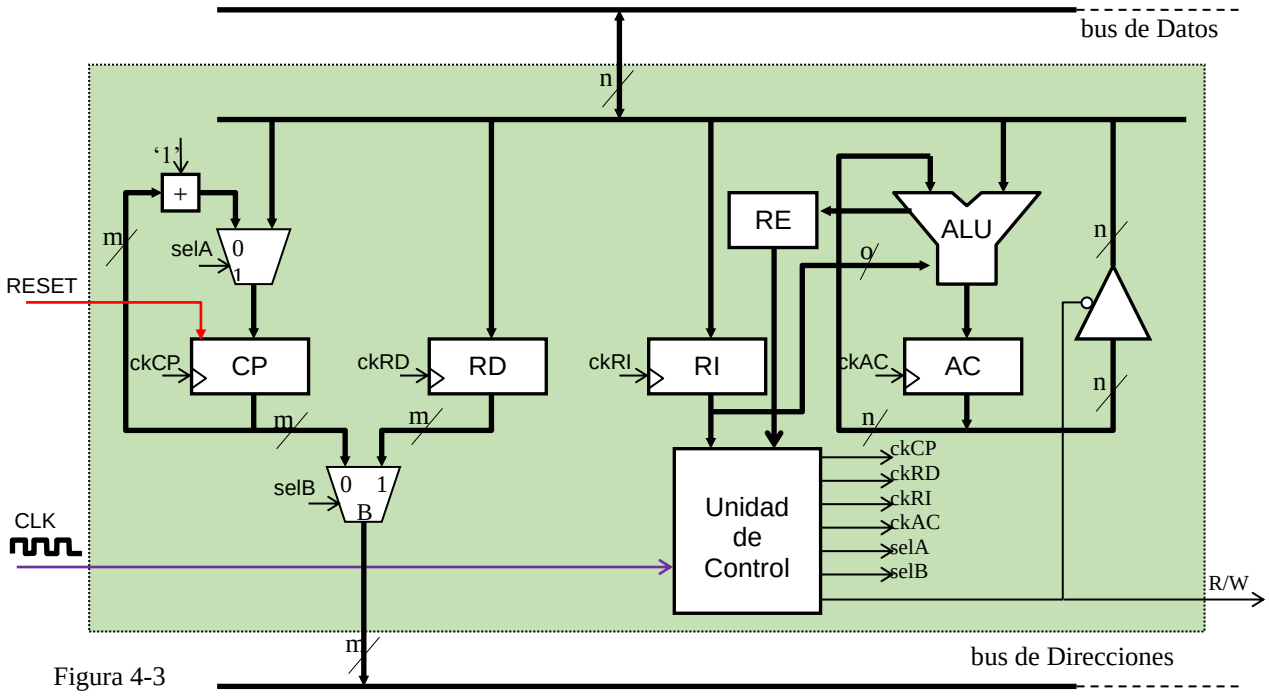


Figura 4-3

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 **activados por flanco** (*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.
La clave 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 velocidad con la que se generan los pulsos de la secuencia depende de la velocidad de la señal **CLK**, que es el **reloj** del sistema.
La UC de nuestro microprocesador elemental realiza sólo cuatro tipos de secuencias:

1. lectura *inmediata.* Para operar (*leer, sumar, restar etc*) constantes
2. lectura *direccionada* Para operar (*leer, sumar, restar etc*) variables de RAM
3. escritura Para escribir variables en RAM
4. salto. Para saltar a otro punto del programa

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, que sería la instrucción en *assembler*.

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: Set de instrucciones de nuestro microprocesador **ficticio** (¡no memorizar!)

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.

Ejemplo de ejecución de un programa elemental

En este ejemplo el µP está conectado a una memoria RAM|ROM que tiene cargada un programa correspondiente a la operación c=a+b en las direcciones 00 a 05 (zona de instrucciones), y las variables x, y en las direcciones 81 y 82 (zona de datos).

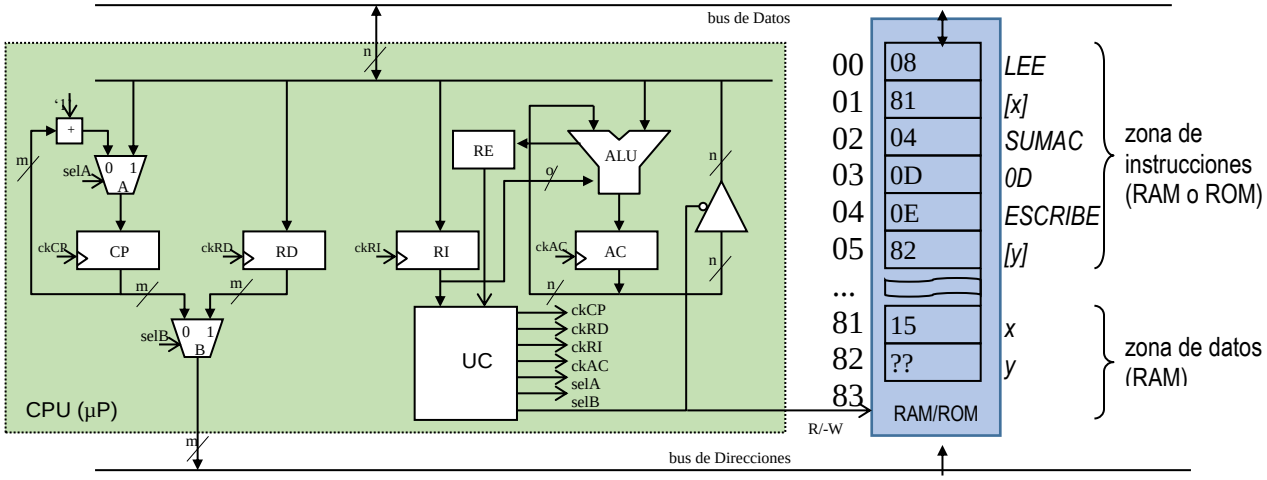


Figura 4-5: Conexión CPU y RAM/ROM

1. Al inicio, mediante la señal RESET, se pone en '0' el Contador de Programa, es decir CP=00. Además, las líneas selA=0, selB=0, R/-W=1.
   Observe que como selB=0, al bus de direcciones sale el valor de CP, que es 00. Como su entrada R/-W=1 la memoria está en **modo lectura** y presenta el contenido de la posición 00 al bus de datos (en este caso el 08).
2. La primera señal en ser activada por la UC (luego del RESET) es **clkRI** (cargar el Registro de instrucciones). El valor cargado en el RI es el 08 que entrega la memoria, es decir el código de instrucción ***LEE***. 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" segunda parte de la instrucción, que es 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 en 1). 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.
4. Este valor es la dirección del operando **x**, y debe ser utilizado para acceder a dicho operando. Para eso la UC activa la señal **clkRD** (carga RD con el 81).
5. La UC pone **selB='1'**, de modo que este valor 81 del RD se "reinyecta" por el bus de direcciones. Como R/-W=1 la memoria presenta el número '15' (valor de **x**) al bus de datos.
6. La UC activa **clkAC** y el '15' del bus de datos pasa por la ALU y se carga en el Acumulador AC.
7. La UC pone **selB='0'** para volver a conectar CP al bus de direcciones.
8. Corresponde ahora buscar la próxima instrucción (que está en la posición 02 de memoria) , para lo cual la UC activa **clkCP** (se incrementa CP a 02.) Este valor 02 sale al bus de direcciones.
9. La UC repite ahora el paso 2, es decir activa **clkRI** (cargar el Registro de instrucciones). En este caso se carga en RI el valor '04' de la memoria, es decir la instrucción ***SUMAC***. Este código '04' se presenta tanto a la UC como a la ALU. Con el código '04' la ALU se pone en modo de suma entre ambas ramas. Por su parte, la UC realizará la secuencia de *lectura inmediata*. El próximo dato a leer de la memoria es directamente el operando a sumar.
10. Para esto la UC activa **clkCP**, lo que provoca que CP se incremente a 03. Sale al bus de direcciones este valor 03, y la memoria dará el valor **0D**, que es **directamente** el operando a sumar. En este momento la ALU tiene presentado en su entrada izquierda el valor del registro Acumulador, es decir el 15 almacenado en el paso 6, y en su entrada derecha el valor 0D tomado del bus de datos (de la memoria). Como está en modo suma, en su salida se está produciendo la suma de ambos valores (15h + 0Dh = 22h).

11. La UC activa entonces **clkAC** y el '22' de la salida de la ALU se carga en el Acumulador AC.
12. Corresponde ahora avanzar a la próxima instrucción, por lo que se incrementa CP (la UC activa **clkCP**).

En síntesis, para la instrucción LEE [**x**] la UC realizó la secuencia (pasos 2 a 8)

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

Para ejecutar la instrucción SUMAC 0D la UC realizó la secuencia (pasos 9 a 12)

clkRI, clkCP, clkAC, clkCP

Como se ve ambas secuencias son similares, aunque la secuencia con variable tiene unos pasos adicionales (en celeste).
Luego del paso 12 sale al bus de direcciones el valor de CP, que es '04'. La memoria presenta el número '0E' (instrucción ***ESCRIBE***) al bus de datos.

Para realizar la instrucción ESCRIBE [**y**] la UC realiza la siguiente secuencia

13. Se activa **clkRI**. Ahora no va a importar en qué modo esté la ALU puesto que no se transferirá información a AC. La UC realizará una secuencia de *escritura*.
14. Corresponde "buscar" la dirección del operando donde debe escribir. Para esto activa **clkCP**, y CP se incrementa a '05'. Sale al bus de direcciones este el valor. Como R/W=1 la memoria presenta el número '82' al bus de datos.
15. Este valor es la dirección de la variable **y**, y debe ser utilizada para acceder a dicha variable. Para eso la UC activa la señal **clkRD** (carga RD con el '82').
16. Al poner **selB='1'** sale al bus de direcciones el valor de RD, que es '82'. Obsérvese que hasta este momento siempre ha estado **R/-W**=1, es decir se ha estado leyendo la memoria. En el bus de datos estará el valor actual de y (que no interesa y hemos simbolizado con "??").
17. **Ahora no se activa la señal clkAC, sino se hace R/W=0**. 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 (22). Es decir, se transfiere el valor 22 a la posición 82 de RAM (variable **y**).
18. Antes de cambiar la dirección del registro accedido debe volverse la memoria a modo lectura, es decir R/W=1

En síntesis, para la instrucción ESCRIBE [**c**] la UC realiza la secuencia (pasos 13 a 19)

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

Comparando la secuencia de leer variable y la de escribir variable, se ve que la diferencia es que en vez de clkAC se produce un pulso de bajada en R/-W.

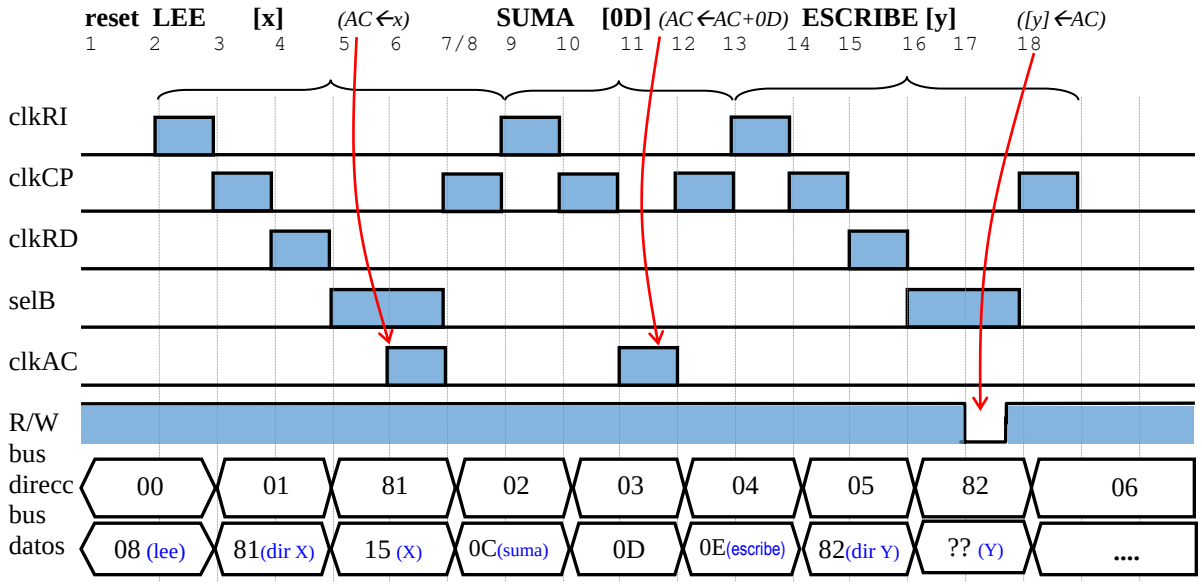


Figura 4-6: Diagrama de tiempos de las señales en el ejemplo anterior

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.

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 b y c (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]
            saltaN    encender      ; si resultado negativo salta
            lee       0
            escribe   [Salida]
            ir a      ciclo
encender    leer      1
            escribe   [Salida]
            ir a      ciclo
```

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.
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.

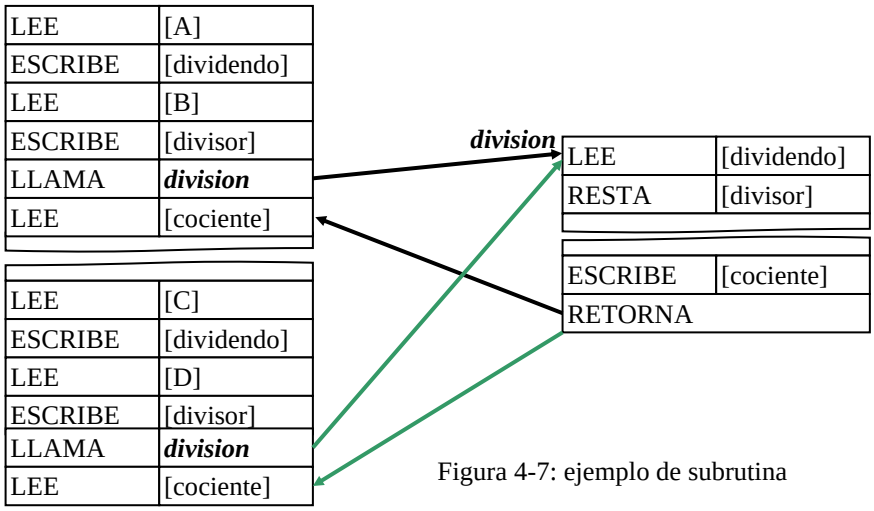


Figura 4-7: ejemplo de subrutina

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*) o una llamada recursiva de una subrutina a sí misma se excede el área de memoria reservada para la pila.
En algunos µPs 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**.
- 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.

b - Interfaces de Entrada / Salida

Los sistemas de cómputo, sean computadoras personales, controladores industriales, microcontroladores tipo Arduino u otros, necesitan interfaces para comunicarse con dispositivos externos.
En el caso de una PC, se intercambia datos con los periféricos (monitor, mouse, teclado, impresora, modem, sonido, webcam) y unidades de almacenamiento (disco rígido, CDROM, pendrives). Cada uno de estos dispositivos tiene sus propios requerimientos, que son atendidos por un hardware de interfaz, por ejemplo:

Disco rígido o sólido:	Controladora (SATA, PCIe etc)
Monitor:	Controladora de video
Impresora, mouse, webcam:	Puerto serie o puerto USB
Sonido:	Placa multimedia
Comunicaciones:	Placa de red, placa WiFi, Bluetooth etc.

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.

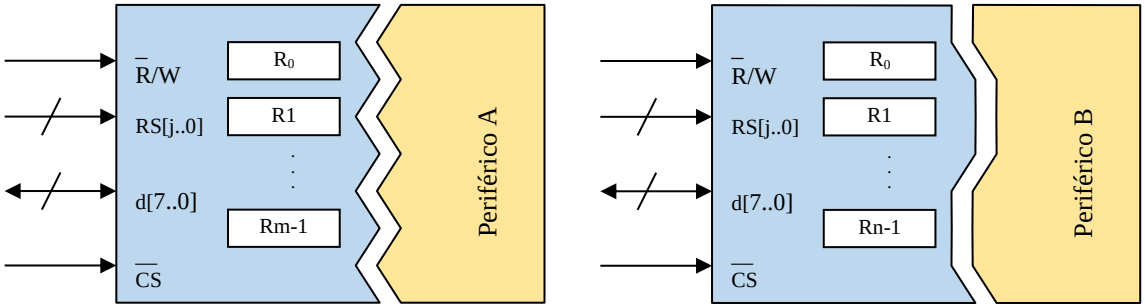


Figura 4-8. Interfaces de E/S: señales básicas

En la **figura 4-8** se representan interfaces con sus señales básicas (costado izquierdo, celeste), y su vinculación con el periférico (a la derecha, amarillo). 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 (celeste) 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). 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]).

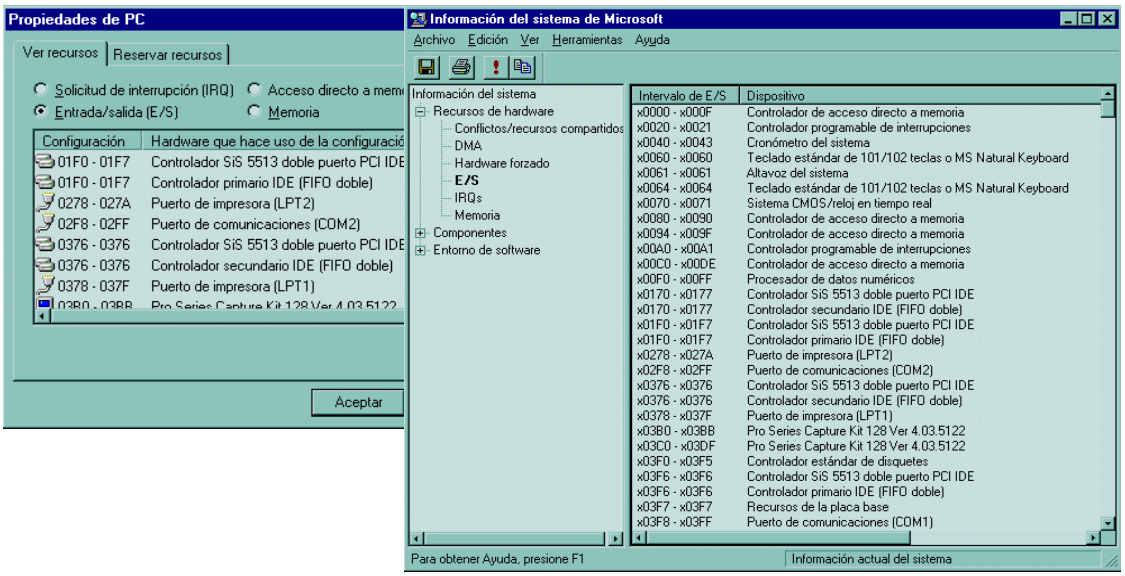


Figura 4-9. Reportes de E/S en Windows en una PC

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 notebook, 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.

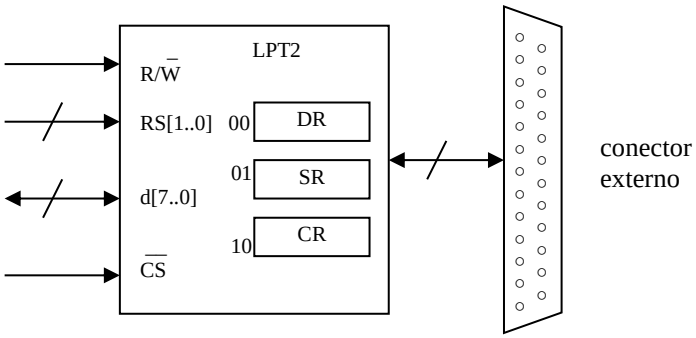


Figura 4-10. Puerto paralelo, señales básicas

- 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[15..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 0278h-027Ah. (el 027Bh no se utiliza)

		a[15..0] (bus de direcciones)				
		a[15..8]	a[7..4]	a[3..0]		registro
otro	0277h	00000010	0111	01	11	
LPT2	0278h	00000010	0111	10	00	DR
	0279h	00000010	0111	10	01	SR
	027Ah	00000010	0111	10	10	CR
	027Bh	00000010	0111	10	11	--
otro	027Ch	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.

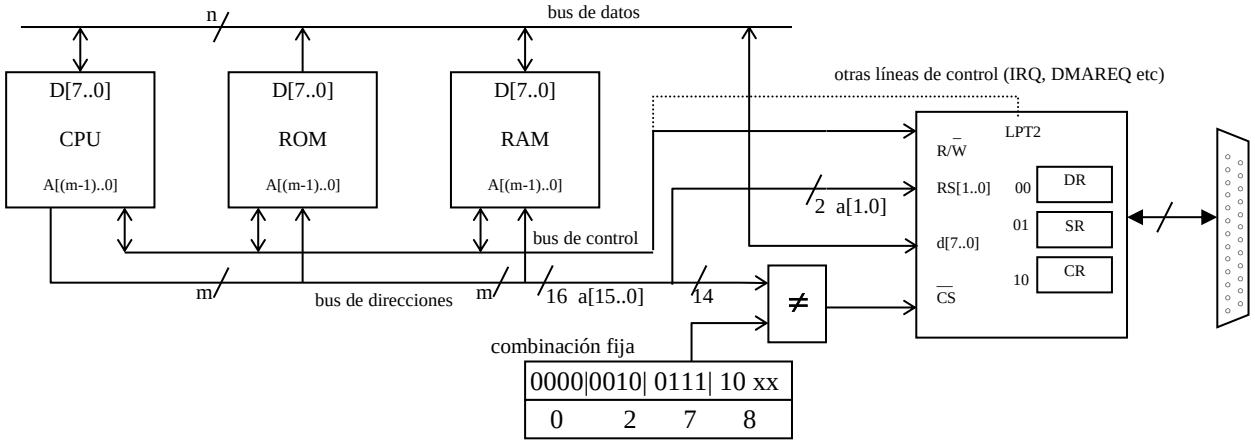


Figura 4-11. Modo de integración de una interfaz de E/S al sistema de buses de la PC

En los microcontroladores todos los dispositivos de E/S están dentro del mismo chip, así como la memoria RAM y ROM/FLASH. Por ejemplo en los microcontroladores AVR,las primeras 255 posiciones de memoria RAM están reservadas para los registros que manejan estos dispositivos de E/S. En la tabla 4-12 se muestra un fragmento, desde la dirección 0x20 a la dirección 0x61.

0x15 (0x35)	TIFR0	0x2B (0x4B)	GPIOR2	(0x61)	CLKPR
0x14 (0x34)	Reserved	0x2A (0x4A)	GPIOR1	(0x60)	WDTCSR
0x13 (0x33)	Reserved	0x29 (0x49)	Reserved	0x3F (0x5F)	SREG
0x12 (0x32)	Reserved	0x28 (0x48)	OCR0B	0x3E (0x5E)	SPH
0x11 (0x31)	Reserved	0x27 (0x47)	OCR0A	0x3D (0x5D)	SPL
0x10 (0x30)	Reserved	0x26 (0x46)	TCNT0	0x3C (0x5C)	Reserved
0x0F (0x2F)	Reserved	0x25 (0x45)	TCCR0B	0x3B (0x5B)	Reserved
0x0E (0x2E)	Reserved	0x24 (0x44)	TCCR0A	0x3A (0x5A)	Reserved
0x0D (0x2D)	Reserved	0x23 (0x43)	GTCCR	0x39 (0x59)	Reserved
0x0C (0x2C)	Reserved	0x22 (0x42)	EEARH	0x38 (0x58)	Reserved
0x0B (0x2B)	PORTD	0x21 (0x41)	EEARL	0x37 (0x57)	SPMCSR
0x0A (0x2A)	DDRD	0x20 (0x40)	EEDR	0x36 (0x56)	Reserved
0x09 (0x29)	PIND	0x1F (0x3F)	EECR	0x35 (0x55)	MCUCR
0x08 (0x28)	PORTC	0x1E (0x3E)	GPIOR0	0x34 (0x54)	MCUSR
0x07 (0x27)	DDRC	0x1D (0x3D)	EIMSK	0x33 (0x53)	SMCR
0x06 (0x26)	PINC	0x1C (0x3C)	EIFR	0x32 (0x52)	Reserved
0x05 (0x25)	PORTB	0x1B (0x3B)	PCIFR	0x31 (0x51)	Reserved
0x04 (0x24)	DDRB	0x1A (0x3A)	Reserved	0x30 (0x50)	ACSR
0x03 (0x23)	PINB	0x19 (0x39)	Reserved	0x2F (0x4F)	Reserved
0x02 (0x22)	Reserved	0x18 (0x38)	Reserved	0x2E (0x4E)	SPDR
0x01 (0x21)	Reserved	0x17 (0x37)	TIFR2	0x2D (0x4D)	SPSR
0x0 (0x20)	Reserved	0x16 (0x36)	TIFR1	0x2C (0x4C)	SPCR

Tabla 4-12. Fragmento de mapa de E/S de un microcontrolador Atmega328 (Arduino Nano y Uno)

Se han resaltado en celeste los puertos paralelos, que son la conexión directa con los pines de salida del microcontrolador, pero puede haber también puertos de comunicaciones serie síncronos y asíncronos, temporizadores y otros específicos. Por ejemplo, para escribir un valor binario '01010101' en el puerto PORTD se puede utilizar la instrucción
PORTD=0b01010101;

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

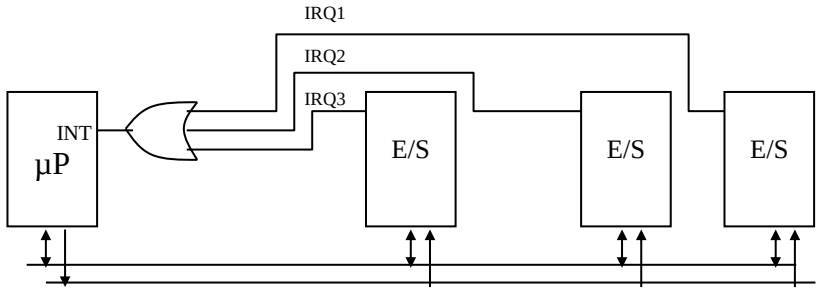
polling

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.

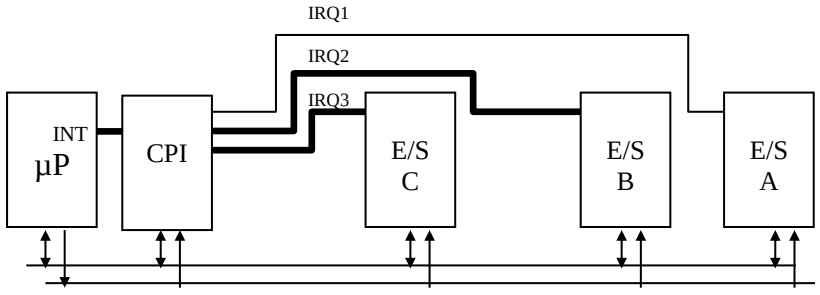
Interrupción externa

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 4-13a** 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 (es decir un *polling*) 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 4-13. (a)Atención de varios dispositivos con una única línea de interrupción. (b) Controlador de interrupciones.

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 4-13b.
El CPI es en sí mismo un dispositivo de E/S basado en el esquema de la figura 4-14. 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 4-14

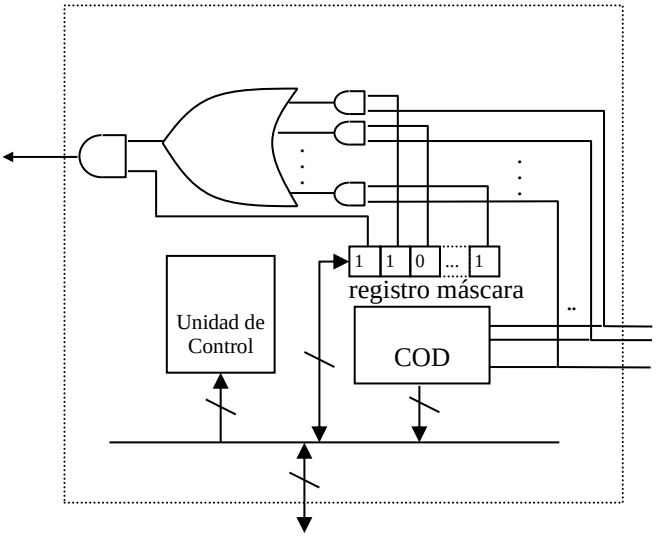


Figura 4-14. Controlador de interrupciones

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, como error de división, código de operación inválido, dispositivo no disponible y otros.

Concepto de interrupción vectorizada:

Cada interrupción debe tener su propia rutina de servicio, cuyo comienzo estará en una dirección de RAM o ROM. En los sistemas con interrupciones vectorizadas (como las PC o los microcontroladores AVR) dichas direcciones están escritas en una tabla o *vector* de memoria. (en el caso de los AVR es en la memoria FLASH de programa).
Cuando se produce una interrupción, el programa salta a una posición específica de dicha tabla, desde donde salta inmediatamente a la rutina de servicio.
Este mecanismo permite disminuir la latencia en la atención de la interrupción. En microcontroladores muy simples, sin interrupciones vectorizadas, todas las interrupciones saltan a un mismo punto de la memoria de programa, por lo cual allí debe realizarse la verificación o *polling* para saber qué dispositivo requiere atención.

d - Transferencia masiva de datos-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.

e - 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 periféricos 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.

f - Microcontroladores

Hace varias décadas los sistemas programables aplicados a automatización y control, que denominaremos Controladores Programables (ejemplo PLC, PID y otros) constaban de un chip microprocesador (CPU) y un conjunto de chips de memoria RAM, ROM, comunicaciones y otros periféricos, de forma similar a una computadora personal aunque con algunas diferencias:

	Computadora Personal (PC)	**Controlador Programable**
	Propósito general (oficina, diseño, juegos)	Control, automatización, adquisición, monitoreo.
Software vs Firmware	Múltiples aplicaciones instaladas en unidad de almacenamiento (disco), que se copian a la RAM y desde allí se ejecutan. "Software"	Un solo programa (en general) grabado en una memoria ROM/EEPROM/FLASH, que se ejecuta desde esa misma memoria (a veces se copia a RAM y se ejecuta). "Firmware"
	Susceptible a eventuales fallas y con tiempos de respuesta variables, por la propia complejidad en la ejecución de múltiples tareas y aplicaciones.	Robusto y determinista (tiempos de respuesta bien determinados).
Sistema Operativo, RTOS y *Bare metal*	Siempre lleva un sistema operativo (Linux, Windows, etc.) como intermediario entre el hardware (periféricos) y las aplicaciones.	A veces lleva un Sistema Operativo, aunque mayormente las aplicaciones trabajan directamente sobre el hardware (*Bare-Metal*)
	Gran capacidad de cómputo, de almacenamiento en disco y en RAM.	Capacidad de cómputo y almacenamiento reducidas.

El desarrollo de un controlador programable implicaba la compleja tarea de desarrollar la **placa** para comunicar CPU, RAM, ROM y periféricos. Dado que un sistema programable es por definición flexible, se pensó en desarrollar un chip que integrara estos sistemas.

8048: El primer Microcontrolador

El primer microcontrolador, el 8048 de Intel (1976), integraba la CPU, una ROM de programa de 1024 bytes, una RAM de 64 bytes, 27 líneas de E/S y un temporizador/contador (*Timer/counter*) de 8 bits. Sólo 4 años después, en 1980, apareció el famoso Intel 8051, cuya arquitectura es aún utilizada en muchos microcontroladores modernos. En este microcontrolador se agregaron más temporizadores y un puerto de comunicaciones (UART).

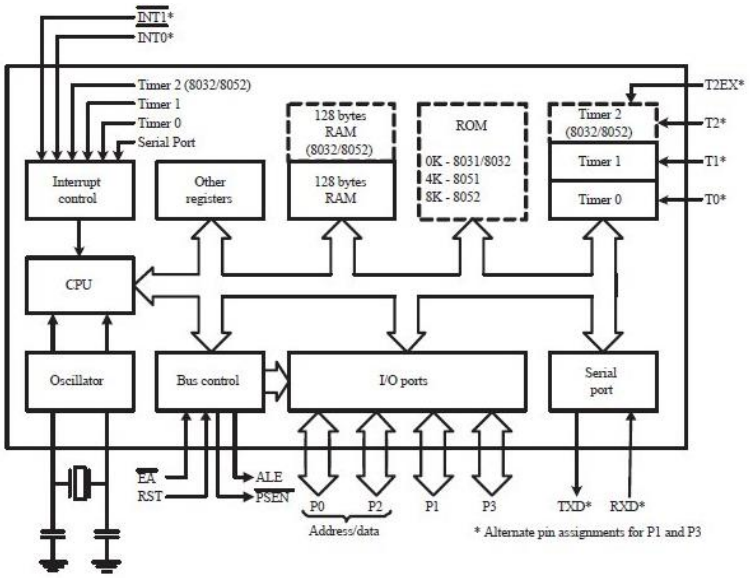


Figura 4-15. El 8051 de Intel (y variantes)

Elementos de un microcontrolador:

GPIO

Puerto

Timer

UART

Como vimos, ya en 1976 el 8048 tenía 27 líneas de Entrada/Salida. A estas líneas se las denomina habitualmente **GPIO** (*General Purpose Input Output*), y se las puede usar como salidas (para activar/desactivar algún actuador externo) o como entradas (para leer el estado de un pulsador o sensor digital externo). Estas salidas están agrupadas en **Puertos** (IO Ports).
También el 8048 disponía de un **Timer**/counter de 8 bits con el que podía contar eventos y medir tiempos.
El 8051 incorporó una UART (Universal Asynchronous Receiver/Transmitter) de 8 bits y más *timers*.
Luego de Intel siguieron otros fabricantes. Motorola lanzó su MC6801 en 1977 (solo un año después de Intel) y luego se sumaron otros, incorporando más periféricos, diversificando en gamas y especializándose. Hoy existen **miles** de variantes de microcontroladores de 8, 16 y 32 bits,con diferentes arquitecturas y periféricos. De manera general mostramos en el esquema de la **figura 4-16** los subsistemas que componen los microcontroladores actuales, identificando los periféricos más comunes. Como se ha intentado representar, el bloque de E/S es normalmente contiguo al bloque de RAM. En la mayoría de los microcontroladores los registros para manejar los periféricos están **mapeados en RAM**, es decir se puede acceder a ellos como si fueran posiciones de memoria RAM. Por ejemplo, escribiendo un cierto byte en un registro correspondiente a un **puerto** de GPIO se puede cambiar el estado (alto/bajo) de los pines de ese puerto.

Periféricos mapeados en RAM

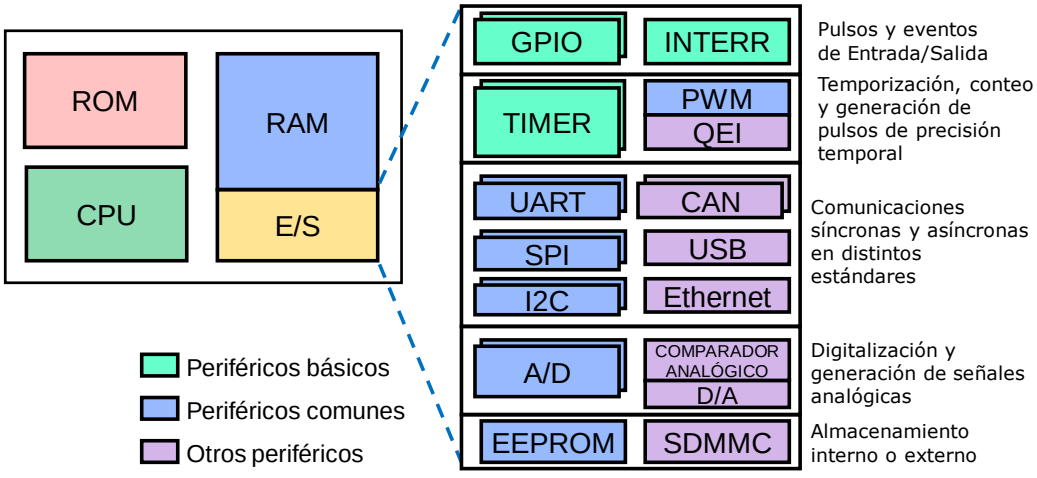


Figura 4-16. Elementos de un microcontrolador

Por ejemplo, el ATmega328P, microcontrolador presente tanto en las placas Arduino Nano como Arduino UNO, posee los siguientes periféricos:

- 3 puertos GPIO, PB[7..0], PC[6..0] y PD[7..0] (PB7 y PB6 son usados por oscilador a cristal)
- 3 Timers, que permiten medir tiempos y generar hasta 6 señales PWM.
- 1 UART (Universal Asynchronous Receiver-Transmitter). Interfaz serie asíncrona que también se puede programar como unidad síncrona.
- 1 SPI (Serial Peripheral Interface). Interfaz serie síncrona full duplex.
- 1 I2C (Inter-Integrated Circuit). Interfaz serie síncrona que permite conexión en bus.
- 1 A/D Conversor Analógico/Digital de 10 bits de resolución y hasta 8 canales multiplexados
- 1 EEPROM. Memoria de datos persistentes de 1kB (independiente de la memoria FLASH de programa)

Como se observa en las figuras 4-17, los pines están compartidos por los distintos periféricos, por ejemplo el pin 28 puede ser un GPIO (PC5), o el clock del I2C (SCL), el canal 5 del A/D (ADC5) etc. Según el periférico que se habilite (mediante el programa) será la función del pin.

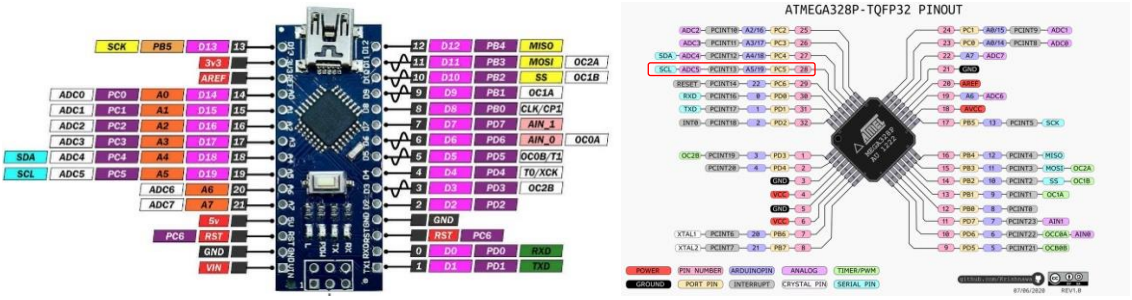


Figura 4-17. Placa Arduino NANO y su microcontrolador ATmega328P