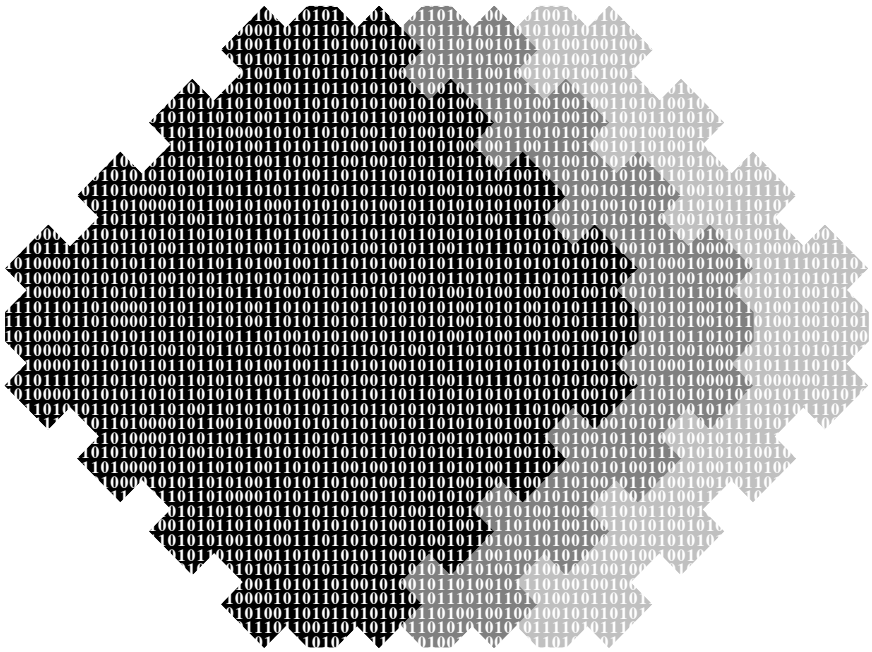
# Sección 1

## Arquitectura de Computadoras, Microprocesadores y Microcontroladores



## Introducción

Durante los últimos 30 años, la revolución en las computadoras ha cambiado dramáticamente nuestro mundo y promete traer aun grandes cambios en los próximos años. Las computadoras digitales de uso general construidas hoy en día son mucho más rápidas, pequeñas y más confiables que las primeras computadoras y pueden producirse a un costo más bajo. Las nuevas tecnologías, diferentes arquitecturas y memorias mucho más rápidas están teniendo gran impacto en las computadoras. Pero además, lo más significante son las diversas formas en las cuales se ha aprendido a utilizar las computadoras.

Las primeras computadoras electrónicas de gran tamaño fueron utilizadas como supercalculadoras para resolver problemas matemáticos complejos que habían sido imposibles de resolver anteriormente. En años recientes, los especialistas junto con programadores de computadoras han empezado a utilizar las computadoras para aplicaciones de tipo diferente al numérico, tales como sistemas de control, comunicaciones, inteligencia artificial, reconocimiento de patrones y procesamiento de señales.

Recientemente, la presencia de microcomputadoras simultáneamente ha acelerado y expandido el impacto de la revolución en computadoras. La presencia de una computadora en un IC[1], ha resultado en la disponibilidad de sistemas de microcomputadoras con la funcionalidad y comportamiento de sistemas de minicomputadoras a un costo mucho menor. Los microprocesadores han aparecido como tipo de controladores realmente económicos e impresionantemente inteligentes en electrodomésticos, juegos, juguetes, cámaras fotográficas, automóviles y un gran número de artículos de consumo general. Por consiguientes, la función e importancia del programador de computadoras se ha extendido considerablemente tanto en amplitud como en profundidad.

Los lenguajes naturales son imprácticos para el uso en computadoras, los lenguajes de programación tales como FORTRAN, BASIC, PASCAL y C, que contienen estructura definida, precisa y con sintaxis, simplifican la comunicación con una computadora, dichos lenguajes están orientados hacia problemas y contienen expresiones y palabras familiares. Los lenguajes de programación se han desarrollado para tener control programado de máquinas, adquisición de datos, instrucciones ayudadas por computadora y muchas otras aplicaciones. En el futuro surgirán muchos más lenguajes de programación. Cada lenguaje nuevo desarrollado permitirá al usuario aplicar más fácilmente la potencia de la computadora a su problema en particular.

[1]Siglas de *Integarted Circuit*

## 1.1 ELEMENTOS ESENCIALES DE UNA COMPUTADORA

Las computadoras se han utilizado de forma general desde los años 50's. En un principio las computadoras digitales eran sistemas grandes y costosos utilizados por el gobierno, universidades y grandes empresas. El tamaño y forma de las computadoras digitales cambiaron gracias a la invención del circuito integrado (IC). El cual permitió tener todo un procesador en una sola pastilla denominándolo *microprocesador*. El microprocesador es un pequeño, pero extremadamente complejo dispositivo LSI[2] o VLSI[3]. Las computadoras utilizan un *programa almacenado*. Una computadora utiliza un microprocesador y algún tipo de memoria semiconductora.

Las computadoras habitualmente son unidades de *propósito general*. Normalmente se programan muchas veces y se utilizan para realizar varias tareas. Las *computadoras dedicadas* o *sistemas incrustados o empotrados*[4] se emplean cada vez más debido al uso de los microprocesadores que actualmente son pequeños y de bajo costo. Un sistema incrustado se programa para realizar solo pocas tareas, como ocurre en los juguetes, automóviles, herramientas, sistemas automatizados etc...

Debido a que los sistemas incrustados son computadoras dedicadas, su organización es igual a la de una computadora y la base que conforma una computadora es la unidad central de procesamiento o CPU (*Central Processing Unit*), la sección memoria y la sección de entrada salida o E/S. Estas tres secciones están interconectadas por tres conjuntos de líneas paralelas llamados buses o ductos. Estos tres ductos son: el *ducto de direcciones*, el *ducto de datos* y el *ducto de control*. La figura 1 y 2 muestran un diagrama de bloques simplificado de una microcomputadora.

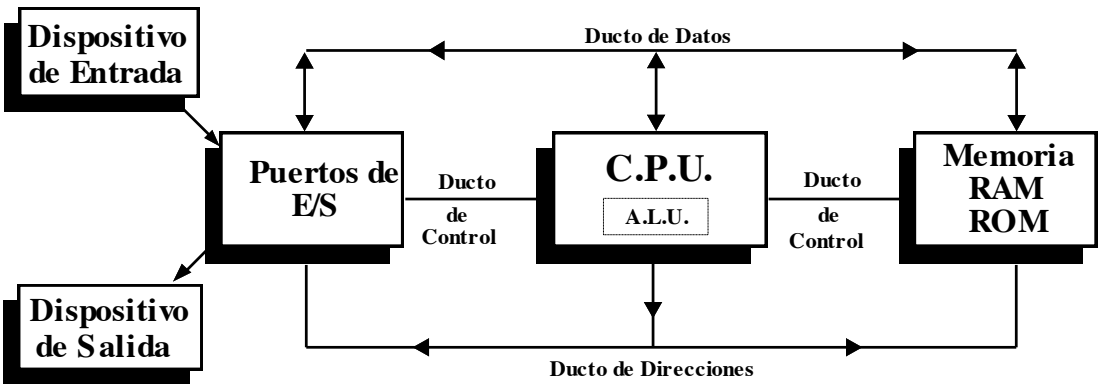



**Figura 1.** Diagrama de bloques simplificado de una microcomputadora.

[2] *Large Scale Integration* (Alta Escala de Integración)
[3] *Very Large Scale Integration* (Muy Alta Escala de Integración)
[4] Traducción de *Embedded Systems*

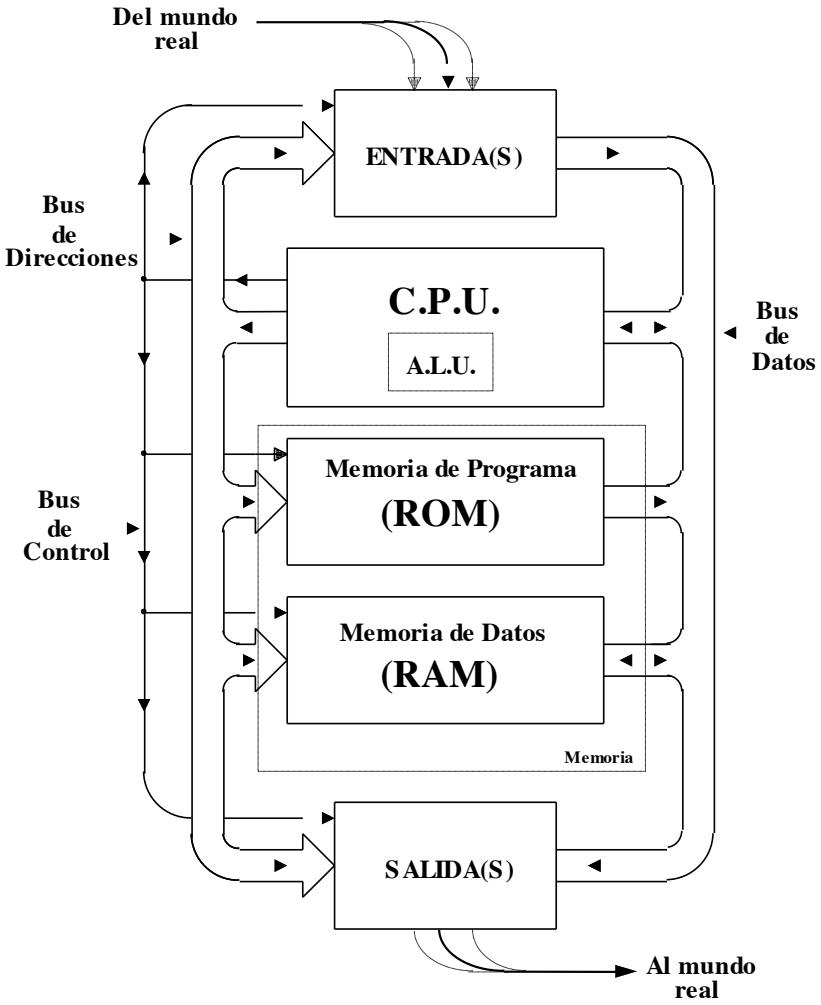



**Figura 1.2** Diagrama de bloques de un sistema basado en microprocesador.

### 1.1.1 Unidad Central de Procesamiento (CPU)

La unidad central de procesamiento o CPU controla las operaciones del sistema computarizado, este trae el código binario de la instrucción desde la memoria, decodifica las instrucciones a una serie de acciones simples y lleva acabo tales acciones. EL CPU contiene una unidad aritmética y lógica o ALU, la cual realiza operaciones como sumar, restar, or, and, xor, invertir etc. sobre palabras binarias, cuando las instrucciones así lo requieran. El CPU también contiene un *contador de direcciones o contador de programa* el cual se utiliza para retener la dirección de la próxima instrucción o dato a ser traído desde la memoria, además contiene registros de propósito general los cuales se utilizan para almacenar temporalmente datos binarios y una circuitería de control que genera las señales del ducto de control.

### 1.1.2 Unidad Aritmética y Lógica

La unidad aritmética y lógica o ALU (*Arithmetic Logic Unit*) es una sección de la unidad central de procesamiento (CPU) que realiza operaciones aritméticas y lógicas sobre datos llevados a ella. La ALU típicamente opera sobre uno o dos valores llamados *operandos* y los cambia de alguna manera en acorde a un operador que se especifica. Por ejemplo, la instrucción:

$$C \Leftarrow A + B$$

A y B son los operandos, C el resultado, + es un operador lógico OR y $\Leftarrow$ es un operador de devolución. Esto se muestra gráficamente en la figura 1.3.

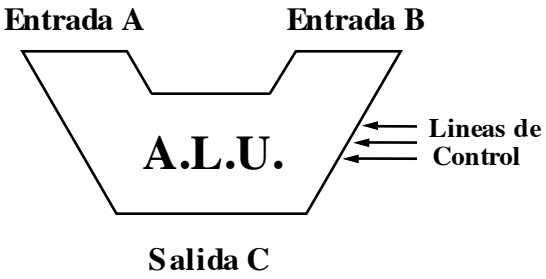



**Figura 1.3 U**nidad Aritmética y Lógica básica que muestra dos valores de entrada, señales de control para seleccionar la operación a realizar y su salida.

La ALU puede realizar multiplicaciones de operandos, esto seleccionando esta operación particular mediante las líneas de control.

### 1.1.3 Unidad de Entrada/Salida (E/S)

La sección de entrada y salida (E/S) permite a la computadora tomar datos del mundo real o mandar datos al mundo real. Los periféricos tales como teclados, pantalla, impresores y módems se conectan a la sección de E/S. Esta sección permite que el usuario y la computadora se comuniquen en diferentes direcciones, es decir usuario→ computadora y computadora→usuario.

Los dispositivos físicos utilizados para interfazar los ductos de la computadora a sistemas externos se les denominan puertos. Un puerto de entrada permite que la información de un teclado o un convertidor analógico digital (ADC *Analog to Digital Converter*) o alguna otra fuente pueda ser leído por la computadora bajo el control del CPU. Un puerto de salida se utiliza para mandar información de la computadora a algún periférico como una pantalla, impresora o un convertido digital analógico (DAC *Digital to Analog Converter*).

Físicamente un puerto de entrada o de salida es un conjunto de flip-flops tipo D los cuales permiten el paso de la información cuando son habilitados o activados por una señal de control del CPU.

### 1.1.4 Unidad de Memoria

La unidad de memoria generalmente de una mezcla de RAM (*Random Access Memory*) y ROM (*Read Only Memory*). También puede tener dispositivos de memoria diferentes a RAM y ROM (memorias de estado sólido) como lo son los discos ópticos, discos duros y discos flexibles. La memoria tiene dos propósitos principales, los cuales que se muestran a continuación:

- Almacenar códigos binarios de la secuencia de instrucciones que se quiere ejecutar por el sistema.
- Almacenar el código binario de los datos con los cuales se trabajará.

La unidad de memoria almacena información binaria en grupos de bits denominados *palabras*. Una palabra en la memoria es una entidad de bits que se introducen o se sacan del almacenamiento como una unidad. Una palabra de memoria es un grupo de unos (1's) y ceros (0's) que puede representar un número, carácter o código de instrucción, etc., es decir información para el CPU. Un grupo de ocho bits se denomina *byte*. La mayoría de las computadoras utilizan palabras cuyo número de bits es múltiplo de 8, por lo tanto, una palabra de 16 bits contiene dos bytes y una de 32 bits se forma por 4 bytes. La capacidad de memoria en las computadoras comerciales se define como la cantidad total de bytes que puede almacenar.

### 1.1.5. Hardware, Software y Firmware

Cuando se trabaja en el ambiente de las computadoras constantemente se utilizan los términos hardware, software y firmware. *Hardware* es el nombre que se le da a los dispositivos físicos y circuitos de la computadora. *Software* se refiere a los programas escritos para la computadora. *Firmware* es el término que se le da a los programas almacenados permanentemente (programas en ROM).

## 1.2 MICROPROCESADOR: COMPUTADORAS DE MAS DE UN CIRCUITO INTEGRADO (IC)

El primer microprocesador comercial fue el 4004, introducido en 1971. Desde entonces muchas mejoras se han hecho por muchos fabricantes, una de ellas es la expansión de la palabra del microprocesador de 4 bits a 8, 16, 32 bits (en algunos casos hasta 64 bits). Además de ello, muchas de las capacidades de las computadoras de gran escala (computadoras, minicomputadoras) se han incorporado es estos microprocesadores, por ejemplo ALU de punto flotante, unidades de administración de memoria etc.; los microprocesadores han evolucionado como resultado de intentar incorporar más elementos de una computadora en un solo IC.

Anteriormente la unidades centrales de procesamiento de las computadoras eran un conjunto de circuitos integrados interconectados para trabajar en conjuntamente y realizar la funciones de dichas unidades. Ahora los procesadores están diseñados en un solo microcircuito, es por ello que se le denomina **micro**procesador, pero aun no son computadoras de un solo IC, debido a que carece de memoria y de puertos de entrada/salida (E/S).

### 1.2.1 Ductos del Microprocesador

Existen dos conjuntos de ductos distintos que pueden ser identificados en un microprocesador. Uno de estos conjuntos son los ductos internos que forman parte de la organización interna del microprocesador y este es de gran interés para los diseñadores del IC y poco concerniente para los diseñadores de aplicaciones con microprocesadores. Estos ductos internos no tienen efecto sobre la interfaz del IC con el exterior y solo afectan sobre la velocidad efectiva del microprocesador.

El segundo conjunto de ductos se refiere a la estructura de los ductos externos del microprocesador. Existen alternativas para ser diferenciados, las cuales son el número de direcciones o número de bits que simultáneamente puede presentar (ancho del bus), el tipo de interacción del bus con el CPU y los dispositivos externos, y la dedicación o función del bus. Generalmente los microprocesadores y computadoras poseen tres ductos los cuales son: *Ducto de Direcciones, Ducto de Datos y Ducto de Control*.

#### 1.2.1.1 Ducto de Direcciones

El bus de direcciones consiste de 16, 20, 24 o más líneas de señales en paralelo. Por estas líneas el CPU envía la localidad de memoria en la cual va escribir o leer. El número de localidades que el CPU puede direccionar o acceder se determina por el número de líneas del bus de direcciones. Si el CPU tiene N líneas de dirección entonces puede direccionar $2^N$ localidades. Cuando el CPU lee o manda datos a o desde un puerto, la dirección del puerto también se envía por el bus de direcciones.

#### 1.2.1.2 Ducto de Datos

El ducto de datos consiste de 8,16, 32 o más líneas de señales en paralelo, estas líneas son *bidireccionales*. Esto significa que el CPU puede leer datos por estas líneas desde la memoria o un puerto, así también puede mandar datos a una localidad de memoria a un puerto. Muchos dispositivos en un sistema pueden tener sus salidas conectadas al ducto de datos, pero las salidas de solamente un dispositivo pueden estar habilitadas. Cualquier dispositivo conectado al ducto de datos debe ser de tres estados así estos dispositivos pueden estar flotados cuando no estén en uso.

#### 1.2.1.3 Ducto de Control

El ducto de control consiste de 4 a 10 líneas de señales en paralelo. El CPU manda señales sobre el ducto de control para habilitar las salidas de los dispositivos de memoria o puertos direccionados. Generalmente las señales del ducto de control son *leer memoria*, *escribir en memoria*, *leer E/S* y *escribir E/S*. Por ejemplo para leer un dato de un byte de una localidad de memoria, el CPU manda la dirección de la localidad de memoria deseada por el ducto de direcciones y después manda la señal de lectura de memoria por el ducto de control. La señal de lectura habilita al dispositivo de memoria direccionado para proporcionar el dato de un byte en el ducto de datos de donde es leído por el CPU.

### 1.2.2 Organización simplificada del CPU y la ALU

La principal unidad de funcionamiento de cualquier sistema de computadora es la unidad central de procesamiento (CPU). Las principales funciones del CPU de una microcomputadora son:

- Seleccionar, decodificar y ejecutar instrucciones de programa en el orden adecuado.
- Transferir datos hacia y desde la memoria, y hacia y desde las secciones de E/S.
- Responder a interrupciones externas.
- Proporcionar las señales de control y de tiempo necesarias para la totalidad del sistema.

Por lo general los CPU contienen como mínimo los elementos mostrados en la figura 1.4. Las secciones principales incluyen diversos registros, la unidad aritmética y lógica, el decodificador de instrucciones y la sección de temporización y control. La mayoría de los CPU realmente contienen más registros especiales, así como muchas entradas y salidas no detalladas en la figura 1.4.

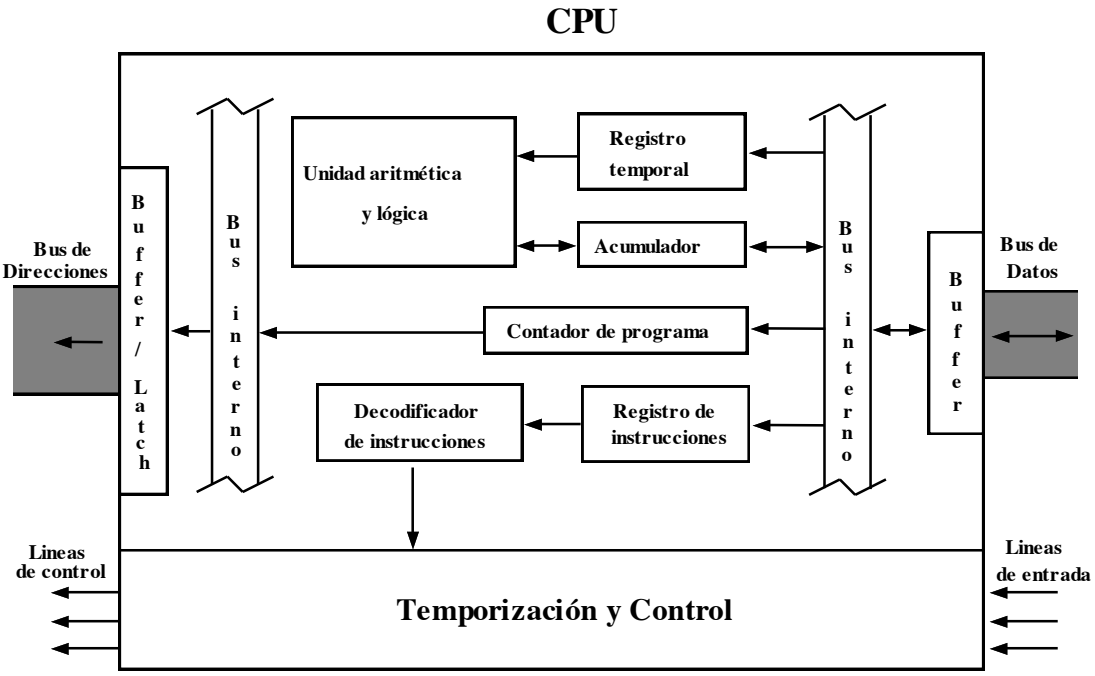



**Figura 1.4** Arquitectura simplificada del CPU.

La *unidad aritmética y lógica* (ALU) del CPU realiza operaciones tales como suma, corrimiento circular, comparación, incrementar, decrementar, negar, AND, OR, XOR, complemento, limpiar y preestablecer. Si la ALU fuera dirigida por medio de la instrucción para sumar, el procedimiento sería algo como lo que se vería en la figura siguiente:

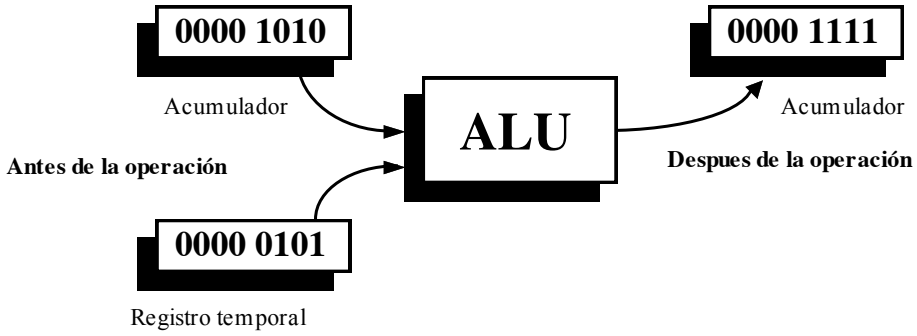



**Figura 1.5** Ejecución de la instrucción de sumar en la ALU.

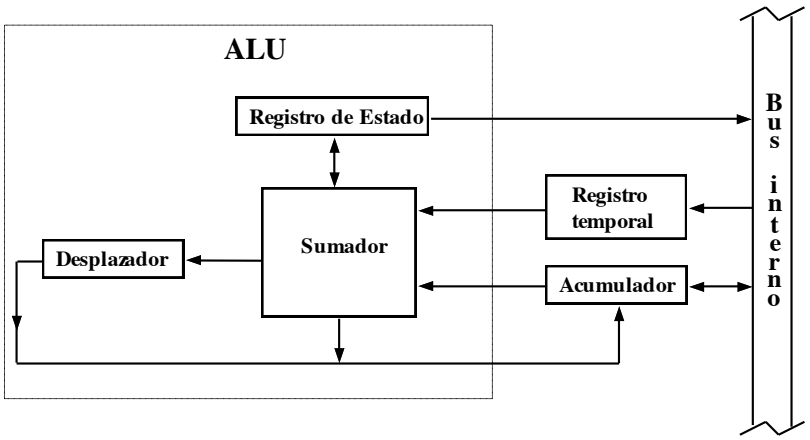



**Figura 1.6** Organización de la ALU.

Como se ve en la figura 1.5, el contenido del acumulador ($0A_{16}$) se suma al contenido del registro temporal ($05_{16}$) luego la suma ($0F_{16}$) se coloca de regreso en el acumulador. Esta operación puede ser extrapolada a la figura 1.6 la cual muestra un diagrama a bloques de una ALU genérica.

Los flip-flop individuales o banderas, incluyen indicadores de cero, resultado negativo, acarreo, etc. Las banderas sirven para la toma de decisiones cuando se utilizan instrucciones subsecuentes de bifurcación. Los registros temporales y de acumulador a menudo se consideran parte de la ALU.

Existen una diversidad de familias de microprocesadores en el mercado, para dar un ejemplo de esto a continuación se presenta una breve reseña histórica de la familia de microprocesadores de la compañía Intel Corporation, una de las cuales se encuentra fuertemente presente en el mercado actual y pionera en este ámbito.

## Familia de Microprocesadores de Intel

En 1968 se fundó la Intel Corporation con el fin de hacer pastillas de memorias. Al poco tiempo se acercó a Intel un fabricante de calculadoras que solicitó un CPU en una sola pastilla, y otro, que deseaba una pastilla controladora para terminales. Intel fabricó para estos clientes, las que fueron las primeras Unidades Centrales de Procesamiento en una sola pastilla: la 4004 de 4 bits y la 8008 de 8 bits.

Estos productos despertaron un gran interés, por lo que la compañía se dedicó a diseñar un CPU de propósito general, similar al 8008 de 16K de memoria máxima obteniendo como resultado en 1974 el 8080.

Dos años más tarde, en 1976, Intel lazó al mercado el procesador 8085 una variante del 8080 con componentes adicionales para algunas funciones de entrada/salida. Después vino el 8086, un verdadero CPU de 16 bits en una sola pastilla, similar al 8080, pero no compatible con éste.

Siguió el 8088, que tenía la misma arquitectura que su predecesor y podía ejecutar los mismos programas, pero con un ducto de 8 bits en vez de 16, que lo hacía más lento y más barato que el 8086.

Cuando IBM (*International Business Machines*) seleccionó al procesador 8088 como CPU para la IBM-PC original, esta pastilla se convirtió en el estándar de la industria de las computadoras personales.

| Nombre | Año | Longitud de Registros | Capacidad del Ducto de Datos | Espacio de Direcciones | Observaciones |
|---|---|---|---|---|---|
| 4004 | 1971 | 4 | 4 | 1K | Primer microprocesador en un IC |
| 8008 | 1972 | 8 | 8 | 16K | Primer microprocesador de 8 bits |
| 8080 | 1974 | 8 | 8 | 64K | Primer CPU de propósito general en un IC |
| 8085 | 1974 | 8 | 8 | 1M | 8080 embellecido |
| 8086 | 1978 | 16 | 16 | 1M | Primer CPU de 16 bits en un IC |
| 8088 | 1980 | 16 | 8 | 1M | Procesador usado por la IBM PC |
| 80186 | 1982 | 16 | 16 | 1M | 8086 + manejo de E/S en un IC |
| 80188 | 1982 | 16 | 16 | 1M | 8088 + manejo de E/S en un IC |
| 80286 | 1982 | 16 | 16 | 16M | Espacio de direcciones incrementado a 16M |
| 80386 | 1985 | 32 | 32 | 70T | Auténtico CPU de 32 bits en un IC |
| 80386SX | 1988 | 32 | 16 | 70T | 80386 con un ducto de 80286 |
| 80486 | 1989 | 32 | 32 | 70T | Versión más rápida del 80386 |
| PENTIUM | 1993 | 32 | 64 | 70T | Una procesador híbrido (CISC/RISC) |

**Tabla 1.1** Familia de microprocesadores Intel.

En los años siguientes, Intel fabricó el 80186 y el 80188, que fueron en esencia nuevas versiones del 8086 y 8088 respectivamente, pero con una gran cantidad de circuitos de entrada/salida. Su uso nunca se propagó.

Intel diseñó una versión mejorada del 8086: el microprocesador 80286, cuyo conjunto de instrucciones es prácticamente igual al de los anteriores, pero la organización de la memoria era muy diferente y difícil de manejar debido al requerimiento de compatibilidad con las versiones previas. El 80286 fue utilizado por IBM en su PC/AT y en el modelo mediano PS/2 y al igual que el 8088 tuvo gran éxito.

El siguiente microprocesador fue uno de 32 bits en una sola pastilla, el 80386, que era más o menos compatible con las versiones previas hasta el 8088. Este procesador, igual que el 80286, está siendo muy utilizado.

El 80386SX es una versión especial del 80386, diseñado para conectarse en el soporte del 80286 y permitir la actualización de las máquinas basadas en el 80286. El 80486 es la versión mejorada del 80386, totalmente compatible, es decir, todos los programas del 80386 correrán en el 80486 sin modificación. La primera diferencia entre ambos es la presencia de un co-procesador de punto flotante, un controlador de memoria y 8K de memoria caché en la misma pastilla del 80486. Además es dos a cuatro veces más rápido que el 80386 y está mejor adaptado para sistemas de multiproceso.

En marzo de 1993 Intel lanzó al mercado el procesador PENTIUM que es compatible con sus antecesores, posee un ducto de datos de 64 bits y uno de direcciones de 32 bits. Este microprocesador contiene 3.1 millones de transistores, trabaja a 66MHz (existen versiones de 75 MHz, 90MHz y hasta 166MHz) y es 150 a 200 veces más rápido que el 8086.

Este microprocesador tiene dos unidades de aritmética y lógica (ALU), una unidad de punto flotante FPU (*Floating Point Unit*) y circuitería dedicada para las operaciones de suma, resta, multiplicación y división, además posee 2 bloques independientes memoria caché de 8 Kbytes.

La evolución en la línea de los CPU de Intel refleja en su conjunto los avances de la industria de la computación. En una década y media se ha pasado de un CPU de 4 bits a uno de 32 bits. Por otro lado, el 8086 contenía 30,000 transistores mientras que el PENTIUM tiene más de 3 millones de transistores.

### 1.2.3 Organización simplificada de la sección de E/S

Una operación de entrada o salida es el acto de transferir datos a o desde un dispositivo periférico seleccionado. El microprocesador es el foco de todas las operaciones, por tanto una entrada significa que el dato fluye hacia el microprocesador mientras que una salida significa que el dato fluye del microprocesador.

Generalmente un microprocesador utiliza instrucciones tales como IN y OUT para transferir datos a y desde los puertos de entrada/salida. La instrucción de salida se representa por el mnemotécnico OUT en los programas en lenguaje ensamblador, mientras que la instrucción de entrada utiliza el mnemotécnico IN. Cuando se utiliza la operación OUT, se utiliza una señal especial de *escritura de entrada/salida* $(\overline{\mathrm{I/OW}})$, la operación IN también requiere el uso de una señal especial llamada *lectura de entrada/salida* $(\overline{\mathrm{I/OR}})$. Ambas señales de salida generalmente se activan en bajo y pertenecen al ducto de control.

#### 1.2.3.1 Decodificador de Direcciones

El decodificador de direcciones es una parte del control lógico. Este genera señales para seleccionar dispositivos cuando una cierta dirección (o rango de direcciones) se presenta en el ducto de direcciones. Por ejemplo la figura 1.7 muestra un decodificador para la dirección 3000H (0011 0000 0000 0000 binario). La salida del decodificador es verdadera (lógica 0) solamente cuando la dirección exacta se presenta en el ducto de direcciones. Esta salida puede ser utilizada para habilitar un puerto que tenga asignada la dirección 3000H.

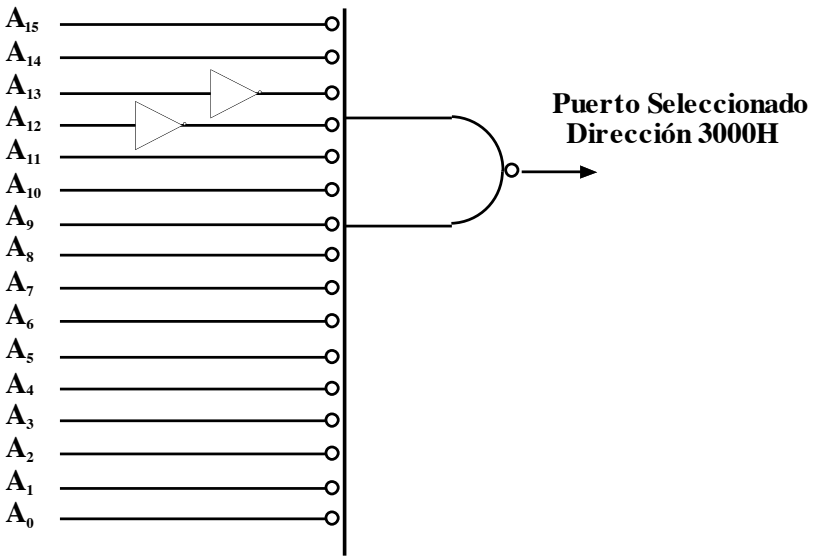



**Figura 1.7** Decodificador de direcciones (dirección 3000H).

### 1.2.3.2 Puertos de Salida

La figura 1.8 muestra un puerto de salida con la asignación de dirección 3000H. El latch o retenedor es activado cuando la dirección 3000H esta presenta en el ducto de direcciones y además una activación de las señal de control ($\overline{I/OW}$) ocurre. Una vez activado el latch, el dato del ducto de datos es almacenado en él. Por medio de este sistema el microprocesador puede causar que un dato especificado por el programa aparezca en las salidas del latch.

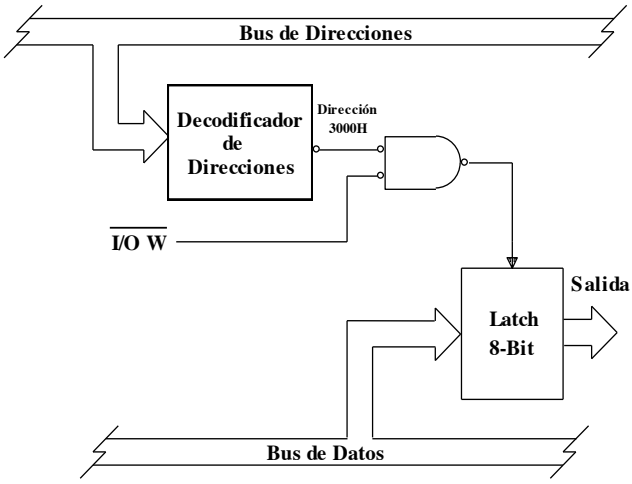



**Figura 1.8** Puerto de salida decodificado en la dirección 3000H.

### 1.2.3.3 Puertos de Entrada

Los puertos de entrada son conectados de manera similar a la figura 1.9. A la salida del decodificador de direcciones se le aplica la operación AND con una señal de control ($\overline{I/OR}$) para generar la habilitación del puerto. El puerto de entrada es un latch con salida de tres estados las cuales solo se activan cuando es habilitado el latch. De esta manera el microprocesador puede leer un dato de la línea de salida del latch por medio de una operación de lectura a la dirección apropiada. Entonces el microprocesador puede almacenar el dato en algún registro interno.

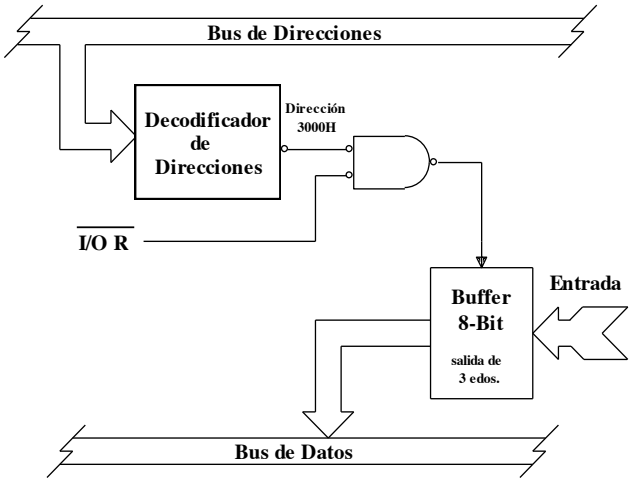



**Figura 1.9** Puerto de entrada decodificado en la dirección 3000H.

### 1.2.3.4 Decodificadores de Dirección Para Varios Dispositivos

Supóngase que se requiere de un decodificador de direcciones para controlar ocho puertos de E/S en vez de solo uno. Ocho decodificadores similares al de la figura 1.7 podían utilizarse, pero existe un método más sencillo.

La figura 1.10 muestra un decodificador de direcciones el cual genera señales de selección para las direcciones 3000, 3001,... ,3007. Para esas ocho direcciones, solamente cambian tres bits de orden más bajo ($A_0$, $A_1$ y $A_3$) de los 16 bits que forman la dirección. Los trece bits de orden más alto pueden decodificarse mediante un circuito similar al de la figura 1.7. La salida de este circuito se utiliza como habilitador de un decodificador como el 74LS138. Este decodificador entonces puede generar ocho salidas separadas, una para cada posible combinación de $A_0$, $A_1$ y $A_2$. El decodificador es deshabilitado (todas las salidas falsas) si los trece bits de mayor orden no son del valor adecuado.

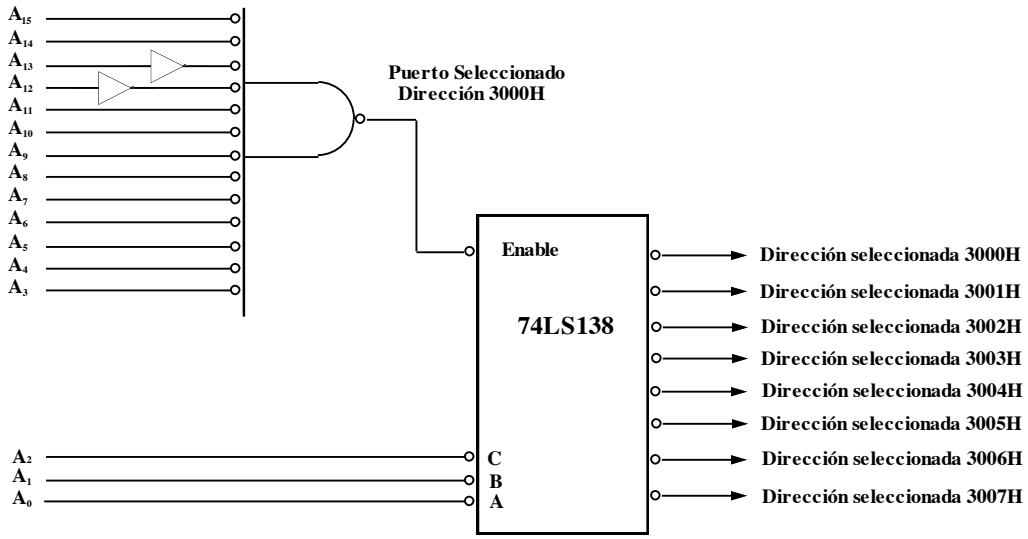



**Figura 1.10** Decodificador múltiple utilizando un IC 74LS138.

## 1.2.4. Organización simplificada de la Memoria

Los sistemas basados en microprocesadores generalmente utilizan circuitos integrados de memoria para almacenar programas y datos. El más simple dispositivo de memoria es el flip-flop, el cual almacena un bit de información. Escribir o leer en una posición de memoria se denomina *acceder* a la memoria. En una *memoria de acceso aleatorio*, cualquier posición de memoria puede ser escrita o leída en un tiempo determinado llamado *tiempo de acceso*.

En una microcomputadora el microprocesador tiene comunicación con la memoria mediante el ducto de direcciones, las líneas de ducto de direcciones pueden generar combinaciones diferentes de ceros y unos. Algunas combinaciones se muestran bajo el ducto de direcciones en la figura 1.11. La representación de las direcciones en forma hexadecimal es más eficiente que la representación binaria o decimal, esto debido a que con menor número de dígitos se puede representar cantidades mayores. Obsérvese el uso de la letra H para designar la representación hexadecimal.

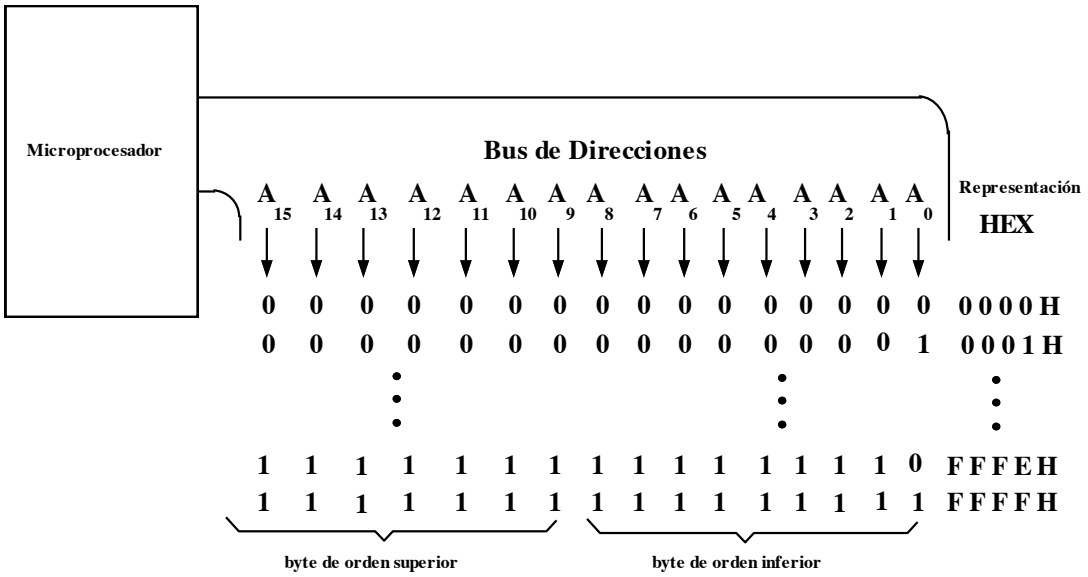



**Figura 1.11** Representación de las direcciones del microprocesador.

La figura 1.12 muestra la arquitectura interna básica de una memoria del tipo SRAM (*Static Random Access Memory*), en ella puede observarse que consta de un arreglo matricial de flip-flops en los que se retiene la información (bits) y un decodificador el cual convierte la dirección binaria a una posición de memoria la cual está constituida por un renglón de la matriz de flip-flops.

La figura 1.13 muestra una memoria RAM de 1K × 8. Esta RAM contiene 1024 localidades de 8 bits cada una. Las líneas de datos son bidireccionales dado que el dato puede salir o entrar a la RAM.

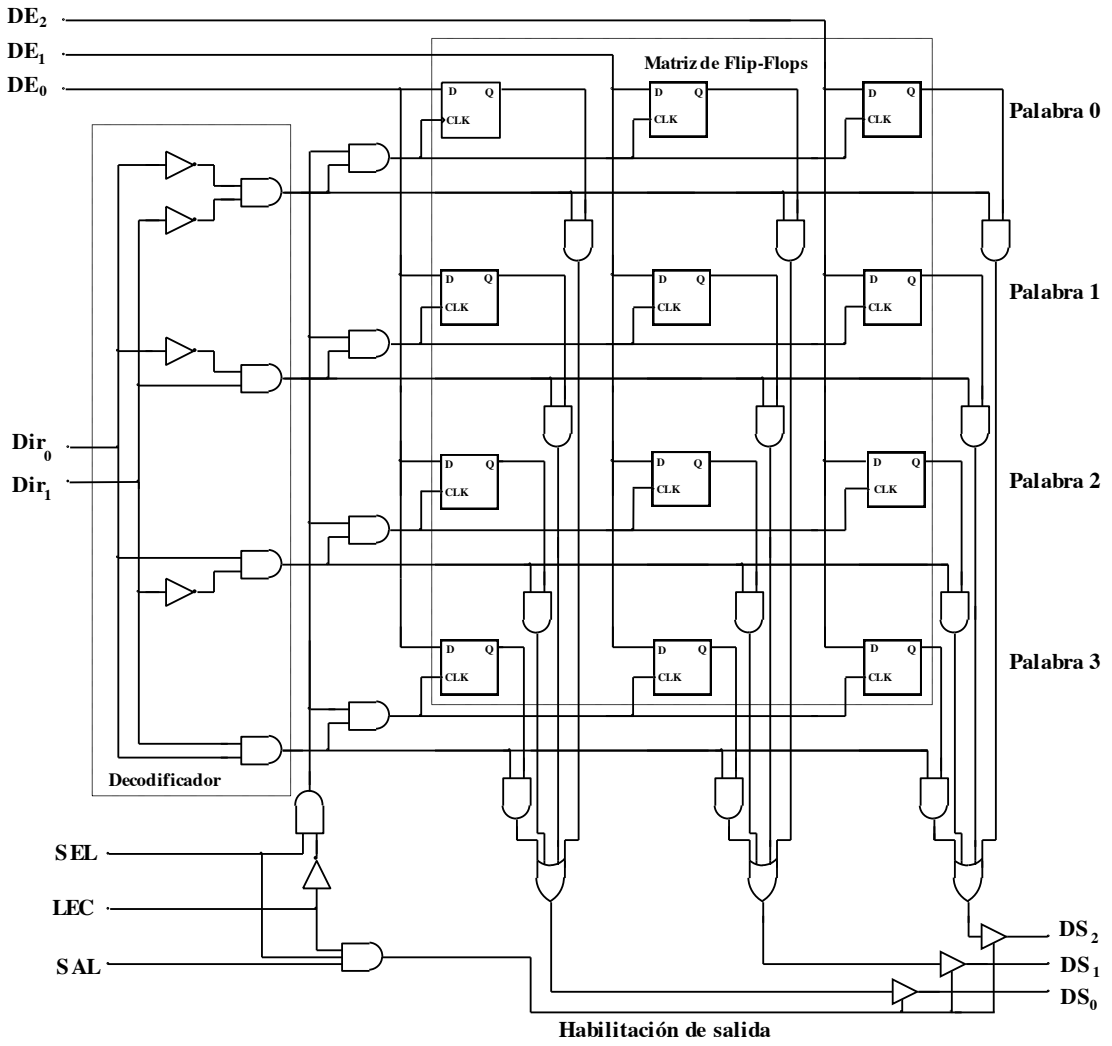



**Figura 1.11** Arquitectura interna básica de una SRAM.

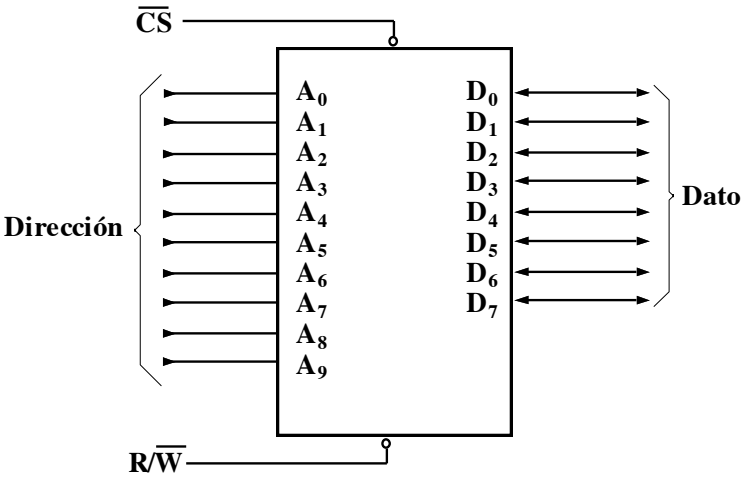



**Figura 1.12** Configuración de una RAM de 1K × 8 bits o 1K bytes.

## 1.3 MICROCOMPUTADORAS EN UN SOLO CIRCUITO INTEGRADO (IC)

En base a los objetivos anteriores se define una computadora de un solo IC, del cual los microcontroladores son un subconjunto, una computadora de un solo IC es un circuito integrado que contiene los elementos esenciales de una computadora los cuales son: sección de entrada y salida (E/S), un CPU (el cual contiene una ALU) y memoria.

Un ejemplo de la primera computadora de un solo IC, es el 8048 de Intel el cual fue introducido al mercado en 1976. La estructura interna del IC 8048 se muestra en la figura 1.13. Esta familia de computadoras de un solo IC tiene el nombre de MCS-48 y existe una variedad de modelos que difieren solo en cantidad de memoria y capacidad de E/S. Esta computadora opera con una sola fuente de poder de 5 volts. La principal diferencia de los diferentes modelos de la familia es el tipo de almacenamiento del programa dentro de la pastilla. La tabla 1.2 muestra los miembros más importantes de esta familia.

| DISPOSITIVO | MEMORIA (BYTES) | | | PUERTOS E/S |
|---|---|---|---|---|
| | ROM | EPROM | RAM | |
| 8035 | 0 | 0 | 64 | 3×8 |
| 8048 | 1024 (1K) | 0 | 64 | 3×8 |
| 8748 | 0 | 1024 (1K) | 64 | 3×8 |
| 8039 | 0 | 0 | 128 | 3×8 |
| 8049 | 2048 (2K) | 0 | 128 | 3×8 |
| 8749 | 0 | 2048 (2K) | 128 | 3×8 |
| 8040 | 0 | 0 | 128 | 3×8 |
| 8050 | 4096 (4K) | 0 | 256 | 3×8 |

**Tabla 1.2** Familia de computadoras de un solo IC MCS-48.

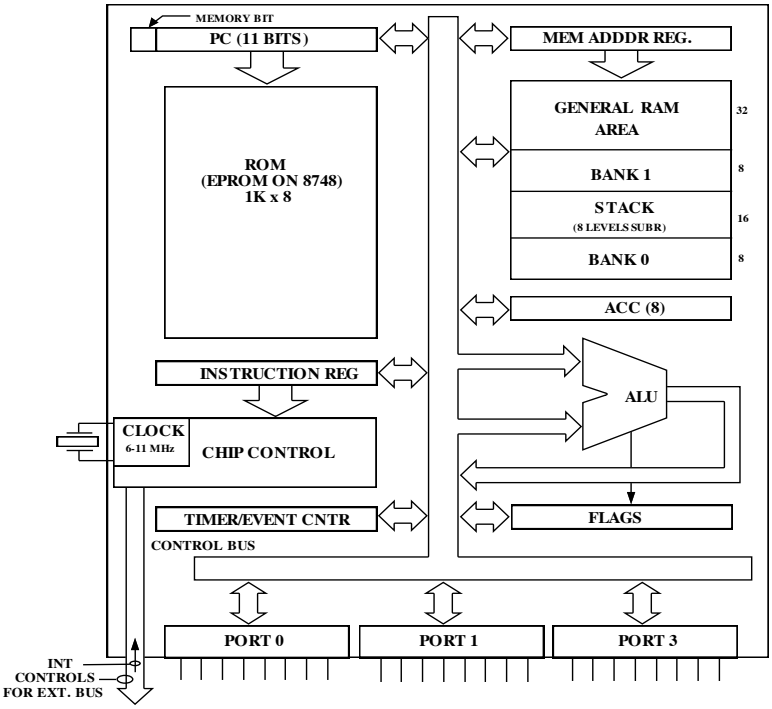



**Figura 1.13** Arquitectura interna de la familia MCS-48.

## 1.4 MICROCONTROLADORES: MICROCOMPUTADORAS UN SOLO IC ORIENTADAS A E/S

Los microcontroladores son computadoras digitales, diseñadas particularmente, para supervisar, manejar, monitorear y controlar varios procesos en la industria, negocios, aeronáutica, y otras áreas de aplicación. Con la llegada de la tecnología VLSI, los microcontroladores se convirtieron en microcomputadoras contenidas en un solo integrado. En algunos microcontrolador además de una Unidad Central de Procesamiento (CPU) contienen memoria *cache*, memoria principal, Interfaz de Entrada/Salida (E/S), Controlador de Acceso Directo a Memoria (DMA), Manejador de Interrupciones, Temporizadores y otros subsistemas necesarios para llevar a cabo un eficiente sistema basado en microcontrolador.

En un sistema basado en microprocesador, cada elemento por lo general, se encuentra en un encapsulado aparte del microprocesador. En un microcontrolador, todos los elementos del sistema se diseñan en la misma pieza de silicio. Entonces, un microcontrolador representa un sistema de microprocesador incrustado en un mismo encapsulado. En muchos casos, lo único que se necesita es un elemento de temporizado, por ejemplo un cristal de cuarzo para hacerlo un sistema funcional.

Un microcontrolador comparado con un sistema basado en microprocesador tiene un número limitado de recursos. Un sistema basado en microprocesador, típico, puede llegar a tener 64K bytes o más, de RAM o ROM y varios puertos de E/S en su diseño. Un microcontrolador tiene entre 1 y 4K bytes dedicados a EPROM/ROM, y solamente de 64 a 512 bytes de RAM, y de tres a cuatro puertos de E/S de 8 bit de ancho.

Generalmente un microprocesador cuenta con un espacio grande de memoria, dentro de este espacio de memoria se pueden asignar libremente áreas para ROM y RAM, el conjunto de instrucciones del microprocesador no hace distinción entre estos dos tipos de memoria. Un microcontrolador, sin embargo, tiene dos espacios de memoria, uno para almacenar las instrucciones del programa y otro para almacenar los datos. Esta separación de las memorias de programa y de datos es una de las más grandes diferencias en la arquitectura de los microcontroladores y microprocesadores.

Otra de las diferencias se encuentra en el conjunto de instrucciones. Generalmente, los microprocesadores tienen un conjunto óptimo de instrucciones para mover información de una localidad a otra, y llevar a cabo una serie de manipulaciones y cálculos de datos. Su conjunto de instrucciones proporciona una capacidad limitada en el manejo de los puertos de E/S. En esta área, puede considerarse que los microprocesadores son no eficientes en tareas como; cambiar el nivel lógico de un bit de un puerto de E/S. Por otro lado, los microcontroladores son relativamente malos en el manejo de memoria, sobre todo cuando se trata de bloques grandes de memoria. Sin embargo, su conjunto de instrucciones esta optimizado en dirección a la manipulación de información de E/S.

### 1.4.1 Unidad de E/S en Microcontroladores

Una de las diferencias drásticas en la arquitectura de los microcontroladores y microprocesadores es el método de acceder los puertos de entrada/salida. Generalmente los microprocesadores utilizan alguno de los siguientes enfoques para acceso a E/S, los cuales son: *Mapeo de sección de E/S sobre sección de memoria y espacios separados para memoria y E/S*. Estos ejemplificados por la compañía Motorola e Intel respectivamente.

#### 1.4.1.1 Mapeo de sección de E/S sobre sección de Memoria

Este enfoque considera todas las direcciones de E/S como direcciones de memoria. La lectura/escritura a un dispositivo de E/S se realiza como si dicho dispositivo fuese una localidad más de memoria. Esta técnica tiene la ventaja de que todas las operaciones matemáticas y lógicas que se pueden realizar sobre alguna localidad de memoria también es posible hacerlas sobre un puerto de E/S ya que este es una localidad de memoria. Por ejemplo el contenido de un puerto de E/S puede leerse, incrementarse y escribirse de al puerto, esto en una sola instrucción. También un dato puede ser transferido de un puerto a otro con una sola instrucción de movimiento, otra operación posible es transferir un bloque de memoria de una sección de memoria a un puerto con pocas instrucciones. La figura 1.14 muestra un ejemplo de un mapa de direcciones que utiliza este enfoque.

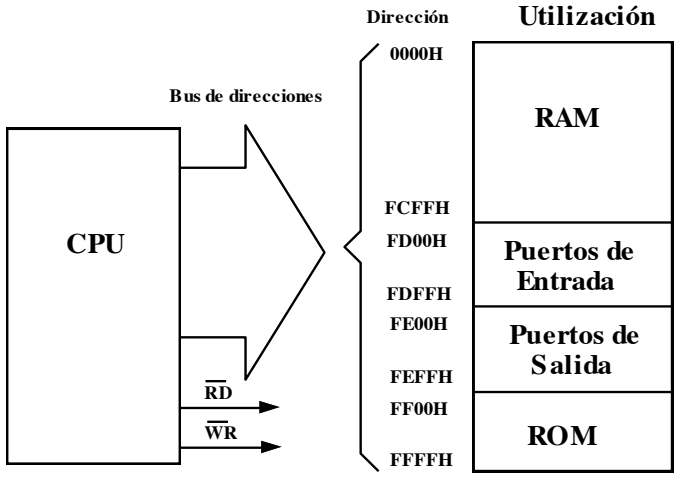



**Figura 1.14** Espacio de E/S mapeado sobre el espacio de memoria.

#### 1.4.1.2 Espacios separados para Memoria y E/S

En esta técnica se manejan espacios separados para memoria y para sección de E/S, esto es, el CPU genera direcciones que pueden ser para la sección de E/S o para la sección de memoria, ahora bien ¿cómo es posible entonces diferenciar que dirección le pertenece a memoria y cual a E/S? La respuesta es una línea extra en el ducto de control la cual se denomina $IO/\overline{M}$ y que es puesta a 0 lógico por el CPU cuando se hace referencia a una dirección de memoria y llevada a 1 lógico si es de E/S.

En esta técnica tiene una ventaja sobre la anterior, en ella es la posible tener mayor espacio de memoria y de E/S puesto que no es un solo espacio compartido. Sin embargo la desventaja es que las instrucciones para acceder a puertos son distintas a las de acceso a memoria y generalmente las instrucciones de acceso a puertos solo de limitan a entradas y salidas (no permiten hacer operaciones lógicas o aritméticas sobre ellos). Otra desventaja es la necesidad de circuitería adicional para la lógica de control de escritura/lectura para los distintos espacios de direcciones. La figura 1.15 muestra una ejemplificación de esta técnica.

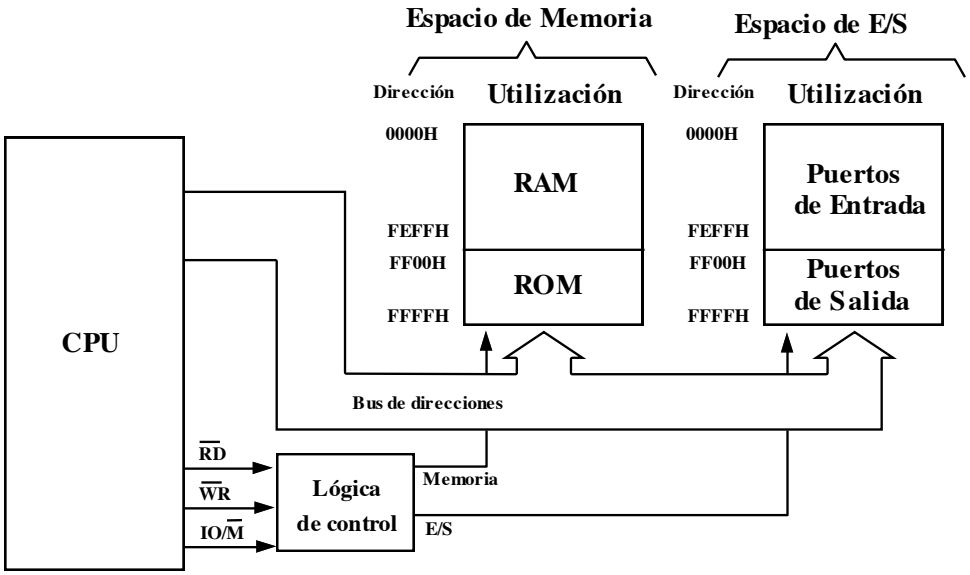



**Figura 1.15** Espacios diferentes para sección E/S y memoria.

## 1.4.2 Interrupciones

Una interrupción es un acontecimiento de una condición o evento que causa una suspensión de un programa mientras la condición es atendida por otro programa. Las interrupciones juegan un papel importante en el diseño e implementación de aplicaciones de microcontroladores. Además permiten a un sistema una respuesta asíncrona a un evento y maneja dicho evento mientras otro programa se ejecuta. Un *sistema manejador de interrupciones* da la ilusión de hacer varias cosas simultáneamente. Por supuesto, el CPU no puede ejecutar más de una instrucción en un mismo tiempo; para ello suspende temporalmente la ejecución de un programa, para ejecutar otro, entonces regresa al primer programa. De esta manera, esto es parecido a una subrutina o procedimiento. El CPU ejecuta otro programa o subrutina y entonces regresa al programa original. La diferencia es que en un *interrupt-driven system*, la interrupción no ocurre como resultado de una instrucción, si no como respuesta a un "evento", esto sucede de forma asíncrona al programa principal. No se sabe cuándo será interrumpido el programa principal.

El programa que trata con una interrupción se le llama rutina de servicio de interrupción (ISR *Interrupt Service Routine*).

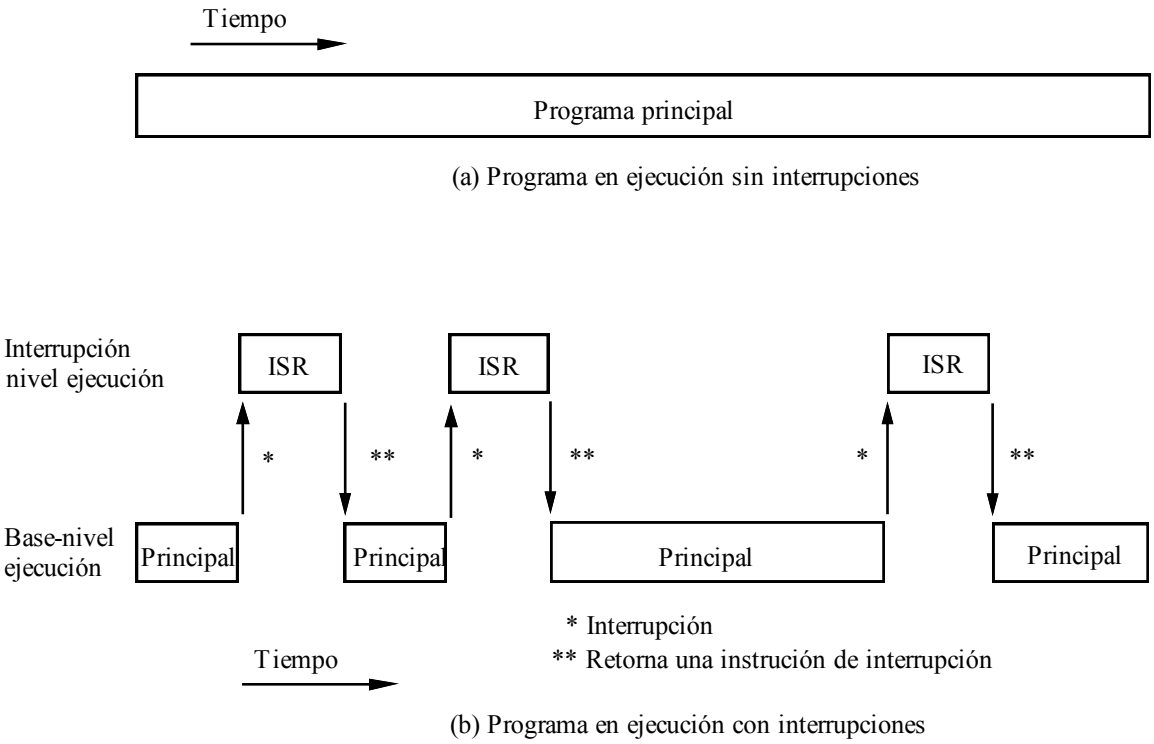



**Figura 1.16** Programa ejecución con y sin interrupciones.
(a) Sin interrupciones (b) Con interrupciones

### 1.4.3 ALU

La diferencia significativa entre la ALU de un microcontrolador y la una computadoras de propósito general son las instrucciones para manipulación de bits. Todas la computadoras poseen instrucciones lógicas generales que pueden ser utilizadas para realizar esa misma manipulación de bits pero no con la facilidad que poseen los microcontroladores. Por ejemplo, si se desea verificar el estado de un bit particular de un puerto, es una computadora es necesario realizar las siguientes instrucciones:

1) Leer el puerto a un registro.
2) Aplicar una operación AND del registro con una máscara consistente de ceros excepto para la posición del bit a verificar.
3) Realizar una instrucción de salto si el resultado es cero.

En cambio un microcontrolador posee instrucciones que permiten realizar saltos en base a la verificación de un bit particular de un puerto de entrada, es decir realiza la misma tarea de una forma directa, solo necesita una instrucción.

### 1.4.4 Temporizadores

Un temporizador es una serie de flip-flops que recibe una señal de entrada una fuente de reloj. El reloj se aplica al primer flip-flop, el cual divide la frecuencia en 2. La salida del primer flip-flop dispara al segundo flip-flop, el cual también divide en 2 su entrada, y así sucesivamente. Desde que cada sucesión de bloques o periodos la frecuencia inicial es dividida en 2, por tanto un temporizador con *n* bloques o periodos divide la frecuencia de reloj en $2^n$. La salida del último periodo marca un tiempo sobre flujo, o **bandera**, el cual es verificado mediante software o puede generar una interrupción. El valor binario del conjunto de flip-flops puede ser considerado como un "contador" del número de pulsos ("eventos") desde que el temporizador inició. Por ejemplo un temporizador de 16-bit podría contar de 0000H hasta FFFFH. La bandera de sobre flujo seria en la transición de FFFFH a 0000H del contador.

La operación de un temporizador se ilustra en la figura 1.17. Cada bloque se muestra como un flip-flop tipo-D (con disparo en flanco negativo) operando como divisor.

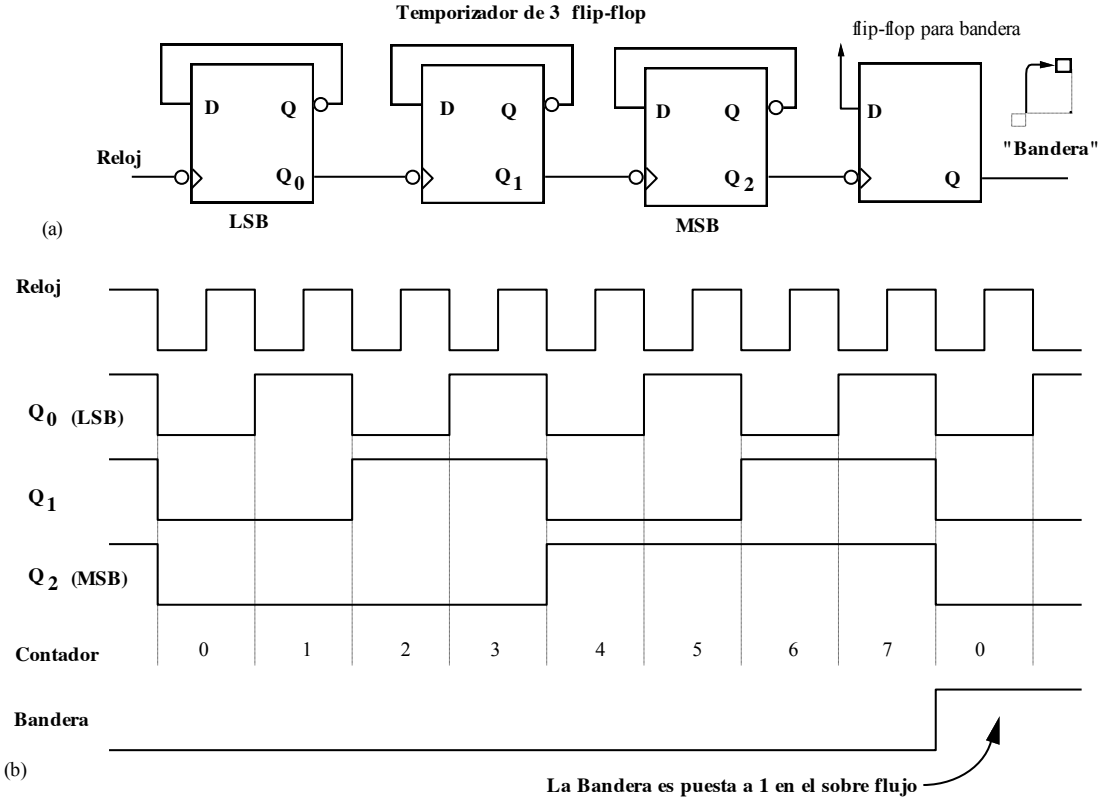



**Figura 1.17** Temporizador de 3 bits (a) Diagrama esquemático (b) Diagrama de tiempo.

### 1.4.5 Entrada/Salida en paralelo y serie

La comunicación serie y paralelo está disponible en la mayoría de microcontroladores en menor y mayor grado. El objetivo de esto es que dichos dispositivos puedan comunicarse con otros dispositivos similares ya sea en forma paralela o serie según las necesidades.

Dentro de las computadoras, microprocesadores y microcontroladores la transferencia de información es en forma paralela, debido a que es más veloz hacerla así. Sin embargo, la transmisión de datos a través de una gran distancia, cuando se hace en forma paralela se requiere gran cantidad de cable. Por tanto, cuando se mandan datos en una distancia considerable generalmente se convierten de forma paralelo a forma serie, para así mandar la información por uno o dos cables. Los datos serie recibidos son transformados a forma paralela ya que son más fácil el manejo de ellos en el CPU o microcontrolador.

En la literatura de los sistemas de comunicación serie se encuentran tres términos frecuentemente utilizados, que son: comunicación serie tipo *simplex*, *half-duplex* y *full-duplex*. Una línea de datos tipo simplex puede transmitir solamente en una sola dirección. La transmisión tipo half duplex significa que puede transmitir y recibir información por la misma línea, pero no simultáneamente, esto es por un tiempo transmite y luego conmuta para recibir información. El termino full duplex significa que cada sistema puede mandar y recibir información al mismo tiempo y necesariamente requiere dos cables, uno para transmisión y otro para recepción. La figura 1.18 muestra un esquema de estos términos.

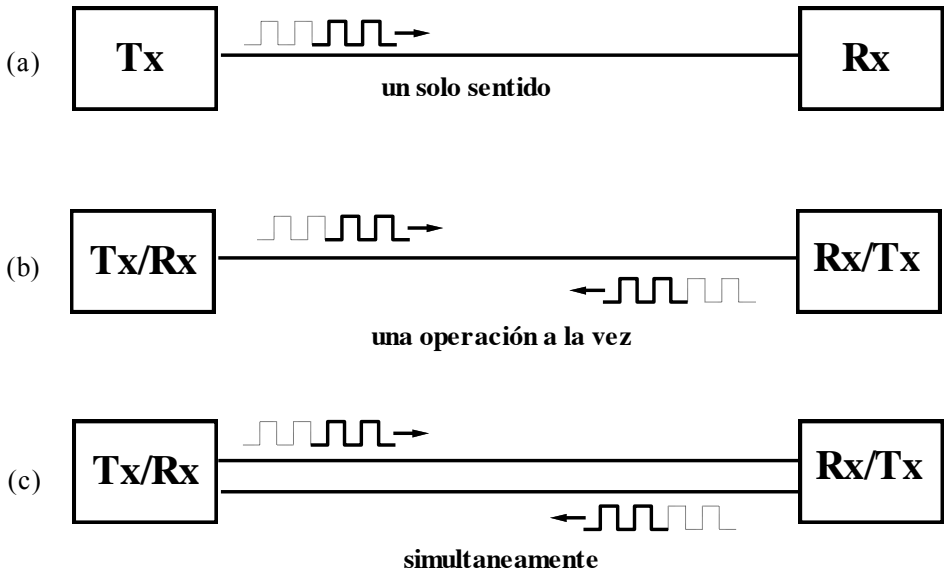



**Figura 1.18** Tipos de comunicación serie, (a) simplex b) haft-duplex y c) full-duplex.

### 1.4.6 Configuraciones de Microcontroladores

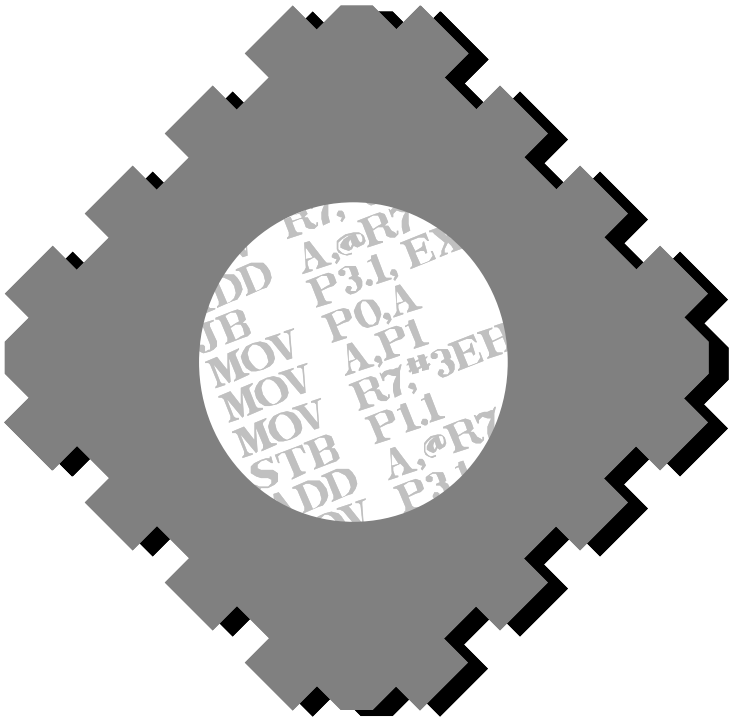
La tabla 1.3 lista algunas de las capacidades de varios microcontroladores de las dos compañías más difundidas en la industria.

| IC | RAM | ROM/EPROM EEPROM | Clock μs | I/O Ports | A/D | Timers |
|---|---|---|---|---|---|---|
| M68HC11A0 | 256 | - | 0.476 | 4×8 1×6 | 4/8 | 9 |
| M68HC11A1 | 256 | 512 EEPROM | 0.476 | 4×8 1×6 | 4/8 | 9 |
| M68HC11A2 | | 2048 EEPROM | 0.476 | 4×8 1×6 | 4/8 | 9 |
| M68HC11A8 | 256 | 8K ROM, 512 EEPROM | 0.476 | 4×8 1×6 | 4/8 | 9 |
| M68HC11E0 | 512 | - | 0.476 | 4×8 1×6 | 8 | 9 |
| M68HC11E1 | 512 | 512 EEPROM | 0.476 | 4×8 1×6 | 8 | 9 |
| M68HC11E2 | 256 | 2048 (EE) | 0.476 | 4×8 1×6 | 4/8 | 9 |
| M68HC11E9 | 512 | 12K (ROM), 512 (EE) | 0.476 | 4×8 1×6 | 8 | 9 |
| M68HC11D3 | 192 | 4096 (ROM) | 0.476 | 4×8 1×6 | 8 | 9 |
| M68HC11F1 | 1024 | 512 (EEPROM) | 0.476 | 4×8 1×6 | 8 | 9 |
| Intel 8021 | 64 | 1024 (ROM) | 2.5 | 2×8 1×6 | - | 2 |
| Intel 8022 | 64 | 2048 (ROM) | 2.5 | 3×8 | - | 2 |
| Intel 8035 | 64 | - | 2.5 | 3×8 | - | 2 |
| Intel 8039 | 128 | - | 1.4 | 3×8 | - | 2 |
| Intel 8041 | 64 | 1024 (ROM) | 2.5 | 3×8 | - | 2 |
| Intel 8048 | 64 | 1024 (ROM) | 2.5 | 3×8 | - | 2 |
| Intel 8049 | 128 | 2048 (ROM) | 1.4 | 3×8 | - | 2 |
| Intel 8748 | 64 | 1024 (EPROM) | 2.5 | 3×8 | - | 2 |
| Intel 8031 | 128 | - | 1 | 4×8 | - | 2 |
| Intel 8051 | 128 | 4096 (ROM) | 1 | 4×8 | - | 2 |
| Intel 8751 | 128 | 4096 (EPROM) | 1 | 4×8 | - | 2 |

**Tabla 1.4** Capacidades de microcontroladores de 8 bits de dos familias.

# Sección 2

## Implementación de Programas en Microprocesadores y Microcontroladores



## INTRODUCCIÓN

En esta unidad se pesenta un panorama general del desarrollo de programas en alto y bajo nivel, así como ventajas y desventajas que conlleva trabajar en cada uno de estos niveles. Para iniciar esto es necesario tener la definición de algunos términos comúnmente utilizados en el desarrollo se programas para computadoras o microcontroladores.

Los programas que convierten un programa escrito en un lenguaje a otro lenguaje distinto se le llama *traductor*. Al lenguaje en que esta escrito el programa original se le llama *lenguaje fuente* y al que se convierte se la denomina *lenguaje objeto*. Ambos lenguajes, el fuente y el objeto definen niveles. Si se contara con un procesador que ejecutara directamente programas escritos en lenguaje fuente no seria necesario traducirlo a un lenguaje objeto.

La traducción se utiliza cuando se tiene un procesador para un lenguaje objeto y no se cuenta con uno para el lenguaje fuente. Es por ello que se realizan en lenguaje fuente y luego son traducidos para poder ejecutarlos. Los programas tanto el fuente como el objeto desde el punto de vista funcional son equivalentes, esto es producen los mismos resultados.

## 2.1 PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR

Los traductores se pueden dividir en dos grupos, según la relación que existe entre el lenguaje fuente y el objeto. Cuando el lenguaje fuente solamente muestra una representación simbólica de un lenguaje numérico, el traductor se denomina *ensamblador* y al lenguaje fuente se le conoce como *lenguaje ensamblador*. Sin embargo cuando el lenguaje fuente es un lenguaje mucho más complejo que el lenguaje objeto, es decir cuando una instrucción en el lenguaje fuente es traducida a varias instrucciones de lenguaje objeto de le denomina *compilador*.

Un lenguaje ensamblador puro es aquel que cada sentencia produce exactamente una instrucción máquina. Una razón para utilizar lenguaje ensamblador en lugar de programar en lenguaje numérico o *Lenguaje maquina*[5], es que es mucho más fácil programar en ensamblador. Es una gran diferencia utilizar nombres (mnemónicos) para instrucciones que utilizar únicamente números en alguna base (decimal, hexadecimal o binario). Esto es porque podemos recordar mas fácilmente abreviaturas para ciertas operaciones que números asignados a dichas operaciones, por ejemplo: es mas sencillo recordar que la instrucción ADD se utiliza para sumar, que recordar que el número 24 hace la misma operación. De esta manera el programador sólo necesita recordar los nombres simbólicos, ya que el ensamblador los traduce a las instrucciones numéricas. Esto mismo tambien es aplicado a las direcciones de memoria, es decir el programador puede dar nombres a las localidades de memoria.

[5] Lenguaje máquina se le denomina al lenguaje nato del procesaodor.

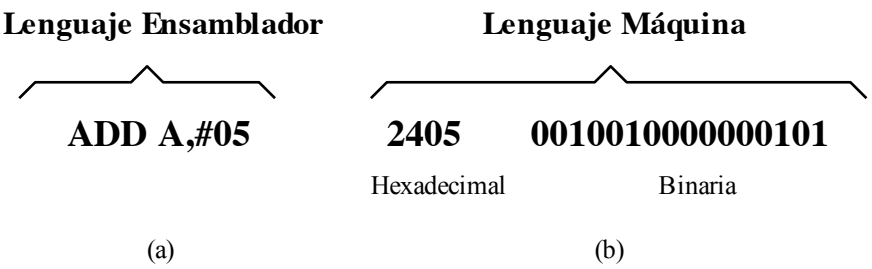



**Figura 2.1** Instrucción para suma. a) leguaje ensamblador b) lenguaje máquina.

La figura 2.1 muestra los distintos formatos de la instrucción para sumar el acumulador y el numero 5. Como puede observarse, el lenguaje ensamblador tiene una gran ventaja sobre el lenguaje máquina, esto debido al manejo de mnemónicos (símbolos) que son equivalentes en cuanto a funcionalidad con los conjuntos de números ya sea en forma binaria o hexadecimal.

## 2.2 LENGUAJES DE ALTO NIVEL

El concepto de lenguaje de alto nivel se define como un lenguaje orientado a la solución de problemas, ademas están diseñados para permitir al programador concentrarse en la lógica del problema a resolver, liberándolo de la necesidad de un conocimiento profundo de lenguaje máquina de la computadora. El termino alto nivel significa que están orientados hacia la gente, en contraste con el lenguaje ensamblador de bajo nivel el cual esta orientado hacia la máquina.

Las características que identifican un lenguaje de alto nivel son:

- **Utiliza una notación especial orientada al problema a resolver.**

  Las expresiones de las instrucciones son congruentes con el tipo de problema. Con ensamblador para resolver una formula matemática puede tener la siguiente forma:

```
MOV A,X
MOV B,Y
MUL AB
SUB A,Z
MOV R,Z
```

  En un lenguaje de alto nivel , la instrucción para resolver el mismo problema puede expresarse como:

```
R=Z-X*Y
```

- **No se requiere conocimiento del código máquina.**

  El programador no necesita tener conocimientos del lenguaje máquina o de los códigos que la computadora maneja internamente. Esta facilidad ha permitido que una persona aprenda a programar y usar un lenguaje de programación de alto nivel prácticamente con cualquier computadora, ya que éste se independiza de las características de la máquina. Sin embargo siempre es conveniente conocer la arquitectura de la computadora y la forma en que internamente maneja la información, para así poder obtener el máximo provecho de su capacidad.

- **Facilidad para procesar el programa en otra computadoras (portabilidad).**

  El concepto de independencia del lenguaje con respecto a la computadora significa que un programa escrito para procesarse en una máquina puede ser ejecutado en otro sin tener problemas. Esto tiene una validez relativa, puesto que es necesario compilar el programa para esa nueva máquina, utilizando un compilador para dicha máquina. Generalmente es necesario hacer algunas modificaciones en el código fuente.

- **Expansión de las instrucciones.**

  La facilidad de programar con lenguaje de alto nivel se debe, en buena parte a que no es necesario escribir una docena de instrucciones cuando se puede ordenar lo mismo con una sola expresión. Sin embargo la máquina no puede procesar este tipo de instrucciones con tal grado de complejidad y por tal motivo tiene que expandir cada instrucción a una serie de instrucciones en lenguaje máquina.

**Ventajas**

- Facilidad de aprendizaje.
- Facilidad de utilizarlo.
- Facilidad para documentación y el mantenimiento.
- Facilidad de depuración.
- Facilidad para transportarlo a otra computadora o microcontrolador.
- Reducción del tiempo total para el desarrollo de programas.

**Desventajas**

- Tiempo necesario para la compilación.
- No siempre se tiene el programa objeto en forma optima.

Existen tres requerimientos suficientes para calificar a un lenguaje de alto nivel para su utilización en microcontroladores.

1.- La posibilidad de definir nuevas estructuras de alto nivel para E/S.
2.- El Acceso directo a memoria y al hardware para lectura y escritura.
3.- La posibilidad de llamar rutinas creadas en lenguaje ensamblador.

## 2.3 FUNCIONES Y MACROS

Antes de iniciar una comparación entre programación en lenguaje ensamblador y lenguajes de alto nivel (HLL[6]), existen dos puntos que se aplican a ambos tipos de lenguaje, uno de ellos son las *macro*[7] instrucciones y las funciones.

Los programadores de lenguaje ensamblador necesitan repetir frecuentemente grupos de instrucciones dentro de un programa. La forma mas sencilla de resolver esto es escribirlas cada vez que sea necesario, pero esto es un procedimiento tedioso. Los macros proporcionan una solución sencilla y eficiente al problema a esos bloques o grupos de instrucciones que se utilizan repetidamente en el programa.

La definición de un macro es un método que permite asignar un nombre a una porción de texto (varias instrucciones). Después de haber definido un macro, el programador puede escribir el nombre de dicho macro en vez de escribir toda la porción del texto. Por ejemplo, en un microcontrolador de la familia MCS-51 de Intel, la secuencia para sumar el puerto 1 con el puerto 2 y dejar el resultado en el puerto1 es de la siguiente manera:

```
MOV    A,P1     ; Mover el dato del puerto 1 al acumulador
ADD    A,P2     ; Sumar el contenido del puerto 2 al acumulador
MOV    P1,A     ; Sacar la suma por el puerto 1
```

Si es necesario realizar esta secuencia a lo largo del programa, una posibilidad de hacerlo seria repetir la secuencia cada vez que se necesite, otra es definir el siguiente macro para esta secuencia.

```
AddPorts    MACRO
            MOV A,P1      ; Mover el dato del puerto 1 al acumulador
            ADD A,P2      ; Sumar el contenido del puerto 2 al acumulador
            MOV    P1,A   ; Sacar la suma por el puerto 1
            ENDM
```

Una vez definido podemos utilizar la macro instrucción AddPorts para realizar la secuencia. Esto hace mas inteligible el programa. Sin embargo la utilización de macro instrucciones tiene una desventaja, esto es, cada vez que utilizamos o hacemos referencia a un macro, se repite cada una de las instrucciones que componen el macro, por tanto; el total de veces que se llamada al macro es el total de veces que se almacena la secuencia en memoria, esto hace crecer el requerimiento de memoria.
Una forma de re-utilizar el código, es almacenar en memoria la secuencia una unica vez y que se tenga la posibilidad de hacer referencia a dicha secuencia cuando se desee, dicho lo anterior es posible utilizando los *procedimientos* o *funciones*.

[6] HLL del ingle *High Level Lenguage*.
[7] Prefijo (del gr. *makros*), que significa muy grande.

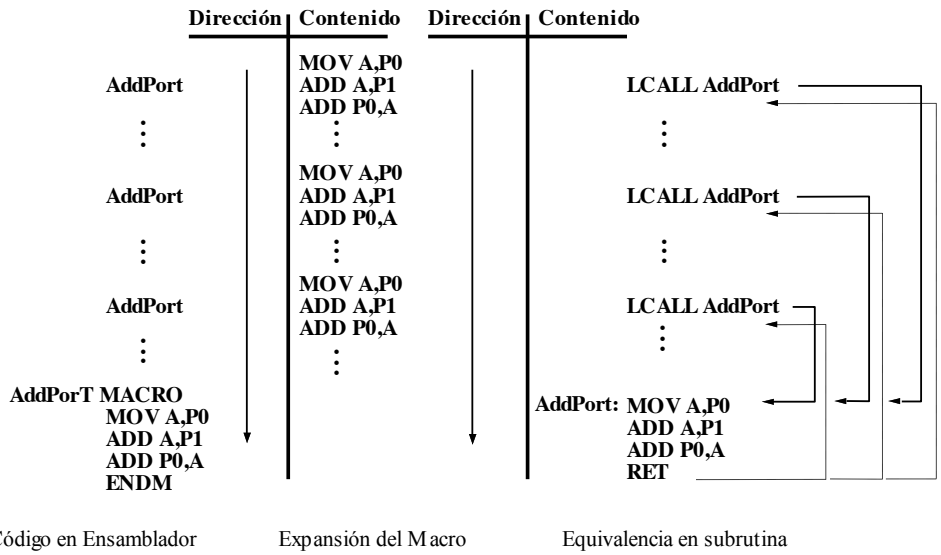



**Figura 2.2** Comparación de la expansión de macros y la utilización de subrutinas.

Los terminos *función* y *procedimiento* son algunas veces utilizados como términos equivalentes, pero existen distinción entre ellos. Las funciones siempre retornan un valor y los procedimientos no.

## Procedimientos

El procedimiento o *subrutina* es una parte importante de la arquitectura del sistema de cualquier computadora. Un procedimiento es un grupo de instrucciones que usualmente desarrollan una tarea. Un procedimiento es una porción que tiene la posibilidad de ser nuevamente usada dentro del programa y que se almacena en memoria una vez, pero se usa tan seguido como sea necesario. Esto permite ahorrar espacio de memoria y hace más fácil el desarrollo de programas. La única desventaja de un procedimiento es que hacen que la computadora tome una pequeña cantidad de tiempo para encadenarse al procedimiento y regresar de él. La instrucción LCALL se encadena a las subrutinas, y la instrucción RET hace que se regrese de ella.

La pila almacena la dirección de retorno siempre que una subrutina es llamada durante la ejecución del programa. La instrucción LCALL empuja en la pila la dirección de la siguiente instrucción. La instrucción RET remueve una dirección de la pila así que el programa regresa a la siguiente instrucción del LCALL. La figura 2.2 muestra la comparación de el uso de macros y subrutinas.

## 2.5 PROGRAMACIÓN EN LENGUAJE "C"

Este lenguaje es el producto final de una serie de tres pasos que se dieron para diseñar un lenguaje de programación. El primero fue BCPL, escrito por Martín Richards, el segundo fue B, producido por Ken Thompson, y el tercero fue C, creado por Dennis Ritchie en 1972 en los Laboratorios Bell establecido en Murray, Nueva Jersey. Originalmente fue diseñado para procesarse con un sistema operativo UNIX de la minicomputadora PDP-11, de Digital Equipment Coporation, pero después se extendió a otras máquinas y sistemas operativos. Actualmente tiene un gran aceptación en computadoras personales.

El lenguaje C aún considerado como lenguaje de alto nivel, también permite el acceso a la programación a bajo nivel (haciendo referencia al nivel de bits) esto lo hace muy poderoso en la programación de sistemas. Lo que ha contribuido a su popularidad son los siguientes puntos:

1) Maneja todo tipo de organización de datos.
2) Tiene un completo conjunto de operadores y un moderno control de estructuras.
3) Maneja bibliotecas que facilitan el manejo de la entrada y salida de datos.
4) Es eficaz y compacto.
5) Tiene un alto grado de portabilidad.

El siguientes listado muestra un pequeño programa escrito en lenguaje C que imprime los números impares del 1 al 100;

```
/* programa escrito en lenguaje C */

#include <stdio.h>

#define MAX   100

main()
{
      int num=1;
      while(num<MAX)
      {
            if(num%2)
            {
              printf('%d  ',num);
            }
            num++;
      }
}
```

El lenguaje C tiene una jerarquía descendiente de funciones con formato para entrada o salida, posee las funciones printf() y scanf() descendientes de funciones de E/S para un sólo carácter como putchar(), putc(), getchar() y getc(); las cuales a su vez están definidas de secuencias de lenguaje ensamblador.

Algunas veces es necesario escribir funciones en ensamblador, C permite que se pueda mezclar rutinas de ensamblador en el programas, ya sea de una manera externa o interna (directamente en el programa). Por ejemplo, la secuencia del macro **AddPort** puede ser codificada en una línea de C de la siguiente manera:

```
asm(" AddPorts: \MOV A,P1 \  ADD A,P2  \ MOV P1,A");
```

De esta manera puede verse ese poder del lenguaje C, puesto que tiene las estructuras de control de un lenguaje de alto nivel, así como la posibilidad de manejar código ensamblador directamente, para así tener acceso directo a registros de microcontrolador o microprocesador.

## 2.6 COMPARACIÓN LENGUAJE "C" Y LENGUAJE ENSAMBLADOR

La comparación del lenguaje C y el lenguaje ensamblador puede ser realizada sobre algunos niveles. El primer nivel es la productividad, como una medida en el tiempo de producir un código ejecutable sin errores,  C tiene la ventaja en este nivel. Esto es debido a que C trabaja con sentencias de mucho mas alto nivel que ensamblador, y ademas el programador no tiene la preocupación del contenido de cada registro y bits de estado, así puede concentrarse más en el flujo lógico del problema.

Un segundo nivel es la accesibilidad al programador de los recursos del hardware de la computadora, microcontrolador o sistema embebido.  Obviamente todos esos recursos están disponibles para el programador en el lenguaje ensamblador, sin embargo muchos de estos recursos (si no es que todos) pueden ser accesados por el lenguaje C. En este punto el lenguaje C no se queda atrás. Por ejemplo, si se desea escribir o leer  de una localidad o un puerto del la sección de E/S mapeada sobre la sección de memoria, un apuntador puede declararse para el tipo de acceso deseado. Por ejemplo, si el hardware tiene un puerto de 8 bits para controlar un convertidor A/D, en este lenguaje se puede utilizar una directiva para direccionar el puerto y un apuntador a un carácter[8] sin signo, como es mostrado a continuación.

```
#define CTL_AD 0x10ff       /* definición del macro CTL_AD igual a 0x10ff      */
unsigned char *DireccAD     /* declaración del apuntador a carácter DireccAD   */
```

Una vez hecho esto, es necesario hacer la asignación de la dirección del A/D al apuntador con la instrucción:

```
DireccAD=(unsigned char *) CTL_AD;       /*asigna dirección del A/D */
```

Ahora, el control del A/D puede hacerse mediante un acceso por referencia al apuntador como:

```
*DireccAD=Palabra_de_Control;      /* inicializa el convertidor para lectura */
```

[8] Se refiere a un tipo de variable de 8 bits en lengauje C llamado *char*.

En algunos casos cuando se lee un convertidor A/D es necesario esperar un bit (EOC[9]) que proporciona el convertidor para indicar que la conversión a finalizado y el dato es válido. En C esta espera de ese bit se puede hacer mediante:

```
while(!(*DireccAD & EOC));                /* espera  conversión completa */
```

donde EOC esta definido con 0x80, como una mascara para verificar únicamente el bit mas significativo del valor leído de la dirección de control del convertidor. En la instrucción se puede ver un símbolo '&' este es el operador AND para bits en C, ademas de este operador, C posee una gran diversidad símbolos para operadores lógicos para bits (bitwise), la tabla 2.1 muestra todos los operadores lógicos que maneja C.

| Operador | Acción |
|---|---|
| & | AND |
| \| | OR |
| ^ | OR exclusiva (XOR) |
| ~ | NOT |
| >> | Corrimiento a la derecha |
| << | Corrimiento a la izquierda. |

**Tabla 2.1** Operadores para bits del lenguaje C.

Ejemplos:

```
char A=15;     /* A=00001111        */

B=A|0x80;      /* B=10001111        */

B=~A;          /* B=11110000        */

B=A<<3;        /* B=01111000        */

B=A>>2;        /* B=00000011        */
```

## 2.7 DESARROLLO DE SOFTWARE

Un enfoque muy utilizado en el desarrollo de programas es el enfoque sistemático llamado *top-down*. En este enfoque un gran problema es partido en pequeños *módulos*. El nivel más alto muestra relaciones y funciones entre esos módulos. Este nivel muestra un vista general del programa completo. Cada uno de los módulos es dividido en módulos aun mas pequeños. La división es continuamente hasta que los pasos en cada modulo sean claramente comprendidos. Ahora a cada programador se le asigna un módulo o un conjunto de módulos para que escriba el programa de dicho módulo.

[9] EOC del ingles *End Of Convertion*.

Un enfoque opuesto a esto es el diseño *botton-up*. En este enfoque cada programador inicia escribiendo módulos en bajo nivel y esperan que todas las piezas sean unidas posteriormente. Muchos equipos modernos de programación usan una combinación de estas técnicas. El desarrollo de herramientas para programación fue ayudado por el descubrimiento de que cualquier programa puede ser representado por tres tipos de operaciones básicas. Primer tipo de operación básica es la *secuencia* la cual significa simplemente hacer una serie de acciones. El segundo tipo de operación es la *decisión* o *selección* esto significa escoger entre dos o más acciones. La tercera operación básica es la *repetición* o *iteración*, la cual significa repetir una serie de acciones hasta que una condición esté o no presente.

Todos los programadores utilizan tres a siete estructuras estándares para representar todas las operaciones en sus programas. Actualmente sólo tres estructuras (SECUENCIA, IF-THEN-ELSE y WHILE-DO) se requieren para representar cualquier acción de un programa, las otras cuatro estructuras se derivan de las ya mencionadas. La representación gráfica de estas estructuras se muestra en la figura 2.3 y 2.4.

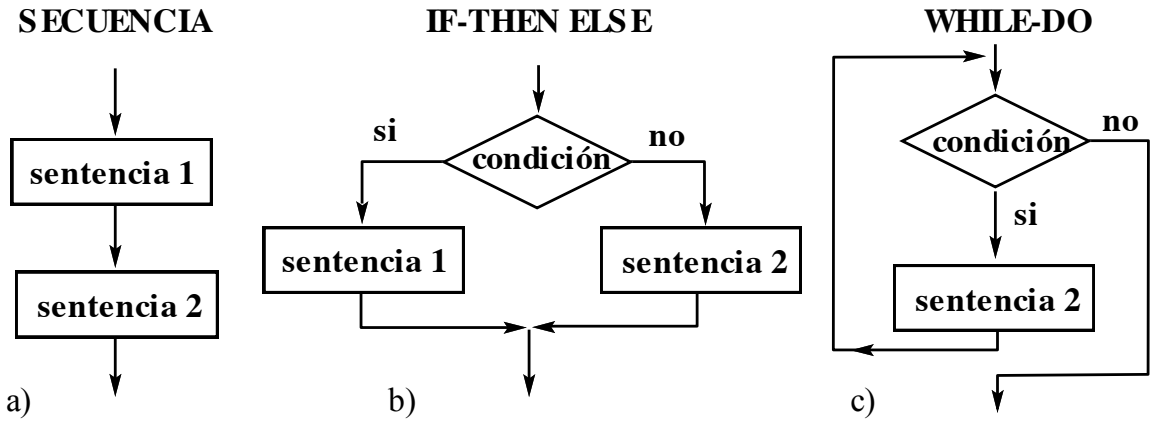



**Figura 2.3** Estructuras básicas para representación de acciones en programas.
a) SENTENCIA b) IF-THEN-ELSE y c) WHILE-DO.

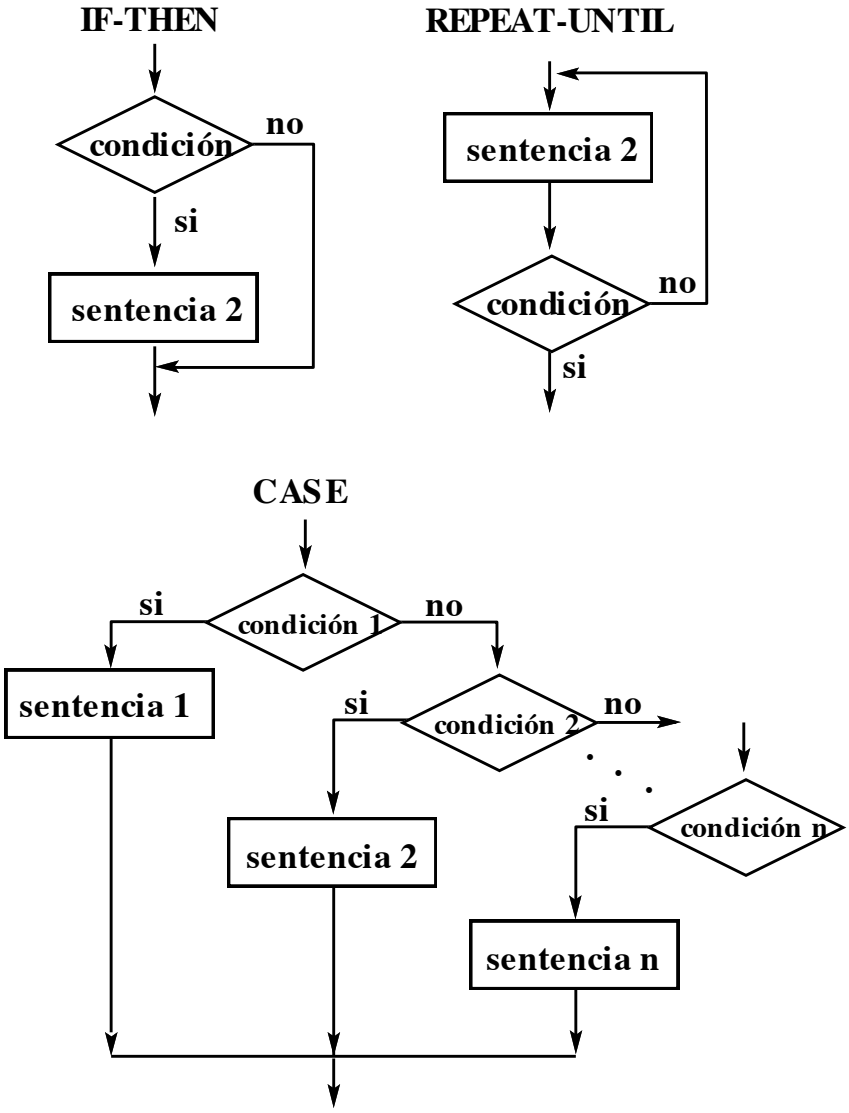



**Figura 2.4** Estructuras derivadas para representación de acciones en programas.
a) IF-THEN b) REPEA-UNTIL y c) CASE.

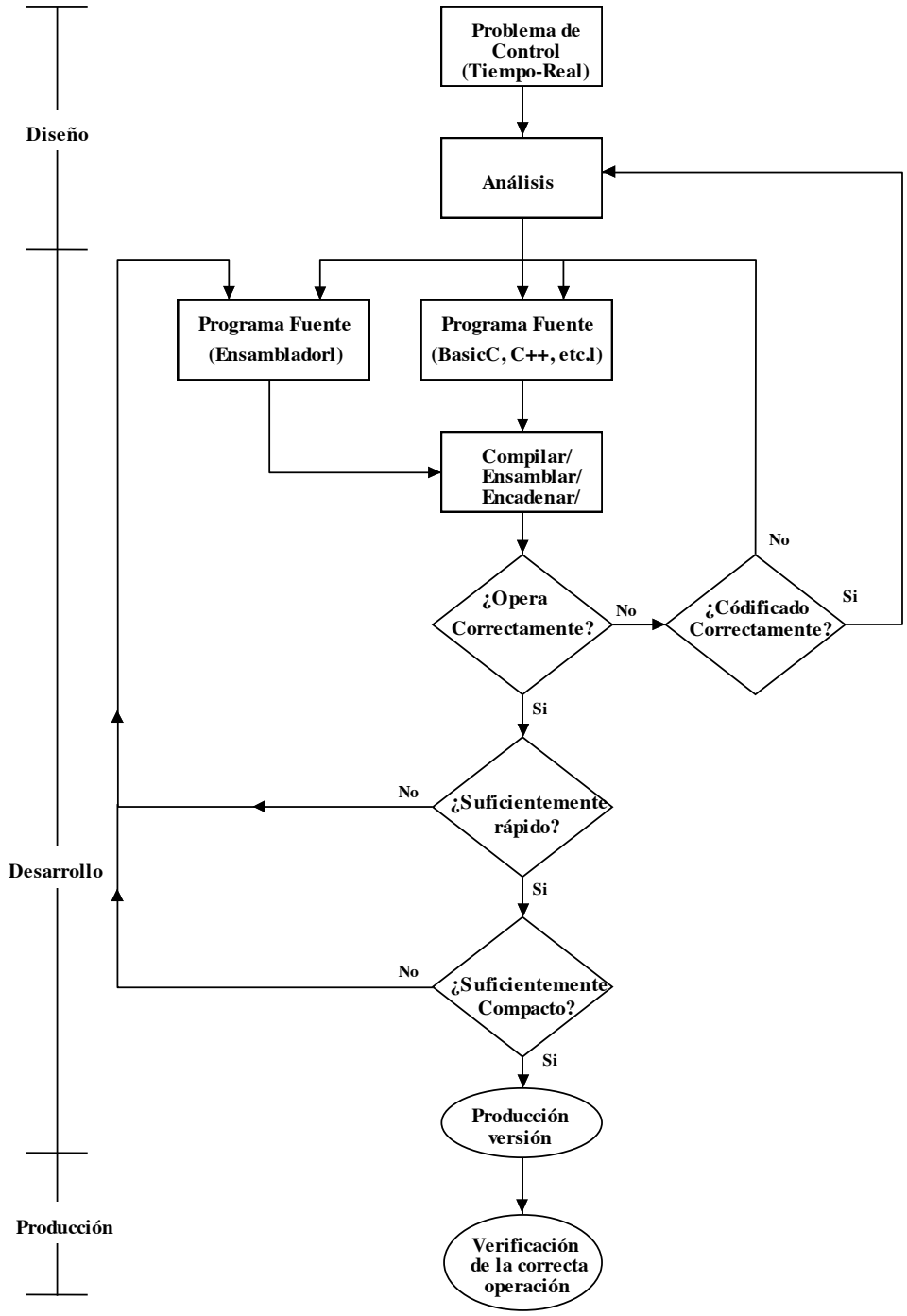



**Figura 2.5** Proceso de Arriba hacia abajo (Top-down) para el desarrollo de programas para microcontroladores.

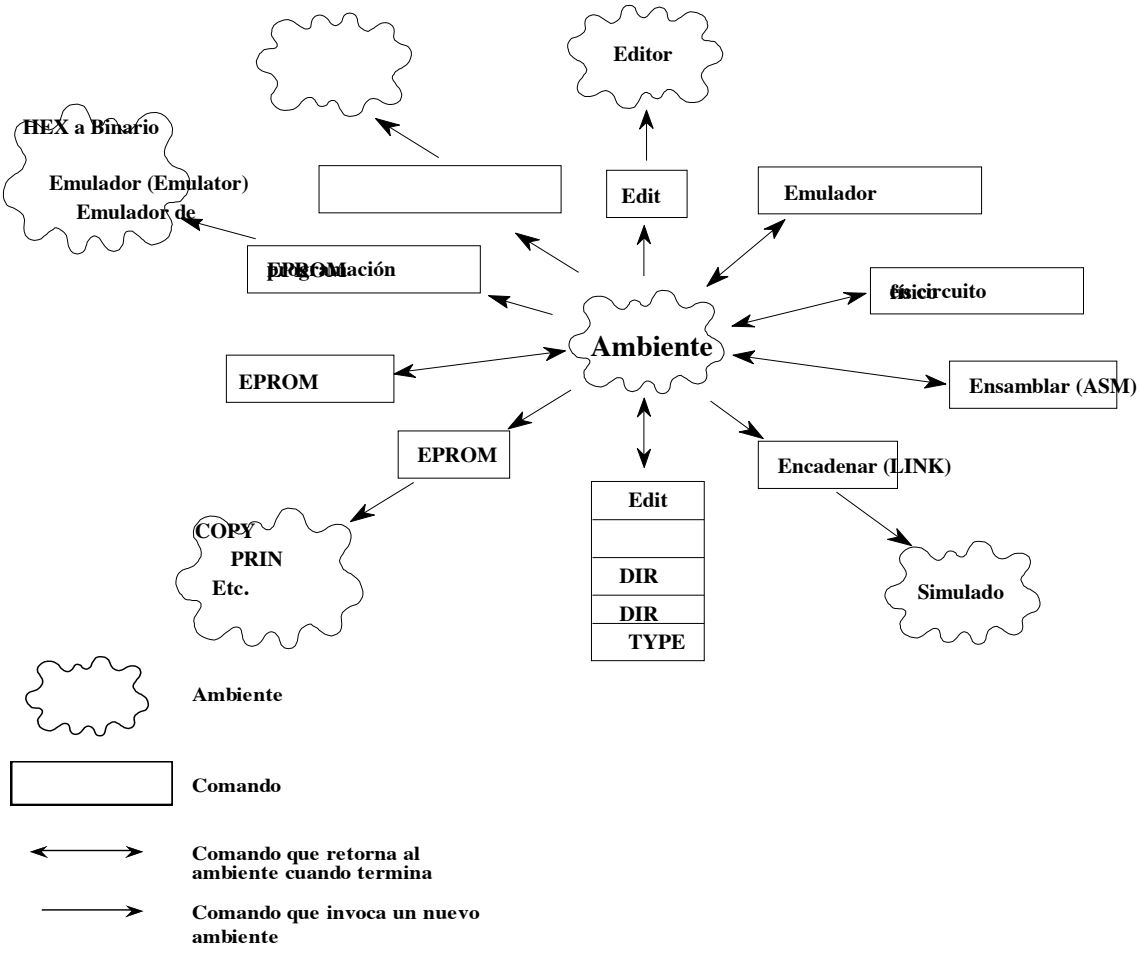



**Figura 2.6** Ambiente del Desarrollo de programas para microcontroladores.

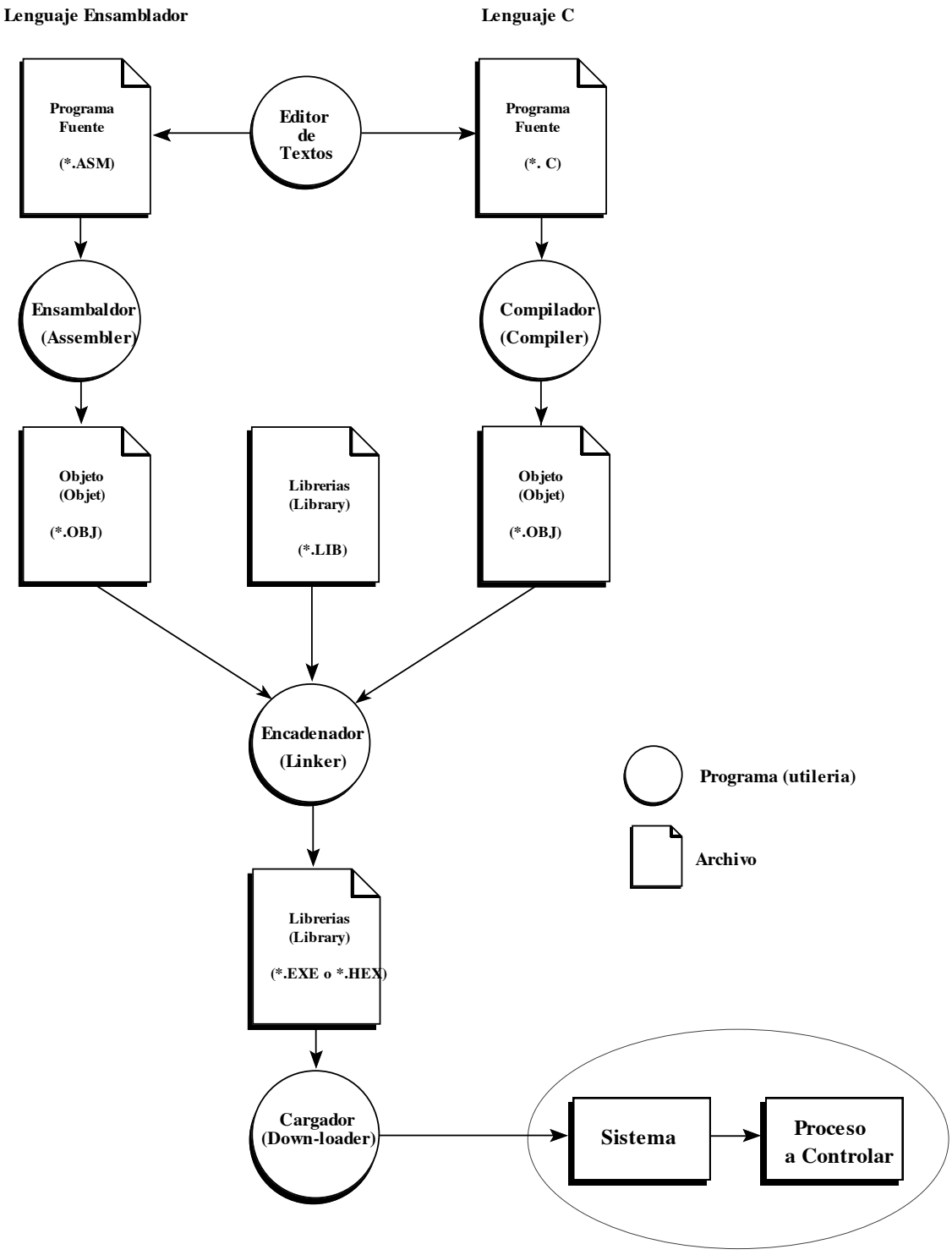


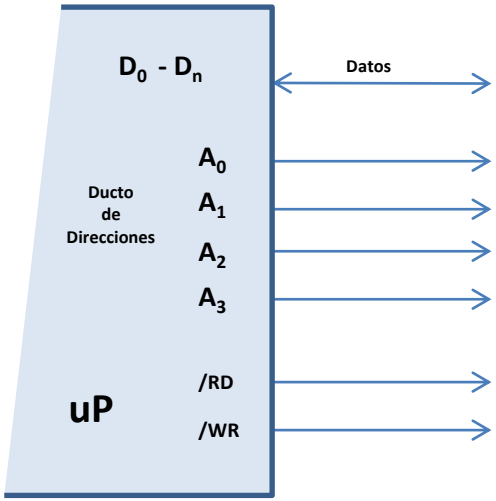

**Figura 2.7** Diagramas de bloques de la conversión de programas de un código fuente a un programa ejecutable por el microcontrolador.

# Decodificadores de Memoria

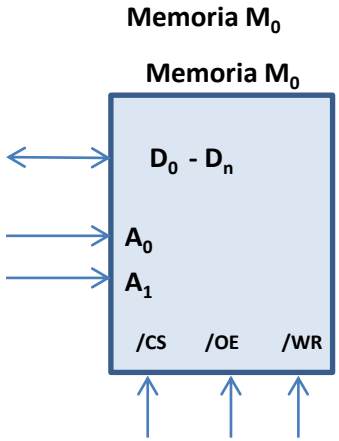
Con fines de explicación considere lo siguiente:

- Un procesador (uP) con un ducto de datos de n bits y un ducto direcciones de 4 bits ($A_3$-$A_0$)
- Una me memoria ($M_0$) de un ducto de datos de n bits y un ducto direcciones de 2 bits ($A_1$,$A_0$)

**Nota**: para este caso explicativo no importa el número de bits del ducto de datos.




1 El uP tiene un espacio de direcciones de $2^4$ (16 localidades) pues su ducto de direcciones es de 4 bits



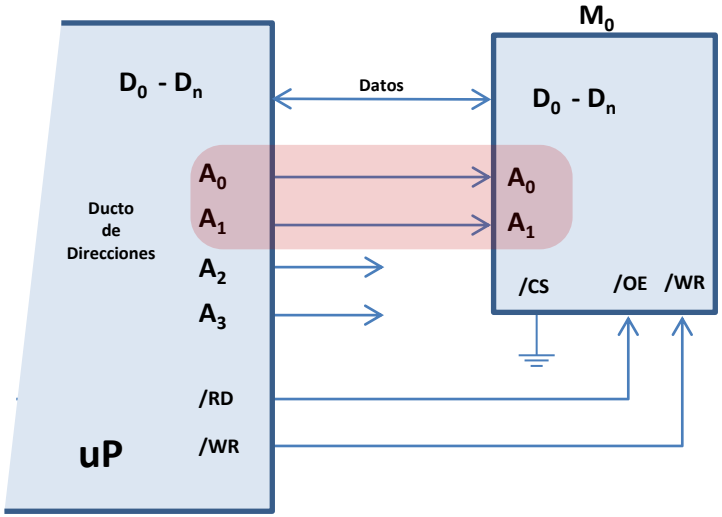

2 La memoria tiene un espacio de direcciones de $2^2$ (4 localidades) pues su ducto de direcciones es de 2 bits

3 Si se desea conectar la memoria al procesador puede verse que el procesador tiene más líneas de direcciones que la memoria; por tanto tiene más localidades diferentes para acceder (16 en total) que las localidades que tiene la memoria (solamente 4).

# Decodificadores de Memoria

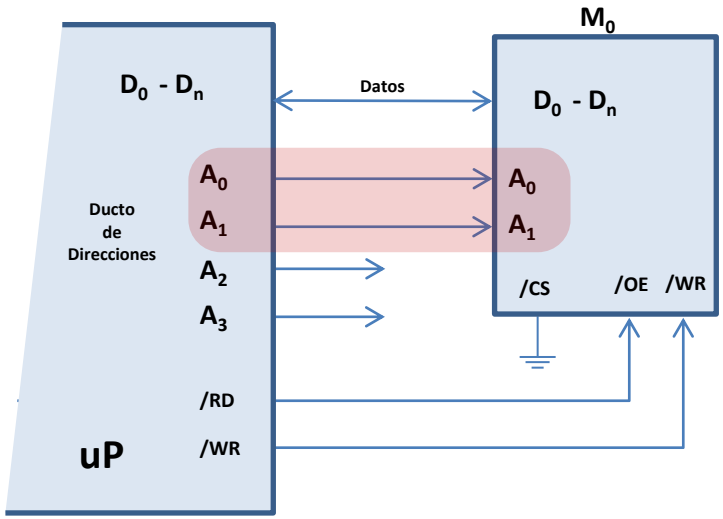
4

Para conectar la memoria al procesador lo lógico es conectar los correspondientes ductos (direcciones, datos y control) con se muestra en la siguiente figura.
**Nota:** dado que se tiene una única memoria podemos tener el selector de la memoria (/CS) siempre activo (conectado a tierra) sin causar un conflicto en el ducto.



# Decodificadores de Memoria

**5** Bajo estas condiciones tenemos entonces la siguiente tabla de direcciones del uP con su correspondiente dirección de la localidad accedida de la memoria.

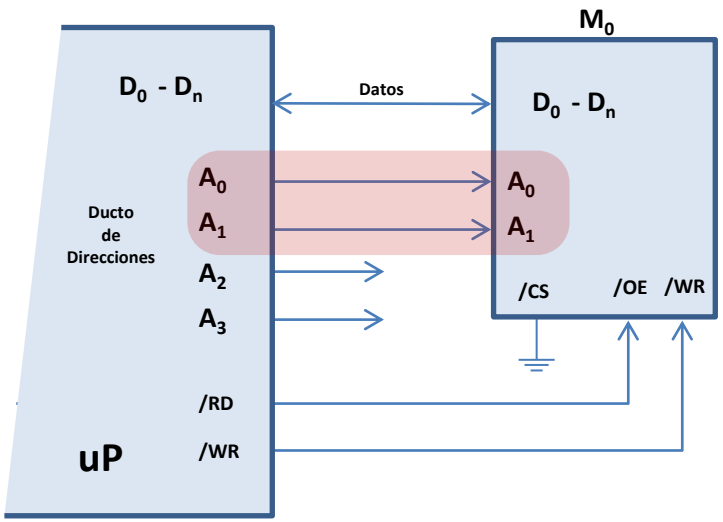



| Dirección (uP) | Dir. Mem (Hex) | Localidad (Hex) |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0010 | 2 | 2 |
| 0011 | 3 | 3 |
| 0100 | 0 | 4 |
| 0101 | 1 | 5 |
| 0110 | 2 | 6 |
| 0111 | 3 | 7 |
| 1000 | 0 | 8 |
| 1001 | 1 | 9 |
| 1010 | 2 | A |
| 1011 | 3 | B |
| 1100 | 0 | C |
| 1101 | 1 | D |
| 1110 | 2 | E |
| 1111 | 3 | F |

# Decodificadores de Memoria

6

Como puede observarse existen diferentes direcciones del uP para las cuales se acceda a la misma localidad de memoria y eso podemos verlo en la tabla donde se muestra que el espacio de memoria (de 4 localidades) se repite a lo largo del espacio de direcciones del uP.

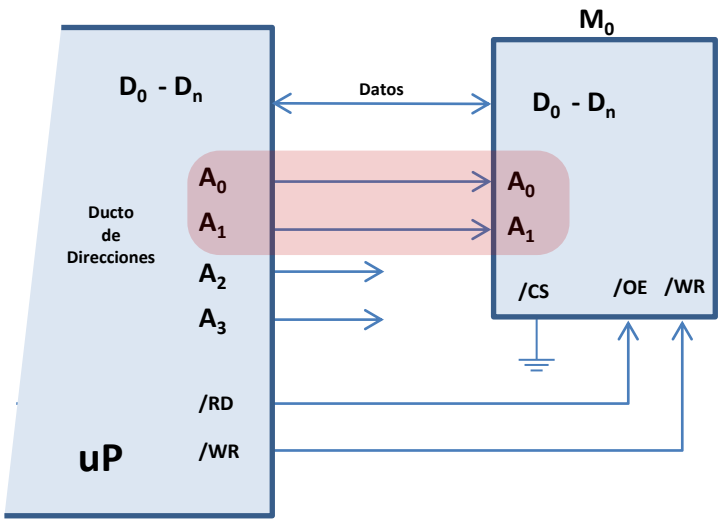



| Dirección (uP) | Dir. Mem (Hex) | Localidad (Hex) |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0010 | 2 | 2 |
| 0011 | 3 | 3 |
| 0100 | 0 | 4 |
| 0101 | 1 | 5 |
| 0110 | 2 | 6 |
| 0111 | 3 | 7 |
| 1000 | 0 | 8 |
| 1001 | 1 | 9 |
| 1010 | 2 | A |
| 1011 | 3 | B |
| 1100 | 0 | C |
| 1101 | 1 | D |
| 1110 | 2 | E |
| 1111 | 3 | F |

# Decodificadores de Memoria

**7**

Esto puede verse que es debido a que la memoria sólo puede detectar (digamos ver) los dos primero bits ($A_1$ y $A_0$) del ducto de direcciones del uP causando entonces que existan **direcciones espejo.** Es decir una dirección de memoria tiene una correspondiente dirección del uP pero también puede verse reflejada en otra dirección diferente del uP.




| Dirección (uP) | Dir. Mem (Hex) | Localidad (Hex) |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0010 | 2 | 2 |
| 0011 | 3 | 3 |
| 0100 | 0 | 4 |
| 0101 | 1 | 5 |
| 0110 | 2 | 6 |
| 0111 | 3 | 7 |
| 1000 | 0 | 8 |
| 1001 | 1 | 9 |
| 1010 | 2 | A |
| 1011 | 3 | B |
| 1100 | 0 | C |
| 1101 | 1 | D |
| 1110 | 2 | E |
| 1111 | 3 | F |

**8**

**¿Es prroblema o no tener direcciones espejo?** Depende…

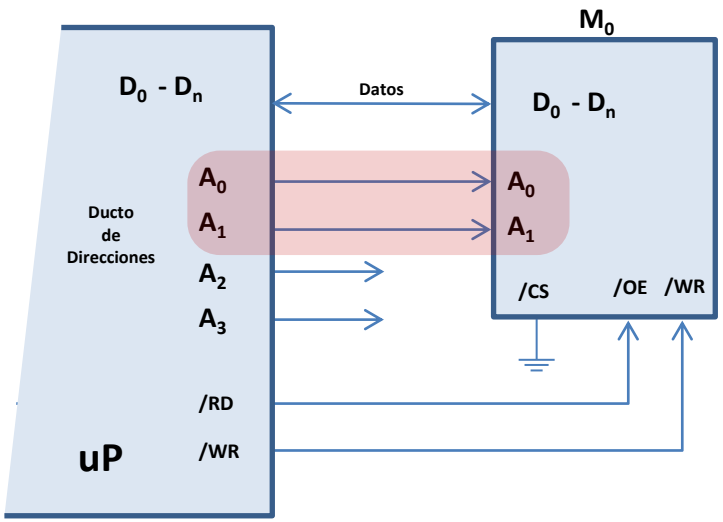
a) Si el programador lo sabe entonces no existe problema alguno pues él deberá tener el suficiente cuidado de usar las direcciones correctas.

b) Si el programador desconoce la existencia de zona espejos puede pensar que tiene más memoria de la que realmente existe y sobre escribir datos causando así problemas.

# Decodificadores de Memoria

**9**

**¿Cómo evitar que existan estas zonas espejo de la memoria?**
Se tendría que forzar la asignación de un rango exclusivo de direcciones del uP a la memoria. Como ejemplo podemos decir que la memoria $M_0$ debería ser accedida exclusivamente en el rango de 0 a 3 de direcciones del uP.



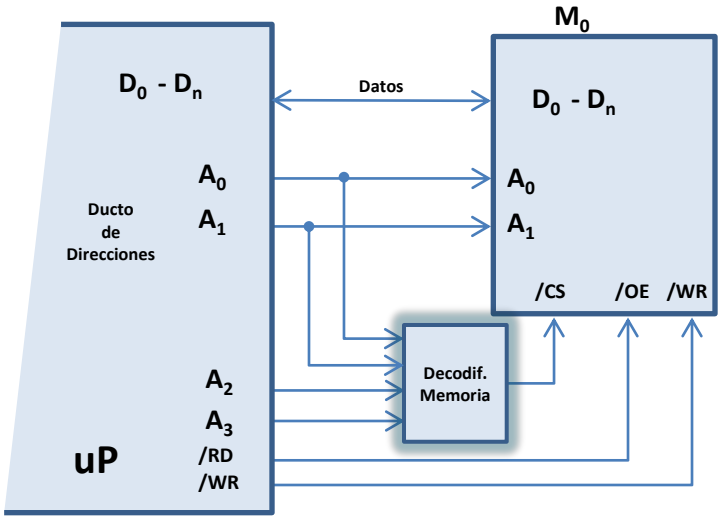

| Dirección (uP) | Dir. Mem (Hex) | Localidad (Hex) |
|---|---|---|
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0010 | 2 | 2 |
| 0011 | 3 | 3 |
| 0100 | 0 | 4 |
| 0101 | 1 | 5 |
| 0110 | 2 | 6 |
| 0111 | 3 | 7 |
| 1000 | 0 | 8 |
| 1001 | 1 | 9 |
| 1010 | 2 | A |
| 1011 | 3 | B |
| 1100 | 0 | C |
| 1101 | 1 | D |
| 1110 | 2 | E |
| 1111 | 3 | F |

**10**

Para esto entonces es necesario un decodificador de memoria para que $M_0$ sólo se active en el rango de direcciones correspondientes.

# Decodificadores de Memoria

**11** Para esto entonces es necesario un decodificador de memoria para que $M_0$ sólo se active en el rango de direcciones correspondientes.




| Dirección (uP) | Dir. Mem (Hex) | Localidad (Hex) | $M_0$ /CS |
|---|---|---|---|
| 0000 | 0 | 0 | 0 |
| 0001 | 1 | 1 | 0 |
| 0010 | 2 | 2 | 0 |
| 0011 | 3 | 3 | 0 |
| 0100 | 0 | 4 | 1 |
| 0101 | 1 | 5 | 1 |
| 0110 | 2 | 6 | 1 |
| 0111 | 3 | 7 | 1 |
| 1000 | 0 | 8 | 1 |
| 1001 | 1 | 9 | 1 |
| 1010 | 2 | A | 1 |
| 1011 | 3 | B | 1 |
| 1100 | 0 | C | 1 |
| 1101 | 1 | D | 1 |
| 1110 | 2 | E | 1 |
| 1111 | 3 | F | 1 |

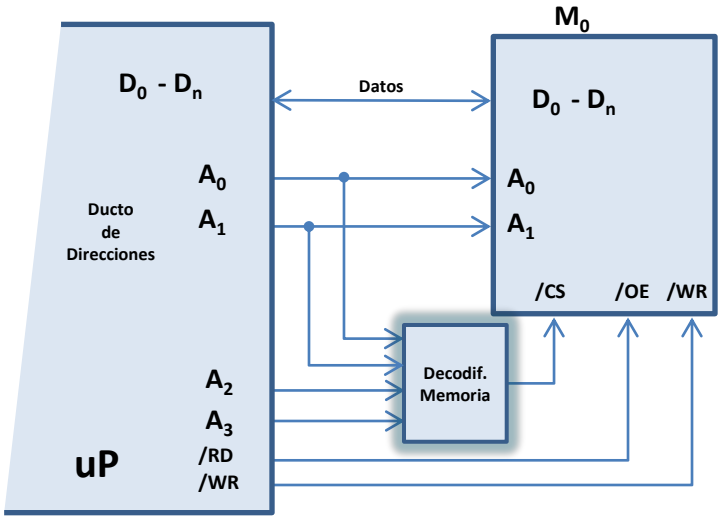
**Activar memoria**

**Desactivar memoria**

**12** Por tanto el decodificador de memoria tendría como entradas las líneas de ducto de direcciones (A3-A0) y como salidas el selector de memoria (/CS). Así el decodificador determinará si la dirección presente corresponde al rango y entonces activar (hacer cero) la salida **/CS**; o desactivar (hacer uno) en cualquier otro caso en la que no corresponde al rango – ver tabla.

# Decodificadores de Memoria

**13** El decodificador es un circuito combinaciones donde su salida depende de la entrada (para al entrada, tal salida) y para el diseño se hace uso de mapas de Karnaugh .




| Dirección (uP) | Dir. Mem (Hex) | Localidad (Hex) | $M_0$ /CS |
|---|---|---|---|
| 0000 | 0 | 0 | 0 |
| 0001 | 1 | 1 | 0 |
| 0010 | 2 | 2 | 0 |
| 0011 | 3 | 3 | 0 |
| 0100 | 0 | 4 | 1 |
| 0101 | 1 | 5 | 1 |
| 0110 | 2 | 6 | 1 |
| 0111 | 3 | 7 | 1 |
| 1000 | 0 | 8 | 1 |
| 1001 | 1 | 9 | 1 |
| 1010 | 2 | A | 1 |
| 1011 | 3 | B | 1 |
| 1100 | 0 | C | 1 |
| 1101 | 1 | D | 1 |
| 1110 | 2 | E | 1 |
| 1111 | 3 | F | 1 |

Activar memoria

Desactivar memoria

/CS | $A_3$ $A_2$ \ $A_1$ $A_0$

| $A_3$ $A_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 1 | 1 | 1 |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |

$/CS' = A_3' A_2'$

$/CS = A_3 + A_2$

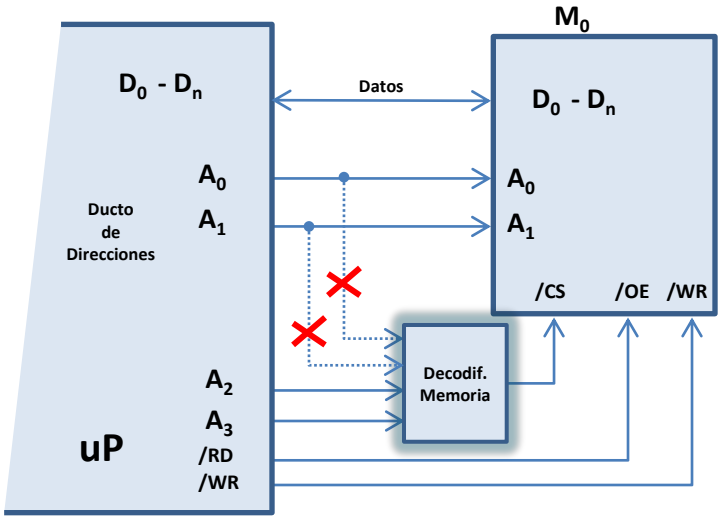
**14** Llenando el mapa y tomando ceros (son menos) tenemos como resultado:

$$/CS = A_3 + A_2$$

**Nota:** En este caso particular resultaría la misma ecuación si tomamos los unos en lugar de los ceros.

# Decodificadores de Memoria

**15** Se puede observar que en el proceso desaparecen las líneas $A_0$ y $A_1$ – líneas que van directamente a la memoria.




| Dirección (uP) | Dir. Mem (Hex) | Localidad (Hex) | $M_0$ /CS | |
|---|---|---|---|---|
| 0 0 0 0 | 0 | 0 | 0 | Activar memoria |
| 0 0 0 1 | 1 | 1 | 0 | |
| 0 0 1 0 | 2 | 2 | 0 | |
| 0 0 1 1 | 3 | 3 | 0 | |
| 0 1 0 0 | – | 4 | 1 | Desactivar memoria |
| 0 1 0 1 | – | 5 | 1 | |
| 0 1 1 0 | – | 6 | 1 | |
| 0 1 1 1 | – | 7 | 1 | |
| 1 0 0 0 | – | 8 | 1 | |
| 1 0 0 1 | – | 9 | 1 | |
| 1 0 1 0 | – | A | 1 | |
| 1 0 1 1 | – | B | 1 | |
| 1 1 0 0 | – | C | 1 | |
| 1 1 0 1 | – | D | 1 | |
| 1 1 1 0 | – | E | 1 | |
| 1 1 1 1 | – | F | 1 | |

| /CS | $A_1$ $A_0$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|---|
| $A_3$ $A_2$ | | | | | |
| 00 | | 0 | 0 | 0 | 0 |
| 01 | | 1 | 1 | 1 | 1 |
| 11 | | 1 | 1 | 1 | 1 |
| 10 | | 1 | 1 | 1 | 1 |

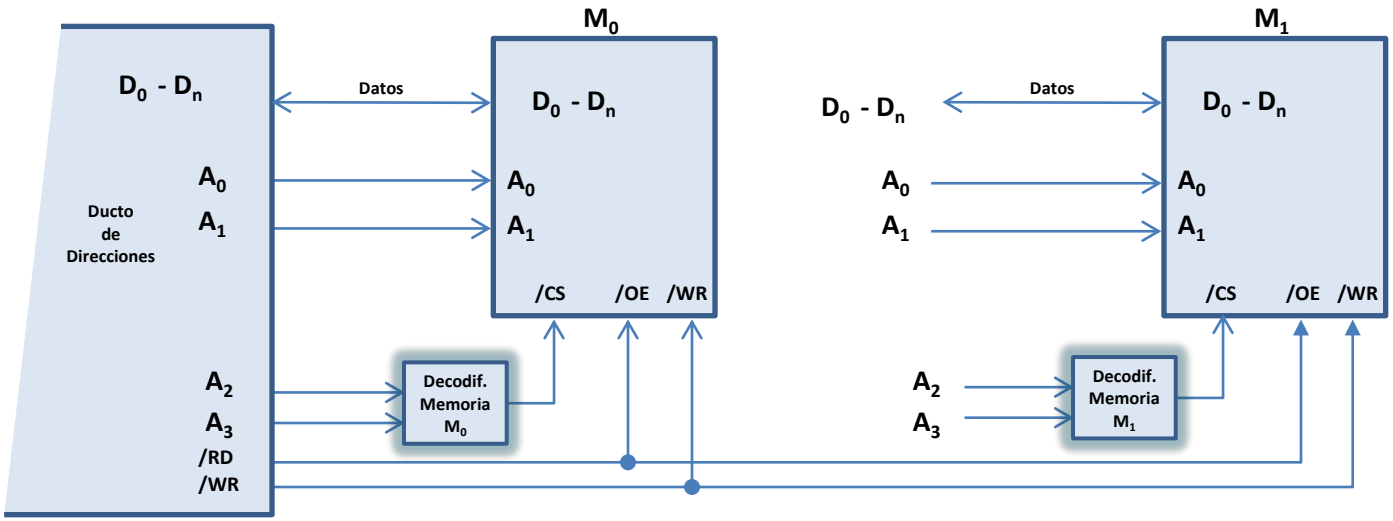
$/CS' = A_3' A_2'$

$/CS = A_3 + A_2$

**16** La razón de que $A_0$ y $A_1$ se eliminen es porque esas líneas no nos ayudan a discriminar el rango exclusivo deseado pues en todo el especio de direcciones del uP siempre cambiaron – ver zona azul. En cambio $A_3$ y $A_2$ si ayudan a discriminar la zona pues estás permanecen en cero en la zona exclusiva que se desea activar la memoria.

# Decodificadores de Memoria

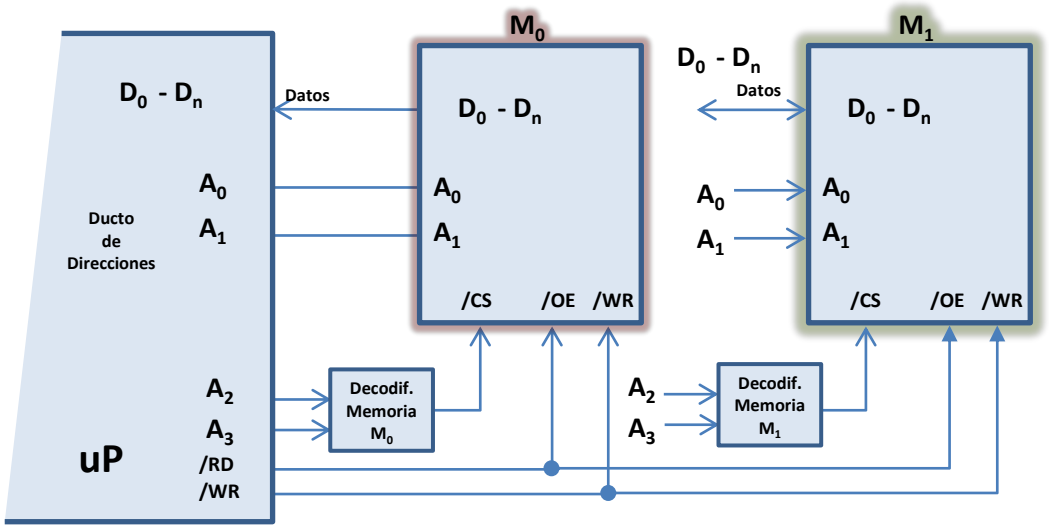
**17** ¿Qué sucede si ahora tenemos dos memorias?




**18** Para cada memoria debe existir un decodificador para que cada una de las memorias de active exclusivamente en su rango de direcciones válido.

# Decodificadores de Memoria

19 Un ejemplo de esto sería el siguiente esquema de dos memoria de 4 localidades cada una.




| Dirección (uP) | Dir. Mem 0 (Hex) | $M_0$ /CS | Dir. Mem 1 (Hex) | $M_1$ /CS | |
|---|---|---|---|---|---|
| 0000 | 0 | 0 | - | 1 | Activar $M_0$ |
| 0001 | 1 | 0 | - | 1 | |
| 0010 | 2 | 0 | - | 1 | |
| 0011 | 3 | 0 | - | 1 | |
| 0100 | - | 1 | 0 | 0 | Activar $M_1$ |
| 0101 | - | 1 | 1 | 0 | |
| 0110 | - | 1 | 2 | 0 | |
| 0111 | - | 1 | 3 | 0 | |
| 1000 | - | 1 | - | 1 | |
| 1001 | - | 1 | - | 1 | |
| 1010 | - | 1 | - | 1 | |
| 1011 | - | 1 | - | 1 | |
| 1100 | - | 1 | - | 1 | |
| 1101 | - | 1 | - | 1 | |
| 1110 | - | 1 | - | 1 | |
| 1111 | - | 1 | - | 1 | |

20 Cada memoria tiene su decodificador y para su diseño para cada uno se realiza su propio proceso según el rango a activar