

DISEÑO DE SISTEMAS DIGITALES Y ARQUITECTURA DE COMPUTADORAS

Jorge Luis Ortega Arjona

TEMAS DE COMPUTACIÓN



JORGE LUIS ORTEGA ARJONA

DISEÑO DE SISTEMAS
DIGITALES Y ARQUITECTURA
DE COMPUTADORAS

FACULTAD DE CIENCIAS
2019



621.392

Ortega Arjona, Jorge Luis, autor.

Diseño de sistemas digitales y arquitectura de computadoras / Jorge Luis Ortega Arjona. -- 1a. edición. -- Ciudad de México : Universidad Nacional Autónoma de México, Facultad de Ciencias, 2019.

xviii, 334 páginas : ilustraciones ; 23 cm. -- (Temas de computación)

Incluye índice.

ISBN: 978-607-30-2359-7

1. Circuitos lógicos -- Diseño y construcción. 2. Computadoras digitales electrónicas -- Circuitos -- Diseño y construcción. 3. Arquitectura de computadoras. I. Universidad Nacional Autónoma de México. Facultad de Ciencias, editor. II. Título. III. Serie.

Biblioteca Nacional de México

No. de sistema[000714960]scdd 22

Esta obra contó con el apoyo PAPIME PE-102619

Diseño de sistemas digitales y arquitectura de computadoras

1ª edición, 5 de septiembre de 2019

© DR. 2019. Universidad Nacional Autónoma de México

Facultad de Ciencias

Ciudad Universitaria, Delegación Coyoacán.

C.P. 04510. Ciudad de México

Coordinación de servicios editoriales: editoriales@ciencias.unam.mx

Plaza Prometeo: tienda.fcencias.unam.mx

ISBN: 978-607-30-2359-7

Diseño de portada: Eliete Martín del Campo Treviño

Formación de interiores en LaTeX: Rafael Reyes Sánchez

Prohibida la reproducción total o parcial de la obra, por cualquier medio, sin la autorización por escrito del titular de los derechos.

Impreso y hecho en México

A Lucía y Jorge Luis

Índice general

Introducción	XIX
1. ¿Qué es una computadora?	1
Resumen	1
Objetivos del capítulo	1
1.1. Datos, información y el concepto de programa almacenado	2
1.1.1. Resumen del desarrollo histórico de las computadoras	3
1.1.2. La computadora y las cosas que se pueden hacer con ella	8
1.1.3. Ideas básicas sobre computadoras digitales	10
1.2. Los subsistemas básicos de la computadora	11
1.2.1. Componentes del sistema: componentes básicos	11
1.2.2. Memoria y discos	12
1.2.3. Dispositivos periféricos	13
1.2.4. <i>Buses</i>	14
1.2.5. Estudiando la arquitectura de sistemas de cómputo	17
2. Lógica y secuenciación	23
Resumen	23
Objetivos del capítulo	23
2.1. Sistemas numéricos	24
2.1.1. Ideas básicas de los sistemas numéricos	24
2.1.2. Sistema numérico binario	27
2.1.3. Sistemas octal y hexadecimal	30
2.1.4. Parte fraccional	32
2.1.5. Cambiando de una base a otra	34

2.1.6.	¿Porqué utilizar binarios en sistemas de cómputo?	41
2.1.7.	Algo de aritmética binaria elemental	41
2.2.	Álgebra booleana y compuertas lógicas	44
2.2.1.	Álgebra booleana	44
2.2.2.	Lógica digital	46
2.2.3.	Compuertas lógicas	49
2.2.4.	Interconexión de compuertas para obtener otras compuertas	60
2.2.5.	Identidades básicas y leyes del álgebra booleana	64
2.2.6.	Formas canónicas	67
2.2.7.	Minimización de circuitos: mapas de Karnaugh	68
2.3.	Circuitos combinacionales simples	76
2.3.1.	Circuitos combinacionales	76
2.3.2.	Decodificadores y codificadores	76
2.3.3.	Multiplexores y demultiplexores	79
2.3.4.	Circuitos aritméticos. El sumador medio	82
2.3.5.	El sumador completo	85
2.3.6.	El sumador binario de n bits	87
2.3.7.	Substractores	88
2.3.8.	Multiplicación y división	88
2.4.	Lógica secuencial y máquinas de estado finito	89
2.5.	Flip-flops	92
2.5.1.	El flip-flop RS	93
2.5.2.	El flip-flop D	97
2.5.3.	El flip-flop JK	98
2.5.4.	El flip-flop T	100
2.5.5.	Entradas de inicialización	101
2.5.6.	Señales de reloj	101
2.6.	Registros	103
2.6.1.	El registro básico	104
2.6.2.	El registro de corrimiento	105
2.6.3.	Contadores	112
2.6.4.	Detectores de secuencia y generadores de secuencia	114
2.6.5.	Utilizando registros en lógica secuencial	116
2.7.	Cómputo digital básico	117
2.7.1.	Aritmética modular	121
2.7.2.	Aritmética de complemento a 2	125

2.7.3.	Complemento a 2	127
2.7.4.	Uso del complemento a 2 para la sustracción	130
2.7.5.	Multiplicación y división	131
2.7.6.	Números de punto flotante	133
2.7.7.	Suma y resta de punto flotante	137
2.7.8.	Multiplicación y división de punto flotante	138
2.8.	La unidad lógica-aritmética (ALU)	139
2.8.1.	Adición e incremento	140
2.8.2.	Limpieza (CLEAR)	142
2.8.3.	AND lógica	142
2.8.4.	OR lógica	143
2.8.5.	XOR lógica	144
2.8.6.	Corrimiento a la derecha	145
2.8.7.	Corrimiento a la izquierda	145
2.8.8.	Complemento	146
2.8.9.	Conexión de entradas	146
2.8.10.	Inspección de salidas	147
2.8.11.	Conexión de salidas	149
3.	Componentes de un sistema de cómputo	155
	Resumen	155
	Objetivos del capítulo	155
3.1.	Organización de una computadora digital	156
3.2.	La unidad central de proceso	158
3.2.1.	La unidad central de proceso (CPU)	158
3.2.2.	Funcionamiento de la CPU	159
3.2.3.	Tipos de datos	159
3.2.4.	<i>Software</i> del sistema	160
3.3.	<i>Buses</i> , memoria y dispositivos de entrada/salida	165
3.3.1.	Los <i>buses</i>	166
3.3.2.	Organización y capacidad de la memoria	169
3.3.3.	La estructura básica de la memoria	170
3.3.4.	Dispositivos de entrada/salida	172
3.4.	Instrucciones y secuenciación	173
3.4.1.	Estructura de la palabra	175
3.4.2.	El ciclo de instrucción	177

3.4.3. El ciclo de búsqueda	178
3.4.4. El ciclo de ejecución	179
3.5. Ejecución de instrucciones	179
3.6. La computadora digital completa	185
3.7. Programación en lenguaje de máquina	188
3.7.1. Programas simples	191
3.7.2. Salida de datos	196
3.8. Lenguaje ensamblador	201
3.9. Lenguajes de alto nivel	207
4. Sistemas de memoria	215
Resumen	215
Objetivos del capítulo	215
4.1. Generalidades	216
4.2. La jerarquía de la memoria	217
4.2.1. El sistema de memoria	217
4.2.2. El compromiso desempeño/precio	219
4.3. Registros y memoria caché	221
4.4. Memoria principal y virtual	222
4.5. Tipos de memorias	223
4.5.1. Memorias semiconductoras	223
4.5.2. Memoria de acceso aleatorio (RAM)	223
4.5.3. Memoria de solo lectura (ROM)	230
4.5.4. Paralelización de dispositivos de memoria	234
4.5.5. La terminal <i>Output Enable</i> (OE)	236
4.6. Dispositivos de almacenamiento auxiliar	239
4.6.1. Generalidades	239
4.6.2. Interfaces con dispositivos de almacenamiento	241
4.6.3. Discos	242
4.6.4. Cintas	245
4.7. Códigos	247
4.7.1. Códigos de detección de errores	249
5. Desempeño	255
Resumen	255
Objetivos del capítulo	255

5.1. Factores que afectan el desempeño	256
5.1.1. ¿Qué es la velocidad de una computadora?	256
5.2. Velocidad de reloj y ciclos de instrucción	257
5.3. Aumentando el desempeño del procesador	258
5.3.1. Aumentando la velocidad de reloj	258
5.3.2. Reduciendo los ciclos por instrucción (CPI)	260
5.3.3. Pipelines	260
5.3.4. Computadoras de conjunto reducido de instrucciones (RISC)	262
5.3.5. Procesadores súperescalares	263
5.3.6. El papel del compilador	264
5.4. MIPS y FLOPS	265
5.4.1. La medida de desempeño MIPS	265
5.5. Ancho de banda y cuellos de botella	266
5.5.1. El procesador de cuatro ciclos	266
5.5.2. El procesador súperescalar	267
5.5.3. Cachés	267
6. Periféricos	275
Resumen	275
Objetivos del capítulo	275
6.1. Tipos de periféricos	275
6.1.1. Clasificando periféricos	276
6.2. Conectando periféricos	276
6.2.1. Dispositivos paralelos	277
6.2.2. Dispositivos seriales	277
6.2.3. Dispositivos de interface con registro y memoria	278
6.3. Dispositivos interactivos	279
6.3.1. Teclado	279
6.3.2. Ratones y dispositivos apuntadores	279
6.3.3. Sonido	280
6.4. Pantallas	281
6.4.1. El tubo de rayos catódicos	281
6.4.2. Pantallas de cristal líquido (LCD)	283
6.4.3. Pantallas de plasma	283
6.5. Impresoras	283
6.5.1. Tipos de impresoras	284

6.5.2. Impresoras de inyección de tinta	284
6.5.3. Impresoras láser	285
6.5.4. Impresoras especializadas	285
6.6. Periféricos especiales	285
6.6.1. Modems	285
6.6.2. Escaners	286
6.6.3. Controladores de juegos	286
6.6.4. Tarjetas de TV	286
7. Introducción a sistemas operativos	289
Resumen	289
Objetivos del capítulo	289
7.1. ¿Qué es un sistema operativo?	290
7.1.1. La historia	290
7.2. Componentes de un sistema operativo	291
7.3. Administración de memoria	293
7.3.1. Mapeo en memoria segmentada	296
7.3.2. Mapeo en memoria paginada	297
7.3.3. Memoria virtual	298
7.4. Sistemas de archivos	299
7.4.1. Lo que los sistemas de archivos hacen	299
7.4.2. Bloques y bytes	299
7.4.3. Sistemas de archivos	300
7.5. Administración de dispositivos	300
7.5.1. ¿Porqué los dispositivos necesitan una administración? . . .	301
7.6. Administración de procesos y multitareas	301
Bibliografía	309

Índice de figuras

1.1. Esquema de una computadora como máquina de procesamiento de datos.	2
1.2. Esquema de una computadora.	15
1.3. Niveles de estudio y abstracción de un sistema de cómputo.	17
2.1. Una representación gráfica de un registro de 8 bits almacenando el número 10110101 ₂	43
2.2. Un circuito lógico simple.	47
2.3. Otro circuito lógico.	48
2.4. Un circuito AND con interruptores.	50
2.5. Símbolos para la compuerta AND de dos y tres entradas.	51
2.6. AND.	51
2.7. Un circuito de interruptores para la operación lógica OR.	52
2.8. Símbolos para la compuerta OR.	52
2.9. Un circuito con compuertas para el circuito de la Figura 2.3.	53
2.10. OR.	54
2.11. El diagrama lógico de la compuerta NOT.	54
2.12. Un circuito lógico que realiza la operación $B = A_1 A_2'$	54
2.13. NOT (inversor).	55
2.14. El diagrama lógico de la compuerta NOR.	56
2.15. NOR (NOT OR).	56
2.16. El diagrama lógico de la compuerta NAND.	57
2.17. NAND (NOT AND).	57
2.18. El diagrama lógico de la compuerta XOR.	58
2.19. Exclusive OR (XOR o diferencia).	59

2.20. Exclusive NOR (XNOR o igualdad).	59
2.21. Interconexiones entre compuertas AND, OR y NOT para obtener compuertas NOR, NAND y XOR.	61
2.22. Interconexiones entre compuertas NOR para obtener compuertas NOT, OR y AND.	62
2.23. Interconexiones entre compuertas NAND para obtener compuertas NOT, AND y OR.	64
2.24. Un sistema elevador	69
2.25. Un mapa de Karnaugh de dos variables.	69
2.26. Un mapa de Karnaugh para $f(x_1, x_2)$	70
2.27. Ligando celdas.	71
2.28. Un mapa de Karnaugh para tres variables.	72
2.29. Ligando celdas.	73
2.30. Un mapa de Karnaugh para cuatro variables.	73
2.31. Mapa de Karnaugh para el problema del elevador.	75
2.32. Un circuito de control para subir un elevador	75
2.33. Un decodificador en general, y un ejemplo de decodificador de 2×4	77
2.34. Un decodificador de 3×8 formado por dos decodificadores de 2×4	78
2.35. Diagrama de bloque de un multiplexor con cuatro entradas.	79
2.36. Un multiplexor con cuatro entradas.	81
2.37. Un multiplexor y un demultiplexor utilizados en comunicaciones digitales.	82
2.38. Interconexiones de compuertas para obtener un sumador medio.	84
2.39. Interconexiones de compuertas para obtener un sumador completo.	86
2.40. Un sumador binario de n bits.	87
2.41. Un semáforo para tráfico vehicular.	89
2.42. Estados de un semáforo.	90
2.43. Estados de un contador.	91
2.44. Diagrama de bloques de un flip-flop.	93
2.45. El flip-flop RS: diagrama de bloques e implementación con com- puertas NOR.	94
2.46. Un tren de pulsos de reloj.	96
2.47. Un flip-flop RS con señal de reloj.	96
2.48. Un flip-flop D y su implementación mediante flip-flop RS.	98
2.49. Un flip-flop JK y su implementación mediante flip-flop RS y com- puertas AND.	99

2.50. Un flip-flop T y su implementación mediante flip-flop JK.	100
2.51. Un flip-flop RS maestro-esclavo.	103
2.52. Registros y su representación.	105
2.53. La información binaria contenida en un registro de corrimiento durante una serie de pulsos de reloj sucesivos.	106
2.54. Dos registros de corrimiento de tres bits.	107
2.55. Un registro.	109
2.56. Una etapa del registro.	110
2.57. Contador módulo 8 usando flip-flops JK.	113
2.58. Detector de la secuencia 010.	115
2.59. Un circuito secuencial general.	116
2.60. Circuito secuencial para el ejemplo del contador.	117
2.61. Diagrama de bloque de un sumador completo.	118
2.62. Un sumador de tres bits.	119
2.63. Una ALU simple de tres bits.	120
2.64. Entradas al i -ésimo flip-flop de la ALU.	139
2.65. Etapa de la ALU que realiza ya sea la suma o el incremento. . . .	140
2.66. Etapa de la ALU que realiza la limpieza del acumulador.	142
2.67. Etapa de la ALU que realiza la AND lógica.	143
2.68. Etapa de la ALU que realiza la OR lógica.	144
2.69. Etapa de la ALU que realiza la XOR lógica.	144
2.70. Etapa de la ALU que realiza el corrimiento a la derecha.	145
2.71. Etapa de la ALU que realiza el corrimiento a la izquierda.	146
2.72. Etapa de la ALU que realiza complemento.	146
2.73. Circuito de inspección negativa.	148
2.74. Circuito de inspección de cero.	148
2.75. Circuito que provee una sola salida para las inspecciones negativa y de cero.	149
2.76. Circuito del ejercicio 2.	150
2.77. Circuito del ejercicio 3.	150
2.78. Circuito de la pregunta 11.	153
3.1. Diagrama de bloques de una computadora digital básica.	156
3.2. Ejemplo del formato de una instrucción.	160
3.3. Una ALU simple de cuatro funciones controladas por dos líneas. Las funciones son $A \text{ AND } B$, $A \text{ OR } B$, $\text{NOT } B$ y $A + B$	161

3.4. Relación entre la ALU, los registros y el registro de banderas durante una operación.	162
3.5. Diagrama de bloques de una ALU.	162
3.6. Diagrama de bloques de una unidad de control.	164
3.7. <i>Buses</i> del sistema y de entrada/salida.	166
3.8. Dos registros conectados mediante un <i>bus</i> de datos.	168
3.9. Símbolo del circuito triestado.	168
3.10. Estructura básica de una memoria.	170
3.11. Una memoria RAM.	171
3.12. Un sistema de <i>buses</i> para los dispositivos de entrada/salida.	173
3.13. (a) Una palabra de 16 bits con un campo de instrucción de 8 bits y un campo de dirección de 8 bits; (b) una representación más simple de esta palabra.	175
3.14. Un sistema sencillo.	177
3.15. Parte de un decodificador de instrucciones.	183
3.16. Diagrama de bloques de una pequeña computadora digital.	186
3.17. Pulsos de reloj (a) con temporización normal; (b) con temporización más lenta, para usarse con la memoria.	188
3.18. Diagrama de flujo del algoritmo que suma los primeros 100 números enteros.	197
4.1. Una jerarquía de memoria.	220
4.2. Celda binaria en diagrama de bloque e implementación usando flip-flop RS.	225
4.3. Una memoria de cuatro palabras de cuatro bits. La señal de reloj se omite por claridad.	227
4.4. Un decodificador de dirección. Las señales de reloj se omiten.	229
4.5. Conexiones de una ROM.	232
4.6. Modificación de un decodificador de dirección para contar con una terminal <i>chip select</i>	236
4.7. Una memoria de doce palabras construida mediante la paralelización de tres dispositivos de memoria de cuatro palabras.	237
4.8. Diagrama de grabado en una película ferromagnética.	240
4.9. Un disco magnético.	244
5.1. Un ejemplo de <i>pipeline</i> de cinco etapas.	261

5.2.	5 instrucciones en un <i>pipeline</i>	262
5.3.	Procesador con dos <i>pipelines</i>	264
5.4.	Procesador con un <i>pipeline</i> de cuatro pasos.	273
6.1.	Una interface serial genérica.	277
7.1.	Modelo de capas de cebolla mostrando las capas de abstracción que Microsoft usa para su sistema operativo Windows NT.	292
7.2.	Un mapeo de memoria segmentada.	296
7.3.	Un mapeo de memoria paginada.	297

Introducción

¿Qué cursos cubren este libro?

Este libro describe cómo funcionan las computadoras. Está escrito para personas que usan computadoras, pretenden usarlas, o que simplemente están interesadas en general por ellas. Comenzando con las computas más elementales y construyendo hacia una computadora completa, se discuten todas las fases de descripción de computadoras. También se presenta y discute el sistema numérico binario y la forma en que una computadora funciona con tales números.

Este libro se desarrolla para estudiantes que cursan materias introductorias de arquitectura de computadoras como parte de algún plan de estudios en computación. Diferentes instituciones tienden a tomar diferentes puntos de vista respecto a cuestiones referentes al *hardware* y *software* o arquitectura, que debieran cubrirse en los primeros cursos del plan de estudios; sin embargo, la mayoría todavía consideran esta área lo suficientemente importante como para contar con cursos relativos en sus planes de estudios. Estos cursos presentan varios títulos, entre los que se incluye arquitectura de computadoras, sistemas de cómputo, plataformas de cómputo, arquitectura de máquinas, y otros. Frecuentemente, estas frases se combinan con las palabras “introducción a”, o son numeradas (I, II, etc.). Si el estudiante se encuentra inscrito en algún curso cuyo título tiene estas características, entonces su curso tiene un contenido que se traslapa en mucho con el contenido de este libro.

Respecto a su contenido, se consideran los cursos de diseño de sistemas digitales y de arquitectura de computadoras como se han enseñado por los últimos años por el autor; sin embargo, este libro tiene un objetivo más en entender los principios subyacentes detrás del estudio más que demostrar habilidades prácticas

(que son también importantes). Esto parece cubrir un buen número de temas que se dan en cursos similares.

Para los propósitos actuales, una arquitectura o plataforma significa la infraestructura computacional necesaria para soportar la ejecución de un programa o aplicación moderna. Por esta razón, se incluye un tema sobre componentes de software como sistemas operativos.

Además de describir las partes de una computadora, se discute la programación elemental en lenguaje de máquina, y se habla de lenguajes ensamblador y de alto nivel; sin embargo, el objetivo aquí no es la enseñanza de lenguajes específicos, sino proveer de un entendimiento de qué lenguajes hay y cómo operan con la computadora.

Objetivos de aprendizaje

Este libro se desarrolla basado en el concepto de objetivos de aprendizaje por capítulo, es decir, en términos de objetivos, habilidades y conocimientos explícitamente expuestos y que el estudiante debe ser capaz de adquirir como resultado del estudio. En términos generales, tiene objetivos de aprendizaje también, que se presentan a continuación:

1. Ser capaz de utilizar los métodos de diseño y descripción básicos del hardware de computadoras, es decir, el álgebra booleana y los diagramas lógicos a nivel compuertas, para describir la operación de los subsistemas de una computadora von Neumann en su más bajo nivel.
2. Poder describir los subsistemas básicos de hardware de una computadora, que incluyen al procesador, la jerarquía de la memoria, dispositivos periféricos, y los medios con los cuales éstos se interconectan.
3. Ser capaz de describir el impacto de la especificación de diferentes partes de una computadora respecto a su desempeño para diferentes aplicaciones y utilizar esta información para un análisis cuantitativo de cuestiones de desempeño, así como para planear apropiadamente actualizaciones.
4. Poder discutir la función de un sistema operativo para proveer una interfaz de programación y de usuario para programas y aplicaciones, así como poder describir los subsistemas de sistemas operativos modernos.

¿Qué hay en este libro?

El papel principal de los cursos de arquitectura de computadoras al inicio de un plan de estudios en computación es proveer con los conceptos y términos básicos y necesarios para comprender qué sucede dentro de las computadoras digitales, y por lo tanto, representa una gran cantidad de información como antecedentes para otros cursos del plan de estudios que consideran su funcionamiento como básico y garantizado. Para estudiantes de computación, significa dejar de percibir a la computadora como una caja negra, y comprender tal funcionamiento.

El estudio debe hacerse tan actualizado como sea posible, ya que una computadora personal moderna incluye componentes ahora que solo podían encontrarse en supercomputadoras tan solo hace diez años. Aun cuando un completo entendimiento de todos estos temas está sin duda más allá de los objetivos de este libro, contar con el conocimiento básico de cómo trabaja la computadora que está frente al estudiante significa saber (o al menos describir) los temas que se presentan aquí.

Capítulo 1

¿Qué es una computadora?

Resumen

El propósito de este capítulo es establecer el contexto para los siguientes capítulos, mediante proveer una explicación acerca de lo que las computadoras hacen, presentar un breve historia de las computadoras, e introducir los componentes principales de una computadora de propósito general y la manera en que estos componentes interactúan para procesar datos.

Objetivos del capítulo

Al final del capítulo el estudiante debe ser capaz de:

1. Explicar el significado de los términos datos e información.
2. Tener nociones sobre la evolución de la tecnología de computadoras y ser capaz de elaborar sobre el futuro de las computadoras.
3. Enunciar los componentes principales de un sistema de cómputo.
4. Explicar el concepto de programa almacenado.
5. Explicar el papel de la unidad central de proceso, los sistemas de entrada/salida y la memoria dentro de una computadora digital.

1.1. Datos, información y el concepto de programa almacenado

Esta sección comienza aclarando los conceptos de datos e información en el contexto de la computación, e introduce el concepto de programa almacenado (*stored program*).

Tyche Brahe, el astrónomo medieval, pasó su vida adulta observando y registrando la posición de los planetas. Su sucesor, Johannes Kepler, analizó esos registros para producir leyes de movimiento planetario. Brahe conjuntó datos, mientras que las Leyes de Kepler son información.

Procesar datos extrae su significado. Una computadora es una máquina de procesamiento de datos. Los datos fluyen a la máquina como entrada, e información fluye desde la máquina como salida (Figura 1.1).

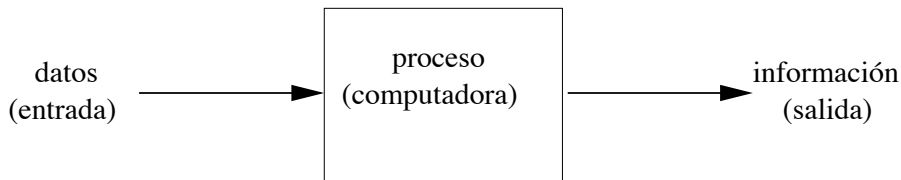


Figura 1.1: Esquema de una computadora como máquina de procesamiento de datos.

¿Qué hace a una computadora diferente a una calculadora, dado que ambas se utilizan para hacer operaciones? Por ejemplo, para sumar dos números con una calculadora se debe:

1. Teclear el primer número.
2. Presionar el botón de suma (+).
3. Teclear el segundo número.

4. Presionar el botón de resultado (=).
5. Registrar el número para referencias posteriores.

La calculadora encuentra el resultado de la suma, pero el usuario debe proveer el control mediante decidir qué botón presionar. A diferencia, una computadora procesa datos automáticamente. Se le debe proveer de un conjunto de instrucciones, en forma de un programa que la guíe. El programa se almacena dentro de la máquina, y por esto, se le conoce con el nombre de programa almacenado.

Una **computadora** es una máquina que procesa datos bajo el control de un programa almacenado a fin de obtener información.

1.1.1. Resumen del desarrollo histórico de las computadoras

Las computadoras digitales originales utilizaban tubos al vacío o bulbos. La introducción del transistor permitió el desarrollo de poderosas computadoras que eran físicamente más pequeñas y más confiables; sin embargo, estas computadoras eran todavía grandes y costosas. El desarrollo de los circuitos integrados cambió todo eso. Muchos transistores y circuitería asociada pueden construirse en un pequeñísimo circuito de silicio. El costo y tamaño de un circuito complejo llegó a ser similar a aquél que utilizaba transistores solo unos años antes. Los circuitos integrados de alta escala han llevado esto un paso más adelante. Actualmente, circuitos de computadora extremadamente complejos pueden construirse en un fragmento de silicio. Esto ha dado como resultado computadoras pequeñas y relativamente baratas.

Los desarrollos tecnológicos permitieron que cada vez un número mayor de personas hicieran uso de las computadoras; sin embargo, tal número de personas era muy limitado. Las computadoras se usaban generalmente para resolver problemas matemáticos complejos de ingeniería o ciencia, o también para realizar nóminas y otras operaciones de procesamiento de datos de grandes compañías.

Una computadora hoy está disponible para un mayor número de personas, y con diversos usos. Se utilizan, por ejemplo, desde el control de inyección de combustible de los automóviles, hasta la dirección de operaciones de casi todas las compañías. Se utilizan también en todo tipo de instrumentos para proveer de

información inmediata que antes no era disponible. A nivel personal, se usan para controlar las finanzas, jugar, y hacer otras operaciones, que se limitan tan solo por la imaginación del usuario.

Para comprender y apreciar el impacto de las computadoras en la vida humana, así como las promesas que representan en el futuro, es importante entender su evolución histórica.

Antes de las computadoras. Máquinas de cómputo iniciales y sus inventores

Algunos precursores de la computación moderna son:

- el ábaco, que emerge hace alrededor de 5000 años en Asia;
- las tablas de navegación, que se comenzaron a utilizar en los años 1600;
- los logaritmos y una regla primitiva de cálculo, que se utilizó para 1617;
- la máquina sumadora construida por Blaise Pascal, hecha en 1642.

En los años 1800, la Revolución Industrial trajo consigo una necesidad mayor de procesamiento de datos, así como una rápida evolución de la tecnología. En 1822, Charles Babbage propone una máquina que realiza ecuaciones diferenciales. Diez años más tarde, Babbage comienza a trabajar en la primera computadora de propósito general, llamada máquina analítica (*Analytical Engine*). Esta máquina cuenta con un programa almacenado, describiendo los elementos básicos de las computadoras modernas.

En 1889, Herman Hollerith construye una máquina para recopilar resultados mecánicamente, accedendo datos almacenados en tarjetas perforadas. La compañía de Hollerith más adelante se convierte en la *International Business Machine* (IBM).

Primera generación de computadoras (1945 - 1956)

Los circuitos electrónicos digitales dentro de la unidad central de proceso y la memoria principal se componen de interruptores que pueden encontrarse encendidos o apagados. Las primeras computadoras realmente utilizaban interruptores controlados eléctricamente, llamados relevadores. Los relevadores tenían dos ventajas: eran relativamente lentos, y ocupaban mucho espacio. Eran necesarios

varios milisegundos para que un relevador cambiara su estado de abierto a cerrado. Al principio, tal velocidad puede parecer muy rápida; sin embargo, durante la ejecución de un programa de computadora común, se requiere abrir y cerrar los relevadores varios miles de millones por segundo, a fin de obtener una respuesta en un tiempo aceptable para el usuario. De tal modo, una computadora basada en relevadores requeriría mucho tiempo para operar. Además, ya que los relevadores ocupan un espacio relativamente grande, las computadoras basadas en relevadores también tendían a ser grandes. La computadora digital moderna era originalmente un dispositivo grande, tanto que requería una gran habitación para contenerla. Además, era cara, costando comúnmente más de un millón de dólares. Debido a su tamaño y costo, las primeras computadoras pertenecían sólo a grandes compañías y universidades, y se utilizaban principalmente para operaciones de procesamiento de datos.

A mediados de la década de 1940, los relevadores fueron reemplazados por tubos al vacío o bulbos, los cuales operaban como interruptores eléctricos controlados por electricidad. Tales bulbos eran más pequeños y rápidos que los relevadores. De hecho, un bulbo rápido puede operar en el orden de microsegundos ($\mu s = 10^{-6}$ segundos). Así, la velocidad de las computadoras se incrementó grandemente; sin embargo, los bulbos tienen una gran desventaja: se funden y deben ser cambiados con mucha frecuencia. Y dado que una computadora de bulbos tenía un gran número de éstos, la posibilidad de la falla de un bulbo durante la operación, y la falla subsecuente de toda la computadora, era muy alta.

A finales de los años 1940, aparecieron las computadoras Colossus en Gran Bretaña y ENIAC en Estados Unidos; sin embargo, ninguna de las dos podía almacenar su propio programa, requiriendo intervención manual constante. En 1945, John von Neumann diseña la *Electronic Discrete Variable Automatic Computer* (o EDVAC) que cuenta con una memoria para almacenar un programa así como datos, y una unidad central de proceso (*Central Processing Unit* o CPU), la cual permite que todas las funciones de la computadora se coordinen a través de este único recurso.

La primera máquina con un programa almacenado se construye en 1948, en la Universidad de Manchester.

La primera generación de computadoras son máquinas grandes que consumen mucha energía, ya que utilizan bulbos y tambores magnéticos para el almacenamiento de datos. Para funcionar, era necesario un ejército de operadores.

Cada computadora cuenta con un lenguaje de máquina (*machine language*) diferente, que le indica cómo operar.

Segunda generación de computadoras (1956 - 1963)

Durante las décadas de 1950 y 1960, se desarrollaron las computadoras que usaban transistores en lugar de bulbos. Los transistores son mucho más pequeños que los bulbos, pueden operar a una velocidad en el orden de nanosegundos ($ns = 10^{-9} \text{segundos}$), y no se funden. Con estos dispositivos, la computadora digital ha llegado a velocidades lo suficientemente razonables como para operar sobre programas cada vez más complejos y largos. Más aun, ya que varios programas relativamente cortos podían ejecutarse muy rápido, se desarrolló el uso práctico de tiempo compartido (*time sharing*), en el cual muchos usuarios aparentan utilizar la computadora simultáneamente. En realidad, en una computadora con tiempo compartido, solo una persona realmente utiliza la computadora en un momento dado; sin embargo, la computadora reparte su tiempo tan rápido entre los usuarios que ninguno de ellos nota diferencia alguna.

En la segunda generación de computadoras, los transistores reemplazan a los bulbos. Junto con avances en memorias de núcleo magnético (*magnetic-core memory*), los transistores producen computadoras más pequeñas, rápidas, confiables y más eficientes en términos de energía que sus predecesoras. IBM construye las primeras supercomputadoras, capaces de manejar una gran cantidad de datos. El lenguaje ensamblador es utilizado en lugar del lenguaje máquina, lo que facilita la programación.

La segunda generación de computadoras contiene todos los componentes asociados con las computadoras modernas de hoy: impresoras, almacenamiento en cintas, almacenamiento en disco, memoria, sistema operativo y programas almacenados. Dos lenguajes de alto nivel, COBOL (*COmmon Business Oriented Language*) y FORTRAN (*FORmula TRANslation*) se volvieron de uso común. Comenzaron nuevos tipos de empleos (programador, analista y experto en sistemas de cómputo) así como una industria de desarrollo de software.

Tercera generación de computadoras (1964 - 1971)

En 1958, varios componentes electrónicos se combinaron en un pequeño *chip* de silicio, dando origen al circuito integrado y paso a la tercera generación de computadoras. Los sistemas operativos permitían a las computadoras ejecutar varios pro-

gramas concurrentemente, mediante un programa central que monitorea y coordina los recursos de la computadora.

El desarrollo de los circuitos integrados redujo todavía más el tamaño de las computadoras, a un costo mucho más bajo. Tales circuitos integrados son dispositivos semiconductores en los cuales, mediante técnicas ópticas, muchos miles de transistores se fabrican en una oblea de silicio. Los circuitos integrados son componentes que no requieren alambrarse a mano, como era el caso de los bulbos o los transistores. Consecuentemente, el costo y mano de obra para producir una computadora digital descendieron, y la fabricación de computadoras cada vez más rápidas y baratas es hoy una realidad.

El primer circuito integrado tenía solo unos cuantos componentes como transistores y resistencias. Debido a esto, a este tipo de circuito se le conoce hoy en día como circuito SSI (*Small Scale of Integration*). Poco tiempo después, se produjeron otros circuitos con un número de 50 a 100 componentes, llamados MSI (*Medium Scale of Integration*). La fabricación de miles de componentes en un solo circuito integrado dio lugar a los sistemas LSI (*Large Scale of Integration*), lo cual representó un gran paso en el desarrollo de la computación electrónica. Finalmente, se han desarrollado a últimas fechas los circuitos integrados VLSI (*Very Large Scale of Integration*), donde decenas o cientos de miles de componentes en un solo circuito los hace lo suficientemente poderosos como para contener un sistema de cómputo. De hecho, el microprocesador es un ejemplo de un circuito VLSI que representa una computadora en un solo circuito integrado, realmente constituido por una unidad de control, una unidad lógico-aritmética y algunos registros.

Cuarta generación de computadoras (1971 - presente)

La unidad central de proceso se desarrolla como un solo circuito integrado: un microprocesador. Para 1975 las primeras computadoras personales se venden para aficionados. En 1981, IBM vende la primera computadora personal (*Personal Computer* o PC). El número de computadoras se incrementa de 2 millones en 1981 a 5.5 millones en 1982, hasta alcanzar la cifra de 65 millones diez años después. Estas computadoras continúan su tendencia hacia un menor tamaño, llegando a las computadoras de escritorio (*desktop*), de regazo (*laptop*), de palma (*palmtop*). Año con año, la velocidad se incrementa, y el precio cae.

Las computadoras se hacen más comunes en los trabajos. Pueden conectarse entre sí o en red, para compartir espacio de memoria, *software* o información, y

comunicarse entre ellas. Opuesto a la computadora central (*mainframe*), que es una gran y poderosa computadora que compartía su tiempo entre varias terminales para muchas aplicaciones, las redes de computadoras permiten a las computadoras individuales una forma de cooperación electrónica. Utilizando un cableado directo, llamado red de área local (*Local Area Network* o LAN), o líneas telefónicas, estas redes llegan a tomar proporciones enormes. Una gran red global de computadoras, por ejemplo Internet, liga computadoras alrededor del mundo, como una sola fuente de información.

Quinta generación de computadoras (presente y más allá)

La quinta generación de computadoras se encuentra todavía en desarrollo. Algunas diferencias cualitativas respecto a la cuarta generación pueden incluir varios aspectos de la inteligencia artificial, en que las computadoras puede usar lenguaje natural, aplicar razonamientos deductivos, aprender por experiencia, ver y reconocer objetos, y evolucionar para cumplir requerimientos.

Un elemento que permite que estas capacidades se desarrollen pudiera ser el procesamiento paralelo, que combina muchos procesadores juntos para proveer un enorme procesamiento de datos.

1.1.2. La computadora y las cosas que se pueden hacer con ella

Considérese una computadora en operación. Un ingeniero electrónico, por ejemplo, le suministra las especificaciones de un amplificador de audio de alta fidelidad, y en segundos, la computadora imprime los componentes del amplificador. O tómese al usuario que juega ajedrez con su computadora. Para toda movida que hace, la computadora responde con un movimiento. Tal proceso continúa hasta que el juego termina, muchas veces ganando la computadora. En otro caso aún, el gerente de un departamento de crédito le proporciona una lista de números de cuenta de clientes, así como sus compras y pagos del mes. La computadora se encarga entonces de enviar a cada cliente su estado de cuentas y cobros, basándose en sus compras y pagos, el balance anterior y los cargos por servicios.

En todos estos ejemplos parece como si la computadora acepta datos, “piensa” y produce una salida. En realidad, un proceso algo diferente toma lugar.

La computadora no piensa. Debe ser **programada**, es decir, debe ser dirigida para realizar un conjunto específico de operaciones. En todos los ejemplos anteriores, tal programa (o secuencia de instrucciones) ha sido almacenado previamente en la computadora.

Las computadoras actuales pueden almacenar una gran cantidad de información. Tal información consiste tanto de programas que pueden ejecutarse, como de datos sobre los cuales se ejecutan los programas. En el ejemplo del departamento de crédito, los datos almacenados pueden ser el número de cuenta, nombre, balance de crédito, cargo por servicio, pagos y compras. Además, también es necesario tener almacenado un conjunto de instrucciones (o programa) que dirige la acción de la computadora, a fin de, por ejemplo, imprimir los recibos y estados de cuenta.

El **programa** puede dirigir a la computadora para realizar, paso a paso, complejas operaciones, usando tan solo operaciones aritméticas básicas (suma, resta, multiplicación y división) y algunos otros relativamente simples procedimientos.

Por ejemplo, dos números pueden compararse y la computadora puede determinar si son iguales, o si uno es mayor que el otro. Usando procedimientos sencillos, una computadora puede hacer que se realicen operaciones complejas, produciendo la apariencia de que algún proceso de pensamiento se lleva a cabo. Por ejemplo, si el balance de pago de un cliente excede una cierta cantidad, se puede añadir un mensaje de advertencia para el cliente.

Aun cuando aquí se busca explicar cómo funciona una computadora, por otro lado no se discute en detalle cómo puede ser programada. Para esto, se sugiere al lector referirse a alguno de los tantos libros sobre programación disponibles.

La introducción tecnológica del microprocesador ha permitido reducir el tamaño, y en la actualidad, el costo de las computadoras, que pueden ser adquiridas por personas individuales para utilizarlas en su trabajo o entretenimiento. Estas computadoras personales pueden utilizarse para jugar una variedad de juegos como el ajedrez, o algún otro que haga al usuario imaginar que viaja por el espacio o explora cavernas y laberintos.

El propietario de una computadora personal puede usarla tal y como una gran compañía utiliza una computadora grande para, por ejemplo, llevar control de sus finanzas domésticas, impuestos y presupuestos. Llevado en ocasiones al extremo, la computadora puede utilizarse para controlar varias otras actividades en el hogar. Por ejemplo, la calefacción puede controlarse, de tal modo que se logren ahorros en el consumo de energía. Una computadora puede automáticamente leer información de sensores, a fin de que las alarmas contra incendio o ladrones se hagan más sensibles: la más pequeña aparición de luz en una habitación oscura durante la noche puede automáticamente disparar una alarma.

Muchas de las aplicaciones de la computadora se encuentran en las áreas de ingeniería, ciencia y matemáticas, donde es frecuente la necesidad de resolver conjuntos de ecuaciones simultáneas con muchas incógnitas. Si esto se hiciera a mano (o con una simple calculadora), la solución requeriría días, o hasta meses. Dependiendo de la complejidad del cálculo, la computadora puede realizarlo en minutos, o tal vez, hasta en segundos. Hay, por supuesto, muchas otras aplicaciones de esta naturaleza. Cuando ondas de radar rebotaron en la superficie de Venus, la señal de retorno era tan débil que no podía distinguirse del ruido o la interferencia. Las computadoras se utilizaron entonces para realizar complejos cálculos, que permitieron distinguir las señales del ruido. También, las computadoras se utilizan ampliamente en los negocios a fin de llevar a cabo labores contables, como inventarios, balances y operaciones diarias de envío y recepción de mercancías.

1.1.3. Ideas básicas sobre computadoras digitales

Las computadoras digitales operan sobre números. A veces, estos números son **códigos** que representan instrucciones de un programa, o datos tales como el nombre de una persona.

Por ejemplo, si se introduce el nombre de una persona a la computadora, debe convertirse primero en un conjunto de códigos numéricos apropiados. Más aún, los números pueden representar también números verdaderos.

Si el objetivo es entender cómo funciona una computadora, es necesario conocer su sistema numérico. En general, los seres humanos trabajamos con un sistema numérico decimal que utiliza diez dígitos. Se considera que este sistema se desarrolló debido a que las personas normalmente cuentan con diez dedos; sin

embargo, las computadoras son esencialmente un conjunto de interruptores que se encuentran en uno de dos estados: encendido (*on*) o apagado (*off*). Así, los componentes de una computadora trabajan con un sistema numérico de dos dígitos, llamado sistema numérico binario, el cual usa 0 y 1 para representar los estados de apagado y encendido respectivamente.

Considérese ahora cómo es la organización interna de una computadora digital. En los capítulos subsecuentes, las ideas a continuación se presentan con mayor detalle; sin embargo, parece necesario aclarar que la terminología usada para describir la organización interna de una computadora digital no está completamente estandarizada, de tal modo que la descripción aquí utiliza terminología común utilizada para computadoras pequeñas. A continuación, se describe brevemente cada uno de sus componentes genéricos.

1.2. Los subsistemas básicos de la computadora

Esta sección discute los subsistemas básicos de una computadora, mostrando brevemente cómo se conectan los componentes, cuáles son sus funciones, y cómo se relacionan para formar una computadora del tipo personal (PC).

1.2.1. Componentes del sistema: componentes básicos

Los sistemas de cómputo consisten de una unidad central (que a veces es referida como la computadora propiamente) y varios periféricos. La unidad central, que contiene la mayoría de los componentes electrónicos, se conecta mediante cables a los periféricos.

John von Neumann propone dividir el hardware de computadora en las siguientes cinco partes principales:

- unidad central de proceso;
- entrada;
- salida;
- memoria de trabajo;
- memoria permanente.

La **unidad central de proceso** (CPU) es el componente más importante de un sistema de cómputo. Bajo el control de programas almacenados, esta unidad manipula los datos y almacena los resultados en memoria.

La CPU continuamente recibe instrucciones a ser ejecutadas. Cada instrucción es una orden para procesar datos. El trabajo en sí consiste en su mayoría en operaciones y transferencia de datos. Los datos provienen de la memoria de trabajo (normalmente RAM), memoria permanente (discos) y dispositivos periféricos (por ejemplo, teclado). Después de su procesamiento, los datos se envían de nuevo a la memoria de trabajo o a los dispositivos de salida.

Normalmente, la CPU es un circuito único o microprocesador, en el que se conjuntan la unidad lógico aritmética (*Arithmetic Logic Unit* o ALU) y la unidad de control (*Control Unit* o CU); sin embargo, tal terminología no es completamente estándar, ya que en algunas computadoras ambas unidades aparecen como elementos independientes de una computadora.

- la unidad lógico aritmética se encarga de realizar todas las operaciones lógicas y aritméticas definidas sobre los valores numéricos binarios. Por ejemplo, dos números binarios pueden sumarse para producir un tercero;
- la unidad de control dirige la operación de la computadora. Es el elemento que se encarga de controlar el flujo de instrucciones que la computadora va llevando a cabo. Por ejemplo, puede dar las instrucciones a la ALU para la suma de los dos números binarios.

1.2.2. Memoria y discos

Todos los datos y los programas cuyas instrucciones los modifican se encuentran almacenados en la memoria de la computadora digital.

La **memoria** de una computadora almacena tanto datos como programas.

Todos los datos que una computadora utiliza u opera durante el procesamiento se almacena aquí. Cuando el CPU trabaja con los datos que están almacenados

en discos (normalmente, discos duros), tales datos deben primero enviarse a la memoria de trabajo. En realidad, en una computadora digital, existen varios tipos de unidades de memoria. Por ejemplo, la unidad principal de memoria (*Main Memory Unit* o MMU) es el almacenamiento principal y de trabajo de una computadora. Es en ella donde se encuentran los datos, así como las instrucciones de los programas que se ejecutan sobre tales datos. Normalmente, en computadoras modernas la memoria principal consiste de circuitos semiconductores.

Un tipo particular de memoria semiconductora es la memoria de sólo lectura (*Read Only Memory* o ROM). Este tipo de memoria contiene datos permanentemente almacenados, como por ejemplo, podría ser una tabla de valores de funciones trigonométricas. Tales datos generalmente, aunque no siempre, se incorporan a la memoria durante su manufactura.

Otro tipo particular de memoria semiconductora, pero con mayor versatilidad en su uso durante la ejecución de instrucciones, es la memoria de acceso aleatorio (*Random Access Memory* o RAM). Esta memoria es capaz de modificar su estado mientras permanezca encendida, permitiendo tanto la lectura como la escritura de valores numéricos binarios al ejecutar las instrucciones de un programa.

Además de la memoria principal, una computadora digital cuenta también con unidades de memoria auxiliar. Éstas son capaces de almacenar gran cantidad de información en componentes magnéticos, como son discos o cintas; se consideran de bajo costo en el sentido de que permiten almacenar muchos datos a un costo razonable; sin embargo, las operaciones para escribir y leer información toman mucho más tiempo comparado con el de acceso de la memoria principal semiconductora.

1.2.3. Dispositivos periféricos

Una computadora sería inútil a menos que pudieran comunicarse con el usuario, a fin de que éste proporcionase información a ser procesada y recibiese los resultados de tal procesamiento. ésta es la función de los dispositivos de entrada/salida (*Input/Output* o I/O). Un dispositivo de salida típico es el monitor de una computadora personal, mientras que un dispositivo de entrada típico es el teclado de la misma. El usuario teclea los símbolos sobre el teclado, que genera las señales eléctricas, en forma de ceros y unos, para introducirlas a la computadora.

De forma similar, otras señales eléctricas se envían de la computadora al monitor, de tal modo que éste muestra la información que el usuario teclea.

Otra forma de dispositivo de salida es la impresora, que puede imprimir la información de la computadora en una hoja de papel, a fin de que pueda ser examinada por el usuario. Existe un gran número de tipos de impresoras, que operan a diferentes velocidades y resoluciones, permitiendo que grandes cantidades de datos puedan ser analizados.

Típicamente, algo más que el 80 % del costo de una computadora se debe a los dispositivos periféricos.

Un **dispositivo periférico** se encuentra alrededor de la computadora, a diferencia del procesador central o la memoria de trabajo, las cuales conforman la propia computadora.

El número y tipo de dispositivos periféricos dependen de la aplicación principal que tenga el sistema de cómputo. Una pequeña computadora personal puede tener apenas unos cuantos dispositivos, tales como teclado, monitor y una impresora. Un sistema comercial grande puede tener muchos dispositivos conectados, y muchos componentes de un mismo dispositivo. Generalmente, los dispositivos periféricos se clasifican como:

- dispositivos de entrada. Proveen facilidades para la entrada de datos. Ejemplo: teclado;
- dispositivos de salida. Proveen al usuario con datos y/o información procesada. Ejemplos: monitor e impresora.

Con esta descripción, un sistema de cómputo puede describirse como se muestra en la Figura 1.2.

1.2.4. *Buses*

Los *buses* se encargan de conectar a la unidad central de proceso con todos los demás componentes de la computadora. La computadora recibe y envía datos a través de los *buses*. Normalmente, los *buses* se clasifican como:

- *bus* del sistema, que conecta a la unidad central de proceso con la memoria de trabajo RAM;

- *buses* de entrada/salida, que conectan a la unidad central de proceso con otros componentes.

El *bus* del sistema es la conexión más importante dentro de la computadora, en tanto que los *buses* de entrada/salida se encargan de transportar datos, conectando los dispositivos de entrada/salida con la unidad central de proceso y la memoria.

El **ciclo de entrada/proceso/salida** es la secuencia básica mediante la cual una computadora procesa datos.

Un ejemplo de los pasos que deben realizarse por una computadora para realizar una tarea se presenta a continuación:

1. Almacenar un programa en memoria (dado el programa, la unidad central de proceso puede comenzar a procesar datos).
2. Aceptar datos de entrada (desde el teclado, por ejemplo) y almacenarlos en memoria.
3. La unidad central de proceso manipula los datos y almacena los resultados de nuevo en memoria.

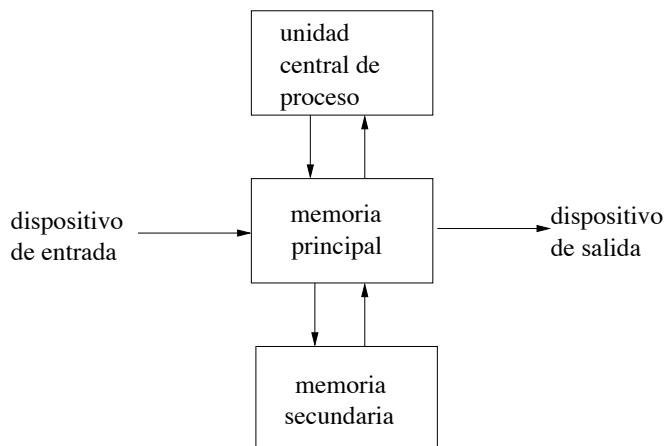


Figura 1.2: Esquema de una computadora.

4. Los resultados se entregan al dispositivo de salida (por ejemplo, pantalla o impresora).

Los componentes físicos de una computadora (unidad central de proceso, memoria y dispositivos de entrada/salida) se conocen con el nombre genérico de **hardware**. Los programas y datos sólo existen como pulsos electrónicos que se encuentran en la memoria, y se les conoce con el nombre genérico de **software**. Una serie de instrucciones que realiza una tarea en particular se le llama **programa**.

Las dos categorías más importantes en que se clasifica el *software* son: *software* de sistema y *software* de aplicación. El *software* del sistema se conforma de los programas de control como el sistema operativo y los sistemas de administración de bases de datos.

Un error común es considerar que el *software* son sólo datos. El *software* le indica al *hardware* qué debe hacer y cómo debe proceder para procesar datos. El *software* se ejecuta. Los datos se procesan.

El **software del sistema** se refiere a los programas que se utilizan para controlar la computadora y desarrollar y ejecutar *software* de aplicación (o simplemente, aplicaciones).

El **software de aplicación** es cualquier entrada de datos, actualización, solicitud, reporte o otro programa que procesa datos para el usuario. El *software* de aplicación incluye los paquetes genéricos de *software* (hojas de cálculo, procesadores de palabras, programas de bases de datos, etc.) así como programas de paquetería o hechos a la medida como nóminas, pagos, inventarios, y otros.

El término *hardware* se refiere a la maquinaria y equipo (unidad central de proceso, discos, cintas, etc.). El *hardware* se encarga del almacenamiento y la transmisión. Mientras más memoria y almacenamiento en disco tenga una computadora,

más trabajo puede realizar. Mientras más rápida sea la memoria y los discos para transmitir datos e instrucciones a la unidad central de proceso, más rápido se realiza el trabajo.

1.2.5. Estudiando la arquitectura de sistemas de cómputo

Lo anterior es una panorámica general de un sistema de cómputo y cómo éste opera. Ahora es posible comenzar a revisar cada uno de los componentes principales en detalle.

La Figura 1.3 muestra cómo se construyen diferentes niveles de estudio y abstracción sobre descripciones más simples. Aquí, de abajo hacia arriba, se comienza describiendo los elementos lógicos que forman los bloques básicos de construcción del *hardware* de un sistema de cómputo, y que se combinan para obtener una máquina simple de cómputo. Después de esto, se provee alguna información introductoria de los sistemas operativos. Los dos últimos niveles más altos en el estudio de sistemas de cómputo, lenguajes de programación de alto nivel y aplicaciones, no forman parte del objeto de estudio de este libro.

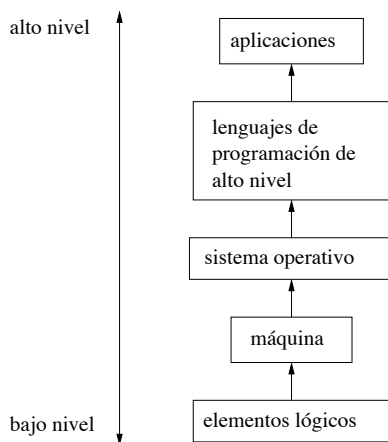


Figura 1.3: Niveles de estudio y abstracción de un sistema de cómputo.

Fuentes útiles de información

The Computer Language Co. Inc.

Computer Desktop Encyclopedia.

<http://www.ComputerEncyclopedia.com>

Evaluación

Preguntas de opción múltiple

1. Una computadora procesa datos:
 - a) Bajo control directo humano.
 - b) Bajo el control de un programa almacenado.
 - c) Al azar.
 - d) Usando su propia inteligencia.
 - e) Ninguno de los anteriores.
2. ¿Qué distingue a una computadora de otros dispositivos de cálculo?
 - a) El gran número de teclas.
 - b) La existencia de un monitor.
 - c) El programa almacenado.
 - d) El tipo de información.
 - e) Windows.
3. La segunda generación de computadoras se hace posible por:
 - a) La invención del transistor.
 - b) El desarrollo del microprocesador.
 - c) El desarrollo de circuitos integrados.
 - d) La fundación de IBM.
 - e) La invención de lenguajes de alto nivel.

-
4. Los lenguajes de alto nivel comenzaron a utilizarse en:
 - a) La primera generación de computadoras.
 - b) La segunda generación de computadoras.
 - c) La tercera generación de computadoras.
 - d) La cuarta generación de computadoras.
 - e) La quinta generación de computadoras.
 5. Los sistemas operativos comenzaron a utilizarse en:
 - a) La primera generación de computadoras.
 - b) La segunda generación de computadoras.
 - c) La tercera generación de computadoras.
 - d) La cuarta generación de computadoras.
 - e) La quinta generación de computadoras.
 6. CPU son siglas en inglés de:
 - a) Unidad central de impresión.
 - b) Unidad central de proceso.
 - c) Unidad de proceso comercial.
 - d) Unidad comercial de impresión.
 - e) Unidad de caracteres de impresión.
 7. ¿Cuál de los siguientes es un dispositivo de entrada de un sistema de cómputo?
 - a) Pantalla.
 - b) Impresora.
 - c) Modem.
 - d) Micrófono.
 - e) Cámara.
 - f) Teclado.

-
8. ¿Cuál de los siguientes es un dispositivo de salida de un sistema de cómputo?
- a) Escáner.
 - b) Cámara digital.
 - c) Modem.
 - d) Bocinas.
 - e) Disco duro.
 - f) Unidades ópticas.
9. La memoria de trabajo de un sistema de cómputo es:
- a) El disco duro.
 - b) La RAM.
 - c) El disco flexible.
 - d) El CD-ROM.
 - e) La CPU.
10. ¿Cuál de los siguientes dispositivos representa el centro de un sistema de cómputo?
- a) La CPU.
 - b) El CD-ROM.
 - c) La memoria.
 - d) La pantalla.
 - e) El disco duro.
11. Los dispositivos periféricos son:
- a) Parte del procesador principal de la computadora.
 - b) Distintos del procesador central y la memoria principal.
 - c) Parte de la memoria principal.
 - d) Parte del *software*.
 - e) Dispositivos emulados por *software*.

12. El bus de entrada/salida conecta:

- a) A la CPU con la RAM.
- b) A la CPU con los dispositivos periféricos.
- c) A la RAM con los dispositivos periféricos.
- d) La estación de autobuses con la estación de trenes.
- e) Ninguno de los anteriores.

Respuestas a las preguntas de opción múltiple

- 1. *b*
- 2. *c*
- 3. *a*
- 4. *b*
- 5. *c*
- 6. *b*
- 7. *c, d, e, f*
- 8. *c, d*
- 9. *b*
- 10. *a*
- 11. *b*
- 12. *b*

Capítulo 2

Lógica y secuenciación

Resumen

Este capítulo cubre el álgebra booleana y el sistema de numeración binario, las compuertas lógicas fundamentales y tablas de verdad, así como el análisis y la síntesis de circuitos digitales simples. Se discuten tanto circuitos combinacionales como secuenciales. Las máquinas de estado finito se presentan como medios para el diseño de circuitos secuenciales.

Objetivos del capítulo

Al final del capítulo, el estudiante debe ser capaz de:

1. Entender la relación entre los sistemas numéricos decimal y binario, cómo representar número decimales en binario, y porqué los números binarios se utilizan en sistemas de cómputo.
2. Reconocer los símbolos de las compuertas lógicas básicas y definir por medio de tablas de verdad las operaciones lógicas básicas.
3. Usar identidades básicas del álgebra booleana para manipular expresiones lógicas.

4. Analizar la funcionalidad de circuitos digitales simples mediante tablas de verdad, así como sintetizar circuitos digitales simples partiendo de su tabla de verdad.
5. Entender los bloques principales combinacionales o secuenciales, y su uso en sistemas de cómputo.
6. Familiarizarse con los conceptos básicos de máquinas de estado finito y su papel en el diseño de circuitos secuenciales.

2.1. Sistemas numéricos

En el capítulo anterior se menciona que las computadoras digitales trabajan con un sistema numérico binario que utiliza los dígitos 0 y 1. En esta sección se discuten las ideas básicas de tal sistema numérico. Además, se mencionan otros dos sistemas numéricos que realmente no se utilizan por las computadoras digitales, pero que son extremadamente convenientes para el uso de las personas que trabajan con ellas.

2.1.1. Ideas básicas de los sistemas numéricos

El sistema numérico que se utiliza en la vida diaria se basa en diez símbolos separados $(0, 1, \dots, 9)$ y se conoce como sistema numérico decimal. Un dígito es una posición en la representación de un número, y puede tomar un valor entre 0 y 9.

En todo sistema numérico posicional, los dígitos se combinan para formar grupos ordenados mediante una **notación posicional** para crear grandes números. Eso significa que cada dígito se multiplica por diferentes potencias de la base de acuerdo a su posición dentro de la representación del número.

El sistema decimal es base 10, y es el más familiar para los seres humanos; sin embargo, los sistemas numéricos pueden formarse de cualquier número de símbolos, siempre y cuando tal número sea mayor que uno.

El número de símbolos que un sistema numérico utiliza se conoce como **base** del sistema.

Aquí, se consideran *(a)* el sistema binario o base 2, *(b)* el sistema octal óo base 8 y *(c)* el sistema hexadecimal o base 16.

Para comenzar la discusión, considérese la siguiente tabla, que compara parte de los sistemas numéricos de interés. Nótese que el sistema hexadecimal requiere de 16 símbolos, por lo que por convención se utilizan las letras A, B, C, D, E y F para representar los dígitos de los valores decimales 10, 11, 12, 13, 14 y 15, respectivamente. La tabla muestra los valores numéricos de cada base entre 0 y 20 decimal.

decimal	binario	octal	hexadecimal
0	00000	0	0
1	00001	1	1
2	00010	2	2
3	00011	3	3
4	00100	4	4
5	00101	5	5
6	00110	6	6
7	00111	7	7
8	01000	10	8
9	01001	11	9
10	01010	12	A
11	01011	13	B
12	01100	14	C
13	01101	15	D
14	01110	16	E
15	01111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

Para saber qué valor representa un número, es necesario primero conocer su base. Para especificar la base se hace uso de un subíndice. Por ejemplo, de la tabla anterior, se tiene que:

$$12_{10} = 1100_2 = 14_8 = C_{16}$$

Nótese que el subíndice siempre se escribe en base 10. Comúnmente, cuando se conoce la base que se utiliza, normalmente el subíndice puede omitirse.

Se discute a continuación algunas ideas básicas sobre los sistemas numéricos. Se comienza con la base 10, que es la más conocida, para tratar después las otras bases. También, se inicia con números enteros, ya que posteriormente se tratan los números fraccionarios.

Considérese el número 293_{10} . Es sencillo notar que este número decimal se encuentra compuesto por:

$$2 \text{ centenas} + 9 \text{ decenas} + 3 \text{ unidades}$$

Es posible expresar esto en forma más compacta utilizando las siguientes equivalencias:

$$\begin{aligned} 10^0 &= 1 \\ 10^1 &= 10 \\ 10^2 &= 10 \times 10 = 100 \\ 10^3 &= 10 \times 10 \times 10 = 1000 \\ 10^4 &= 10 \times 10 \times 10 \times 10 = 10000 \end{aligned}$$

Por tanto, se puede escribir:

$$293_{10} = (2 \times 10^2) + (9 \times 10^1) + (3 \times 10^0)$$

lo que significa que 293_{10} representa 2 centenas, 9 decenas y 3 unidades. De manera similar:

$$32864_{10} = (3 \times 10^4) + (2 \times 10^3) + (8 \times 10^2) + (6 \times 10^1) + (4 \times 10^0)$$

o 3 decenas de millar, 2 millares, 8 centenas, 6 decenas y 4 unidades.

Ejemplo 2.1 2875_{10} tiene cuatro dígitos; el 5 está en el espacio de las unidades (o 10^0), el 7 en el espacio de las decenas (o 10^1), el 8 en el espacio de las centenas (o 10^2), y el 2 en el espacio de las unidades de millar o miles (o 10^3).

El número 2875_{10} puede por tanto escribirse en forma extendida:

$$2875_{10} = (2 \times 10^3) + (8 \times 10^2) + (7 \times 10^1) + (5 \times 10^0)$$

Cada posición representa una multiplicación de la potencia de 10 por el dígito en esa posición, y cada nueva posición representa la multiplicación por la siguiente potencia de 10, comenzando con el primer dígito a la derecha, donde la potencia de 10 es 0.

La regla general para representar números en sistema decimal usando notación posicional es como sigue:

$$\begin{aligned} a_{n-1}a_{n-2} \dots a_2a_1a_0.a_{-1}a_{-2} \dots a_{-(m-1)}a_{-m} = \\ (a_{n-1} \times 10^{n-1}) + (a_{n-2} \times 10^{n-2}) + \dots \\ + (a_2 \times 10^2) + (a_1 \times 10^1) + (a_0 \times 10^0) \\ + (a_{-1} \times 10^{-1}) + (a_{-2} \times 10^{-2}) + \dots \\ + (a_{-(m-1)} \times 10^{-(m-1)}) + (a_{-m} \times 10^{-m}) \end{aligned}$$

donde n es el número de dígitos a la izquierda del punto decimal, y m es el número de dígitos a la derecha del punto decimal.

2.1.2. Sistema numérico binario

El sistema utilizado en las computadoras para representar y procesar datos e información es el **sistema numérico binario**, o base 2. Cualquier número puede representarse usando dos símbolos: 0 y 1, mediante utilizar la misma notación posicional como en el sistema decimal.

En cuanto al sistema numérico binario, y basándose en las mismas ideas excepto que la base es 2, se tienen las siguientes potencias:

$$\begin{aligned}2^0 &= 1 \\2^1 &= 2 \\2^2 &= 2 \times 2 = 4 \\2^3 &= 2 \times 2 \times 2 = 8 \\2^4 &= 2 \times 2 \times 2 \times 2 = 16 \\2^5 &= 2 \times 2 \times 2 \times 2 \times 2 = 32 \\2^6 &= 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 64\end{aligned}$$

Por lo tanto, se puede escribir que:

$$101_2 = (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$

Similarmente, se tiene que:

$$110101_2 = (1 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 53_{10}$$

Sin embargo, ¿cómo se sabe qué valor numérico tiene un número binario, como por ejemplo, 1101_2 ? De hecho, esta notación es igual a la utilizada en el ejemplo anterior, como se muestra enseguida:

Ejemplo 2.2

$$1101_2 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 13_{10}$$

Cada dígito binario representa el valor de una potencia de 2 que se va incrementando de derecha a izquierda.

Un dígito binario se conoce como **bit** (*binary digit*).

Ejemplo 2.3 Contando en binario:

decimal	binario
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

Nótese que, en las primeras dos líneas de la tabla, los números 0 y 1 son iguales en los sistemas decimal y binario. A partir de la tercera línea, el número 2 decimal tiene una representación binaria con un acarreo: si un bit tiene un valor de 1 y se le suma 1, el bit actual toma el valor 0 y el siguiente bit a la izquierda toma el valor de 1. Entonces, se requieren dos bits para representar números entre 0 y 3, tres bits para números entre 0 y 7, etc.

¿Cuántos bits se necesitan? La regla general es: **con n bits se pueden representar los números del 0 al $(2^n - 1)$.**

En un sistema de cómputo, un grupo de 8 bits se conoce como *byte* u octeto. Siguiendo la regla anterior, un byte puede representar números del 0 al 255. Los bytes se utilizan frecuentemente para contener caracteres individuales en un documento de texto. En el conjunto de caracteres ASCII, se asigna a cada caracter un valor binario entre 0 y 127. Los otros 128 valores se utilizan para caracteres especiales, como acentos, que son comunes en muchos lenguajes.

Para representar grandes números binarios, se utilizan los prefijos kilo ($2^{10} = 1024 = 1\text{k}$), mega ($2^{20} = 1048576 = 1\text{M}$), giga ($2^{30} = 1073741824 = 1\text{G}$), etc. Considérese la siguiente tabla:

	1k = 1024	1M = 1048576	1G = 1073741824	...
$2^0 = 1$	$2^{10} = 1\text{k}$	$2^{20} = 1\text{M}$	$2^{30} = 1\text{G}$...
$2^1 = 2$	$2^{11} = 2\text{k}$	$2^{21} = 2\text{M}$	$2^{31} = 2\text{G}$...
$2^2 = 4$	$2^{12} = 4\text{k}$	$2^{22} = 4\text{M}$	$2^{32} = 4\text{G}$...
$2^3 = 8$	$2^{13} = 8\text{k}$	$2^{23} = 8\text{M}$	$2^{33} = 8\text{G}$...
$2^4 = 16$	$2^{14} = 16\text{k}$	$2^{24} = 16\text{M}$	$2^{34} = 16\text{G}$...
$2^5 = 32$	$2^{15} = 32\text{k}$	$2^{25} = 32\text{M}$	$2^{35} = 32\text{G}$...
$2^6 = 64$	$2^{16} = 64\text{k}$	$2^{26} = 64\text{M}$	$2^{36} = 64\text{G}$...
$2^7 = 128$	$2^{17} = 128\text{k}$	$2^{27} = 128\text{M}$	$2^{37} = 128\text{G}$...
$2^8 = 256$	$2^{18} = 256\text{k}$	$2^{28} = 256\text{M}$	$2^{38} = 256\text{G}$...
$2^9 = 512$	$2^{19} = 512\text{k}$	$2^{29} = 512\text{M}$	$2^{39} = 512\text{G}$...

Cada renglón de la tabla representa el mismo valor decimal equivalente a la potencia de 2, mientras que las columnas representan las potencias con incrementos de 10 en 10. De tal modo, utilizando los prefijos, es posible estimar la cantidad de bits que se representan.

2.1.3. Sistemas octal y hexadecimal

Los sistemas numéricos octal (base 8) y hexadecimal (base 16) proveen de una representación más corta y clara para números con múltiples bits en un sistema digital, dado que sus bases son potencias de 2. El sistema octal utiliza los dígitos del 0 al 7 del sistema decimal, dado que solo requiere 8 dígitos para representar números. El sistema hexadecimal utiliza los dígitos del 0 al 9 y las letras A, B, C, D, E y F, dado que requiere de 16 símbolos para representar cualquier número.

En cuanto a que los sistemas octal y hexadecimal son posicionales, las ideas son muy similares a los sistemas decimal y binario. Utilizando las siguientes equivalencias:

$$\begin{aligned}
 8^0 &= 1 \\
 8^1 &= 8 \\
 8^2 &= 8 \times 8 = 64 \\
 8^3 &= 8 \times 8 \times 8 = 512 \\
 8^4 &= 8 \times 8 \times 8 \times 8 = 4096 \\
 8^5 &= 8 \times 8 \times 8 \times 8 \times 8 = 32768
 \end{aligned}$$

$$\begin{aligned}16^0 &= 1 \\16^1 &= 16 \\16^2 &= 16 \times 16 = 256 \\16^3 &= 16 \times 16 \times 16 = 4096 \\16^4 &= 16 \times 16 \times 16 \times 16 = 65536\end{aligned}$$

Se tiene, por ejemplo, que:

$$\begin{aligned}731_8 &= (7 \times 8^2) + (3 \times 8^1) + (1 \times 8^0) = 473_{10} \\4132_8 &= (4 \times 8^3) + (1 \times 8^2) + (3 \times 8^1) + (2 \times 8^0) = 2138_{10} \\1E2_{16} &= (1 \times 16^2) + (14 \times 16^1) + (2 \times 16^0) = 482_{10} \\263A_{16} &= (2 \times 16^3) + (6 \times 16^2) + (3 \times 16^1) + (10 \times 16^0) = 9786_{10}\end{aligned}$$

donde $E_{16} = 14_{10}$ y $A_{16} = 10_{10}$.

Cuando se utilizan para representar números con múltiples bits, se sabe que una cadena de tres bits binarios puede tomar una de 8 combinaciones diferentes, de modo que cada cadena de tres bits puede ser representada de forma única mediante un dígito octal. De manera similar, una cadena de 4 bits binarios puede representarse por un número hexadecimal.

La utilidad de los sistemas octal y hexadecimal radica en que representan una forma más compacta de los números binarios y es relativamente fácil convertirlos a binario. Para convertir un número de binario a octal, se procede en los siguientes pasos:

1. Comenzando por el extremo derecho de la cadena de bits binarios, y hacia la izquierda.
2. Se añaden ceros a la izquierda hasta hacer el número total de bits un múltiplo de 3.
3. Se separan los bits en grupos de 3.
4. Se reemplaza cada grupo de bits con el correspondiente dígito octal.

El procedimiento para convertir de binario a hexadecimal es similar, excepto que se forman y reemplazan grupos de 4 bits.

binario	decimal	octal	cadena de 3 bits	hexadecimal	cadena de 4 bits
0	0	0	000	0	0000
1	1	1	001	1	0001
10	2	2	010	2	0010
11	3	3	011	3	0011
100	4	4	100	4	0100
101	5	5	101	5	0101
110	6	6	110	6	0110
111	7	7	111	7	0111
1000	8	10	-	8	1000
1001	9	11	-	9	1001
1010	10	12	-	A	1010
1011	11	13	-	B	1011
1100	12	14	-	C	1100
1101	13	15	-	D	1101
1110	14	16	-	E	1110
1111	15	17	-	F	1111

Ejemplo 2.4 Convierta el siguiente número binario en su equivalente octal y hexadecimal.

$$11101110001011_2 = 011101110001011_2 = 011, 101, 110, 001, 011_2 = 35613_8$$

$$11101110001011_2 = 0011101110001011_2 = 0011, 1011, 1000, 1011_2 = 3B8B_{16}$$

2.1.4. Parte fraccional

Todo lo anterior se realiza respecto exclusivamente a números enteros; sin embargo, el tratamiento puede extender para números con parte fraccional. A continuación, se discute cómo diferentes bases consideran la parte fraccional de un número. De nuevo, se inicia con un ejemplo en base decimal, ya que se trata del sistema numérico más familiar. La parte fraccional de un número decimal se indica mediante un punto decimal. Por ejemplo, el número 0.136_{10} tiene:

$$1 \text{ décima} + 3 \text{ centésimas} + 6 \text{ milésimas}$$

Las potencias en base 10 útiles aquí son:

$$\begin{aligned}
 10^{-1} &= 1/10 = 0.1 \\
 10^{-2} &= 1/(10 \times 10) = 0.01 \\
 10^{-3} &= 1/(10 \times 10 \times 10) = 0.001 \\
 10^{-4} &= 1/(10 \times 10 \times 10 \times 10) = 0.0001 \\
 10^{-5} &= 1/(10 \times 10 \times 10 \times 10 \times 10) = 0.00001
 \end{aligned}$$

De este modo, se tiene que:

$$0.136_{10} = (1 \times 10^{-1}) + (3 \times 10^{-2}) + (6 \times 10^{-3})$$

Un ejemplo que considera parte entera y fraccionaria es el siguiente:

$$13.3174_{10} = (1 \times 10^1) + (3 \times 10^0) + (3 \times 10^{-1}) + (1 \times 10^{-2}) + (7 \times 10^{-3}) + (4 \times 10^{-4})$$

Considérese ahora números fraccionales en una base diferente a la base 10. El punto que separa la parte entera de la parte fraccionaria se llama punto decimal para la base 10. En el caso general, se llama punto a la base. En particular, en base 2 se llama punto binario, en base 8 se llama punto octal, y en base 16, punto hexadecimal.

Las fracciones binarias se presentan en un número binario, como por ejemplo 0.101_2 tiene:

$$1 \text{ mitad} + 0 \text{ cuartos} + 1 \text{ octavo}$$

Para los números binarios fraccionales, entonces, se requiere de otro conjunto de potencias de 2:

$$\begin{aligned}
 2^{-1} &= 1/2 = 0.5_{10} \\
 2^{-2} &= 1/(2 \times 2) = 0.25_{10} \\
 2^{-3} &= 1/(2 \times 2 \times 2) = 0.125_{10} \\
 2^{-4} &= 1/(2 \times 2 \times 2 \times 2) = 0.0625_{10} \\
 2^{-5} &= 1/(2 \times 2 \times 2 \times 2 \times 2) = 0.03125_{10}
 \end{aligned}$$

Y por lo tanto, se tiene que:

$$0.101_2 = (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) = 0.625_{10}$$

De forma similar:

$$11.101_2 = (1 \times 10^1) + (1 \times 10^0) + (1 \times 10^{-1}) + (0 \times 10^{-2}) + (1 \times 10^{-3}) = 3.625_{10}$$

Algo similar sucede con el sistema octal, para el cual son útiles las siguientes potencias de 8:

$$\begin{aligned} 8^{-1} &= 1/8 = 0.125_{10} \\ 8^{-2} &= 1/(8 \times 8) = 0.015625_{10} \\ 8^{-3} &= 1/(8 \times 8 \times 8) = 0.001953125_{10} \\ 8^{-4} &= 1/(8 \times 8 \times 8 \times 8) = 0.000244140625_{10} \end{aligned}$$

Así, por ejemplo:

$$0.213_8 = (2 \times 8^{-1}) + (1 \times 8^{-2}) + (3 \times 8^{-3}) = 0.271484735_{10}$$

Finalmente, las potencias de 16 útiles para el cálculo de fracciones hexadecimales son las siguientes:

$$\begin{aligned} 16^{-1} &= 1/16 = 0.0625_{10} \\ 16^{-2} &= 1/(16 \times 16) = 0.00390625_{10} \\ 16^{-3} &= 1/(16 \times 16 \times 16) = 0.000244140625_{10} \\ 16^{-4} &= 1/(16 \times 16 \times 16 \times 16) = 0.00001525878906_{10} \end{aligned}$$

De este modo, se tiene que:

$$1E3.0A1_{16} = 483.0393066406_{10}$$

2.1.5. Cambiando de una base a otra

Tras revisar algunos de los sistemas numéricos más importantes en la computación digital, se analiza ahora algunos de los procedimientos convenientes que pueden utilizarse para cambiar números de una base a otra. Por ejemplo, debe ser posible expresar un número binario en términos de su equivalente decimal o expresar tal número binario en términos de su equivalente octal.

Como se muestra en las igualdades anteriores, es relativamente sencillo convertir de un sistema numérico cualquiera a su equivalente en base 10. Por ahora, se

presentan algunas otras conversiones, comenzando con la conversión de un número decimal a su equivalente binario. Supóngase que se tiene el número decimal 53_{10} , y se desea convertirlo en su equivalente binario. Para esto, se hace necesario obtener cuántos dígitos binarios se encuentran en la primera posición del número entero, es decir, el dígito que se multiplica por 2^0 . Para hacer esto, se divide el número entre 2:

$$\frac{53}{2} = 26 + \frac{1}{2}$$

El residuo de la división (en este caso, 1) representa el dígito binario del número multiplicado por 2^0 . Ahora, tomando el cociente, pero sin considerar el residuo (aquí, el valor 26), se divide de nuevo entre 2. Esto da como resultado el dígito que se multiplica, ahora por la segunda posición a la izquierda o el número multiplicado por 2^1 :

$$\frac{26}{2} = 13 + \frac{0}{2}$$

Ya que no hay residuo, no hay un dígito que multiplique a 2^1 para la representación binaria de 53_{10} . Continuando con la división del cociente, se tiene que:

$$\frac{13}{2} = 6 + \frac{1}{2}$$

El residuo 1 indica que hay un dígito (el propio 1) que multiplica a 2^2 en la expresión binaria de 53_{10} . Al continuar, se hace:

$$\frac{6}{2} = 3 + \frac{0}{2}$$

Por tanto, no hay un dígito binario multiplicando a 2^3 para el equivalente de 53_{10} . Así, se llega a que:

$$\frac{3}{2} = 1 + \frac{1}{2}$$

Que indica que hay un dígito binario multiplicando a 2^4 . Finalmente, el procedimiento termina con la expresión:

$$\frac{1}{2} = 0 + \frac{1}{2}$$

Esto muestra que hay un dígito para 2^5 en la expresión binaria de 53_{10} . Aquí termina el procedimiento, y se obtiene por lo tanto que:

$$53_{10} = 110101_2$$

Si se compara esto con la expansión hecha anteriormente para el número binario 110101_2 , se puede comprobar que es equivalente a 53_{10} . En general, se puede utilizar el procedimiento de la división para obtener el equivalente binario (octal o hexadecimal) de cualquier número decimal. Como ejemplo, exprese 75_{10} en base 2:

$$\begin{array}{rcl} \frac{75}{2} & = & 37 + \frac{1}{2} \\ \frac{37}{2} & = & 18 + \frac{1}{2} \\ \frac{18}{2} & = & 9 + \frac{0}{2} \\ \frac{9}{2} & = & 4 + \frac{1}{2} \\ \frac{4}{2} & = & 2 + \frac{0}{2} \\ \frac{2}{2} & = & 1 + \frac{0}{2} \\ \frac{1}{2} & = & 0 + \frac{1}{2} \end{array}$$

El número binario se obtiene mediante listar todos los residuos en orden reverso, es decir, el primer residuo es el dígito más a la izquierda o bit menos significativo. Así, se tiene que:

$$75_{10} = 1001011_2$$

Hasta aquí, se ha considerado la conversión de un número entero. A continuación, se examina cómo una fracción expresada en base 10 puede convertirse en binario. En este caso, el proceso involucra la multiplicación con la base en lugar de la división entre la base, como en el caso de los números enteros. Por ejemplo, considérese obtener el equivalente binario de 0.125_{10} . Se comienza multiplicando el número por 2:

$$0.125 \times 2 = 0.25$$

La parte entera del resultado es 0, por lo que se sabe que el dígito binario que multiplica a 2^{-1} del equivalente binario es 0. Ahora, se multiplica de nuevo solo la parte fraccional del resultado por 2:

$$0.25 \times 2 = 0.5$$

De nuevo, la parte entera del resultado es 0, por lo que el dígito que multiplica a 2^{-2} también es 0. Multiplicando una vez más la parte fraccional del resultado por 2, se tiene que:

$$0.5 \times 2 = 1.0$$

La parte entera del resultado es ahora 1, de tal modo que el siguiente dígito (que multiplica a 2^{-3}) es 1. Además, la parte fraccional del resultado es 0, por lo que el procedimiento puede terminarse aquí. Así se tiene que:

$$0.125_{10} = 0.001_2$$

Considérese otro ejemplo: exprese 0.257_{10} en binario. Recuerdese que sólo la parte fraccional se multiplica cada vez por 2:

$$\begin{array}{rcl} 0.257 \times 2 & = & 0.514 \\ 0.514 \times 2 & = & 1.028 \\ 0.028 \times 2 & = & 0.056 \\ 0.056 \times 2 & = & 0.112 \\ 0.112 \times 2 & = & 0.224 \\ 0.224 \times 2 & = & 0.448 \\ 0.448 \times 2 & = & 0.896 \\ 0.896 \times 2 & = & 1.792 \\ 0.792 \times 2 & = & 1.584 \\ & & \vdots \end{array}$$

Por lo tanto:

$$0.257_{10} = 0.010000011..._2$$

Nótese que este procedimiento se puede continuar, repitiéndose indefinidamente; esto significa que no hay una representación binaria exacta para este número. Esto es, hay un número infinito de términos a la derecha del punto binario.

La conversión de algunas fracciones decimales a su equivalente binario puede resultar en algunas inexactitudes, ya que ni las personas ni las computadoras pueden trabajar con un número infinito de términos. Tal inexactitud se conoce como **error de redondeo** (*round-off error*). El uso de suficientes términos puede hacer que la inexactitud sea despreciable.

Ahora bien, cuando se convierte un número que consta tanto de parte entera como fraccional, la conversión de cada parte se realiza por separado. Por ejemplo, para expresar 53.125_{10} en binario, se convierten 53_{10} y 0.125_{10} como se ha mostrado anteriormente para obtener que:

$$53.125_{10} = 110101.001_2$$

Si se desea convertir un número decimal a su equivalente octal o hexadecimal, se utiliza el mismo procedimiento, excepto que en el caso octal la parte entera se divide entre 8 y la parte fraccionaria se multiplica por 8. Obviamente, en el caso hexadecimal, se utiliza 16. Por ejemplo, a continuación se convierte 31_{10} a base octal:

$$\begin{aligned}\frac{31}{8} &= 3 + \frac{7}{8} \\ \frac{3}{8} &= 0 + \frac{3}{8}\end{aligned}$$

Por lo tanto:

$$31_{10} = 37_8$$

En seguida, se convierte 0.125_{10} a su equivalente octal:

$$0.125 \times 8 = 1.000$$

Y por tanto:

$$0.125_{10} = 0.1_8$$

Algo similar se aplica a la conversión a base hexadecimal. Por ejemplo, 31_{10} se convierte a hexadecimal de la siguiente manera:

$$\begin{aligned}\frac{31}{16} &= 1 + \frac{15}{16} \\ \frac{1}{16} &= 0 + \frac{1}{16}\end{aligned}$$

Sin embargo, el número 15 se representa en hexadecimal mediante el símbolo F , por lo que:

$$31_{10} = 1F_{16}$$

Finalmente, se muestra cómo convertir 0.125_{10} a hexadecimal:

$$0.125 \times 16 = 2.000$$

Como no hay parte fraccional remanente, se tiene que:

$$0.125_{10} = 0.2_{16}$$

Finalmente, se considera la conversión entre bases binaria, octal y hexadecimal. Tal conversión es relativamente simple, y puede realizarse por inspección. Supóngase que se desea convertir el número octal 2637_8 a binario. Simplemente, la conversión se realiza mediante escribir cada dígito del número octal como un número binario de tres dígitos. Cuando estos dígitos se escriben en el orden en que aparecen en el número octal, se obtiene el equivalente binario deseado. En el ejemplo actual:

$$2_8 = 010_2$$

$$6_8 = 110_2$$

$$3_8 = 011_2$$

$$7_8 = 111_2$$

Entonces:

$$2637_8 = 10110011111_2$$

Este simple procedimiento también es válido para números con parte fraccional, siempre considerando dónde se encuentra el punto. Por ejemplo:

$$2637.126_8 = 10110011111.00101011_2$$

El procedimiento, por supuesto, puede usarse en forma inversa para convertir de binario a octal. Por ejemplo, supóngase que se tiene 1011_2 y se desea convertir a su equivalente octal. Primero, se añaden ceros a la izquierda del número binario a fin de que el número de dígitos en él sea un múltiplo de tres. En el ejemplo presente, esto hace que se tenga el número binario como 001011_2 . En seguida,

se divide el número binario en grupos de tres dígitos. Finalmente, se escribe el equivalente octal de cada uno de los grupos de tres dígitos. Así, se tiene que:

$$001011_2 = 13_8$$

El mismo método se utiliza para cualquier número binario con parte fraccional, siempre tomando en cuenta la posición del punto. Pero ahora se añaden ceros tanto a la izquierda como a la derecha del número, de tal forma que el número de dígitos hacia la izquierda y derecha del punto binario sea un múltiplo de tres. Nótese que añadir ceros a la izquierda y derecha del número binario original no cambia su valor numérico. Por ejemplo, conviértase 1011.10111_2 a base octal. Añadiendo ceros, se tiene:

$$001011.101110_2 = 13.56_8$$

La conversión de hexadecimal a binario y de binario a hexadecimal son muy similares a las conversiones entre binario y octal, con la excepción de que ahora los grupos de dígitos binarios son de cuatro. Por ejemplo:

$$1EA.26B_{16} = 0001\ 1110\ 1010.0010\ 0110\ 1011_2$$

Nótese que las conversiones entre binario, octal y hexadecimal no están sujetas al error de redondeo.

En general, las computadoras trabajan sólo con números binarios; sin embargo, resulta más fácil para las personas utilizar números octales o hexadecimales, ya que tienen un número menor de dígitos y la conversión de binario a octal o hexadecimal (y viceversa) es fácil de realizar utilizando un simple programa de computadora. Por otro lado, algunas computadoras se han construido de tal modo que pueden aceptar en su entrada y producir en su salida números octales y hexadecimales; sin embargo, todo procesamiento de datos se realiza en binario. Además, en la mayoría de las aplicaciones, la programación y la entrada de datos se hace mediante números decimales. Los números octales y hexadecimales se usan cuando se requiere estudiar los números binarios dentro de la computadora. Esto es muy útil si se desea construir una computadora, y se desea probar que funciona apropiadamente. Finalmente, los números binarios y sus equivalentes octales y hexadecimales se usan también en algunos tipos de programas que se discuten más adelante.

2.1.6. ¿Porqué utilizar binarios en sistemas de cómputo?

Los circuitos de dos estados o digitales (también llamados binarios) son resistentes al ruido, fáciles de diseñar, simples de comprender y extremadamente confiables. Los datos o información puede ser fácilmente manipulable mediante utilizar circuitos electrónicos muy sencillos conocidos como compuertas, como se muestra más adelante.

2.1.7. Algo de aritmética binaria elemental

En esta sección se discuten algunas ideas sobre la adición utilizando números binarios. Posteriormente, se discute la aritmética binaria en mayor detalle, cuando se describa cómo la aritmética se realiza por una computadora. Se discute, además, algunas restricciones resultantes cuando la aritmética se realiza por computadoras.

Ya que la base decimal es más familiar, se inicia la discusión sobre aritmética binaria elemental con un ejercicio de adición. Considérese el siguiente ejemplo:

$$\begin{array}{r} 1\ 2\ 4\ 3\ 6 \\ + 1\ 3\ 2\ 5\ 3 \\ \hline 2\ 5\ 6\ 8\ 9 \end{array}$$

Como se sabe, cada columna (unidades, decenas, centenas, unidades de millar, etc.) se suma para obtener el resultado deseado. En este ejemplo, la suma de cada columna no excede o es igual a la base. Así, no se tiene la necesidad de llevar un acarreo de una columna a la siguiente (consideradas de derecha a izquierda). Obsérvese lo que ocurre cuando existe un acarreo (es decir, el resultado excede o es igual a la base):

$$\begin{array}{r} 1\ 5\ 7\ 8 \\ + 2\ 6\ 9\ 4 \\ \hline 4\ 2\ 7\ 2 \end{array}$$

El resultado de la primer columna (de unidades) es 12, lo cual excede la base 10 en este caso; se escribe el 2, y los restantes 10 se acarrean a la siguiente columna (de decenas) mediante sumarle un 1. Tal procedimiento se repite para todas las columnas de los sumandos. Por ejemplo, es necesario añadir un 1 a la tercera columna (de centenas) para continuar.

Exactamente la misma idea básica se aplica para sumar cualquier par de números, independientemente de su base. Por ejemplo, considérese la siguiente adición octal:

$$\begin{array}{r}
 1\ 2\ 5\ 3\ 8 \\
 + 4\ 3\ 2\ 1\ 8 \\
 \hline
 5\ 5\ 7\ 4\ 8
 \end{array}$$

Es posible comprobar que la adición es correcta mediante convertir los números octales a decimales.

Ahora bien, considérese una adición octal con acarreo:

$$\begin{array}{r}
 1\ 4\ 7\ 6\ 8 \\
 + 1\ 6\ 3\ 4\ 8 \\
 \hline
 3\ 3\ 3\ 2\ 8
 \end{array}$$

Puede verificarse que $6_8 + 4_8 = 12_8 = 10_{10}$. Por tanto, se escribe el 2 en la columna de unidades y un 1 se acarrea a la siguiente columna (de 8^1). Ahora, se suman $1_8 + 7_8 + 3_8 = 13_8 = 11_{10}$. Se escribe el 3 en la segunda columna y se acarrea otro 1 a la tercera columna (de 8^2). Este procedimiento parece más complicado que la adición en base 10, pero esto se debe a la poca familiaridad con los números octales, y además, que se requiere conocer las tablas de la adición en base 8.

La adición binaria utiliza el mismo procedimiento básico. Un ejemplo que no requiere acarreo se muestra a continuación:

$$\begin{array}{r}
 1\ 0\ 1\ 1\ 0\ 2 \\
 + 0\ 1\ 0\ 0\ 1\ 2 \\
 \hline
 1\ 1\ 1\ 1\ 1\ 2
 \end{array}$$

Otro ejemplo que lleva a cabo acarreo es el siguiente:

$$\begin{array}{r}
 1\ 0\ 1\ 1\ 1\ 2 \\
 + 0\ 0\ 0\ 1\ 1\ 2 \\
 \hline
 1\ 1\ 0\ 1\ 0\ 2
 \end{array}$$

Considérese la primera columna. Se tiene que sumar $1_2 + 1_2 = 10_2$. El 0 se coloca en la primera columna del resultado y se acarrea el 1 a la segunda columna (de 2^1). En la segunda columna se suma el acarreo, teniendo $1_2 + 1_2 + 1_2 = 11_2$. El 1 se escribe como resultado en la segunda columna y se acarrea otro 1 a la tercera columna (de 2^2). Como puede verse, el procedimiento se repite por tantas columnas como las haya entre los dos números binarios. Nótese que las reglas de la adición binaria son exactamente las mismas que las reglas para la adición decimal.

En el siguiente ejemplo, se suman dos números binarios con parte fraccional, que igualmente siguen las reglas básicas de la adición. Nótese la alineación de los dos números respecto al punto binario:

$$\begin{array}{r} 101101.101_2 \\ + 000110.001_2 \\ \hline 110011.110_2 \end{array}$$

Un problema que no sucede cuando realizamos adiciones manualmente, pero que puede ocurrir cuando se utiliza una computadora, se ilustra en el siguiente ejemplo:

$$\begin{array}{r} 10110111_2 \\ + 10111001_2 \\ \hline 101110000_2 \end{array}$$

En este último ejemplo, los dos números que se suman tienen 8 dígitos cada uno. Un dígito binario recibe el nombre de bit. Así, se dice que se han sumado dos números de 8 bits. Ahora bien, debido al acarreo de la columna de la extrema izquierda, el resultado de la suma tiene 9 bits. Esto no representa ningún problema cuando las personas suman números; sin embargo, en las computadoras, los números deben almacenarse en dispositivos físicos llamados registros. Un registro puede almacenar hasta un cierto número de bits. Si el número binario que debe almacenarse tiene más bits que los que pueden almacenarse en el registro, el exceso de bits se pierde. Notoriamente, esto puede provocar un error muy substancial. Por ello, es importante analizar a detalle cómo tal error sucede.

Un registro que puede almacenar un número binario de 8 bits puede representarse gráficamente como se muestra en la Figura 2.1 (más adelante se discuten circuitos reales para la implementación de registros).

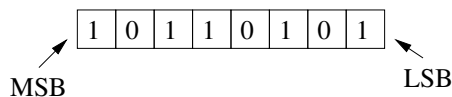


Figura 2.1: Una representación gráfica de un registro de 8 bits almacenando el número 10110101_2 .

El bit a la extrema derecha se conoce con el nombre de bit menos significativo (*Least Significant Bit* o LSB) ya que tiene el valor numérico más pequeño de

todos los dígitos que componen al número binario. En forma similar, el bit más a la extrema izquierda se le conoce como bit más significativo (*Most Significant Bit* o MSB), debido a que tiene el mayor valor numérico dentro de número binario. Por el momento, supóngase que se trabaja con valores enteros (posteriormente se aplican las mismas ideas a números con parte fraccional).

Ahora bien, si por una operación aritmética como la adición, el resultado que se obtiene tiene más bits que los que pueden almacenarse en el registro, entonces (a) los bits que se retienen son los menos significativos; y (b) los bits más significativos se pierden. Por lo tanto, en el último ejemplo de adición, si el resultado se coloca en un registro de 8 bits, la respuesta sería:

$$01110000_2 = 112_{10}$$

en lugar de la respuesta correcta, que sería:

$$101110000_2 = 368_{10}$$

Como puede observarse, esto representa un error substancial e importante, conocido con el nombre de sobrecapacidad u *overflow*. Los constructores de computadoras, así como los usuarios de las mismas, deben entender claramente el almacenamiento de números binarios en registros para evitar este tipo de errores. En las secciones 2.3 y 2.7 se discuten la teoría y aritmética del almacenamiento de números binarios en mayor detalle, así como las formas en que el *overflow* puede, en ocasiones, utilizarse como ayuda en el cómputo.

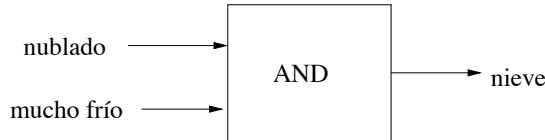
2.2. Álgebra booleana y compuertas lógicas

Esta sección presenta las compuertas (los elementos o circuitos lógicos más primitivos utilizados en computadoras modernas) como representaciones o implementaciones en hardware de los operadores booleanos AND, OR y NOT.

2.2.1. Álgebra booleana

En lógica proposicional, las proposiciones pueden ser verdaderas (*TRUE*) o falsas (*FALSE*), y se presentan como funciones de otras proposiciones que se conectan entre sí mediante tres conectivos lógicos básicos: AND, OR y NOT. Considérese el siguiente ejemplo:

Ejemplo 2.5 El enunciado: “SI está nublado Y hace mucho frío ENTONCES nevará”, se compone de las proposiciones de entrada “está nublado” y “hace mucho frío”, y tiene como salida la proposición “nevará”. El significado del conector Y (conocido como AND) es que la proposición de salida es verdadera (*TRUE*) si y sólo si ambas proposiciones de entrada son verdaderas (*TRUE*).



Como sólo hay dos valores posibles para cada proposición de entrada, se puede obtener un valor de verdad de la proposición de salida para todas las posibles combinaciones de las entradas, como se describe en la siguiente tabla de verdad.

entrada 1: “está nublado”	entrada 2: “mucho frío”	salida: “nevará”
<i>FALSE</i>	<i>FALSE</i>	<i>FALSE</i>
<i>FALSE</i>	<i>TRUE</i>	<i>FALSE</i>
<i>TRUE</i>	<i>FALSE</i>	<i>FALSE</i>
<i>TRUE</i>	<i>TRUE</i>	<i>TRUE</i>

Las proposiciones lógicas pueden hacerse tan complejas como se requiera mediante utilizar los conectivos AND, OR y NOT. El álgebra booleana (desarrollada por George Boole durante el siglo XIX) simplifica el manejo de conectivos binarios, utilizando una notación algebraica ordinaria y los valores de 1 para *TRUE* y 0 para *FALSE*.

Comúnmente, la AND se representa como **producto lógico**, ya sea como dos variables unidas (por ejemplo, AB), o por los símbolos \cdot (por ejemplo, $A \cdot B$) o $*$ (por ejemplo, $A * B$); la OR se representa como **adición lógica** mediante el símbolo $+$ (por ejemplo, $A + B$); y la NOT se representa como una barra sobre la variable (por ejemplo \bar{A}) o simplemente mediante una comilla (por ejemplo, A').

Las tablas de verdad de las tres operaciones básicas binarias se presentan a continuación, donde A y B son entradas, y Y es la salida.

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

A	A'
0	1
1	0

Para n entradas, el número total de combinaciones es 2^n .

Una función booleana puede transformarse de una expresión algebraica a un diagrama lógico compuesto por compuertas AND, OR y NOT. Mediante implementar tal diagrama lógico en hardware, se obtiene un circuito digital.

2.2.2. Lógica digital

En una computadora digital, todos los valores se representan por ceros (0's) o unos (1's). Se puede decir que un valor, en cualquier momento, es una señal (una variable física cuya magnitud cambia en el tiempo), y que tal señal es digital cuando sólo puede tener los valores de 0 o 1. A tales valores se les conoce con el nombre de valores lógicos. Tal nombre proviene de una rama de las matemáticas conocida como lógica matemática. En tal rama, el objetivo es estudiar aquellas situaciones o hechos que son falsos o verdaderos. Respecto a las computadoras digitales, éstas se construyen para funcionar con 0's y 1's, es decir, son una implementación física de un sistema matemático de dos valores, lo que hace que la lógica matemática tenga una gran aplicación e importancia en el diseño de computadoras; sin embargo, para ello no es necesario considerar todos los detalles de la lógica matemática. Basta considerar algunas ideas simples que ayudan a entender de forma fácil la circuitería de una computadora digital.

Inicialmente, se requiere definir qué es una variable binaria. Esto no es otra cosa que una variable que puede tener un valor 0 o 1, es decir, en un momento dado, tal variable puede contener como valor un 0 o un 1. Las variables binarias se utilizan para representar los valores de las señales digitales dentro de una computadora. Nótese la razón por lo que esto es conveniente. En la Figura 2.2

se muestra un simple interruptor. En una computadora, tal interruptor se construye utilizando transistores u otros dispositivos semiconductores; sin embargo, para los propósitos actuales y por simplicidad, se muestra aquí como un simple interruptor. Nótese que en la figura, la letra A representa una variable binaria: cuando $A = 0$, el interruptor se encuentra abierto, y cuando $A = 1$, el interruptor se cierra. El símbolo a la izquierda del diagrama es una batería con voltaje V , y el símbolo a la derecha del diagrama es una resistencia eléctrica R . Es conveniente pensar en ella como un pequeño foco.

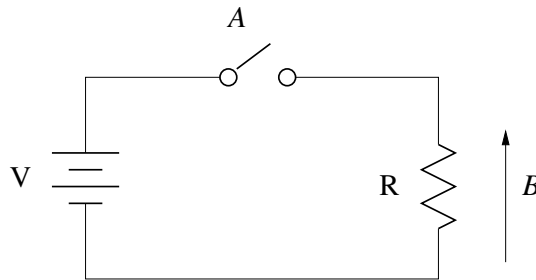


Figura 2.2: Un circuito lógico simple.

Comúnmente, al circuito de la Figura 2.2 se le conoce como circuito lógico, donde B representa la caída de voltaje en la resistencia. Cuando el interruptor está abierto, $B = 0$. Cuando el interruptor se cierra, el voltaje de la batería se presenta en la resistencia (el foco se enciende). Se considera que esto corresponde a $B = 1$. Por lo tanto, cuando $A = 1$, entonces $B = 1$; y cuando $A = 0$, entonces $B = 0$. Se puede escribir esto mediante la ecuación:

$$A = B$$

Otra forma de describir este comportamiento del circuito lógico es mediante una tabla de verdad. En tal tabla, se listan los valores que va tomando la salida B para todos los posible valores de entrada de A . Para el circuito lógico de la Figura 2.2 se tiene que:

A	B
0	0
1	1

B se conoce como variable dependiente, ya que su valor depende del valor de A . Complementariamente, A se conoce como variable independiente, pues su valor no depende de ninguna otra variable.

Considérese ahora el circuito de la Figura 2.3, que es un poco más complicado que el anterior. Recuérdese que los interruptores están cerrados cuando su valor es 1, y se encuentran abiertos cuando su valor es 0.

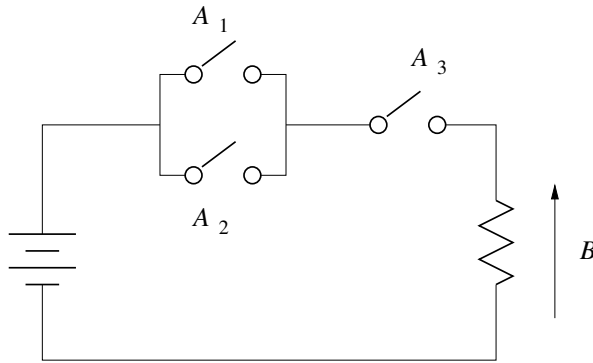


Figura 2.3: Otro circuito lógico.

Para este circuito, B toma el valor de 1 si A_1 y A_3 son ambos 1, o si A_2 y A_3 son ambos 1, o si A_1 , A_2 y A_3 son todos 1. La tabla de verdad es ahora una forma conveniente de mostrar esta información. La columna de salida B muestra los valores que tal variable puede presentar para todas las combinaciones posibles de las variables independientes A_1 , A_2 y A_3 .

A_1	A_2	A_3	B
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Para finalizar, es importante mencionar que en esta sección se consideran tan solo algunas formas de representar circuitos lógicos. En realidad, los circuitos que se han discutido se conocen como circuitos combinacionales o lógica combinacional. En la siguiente sección se continúa la discusión de algunos circuitos combinacionales que forman los bloques básicos de construcción de computadoras.

2.2.3. Compuertas lógicas

Los circuitos digitales son componentes de *hardware* que manipulan información binaria. Los circuitos se implementan utilizando transistores dentro de circuitos integrados. Cada circuito básico de transistores se conoce como **compuerta lógica**.

Esta sección presenta las compuertas, pero no toma en cuenta los elementos electrónicos internos de las compuertas individuales, sino solo sus propiedades externas y lógicas. Las compuertas lógicas operan en una o más señales binarias de entrada para producir una señal binaria de salida. La funcionalidad de una compuerta lógica, y en general de los circuitos lógicos, se describe utilizando tablas de verdad.

Hay siete compuertas lógicas simples, que combinadas pueden implementar cualquier bloque funcional dentro de un sistema de cómputo. Cada tipo de compuerta tiene un nombre, un símbolo gráfico, una función lógica booleana y una tabla de verdad.

Los circuitos lógicos digitales se encuentran disponibles en forma de circuitos integrados (o *chips*), de tal modo que pueden utilizarse para construir cualquier tipo de circuitos lógicos. En las siguientes secciones se muestra cómo ciertos circuitos básicos, genéricamente llamados compuertas lógicas (*logic gates*) o simplemente compuertas (*gates*), se usan para construir en la práctica varios dispositivos lógicos. Tales compuertas realizan algún tipo de operación lógica. Por tanto, se presenta la operación y el circuito que la representa y realiza.

La compuerta AND

Una de las más comunes e importantes operaciones lógicas se ilustra en la Figura 2.4, la cual muestra dos interruptores conectados mediante un circuito en serie.

Nótese que $B = 1$ sólo en el caso en que tanto el interruptor A_1 como el interruptor A_2 tengan ambos un valor de 1. Es por esto que a este circuito se le conoce con el nombre de circuito AND.

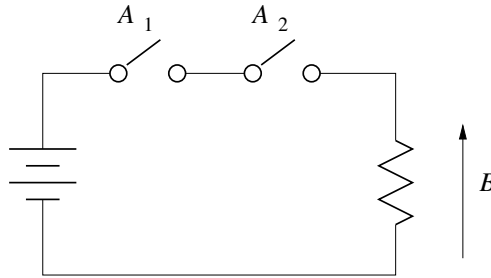


Figura 2.4: Un circuito AND con interruptores.

La tabla de verdad de la operación lógica AND se muestra como sigue:

A_1	A_2	B
0	0	0
0	1	0
1	0	0
1	1	1

Con la finalidad de obtener una representación más breve, se utilizan algunos símbolos especiales entre las variables para indicar operaciones lógicas. En el caso de la operación AND, comúnmente se utiliza colocar las variables juntas. De tal modo, la operación de la Figura 2.4 puede representarse mediante la siguiente ecuación:

$$B = A_1 A_2$$

Actualmente, existen algunas compuertas semiconductoras que han sido construidas para realizar la operación lógica AND. Más aun, en lugar de utilizar interruptores para representar compuertas, se utilizan algunos símbolos o diagramas lógicos, que son más pequeños y fáciles de dibujar. El símbolo para la compuerta AND se muestra en la Figura 2.5.

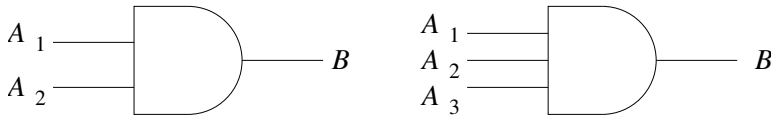


Figura 2.5: Símbolos para la compuerta AND de dos y tres entradas.

Nótese que inicialmente se muestra una compuerta AND con dos entradas. En realidad, las compuertas AND pueden construirse con muchas más entradas (o compuertas AND pueden conectarse entre sí de modo que se obtenga un circuito lógico cuya salida sea equivalente a una compuerta AND de varias entradas). La Figura 2.5 muestra también una compuerta AND de tres entradas. La variable de salida B tiene como valor 1 sólo si las variables de entrada A_1 , A_2 y A_3 tienen todas valor de 1.

La Figura 2.6 resume las principales características de la compuerta AND.

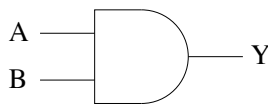
símbolo	función lógica	tabla de verdad															
	$Y = A B$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>Y</th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1
A	B	Y															
0	0	0															
0	1	0															
1	0	0															
1	1	1															

Figura 2.6: AND.

La compuerta OR

La operación OR, que consiste en dos interruptores conectados en paralelo, se muestra en la Figura 2.7. Aquí, B tiene valor de 1 si A_1 es 1, o si A_2 es 1, o si ambos, A_1 , A_2 , son 1.

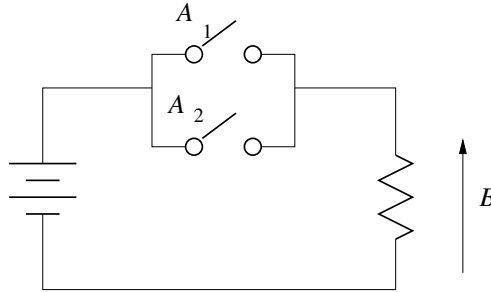


Figura 2.7: Un circuito de interruptores para la operación lógica OR.

La tabla de verdad de la operación lógica OR es como sigue:

A_1	A_2	B
0	0	0
0	1	1
1	0	1
1	1	1

El símbolo usado para designar la operación lógica OR en forma de ecuación es $+$. Por tanto, para el circuito de la Figura 2.7, se tiene que:

$$B = A_1 + A_2$$

Nótese que cuando las ecuaciones lógicas se escriben, los símbolos que utilizan son los mismos que se usan en aritmética ordinaria; sin embargo su significado no es el mismo.

La Figura 2.8 muestra los símbolos para una compuerta OR de dos y cuatro entradas.

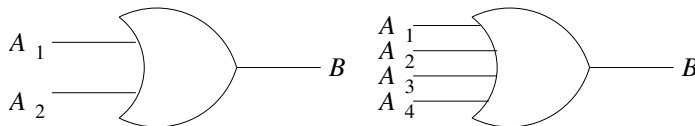


Figura 2.8: Símbolos para la compuerta OR.

Para la compuerta OR de cuatro entradas, se tiene la siguiente ecuación:

$$B = A_1 + A_2 + A_3 + A_4$$

En este caso, B tiene el valor de 1 si A_1 o A_2 o A_3 o A_4 , en cualquier combinación, es 1.

Un ejemplo de circuitos construidos con compuertas se muestra en la Figura 2.9.

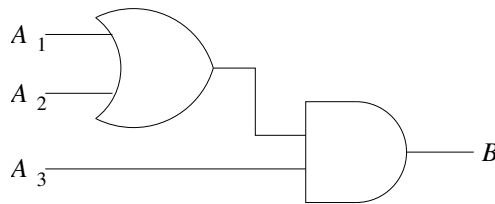


Figura 2.9: Un circuito con compuertas para el circuito de la Figura 2.3.

Nótese que la variable de salida B tiene valor 1 si A_1 o A_2 tienen valor 1, y además si A_3 tiene valor 1. De hecho, el circuito de la Figura 2.9 representa mediante un circuito con compuertas la función del circuito con interruptores de la Figura 2.3.

La ecuación lógica que representa a ambos circuitos es:

$$B = (A_1 + A_2)A_3$$

El paréntesis en esta expresión se interpreta de la siguiente forma: todos los términos o variables dentro del paréntesis se tratan como una sola variable en relación con los términos o variables fuera del paréntesis. Por tanto, B tiene valor 1 si A_1 o A_2 da como resultado 1, y este resultado y A_3 da como resultado 1.

La Figura 2.10 resume las principales características de la compuerta OR.

La compuerta NOT

Considérese ahora la operación lógica llamada negación o complemento. Si la variable A tiene como valor 1, su negación es 0; si A tiene valor 0, su negación es 1. Esto significa que cuanto se obtiene la negación o complemento de una variable,

ésta cambia su valor de 0 a 1 o viceversa. El símbolo que se utiliza para la negación es una comilla ('). Por lo tanto, si B es la negación de A , se escribe la ecuación:

$$B = A'$$

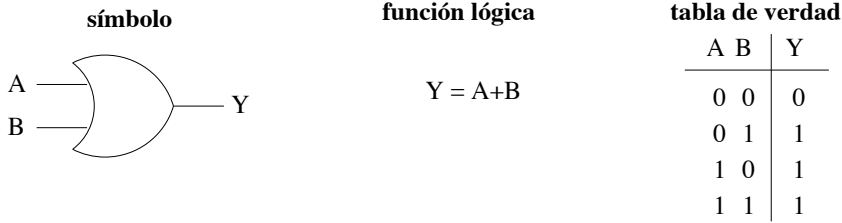


Figura 2.10: OR.

La compuerta que realiza la negación de una variable se conoce como compuerta NOT. Su diagrama lógico se presenta en la Figura 2.11.

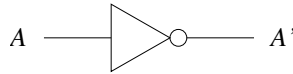


Figura 2.11: El diagrama lógico de la compuerta NOT.

Nótese que las compuertas NOT tienen tan solo una entrada. El pequeño círculo en el diagrama de la compuerta NOT se utiliza frecuentemente para representar la operación de negación. Por ejemplo, para la Figura 2.12, se tiene la ecuación:

$$B = A_1 A_2'$$

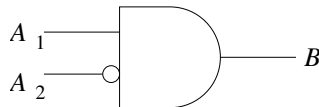


Figura 2.12: Un circuito lógico que realiza la operación $B = A_1 A_2'$.

Efectivamente, el pequeño círculo representa una compuerta NOT que toma como entrada la variable A_2 ; sin embargo, el símbolo del pequeño círculo nunca se utiliza solo, sino en conjunto con otras compuertas.

La Figura 2.13 resume las principales características de la compuerta AND.

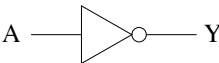
símbolo	función lógica	tabla de verdad						
	$Y = A'$	<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	Y	0	1	1	0
A	Y							
0	1							
1	0							

Figura 2.13: NOT (inversor).

La compuerta NOR

La operación lógica NOR consiste en realizar una operación OR y negar el resultado. Se define mediante la ecuación:

$$B = (A_1 + A_2)'$$

La tabla de verdad para una operación NOR es:

A_1	A_2	B
0	0	1
0	1	0
1	0	0
1	1	0

Nótese que para esta operación, la variable de salida B es tan solo la negación de la operación OR de las variables de entrada.

El diagrama lógico de la compuerta NOR se muestra en la Figura 2.14.

También se muestra en la Figura 2.14 una compuerta NOR de tres entradas, cuya ecuación lógica es:

$$B = (A_1 + A_2 + A_3)'$$

Para la compuerta NOR, la variable de salida B tiene un valor de 0 cuando A_1 , A_2 , A_3 , o cualquiera de sus combinaciones entre sí tiene valor 1. De hecho, las

compuertas NOR, así como las compuertas NAND que se discuten a continuación, puede construirse de una manera muy sencilla como semiconductores en forma de circuitos integrados, siendo su uso en general y en la práctica muy diseminado. Además, toda función lógica puede construirse a partir de compuertas NOR y NAND, como se discute más adelante.

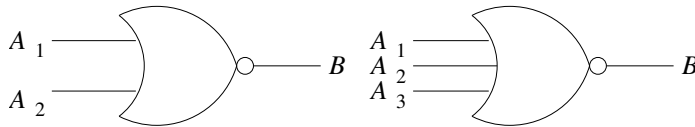


Figura 2.14: El diagrama lógico de la compuerta NOR.

La Figura 2.15 resume las principales características de la compuerta NOR.

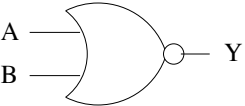
símbolo	función lógica	tabla de verdad															
	$Y = (A+B)'$	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>Y</th></tr> </thead> <tbody> <tr> <td>0</td><td>0</td><td>1</td></tr> <tr> <td>0</td><td>1</td><td>0</td></tr> <tr> <td>1</td><td>0</td><td>0</td></tr> <tr> <td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0
A	B	Y															
0	0	1															
0	1	0															
1	0	0															
1	1	0															

Figura 2.15: NOR (NOT OR).

La compuerta NAND

La operación lógica NAND consiste en realizar una operación AND y negar el resultado. Se define mediante la ecuación siguiente:

$$B = (A_1 A_2)'$$

La tabla de verdad de la operación NAND es como sigue:

A_1	A_2	B
0	0	1
0	1	1
1	0	1
1	1	0

Nótese que el valor de la variable B es tan solo la negación de la operación AND entre las variables de entrada A_1 y A_2 . El diagrama lógico para una compuerta NAND se muestra en la Figura 2.16

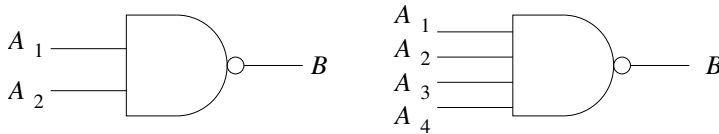


Figura 2.16: El diagrama lógico de la compuerta NAND.

Como se muestra en la Figura 2.16 y es el caso de la mayoría de las compuertas, la compuerta NAND puede contar con más de dos entradas. La compuerta NAND de cuatro entradas que se muestra tiene la ecuación lógica:

$$B = (A_1 A_2 A_3 A_4)'$$

Nótese que B tiene valor 0 sólo si A_1 , A_2 , A_3 y A_4 tienen todas valor 1.

La Figura 2.17 resume las principales características de la compuerta NAND.

símbolo	función lógica	tabla de verdad															
	$Y = (A B)'$	<table> <tr> <th>A</th><th>B</th><th>Y</th></tr> <tr> <td>0</td><td>0</td><td>1</td></tr> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>0</td></tr> </table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0
A	B	Y															
0	0	1															
0	1	1															
1	0	1															
1	1	0															

Figura 2.17: NAND (NOT AND).

La compuerta XOR

La operación lógica OR exclusiva, que se conoce con el nombre de XOR, es similar a la operación OR excepto que, para el caso de dos entradas, la salida B tiene valor 0 cuando ambas variables de entrada A_1 y A_2 tienen valor 1. El símbolo dado a la operación XOR es \oplus . Por lo tanto, la ecuación para la operación XOR se escribe como:

$$B = A_1 \oplus A_2$$

La tabla de verdad de la operación XOR es como sigue:

A_1	A_2	B
0	0	0
0	1	1
1	0	1
1	1	0

El diagrama lógico de la compuerta XOR se muestra en la Figura 2.18.

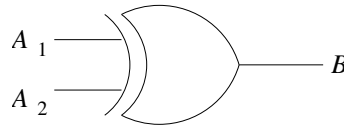


Figura 2.18: El diagrama lógico de la compuerta XOR.

Cuando una compuerta XOR tiene más de dos entradas, el resultado a la salida es algo más complicado. Supóngase que se tiene la siguiente ecuación:

$$B = A_1 \oplus A_2 \oplus A_3$$

Es posible reescribirla de la siguiente manera:

$$B = (A_1 \oplus A_2) \oplus A_3$$

De este modo, es posible obtener la operación XOR entre A_1 y A_2 , a su resultado, hacer de nuevo una operación XOR con A_3 . De hecho, la agrupación de las dos primeras variables puede considerar cualquier combinación de entre dos variables de las tres existentes. El resultado de la operación XOR con tres entradas se muestra en la siguiente tabla de verdad:

A_1	A_2	A_3	$A_1 \oplus A_2$	B
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

Nótese que si $A_1 = 1$, $A_2 = 1$, y $A_3 = 1$, entonces $A_1 \oplus A_2 \oplus A_3 = 1$, y no 0, como podría haberse esperado.

Las Figuras 2.19 y 2.20 resumen las principales características de la compuerta XOR y su complemento, la compuerta XNOR.

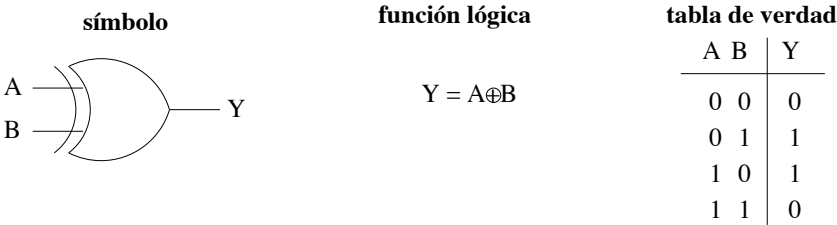


Figura 2.19: Exclusive OR (XOR o diferencia).

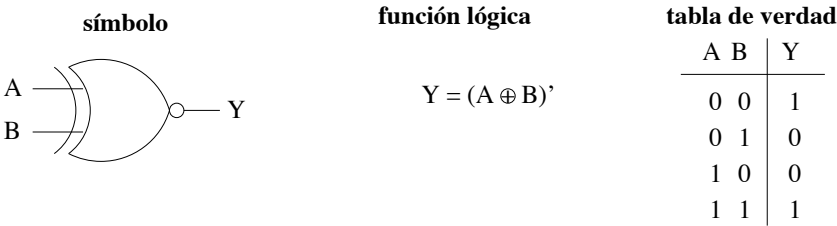


Figura 2.20: Exclusive NOR (XNOR o igualdad).

Hasta ahora, se han mostrado una serie de compuertas que operan únicamente entre variables de entrada y salida. En realidad, las compuertas de los circuitos digitales integrados tienen otras variables, que físicamente son terminales o conexiones. De tal modo, las señales lógicas consisten de voltajes que se aplican (en el caso de las entradas) o se miden (en el caso de las salidas) entre pares de terminales. Una de tales terminales es llamada en circuitos eléctricos la tierra (*ground*, o GND), la cual es común para todas las entradas y salidas. Obsérvese que en los diagramas lógicos no se muestra tal conexión, ya que tales diagramas se presentan de una manera únicamente convencional.

Sin embargo, a fin de que las compuertas (o cualquier circuito digital integrado) funcione apropiadamente, debe conectarse a un voltaje directo, como el que proporciona una batería o una fuente de voltaje. Las conexiones a la fuente de voltaje también se requieren como parte del circuito integrado, pero tampoco se muestran en los diagramas lógicos. Nótese que el voltaje directo que debe aplicarse debe tener una magnitud determinada (para la mayoría de los circuitos digitales integrados, tal valor es de 5 V). Si no es así, el circuito podría no funcionar apropiadamente si el voltaje de alimentación es muy pequeño, o podría quemarse si es muy grande.

2.2.4. Interconexión de compuertas para obtener otras compuertas

Todas las compuertas que se han mencionado hasta ahora son utilizadas cuando se diseñan e implementan circuitos lógicos para una computadora digital; sin embargo, comúnmente en la práctica no se requiere que todos los tipos de compuertas sean físicamente construidos, ya que la operación de algunas compuertas puede obtenerse a partir de interconectar otras compuertas más pequeñas y sencillas de implementar. Por ejemplo, considérese las interconexiones que se muestran en la Figura 2.21.

Nótese que las operaciones NOR y NAND son tan solo negaciones de las operaciones OR y AND, respectivamente. Por tanto, tales operaciones lógicas se obtienen mediante aplicar una compuerta NOT a la salida de las compuertas OR o AND, según sea el caso.

Sin embargo, la implementación de la compuerta XOR es algo más complicada. De hecho, es necesario hacer un cierto análisis para demostrar que la interconexión

$(A_1 + A_2)(A_1A_2)'$ es equivalente a la operación XOR. La forma más sencilla es escribir la tabla de verdad para esta interconexión:

A_1	A_2	$A_1 + A_2$	A_1A_2	$(A_1A_2)'$	$(A_1 + A_2)(A_1A_2)'$	$A_1 \oplus A_2$
0	0	0	0	1	0	0
0	1	1	0	1	1	1
1	0	1	0	1	1	1
1	1	1	1	0	0	0

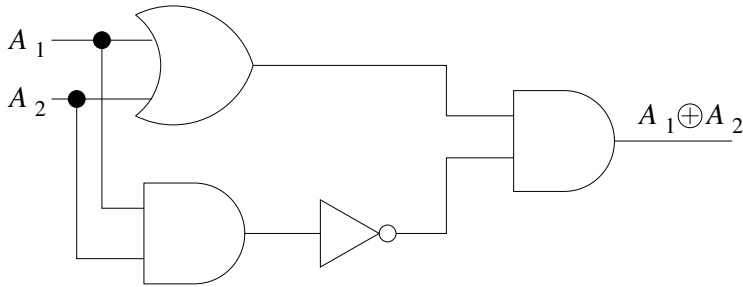
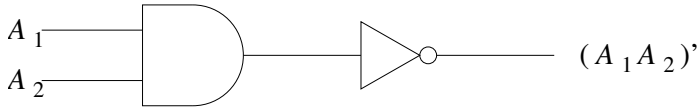
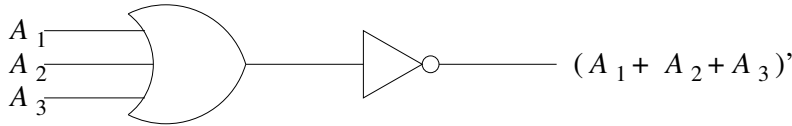


Figura 2.21: Interconexiones entre compuertas AND, OR y NOT para obtener compuertas NOR, NAND y XOR.

Comparando las dos últimas columnas de la tabla, es claro que ambas representan la misma operación. Por lo tanto, $(A_1 + A_2)(A_1A_2)' = A_1 \oplus A_2$, por lo que el tercer diagrama lógico de la Figura 2.21 es equivalente a una operación XOR.

En realidad, las compuertas que son más fáciles de implementar utilizando tecnología de semiconductores son las compuertas NOR y NAND. De hecho, todas las operaciones lógicas pueden implementarse mediante una interconexión que utiliza únicamente compuertas NOR o únicamente compuertas NAND.

La Figura 2.22 muestra las interconexiones de compuertas NOR para obtener compuertas NOT, OR y AND.

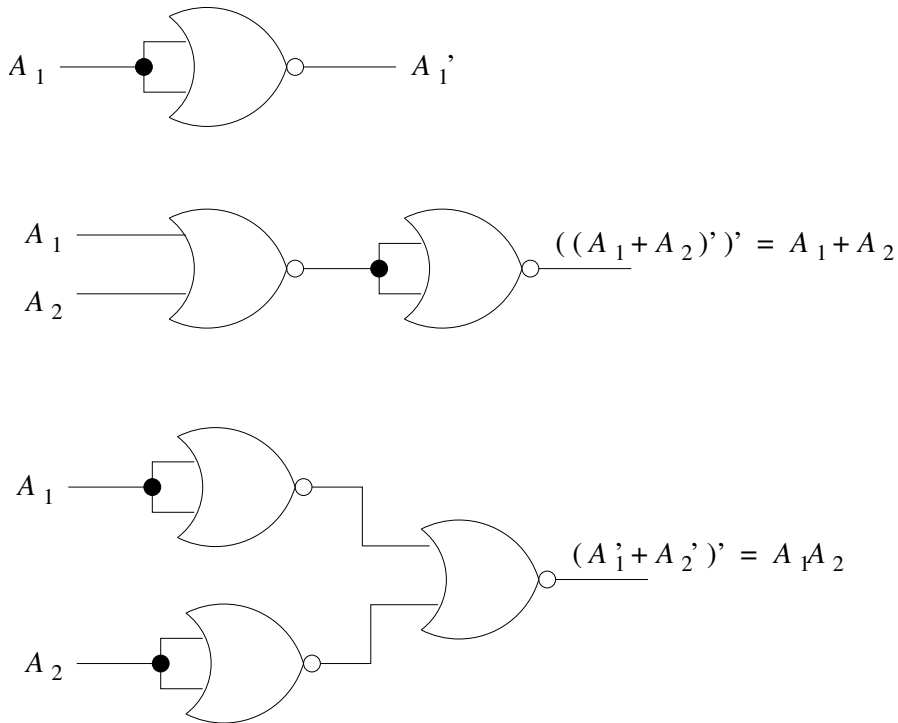


Figura 2.22: Interconexiones entre compuertas NOR para obtener compuertas NOT, OR y AND.

Una compuerta NOT se obtiene mediante conectar las dos entradas de una compuerta NOR. Así, si la variable de entrada A_1 tiene valor 1, la salida de la compuerta NOR es 0; si la variable de entrada A_1 tiene valor 0, la salida de la compuerta NOR es 1. Por tanto, la salida es efectivamente la negación de la entrada, lo que equivale a una compuerta NOT.

La segunda interconexión de compuertas NOR de la Figura 2.22 implementa la operación de una compuerta OR. Si se realiza la negación dos veces, se obtiene la variable original, es decir:

$$(A')' = A$$

De este modo, la salida de la compuerta NOR se niega con otra compuerta NOR conectada como compuerta NOT. Así, negando la salida de la NOR, se obtiene una compuerta OR.

La tercera interconexión de la Figura 2.22 representa una compuerta AND. Dado que se trata de una interconexión más complicada, se recurre a una tabla de verdad para comprobar su operación:

A_1	A_2	A_1'	A_2'	$A_1' + A_2'$	$(A_1' + A_2')'$	$A_1 A_2$
0	0	1	1	1	0	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	0	1	1

Cuando se comparan las dos últimas columnas, es notorio que:

$$(A_1' + A_2')' = A_1 A_2$$

por lo que tal interconexión representa una operación AND.

De forma similar, se muestra a continuación cómo las compuertas NOT, AND y OR pueden obtenerse mediante una interconexión de compuertas NAND, como se muestra en la Figura 2.23.

La primera interconexión de la Figura 2.23 muestra la implementación de una compuerta NOT mediante una compuerta NAND cuyas entradas han sido conectadas juntas. Por tanto, cuando la variable de entrada A_1 tiene valor 1, la salida de la compuerta NAND es 0. Similarmente, si la variable de entrada A_1 tiene valor 0, la salida de la compuerta NAND es 1. Por tanto, se obtiene una operación NOT.

La segunda interconexión de la Figura 2.23 representa una operación AND. Esto es notorio debido a que la operación NAND es solamente la negación de la operación AND. De tal modo, al obtener la negación de la negación, se obtiene la operación deseada AND.

Finalmente, se puede mostrar que la tercera interconexión de la Figura 2.23 es una operación OR mediante la siguiente tabla de verdad:

A_1	A_2	A_1'	A_2'	$A_1'A_2'$	$(A_1'A_2')'$	$A_1 + A_2$
0	0	1	1	1	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	0	1	1

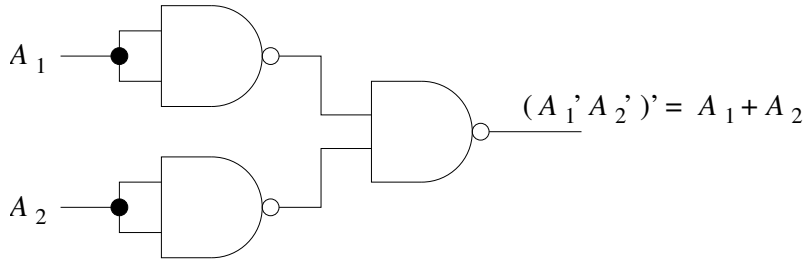
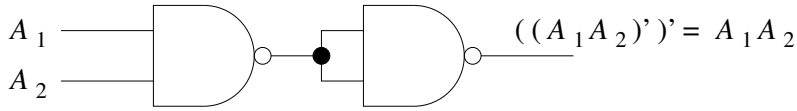
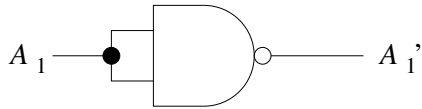


Figura 2.23: Interconexiones entre compuertas NAND para obtener compuertas NOT, AND y OR.

Al comparar las últimas dos columnas, es notorio que:

$$(A_1'A_2')' = A_1 + A_2$$

lo que indica que tal interconexión representa una operación OR.

2.2.5. Identidades básicas y leyes del álgebra booleana

El álgebra booleana facilita el análisis y diseño de circuitos digitales, ya que provee herramientas convenientes para:

- expresar en forma algebraica una tabla de verdad que relaciona variables binarias;
- expresar en forma algebraica la relación de entrada/salida de los diagramas lógicos;
- encontrar circuitos más sencillos para la misma función.

Mediante manipular una expresión booleana de acuerdo con las reglas del álgebra booleana, se puede obtener una expresión más simple de una función que requiere menos compuertas en su implementación. Las identidades básicas y leyes del álgebra booleana mediante las cuales estas manipulaciones pueden realizarse se presentan a continuación.

1. Leyes conmutativas

$$\begin{aligned}A + B &= B + A \\AB &= BA\end{aligned}$$

2. Leyes asociativas

$$\begin{aligned}(A + B) + C &= A + (B + C) = A + B + C \\(AB)C &= A(BC) = ABC\end{aligned}$$

3. Leyes distributivas

$$\begin{aligned}A(B + C) &= (AB) + (AC) \\A + (BC) &= (A + B)(A + C)\end{aligned}$$

4. Otras leyes

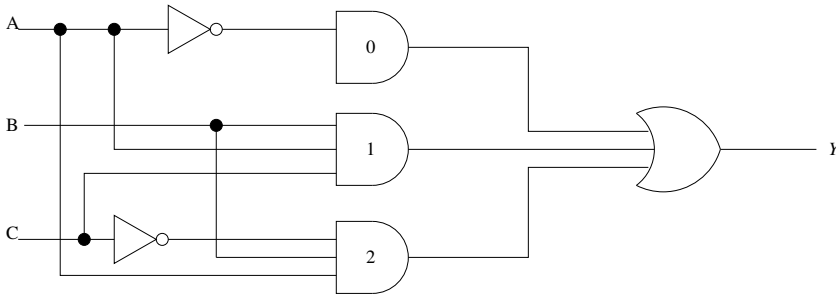
$$\begin{aligned}A + A &= A \\AA &= A \\A + 1 &= 1 \\A \cdot 1 &= A \\A + 0 &= A\end{aligned}$$

$$\begin{aligned}
 A \cdot 0 &= 0 \\
 A + A' &= 1 \\
 AA' &= 0 \\
 A'' &= A
 \end{aligned}$$

5. Leyes de De Morgan

$$\begin{aligned}
 (A + B)' &= A'B' \\
 (AB)' &= A' + B'
 \end{aligned}$$

El siguiente ejemplo muestra cómo la manipulación mediante álgebra booleana puede utilizarse para simplificar circuitos digitales. Considérese el siguiente circuito:



La salida de este circuito puede expresarse algebraicamente como:

$$Y = ABC + ABC' + A'C$$

La expresión puede simplificarse usando las identidades del álgebra booleana:

$$\begin{aligned}
 Y &= ABC + ABC' + A'C \\
 &= AB(C + C') + A'C \\
 &= AB + A'C
 \end{aligned}$$

Nótese que, a partir de las identidades anteriores $C + C' = 1$ y que $AB \cdot 1 = AB$. Como ejercicio, dibújese el diagrama lógico del circuito simplificado.

2.2.6. Formas canónicas

Una forma canónica es la **manera estándar** de escribir una ecuación matemática.

Cualquier combinación de circuitos puede expresarse en una de dos formas básicas: suma de productos (ORs de ANDs) o producto de sumas (ANDs de ORs). De ambas, solo la suma de productos se discute aquí, dado que es la más comúnmente utilizada para representar circuitos digitales. Esta forma canónica se compone de un conjunto de ecuaciones booleanas, una por cada salida del circuito, en la que cada ecuación es una suma booleana de mintérminos, que son términos de productos booleanos en los que cada entrada, como variable booleana, y en los que aparece exactamente una vez cada variable, ya sea en forma positiva o negativa. Por cada combinación de entradas que tengan una salida 1, el mintérmino correspondiente debe aparecer en la expresión canónica de la función booleana.

Considérese por ejemplo la siguiente tabla de verdad de un voto por mayoría, donde A , B , y C son entradas, y Y es la salida del circuito. La salida tiene valor de 1 si y solo si dos o más entradas tienen valor de 1.

A	B	C	Y	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	mintérmino
1	0	0	0	
1	0	1	1	mintérmino
1	1	0	1	mintérmino
1	1	1	1	mintérmino

La forma canónica de la función booleana de la tabla anterior es:

$$Y = A'BC + AB'C + ABC' + ABC$$

Nótese cómo la forma canónica puede deducirse en forma de suma de productos, ya que refleja que la función se describe como una suma booleana de mintérminos, que a la vez son productos booleanos de las entradas. La forma

canónica permite el desarrollo de algoritmos sencillos para sintetizar circuitos digitales, comenzando de un conjunto de ecuaciones booleanas y finalizando con la descripción de un circuito digital.

2.2.7. Minimización de circuitos: mapas de Karnaugh

Mucha maquinaria, dispositivos y aparatos actuales requieren un control lógico: máquinas vendedoras, sistemas de encendido y carburación de automóviles, elevadores, y otros. Todos ellos reciben y emiten señales que realizan una labor. Por ejemplo, el dispositivo de control de un elevador recibe peticiones de los diferentes pisos como entradas, y genera señales de control al motor como salida (Figura 2.24). ¿Qué tan complicada es la función esencial de control? Sorprendentemente, ésta resulta ser relativamente simple.

El problema de minimizar un circuito lógico se conoce como minimización booleana, y una de las mejores técnicas para resolver pequeños ejemplos de problemas es la técnica de mapas de Karnaugh. Tales mapas proveen de visualizaciones bidimensionales de funciones booleanas, que son relativamente fáciles de inspeccionar, guiando directamente a una expresión sencilla para la función; mientras más sencilla sea la expresión, más simple es el circuito (véase el capítulo anterior). Los mapas de Karnaugh resultan útiles para funciones con dos, tres y cuatro variables lógicas; sin embargo, su complejidad aumenta considerablemente para funciones con cinco o más variables lógicas.

Específicamente, un **mapa de Karnaugh** es un arreglo bidimensional de celdas, cada una de las cuales representa un producto específico de las variables y sus complementos.

Por ejemplo, un mapa de dos variables lógicas tiene cuatro celdas para los productos de sus dos variables. Considérese, por ejemplo, dos variables x_1 y x_2 . Los productos del mapa serían $x_1'x_2'$, $x_1'x_2$, x_1x_2' , y x_1x_2 . El mapa se representa como se muestra en la Figura 2.25.

Cada renglón y columna del mapa corresponden a un valor de cada variable lógica. Estos valores se asignan de tal modo que produzcan un 1 cuando se substituyen en el producto que se encuentra en la intersección de su renglón y columna respectivos. Por ejemplo, el renglón $x_2 = 1$ intersecta la columna $x_1 = 0$ en $x_1'x_2$,

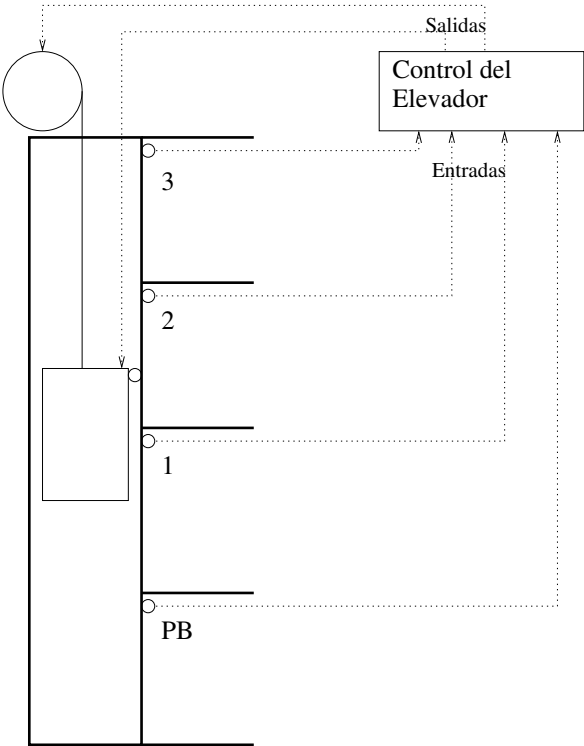


Figura 2.24: Un sistema elevador

		x_1	
		0	1
x_2	0	$x_1' x_2'$	$x_1 x_2'$
	1	$x_1' x_2$	$x_1 x_2$

Figura 2.25: Un mapa de Karnaugh de dos variables.

y para estos valores $x_1'x_2 = 1$. La razón de este arreglo se hace notoria cuando se muestre cómo se usan los mapas para la minimización.

Considere la función $f(x_1, x_2)$ escrita en forma normal disyuntiva como $x_1x_2' + x_1'x_2 + x_1x_2$. Para cada producto que aparece en esta expresión, se coloca un 1 en la celda correspondiente del mapa de Karnaugh de dos variables (Figura 2.26).

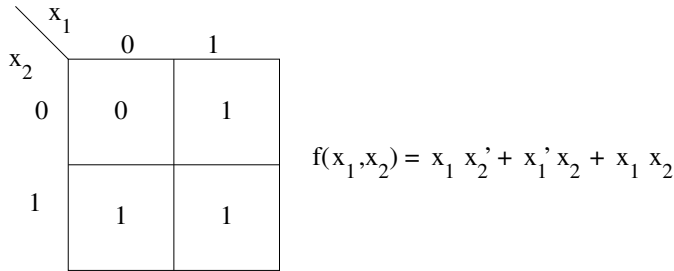


Figura 2.26: Un mapa de Karnaugh para $f(x_1, x_2)$.

Nótese que celdas adyacentes corresponden a productos con factores comunes. Por ejemplo, las dos celdas en la parte inferior corresponde a $x_1'x_2$ y x_1x_2 . Estos productos tienen como factor común a x_2 , por lo que podemos escribir:

$$\begin{aligned} x_1x_2 + x_1'x_2 &= x_2(x_1 + x_1') \\ &= x_2 \end{aligned}$$

obteniendo una simplificación en parte de la expresión para f . De acuerdo con esto, podemos ligar las dos celdas con un rectángulo. De manera similar, es posible ligar los unos de la segunda columna (Figura 2.27).

En esencia, lo que se hace con la expresión para f en forma gráfica es lo siguiente:

$$\begin{aligned} f(x_1, x_2) &= x_1x_2' + x_1'x_2 + x_1x_2 \\ &= x_1x_2' + x_1x_2 + x_1'x_2 + x_1x_2 \\ &= x_1(x_2' + x_2) + (x_1' + x_1)x_2 \\ f(x_1, x_2) &= x_1 + x_2 \end{aligned}$$

Para evitar toda esta manipulación algebraica, pero llegar a la misma conclusión, la regla usada en mapas de Karnaugh es simplemente substituir cada

$x_1 \backslash x_2$	0	1
0	0	1
1	1	1

Figura 2.27: Ligando celdas.

rectángulo que dibujamos sobre el mapa por el factor constante de los productos correspondientes: el segundo renglón del mapa contiene a los productos $x_1'x_2$ y x_1x_2 , de los cuales x_2 se considera la parte constante. Respecto al rectángulo vertical de la segunda columna, éste contiene a x_1x_2' y a x_1x_2 , de donde la parte constante es x_1 . De esta forma, directamente del mapa es posible deducir la minimización de la función como:

$$f(x_1, x_2) = x_1 + x_2$$

Los mapas de Karnaugh de tres variables son doblemente complicados que los mapas para dos variables. Aquí, las celdas corresponden a todos los productos posibles de las tres variables (por ejemplo, x_1 , x_2 y x_3) y sus complementos (Figura 2.28).

En este mapa, los renglones corresponden a la variable x_1 , pero las columnas corresponden a pares de valores de las variables x_2x_3 . De nuevo, estos valores se asignan de tal manera que el producto de la intersección de un renglón dado y una columna toma el valor de 1 cuando las substituciones correspondientes se hacen en el producto. Al mismo tiempo, los pares de valores asignados a las columnas tienen la interesante propiedad de que sólo un bit cambia cuando se va de un par al siguiente. Esto es cierto aun cuando se vaya del último par de regreso al primero. Tal secuencia es un ejemplo sencillo del código Gray reflejado. Si uno examina cualquier par de celdas adyacentes en el mapa, es notorio que el producto correspondiente tiene dos variables en su parte común. Es también cierto que los productos que residen dentro de una configuración cuadrada de cuatro celdas adyacentes tienen solo una variable como parte común.

Considérese la siguiente función:

$$f(x_1, x_2, x_3) = x_1'x_2x_3 + x_1'x_2x_3' + x_1x_2x_3 + x_1x_2x_3' + x_1x_2'x_3'$$

Cuando se llena el mapa de Karnaugh para tres variables, escribiendo un 1 donde la función tiene un producto, es notorio que todos los 1s se pueden cubrir con dos rectángulos (Figura 2.29).

Uno de los rectángulos envuelve al mapa, cubriendo tanto la primera columna como la cuarta, agrupando los dos 1s que las celdas contienen. Tal y como sucede en el mapa de dos variables, se extraen las partes comunes de todos los productos contenidos en un rectángulo. El cuadrado resulta simplemente en la expresión x_2 , mientras que el rectángulo horizontal resulta en la expresión x_1x_3' . Por lo tanto, la función minimizada resulta:

$$f(x_1, x_2, x_3) = x_1x_3' + x_2$$

Los mapas de Karnaugh de cuatro variables son el doble de complicados que los mapas de tres variables. Los productos son tan largos, que resulta más conveniente numerarlos de acuerdo a la combinación de valores por el que se les indexa. Por ejemplo, en lugar de escribir $x_1x_2'x_3'x_4$, escribimos el equivalente decimal a 1001, que es 9 (Figura 2.30).

		$x_2 \ x_3$			
		00	01	11	10
x_1	0	$x_1'x_2'x_3'$	$x_1'x_2'x_3$	$x_1'x_2x_3$	$x_1'x_2x_3'$
	1	$x_1x_2'x_3'$	$x_1x_2'x_3$	$x_1x_2x_3$	$x_1x_2x_3'$

Figura 2.28: Un mapa de Karnaugh para tres variables.

$x_2 \ x_3$		00	01	11	10
x_1	0	0	0	1	1
	1	1	0	1	1

Figura 2.29: Ligando celdas.

$x_3 \ x_4$		00	01	11	10
$x_1 \ x_2$	00	0	1	3	2
	01	4	5	7	6
	11	12	13	15	14
	10	8	9	11	10

Figura 2.30: Un mapa de Karnaugh para cuatro variables.

Usando la misma secuencia de código Gray reflejado tanto en renglones como columnas, las propiedades deseables de reducción de celdas adyacentes se preserva: es posible tener 2, 4, 8, o hasta 16 celdas adyacentes en varias configuraciones rectangulares. Aquí, es importante recordar en todo momento que **se busca encontrar el número más pequeño de rectángulos que cubran el mayor número de 1s para una función dada.**

Entonces, la expresión reducida correspondiente se escribe en forma de suma. Esto implica la forma minimizada de la función.

Regresando ahora al ejemplo del elevador, se analizan las operaciones lógicas requeridas y se usa la función de cuatro variables resultante para llenar el mapa de Karnaugh correspondiente. Las llamadas al elevador requiriendo servicio se codifica para todos los cuatro pisos considerados mediante un número binario de 2 bits, como sigue:

x₁	x₂	piso
0	0	planta baja
0	1	primer piso
1	0	segundo piso
1	1	tercer piso

De manera similar se codifica la posición del elevador, pero utilizando otras dos variables:

x₃	x₄	piso
0	0	planta baja
0	1	primer piso
1	0	segundo piso
1	1	tercer piso

Las salidas del circuito de control deben considerar señales para hacer que el elevador vaya hacia arriba (*up*), hacia abajo (*down*), y para detenerse (*stop*). En el presente ejemplo sólo se desarrolla el circuito para hacer que el elevador vaya hacia arriba.

El mapa de Karnaugh de cuatro variables se llena por inspección. Cada celda se examina, y si la posición actual se encuentra debajo del piso requerido, se coloca un 1 en tal celda; de otra forma, se coloca un 0 (Figura 2.31).

Tres rectángulos son suficientes para cubrir todos los 1s del mapa, y la expresión resultante para *up* es:

$$up(x_1, x_2, x_3, x_4) = x_1x_3' + x_2x_3'x_4' + x_1x_2x_4'$$

Aún cuando la expresión resultante es la mínima forma disyuntiva, todavía es posible hacer alguna simplificación algebraica:

$$x_1x_3' + x_2x_3'x_4' + x_1x_2x_4' = x_1x_3' + x_2x_4'(x_3' + x_1)$$

El circuito correspondiente tiene tan solo siete compuertas lógicas (figura 2.32).

$x_1 \ x_2 \backslash x_3 \ x_4$					
		00	01	11	10
00		0	0	0	0
01		1	0	0	0
11		1	1	0	1
10		1	1	0	0

Figura 2.31: Mapa de Karnaugh para el problema del elevador.

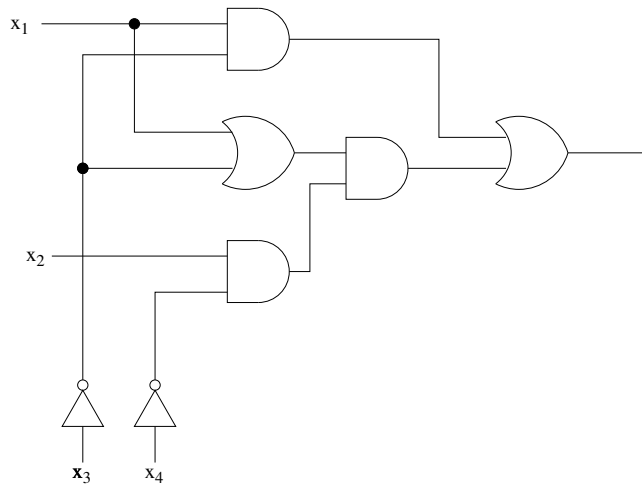


Figura 2.32: Un circuito de control para subir un elevador

Como puede observarse, el tamaño de un mapa de Karnaugh se dobla con cada variable que se añade. Debido a esto, y por la creciente complejidad de configuraciones disponibles, el método no se utiliza mucho cuando se involucran más de seis variables. Debe mencionarse, más aún, que los mapas de Karnaugh tienen la intención de usarse exclusivamente por diseñadores humanos. Debido a la habilidad humana de tomar grandes cantidades de información visualmente, el método funciona más rápido que un proceso de revisar líneas de expresiones algebraicas buscando factores comunes. Al mismo tiempo, se han desarrollado en la actualidad elaborados algoritmos para la reducción de funciones booleanas.

2.3. Circuitos combinacionales simples

2.3.1. Circuitos combinacionales

Los **circuitos combinacionales** son circuitos digitales en los que la salida refleja inmediatamente el estado de sus variables de entrada.

Para un circuito combinacional de n variables booleanas de entrada, existen 2^n combinaciones de valores. De tal modo, un circuito combinacional puede especificarse como una tabla de verdad que lista los valores de la salida dependiendo de cada combinación de las variables de entrada, que a la vez puede simplemente convertirse en una forma canónica de suma de productos.

Para todos los circuitos combinacionales, el enfoque de esta sección se mantiene en varios bloques reusables (bloques que se utilizan en más de un circuito dentro de la estructura de hardware de una computadora), los cuales proveen funciones que son muy útiles. Estos bloques se les conoce frecuentemente como bloques funcionales.

2.3.2. Decodificadores y codificadores

Un decodificador es un circuito combinacional que convierte información binaria de n entradas codificadas a un máximo de 2^n salidas únicas (Figura 2.33).

También la Figura 2.33 muestra el diagrama lógico de un decodificador de 2×4 . Dos entradas se decodifican en cuatro salidas. La tabla de verdad de este circuito se muestra a continuación:

X1	X0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Como un ejercicio, se rediseña el decodificador de 2×4 para incluir una entrada de habilitación (*enable* E), de modo que cuando $E = 0$, todas las salidas del decodificador toman el valor de 0, y cuando $E = 1$, la salida correspondiente al valor binario de la entrada toma el valor de 1, como se muestra en la siguiente tabla de verdad:

E	X1	X0	Y3	Y2	Y1	Y0
0	*	*	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

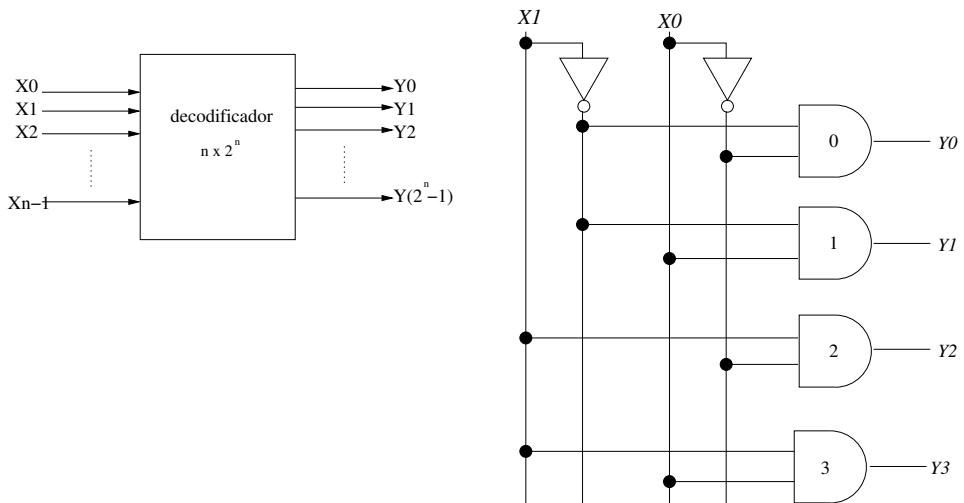


Figura 2.33: Un decodificador en general, y un ejemplo de decodificador de 2×4 .

Obtener decodificadores más complejos a partir de decodificadores más sencillos es una aplicación que utiliza la entrada de *enable*. Por ejemplo, un decodificador de 3×8 puede obtenerse mediante conectar dos decodificadores de 2×4 como se muestra en la Figura 2.34.

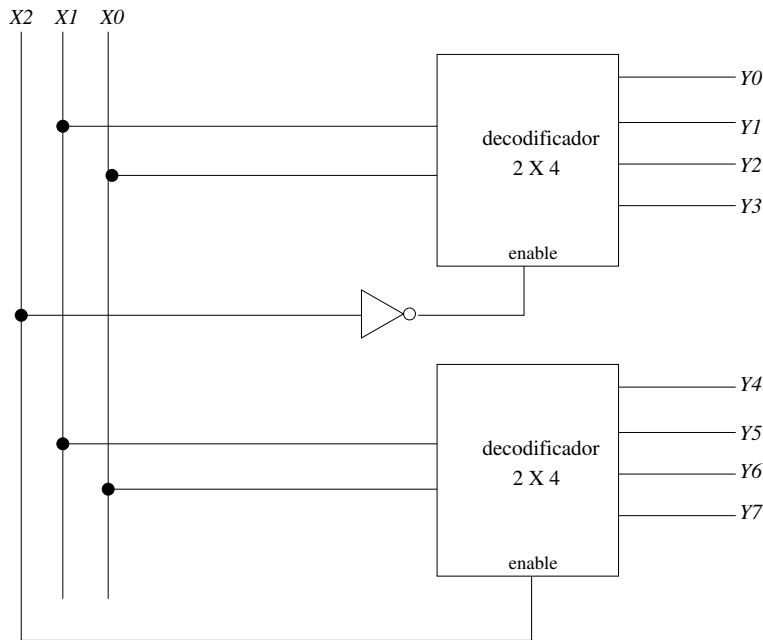


Figura 2.34: Un decodificador de 3×8 formado por dos decodificadores de 2×4 .

Un codificador es un circuito digital combinacional que realiza la función inversa del decodificador. Tiene 2^n entradas (a veces, menos) y n salidas. Un ejemplo de codificador es el codificador octal a binario, cuya tabla de verdad se muestra a continuación. Tiene ocho entradas ($D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0$), una por cada dígito octal, y tres salidas (A_2, A_1, A_0) que generan el valor binario correspondiente. Supóngase que en un momento dado, solo una entrada tiene el valor de 1.

D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

2.3.3. Multiplexores y demultiplexores

Un multiplexor es un interruptor controlado digitalmente. La Figura 2.35 muestra el diagrama de bloque de un multiplexor con cuatro entradas: a_0 , a_1 , a_2 y a_3 , y una salida b .

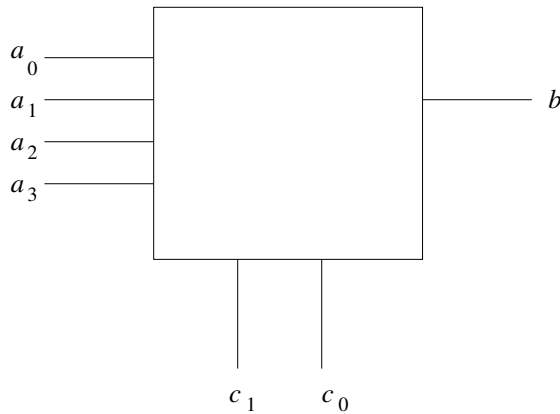


Figura 2.35: Diagrama de bloque de un multiplexor con cuatro entradas.

La función del multiplexor es conectar una de las variables de entrada a la variable de salida. La variable de entrada se selecciona mediante un valor digital que se coloca en las terminales c_1 y c_0 . La selección se hace de la siguiente forma: c_1 y c_0 se consideran bits independientes, pero en conjunción, se pueden ver como un valor binario de dos bits, siendo c_1 el bit más significativo y c_0 el bit menos significativo. Por ejemplo, si $c_1 = 1$ y $c_0 = 0$, entonces el valor de la selección es

$10_2 = 2_{10}$. Si $c_1 = 1$ y $c_0 = 1$, entonces el valor de la selección es $11_2 = 3_{10}$. Ahora bien, el valor base 10 del valor binario c_1c_0 indica el subíndice de la entrada que se conecta a b . Por ejemplo, si $c_1 = 1$ y $c_0 = 1$ ($11_2 = 3_{10}$), entonces:

$$b = a_3$$

Es decir, si $a_3 = 1$, entonces b toma el valor de 1; si $a_3 = 0$, entonces b toma el valor de 0.

Puede obtenerse una expresión lógica que describe la operación del multiplexor de cuatro entradas:

$$b = c_1'c_0'a_0 + c_1'c_0a_1 + c_1c_0'a_2 + c_1c_0a_3$$

Esta expresión muestra la operación OR entre cuatro operaciones AND. Debido a la operación OR, si cualquiera de las AND da como resultado 1, entonces b toma el valor 1. Supóngase que $c_1 = 0$ y $c_0 = 0$. Esto da como resultado los las siguientes operaciones en cada AND:

$$\begin{aligned} c_1'c_0'a_0 &= 1 \cdot 1 \cdot a_0 = a_0 \\ c_1'c_0a_1 &= 1 \cdot 0 \cdot a_1 = 0 \\ c_1c_0'a_2 &= 0 \cdot 1 \cdot a_2 = 0 \\ c_1c_0a_3 &= 0 \cdot 0 \cdot a_3 = 0 \end{aligned}$$

De tal modo, para $c_1 = 0$ y $c_0 = 0$, se tiene que:

$$b = a_0$$

Nótese que en este caso la salida b depende del valor de a_0 : si $a_0 = 1$, entonces $b = 1$; si $a_0 = 0$, entonces $b = 0$.

A partir de la ecuación que describe la operación del multiplexor de cuatro entradas, se puede construir un multiplexor utilizando compuertas lógicas, como se muestra en la Figura 2.36.

Los multiplexores se utilizan en circuitos digitales del mismo modo que los interruptores se utilizan en circuitos eléctricos ordinarios. Dado que pueden servir para la conexión de la salida con varias probables entradas, se utilizan comúnmente para la selección de información binaria.

Como los decodificadores, los multiplexores cuentan con una entrada de habilitación *enable* que controla la operación del circuito. Cuando esta entrada está en

estado inactivo, las salidas se deshabilitan, y cuando está en estado activo, el circuito funciona normalmente.

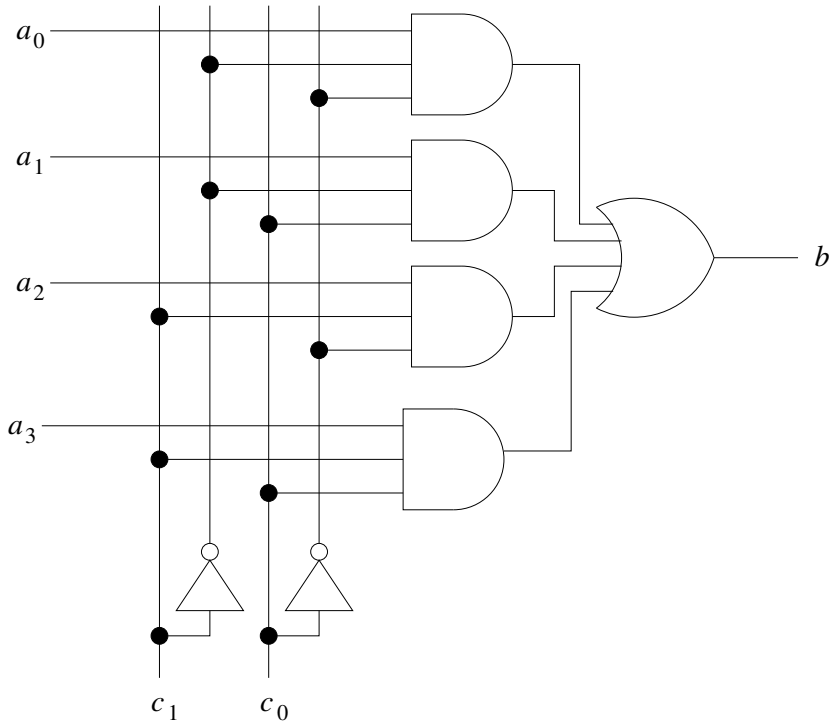


Figura 2.36: Un multiplexor con cuatro entradas.

Un demultiplexor es un circuito digital combinacional que realiza la función inversa de un multiplexor. Un demultiplexor recibe una sola entrada, y la transmite a una de sus 2^n salidas posibles. La selección de una salida específica se controla mediante la combinación binaria de n entradas de selección.

Un multiplexor puede ser utilizado para seleccionar una de n fuentes de datos binarios para transmitirse en una sola línea de comunicación, mientras que al otro lado de la misma un demultiplexor puede utilizarse para reestablecer el paso de la comunicación a uno de m posibles destinos.

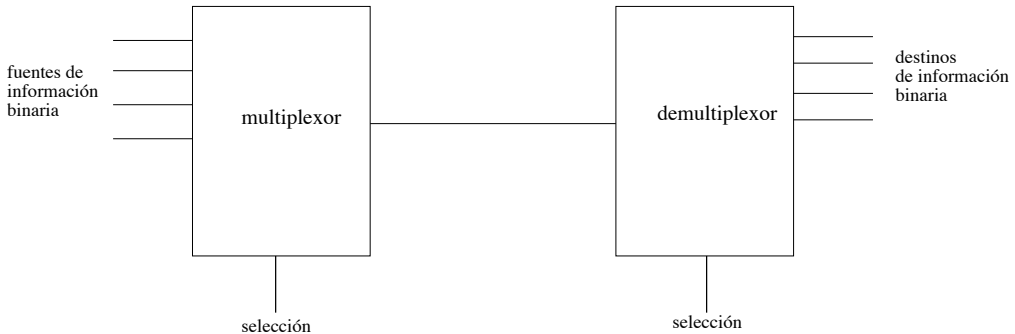


Figura 2.37: Un multiplexor y un demultiplexor utilizados en comunicaciones digitales.

2.3.4. Circuitos aritméticos. El sumador medio

Un **circuito aritmético** es un circuito digital combinacional que realiza operaciones aritméticas como adición, sustracción, multiplicación y división sobre números binarios o con números decimales en código binario.

Tras una breve introducción a los principales tipos de compuertas lógicas, en esta sección se muestra cómo las compuertas lógicas pueden interconectarse de modo que se produzca un circuito digital que realice una suma binaria. Supóngase que se desea sumar dos números de un solo bit, a_1 y b_1 . La suma se expresa de la siguiente forma:

$$\begin{array}{r} + \quad a_1 \\ \quad b_1 \\ \hline c_1 \quad s_1 \end{array}$$

Aquí, s_1 y c_1 son a la vez números de un solo bit, siendo s_1 el dígito de suma y c_1 el dígito de acarreo. Un circuito combinacional que realiza la adición de dos bits se conoce como sumador medio (*half adder*). La adición simple de dos bits se presenta como sigue:

$$0 + 0 = 00$$

$$0 + 1 = 01$$

$$1 + 0 = 01$$

$$1 + 1 = 10$$

Las primeras tres operaciones producen un resultado de un solo bit, pero la cuarta operación requiere de dos bits para su representación. Por tal razón, se utilizan siempre dos bits para la representación del resultado de la suma de dos bits: un bit representando la suma, y el otro representando el acarreo. A continuación se presenta la tabla de verdad de un sumador medio, donde a_1 y b_1 son las entradas (los bits a sumar) y c_1 y s_1 son las salidas (donde c_1 es el acarreo y s_1 la suma).

a_1	b_1	s_1	c_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Nótese que los valores binarios de s_1 y c_1 se obtienen siguiendo las reglas de la suma binaria. Considerando s_1 y c_1 separadamente, se construye a continuación un circuito lógico que cumpla con lo expresado en la tabla de verdad. Tal circuito lógico se conoce como sumador medio por razones que se discuten más adelante.

Considérese la variable c_1 . El circuito debe ser tal que produzca un valor 1 como salida sólo cuando a_1 y b_1 son ambos 1, es decir:

$$c_1 = a_1 b_1$$

Es sencillo notar que que c_1 se obtiene de aplicar la operación AND entre a_1 y b_1 , por lo que se utiliza una compuerta AND que cumple con tal funcionalidad.

Ahora bien, considérese la variable s_1 . Tal variable tiene valor de 1 en dos renglones de la tabla de verdad, de tal modo que hay dos conjuntos de entradas que producen un valor de 1 para s_1 . Un conjunto de tales entradas sucede cuando $a_1 = 0$ y $b_1 = 1$. El otro conjunto sucede cuando $a_1 = 1$ y $b_1 = 0$. De este modo, se tiene la siguiente expresión para s_1 :

$$s_1 = a_1' b_1 + a_1 b_1'$$

De acuerdo con esta ecuación, se desea que s_1 tenga valor 1:

- cuando $a_1 = 0$ y $b_1 = 1$
o
- cuando $a_1 = 1$ y $b_1 = 0$

La Figura 2.38 muestra la interconexión de compuertas para obtener las variables de salida s_1 y c_1 a partir de las variables de entrada a_1 y b_1 .

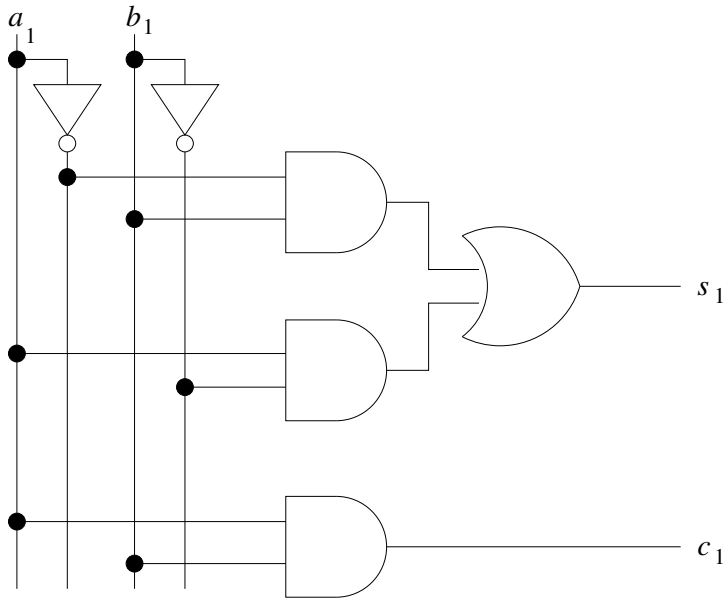


Figura 2.38: Interconexiones de compuertas para obtener un sumador medio.

El razonamiento detrás de esta interconexión se basa en considerar que la variable de salida s_1 tiene valor 1, primero, si $a_1 = 0$ y $b_1 = 1$. Pensando en términos positivos, se sabe que si $a_1 = 0$, implica que $a_1' = 1$. Ahora bien, $a_1'b_1$ tiene valor de 1 cuando a_1' AND b_1 tiene valor 1. De hecho, esto es similar que decir $a_1 = 0$ y $b_1 = 1$, que es el punto inicial de este análisis.

De forma similar, s_1 tiene valor 1 si $a_1 = 1$ y $b_1 = 0$. De nuevo, pensando en términos positivos, se sabe que si $b_1 = 0$, implica que $b_1' = 1$. Ahora bien, a_1b_1' tiene valor de 1 cuando a_1 AND b_1' tiene valor 1.

La expresión para s_1 consiste de una operación OR aplicada entre $a_1'b_1$ y a_1b_1' . Este razonamiento da como resultado el circuito de la Figura 2.38.

El sumador medio puede utilizarse para sumar únicamente dos bits; sin embargo, considérese ahora que se desea sumar dos valores binarios de más de un bit. Parecería que cada par de bits dentro de los valores binarios pudieran ser sumados utilizando sumadores medios entre ellos. Este no es el caso, ya que para sumar dos valores binarios de más de un bit siguiendo las reglas de la adición, es necesario considerar el acarreo que resulta de la suma de los dos bits anteriores. Por lo tanto, es necesario que cada sumador de dos bits considere tales valores y además el acarreo que se produce de la suma de los bits en la columna anterior. Un sumador que toma en cuenta tal acarreo se conoce como sumador completo (*full-adder*). De este modo, para sumar dos valores binarios de más de un bit, se utilizan sumadores completos para cada par de bits. Por ejemplo, si se desea sumar dos números de 8 bits cada uno, son necesarios 8 sumadores completos.

2.3.5. El sumador completo

Un sumador completo realiza la suma aritmética de tres bits de entrada produciendo dos salidas. La tercera entrada, c_{j-1} , representa el acarreo proveniente de la posición significativa anterior más baja. El sumador completo puede implementarse con dos sumadores medios y una compuerta OR.

Considérese la operación de un sumador completo. Supóngase que tal sumador adiciona los j -ésimos bits de dos valores binarios. Se tiene, entonces, que:

$$\begin{array}{r} c_{j-1} \\ + \quad a_j \\ \quad b_j \\ \hline c_j \quad s_j \end{array}$$

Nótese que c_{j-1} es el acarreo de la columna anterior $j - 1$, que es la columna inmediatamente a la derecha de la columna de bits que se suma aquí. De nuevo, utilizando las reglas de la adición binaria, se tiene la siguiente tabla de verdad para el sumador completo:

a_j	b_j	c_{j-1}	s_j	c_j
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Mediante el razonamiento utilizado para obtener el circuito del sumador medio, se obtiene el circuito de la Figura 2.39.

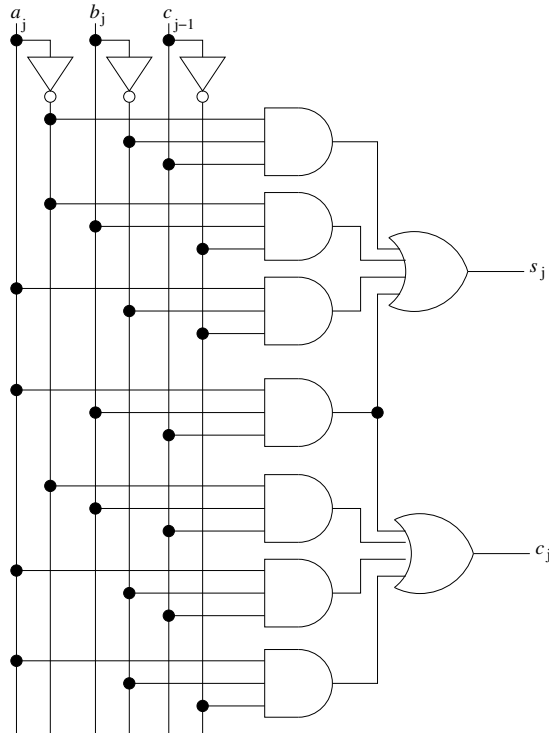


Figura 2.39: Interconexiones de compuertas para obtener un sumador completo.

Cada 1 en la columna de s_j resulta en una compuerta AND en el circuito lógico para s_j , Similarmente, cada 1 en la columna de c_j resulta en una compuerta AND en el circuito lógico para c_j . De este modo, se pueden escribir las ecuaciones para ambas variables de salida del sumador completo:

$$\begin{aligned} s_j &= a_j'b_j'c_{j-1} + a_j'b_jc_{j-1}' + a_jb_j'c_{j-1}' + a_jb_jc_{j-1} \\ c_j &= a_j'b_jc_{j-1} + a_jb_j'c_{j-1} + a_jb_jc_{j-1}' + a_jb_jc_{j-1} \end{aligned}$$

En la Figura 2.39, s_j se implementa mediante cuatro compuertas AND, cada una con tres entradas, cuyas salidas se conectan a la vez a una compuerta OR de cuatro entradas. Similarmente, c_j requiere de cuatro compuertas AND conectadas a una compuerta OR; sin embargo, nótese que en la figura sólo se utilizan siete compuertas AND, dado que s_j y c_j pueden compartir la salida de una compuerta AND. La intención aquí es reducir el número de compuertas que se conectan en el circuito.

2.3.6. El sumador binario de n bits

Un sumador paralelo binario utiliza n sumadores completos en paralelo, donde todos los bits de entrada se operan simultáneamente para producir una suma binaria. Los sumadores completos se conectan de modo que la salida de acarreo de uno se conecta a la entrada de acarreo del siguiente. Mientras mayor sea el número de bits a sumar, mayor es el número de sumadores requeridos.

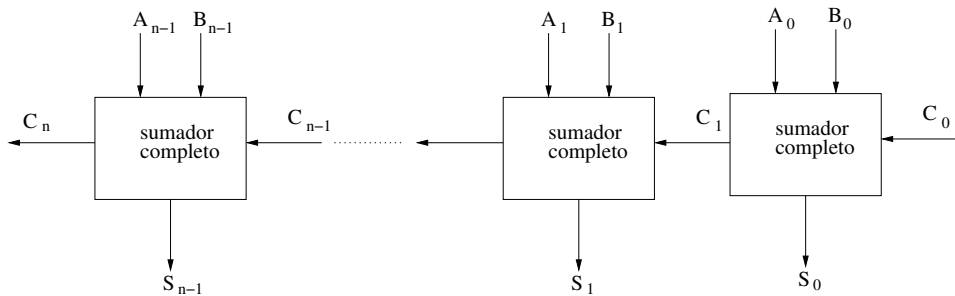


Figura 2.40: Un sumador binario de n bits.

2.3.7. Substractores

Los circuitos de sustracción pueden diseñarse siguiendo el mismo principio para el sumador medio y el sumador completo; sin embargo, con la finalidad de mantener la modularidad y reúso de bloques, en la práctica la sustracción se realiza mediante el uso del complemento a dos y sumadores completos. Considérese dos enteros A y B. Para restar B de A, se requieren los siguientes pasos:

1. Calcular $(-B)$ mediante encontrar el complemento a dos de B.
2. Sumar el resultado a A.

El complemento a dos de un número binario se obtiene mediante complementar cada uno de sus bits individualmente, y sumar uno al bit menos significativo:

$$-B = B' + 1$$

2.3.8. Multiplicación y división

La multiplicación de dos números binarios se realiza mediante multiplicar todas las combinaciones de los dígitos individuales y sumarlos en las posiciones apropiadas. Un ejemplo de tal procedimiento que se sigue para multiplicación decimal y binaria se muestra a continuación. Supóngase que se desea multiplicar 21 y 43:

$$21 \times 43 = (2 \times 4) \times 10^2 + (2 \times 3) \times 10^1 + (1 \times 4) \times 10^1 + (1 \times 3) \times 10^0$$

Aplicando el mismo principio para dos números binarios de dos bits, $A = A_1A_0$ y $B = B_1B_0$, se obtiene:

$$A_1A_0 \times B_1B_0 = (A_1 \times B_1) \times 2^2 + (A_1 \times B_0) \times 2^1 + (A_0 \times B_1) \times 2^1 + (A_0 \times B_0) \times 2^0$$

Como todos los bits involucrados son binarios, las multiplicaciones pueden reemplazarse por ANDs, y los múltiplos de 2 pueden reemplazarse por corrimientos a la derecha.

La división no se construye normalmente como un circuito combinacional. En lugar de eso, se realiza proceduralmente mediante un circuito secuencial, o utilizando código de máquina.

2.4. Lógica secuencial y máquinas de estado finito

Esta sección discute los circuitos lógicos secuenciales y sus usos en los sistemas de cómputo. Muestra cómo circuitos secuenciales sencillos pueden construirse a partir de compuertas, dando algunos ejemplos de diseño de circuitos secuenciales.

En muchos diseños digitales hay la necesidad de circuitos lógicos cuya salida depende no sólo de las entradas presentes, sino también de la historia pasada o secuencia de acciones del circuito. Esto se puede lograr mediante construir circuitos con memoria, los cuales son capaces de almacenar información acerca de la historia pasada del propio circuito. Tales circuitos se conocen como **circuitos de lógica secuencial**.

Un ejemplo sencillo del uso de tales circuitos secuenciales es un semáforo para control de tráfico vehicular (Figura 2.41).

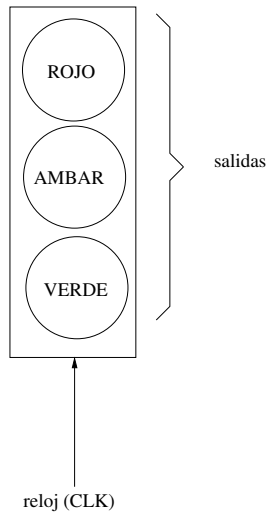


Figura 2.41: Un semáforo para tráfico vehicular.

El semáforo funciona como sigue: si el color que se muestra es verde, en alguno de las siguientes señales de reloj (CLK) debe cambiar a ámbar, y luego a rojo. Después de un tiempo, el rojo debe cambiar de nuevo a verde. Nótese que el siguiente color a mostrarse depende del color que se muestra actualmente.

El **estado** de un sistema se refiere a su condición interna en un momento en particular, que se describe mediante el valor de sus salidas o variables.

El semáforo requiere de cuatro estados: S_0 , S_1 , S_2 y S_3 (Figura 2.42). El circuito que lo controla va cambiando de estado a estado bajo el control de la señal de reloj CLK. De tal modo, estos circuitos secuenciales se les conoce como síncronos.

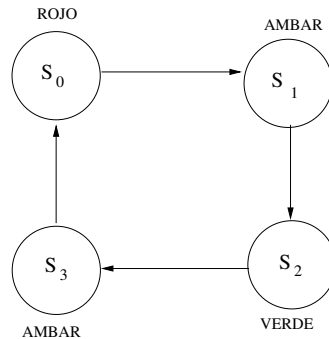


Figura 2.42: Estados de un semáforo.

En el diagrama de estados de la Figura 2.42, las flechas significan transiciones de un estado a otro, que suceden por solo recibir un pulso del reloj CLK.

Un circuito secuencial con n variables de estado tiene hasta 2^n posibles estados. Como el número de posibles estados es siempre finito, los circuitos secuenciales se les conoce comúnmente como máquinas de estado finito (*Finite State Machines* o FSM). El comportamiento de una FSM se describe mediante tablas de estado, que normalmente contienen las siguientes columnas: entradas, estado presente, estado siguiente, y salidas. Todo renglón de una tabla de estados representa una de todas las posibles combinaciones de estado presente y entradas.

Supóngase que se requiere diseñar un contador binario que realice la secuencia 0, 1, 2, 3. El primer paso en el procedimiento de diseño es traducir la especificación del circuito en un diagrama de estados. El diagrama de estados se convierte entonces en una tabla de estados, a partir de la cual se diseña el circuito lógico. Como en este ejemplo el circuito tiene que contener cuatro estados, 0, 1, 2 y 3, es suficiente considerar un número binario de dos bits para los estados, que por tanto son: 00, 01, 10 y 11. El circuito tiene que cambiar de estado por cada señal de reloj y cuando una entrada externa X tenga el valor de 1 (verdadero). Cuando X tiene valor 0 (falso), no hay cambio. Estos valores de entrada sirven como etiquetas de la flecha de transición a la que se refiere (Figura 2.43).

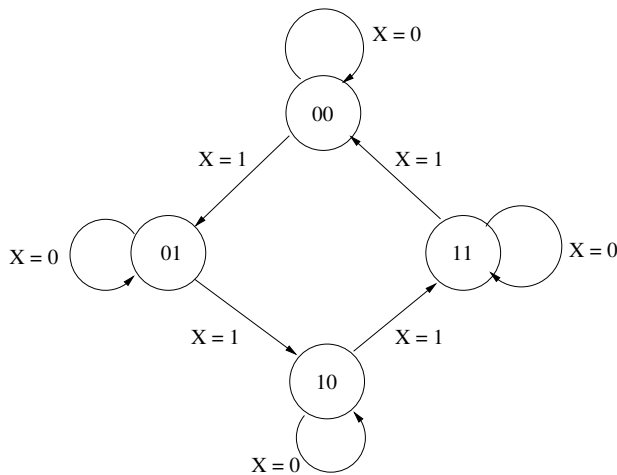


Figura 2.43: Estados de un contador.

La tabla de estados se presenta enseguida, donde A y B representan las variables de estado, y X es la entrada externa. Nótese que en la tabla $A+$ y $B+$ representan los valores de las variables de estado en el siguiente estado.

A	B	X	A+	B+
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Para implementar esto, es necesario ahora conocer algunos otros componentes digitales que permiten almacenar la información binaria de estos estados, es decir, un dispositivo básico de memoria.

2.5. Flip-flops

El circuito secuencial más sencillo es el **flip-flop**, que es un elemento de memoria y/o almacenamiento de un solo bit.

Hasta este punto, los circuitos digitales que se han discutido son conocidos bajo la denominación de circuitos lógicos combinacionales, en los cuales el valor de la variable de salida, en cualquier momento, depende sólo de los valores de las variables de entrada disponibles en ese momento. De este modo, estos circuitos no tienen memoria, es decir, el valor de las variables de salida no dependen de los valores en el pasado de las variables de entrada. El valor de la variable salida, entonces, se dice que es únicamente una combinación lógica de los valores actuales de las variables de entrada.

En esta sección se inicia la descripción de circuitos digitales que sí cuentan con una capacidad de memoria. Tales circuitos se conocen con el nombre de circuitos lógicos secuenciales. Particularmente, en esta sección se discuten algunos circuitos secuenciales básicos llamados flip-flops, los cuales representan importantes bloques básicos de construcción de muchos otros circuitos secuenciales de mayor tamaño.

Un flip-flop o multivibrador biestable es un circuito cuya variable de salida permanece con un valor de 0 o 1 hasta que uno o más valores binarios se apliquen a sus terminales de entrada, en cuyo caso el valor de la variable de salida cambia en

el siguiente tiempo. Por ejemplo, si la salida tiene valor 0, permanece en tal valor hasta que los valores binarios adecuados se apliquen a sus entradas, causando su cambio a un valor de 1. En forma similar, si la salida tiene valor 1, permanece en tal valor hasta que los valores binarios adecuados se apliquen a sus entradas, causando su cambio a un valor de 0. Ya que la salida del flip-flop no cambia hasta que los valores binarios adecuados se apliquen a sus entradas, este dispositivo es capaz de recordar el valor de un solo bit.

La Figura 2.44 muestra un diagrama de bloques para un flip-flop. Nótese que este dispositivo tiene dos salidas, etiquetadas Q y Q' . Como se ha indicado anteriormente, Q' representa la negación o complemento de Q . Al trabajar con computadoras, es útil y conveniente contar con valores binarios no sólo de las variables, sino también de su complemento, lo que elimina la necesidad de muchas compuertas NOT. De tal modo, la mayoría de los flip-flops se construyen para tener disponibles las dos salidas.

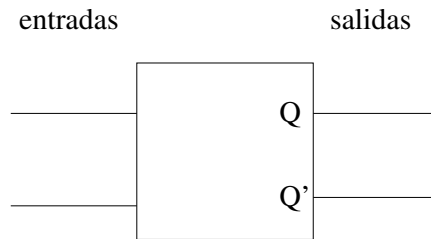


Figura 2.44: Diagrama de bloques de un flip-flop.

Comúnmente, el valor de Q es llamado el estado del flip-flop. Si $Q = 1$, entonces el estado del flip-flop es 1. Similarmente, si $Q = 0$, el estado del flip-flop es 0. A continuación, en las siguientes secciones se mencionan algunos de los flip-flops más útiles y conocidos.

2.5.1. El flip-flop RS

La Figura 2.45 muestra el diagrama de bloques de un flip-flop RS. En este flip-flop, si las variables de entrada tienen los valores $R = 0$ y $S = 0$, entonces la variable de salida Q no cambiará en el siguiente tiempo. Por ejemplo, si $Q = 1$ cuando se reciben las entradas $R = 0$ y $S = 0$, entonces Q permanece con valor 1 para el

siguiente tiempo (y obviamente, $Q' = 0$ también permanece con tal valor para el siguiente tiempo).

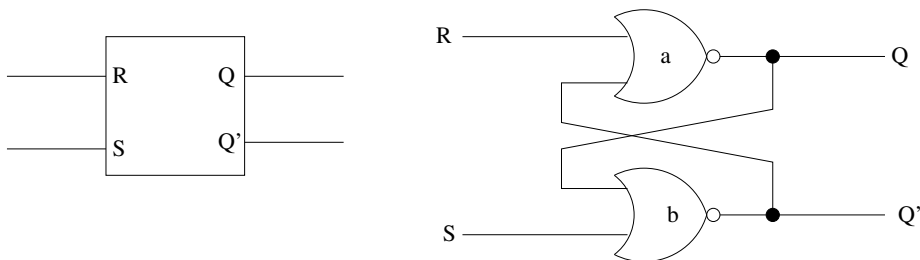


Figura 2.45: El flip-flop RS: diagrama de bloques e implementación con compuertas NOR.

Ahora bien, si $R = 0$ y $S = 1$, entonces la salida Q (el estado) del flip-flop RS toma un valor de 1 ($Q = 1$, $Q' = 0$). Sin importar cuál era el estado anterior del flip-flop RS, después de que se aplican los valores de $R = 0$ y $S = 1$ a las entradas correspondientes, el valor de su salida es $Q = 1$.

Similarmente, si los valores a las entradas del flip-flop RS se hacen $R = 1$ y $S = 0$, entonces el estado del flip-flop se vuelve 0. Sin importar el valor previo de su salida, tras aplicar $R = 1$ y $S = 0$, el valor de su salida es de $Q = 0$.

Finalmente, no es permitido que los valores a las entradas de un flip-flop RS sean $R = 1$ y $S = 1$. En general, si se intenta aplicar tales entradas a un flip-flop RS, se obtienen resultados erráticos. Por ejemplo, supóngase que se tienen dos flip-flops RS contruidos por dos fabricantes diferentes. Ambos flip-flops tienen un comportamiento similar para todas las entradas permitidas; sin embargo, si se aplica $R = 1$ y $S = 1$ a estos flip-flops, pueden actuar en formas diferentes. De hecho, las designaciones de las entradas R y S tienen como origen las palabras *reset* y *set*. Así, las entradas $R = 0$ y $S = 1$ indican al flip-flop tomar el estado $Q = 1$, mientras que las entradas $R = 1$ y $S = 0$ le indican tomar el estado $Q = 0$. La pregunta es ¿qué significaría el hecho de que ambas entradas tomaran el valor de 1 al mismo tiempo?

Por otro lado, hay muchas formas de construir flip-flops. Un flip-flop RS construido con compuertas NOR se muestra en la Figura 2.45. Considérese su operación: supóngase que $Q = 0$ y $Q' = 1$, y que $R = 0$ y $S = 0$. Una entrada a la compuerta NOR a tiene valor 1. Por lo tanto, Q permanece con valor 0. Además,

ambas entradas a la compuerta NOR **b** son 0; por lo tanto, Q' permanece con valor 1. En resumen, la salida (estado) del flip-flop no cambia.

Ahora, supóngase que las entradas toman valores $R = 0$ y $S = 1$. Después de este cambio, una entrada de la compuerta **b** toma el valor de 1. Por lo tanto, su salida cambia a 0, es decir, Q' cambia a 0. Ahora bien, ambas entradas de la compuerta NOR **a** tienen valor 0, por lo que Q toma el valor 1, cambiando el estado del flip-flop como se había dispuesto.

En el caso en que las entrada tomaran valores $R = 1$ y $S = 0$ con $Q = 1$, una de las entradas a la compuerta NOR **a** toma el valor de 1, por lo que Q toma el valor 0. Después de que este cambio ocurre, las dos entradas a la compuerta NOR **b** tienen valor 0, por lo que Q' se vuelve 1. Esto implica que se han operado los cambios adecuados.

En forma similar, es posible mostrarse que cada uno de los cambios apropiados de estado suceden para cada conjunto o combinación de valores de las variables de entrada permisibles. Nótese que los cambios operados en las salidas Q y Q' no ocurren instantáneamente, o al mismo tiempo. Por ejemplo, el cambio del estado en que $Q = 0$ ($Q' = 1$) cuando se reciben las entradas $R = 0$ y $S = 1$, implica sólo que al final de la operación $Q = 1$ ($Q' = 0$); sin embargo, primero cambia Q' , tomando el valor de 0, y solo entonces Q cambia su valor a 1. En general, estos cambios ocurren muy rápido, pero siempre requieren un tiempo para realizarse.

En una computadora digital hay un gran número de circuitos diferentes, y éstos no tienden a responder al mismo tiempo o con la misma velocidad. Pero si la computadora debe funcionar apropiadamente, entonces debe mantenerse cuidadosamente un orden secuencial correcto de respuestas de los circuitos. Si se ignora este orden, podría intentarse utilizar la salida de un circuito antes de que éste complete su cambio de estado. Existen varias técnicas utilizadas para mantener la operación ordenada de una computadora, y para asegurar que todos los circuitos responden en los momentos apropiados.

Una técnica muy importante para mantener la sincronización de los circuitos lógicos de una computadora es llamado temporización por reloj (o simplemente *clocking*). Un circuito oscilador electrónico, llamado reloj, genera un tren de pulsos parecido al que se muestra en la Figura 2.46.

Cuando el pulso está presente, el nivel de la señal corresponde a un valor de 1 lógico; cuando el pulso está ausente, corresponde a un valor 0 lógico. Esto es importante, ya que la mayoría de los flip-flops se construyen de tal manera que cambian su estado solo durante el tiempo en que el pulso está presente, como en

los tiempo entre 0 y T_1 . Por lo tanto, para proveer de un control basado en un reloj, la construcción del flip-flop debe modificarse.

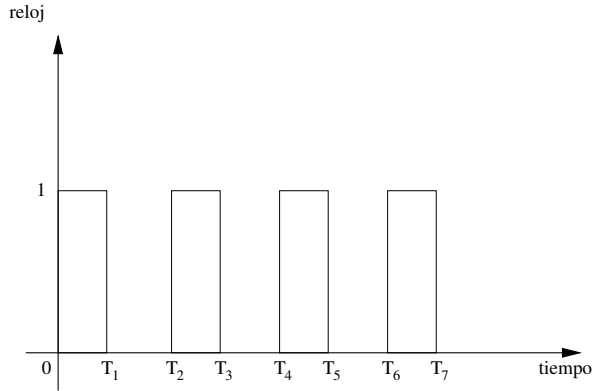


Figura 2.46: Un tren de pulsos de reloj.

El diagrama de bloques y el circuito que implementa un flip-flop RS con señal de reloj se muestra en la Figura 2.47.

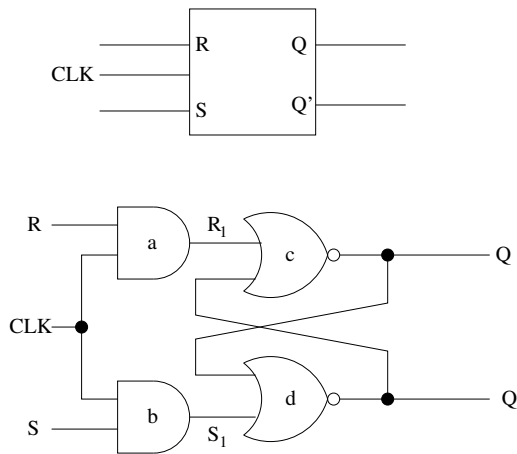


Figura 2.47: Un flip-flop RS con señal de reloj.

Las compuertas NOR **c** y **d** conforman un flip-flop RS ordinario, cuyas entradas son R_1 y S_1 . Ahora bien, considérense las compuertas AND **a** y **b**. Cuando el pulso de reloj está ausente, la entrada CLK tiene valor 0. Por lo tanto, las salidas de las compuertas AND **a** y **b** tienen valor de 0, y $R_1 = 0$ y $S_1 = 0$. Así, cuando el pulso del reloj está ausente, el flip-flop no puede cambiar su estado.

Ahora, supóngase que el pulso de reloj se encuentra presente. Su nivel es tal que actúa como un valor de 1, por lo que la entrada CLK a las compuertas AND **a** y **b** tiene valor 1. Considérese la compuerta **a**. Si $CLK = 1$ (es decir, el pulso de reloj está presente) y $R = 1$, entonces $R_1 = 1$. Si $CLK = 1$ y $R = 0$, entonces $R_1 = 0$. Por lo tanto, para los tiempos en que el pulso de reloj está presente, se tiene que:

$$\begin{aligned} R_1 &= R \\ S_1 &= S \end{aligned}$$

Entonces, si las entradas se aplican durante la presencia de un pulso de reloj, el flip-flop cambia su estado, y se comporta como un flip-flop RS sin reloj. Cuando el pulso de reloj está ausente, el flip-flop no cambia su estado, sin importar los valores de las variables de entrada R y S .

En las siguientes secciones se describen algunos otros tipos particulares de flip-flops, los cuales se consideran todos con señal de reloj.

2.5.2. El flip-flop D

El flip-flop D se diseña con una sola entrada. Su diagrama de bloques se muestra en la Figura 2.48.

Si durante el pulso de reloj $D = 1$, entonces el estado del flip-flop toma el valor de 1 ($Q = 1$, $Q' = 0$). En forma similar, si durante el pulso de reloj $D = 0$, entonces el estado del flip-flop toma el valor de 0 ($Q = 0$, $Q' = 1$). Parecería que este tipo de circuitos no son muy útiles, pero como se ve más adelante, hay varios circuitos en que el flip-flop D resulta de mucha utilidad práctica.

Un flip-flop D se construye utilizando una compuerta NOT y un flip-flop RS conectados como se muestra en la Figura 2.48. Nótese que cuando $D = 0$, entonces $R = 1$ y $S = 0$. Esto causa que el estado del flip-flop RS tome el valor de 0 ($Q = 0$). En forma similar, cuando $D = 1$, entonces $R = 0$ y $S = 1$, lo que causa que el estado del flip-flop RS tome el valor de 1 ($Q = 1$).

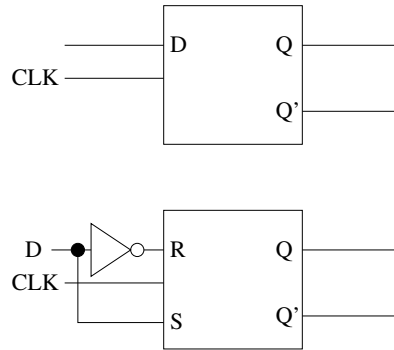


Figura 2.48: Un flip-flop D y su implementación mediante flip-flop RS.

2.5.3. El flip-flop JK

En ocasiones, es conveniente contar con un flip-flop que funcione como un flip-flop RS, pero que pueda aceptar como valores de entrada $R = 1$ y $S = 1$. Tal flip-flop se conoce con el nombre de flip-flop JK. En este flip-flop, la variable de entrada J tiene la misma función que S y la variable de entrada K tiene la misma función que R . Cuando se aplican las entradas $J = 1$ y $K = 1$ durante el pulso de reloj, el flip-flop cambia su estado al estado complementario. Esto es, si $Q = 1$ y se aplica $J = 1$ y $K = 1$, entonces el flip-flop cambia al estado $Q = 0$. Similarmente, si $Q = 0$ y se aplica $J = 1$ y $K = 1$, entonces el flip-flop cambia al estado $Q = 1$.

El diagrama de bloques de un flip-flop JK se muestra en la Figura 2.49, así como su implementación utilizando un flip-flop RS y dos compuertas AND.

El flip-flop JK funciona de la siguiente manera: supóngase que $Q = 0$, $J = 0$ y $K = 0$. Las salidas de las compuertas AND **a** y **b** toman un valor de 0. Por tanto, $R = 0$ y $S = 0$, y el estado permanece sin cambio ($Q = 0$).

Ahora, supóngase que durante el pulso de reloj se tiene que $Q = 0$, $J = 1$ y $K = 0$. Esto implica que todas las entradas de la compuerta AND **b** tienen valor 1, y por tanto, $S = 1$ para el flip-flop RS. Por otro lado, dos entradas a la compuerta AND **a** tienen valor 0, y por lo tanto, $R = 0$ para el flip-flop RS; sin embargo, la combinación de entradas $R = 0$ y $S = 1$ causa que el flip-flop RS cambie al estado $Q = 1$ (con $Q' = 0$). Nótese que una vez que el cambio de estado

ha ocurrido, al menos una entrada de cada compuerta AND es 0, de tal modo que el estado no continúa cambiando.

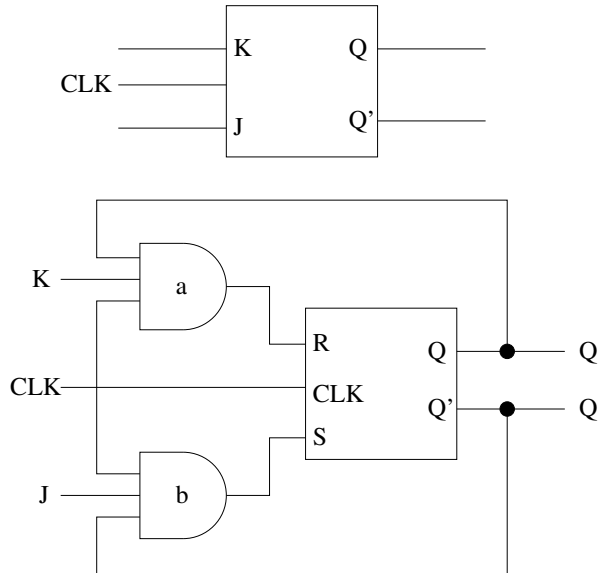


Figura 2.49: Un flip-flop JK y su implementación mediante flip-flop RS y compuertas AND.

En seguida, supóngase que $Q = 1$ ($Q' = 0$) y que los valores de las entradas durante el pulso son $J = 0$ y $K = 1$. Todas las entradas a la compuerta AND **a** tienen valor 1, por lo que $R = 1$. Dos entradas a la compuerta AND **b** tienen valor 0, por lo que $S = 0$. La combinación $R = 1$ y $S = 0$ hace que el flip-flop RS cambie de estado a 0 ($Q = 0$ y $Q' = 1$). Continuando con las combinaciones, se demuestra que el flip-flop JK funciona correctamente para todas las combinaciones de entrada y salida.

Considérese la operación del flip-flop JK cuando sus entradas son $J = 1$ y $K = 1$. Supóngase que el estado es $Q = 1$. Durante el pulso de reloj, todas las entradas para la compuerta AND **a** tienen valor 1, por lo que $R = 1$. Por otro lado, ya que $Q' = 0$, al menos una entrada a la compuerta AND **b** tiene valor 0, y por esto, $S = 0$. Tal combinación de valores de entrada al flip-flop RS provocan que éste cambie de estado a $Q = 0$.

En forma complementaria, supóngase que el estado del flip-flop JK es $Q = 0$, y se tiene la combinación $J = 1$ y $K = 1$ durante el pulso de reloj. Al menos una entrada a la compuerta AND **a** tienen valor 0, por lo que $R = 0$, mientras que todas las entradas a la compuerta AND **b** tienen valor 1, por lo que $S = 1$. Tal combinación de valores de entrada al flip-flop RS provocan que éste cambie de estado a $Q = 1$.

Aun cuando el circuito de la Figura 2.49 funciona propiamente como flip-flop JK para las combinaciones de entrada y salida, un problema puede surgir cuando $J = 1$ y $K = 1$. Supóngase que el flip-flop puede cambiar su estado en un tiempo mucho menor que el pulso de reloj. Cuando se tiene a las entradas $J = 1$ y $K = 1$, el flip-flop JK simplemente cambia su estado. Pero como el pulso de reloj puede todavía estar presente, el flip-flop JK puede cambiar de nuevo su estado. Esto en la mayoría de los casos es indeseable. Es por esto que más adelante se discuten circuitos que eliminan este tipo de comportamiento errático.

2.5.4. El flip-flop T

Otro flip-flop con una sola entrada es el flip-flop T, que se muestra en la Figura 2.50.

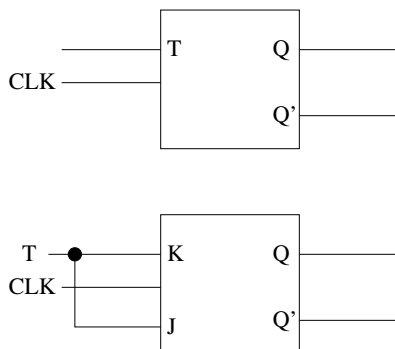


Figura 2.50: Un flip-flop T y su implementación mediante flip-flop JK.

Si $T = 1$ durante el pulso de reloj, el flip-flop cambia de estado. Si $T = 0$, el flip-flop permanece en su estado original. La implementación del flip-flop T se muestra también en la Figura 2.50. Esta implementación consiste de un flip-flop

JK cuyas entradas han sido conectadas juntas. Nótese que cuando $T = 0$, $J = 0$ y $K = 0$, y el estado del flip-flop no cambia. Cuando $T = 1$, entonces $J = 1$ y $K = 1$, por lo que el flip-flop cambia a su estado complementario.

2.5.5. Entradas de inicialización

Los flip-flops en forma de circuitos integrados frecuentemente cuentan con otras entradas de inicialización llamadas *preset* y *clear*. Cuando se aplica un valor lógico 1 a la entrada de *preset*, el flip-flop toma el estado $Q = 1$ ($Q' = 0$). Cuando se aplica un valor lógico de 0 a la entrada *clear*, el estado del flip-flop se vuelve $Q = 0$ ($Q' = 1$). Generalmente, las entradas *preset* y *clear* funcionan independientemente de la entrada para el pulso de reloj, y se utilizan para inicializar el estado de los flip-flops antes de realizar un cómputo dado.

Algunas entradas *preset* y *clear* se operan en forma diferente a las que se han descrito, en el sentido de que los niveles de voltaje que corresponden a un valor 1 lógico deben aplicarse continuamente a las entradas de *preset* o *clear*. Si se desea aplicar un *preset*, la entrada correspondiente debe entonces tomar un valor 0 lógico. En forma similar, si se desea aplicar un *clear*, la entrada correspondiente debe tomar un valor de 0 lógico.

2.5.6. Señales de reloj

En la sección anterior se menciona un problema que puede generarse al aplicar las entradas $J = 1$ y $K = 1$ a un flip-flop JK. Si el tiempo que requieren los componentes del flip-flop para cambiar fuese más rápido que el periodo de tiempo de un pulso de reloj, la salida podría cambiar varias veces durante el propio pulso de reloj. Problemas similares puede surgir en otros circuitos de la computadora. Por ejemplo, supóngase que la salida de uno de los flip-flops fuera la entrada de otro, y que una señal se aplica al primer flip-flop de modo que modifique su estado; su salida cambia, por lo que la entrada al segundo flip-flop cambia.

Dependiendo de la velocidad de operación, y cuándo se aplique la señal de entrada al flip-flop, el cambio en la entrada del segundo flip-flop puede ocurrir en varios momentos. Por ejemplo, puede cambiar antes del final del pulso de reloj, o puede ocurrir cuando ya ha pasado el pulso; o posiblemente puede cambiar parcialmente al final del pulso y parcialmente en el siguiente. Si el cambio en la entrada del segundo flip-flop ocurre mucho antes que el final del pulso, entonces el

segundo flip-flop cambia su estado inmediatamente. Si el cambio en la entrada del segundo flip-flop ocurre después del pulso de reloj, el segundo flip-flop no cambia de estado durante el pulso, sino muy cerca de su final, por lo que el segundo flip-flop podría no experimentar el cambio requerido de estado. El resultado de todo esto es una operación errática; dependiendo del momento, el flip-flop puede o no cambiar su estado en cierto tiempo.

Tal operación errática no es permisible dentro de una computadora digital, donde la precisión en las operaciones es preponderante; sin embargo, se pueden evitar los problemas debidos a las diferentes velocidades de los circuitos si la operación de los flip-flops pudiera modificarse apropiadamente. Supóngase que, como anteriormente se describe, el flip-flop puede cambiar su estado en respuesta a señales de entrada aplicadas durante el pulso de reloj. Pero la salida del flip-flop no cambia hasta después de que el pulso de reloj ha pasado. En tal circunstancia, las dificultades se evitan. Considérese el ejemplo previo, donde la salida de un flip-flop se conecta a la entrada de otro flip-flop. Durante el pulso de reloj, la salida del primer flip-flop no cambia; por lo tanto, la entrada al segundo flip-flop no cambia tampoco, y el estado del segundo flip-flop permanece sin cambio. Después de que el pulso de reloj ha ocurrido, por ejemplo, entre los tiempos T_1 y T_2 de la Figura 2.46, la salida del primer flip-flop, y por tanto la entrada al segundo flip-flop, efectivamente cambia; sin embargo, el estado del segundo flip-flop no puede cambiar durante este tiempo, ya que la señal de reloj tiene como valor 0. Al siguiente pulso de reloj, el segundo flip-flop cambia su estado. Así, cualquier ambigüedad en la operación ha sido removida (Nótese que el primer flip-flop no puede cambiar su estado hasta después del siguiente pulso de reloj, por lo que no hay ambigüedad en la entrada al siguiente flip-flop).

Hay varios tipos de flip-flops que responden de esta manera. Uno de ellos se conoce con el nombre de flip-flop de disparo por flanco. Otro tipo es el flip-flop maestro-esclavo. Considérese la operación de un flip-flop RS maestro-esclavo. De hecho, todos los flip-flops pueden ordenarse en esta configuración. La configuración de tal flip-flop, que en realidad consiste de dos flip-flops RS, se muestra en la Figura 2.51

La salida del primer flip-flop (el maestro) es la entrada al segundo flip-flop (el esclavo). Por lo pronto, ignórese la señal de reloj. El flip-flop esclavo siempre toma el valor del estado del flip-flop maestro. Nótese que $Q_1 = S_2$ y que $Q_1' = R_2$. Por lo tanto, si el estado del flip-flop maestro es $Q_1 = 1$, entonces se tiene que $R_2 = 0$

y $S_2 = 1$, de tal modo que el estado del flip-flop esclavo se vuelve $Q = 1$. Una situación similar sucede si el estado del maestro es $Q = 0$.

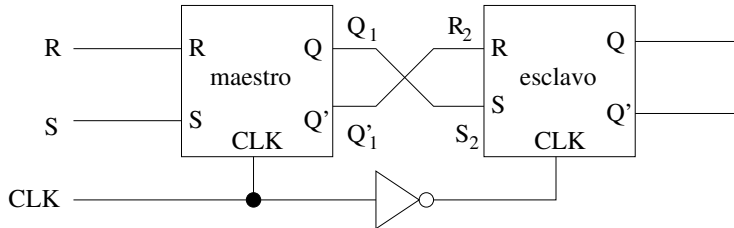


Figura 2.51: Un flip-flop RS maestro-esclavo.

Ahora, considérese el efecto de la conexión de los relojes. La terminal de reloj del flip-flop maestro se conecta directamente a la señal de reloj; sin embargo, la terminal de reloj del flip-flop esclavo es la negación de la señal de reloj. Durante el pulso de reloj, el maestro puede cambiar su estado, pero el esclavo no puede hacerlo, ya que a su entrada de reloj se encuentra un 0 lógico. Por lo tanto, su salida no cambia. Después del pulso de reloj, el estado del maestro no puede cambiar, ya que a su entrada de reloj se presenta un valor de 0 lógico; sin embargo, ahora el esclavo puede cambiar, ya que tiene un valor 1 lógico en su entrada de reloj. Así, como se desea, la salida sólo cambia cuando el pulso de reloj no se encuentra presente. En general, cuando se labora con circuitos temporizados, se asume que todos ellos disparan por flanco o trabajan en forma maestro-esclavo.

2.6. Registros

En las secciones anteriores se discuten los bloques básicos de construcción de los sistemas de cómputo; sin embargo, tales bloques básicos resultan ser elementos demasiado simples y pequeños como para describir un sistema de cómputo completo. Es por eso que en esta sección se tratan elementos de mayor tamaño, compuestos por los bloques básicos, pero que permiten describir una computadora digital.

En esta sección se discute un circuito llamado registro (*register*), que se utiliza para almacenar un número binario. En realidad, se trata de una simple memoria. En el siguiente capítulo se desarrolla más ampliamente el tema de memorias con mayor capacidad. La información se almacena tanto en registros como en todas las

memorias como una secuencia de ceros y unos. En general, esta información se le provee al registro de una de dos maneras: mediante una carga serial o secuencial, en la que un bit se introduce al registro por cada pulso de reloj (utilizando esta carga, si se desea almacenar un número binario de 8 bits, se requieren 8 pulsos de reloj para ello) y mediante una carga paralela, en la que todos los bits se introducen simultáneamente durante un solo pulso de reloj.

Los mismos procedimientos para introducir bits a un registro pueden utilizarse para descargar la información binaria del registro. Por ejemplo, supóngase que la información debe descargarse de un registro con una salida serial. Esto implica que cada bit sale por cada pulso de reloj. En una descarga paralela, todos los bits deben proveerse durante el mismo pulso de reloj.

Aun cuando evidentemente la operación paralela es más rápida que la operación serial, ésta tiene la ventaja de frecuentemente resultar en ahorros en el costo de equipo. Por ejemplo, marcar un teléfono es una operación serial. Se marca primero un número, luego el siguiente, y así hasta haber marcado todo un número telefónico. En una operación paralela, debería haber suficientes teclados para marcar cada dígito del número telefónico y, si se cuenta con suficientes manos, el número podría marcarse en un solo momento. Similarmente, dentro de las computadoras digitales, hay circuitos sumadores seriales y circuitos sumadores paralelos. En éstos últimos, se tiene un sumador completo por cada bit (véase la sección 3.4), mientras que en un sumador serial, se tiene solo un circuito que realiza la suma de cada bit en turno.

2.6.1. El registro básico

El procesamiento de datos en una computadora se realiza normalmente en palabras binarias (secuencias de unos y ceros) de tamaño fijo. Así, se requieren dispositivos capaces de almacenar un número de bits de información durante un tiempo. Un registro es un dispositivo en el cual un número de bits pueden almacenarse y recobrase.

Un arreglo de n flip-flops puede utilizarse para almacenar n bits de información, formando un registro. El diagrama esquemático de un registro de n bits se presenta (Figura 2.52). La convención utilizada es numerar los bits de 0 a $n - 1$. Además, dado un registro, se le representa con un rectángulo, que puede o no dividirse para exhibir secciones o bits individuales.

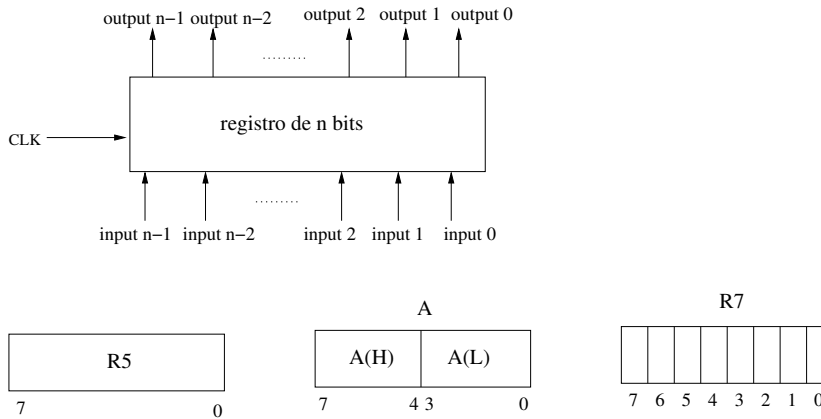


Figura 2.52: Registros y su representación.

Los **registros** son uno de los bloques de construcción fundamentales en sistemas de cómputo. Pueden contener datos, instrucciones, o bits de control. Una de las manipulaciones más comunes de datos en registros es la transferencia de registro, mediante la cual el contenido binario de un registro se copia a otro registro.

2.6.2. El registro de corrimiento

Un registro de corrimiento es una serie o arreglo de flip-flops cuya operación se ilustra en la Figura 2.53.

Primero se discute su operación y en seguida se considera su implementación. Supóngase inicialmente que el registro de corrimiento es capaz de almacenar 4 bits, y que cada uno de ellos inicialmente tiene un valor de 0. Si se introduce un bit con valor 0 (por la izquierda del registro), después de un pulso de reloj, el valor de todos los bits almacenados es 0 (Figura 2.53, paso 1). Ahora, supóngase que la siguiente entrada es un bit de valor 1. Después de un pulso de reloj, el registro almacena un valor binario 1000 (Figura 2.53, paso 2); es decir, el bit más significativo recibe el valor de 1, y los demás bits reciben a su vez un valor

0 procedente de la posición a su izquierda. Si se introduce en seguida otro bit con valor 1, en el siguiente pulso de reloj el contenido del registro tiene el valor binario 1100, es decir, cada bit cambia su valor, tomando el valor que recibe por su izquierda (Figura 2.53, paso 3). Esto da la impresión de que el valor binario corre hacia la derecha. De hecho, el contenido del registro se mueve a la derecha, un bit cada pulso de reloj, mientras que el bit que se introduce por la izquierda ocupa la posición del bit más significativo.

(1) entrada 0	→	0	0	0	0
(2) entrada 1	→	1	0	0	0
(3) entrada 1	→	1	1	0	0
(4) entrada 0	→	0	1	1	0
(5) entrada 1	→	1	0	1	1
(6) entrada 1	→	1	1	0	1
(7) entrada 0	→	0	1	1	0
(8) entrada 0	→	0	0	1	1
(9) entrada 1	→	1	0	0	1

Figura 2.53: La información binaria contenida en un registro de corrimiento durante una serie de pulsos de reloj sucesivos.

La operación del registro de corrimiento continúa conforme se mueven los bits a la derecha, y se introduce un bit a la izquierda. De hecho, la Figura 2.53 representa un corrimiento serial de 9 bits, que en su orden de entrada, son 011011001.

Con cada pulso de reloj la información binaria se corre un bit a la derecha. Cada vez que esto ocurre, el bit menos significativo se pierde. Se supone que tal información puede ser utilizada por otro dispositivo digital de la computadora.

Partiendo de la descripción anterior, a continuación se describe la construcción de un registro de corrimiento. La Figura 2.54 muestra dos implementaciones sencillas para el registro de corrimiento, una implementada con flip-flops JK y la otra con flip-flops D. Nótese que estas implementaciones solo permiten un corrimiento de la información binaria hacia la derecha.

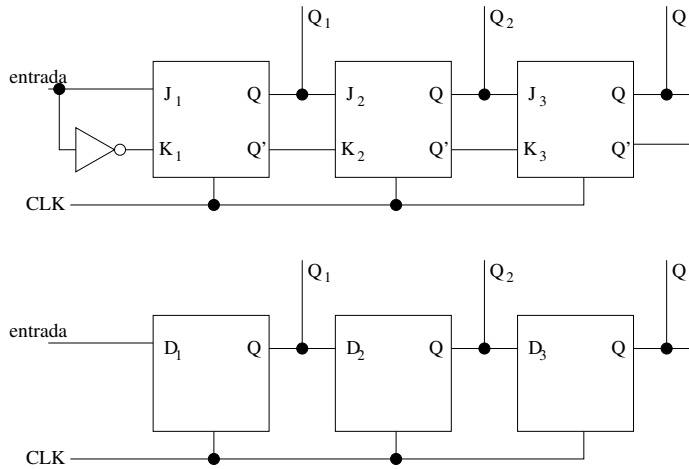


Figura 2.54: Dos registros de corrimiento de tres bits.

Considérese la implementación con flip-flops JK, suponiendo que el estado de inicio de todos los flip-flops es 0. Supóngase que la primera entrada es un 0. Entonces $J_1 = 0$ y $K_1 = 1$. Por lo tanto, el estado del flip-flop más a la izquierda permanece siendo $Q_1 = 0$ después del primer pulso de reloj. (Nótese que se supone el uso de flip-flops maestro-esclavo o de disparo por flanco.) Ya que $Q_1 = 0$ y $Q'_1 = 1$, entonces $J_2 = 0$ y $K_2 = 1$. Así, el estado del segundo flip-flop permanece en el estado $Q_2 = 0$, y de forma similar, el estado del tercer flip-flop permanece siendo $Q_3 = 0$.

Ahora bien, supóngase que la siguiente entrada al primer flip-flop JK es un 1. En tal caso, se tiene que $J_1 = 1$ y $K_1 = 0$. Entonces, el estado del primer flip-flop debe cambiar a $Q_1 = 1$; sin embargo, Q_1 no cambia de valor hasta después de que el pulso de reloj ha pasado. Durante el pulso de reloj $Q_1 = 0$, lo que implica que los otros dos flip-flops no cambian de estado.

Cuando ocurre el siguiente pulso de reloj, entonces el estado del primer flip-flop cambia a $Q_1 = 1$ y $Q'_1 = 0$, lo que fuerza a que el estado del segundo flip-flop cambie a $Q_2 = 1$; sin embargo, tal salida no ocurre hasta después del pulso de reloj (antes de esto $Q_2 = 0$). Por tanto, el estado del tercer flip-flop no cambia. Por supuesto, el estado del primer flip-flop durante este tiempo se establece a partir de la entrada.

Procediendo de esta manera, la información binaria viaja hacia la derecha con cada pulso de reloj, y las entradas sucesivas establecen el estado del flip-flop más a la izquierda.

Por otro lado, si Q_3 se toma como salida, entonces un solo bit sale del registro con cada pulso de reloj. De hecho, este es un ejemplo de salida serial. Pero si las tres terminales Q_1 , Q_2 y Q_3 se utilizan simultáneamente como salidas, se tiene una salida en paralelo. Por lo tanto, el registro de corrimiento basado en flip-flops JK de la Figura 2.54 puede utilizarse con entrada serial y para salida tanto serial como paralela (o hasta ambas). Si la entrada es serial y la salida paralela, se tiene un convertidor de serie a paralelo.

La operación del registro de corrimiento basado en flip-flops D de la Figura 2.54 es esencialmente igual, excepto por supuesto, en el tipo de flip-flops utilizados. Recuérdesse que el estado de un flip-flop D después del pulso de reloj corresponde al valor binario de su entrada. Si se desea incrementar el número de bits en un registro, solo se necesita añadir flip-flops a la derecha.

En una computadora real, los registros de corrimiento son frecuentemente más versátiles. A continuación, se discute el diseño de un registro de corrimiento que puede ser controlado para realizar varias funciones. Durante su operación normal, se proveen entradas por la izquierda y se van recorriendo hacia la derecha con cada pulso de reloj, un operación ordinaria que ya se ha descrito; sin embargo, se desea también que sea capaz de aceptar entradas por la derecha, mientras que el registro recorre sus bits a la izquierda. En este caso, funciona como un registro de corrimiento ordinario, excepto que funciona de derecha a izquierda. Además, en ocasiones no se desea que el registro recorra sus bits con cada pulso de reloj; lo que sí se desea es que el registro sea capaz de retener toda la información binaria almacenada en él, sin cambios. También es deseable que se pueda limpiar el registro, es decir, poner todos sus estados con valor de 0. Finalmente, se desea que los datos puedan ingresar en paralelo al registro.

Los modos de operación de los flip-flops del registro a diseñar se controlan mediante una serie de señales. Por simplicidad, supóngase que hay cuatro de tales señales: c_0 , c_1 , c_2 y c_3 . Si se desea un corrimiento ordinario a la derecha, los valores de las señales de control serán $c_0 = 1$, $c_1 = c_2 = c_3 = 0$. Si se desea un corrimiento a la izquierda, entonces $c_0 = 0$, $c_1 = 1$, $c_2 = c_3 = 0$. Para entrada paralela, $c_0 = c_1 = 0$, $c_2 = 1$, $c_3 = 0$. Si se desea limpiar el registro, entonces $c_0 = c_1 = c_2 = 0$, $c_3 = 1$. Si no se desea realizar ningún cambio en la información almacenada en el registro, entonces $c_0 = c_1 = c_2 = c_3 = 0$.

Un circuito para este registro controlado se muestra en la Figura 2.55. Básicamente, consiste en el conjunto de flip-flops D mostrado en la Figura 2.54. Se discute a continuación cómo las señales de control producen la operación deseada. Para simplificar la explicación, se redibuja el diagrama para una sola etapa, en la Figura 2.56.

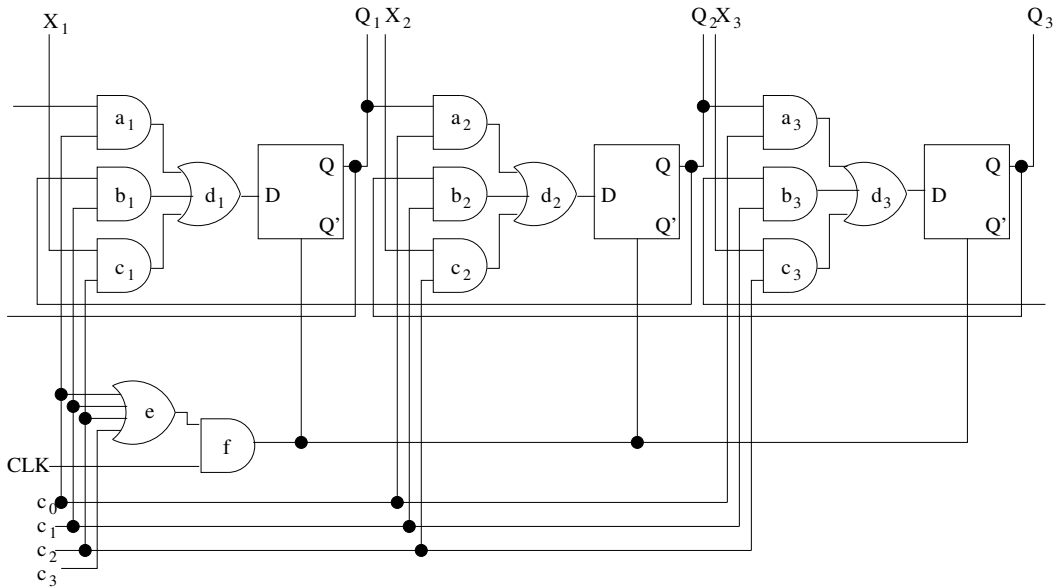


Figura 2.55: Un registro.

Obsérvese la Figura 2.56. Supóngase que todas las señales c_0 , c_1 , c_2 y c_3 tienen valor 0. Entonces, todas las entradas a la compuerta OR e tienen valor 0, por lo que una de las entradas a la compuerta AND f tiene valor 0. Nótese que la otra entrada tiene conectada la señal de reloj, y que la salida de f se conecta a las terminales de reloj de cada uno de los flip-flops D. De tal modo, para este conjunto de señales de control, ninguna señal de reloj alcanza a los flip-flops, y por tanto, el contenido del registro no cambia con los pulsos de reloj. El contenido del registro se preserva. Si cualquiera de las señales c_0 , c_1 , c_2 o c_3 tienen un valor de 1, entonces el pulso de reloj se aplica a las terminales correspondientes de los flip-flops, a fin de generar cambios en sus estados.

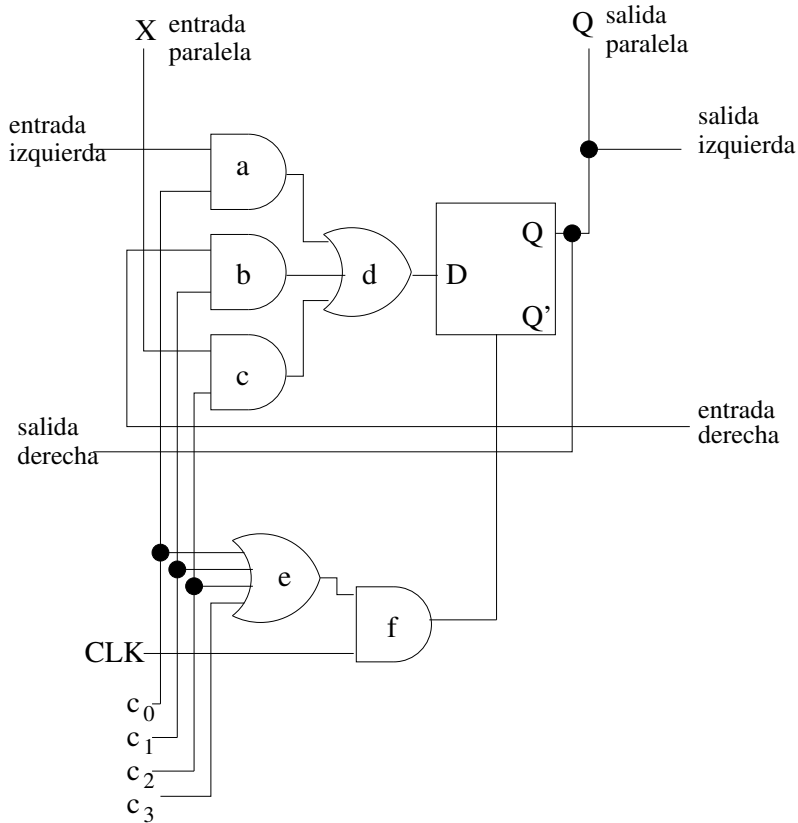


Figura 2.56: Una etapa del registro.

Ahora, supóngase que $c_0 = 1$ y que $c_1 = c_2 = c_3 = 0$. Entonces, la salida de la compuertas AND **b** y **c** tendrán valor de 0, mientras que la salida de la compuerta AND **a** tomará el valor de la entrada izquierda. Así, esta entrada se convierte en la entrada para el flip-flop D. Viendo el diagrama completo del registro en la Figura 2.55, para la condición $c_0 = 1$ y $c_1 = c_2 = c_3 = 0$, todo el circuito funciona como un registro de corrimiento hacia la derecha basado en flip-flops D que se muestra en la Figura 2.54, ya que cada flip-flop recibe su entrada de la etapa anterior inmediatamente a su izquierda.

Ahora, supóngase que las señales de control sean $c_0 = 0$, $c_1 = 1$ y $c_2 = c_3 = 0$. Entonces, en la Figura 2.56, una entrada de las compuertas AND **a** y **c** tiene valor 0, por lo que las salidas de tales compuertas toma el valor 0. Solamente la compuerta AND **b** tiene un valor de 1 a su entrada, por lo que su salida tomará el valor que haya en la entrada derecha. De tal modo, la entrada al flip-flop D se conecta efectivamente a la entrada derecha. Viendo el diagrama completo de la Figura 2.55, si $c_0 = 0$, $c_1 = 1$ y $c_2 = c_3 = 0$, entonces el circuito actúa como un registro de corrimiento basado en flip-flops D, sólo que ahora el bit de entrada proviene de la etapa anterior a la derecha, y el corrimiento se realiza hacia la izquierda.

Supóngase ahora que las señales de control sean $c_0 = c_1 = 0$ y $c_2 = 1$ y $c_3 = 0$. Usando el mismo razonamiento, nótese en la Figura 2.56 que la entrada al flip-flop se conecta ahora a la entrada paralela **X** mediante la compuerta AND **c**. En la Figura 2.55, puede verse que el circuito no funciona ahora como un registro de corrimiento, ya que cada flip-flop D se conecta efectivamente a su propia entrada paralela **X**, y después de que el pulso de reloj ocurre, almacena tal entrada como su estado. De este modo, se tiene un registro con carga en paralelo.

Finalmente, considérese la secuencia de control $c_0 = c_1 = c_2 = 0$ y $c_3 = 1$. Esto implica que una de las dos entradas a cada compuerta AND tiene valor de cero (Figura 2.56). Por lo tanto, la entrada a cada flip-flop D tiene valor de 0. Ya que $c_3 = 1$, en cualquier momento que el pulso de reloj ocurra, la señal de reloj se aplicará en las entradas de reloj de cada flip-flop D. Al aplicarse la señal y teniendo una entrada $D = 0$, el estado siguiente de cada flip-flop será 0. Obsérvese la Figura 2.55. Si $c_3 = 1$ y $c_0 = c_1 = c_2 = 0$, entonces al siguiente pulso de reloj el estado de todos los flip-flops D tendrá el valor de 0. El registro se ha limpiado, como se deseaba en un principio.

Habiendo discutido todos los valores permisibles de las señales de control y mostrado que el registro tiene un funcionamiento como se planteó en un principio, se discuten a continuación las señales de salida del registro. Considérese la salida etiquetada “salida derecha” en la Figura 2.56. Se trata de la salida del flip-flop más a la derecha del registro en la Figura 2.55. Si se recorre hacia la derecha el contenido del registro, y se observan los valores de esta salida, el registro funciona como un registro de corrimiento simple.

En forma similar, si se desea que el registro funcione como un registro de corrimiento pero ahora a la izquierda, entonces la salida a observar es la salida

del flip-flop más a la izquierda del registro. Nótese que en la Figura 2.56, la salida etiquetada “salida izquierda” se conecta a la salida de tal flip-flop en la Figura 2.55.

En la Figura 2.55, las salidas de todos los flip-flops se conectan a terminales etiquetadas Q_1 , Q_2 y Q_3 . De este modo, una salida paralela del registro puede obtenerse en cualquier momento. Como se ha discutido antes, también hay entradas paralelas X_1 , X_2 y X_3 , por lo que este registro puede funcionar sencillamente como un registro de carga y descarga en paralelo. Nótese que las entradas pueden cargarse en paralelo, y mediante manipular las señales de control, proveer de una salida serial. Por tanto, este registro puede utilizarse como un conversor paralelo a serial. De igual forma, puede utilizarse como conversor serial a paralelo.

2.6.3. Contadores

Un circuito digital de gran utilidad se conoce con el nombre de contador. Tal circuito se utiliza para contar el número de pulsos que se le aplican. Los contadores se utilizan en las computadoras para poder seguir el número de operaciones que se han realizado. También se les utiliza en muchas aplicaciones no computacionales. Por ejemplo, supóngase que toda persona que entra al metro pasa por un torniquete. Cada vez que alguien entra, el torniquete produce un pulso mediante un interruptor. El contador se utiliza entonces para determinar el número de personas que han pasado por el torniquete.

Un contador real produce un número binario equivalente a su cuenta. A continuación, se discute un circuito digital capaz de contar hasta 8 pulsos de reloj. Después de llegar a la cuenta de 7, el siguiente evento a contar produce que este contador reinicie con un valor de 0, y comience de nuevo la cuenta. Es por ello que tal contador se conoce con el nombre de contador módulo 8. Esta forma de operación es común en muchos contadores. Por ejemplo, el odómetro de un automóvil es un tipo de contador, que va de 00000 a 99999 kilómetros. Después de que se ha alcanzado tal límite, el odómetro reinicia la cuenta en 00000.

Un circuito para un contador módulo 8 utilizando flip-flops JK se muestra en la Figura 2.57.

Supóngase que el estado de todos los flip-flops inicia en 0. Nótese que los pulsos de entrada se aplican en las terminales de reloj de todos los flip-flops, así como a las entradas J_0 y K_0 . Cuando se aplica el primer pulso, $J_0 = K_0 = 1$ durante el pulso. Por lo tanto, Q_0 toma el valor de 1 después del pulso. Se supone de nuevo

aquí el uso de flip-flops maestro-esclavo o disparados por flanco. Por tanto, Q_0 no pasa de 0 a 1 hasta que el pulso ha pasado. Así, durante el pulso, J_1 , K_1 , J_2 y K_2 permanecen con valor 0, por lo que Q_1 y Q_2 también permanecen en 0. De este modo, la salida después del primer pulso de entrada es $Q_2 = 0$, $Q_1 = 0$ y $Q_0 = 1$, o $001_2 = 1_{10}$, como se supone debe ser. Esto indica que se ha contado un pulso.

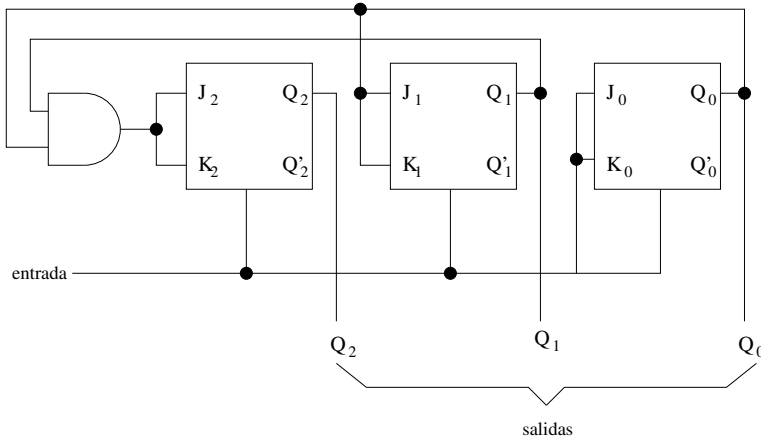


Figura 2.57: Contador módulo 8 usando flip-flops JK.

Cuando un segundo pulso se aplica en la entrada y $Q_0 = 1$, las entradas al segundo flip-flop toman los valores $J_1 = K_1 = 1$. Por tanto, Q_1 cambia de estado hasta que el pulso haya pasado. Además, durante ese tiempo, Q_0 cambia de 1 a 0 ya que $J_0 = K_0 = 1$ durante tal pulso. Nótese que durante el segundo pulso, $J_2 = K_2 = 0$, ya que la entrada de Q_1 a la compuerta AND es 0 hasta después que el pulso haya pasado. De modo que después del segundo pulso, se tiene que $Q_2 = 0$, $Q_1 = 1$ y $Q_0 = 0$, o $010_2 = 2_{10}$.

Cuando un tercer pulso se aplica a la entrada, Q_0 cambia de nuevo su estado de 0 a 1 después del pulso; sin embargo, Q_1 no cambia, ya que durante el pulso, $Q_0 = 0$. También, la entrada conectada a Q_0 a la compuerta AND durante el pulso tiene valor 0, por lo que $J_2 = K_2 = 0$, lo que implica que $Q_2 = 0$. Así, después del tercer pulso, se tiene que $Q_2 = 0$, $Q_1 = 1$ y $Q_0 = 1$, o $011_2 = 3_{10}$.

Ahora, supóngase que llega un cuarto pulso a la entrada. Durante el tiempo que ese pulso se encuentra presente, $Q_0 = 1$ y $Q_1 = 1$, por lo que $J_2 = K_2 = 1$, y

entonces $Q_2 = 1$. Ya que $Q_0 = 1$ durante el pulso, entonces $J_1 = K_1 = 1$ durante el pulso. Por lo tanto, el segundo flip-flop cambia su estado después del pulso. Así, después del pulso $Q_1 = 0$. Por tanto, al final se tiene que $Q_2 = 1$, $Q_1 = 0$ y $Q_0 = 0$, o $100_2 = 4_{10}$.

Continuando así, se puede detallar que el contador llegará al valor de $111_2 = 7_{10}$, para recomenzar con el valor de $000_2 = 0_{10}$ después. Nótese que los contadores de este tipo pueden ensamblarse de modo que pueden contar hasta cualquier potencia de 2; sin embargo, también es posible diseñar contadores que cuenten hasta el valor que se desee.

2.6.4. Detectores de secuencia y generadores de secuencia

En esta sección se describen algunos circuitos conocidos como detectores de secuencias. Estos circuitos dan como salida 1 si se les introduce una secuencia específica de ceros y unos. De otra manera, su salida es 0.

Los detectores de secuencia no son únicamente utilizados en computadoras digitales. Pueden usarse, por ejemplo, en cerrojos de combinación digital. Supóngase que se cuenta con un control de radio para abrir una puerta. Debe abrir solamente a una señal de radio, y a ninguna otra. El radiotransmisor, usando un detector de secuencia, transmite la secuencia apropiada.

Considérese un circuito detector de secuencia que da como salida 1 si la secuencia 010 se le introduce (Figura 2.58). Ya que se trata de flip-flops maestro-esclavo, sólo responden a señales aplicadas durante el pulso de reloj, y solo cambian sus estados después del pulso de reloj. Se supone que una señal de entrada se aplica al detector de secuencia durante cada pulso de reloj. Si el nivel de la señal de entrada durante el pulso tiene valor 0, entonces se introduce un 0. De forma similar, si el nivel de la señal de entrada tiene valor 1 durante el pulso de reloj, se introduce un 1.

Ahora supóngase que un 0 se ha introducido. Entonces, al menos una entrada de la compuerta AND **b** debe tener valor 0. Por lo tanto, D_1 tiene valor 0, y por lo tanto, después del pulso de reloj, $Q_1 = 0$, $Q'_1 = 1$. Ya que la entrada fue 0, al menos una entrada a la compuerta AND **c** es también 0, por lo que $D_2 = 0$, y en el siguiente pulso de reloj $Q_2 = 0$. Ya que una entrada a la compuerta AND **d** tiene valor 0, entonces su salida es 0.

Supóngase que en el siguiente pulso de reloj, la entrada tiene valor 1. Ahora, las tres entradas de la compuerta AND **c** tienen valor 1, y por lo tanto, D_2 toma

el valor de 1. Para el siguiente pulso de reloj, $Q_2 = 1$ y $Q'_2 = 0$. Así, una entrada a la compuerta AND **d** tiene valor 1; sin embargo, la otra entrada tiene valor 0, ya que es el complemento de la entrada. Por tanto, la salida continúa siendo 0.

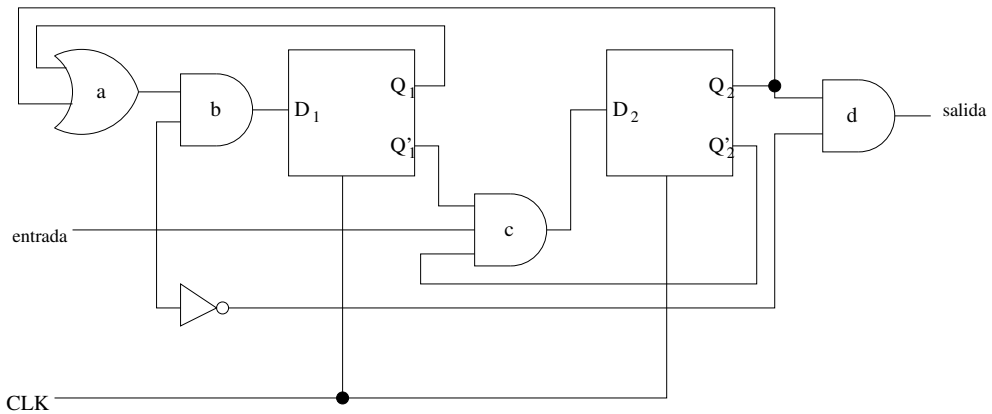


Figura 2.58: Detector de la secuencia 010.

Supóngase ahora que durante el siguiente pulso de reloj, se introduce un 0. Q_2 no puede cambiar hasta que el pulso haya pasado. Entonces, durante el pulso de reloj, la entrada inferior de la compuerta AND **d** tiene valor 1, ya que es el complemento de la entrada. Por tanto, la salida toma el valor de 1. Si se procede de esta forma, con todas las posibles secuencias, se puede demostrar que sólo la secuencia 010 provoca una salida 1, y ninguna otra. De tal modo, este circuito funciona efectivamente como un detector de secuencia.

Ocasionalmente, se requiere generar secuencias de pulsos. Esto puede lograrse mediante un dispositivo digital conocido como generador de secuencias. Por ejemplo, tal dispositivo puede usarse en un radiotransmisor para abrir una puerta. Los generadores de secuencias consisten de interconexiones de flip-flops y compuertas. El contador de la Figura 2.57 es un generador de secuencias. Si Q_0 se toma como salida, entonces se produce la secuencia 01010... Si Q_1 es la salida, la secuencia es 00110011... Si Q_2 es la salida, la secuencia es 000011110000... Ahora bien, si se utiliza una interconexión mediante compuertas de estas tres señales, se puede obtener la secuencia deseada de cero y unos.

2.6.5. Utilizando registros en lógica secuencial

Regresando al ejemplo del contador, se ha descrito hasta ahora el diagrama de estados y la tabla de estados. Un circuito secuencial en general tiene la forma que se muestra en la Figura 2.59.

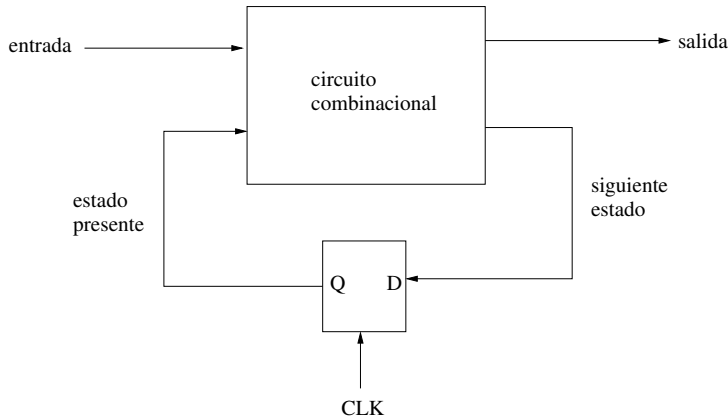


Figura 2.59: Un circuito secuencial general.

Utilizando dos registros D para almacenar el estado de la máquina, se obtiene el siguiente estado como una función del estado actual y las entradas al circuito. Para trabajar con los dos flip-flops D que almacenan el estado, se incrementa la tabla de estados. La entrada CLK (el reloj) se considera implícita. Las entradas a los flip-flops (DA y DB, respectivamente) se llenan considerando las entradas cuyo valor se requiere para que la transición necesaria se lleve a cabo. Por ejemplo, en este caso, para que el circuito proceda del estado $AB = 00$ al estado $A+B+ = 01$, $DA = 0 = A+$ y $DB = 1 = B+$.

A	B	A+	B+	DA	DB
0	0	0	1	0	1
0	1	1	0	1	0
1	0	1	1	1	1
1	1	0	0	0	0

Las expresiones para el siguiente estado ($A+B+ = DADB$) pueden obtenerse usando procedimientos como las siguientes suma de productos:

$$DA = A'B + AB' = A \oplus B$$

$$DB = A'B' + AB' = B'$$

La implementación del contador resulta en un circuito secuencial sencillo (Figura 2.60).

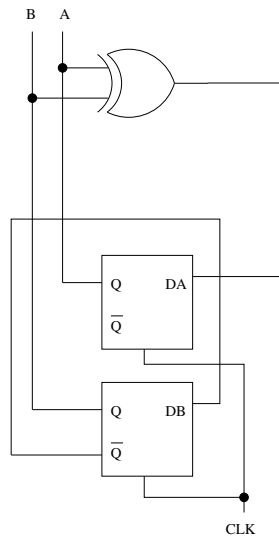


Figura 2.60: Circuito secuencial para el ejemplo del contador.

2.7. Cómputo digital básico

En una computadora digital, las operaciones aritméticas básicas se realizan en un componente digital llamado Unidad Lógico-Aritmética (*Arithmetic Logic Unit*, o simplemente ALU). Esta se relaciona con uno o más registros llamados acumuladores, donde se almacenan los resultados de las operaciones. Por simplicidad, se considera por ahora que la ALU se relaciona con un solo acumulador.

Una operación aritmética típica consiste en lo siguiente: un valor binario se lee de memoria y se da como entrada a la ALU, en donde se puede sumar, restar,

multiplicar o dividir (dependiendo de la operación aritmética que se le instruya a la ALU) con el valor binario que se encuentre en el acumulador. El resultado de esta operación se almacena de nuevo en el acumulador. En el caso de una operación lógica, se realiza exactamente lo mismo, excepto que la operación debe ser lógica (AND, OR, NOT, XOR, etc.).

En la sección 2.3.5 se considera un sumador completo, cuyo circuito lógico se presenta en la Figura 2.39. En este capítulo se considera al sumador completo como parte de la ALU. A continuación, la Figura 2.61 representa al sumador completo como un diagrama de bloque.

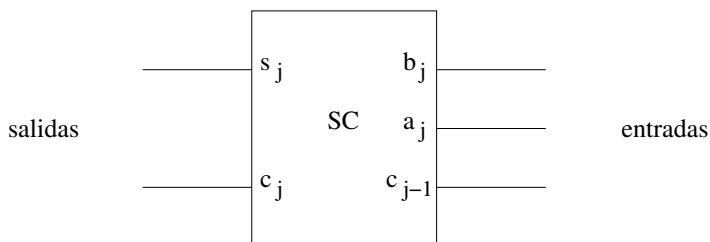


Figura 2.61: Diagrama de bloque de un sumador completo.

Este sumador completo (SC) suma tres números binarios de un solo bit b_j , a_j y c_{j-1} , para obtener su suma s_j y el acarreo a una siguiente etapa c_j . Recuérdese que b_j y a_j representan los j -ésimos bits en un número binario, y que c_{j-1} es el acarreo proveniente de la columna anterior $j - 1$. Para ilustrar esto, la Figura 2.62 presenta un sumador de tres bits, que permite sumar dos números binarios de tres bits ($a_2a_1a_0$ y $b_2b_1b_0$, respectivamente) utilizando sumadores para obtener un resultado también de tres bits ($s_2s_1s_0$).

Nótese que c_2 , el acarreo de la columna más a la derecha, no se utiliza. Si la suma resulta en un número de cuatro bits, entonces se genera un *overflow*. Esto se discute en la sección 2.3. El acarreo c_2 puede utilizarse entonces como indicador de que un overflow ha sucedido (particularmente si $c_2 = 1$). Esto puede generar, por ejemplo, que un mensaje de advertencia aparezca en la pantalla.

Para formar una ALU muy simple, se combina a continuación el sumador con un registro de entrada y salida paralela. Tal registro se discute en la sección 2.6 (nótese que el circuito en la Figura 2.55 funciona como un registro de entrada y salida paralela si las señales de control son $c_0 = c_1 = 0$, $c_2 = 1$, $c_3 = 0$). En la

ALU de la Figura 2.63 se puede utilizar un registro más complejo, pero esto se evita por razones de simplicidad.

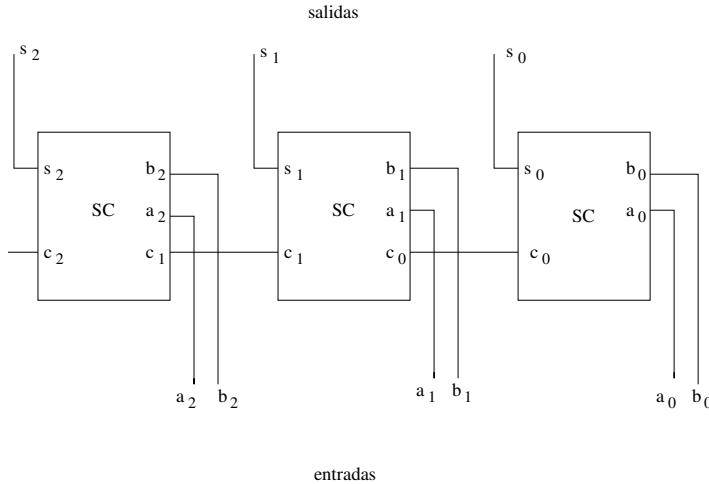


Figura 2.62: Un sumador de tres bits.

En la operación del circuito de la Figura 2.63 pueden verse dos líneas de control: *ADD* y *CLR*. Si se desea realizar una suma, debe colocarse $ADD = 1$ y $CLR = 0$. Supóngase que en tal caso, un número binario de tres bits se encuentra almacenado en el registro acumulador. La salida de cada flip-flop del registro se conecta con la entrada de su correspondiente sumador completo, de tal modo que el número almacenado en el acumulador es uno de los operandos a ser sumados. El otro número binario debe aplicarse en las entradas $b_2b_1b_0$. Después de realizar la adición, el resultado $s_2s_1s_0$ aparece en las salidas $Q_2Q_1Q_0$.

Ahora bien, el resultado debe aparecer almacenado en el acumulador. Ya que $CLR = 0$, entonces una entrada de cada compuerta AND e_2, e_1 y e_0 debe tener valor 1, de modo que $D_2 = s_2, D_1 = s_1$ y $D_0 = s_0$. Por lo tanto, después del siguiente pulso de reloj, el número que se almacena en el acumulador es el resultado de la suma. Nótese que se hace uso de flip-flops maestro-esclavo o de disparo por flanco. Así, la salida no cambia hasta después del pulso de reloj. Por tanto, si un número se encuentra almacenado en el acumulador, tal número permanece ahí hasta el siguiente pulso de reloj. Supóngase que el número a sumarse aparece

a la salida de los flip-flops después del pulso de reloj que (arbitrariamente) puede nombrarse pulso de reloj 1. Durante el tiempo en que el pulso de reloj está apagado (entre los pulsos de reloj 1 y 2) la suma se realiza. Entonces, al siguiente pulso de reloj (pulso de reloj 2) la entrada a los flip-flops es el resultado de la suma, pero la entrada a los flip-flops no ha cambiado. Así, las entradas a_2 , a_1 y a_0 al sumador no cambian hasta el pulso de reloj 2. Los estados internos de los flip-flops han cambiado ya al valor deseado. Por tanto, solo hasta después del pulso de reloj 2, la salida de los flip-flops toma el valor del resultado de la suma. Nótese que toda esta operación supone que los sumadores son suficientemente rápidos, o que los pulsos de reloj se encuentran lo suficientemente apartados entre sí, de modo que la suma puede realizarse entre los dos pulsos de reloj.

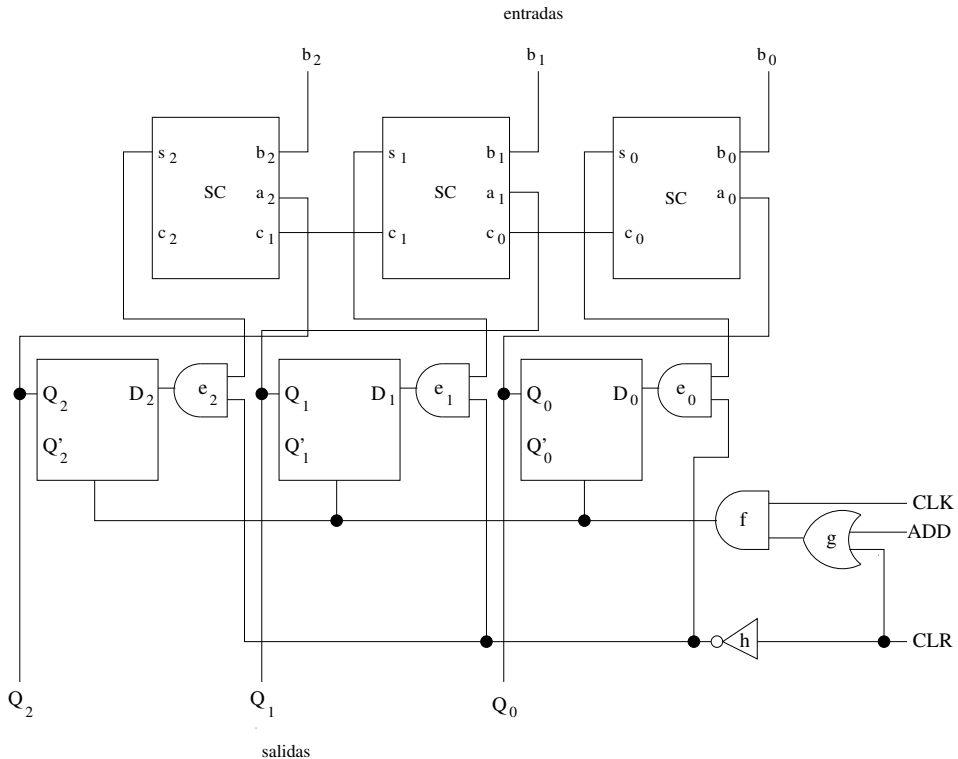


Figura 2.63: Una ALU simple de tres bits.

Por otro lado, si ambas líneas de control CLR y ADD tienen valor 0, entonces ningún pulso de reloj o entrada se aplican a los flip-flops, por lo que el número almacenado en el acumulador permanece ahí. Si $CLR = 1$ y $ADD = 0$, entonces todas las entradas a los flip-flops toman valor de 0, ya que una de las entradas a las compuertas AND e_2 , e_1 y e_0 tiene valor 0. Por lo tanto, se limpia el acumulador; sin embargo, debe hacerse notar un hecho. El número de entrada $b_2b_1b_0$ debe almacenarse en un registro. Este registro podría ser parte de la ALU o ser externo a ella. Si es interno, entonces la ALU tiene asociados dos registros, que se conocen como acumuladores.

2.7.1. Aritmética modular

La suma hecha con una ALU difiere de la suma que se realiza a mano. Ya que el registro acumulador puede almacenar tan solo un número fijo de bits, puede darse una condición de *overflow* (sobrecapacidad) en el resultado. Podría parecer que esta propiedad de la ALU es totalmente desventajosa, pero en esta sección y la siguiente se pretende mostrar cómo puede hacerse uso de ella.

Primero, es necesario familiarizarse con algunos detalles del almacenamiento de números, mediante un acumulador que trabaje en base 10. Un odómetro simple de automóvil puede utilizarse como ejemplo útil. Supóngase que se tiene un odómetro que llega hasta 99, y regresa a 0 (se utiliza el 99 en lugar de un número más grande como 99999 para contar con números pequeños en el ejemplo). Por tanto, las lecturas en el odómetro pueden ser:

00
01
02
03
⋮
10
11
12
⋮

96
97
98
99
00
01
02
⋮

El acumulador base 10 del odómetro puede almacenar números del 0 al 99, pero si se sobrepasa su capacidad con números adicionales, entonces vuelve a empezar de nuevo. De hecho, los acumuladores de las computadoras funcionan también así.

Ahora bien, supóngase que el odómetro muestra un valor de 26. No podemos estar seguros si el automóvil ha sido utilizado por 26 kilómetros o alguna de las siguientes distancias:

26
126
226
⋮

(Podría también tratarse de números negativos, pero este caso se considera en la siguiente sección).

Si se utiliza un registro que puede almacenar números hasta el 99, entonces los números 26, 126, 226, ... producen lecturas similares. De forma parecida, 47, 147, 247, 347, ... serían similares. Por esto, se introduce un símbolo que toma esto en cuenta. Considérese el símbolo \equiv_{100} . Para el primer ejemplo, se puede afirmar que:

$$26 \equiv_{100} 126 \equiv_{100} 226$$

El símbolo \equiv_{100} significa entonces que esta equivalencia se da para un registro que almacena 100 números (0, 1, 2, ..., 99). Esto se lee como equivalente módulo 100 o congruente, módulo 100. Esto es, que 26 y 126 son equivalentes módulo 100. Nótese que el significado es que cada uno de estos resultados produce la misma salida del registro.

Si se desea encontrar los números que son equivalentes módulo 100 para un número dado, sólo se requiere sumar 100 y los múltiplos de 100 a tal número. Por ejemplo, para 26, sus equivalentes módulo 100 son 126, 226, 326, etc.

De hecho, no hay esencialmente ninguna diferencia si se utiliza un registro binario. Por ejemplo, utilizando un acumulador de tres bits, se tiene que los valores representables son:

000
001
010
011
100
101
110
111
000
001
010
011
100
⋮

Como ejemplo, supóngase que el registro tiene una lectura de 011_2 . No se puede estar seguro cuántas vueltas se han dado sobre el mismo registro, de modo que 011_2 podría representar 3_{10} , 11_{10} o 19_{10} . Utilizando la notación anterior, se puede afirmar que:

$$011 \equiv_8 1011 \equiv_8 10011$$

De forma similar, se puede escoger otro ejemplo:

$$100 \equiv_8 1100 \equiv_8 10100 \equiv_8 11100$$

Considerando esto de una forma diferente, supóngase que si se suma 1000_2 al registro de tres bits, se va a través de 8 pasos en un ciclo y se regresa al punto de inicio. De forma similar, para un acumulador de 4 bits, si se suma 16 al número binario almacenado, no hay cambio en el número almacenado. Aun cuando esto parece no llevar a ningún resultado práctico, en la siguiente sección se discute cómo estas ideas pueden utilizarse para simplificar los circuitos digitales de una computadora.

Antes de considerar mayores detalles de la aritmética modular, se discute la sustracción binaria. Supóngase que se desea realizar la resta:

$$\begin{array}{r} 1011 \leftarrow \text{minuendo} \\ -0110 \leftarrow \text{sustraendo} \\ \hline 0101 \leftarrow \text{diferencia} \end{array}$$

Normalmente, la sustracción se realiza columna por columna, comenzando por la derecha; sin embargo, el bit de la tercera columna del sustraendo es mayor que el bit de la tercera columna del minuendo. La solución a este problema es pedir prestado una unidad de bit en la cuarta columna del minuendo. De este modo, la sustracción se puede escribir como:

$$\begin{array}{r} 01011 \\ -0110 \\ \hline 0101 \end{array}$$

Ahora bien, considérese la siguiente sustracción:

$$\begin{array}{r} 0110 \\ -1011 \\ \hline -0101 \end{array}$$

Ya que el sustraendo es mayor que el minuendo, la diferencia es negativa. La manera como se consideran números binarios negativos en la aritmética digital se explica en la siguiente sección.

2.7.2. Aritmética de complemento a 2

En la sección anterior se discute un circuito para una ALU que puede utilizarse para realizar adiciones. Se utilizan sumadores completos para ir sumando cada uno de los bits; sin embargo, si se desea realizar una resta, es siempre posible desarrollar un circuito lógico que la lleve a cabo. De manera análoga al sumador completo, se puede desarrollar un sustractor completo. Esto no se hace en la mayoría de los casos, ya que representaría usar circuitos adicionales y hacer más complejo el *hardware*, lo que incrementaría el costo y tamaño de la computadora. Por otro lado, también incrementaría su rapidez, lo que sería una ventaja.

En esta sección se presenta cómo tomar en cuenta las propiedades modulares de la ALU de modo que se pueda realizar una sustracción utilizando el sumador modular. Así, no es necesario añadir *hardware* adicional para realizar restas, sino más bien se utiliza un procedimiento para trabajar los números negativos convenientemente.

Para entender cómo hacer una sustracción sin un circuito sustractor, supóngase por ejemplo que se cuenta con un registro de 4 bits, y que se desea restar 0100_2 (4_{10}) de 1001_2 (9_{10}). Escribiendo la resta primero en base 10, más adelante se repite en base 2. Por tanto, se tiene que:

$$9 - 4 = 5$$

Ahora bien, supóngase que se suma 16_{10} al número almacenado en el registro. Como se discute en la sección anterior, el valor en el registro da un ciclo completo, por lo que no hay cambio en el número almacenado. Esto se puede escribir como:

$$16 + 9 - 4 = 5 + 16 \equiv_{16} 5$$

En seguida, se reescribe la parte izquierda de la ecuación como:

$$15 + 1 + 9 - 4 = (15 - 4) + 1 + 9$$

Realizando la operación, se hace primero la resta de 15 menos 4 y luego se suma 1, y luego 9. Recuerde que se intenta evitar construir un sustractor, y por lo pronto, parece que no se ha logrado nada ya que todavía es necesario restar 4 de 15; sin embargo, cuando esto se hace en forma binaria, la operación resulta muy simple. 15 es el número más grande que se puede almacenar en el registro de 4 bits:

$$15_{10} = 1111_2$$

Nótese que este número es solo una lista de unos. De modo que considerando la resta de este número, en el ejemplo $15 - 4$ es:

$$\begin{array}{r} 1111 \\ - 0100 \\ \hline 1011 \end{array}$$

El resultado de esta resta puede obtenerse directamente del sustraendo, mediante complementar cada uno de sus bits. De hecho, si se resta cualquier valor binario de 1111_2 , la diferencia es exactamente el complemento de cada bit del sustraendo. A esto se le conoce como complemento a 1's, y es una operación que no requiere hardware adicional. En general, todos los números binarios a procesar se almacenan en registros compuestos de flip-flops. Por ejemplo, supóngase que un número almacenado en el acumulador de la Figura 2.63. Si se desea complementar sus bits, se requiere solamente tomar las salidas negadas de cada flip-flop. Este número se encuentra siempre disponible, de modo que no se requiere hardware adicional.

A continuación, se realiza la sustracción 4 de 9 en binario. Usando este procedimiento, se tiene que $4_{10} = 0100_2$. Obteniendo el complemento a 1's, éste es 1011_2 . El complemento a 2's se obtiene sumando 1 al complemento a 1's, lo que nos da 1100_2 . Finalmente, para obtener la diferencia, se debe sumar este valor a $9_{10} = 1001_2$. Por lo tanto:

$$\begin{array}{r} 1100 \\ + 1001 \\ \hline 10101 \end{array}$$

Nótese el bit más a la izquierda del resultado. Este bit no aparece en un registro de 4 bits. El registro solo contiene:

$$0101_2 = 5_{10}$$

Lo cual es la diferencia correcta. Si no se trabajara con registros, al sumar $15 + 1 = 16$, lo que sería una respuesta errada; sin embargo, sumar 16 en un registro de 4 bits solo hace regresar al valor original de donde se comenzó, ya que el bit más a la izquierda se pierde. Así, se usa una propiedad de la aritmética modular como ventaja.

En términos generales, supóngase que se tiene un registro en el cual el número más grande que se puede almacenar es $R - 1$. Por ejemplo, en el caso del registro de 4 bits, se tiene que $R - 1 = 15$. Utilizando este sistema, supóngase ahora que se desea realizar la sustracción:

$$D = A - B$$

Lo que realmente se realiza es:

$$D = (R - 1) - B + 1 + A$$

La operación $(R - 1) - B$ es fácil de obtener, ya que sólo implica complementar cada bit de B . Las sumas siguientes producen el resultado correcto. De tal modo, solo se requieren sumadores y no sustractores para realizar la sustracción.

2.7.3. Complemento a 2

El nombre de la operación que se discute anteriormente es complemento a 2 de un número N . Se define de la siguiente forma:

$$C_2 = 2^k - N$$

donde k es el número de bits en el registro. Por ejemplo, si se tiene un registro de 4 bits, se tiene que $2^4 = 16_{10} = 10000_2$, y el complemento a 2 de $N = 0100_2$ es:

$$C_2 = 10000_2 - 100_2 = 1100_2$$

Nótese que si se toma $(R - 1) - B + 1$ de la expresión para obtener la diferencia, en realidad lo que se hace es obtener el complemento a 2 de B . De la discusión previa, es fácil obtener el complemento a 2: se requiere sólo reemplazar cada bit por su complemento, y al resultado sumarle 1. Por lo tanto, si se trabaja con 4 bits, el complemento a 2 de 0100_2 es 1100_2 .

Ahora bien, se sabe que el número almacenado en un registro es la representación binaria de un número decimal; sin embargo, podría verse esto de manera diferente. La secuencia de bits, que se conoce como número binario, puede considerarse como un código que representa un número decimal. Es posible arreglar el código de modo que el número binario no fuera igual a su equivalente decimal. Más aun, es posible construir circuitos de compuertas lógicas que operen sobre

estos números, y den el resultado correcto cuando se les convierta de nuevo en su forma decimal. Es notorio que el desarrollo de un código como el que se menciona sería una complicación innecesaria, y que muy probablemente los circuitos lógicos resultantes serían muy complicados. Por lo tanto, esto no se hace para los números positivos; sin embargo, cuando es necesario trabajar con números negativos, es verdaderamente necesario usar algún tipo de código, ya que los registros almacenan ceros o unos, pero no símbolos matemáticos como el punto o el signo menos.

De los muchos códigos que pueden emplearse, se discute a continuación uno que es ampliamente usado y que hace el cómputo simple. Logra esto porque los resultados son correctos en términos de la aritmética modular. De hecho, el complemento a 2 puede usarse para obtener tal codificación. Supóngase que se tiene un registro de 4 bits. Los números que almacena son:

⋮	
1011	
1100	
1101	
1110	
1111	$\leftarrow 2_{10} - 3_{10} = -1_{10}$
0000	
0001	
0010	$\leftarrow 2_{10}$
0011	
0100	
0101	$\leftarrow 2_{10} + 3_{10} = 5_{10}$
0110	
0111	
1000	
⋮	

Supóngase que se suma 3_{10} a un número. Esto significa ir bajando en la numeración hasta llegar a tres posiciones más abajo. Por ejemplo, considérese el número 2_{10} . De su posición, al sumarle 3_{10} , se obtiene la posición 5_{10} . Ahora bien, supóngase que se resta 3_{10} . Esto equivale a moverse hacia arriba en la numeración tres posiciones. El resultado es $1111_2 = 15_{10}$; sin embargo se sabe que:

$$15_{10} \equiv_{16} -1_{10}$$

Esto es, que 15_{10} es equivalente módulo 16 a -1_{10} .

Si se define que 1111_2 represente a -1_{10} en un código, entonces se llega a la respuesta correcta para la resta. De forma similar, 1110_2 representa al -2_{10} . De este modo, se tiene una codificación para el registro de 4 bits que permite representar números de -7_{10} a 7_{10} :

Número decimal	Número binario
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

La información de esta tabla es consistente con la discusión anterior, pero nótese que los números negativos son simplemente el complemento a 2 de sus equivalentes positivos. Por ejemplo, para obtener la representación de -7_{10} , se toma el número $7_{10} = 0111_2$ y se obtiene su complemento:

$$1000_2 + 0001_2 = 1001_2 = -7_{10}$$

De forma similar, es posible obtener todos los otros números negativos en la tabla. Es notorio que si el bit más significativo es 0, se trata de un número positivo, mientras que si es 1, es negativo. Nótese que un número no se hace negativo simplemente por reemplazar su bit más significativo de 0 a 1.

Si se tienen b bits en un registro, se utilizan solamente $b - 1$ de ellos para representar la magnitud de un número. Por ejemplo, en la tabla anterior se tienen 4 bits, pero los números que se representan van de -7_{10} a 7_{10} . Si se trabaja solo con números positivos, entonces el rango de valores sería de 0_{10} a 15_{10} .

2.7.4. Uso del complemento a 2 para la sustracción

Utilizando un sistema de 4 bits y trabajando con la sustracción, se presenta a continuación que el uso del complemento a 2 para representar números negativos realmente funciona. Supóngase que se desea restar 4_{10} de 7_{10} . Esto resulta equivalente a sumar $7_{10} + (-4_{10})$. Considerando sus equivalentes binarios, se tiene que el complemento a 2 de $4_{10} = 0100_2$ es 1100_2 . Sumando, se tiene que:

$$0111_2 + 1100_2 = (1)0011_2$$

Ignorando el bit más significativo entre paréntesis, se tiene que $0011_2 = 3_{10}$, que es la diferencia correcta al operar $7_{10} - 4_{10}$. De nuevo, nótese que se utilizan las propiedades de la aritmética modular para obtener el resultado correcto.

Considérese otro ejemplo: restar 6_{10} de 3_{10} . Entonces, se opera con el complemento a 2 de 0110_2 , que es 1010_2 , de modo que la resta queda como:

$$0011_2 + 1010_2 = 1101_2$$

De la tabla anterior se observa que $1101_2 = -3_{10}$, el resultado correcto.

Para este punto, es necesario aclarar una cosa: en algunos de los ejemplos anteriores se ha ignorado el bit de sobrecapacidad (*overflow*) mediante ignorar o eliminar el bit más a la izquierda del resultado. En realidad, este no es un ejemplo de verdadera sobrecapacidad. La única ocasión en que realmente sucede una sobrecapacidad es cuando el tamaño del resultado de una operación es demasiado grande para representarse con los bits disponibles en el registro. Por ejemplo, para el registro de 4 bits, si se realiza la suma:

$$7_{10} + 5_{10} = 0111_2 + 0101_2 = 1100_2$$

El resultado, de acuerdo con la tabla, es $1100_2 = -4_{10}$. El resultado debería ser 12_{10} . Por tanto, una verdadera sobrecapacidad ha ocurrido, y se obtiene un resultado incorrecto. Para este sistema de registro de 4 bits, una sobrecapacidad verdadera ocurre cuando el resultado es mayor que 7_{10} . De manera similar, en un registro de k bits, una verdadera sobrecapacidad ocurre cuando la magnitud de un resultado excede $2^{k-1} - 1$.

2.7.5. Multiplicación y división

Esta sección discute la multiplicación y división como se realizan por una computadora digital. Se inicia mediante describir qué se hace normalmente cuando se multiplican dos números binarios manualmente. De hecho, las reglas de la multiplicación binaria siguen exactamente las mismas reglas de la multiplicación decimal que se aprende en la educación básica; sin embargo, el uso de números binarios tiende a simplificar en mucho la operación. Supóngase que se desea multiplicar 1101_2 por 1011_2 :

$$\begin{array}{r}
 1101 \\
 \times 1011 \\
 \hline
 1101 \\
 1101 \\
 1101 \\
 1101 \\
 \hline
 10001111
 \end{array}$$

Se comienza multiplicando 1101_2 por el dígito menos significativo (1_2), para obtener el primer producto parcial. En seguida, se multiplica por el segundo dígito (1_2), produciendo el segundo producto parcial que se recorre hacia la izquierda una posición. Se continúa multiplicando con el tercer dígito (0_2). Como su valor es 0, simplemente se considera un recorrido de dos posiciones para la siguiente multiplicación. Al final, se realizan las sumas de cada columna, produciendo el producto final. De hecho, este proceso se repite sin importar el número de dígitos.

La multiplicación binaria resulta más sencilla que la multiplicación decimal, debido a que los productos parciales en binario se obtienen por multiplicar por 0 o 1, lo que resulta en un recorrido a la izquierda en el caso del 0, o una copia del valor multiplicado en el caso del 1.

A continuación se describe cómo el proceso de multiplicación se realiza en una computadora digital. Considérese, por ejemplo, que el producto de dos números de 4 bits da como resultado un número de 8 bits. De este modo, se considera el uso de registros de 8 bits. Esto es si se multiplican dos números como por ejemplo 00001011_2 y 00001101_2 . Nótese que los multiplicandos deben ser lo suficientemente pequeños para que su producto no exceda el tamaño del registro, lo que es el caso que se discute. Ahora bien, supóngase que se tienen a ambos multiplicandos en dos registros de corrimiento. Primero, se examina el bit más a la derecha del registro del primer multiplicando. Si es 1, entonces el segundo multiplicando se pasa a la ALU, donde se suma. Si el bit más a la derecha es 0, entonces se evita la suma. Se puede llevar a cabo esta operación mediante controlar la suma utilizando el bit más a la derecha del primer multiplicando conectado a la ALU. De este modo, la suma se realiza sólo si el bit más a la derecha del primer factor tiene valor de 1.

Cada vez que se encuentra un 0 o un 1 en el primer multiplicando, es necesario recorrer el contenido del registro del segundo multiplicando a la izquierda, de modo que los productos parciales se vayan sumando. El primer multiplicando se recorre a la derecha, lo que causa que el bit más a la derecha se pierda. Recuérdese que este bit controla la multiplicación, pero una vez utilizado, se pierde. Así, mediante corrimientos de los registros es posible multiplicar dos números binarios. El proceso debe repetirse por cada bit en el primer multiplicando. Cuando se termina, el contenido en el registro acumulador contiene el producto deseado.

Algunas computadoras digitales cuentan con circuitos cuya función es realizar la multiplicación. Tales circuitos contienen registros y acumuladores de modo que cuando se les provee de dos números binarios, producen su producto. Un circuito de este tipo se llama multiplicador alambrado (*hardwired multiplier*). No todos los multiplicadores alambrados funcionan como se ha descrito anteriormente. En ocasiones, se hacen simplificaciones a los circuitos; sin embargo, la idea básica es la misma. Si una computadora no cuenta con un multiplicador alambrado, debe ser programada paso a paso mediante instrucciones que realicen la multiplicación. En el siguiente capítulo se discute tal labor de programación, que muchas veces involucra muchos pasos. Pero si se cuenta con un multiplicador alambrado, entonces la única instrucción a ejecutar es el comando para realizar multiplicación; los circuitos de la computadora, sin embargo, son más complejos.

En seguida se discute cómo una computadora digital realiza la división. Así como la multiplicación involucra sumas, la división involucra restas. Supóngase que se quiere dividir 10001111_2 entre 1011_2 (esta es la operación inversa de la multi-

plicación realizada anteriormente). El primer paso es verificar si 1011_2 es mayor que 1000_2 . Ya que lo es, entonces no se intentaría el primer paso de la división; sin embargo, la computadora no puede notar esto, de modo que realiza la siguiente división:

$$\begin{array}{r} 1011 \overline{) 10001111} \\ \underline{1011} \end{array}$$

Después de realizar la resta, se prueba si el resultado es negativo (se discute un circuito que prueba si un número es negativo más adelante). Si lo es, entonces no se realiza el siguiente paso en la división. La computadora se programa, de modo que todos los pasos anteriores y erróneos se ignoren. En el ejemplo, es necesario hacer un nuevo intento:

$$\begin{array}{r} 01 \\ 1011 \overline{) 10001111} \\ \underline{1011} \\ 0110111 \end{array}$$

(De hecho, la sustracción se hace mediante el complemento a 2). De este modo, el procedimiento se repite hasta obtener el cociente 01101_2 .

Las computadoras pueden también contar con un divisor alambrado (*hardwired divider*), o pueden ser programadas para dividir números. El segundo caso resulta en un circuito más sencillo, pero requiere de mucho tiempo de programación.

2.7.6. Números de punto flotante

Es frecuente que en cálculos científicos o tecnológicos es necesario trabajar con números muy grandes o muy pequeños. Para evitar escribir largos números con muchos dígitos, se acostumbra expresar tales números en términos de una expresión que consta de un número multiplicado por 10 elevado a alguna potencia. La potencia es un número entero. Por ejemplo, las siguientes representaciones son equivalentes:

$$1763000 = 0.1763 \times 10^7$$

Similarmente:

$$0.00000000112304 = 0.112304 \times 10^{-7}$$

En una computadora, a esta representación se le conoce como notación de punto flotante (*floating point notation*), ya que el cambio en el exponente efectivamente permite mover la posición del punto decimal, que flota. En seguida se definen los términos utilizados en esta notación. Por ejemplo, los valores enteros 7 y -7 de las expresiones anteriores se les llama exponente, mientras que a 0.1763 y 0.112304 se les conoce con el nombre de mantisa o parte fraccional. De hecho, el primer nombre tiende a ser ambiguo, ya que este valor no es la mantisa de un logaritmo, de modo que aquí se prefiere el término parte fraccional.

Cuando se utilizan números de punto flotante en las computadoras digitales, normalmente se trabaja con potencias de 2 en lugar de potencias de 10. Por ejemplo, $64_{10} = 2^7$ puede escribirse como:

$$0.1 \times (10_2)^{1000_2}$$

Queda tal vez más claro de la siguiente forma:

$$0.1 \times 2^8$$

Es decir, la parte fraccional en binario, y el exponente en decimal (pero recuérdese, la computadora solo trabaja en binario).

A continuación, se aclara un punto que comúnmente causa confusión. Hay una diferencia entre el tamaño de un número y el número de cifras significativas. Por ejemplo, supóngase que se desea expresar una longitud en términos de millonésimas de centímetro como 1,736,000. Aun cuando la medida real fuera de 1,736,401, no es posible medir con la suficiente exactitud para notar esto. En tal caso, se dice que el número (1,736,000) se maneja con cuatro cifras significativas.

Ahora bien, cuando se trabaja con números en punto flotante, en realidad se trabaja con dos números: la parte fraccional y el exponente. Cada uno de ellos debe almacenarse aparte, pero normalmente se les puede incluir en el mismo registro. Por ejemplo, si el registro tiene 12 bits, los 8 bits menos significativos pueden utilizarse para almacenar la parte fraccional, y los 4 bits restantes para el exponente.

Supóngase que se tiene el número:

$$0.00010110111 \times 2^{-6} = 0.10110111 \times 2^{-9}$$

Requeriría de 11 bits para almacenar la parte fraccional de la expresión a la izquierda de la ecuación, mientras que sólo requiere de 8 bits para almacenar la parte fraccional de parte derecha de la ecuación. Y sin embargo, son el mismo número. Si la parte fraccional se almacena en un registro de 8 bits, entonces la expresión a la izquierda del igual tendría que ser 0.00010110_2 . Esto representa una pérdida de cifras significativas, y por tanto, una pérdida en exactitud. Para evitar esto, los exponentes de los números de punto flotante se ajustan para que el bit inmediatamente a la derecha del punto binario tenga valor 1. Se dice que tal número ha sido normalizado o escalado. En tal caso, no se desperdicia espacio de memoria almacenando ceros. Nótese que el punto binario y el cero a la izquierda no se almacenan. Por ejemplo, la parte fraccional almacenada de la última expresión sería 10110111_2 . En un valor punto flotante escalado, la posición del punto binario se considera inmediatamente a la izquierda del número, y por tanto no se necesita almacenar.

Respecto al signo, tanto la parte fraccional como el exponente pueden ser positivos o negativos. Por tanto, debe haber una regla para bits de signo: un bit se utiliza para almacenar el signo del exponente, y otro bit para el signo de la parte fraccional.

Para averiguar de qué tamaño es el número con que se puede trabajar en una computadora, es necesario hablar de almacenamiento. Un registro almacena un número dado en bits. Grupos de 8 bits se conocen con el nombre de *byte*. Normalmente, todos los registros dentro de una computadora tienen la misma longitud. A esto se le conoce con el nombre de tamaño de palabra (*word size*) de una computadora. Valores típicos de tamaño de palabra son 8, 16, 32, y 64.

En seguida, se analiza qué tan grande o qué tan pequeño puede ser un número en la computadora. Supóngase que el tamaño de palabra es de 32 bits. Estos deben almacenar tanto el exponente, como la parte fraccional, cada uno con su signo. Dos bits de signo de 32 bits dejan 30 bits disponibles. Normalmente, 6 bits se utilizan para el exponente, mientras que los otros 24 se dejan para la parte fraccional. Ahora bien, el número más grande que puede almacenarse como exponente es:

$$2^6 - 1 = 63_{10}$$

Y por lo tanto:

$$2^{63} = 9.223 \times 10^{18}$$

Este es un número muy grande. De forma similar, 2^{-63} es un número muy pequeño. Por lo tanto, la gama de valores numéricos de punto flotante que se

pueden expresar en la computadora varían al ir dando valores binarios a la parte fraccional y al exponente.

Se supone anteriormente que en la representación de punto flotante el exponente se aplica únicamente a una potencia de 2. En realidad, se puede utilizar cualquier otro número. Por ejemplo, en algunas computadoras, se trata de potencias de 16; circuitos digitales apropiados se desarrollan para tomar en cuenta esto. Ahora bien, si se cuenta con 6 bits para almacenar la magnitud del exponente, se tiene el mismo número 63_{10} para el exponente; sin embargo, ahora se trata de una potencia de 16_{10} . De este modo, la parte exponencial puede ser tan grande como:

$$16^{63} = 7.237 \times 10^{75}$$

Esto es un número mucho más grande que el obtenido mediante utilizar el exponente como potencia de 2.

En cuanto a la parte fraccional, ésta se almacena en los restantes 24 bits. Por ejemplo, un número almacenado puede ser:

$$0.101100111100110010110011$$

Mientras más bits se almacenen, más precisión se tiene. Supóngase que sólo se almacenan 23 bits. El último bit a la derecha se perdería. Este bit representa un valor de:

$$2^{-24} = 0.0000000596_{10}$$

De hecho, al perder este valor, se tiene que los números

$$0.12461281_{10} \quad \text{y} \quad 0.12461282_{10}$$

se almacenarían de la misma forma binaria. Nótese que hay siete cifras significativas en tales números porque hay siete ceros a la derecha del punto decimal en la representación base 10 de 2^{-24} .

Estos números son bastante precisos. En la mayoría de las aplicaciones científicas y tecnológicas, no es posible medir datos con tal precisión. Por otro lado, se pierde precisión cuando se realizan operaciones. Por ejemplo, cuando se multiplican dos números de 8 bits en su parte fraccional, se obtiene un número con 16 bits en la parte fraccional del resultado. Si se eliminan los 8 bits menos significativos, se pierde exactitud y precisión. Algunos programas de computadora requieren muchas operaciones, y si se va perdiendo precisión en cada una de ellas, el resultado

final tendrá un gran error. Por tanto, se requiere de una gran precisión cuando se realizan operaciones complejas, y en ocasiones, siete u ocho cifras significativas resultan insuficientes; sin embargo, la mayoría de las computadoras pueden ser programadas de modo que dos o más registros o localidades de memoria puedan utilizarse para almacenar un solo número, lo que provee de precisión adicional. Esta acción de agrupar registros se conoce comúnmente como doble precisión.

Anteriormente, se considera un tamaño de palabra de 32 bits. La mayoría de las computadoras actuales tienen tal tamaño de palabra, pero en el pasado, muchas computadoras contaban tan solo con 8 o 16 bits. Aun cuando puede parecer que estas computadoras no eran adecuadas para realizar operaciones, este no es el caso. Muchos programas realizan operaciones numéricas que no requieren muchas cifras significativas o números muy grandes o pequeños. En tal caso, no resulta necesario utilizar muchos bits para almacenar la parte fraccional y el exponente. Además, hay un gran número de programas que simplemente no se utilizan propiamente para realizar operaciones numéricas. Por ejemplo, programas que ordenan listas de nombres, mantienen un inventario, juegos, o controlan sistemas de alarma no requieren complicadas operaciones numéricas, dado que no trabajan con números con muchos dígitos. Y si se encuentra apropiadamente programada, una computadora con un tamaño pequeño de palabra puede utilizarse para cómputo científico o tecnológico. Como se discute anteriormente, se pueden utilizar varios registros o localidades de memoria para almacenar un solo número. Es claro que esto efectivamente representa una pérdida en la cantidad de memoria disponible; sin embargo, se puede lograr operaciones más complejas si se cuenta con la memoria suficiente para mantener la precisión.

2.7.7. Suma y resta de punto flotante

Cuando se suman o restan dos números representados en punto flotante, los exponentes de ambos números deben tener el mismo valor. Por ejemplo, si se desea sumar 0.1001×2^8 y 0.1011×2^7 se requiere mover el punto de alguno de los dos valores, como por ejemplo:

$$0.1001 \times 2^8 + 0.01011 \times 2^8 = 0.11101 \times 2^8$$

Si solo se almacenan 4 bits para la parte fraccional, la operación provoca una pérdida de precisión debido a que en lugar de almacenar la respuesta correcta (0.11101×2^8) se almacena 0.1110×2^8 . A esto se conoce como error de redondeo.

Comúnmente, estos errores son lo suficientemente pequeños como para ser ignorados; sin embargo, cuando se desea mayor precisión, una solución puede ser el uso de doble precisión.

2.7.8. Multiplicación y división de punto flotante

Cuando se multiplican o dividen números en punto flotante, se utilizan las reglas ordinarias de la multiplicación y la división de números con exponentes. Supóngase que se tiene dos números:

$$A = F_a \times 2^{E_a}$$

$$B = F_b \times 2^{E_b}$$

donde F_a y F_b son las partes fraccionarias, y E_a y E_b son los exponentes. Para multiplicar ambos números, se multiplican las partes fraccionarias y se suman los exponentes:

$$A \times B = (F_a \times F_b) \times 2^{E_a + E_b}$$

Por ejemplo, supóngase que:

$$A = 0.1011 \times 2^5$$

$$B = 0.1101 \times 2^7$$

Entonces:

$$A \times B = 0.010001111 \times 2^{12}$$

En una computadora digital, este número sería escalado y almacenado de la siguiente forma:

$$A \times B = 0.10001111 \times 2^{11}$$

Nótese que hay más bits en la parte fraccional del producto que en A o B, lo que significa que fácilmente puede ocurrir un error de redondeo.

Para realizar la división, se utiliza la siguiente expresión:

$$\frac{A}{B} = \frac{F_a}{F_b} \times 2^{E_a - E_b}$$

Es decir, cuando se realiza una división en punto flotante, se dividen las partes fraccionales y se restan los exponentes. Por ejemplo:

$$\frac{0.100001111 \times 2^{11}}{0.1011 \times 2^5} = 0.01101 \times 2^6 = 0.1101 \times 2^5$$

2.8. La unidad lógica-aritmética (ALU)

En la sección anterior se discute una ALU sencilla que puede sumar dos números y almacenar su suma en un registro. En esta sección se discute una ALU más compleja que puede realizar varias operaciones, y que es similar a la ALU interna del procesador de una computadora pequeña.

Donde la ALU sencilla de la Figura 2.63 usa flip-flops D para el registro, en la ALU actual se utilizan flip-flops JK, ya que se obtiene un control adicional mediante su uso. Para cada operación que se desee obtener, se muestra un circuito sencillo. Después de discutir todas las operaciones, se muestra cómo todos estos circuitos simples pueden ser combinados. Ya que todas las etapas de la ALU son idénticas, se trabaja con una sola etapa. Por ejemplo, la ALU de la Figura 2.63 consiste de tres etapas idénticas.

La ALU descrita aquí requiere de muchas terminales de control. Cada una de ellas se utiliza para controlar una sola operación. Se supone por simplicidad que la señal digital en las terminales de todas las demás operaciones tiene valor 0. Por ejemplo, en la Figura 2.63 hay dos terminales de control ADD y CLR. Si se desea que la ALU realice una suma, entonces $ADD = 1$ y $CLR = 0$.

Muchas señales se le aplican a los flip-flops JK mediante un par de compuertas OR como se muestra en la Figura 2.64. El significado de este circuito se aclara cuando se haya discutido la ALU completa.

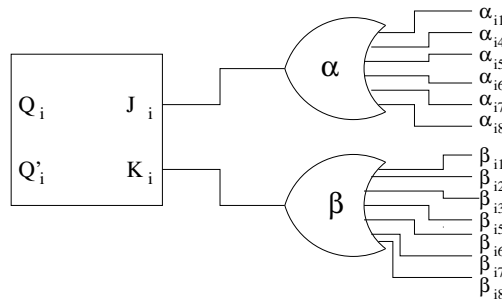


Figura 2.64: Entradas al i -ésimo flip-flop de la ALU.

2.8.1. Adición e incremento

Considérese un circuito que suma un número B al número almacenado en el acumulador. Se supone que hay $N + 1$ bits almacenados en el acumulador; los dígitos del número B son:

$$b_N b_{N-1} \dots b_3 b_2 b_1 b_0$$

Ahora bien, considérese la etapa de la ALU que suma el i -ésimo bit y almacena la suma resultante. Si los dígitos del número que está almacenado en el acumulador son:

$$a_N a_{N-1} \dots a_3 a_2 a_1 a_0$$

entonces la i -ésima etapa realiza la suma $b_i + a_i + c_{i-1}$, donde c_{i-1} es el acarreo de la etapa anterior. El circuito que se considera aquí en realidad realiza dos funciones: además de sumar el número B al contenido del acumulador, también es capaz de añadir el valor 1 al contenido almacenado. Por tanto, el circuito tiene dos terminales de control: una etiquetada ADD para realizar una suma ordinaria y otra etiquetada INC para incrementar el contenido del acumulador en una unidad.

Considérese la operación del circuito digital de la Figura 2.65. Si la señal ADD tiene valor 1, entonces INC debe tener valor 0. Es decir, solo una de estas dos señales de control puede tener valor de 1 en un momento dado.

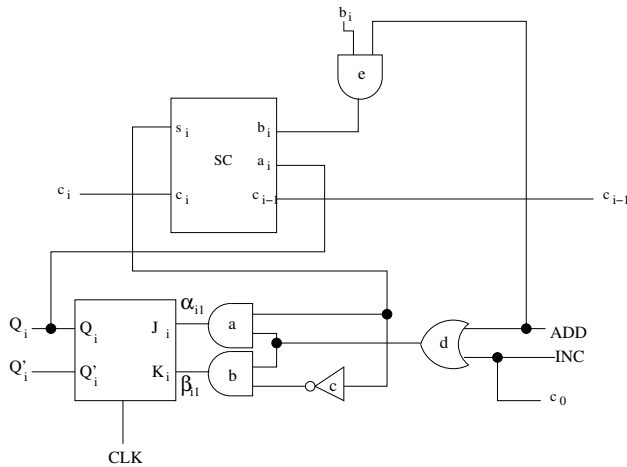


Figura 2.65: Etapa de la ALU que realiza ya sea la suma o el incremento.

En tal situación, el bit b_i se aplica en la entrada b_i del sumador completo (SC). Si el i -ésimo bit de B tiene valor 1, entonces $b_i = 1$; si tiene valor 0, entonces $b_i = 0$. Además, una entrada de cada compuerta AND **a** y **b** tiene valor 1, y por lo tanto:

$$\begin{aligned} J_i &= s_i \\ K_i &= s'_i \end{aligned}$$

En tal caso, el flip-flop JK funciona como un flip-flop D. De hecho, este circuito por lo pronto funciona igual a una de las etapas de la ALU en la Figura 2.63.

Ahora bien, supóngase que $\text{ADD} = 0$ e $\text{INC} = 1$. Una entrada a la compuerta AND **e** tiene valor 0, por lo que el bit b_i no pasa por la compuerta. El circuito aún funciona como un sumador con entrada cero, excepto por una diferencia. La entrada c_0 del primer sumador no se utiliza durante la suma, ya que no hay acarreo de la columna más a la derecha. Cuando $\text{INC} = 1$, entonces tal entrada tiene valor de 1. Nótese que el primer sumador completo toma la suma $b_0 + a_0 + c_0$. Por lo tanto, una entrada c_0 actúa de la misma forma en que actúa b_0 . Así, cuando $\text{INC} = 1$, entonces $c_0 = 1$, lo que hace que un 1 se sume al valor almacenado en el acumulador. Después del siguiente pulso de reloj, tal número se almacena en el acumulador.

Nótese las dos conexiones etiquetadas α_{i1} y β_{i1} . Supóngase que el flip-flop JK fuera reemplazado por el flip-flop con las compuertas OR de la Figura 2.64. En tal caso, la salida de la compuerta AND **a** en la Figura 2.65 se conectarían a la compuerta OR α por su entrada α_{i1} . De manera similar, la salida de la compuerta AND **b** se conectaría a la compuerta OR β , entrada β_{i1} .

Ahora bien, supóngase que todas las demás entradas α_i y β_i en la Figura 2.64 tienen valor 0. El circuito funcionaría exactamente como el circuito de la Figura 2.65, ya que si α_i y β_i tienen valor 0, entonces:

$$\begin{aligned} J_i &= s_i \\ K_i &= s'_i \end{aligned}$$

tal y como originalmente se deseaba. En conclusión, la inclusión de las compuertas OR no cambia la operación del circuito ya que todas las demás entradas α_i y β_i tienen valor 0. De hecho, la presencia de las compuertas OR permite obtener las diferentes funciones de la ALU que se requieren. Nótese que cuando $\text{ADD} = 0$ e $\text{INC} = 0$, entonces la salida es $\alpha_i = \beta_i = 0$. Ahora bien, en cuanto a la operación de las compuertas OR α y β concierne, las entradas α_{i1} y β_{i1} podrían quitarse o

dejar de considerarse, mientras que las otras entradas a las compuertas pueden utilizarse para obtener otras diferentes funciones.

2.8.2. Limpieza (CLEAR)

Si se desea limpiar el registro acumulador, es decir, poner un valor 0 en todos y cada uno de sus bits, entonces se utiliza el circuito que se muestra en la Figura 2.66. En este caso, $J_i = 0$ (nótese que no hay entrada a J). Cuando $\text{CLR} = 1$, entonces $K_i = 1$, y por lo tanto, el flip-flop se coloca en estado $Q_i = 0$ al siguiente pulso de reloj. Cuando se reemplaza el flip-flop por el circuito de la Figura 2.64, la entrada CLR se aplica en la entrada β_{i2} . Todas las demás entradas α_i y β_i deben tener valor 0. Por lo tanto, con $\text{CLR} = 1$, se tiene que $J_i = 0$ y $K_i = 1$, y la limpieza deseada ocurre (nótese que la entrada α_{i2} es innecesaria, ya que su valor es 0).

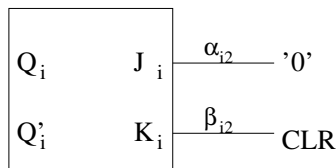


Figura 2.66: Etapa de la ALU que realiza la limpieza del acumulador.

2.8.3. AND lógica

Hay ocasiones cuando se desea realizar operaciones lógicas utilizando como entradas los valores del registro B y el acumulador (b_i y a_i). Considérese la operación AND. El i -ésimo bit del número binario B se compara con la salida Q_i , que es el i -ésimo bit almacenado en el acumulador. Si b_i y Q_i tienen ambos valor 1, entonces el valor de Q_i permanece siendo 1. Si cualquiera de los dos, o ambos b_i o Q_i , tienen valor 0, entonces Q_i toma valor 0. El circuito que realiza tal operación se muestra en la Figura 2.67.

Nótese que si $\text{AND} = 1$ y $b_i = 1$, entonces $\beta_{i3} = 0$, y por lo tanto, el estado del flip-flop no cambia. Si $Q_i = 1$, éste permanece con ese valor. De manera similar, si $Q_i = 0$, entonces el estado permanece igual. Por otro lado, si $b_i = 0$, entonces

cuando $AND = 1$, se tiene que $K_i = 1$ y $J_i = 0$, por lo que al siguiente pulso de reloj, $Q_i = 0$. Si $AND = 0$, entonces $\beta_{i3} = 0$, lo que hace que no haya cambio en el estado del flip-flop.

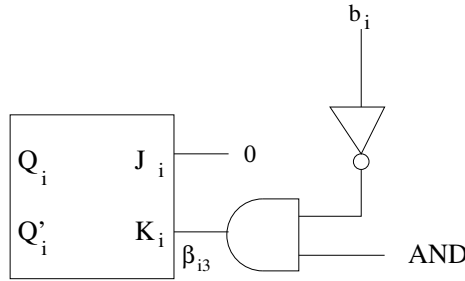


Figura 2.67: Etapa de la ALU que realiza la AND lógica.

Si se reemplaza el flip-flop de la Figura 2.64 con β_{i3} conectado a la entrada correspondiente de la compuerta OR β , el circuito funciona tal y como se ha descrito. Nótese que si $AND = 0$, entonces $\beta_{i3} = 0$, por lo que esta entrada no interfiere con las otras entradas.

2.8.4. OR lógica

La operación OR lógica se realiza de la siguiente manera: el bit b_i de B se compara con el bit Q_i almacenado en el acumulador. Si cualquiera de los dos o ambos tienen valor de 1, entonces se coloca un valor de 1 en el i -ésimo flip-flop del acumulador en el siguiente pulso de reloj. Si ambos, Q_i y b_i tienen valor 0, entonces se coloca un 0 en Q_i . Por lo tanto, Q_i permanece sin cambio a menos que $b_i = 1$ y $Q_i = 0$. Solo entonces Q_i toma valor de 1. Un circuito que realiza esta operación se muestra en la Figura 2.68.

Cuando $OR = 1$ y $Q_i = 1$ entonces Q_i permanece con valor 1, independientemente del valor de b_i . Si $OR = 1$ y $Q_i = 0$ con $b_i = 1$ entonces al siguiente pulso de reloj, Q_i toma el valor de 1. Así, el circuito opera correctamente la OR lógica. Nótese que si $OR = 0$, entonces $\alpha_{i4} = 0$. De este modo, se puede reemplazar el flip-flop por el flip-flop de la Figura 2.64.

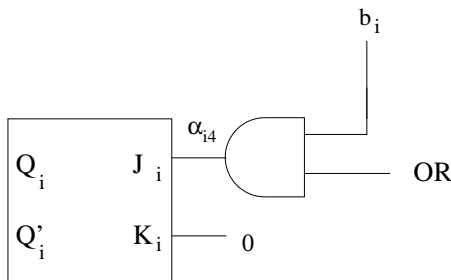


Figura 2.68: Etapa de la ALU que realiza la OR lógica.

2.8.5. XOR lógica

La operación XOR lógica se realiza mediante comparar el bit b_i de B con el bit Q_i del acumulador. Si cualquiera de ellos, Q_i o b_i tienen valor 1, entonces, después del siguiente pulso de reloj, se almacena un valor de 1 en el flip-flop. Si ambos b_i y Q_i tienen valor 0, o si ambos b_i y Q_i tienen valor 1, al siguiente pulso de reloj, Q_i toma valor 0. Un circuito que implementa este comportamiento se muestra en la Figura 2.69.

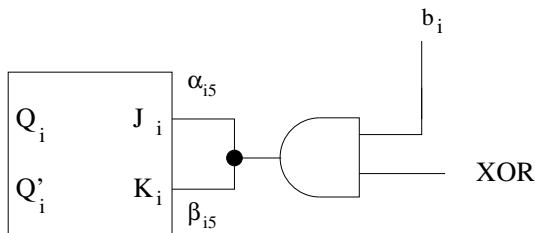


Figura 2.69: Etapa de la ALU que realiza la XOR lógica.

Si $XOR = 1$ y $b_i = 1$, entonces, después del pulso de reloj, el estado del flip-flop cambia. Así, si $Q_i = 0$, resulta en que $Q_i = 1$; de manera similar, si $Q_i = 1$, el resultado es $Q_i = 0$. Esto va de acuerdo con la operación XOR. También, si $XOR = 1$ y $b_i = 0$, si $Q_i = 0$, entonces Q_i permanece en ese estado; si $Q_i = 1$, entonces también permanece en tal estado. Finalmente, si $XOR = 0$, entonces $\alpha_{i5} = 0$ y $\beta_{i5} = 0$, por lo que el flip-flop puede substituirse por el flip-flop en la Figura 2.64.

2.8.6. Corrimiento a la derecha

Comenzando con las operaciones sobre los contenidos del acumulador, se inicia discutiendo las operaciones de corrimiento. Estos circuitos son similares en su operación al circuito de la Figura 2.55. Para correr los contenidos del acumulador a la derecha, la entrada de cada flip-flop debe provenir de la salida del flip-flop inmediatamente a su izquierda. El circuito utilizado se muestra en la Figura 2.70.

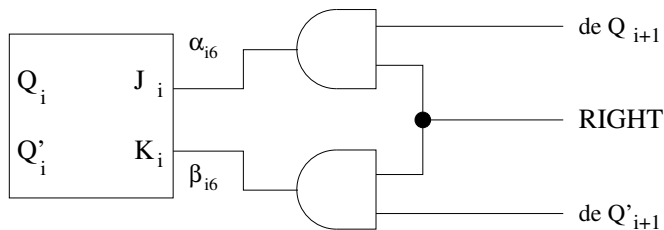


Figura 2.70: Etapa de la ALU que realiza el corrimiento a la derecha.

Nótese que el flip-flop más a la izquierda no tiene entrada. Para darle una entrada se reemplazan Q_{i+1} y Q'_{i+1} de la Figura 2.70 por entradas externas. Así, el acumulador funciona como un registro de corrimiento.

Ahora bien, cuando $RIGHT = 1$, entonces el corrimiento ocurre. Si $RIGHT = 0$, entonces $\alpha_{i6} = \beta_{i6} = 0$, y no hay corrimiento. De nuevo, es posible reemplazar el flip-flop por el circuito de la Figura 2.64.

2.8.7. Corrimiento a la izquierda

El circuito que corre los contenidos del acumulador a la izquierda es esencialmente el mismo que el circuito para corrimiento a la derecha, excepto que ahora las entradas provienen del flip-flop inmediatamente a la derecha. La entrada del flip-flop más a la derecha consiste de una entrada externa y su complemento. Un circuito que realiza el corrimiento a la izquierda se muestra en la Figura 2.71. De nuevo, nótese que cuando $LEFT = 0$, entonces $\alpha_{i7} = \beta_{i7} = 0$, por lo que se puede reemplazar el flip-flop por el circuito de la Figura 2.64.

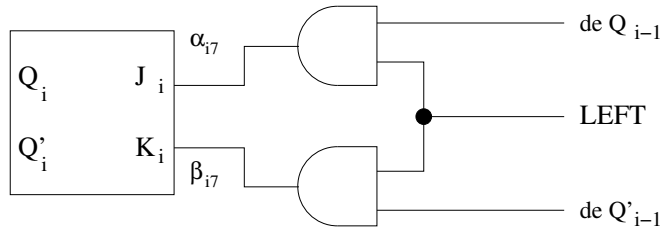


Figura 2.71: Etapa de la ALU que realiza el corrimiento a la izquierda.

2.8.8. Complemento

El circuito que se muestra en la Figura 2.72 muestra cómo se puede complementar todos los bits almacenados en el acumulador. Este es útil para obtener el complemento a 2. El flip-flop se conecta como un flip-flop T. Cuando $COMP = 1$, cambia el estado a su valor complemento. De nuevo, nótese que cuando $COMP = 0$, se tiene que $\alpha_{i8} = \beta_{i8} = 0$, por lo que también se puede reemplazar el flip-flop por el circuito de la Figura 2.64.

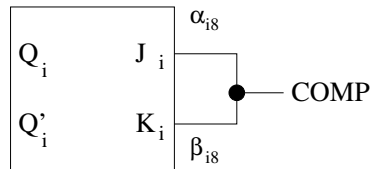


Figura 2.72: Etapa de la ALU que realiza complemento.

2.8.9. Conexión de entradas

Ya que se han considerado un número de operaciones, recuérdese que cuando los circuitos de las Figura 2.65 a 2.72 se conectan juntos, solo una señal de control puede tomar el valor de 1 en cualquier momento, mientras que el resto debe permanecer con valor 0. De hecho, por ahora es notorio que en lugar de utilizar un solo flip-flop, se utiliza la conexión de la Figura 2.64, donde hay dos compuertas OR conectadas a las entradas J y K. Por lo tanto, cada una de las Figuras 2.65 a

2.72 es una parte que se han dividido en los puntos de conexión marcados como α_{ik} y β_{ik} , que se conectan a las entradas de las compuertas OR como se indica en la Figura 2.64. Cada operación funciona como se ha descrito en las secciones anteriores. Por ejemplo, supóngase que $ADD = 1$, y todas las demás señales de control tienen valor 0. Por lo tanto:

$$\alpha_{i4} = \alpha_{i5} = \alpha_{i6} = \alpha_{i7} = \alpha_{i8} = \beta_{i3} = \beta_{i5} = \beta_{i6} = \beta_{i7} = \beta_{i8} = 0$$

Y por lo tanto:

$$J_i = \alpha_{i1}$$

$$K_i = \beta_{i1}$$

Esto es exactamente la condición para el funcionamiento del circuito de la Figura 2.65, y significa que una operación de suma se realiza. Una consideración similar puede tenerse con todas y cada una de las otras operaciones y señales de control.

2.8.10. Inspección de salidas

En esta sección se discuten dos operaciones de la ALU que proveen de información sobre el valor almacenado en el acumulador, pero no cambian los datos almacenados de ninguna forma: inspección negativa y de cero.

Inspección negativa

Para esta inspección, se desea una señal de salida que indique un valor de 1 si el número almacenado en el acumulador es negativo, y 0 de otra forma. Se supone que se utiliza una representación de complemento a 2. Así, el número es negativo si el bit más significativo tiene valor de 1, y será positivo si tiene valor 0. Por lo tanto, sólo se requiere examinar el estado del flip-flop más a la izquierda del acumulador para determinar si el número es negativo.

Un circuito que cumple con esta función se muestra en la Figura 2.73. La señal de salida se marca como NEG, de modo que si $NEG = 1$, entonces γ_1 tiene el valor de Q_N . Ya que Q_N es el bit almacenado más a la izquierda, γ_1 tiene valor de 1 cuando el número sea negativo, y tiene valor de 0 cuando el número sea positivo o cero.

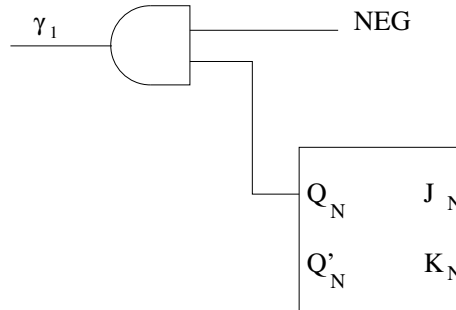


Figura 2.73: Circuito de inspección negativa.

Inspección de cero

También, frecuentemente es necesario verificar si el valor almacenado en el acumulador es 0. Si tal es el caso, entonces se desea saber si:

$$Q_0 = Q_1 = Q_2 = \dots = Q_N = 0$$

En este caso, si:

$$Q'_0 = Q'_1 = Q'_2 = \dots = Q'_N = 1$$

El circuito de la Figura 2.74 realiza la inspección de cero. Cuando la salida ZERO tiene valor de 1, entonces γ_2 tiene valor de 1 si y solo si se satisface la expresión anterior. Si $\gamma_2 = 0$ cuando ZERO = 1, entonces el valor almacenado en el acumulador no es 0. Nótese que cuando ZERO = 0, la salida γ_2 tiene valor 0.

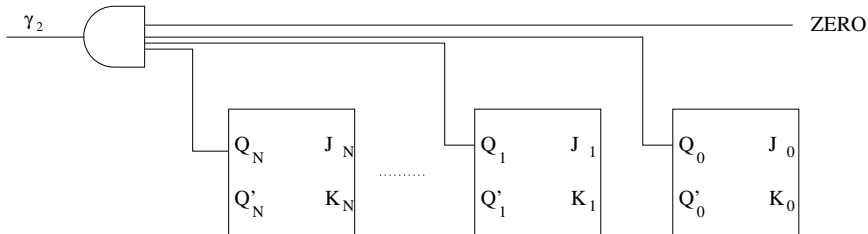


Figura 2.74: Circuito de inspección de cero.

2.8.11. Conexión de salidas

En las dos secciones anteriores se han tratado dos salidas correspondientes a las inspecciones negativa y de cero. Comúnmente, las ALU tienen una sola salida para ambas operaciones. Si $ZERO = 1$, entonces la salida da el resultado de la inspección de cero. Similarmente, si $NEG = 1$, entonces se da la salida de inspección negativa. Tal única salida puede obtenerse de las señales γ_1 y γ_2 , como se muestra en la Figura 2.75. Recuerdese que las dos operaciones de inspección no pueden efectuarse al mismo tiempo.

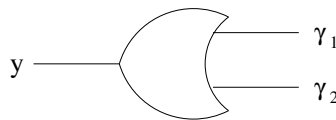


Figura 2.75: Circuito que provee una sola salida para las inspecciones negativa y de cero.

Así, se termina de discutir una ALU completa que realiza muchas operaciones; sin embargo, no se han discutido todas las posibles operaciones que una ALU puede realizar, sino más bien, se han expuesto las más comunes. Más importante aún, es que la comprensión de esta ALU debe permitir el comprender otras ALU, aún cuando éstas realicen otras operaciones. En el siguiente capítulo se discute cómo la ALU se inserta dentro de una computadora digital completa.

Fuentes útiles de información

Knapp, S.

Frequently-Asked Questions about Programmable Logic.

<http://www.optimagic.com/faq.html>

Evaluación

Ejercicios

1. Simplifique las siguientes expresiones utilizando identidades y leyes del álgebra booleana:

$$F1 = ABC(ABC' + AB'C + A'BC)$$

$$F2 = AB + A'C + AB' + A'C'$$

2. El circuito de la Figura 2.76 puede representarse por una sola compuerta ¿cuál?

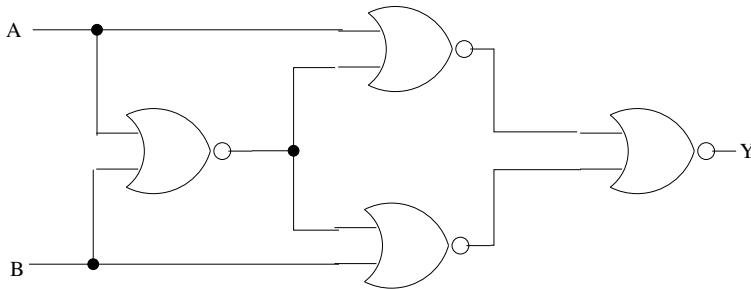


Figura 2.76: Circuito del ejercicio 2.

3. Determine la expresión para Y de la Figura 2.77.

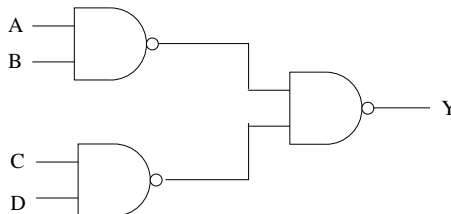


Figura 2.77: Circuito del ejercicio 3.

Preguntas de opción múltiple

1. El número decimal 172 se representa en binario como:
 - a) 10101100.
 - b) 00110101.
 - c) 10101010.
 - d) 11001100.
 - e) 11100111.
2. ¿Cuál es la principal razón para el uso de lógica binaria en sistemas de cómputo?
 - a) Robustez contra el ruido.
 - b) Procesamiento paralelo de datos.
 - c) Menor alambrado.
 - d) Computadoras más pequeñas.
 - e) Por convención.
3. ¿Cuál es el número mínimo de bits requerido para representar todos los caracteres en un teclado que tiene nueve teclas?
 - a) 1.
 - b) 2.
 - c) 3.
 - d) 4.
 - e) 5.
4. ¿Cuál es el mínimo número de bits requerido para representar todos los caracteres en un teclado de 22 teclas?
 - a) 1.
 - b) 2.
 - c) 3.
 - d) 4.

e) 5.

5. La función XOR puede expresarse en términos de ANDs, ORs y NOTs como:

a) $(A'B') + (AB)$.

b) $(A'B) + (AB')$.

c) $(A'A) + (BB)$.

d) $(A' + B')(A + B')$.

e) $((AB')A) + B'$.

6. El resultado de reagrupar la expresión $Y = ((ABC)(AB')) + ((A + B) + A) + (A' + (A'B))$ es:

a) $Y = A$.

b) $Y = B$.

c) $Y = 0$.

d) $Y = 1$.

e) $Y = A'$.

7. Un dígito binario es conocido como:

a) Byte.

b) Bit.

c) Word.

d) And.

e) Or.

8. Un demultiplexor tiene:

a) Una entrada y múltiples salidas.

b) Una salida y múltiples entradas.

c) Una entrada y una salida.

d) Múltiples entradas y múltiples salidas.

e) Ninguna de las anteriores.

9. Un multiplexor tiene:
- a) Una entrada y múltiples salidas.
 - b) Una salida y múltiples entradas.
 - c) Una entrada y una salida.
 - d) Múltiples entradas y múltiples salidas.
 - e) Ninguna de las anteriores.
10. El número de variables de estado necesarias para diseñar una FSM con seis posibles estados es:
- a) 1.
 - b) 2.
 - c) 3.
 - d) 4.
 - e) 5.
11. Considérese la FSM en la Figura 2.78.

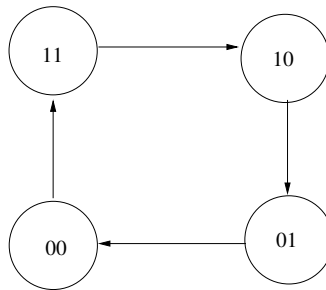


Figura 2.78: Circuito de la pregunta 11.

Este diagrama representa la funcionalidad de:

- a) Un contador síncrono con la secuencia 0-1-2-3.
- b) Un contador asíncrono con la secuencia 0-1-2-3.
- c) Un contador síncrono con la secuencia 3-2-1-0.

- d*) Un contador asíncrono con la secuencia 3-2-1-0.
- e*) Un contador síncrono con la secuencia 3-2-3-0.

Respuestas a las preguntas de opción múltiple

- 1. *a*
- 2. *a*
- 3. *d*
- 4. *e*
- 5. *b*
- 6. *d*
- 7. *b*
- 8. *a*
- 9. *b*
- 10. *c*
- 11. *c*

Capítulo 3

Componentes de un sistema de cómputo

Resumen

Este capítulo discute los principales cuatro componentes de un sistema de cómputo: el procesador, los buses, la memoria y los dispositivos de entrada/salida. Se presenta una introducción a las instrucciones, secuenciación y sistemas de *software* para ayudar al entendimiento del papel y funcionalidad de la unidad central de proceso.

Objetivos del capítulo

Al final del capítulo, el estudiante debe ser capaz de:

1. Enumerar los principales componentes de una computadora digital.
2. Entender la manera en que estos componentes se interconectan y comunican a fin de realizar tareas.
3. Explicar el papel de componentes como la ALU, unidad de control, buses e interfaces dentro de un sistema de cómputo.
4. Diferenciar entre los principales tipos de contenidos de memoria (instrucciones y datos) y explicar cómo las instrucciones se buscan y ejecutan.

5. Reconocer los tipos de memoria principal dentro de un sistema de cómputo y definir su papel.
6. Analizar al nivel de diagramas lógicos la funcionalidad de una celda de memoria de un solo bit.

3.1. Organización de una computadora digital

El diagrama de bloques de la Figura 3.1 puede utilizarse para describir la organización general de una computadora digital. Cuenta con cuatro unidades básicas. Todas las operaciones se realizan en la unidad lógica-aritmética (ALU). En una computadora pequeña, esta unidad consiste del tipo de elementos descritos anteriormente, posiblemente con registros adicionales; la ALU de muchas computadoras (hasta en computadoras pequeñas) frecuentemente contiene varios registros acumuladores. Anteriormente ya se han discutido todos los tipos de circuitos que se utilizan normalmente en una ALU típica.

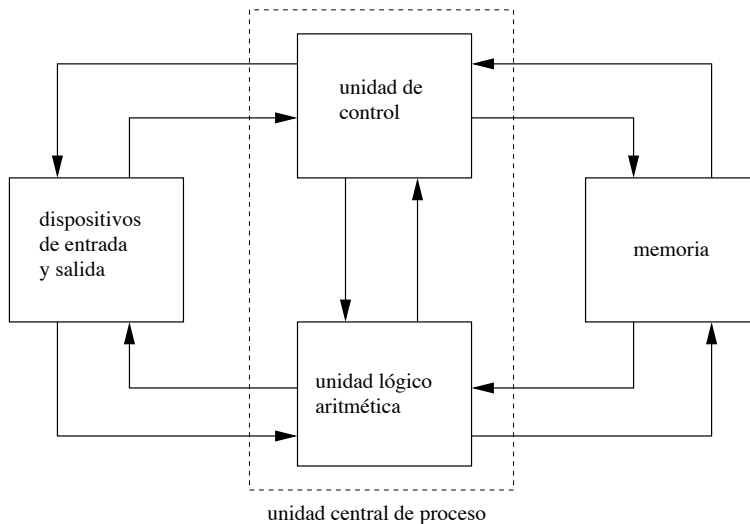


Figura 3.1: Diagrama de bloques de una computadora digital básica.

En la sección 2.8 se muestra que la ALU realiza las operaciones mediante seleccionar de un conjunto de señales de control, tales como ADD, CLEAR y OR. Estas señales se generan por la unidad de control, que recibe sus instrucciones de un programa. Algunas unidades de control son más complejas que otras. Por ejemplo, en la sección 2.7.4 se discute la sustracción mediante complementar el sustraendo, sumarle 1 y sumar el resultado al minuendo. De este modo, tres instrucciones deben proveerse a la ALU: complemento, incremento y suma. Si la unidad de control fuera una muy simple, las tres instrucciones tendrían que proveerse por el programa mismo. Con una unidad de control más compleja, muy probablemente solo requiera una instrucción de sustracción. La propia unidad de control se encargaría entonces de generar las tres señales de control apropiadas, en la secuencia adecuada, y en los tiempos precisos.

Además de controlar el flujo de datos desde y hacia los dispositivos de entrada y salida, la unidad de control también dirige las acciones de lectura y escritura de la memoria. La unidad de control también contiene el reloj maestro que se encarga de sincronizar la operación de todas las partes de la computadora. Nótese la línea punteada de la Figura 3.1, que conjunta la unidad de control y la ALU. En muchas computadoras, ambas unidades se encuentran unidas en un solo circuito integrado. En tal caso, este circuito se conoce como microprocesador. De cualquier modo, la ALU y la unidad de control juntas se consideran componentes de la unidad central de proceso (*central processing unit*, o simplemente, CPU).

A su vez, la memoria puede considerarse dividida en dos partes: la unidad de memoria principal (*main memory unit* o MMU) y la memoria auxiliar. La MMU es una memoria de acceso aleatorio como las que se discuten en el Capítulo 4. En las computadoras pequeñas, esta unidad consiste de una memoria semiconductora. La memoria auxiliar, por otro lado, consiste de discos y cintas, como las descritas en el mismo Capítulo 4.

Para poder utilizar una computadora, es necesario proveerle de programas y datos. Después de que el procesamiento se ha llevado a cabo, la computadora a su vez debe ser capaz de proveer los resultados. La comunicación con la computadora es función de los dispositivos de entrada y salida. El dispositivo de entrada más común es el teclado, en el cual un programador puede escribir, letra por letra, su programa. El teclado genera las secuencias de unos y ceros que la computadora interpreta. Otra forma de entrada puede ser mediante discos y cintas, en los que las secuencias de unos y ceros ya se encuentran almacenados. Nótese que las cintas y los discos pueden considerarse tanto como dispositivos de entrada y salida,

así como memoria auxiliar. Si su función es proveer datos o programas, o si los resultados del procesamiento finalmente se almacenan en ellos, entonces se les considera dispositivos de entrada/salida; si almacenan valores como resultados intermedios, entonces puede considerárseles como parte de la memoria.

Existen otros tipos especiales de dispositivos de entrada. Por ejemplo, las lectoras de los supermercados pueden leer los códigos de barras que representan los precios de los artículos. Estos lectores proveen de información a una computadora, la cual al final obtiene el total de la compra, pero puede además realizar actividades más complejas, como utilizar la información de los artículos comprados para manejar el control de inventarios.

El dispositivo de salida más común es la pantalla. Esta no es otra cosa más que un tubo de rayos catódicos (*cathodic ray tube* o simplemente CRT) parecida a un receptor de televisión. Otro dispositivo de salida común es la impresora, que permite obtener la salida en hojas de papel.

En muchos casos, la entrada y salida de una computadora se manejan de forma enteramente diferente a lo que se describe aquí. Por ejemplo, los datos de entrada pueden provenir de sensores de temperatura, luz o humedad colocados en lugares específicos de un aparato o un edificio. La salida puede consistir entonces en señales que controlen actuadores, los cuales echan a andar calefactores o humidificadores en el propio aparato o edificio.

3.2. La unidad central de proceso

Esta sección discute más a fondo el componente principal de un sistema de cómputo: la unidad central de proceso (*Central Processing Unit* o CPU).

3.2.1. La unidad central de proceso (CPU)

En un sistema de cómputo, la unidad central de proceso (*central processor unit* o CPU) se fabrica como un circuito integrado de silicio que se encapsula y rodea de terminales (o pins) conectadas a las entradas y salidas eléctricas del circuito. El encapsulado se conoce como microprocesador o procesador. Hoy por hoy, los procesadores han logrado duplicar su desempeño cada 18 meses. Esto se conoce como la Ley de Moore, tras su exposición por el vicepresidente de Intel, Thomas Moore. Parece no haber indicaciones que esta tendencia vaya a detenerse.

3.2.2. Funcionamiento de la CPU

La CPU continuamente recibe instrucciones a ser ejecutadas. Su función es la ejecución secuencial de instrucciones. Su labor consiste mayormente en realizar operaciones y transportar datos. Más específicamente:

- decodificar instrucciones;
- recabar y escribir los datos en los que se opera;
- realizar las operaciones requeridas.

3.2.3. Tipos de datos

A la CPU se le alimenta datos binarios mediante el bus del sistema. Se reciben dos tipos de datos:

- instrucciones (en forma de código de operación) que instruyen al procesador qué hacer con los datos;
- datos, que deben ser operados de acuerdo con las instrucciones.

Una **instrucción** es una operación elemental en un lenguaje de programación. Es la más pequeña orden que un programador puede dar a una computadora. La forma como diferentes operaciones se codifican en números binarios se llama **formato de instrucción**.

Cualquier instrucción contiene dos tipos de información:

- qué tiene que hacer la propia instrucción (por ejemplo, sumar, restar, almacenar datos, imprimir) llamado código de operación u *opcode*;
- con qué datos debe operar, llamado campo de datos.

El formato en el que las instrucciones se almacenan internamente se conoce como código de máquina o lenguaje de máquina. Cada microprocesador tiene su propio lenguaje de máquina. Un ejemplo del formato de una instrucción en código de máquina se muestra en la Figura 3.2.

En este ejemplo en particular:

- el campo **OPCODE** especifica la operación;
- el campo **Address** designa la dirección de memoria RAM del primer operando de la instrucción;
- el campo **Register** especifica un operando en un registro;
- el campo **Mode** selecciona de los varios tipos de operandos los que el lenguaje de máquina permita.

OPCODE	Mode	Register	Address

Figura 3.2: Ejemplo del formato de una instrucción.

3.2.4. *Software* del sistema

Los lenguajes de alto nivel hacen la labor del programador más fácil y dan una correspondencia más cercana entre los objetos y operaciones de el lenguaje y aquéllos de la aplicación. Para que la máquina puede ejecutar un programa hecho en un lenguaje de alto nivel (como C o Java), el programa original o fuente debe convertirse en un formato que la máquina pueda ejecutar antes de que la propia ejecución comience. Las dos opciones de conversión son:

- compilación (o traducción), en la cual el programa en lenguaje de alto nivel se compila a una forma de código de máquina, que se almacena y ejecuta cuando se requiera;
- interpretación, en el cual un programa en código de máquina se usa para leer las instrucciones en lenguaje de alto nivel y realizar las operaciones que se especifican conforme se van leyendo.

Desde el punto de vista del hardware, los principales componentes del procesador son: la unidad aritmético lógica (*arithmetic-logic unit* o ALU), la unidad de control (*control unit* o CU) y los registros.

La **unidad aritmético lógica (ALU)** realiza las operaciones aritméticas y lógicas sobre los datos.

Un sumador completo puede extenderse para realizar las operaciones de sustracción, multiplicación y división de enteros, así como realizar operaciones lógicas bit a bit, lo que constituye a la unidad aritmético lógica (ALU). En una procesador moderno, esta sección de la computadora utiliza una porción pequeña del circuito integrado del procesador (Figura 3.3).

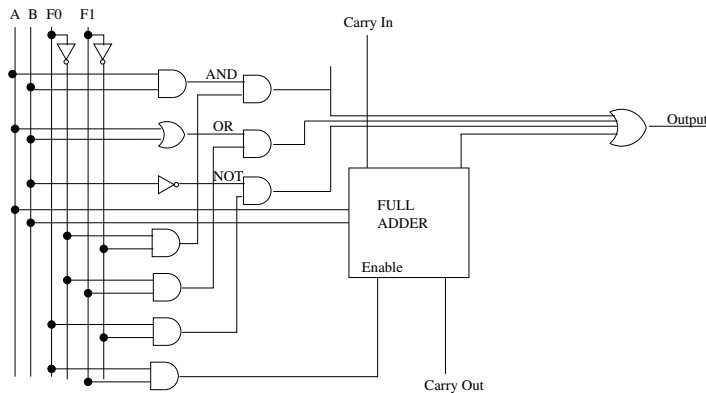


Figura 3.3: Una ALU simple de cuatro funciones controladas por dos líneas. Las funciones son $A \text{ AND } B$, $A \text{ OR } B$, $\text{NOT } B$ y $A + B$.

La ALU se comunica con la memoria de la computadora donde se almacenan los operandos. Para acelerar la operación, algunas computadoras tienen varias localidades de memoria más rápidas (entre 8 y 128, dependiendo de la arquitectura de la computadora), llamadas registros, dentro del propio procesador.

Dentro del sistema de cómputo, la operación de la ALU se dirige por la unidad de control. Las operaciones de la ALU ocurren cuando se le provee de la secuencia correcta de entradas desde la unidad de control. La operación actual puede depender del resultado de una operación previa, lo que deriva en el uso de un registro de banderas o registro de status, que almacena información acerca del resultado de la última operación de la ALU, por ejemplo, si el resultado ha sido cero, negativo, o ha producido un acarreo (Figura 3.4).

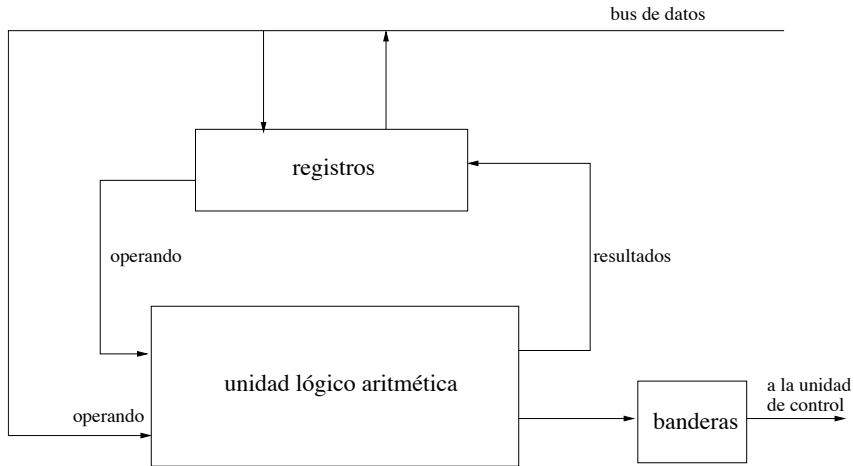


Figura 3.4: Relación entre la ALU, los registros y el registro de banderas durante una operación.

Un diagrama simple de una ALU se muestra en la Figura 3.5.

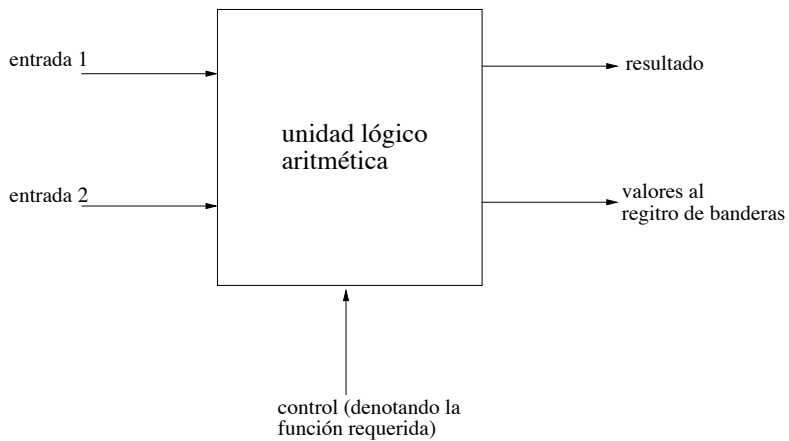


Figura 3.5: Diagrama de bloques de una ALU.

La entrada y salida de la ALU normalmente puede ser de 8, 16, o 32 bits. La señal de control consiste de un número de bits, dependiendo de el número de operaciones que la ALU sea capaz de realizar, las cuales pueden incluir:

- adición/sustracción;
- multiplicación/división (solo en grandes sistemas de cómputo);
- pruebas lógicas (por ejemplo, probar si el resultado de una operación es cero);
- prueba lógica por cero (prueba si un operando tiene todos sus bits en cero);
- AND bit a bit de dos operandos;
- OR bit a bit de dos operandos;
- corrimiento de bits;
- comparaciones ($>$, $<$, $=$).

La **unidad de control** realiza la ejecución de programas. Secuencialmente, trae cada instrucción a ejecutar de la memoria, la decodifica, y genera las señales digitales hacia varios de los elementos del sistema para llevar a cabo la secuencia de operaciones necesarias para realizar la operación.

Una vez que una instrucción ha sido ejecutada, la unidad de control busca la siguiente instrucción a ejecutar. Un diagrama simplificado de una unidad de control se muestra en la Figura 3.6.

El papel de varios componentes de la unidad de control se describen a continuación:

- el registro de instrucción (IR) almacena la instrucción a ejecutarse;
- el decodificador decodifica la instrucción, enviando esta información al secuenciador;

- el secuenciador emite las señales de control apropiadas a las otras unidades del sistema de cómputo. La temporización la determina por un reloj interno del sistema (actualmente, los procesadores utilizan relojes de cuarzo con frecuencias mayores a 1.0 GHz. A mayor velocidad de reloj, más rápido es el procesador);
- el registro de banderas provee al secuenciador de los detalles sobre los resultados de previas operaciones;
- El contador de programa (PC) almacena la dirección en memoria de la **siguiente instrucción** a ejecutar, y normalmente se incrementa conforme cada instrucción se procesa.

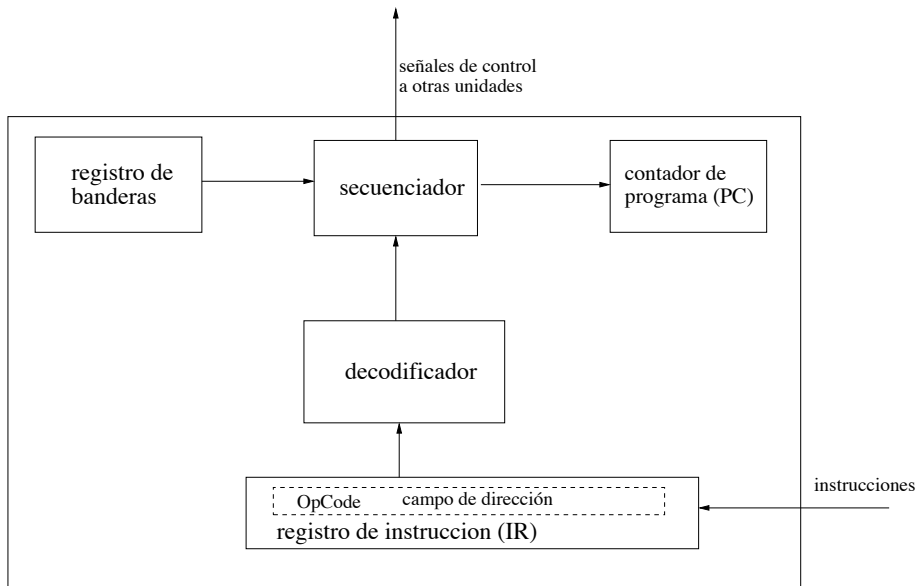


Figura 3.6: Diagrama de bloques de una unidad de control.

3.3. Buses, memoria y dispositivos de entrada/salida

Esta sección se discuten los otros tres componentes esenciales de un sistema de cómputo: los *buses*, la memoria y los dispositivos de entrada/salida.

Aun cuando sólo se muestran dos líneas conectando los varios componentes de una computadora digital en la Figura 3.1, en realidad tales conexiones se realizan mediante un número de pequeños alambres que van de las terminales de un componente a las terminales de otro. Esta colección de alambres se conoce con el nombre de *bus*. Supóngase que la computadora usa palabras de 16 bits. Es necesario entonces que exista un alambre en el *bus* por cada bit de la palabra. En la sección 4.1 se explica que debe haber una línea de entrada por cada dirección binaria utilizada por la memoria. Por ejemplo, en la Figura 4.4 se tienen dos terminales de entrada para las direcciones de una memoria de cuatro palabras. Además, debe haber una terminal de selección del circuito, y finalmente, terminales que conduzcan las señales de lectura y escritura en memoria. Todas estas líneas constituyen conexiones adicionales en el *bus*.

Anteriormente se han discutido el uso por separado de líneas de control para cada operación de la ALU. Cada una de estas líneas de control deben también incluirse en el *bus*, además de otras líneas de control, como por ejemplo, conexiones para la señal de reloj. Ya que los circuitos de la computadora utilizan corriente directa, el *bus* debe contener también líneas de alimentación eléctrica, normalmente de 5 volts (marcado como V_{cc}) y 0 volts (llamado tierra o GND). Finalmente, para permitir la expansión de la computadora, como por ejemplo incrementar la capacidad de memoria, el *bus* debe contener líneas extra.

Parecería que no se requiere que todas las líneas de conexión pasen por todos los componentes de la computadora; sin embargo, para evitar problemas de conexión entre ellos, se hace que todo componente tenga toda conexión disponible, aun cuando no requiera utilizarla. De hecho, normalmente en la actualidad el *bus* de casi todas las computadoras, pero especialmente en las computadoras pequeñas, consiste en un conjunto de alambres de metal impresos en una tarjeta plástica llamada tarjeta madre (*motherboard*). Esta tarjeta provee de acceso a todas las terminales de la computadora, ya que se pueden colocar conectores a ella. Por ejemplo, nuevos componentes pueden añadirse mediante conectarlos al *bus*.

3.3.1. Los *buses*

Los *buses* permiten la transmisión de datos entre los diferentes componentes de una computadora. Los *buses* conectan al procesador con todos los componentes. El procesador a su vez recibe y envía datos de y hacia sus *buses*, los cuales son de dos tipos:

- el *bus* del sistema, que conecta al procesador con la memoria RAM;
- los *buses* de entrada/salida, que conectan al procesador con otros componentes.

El *bus* del sistema se conecta con los *buses* de entrada/salida, el procesador con la RAM, como se muestra en la Figura 3.7 (ésta es una sobreesimplificación, ya que los *buses* modernos son mucho más complejos). El *bus* se diseña para el tipo específico de procesador: la tecnología del procesador determina la dimensión del *bus* del sistema, qué tan rápido es, y las señales de control que utiliza.

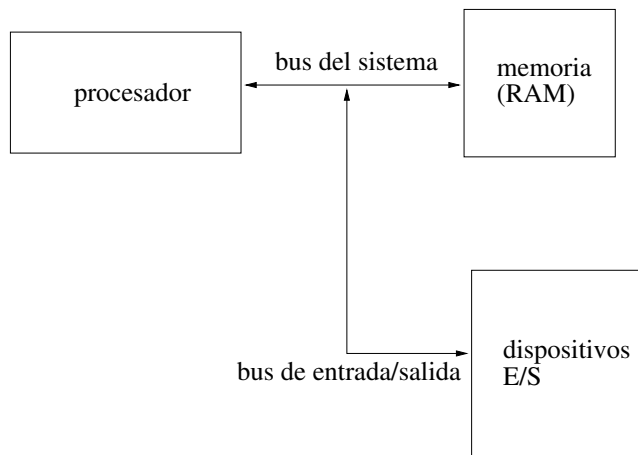


Figura 3.7: *Buses* del sistema y de entrada/salida.

Los *buses* de entrada/salida se usan para transmitir datos, ya que conectan a todos los dispositivos de entrada/salida con el *bus* del sistema, y por lo tanto, con el procesador y la RAM. Los dispositivos de entrada/salida son aquellos componentes que pueden recibir y enviar datos al exterior de la computadora. Puede haber más

de un *bus* de entrada/salida en una computadora, y pueden estar estandarizados a fin de permitir la adición fácil de componentes.

Un *bus* puede dividirse en tres partes principales:

- el *bus* de datos, que lleva propiamente datos, y puede consistir de 8 a 64 alambres, cada uno de los cuales conduce un bit del dato. Al número de alambres se le conoce como ancho de *bus*;
- el *bus* de direcciones, que son un número de alambres que conducen la dirección de memoria del dato a ser accedido. Este *bus* es usualmente unidireccional, es decir, la dirección va del procesador a los demás componentes;
- al *bus* de control, que provee de información acerca de cómo los datos en el *bus* deben transferirse.

Una convención para representar un *bus* es utilizar una sola línea con una diagonal para indicar que en realidad hay un número de alambres paralelos.

Ya que varias unidades dentro del sistema de cómputo comparten las mismas líneas de *bus*, se requieren procedimientos de interfaz para los módulos del *bus* para asegurar que, por ejemplo, dos módulos no intenten escribir datos en el mismo *bus* al mismo tiempo, así como para identificar para qué módulo se envían los datos.

Considérese dos registros de cuatro bits conectados mediante un *bus* de datos (Figura 3.8). Esta simple configuración ya genera conflictos en el *bus*. Las salidas de compuertas ordinarias no deben conectarse juntas. Si, por ejemplo, la salida Q_0 del registro R_0 tiene un valor de 1, mientras que la salida Q_0 del registro R_1 es 0, y dado que se conectan al mismo alambre, se provoca un corto circuito, provocando un daño a los circuitos, además que el resultado es un estado indeterminado en el nivel lógico.

Una solución al conflicto es conectar solo las líneas de salida de un registro al *bus* de datos en un momento dado. Esto se logra mediante insertar un tipo especial de circuito *buffer*, conocido como compuerta triestado. Esta compuerta contiene un interruptor interconstruido, que permite desconectar la salida del registro con la conexión de salida, para cada línea de salida del registro. Este *buffer* tiene la siguiente tabla de verdad, cuyas entradas y salidas corresponden a la Figura 3.9:

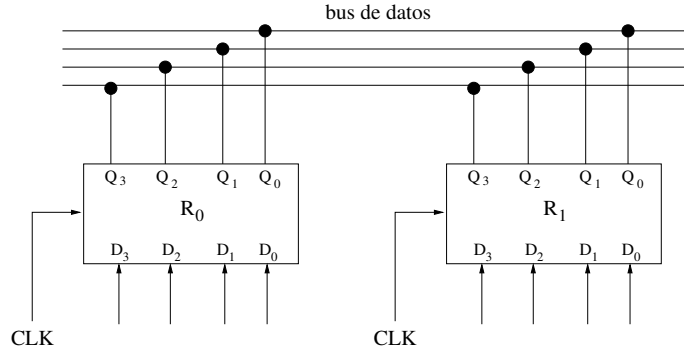


Figura 3.8: Dos registros conectados mediante un *bus* de datos.

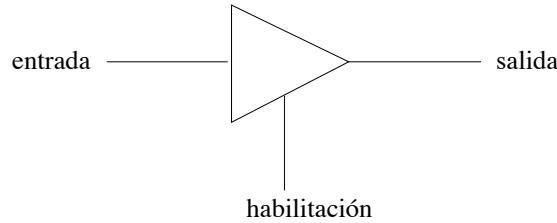


Figura 3.9: Símbolo del circuito triestado.

Entrada	Habilitación	Salida
0	0	No conectada
0	1	0
1	0	No conectada
1	1	1

Si la línea de habilitación está activa (1), entonces el triestado se comporta como un *buffer* ordinario: el interruptor se cierra, y la salida tiene el mismo nivel lógico que la entrada; sin embargo, si la línea de habilitación no está activa (0), entonces el interruptor está abierto, y efectivamente no hay conexión entre las líneas de salida del registro y el *bus* de datos. A este estado se le conoce como de alta impedancia. De tal modo, si cada línea de salida de un registro tiene

una compuerta triestado, se puede controlar tal salida con una sola señal lógica conectada a todas las habilitaciones. Y solo una línea de control de habilitación debe estar activa para un solo registro en todo momento.

3.3.2. Organización y capacidad de la memoria

La **memoria** es un componente esencial para todo sistema de cómputo, ya que en ella se almacenan los programas y datos. Una unidad de memoria es una colección de celdas de almacenamiento binario junto con otros circuitos asociados, que se requiere para transferir datos binarios desde y hacia la propia memoria.

La memoria almacena información binaria en grupos de bits llamados palabras. Una palabra en memoria es el paquete máximo de información que puede moverse como una unidad desde y hacia el almacenamiento. Típicamente, se subdivide en grupos de 8 bits, conocidos como bytes. Cada palabra tiene el mismo tamaño (entre 16 y 64 bits), independientemente del tipo de datos que contiene. Una palabra de memoria o byte puede representar un número, un código de operación, uno o más caracteres alfanuméricos, o cualquier otra información binaria codificada.

La estructura interna de la unidad de memoria se determina por el número de palabras que contiene y el número de bit de cada palabra. Las líneas de entrada, llamadas líneas de dirección, se conectan a decodificadores internos para seleccionar una palabra en particular. La identificación única para cada palabra de memoria, la dirección, comienza desde 0 y termina hasta el valor $(2^m - 1)$, en donde m es el número de líneas de dirección de la memoria.

En cada palabra, los bits se numeran de derecha a izquierda, es decir, el bit más significativo (*Most Significant Bit* o MSB) es el que se encuentra más a la izquierda, mientras que el bit menos significativo (*Least Significant Bit* o LSB) es el bit más a la derecha. Se debe distinguir cuidadosamente entre la dirección de una localidad de almacenamiento y su contenido.

La capacidad de una particular memoria de computadora, es decir, el número de palabras disponibles, se llama tamaño de memoria, mientras que el número de direcciones en la misma se refiere al espacio de direcciones. Si m bits se utilizan para una dirección binaria, entonces el espacio de direcciones es 2^m , y las

direcciones van desde 0 hasta $(2^m - 1)$. Ya que el espacio de direcciones resulta normalmente bastante grande, se expresa comúnmente en kilobytes ($1\text{kB} = 2^{10}$ Bytes = 1024 Bytes), megabytes ($1\text{MB} = 2^{20}$ Bytes = 1048576 Bytes) o gigabytes ($1\text{GB} = 2^{30}$ Bytes = 1073741824 Bytes).

Ejemplo 3.1 Una computadora con 256 kB de memoria tiene $2^{18} = 262144$ localidades direccionables.

3.3.3. La estructura básica de la memoria

Las operaciones básicas en una memoria son la lectura y escritura. Ya que las operaciones básicas se realizan por palabra, entonces la dirección de la palabra en cuestión debe presentarse primero, junto con las señales adecuadas de control de lectura o escritura. Las operaciones de lectura y escritura actúan sobre una sola palabra, y por tanto, una ruta de una palabra de ancho debe existir entre la memoria y los demás dispositivos.

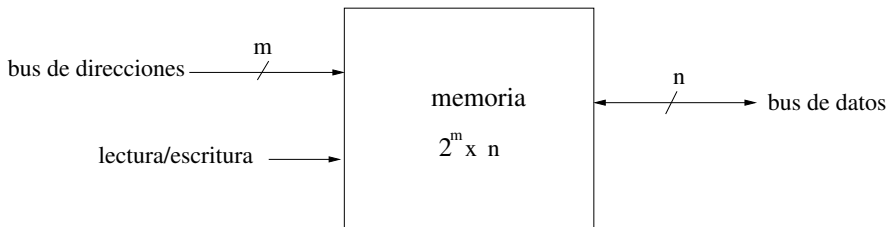


Figura 3.10: Estructura básica de una memoria.

Internamente, una memoria consiste de un gran arreglo de registros. Para una operación de lectura, los datos binarios se copian de un registro en particular al *bus* de datos cuando este registro se selecciona por el decodificador de direcciones y, simultáneamente, se aplica la señal de lectura. Similarmente, para una operación de escritura, se selecciona el registro por el decodificador de direcciones, y los datos binarios se copian del *bus* al registro direccionado cuando se presentan en el *bus* de datos y simultáneamente se presenta la señal de escritura.

Existen dos tipos principales de memoria semiconductora en los sistemas de cómputo digital: la memoria de acceso aleatorio (*random access memory* o RAM) y la memoria de solo lectura (*read only memory* o ROM).

En la **RAM**, las localidades de memoria pueden accederse para transferencia de datos binarios desde cualquier dirección que se desee. El proceso de localizar una palabra en esta memoria es idéntico, sin importar dónde se encuentre la localidad en la memoria física. Una misma cantidad de tiempo se requiere para localizar cualquier palabra. Ambas operaciones de lectura o escritura pueden efectuarse en RAM. En la **ROM**, las unidades de memoria solo permiten operaciones de lectura. Los datos binarios dentro de una ROM no pueden ser alterados por un programa. La ROM también tiene la característica de acceso aleatorio, y su contenido no se pierde cuando se le elimina la alimentación eléctrica.

Ejemplo 3.2 Los circuitos de RAM vienen en una variedad de tamaños. Un ejemplo de un circuito RAM se presenta en la Figura 3.11.

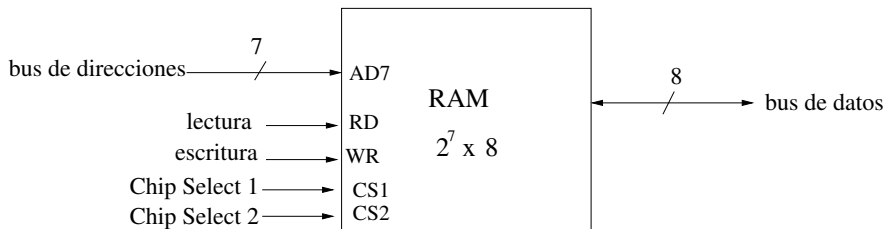


Figura 3.11: Una memoria RAM.

El circuito tiene 128 localidades de memoria, cada una manteniendo una palabra de 8 bits. Las localidades se direccionan con un *bus* de 7 bits de ancho (7 bits son necesarios para direccionar $128 = 2^7$ localidades). Esta memoria se comunica con el procesador mediante un *bus* de datos de 8 bits (que se ajusta a la longitud de palabra de cada localidad). La comunicación en este *bus* es bidireccional, ya que los datos pueden leerse de la memoria y enviarse al procesador, o escribirse en memoria provenientes del procesador, utilizando las mismas líneas de conexión. Las operaciones sobre la memoria se especifican mediante las dos líneas de entrada RD y WR. También hay dos líneas de entrada de control para cada circuito

RAM: *Chip Select 1* y *Chip Select 2*, que habilitan al circuito RAM cuando se le selecciona por el procesador. La razón de contar con dos líneas de control es que facilitan la decodificación de las líneas de dirección cuando existen múltiples circuitos RAM en la computadora.

3.3.4. Dispositivos de entrada/salida

Ninguna descripción de una computadora está completa sin considerar un rango de dispositivos de entrada/salida o periféricos. Los programas y los datos deben introducirse a la memoria de la computadora, y los resultados obtenidos del procesamiento deben registrarse o desplegarse (o ambos) para el usuario. La entrada/salida involucra una parte muy grande de la computadora.

Ejemplos comunes de dispositivos de entrada/salida son: el teclado, las impresoras, el *mouse* (o ratón), la pantalla, y otros. Los dispositivos periféricos propiamente se discuten en el Capítulo 6. Aquí propiamente se describe la comunicación y transferencia de datos entre ellos y el procesador. Como los dispositivos periféricos son frecuentemente electromecánicos, y el procesador y la memoria son semiconductores, lograr una interfaz entre ellos es necesario para que se comuniquen, dado que el procesador y los dispositivos de entrada/salida funcionan a diferentes velocidades, de modo que se requiere un mecanismo de sincronización.

Entre el procesador y los dispositivos de entrada/salida se coloca una electrónica digital especializada. Además, cada dispositivo cuenta con su propio controlador para monitorear y controlar su operación (por lo que frecuentemente se les llama dispositivos inteligentes). El procesador se comunica con la memoria y los dispositivos de entrada/salida de la misma manera, transfiriendo datos a través de un *bus*, aunque es común que haya un *bus* dedicado para los dispositivos de entrada/salida, como se muestra en la Figura 3.12.

- un dispositivo de entrada/salida particular se selecciona mediante colocar su dirección en las líneas de dirección del *bus* de entrada/salida;
- una vez que el dispositivo ha identificado su dirección en el *bus*, se activa;
- la interfaz del dispositivo lee el código de función que provee el procesador en las líneas de control del *bus* de entrada/salida, y responde al mismo.

Hay cuatro tipos de códigos de función que la interfaz puede recibir y ejecutar:

- instrucción de control: se envía para activar el dispositivo e informarle qué hacer;
- estado: se utiliza para probar varias condiciones de estado en la interfaz y el dispositivo;
- datos de salida: causa que la interfaz responda transfiriendo datos de su *bus* al registro de salida de datos. La salida de este registro se conecta con las líneas de datos del dispositivo de entrada/salida;
- datos de entrada: causa que la interfaz reciba datos del dispositivo de entrada/salida y los coloque en su registro de entrada.

Los registros dentro de la interfaz se les conoce como registros o puertos de entrada/salida.

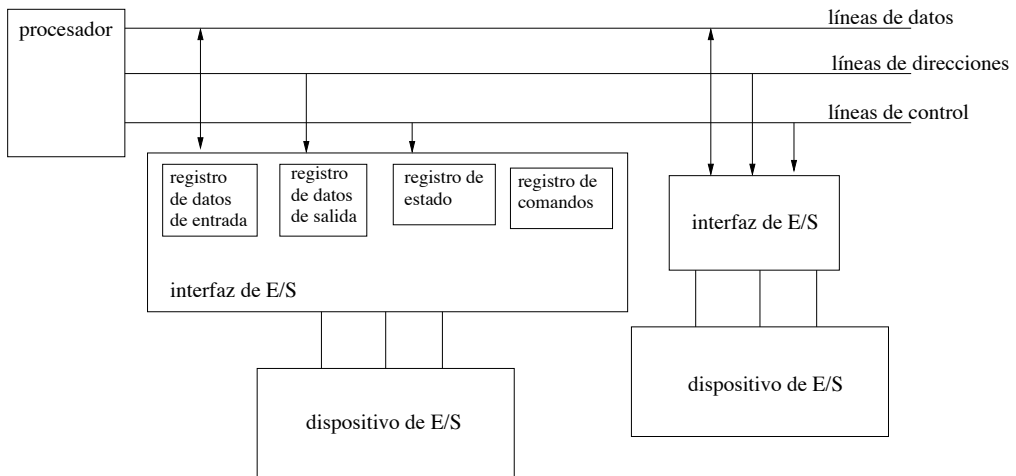


Figura 3.12: Un sistema de *buses* para los dispositivos de entrada/salida.

3.4. Instrucciones y secuenciación

Esta breve sección explica el ciclo ejecución de instrucciones en un sistema de cómputo.

Las operaciones que la ALU es capaz de realizar sobre valores binarios, tomándolas en la forma secuencial como instrucciones en lenguaje de máquina en un programa, dirigen la operación de una computadora. Por lo pronto, en esta sección se discuten los detalles de cómo las instrucciones se almacenan en la computadora, en particular las instrucciones específicas que se utilizan para controlar a la memoria.

Primeramente, se considera cómo se direcciona la memoria. La dirección de la palabra a leer o escribir de la memoria se presenta como un número binario. Hay una terminal por cada dígito binario de la dirección. Para direccionar la memoria, las señales apropiadas deben colocarse en las líneas de dirección del *bus*. Estas señales se proveen de un registro especialmente diseñado para almacenar, en binario, las direcciones de memoria. A este registro se le conoce como registro de dirección de memoria (*memory address register* o MAR). El MAR, entonces, normalmente contiene el valor binario de la dirección de la palabra que se lee o escribe de la memoria. En la Figura 4.4 se muestran un par de flip-flops que, considerados en arreglo, forman un registro que puede ser utilizado como MAR.

Otro registro asociado con la memoria principal es el registro separador de memoria (*memory buffer register* o MBR). Este registro tiene la función de almacenar la palabra que se lee de o se escribe a la memoria.

Considerando ambos registros MAR y MBR, se requieren apenas unas cuantas instrucciones para manejar la memoria:

1. Escribe la palabra almacenada en el MBR a la memoria principal, en la dirección almacenada en el MAR.
2. Lee la palabra almacenada en la dirección almacenada en el MAR, y colócala en el MBR.
3. Limpiar el MAR o copiar una palabra al MAR.

A veces, en computadoras sencillas, estas instrucciones se combinan con otras. Por ejemplo, se discute más adelante una instrucción para almacenar los contenidos del acumulador en una localidad de memoria que se especifica en la instrucción misma. En realidad, tal instrucción sola puede tomar el lugar de varias otras instrucciones, una para almacenar los contenidos del acumulador en el MBR, y entonces otra para almacenar los contenidos del MBR en una localidad específica de memoria; sin embargo, esta secuencia de instrucciones podría ser automáticamente-

te generada por la unidad de control. De hecho, algunas computadoras antiguas no cuentan con un MBR, por lo que utilizan otros registros, como el acumulador.

3.4.1. Estructura de la palabra

Al considerar la estructura de las palabras almacenadas en la memoria, recuérdese que hay dos tipos de información binaria almacenada en la memoria: instrucciones y datos. Los datos pueden ser un valor numérico, que puede ser de punto flotante o un número sin exponente; o puede ser un código para información alfanumérica.

Cada palabra almacenada en la memoria que representa una instrucción normalmente contiene dos partes: un código de instrucción y una dirección de memoria; sin embargo, en algunas computadoras, se utilizan palabras más complejas, conteniendo más información. La instrucción propiamente dicha es el código de la instrucción. De hecho, cada instrucción tiene un código binario que la representa. Por ejemplo, 000001 podría indicar que los contenidos de una dirección de memoria dada deben sumarse con el contenido del acumulador.

Para ilustrar esto, se trabaja con una palabra de 16 bits, como la que se muestra en la Figura 3.13

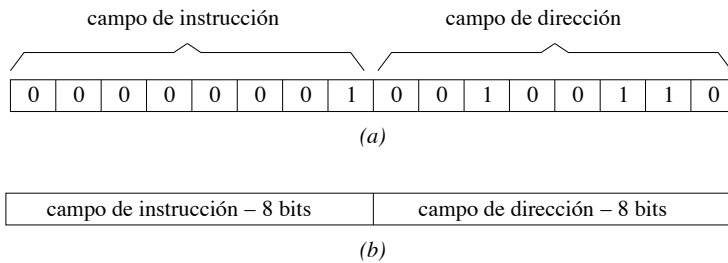


Figura 3.13: (a) Una palabra de 16 bits con un campo de instrucción de 8 bits y un campo de dirección de 8 bits; (b) una representación más simple de esta palabra.

La palabra se divide en dos subpalabras, llamadas campos (*fields*). Un campo se utiliza para el código de instrucción (*operational code* u *op-code*), y el otro para la dirección de memoria. Considérese la palabra específica que se muestra en la Figura 3.13:

00000001 00100110₂

Los primeros 8 bits representan el código de la instrucción, y los 8 bits finales representan una dirección de memoria. Por ejemplo, los primeros 8 bits (00000001_2) pueden indicar a la computadora para sumar los contenidos de la dirección de memoria con los contenidos del acumulador, mientras que los últimos 8 bits (00100110_2) especifican una dirección de memoria. Por lo tanto, la palabra 0000000100100110_2 significa sumar el número binario contenido en la dirección 00100110_2 con el número binario almacenado previamente en el acumulador.

La palabra 0000000100100110_2 es un ejemplo de un enunciado (*statement*) de un programa en lenguaje de máquina, un programa escrito en binario que dirige la operación de la computadora. Para eliminar posibles confusiones o errores al utilizar números binarios, se acostumbra escribir el programa en su equivalente hexadecimal, como por ejemplo:

	campo de instrucción	campo de dirección
binario	00000001	00100110
hexadecimal	01	26

La representación hexadecimal es más fácil de manejar, y muchos programas en lenguaje de máquina se escriben en hexadecimal; sin embargo, el programa real se encuentra, por supuesto, en binario.

Hasta aquí, se discute una palabra que contiene un campo de instrucción y un campo de dirección. La estructura de esta palabra se utiliza para dirigir a la computadora para realizar una operación, posiblemente en conjunción con una dirección de memoria. En el ejemplo anterior, la instrucción causa que el contenido de la dirección de memoria dada se sume con el contenido del acumulador. Otra instrucción puede utilizarse, igualmente, para almacenar el contenido del acumulador en una dirección de memoria específica.

En los ejemplos subsecuentes se utiliza esta estructura de palabra dado que es simple. Esto elimina una gran cantidad de detalles innecesarios en la discusión; sin embargo, pueden utilizarse otros tipos de estructuras de palabra. Por ejemplo, algunas computadoras utilizan palabras con dos campos de dirección. Suponiendo que se desea transferir el contenido de una dirección de memoria a otra dirección de memoria, una palabra con dos campos de dirección podría ser muy conveniente para expresar esto en una sola instrucción.

Además, existen otras formas de instrucciones para las computadoras. Muchas computadoras utilizan 8, 16, 32 o hasta 64 bits. Si 8 o más bits se utilizan para

especificar una dirección en memoria, entonces una palabra de 8 bits no puede proveer espacio para un campo de instrucción (*opcode*) y un campo de dirección. De tal modo, muchas computadoras utilizan secuencias de palabras para desarrollar instrucciones. Por ejemplo, supóngase que hay dos palabras en la secuencia. La primera palabra puede proveer del código de operación, mientras que la segunda puede especificar la localidad de memoria. De hecho, muchas computadoras utilizan 16 bits para especificar una dirección de memoria, con lo que se tiene que pueden direccionarse hasta 65,536 localidades de memoria. En este caso, una instrucción se compone de tres palabras de 8 bits: una para el código de operación, y dos palabras para la dirección de memoria.

Al programar una computadora, la estructura de sus instrucciones debe ser conocida previamente. Normalmente, los manuales de instrucción del procesador de la computadora contienen muchos de estos detalles.

3.4.2. El ciclo de instrucción

Para los propósitos de esta discusión, se considera un procesador sencillo, como se muestra en la Figura 3.14.

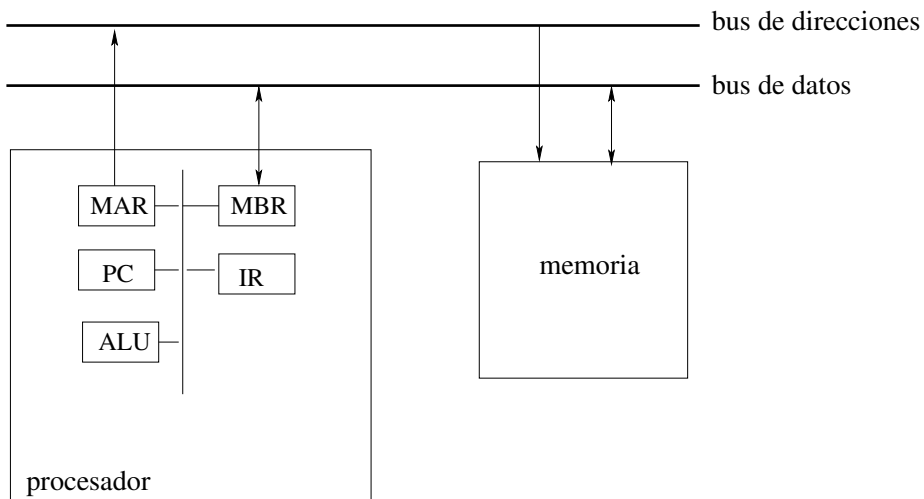


Figura 3.14: Un sistema sencillo.

El ciclo de instrucción, es decir, la secuencia de pasos a realizar por cada instrucción en memoria, se divide en dos partes: ciclo de búsqueda (*fetch*) y ciclo de ejecución.

1. Ciclo de búsqueda:

- a) buscar la siguiente instrucción en la memoria y traerla a la unidad de control (al IR);
- b) decodificar la instrucción.

2. Ciclo de ejecución:

- a) buscar los operandos y procesar la instrucción;
- b) escritura del resultado.

El ciclo de búsqueda es idéntico para todas las instrucciones de la computadora. El ciclo de ejecución depende de la instrucción particular que se ejecuta. Por tanto, hay tantos diferentes ciclos de ejecución como instrucciones definidas para una computadora en particular. Las instrucciones típicamente caen en varios grupos:

- operaciones aritméticas y lógicas (suma, resta, multiplicación, división, AND, OR, XOR, etc.);
- operaciones de transferencia o copia, que permiten mover datos binarios entre localidades de memoria y entre memoria y dispositivos de entrada/salida;
- operaciones de control, que modifican el flujo de instrucciones a lo largo del programa, dependiendo del resultado de una operación previa, forzando que la siguiente instrucción se obtenga de alguna otra localidad de la memoria.

Las señales de temporización para realizar los pasos dentro del ciclo de ejecución de instrucciones se originan de la unidad de control del procesador.

3.4.3. El ciclo de búsqueda

Para que una instrucción se busque en la memoria, se realizan las siguientes operaciones:

1. Cargar el contenido del contador de programa (*Program Counter* o PC) en el registro de direcciones de memoria (*Memory Address Register* o MAR), que lo coloca en el *bus* de direcciones de la computadora.
2. Se envía una señal de lectura a la memoria, lo que resulta en colocar el código de operación (*Operation Code* u *opcode*) en el *bus* de datos de la computadora.
3. Se almacena el *opcode* en el registro *buffer* de memoria (*Memory Buffer Register* o MBR) del procesador.
4. Se transfiere el *opcode* del MBR al registro de instrucción (*Instruction Register* o IR).
5. Comúnmente, se incrementa el contenido del PC.

3.4.4. El ciclo de ejecución

El ciclo de ejecución difiere de instrucción a instrucción. Por ejemplo, considere la instrucción ADD que permite sumar el contenido de un registro al acumulador, dejando el resultado en el propio acumulador. Ya que los registros se encuentran internamente en el procesador, la secuencia de ejecución podría ser:

- transferir los contenidos del acumulador a la unidad lógico aritmética (*Arithmetic Logic Unit* o ALU) mediante el *bus* interno;
- enviar señales a la ALU para realizar la suma;
- almacenar el resultado en el acumulador.

3.5. Ejecución de instrucciones

Para ilustrar la ejecución de instrucciones, a continuación se escribe un programa simple en lenguaje de máquina, especificando algunas instrucciones adicionales. Para fines didácticos, se utilizan palabras de 16 bits. Las instrucciones son:

- sumar los contenidos de una dirección en memoria con el acumulador

$$0000\ 0001_2 = 01_{16};$$

- limpiar el acumulador

$$0000\ 0010_2 = 02_{16};$$

- transferir el contenido del acumulador a una dirección dada de memoria

$$0001\ 0100_2 = 14_{16};$$

- detener el cómputo

$$0111\ 0111_2 = 77_{16}.$$

Con estas instrucciones, se escribe un programa sumamente simple. Se supone que la computadora ejecuta las instrucciones en orden secuencial, es decir, ejecuta la instrucción en la primera dirección de memoria, luego la instrucción en la segunda dirección, etc. Para este programa, se listan tanto las instrucciones como la dirección de memoria donde se encuentran almacenadas. Se supone que ya se encuentran en la memoria de trabajo de la computadora. Se presentan tanto la notación binaria como la notación hexadecimal.

dirección		palabra	
binario	hexadecimal	binario	hexadecimal
00000001	01	0000001000000000	0200
00000010	02	0000000100010000	0110
00000011	03	0000000100010001	0111
00000100	04	0001010000010010	1412
00000101	05	0111011100000000	7700
00010000	10	0000000000000011	0003
00010001	11	0000000000000100	0004

Al operar este programa, recuérdese que los primeros 8 bits de la instrucción especifican el código de operación de la instrucción, y que los últimos 8 bits especifican la dirección en memoria. La primera instrucción es:

$$0200_{16}$$

Esta es la instrucción para limpiar el acumulador. El valor en el campo para la dirección de memoria no tiene significado aquí, y los dos últimos dígitos hexadecimales se ignoran por parte de la computadora. De hecho, podría utilizarse cualquier valor en esas posiciones.

La siguiente instrucción es:

$$0110_{16}$$

Los primeros 8 bits (01_{16}) indican que el número almacenado en la dirección de memoria 10_{16} debe sumarse con el contenido del acumulador. El número almacenado en la dirección 10_{16} es:

$$00000000000000011_2 = 0003_{16}$$

Por lo tanto, después de que esta instrucción se ejecuta, el valor 0003_{16} se coloca en el acumulador (almacenado, por supuesto, en binario).

La siguiente instrucción es:

$$0111_{16}$$

Esta instrucción especifica que el contenido de la dirección de memoria 11_{16} deben sumarse con el contenido del acumulador. El número almacenado en la dirección 11_{16} es:

$$00000000000000100_2 = 0004_{16}$$

Al finalizar la ejecución de esta instrucción, el número almacenado en el acumulador es el resultado de la suma del número previamente almacenado (0003_{16}) más 0004_{16} . La operación es como sigue:

$$0003_{16} + 0004_{16} = 0007_{16}$$

La siguiente instrucción es:

$$1412_{16}$$

Los primeros 8 bits (14_{16}) indican que el contenido del acumulador debe almacenarse en la dirección de memoria 12_{16} . Tras la ejecución de esta instrucción, la palabra almacenada en la dirección de memoria 12_{16} es:

$$00000000000000111_2 = 0007_{16}$$

Finalmente, se ejecuta la instrucción:

$$7700_{16}$$

Los primeros 8 bits (77_{16}) causan que el cómputo se detenga. El resultado de este simple programa es, entonces, sumar dos valores numéricos en diferentes localidades de memoria y escribir el resultado en una tercera localidad de la memoria.

En este simple programa no se toma en cuenta ninguna salida de datos. Esto se considera más adelante.

Por lo pronto, se consideran algunas operaciones de la computadora en un mayor detalle. El código de operación informa a la computadora para que realice alguna operación específica. Cuando la instrucción se ejecuta, a la ALU se le provee de las señales de control apropiadas. Esto se ejemplifica utilizando una computadora sencilla. Supóngase que los 8 bit más a la izquierda de la instrucción (el código de operación) se almacena en el registro de instrucción. El resto de los bits de la instrucción se consideran almacenados en el MAR. Ahora bien, tomando solo el contenido de IR, éste se da como entrada a un decodificador de instrucciones (*instruction decoder*). Este dispositivo convierte la información almacenada en IR al conjunto apropiado de señales de control. Un decodificador de instrucciones muy simplificado se muestra en la Figura 3.15. Este circuito puede decodificar un código de operación de la forma:

$$b_7b_6b_5b_4b_3b_2b_1b_0$$

Por ejemplo, si el código de operación es 00000001_2 , entonces la salida de la compuerta AND **a** toma el valor de 1, lo que hace que la señal de control ADD tome valor 1 y el resto de las señales de control tengan un valor de 0. Similarmente, si el número almacenado en IR es 00000010_2 , entonces sólo la salida de la compuerta AND **b** tiene valor 1, y la señal de control CLR se produce.

Cuando se ejecuta un programa, el número almacenado en cada localidad de memoria es una secuencia de ceros y unos. Tal número no provee de ninguna información que indique si se trata de un código de operación o un dato. Esto se indica solamente por el orden en que cada palabra se lee. Por ejemplo, supóngase que se lee 0110_{16} de una localidad de memoria, y que esto se trata como una instrucción. Tal instrucción indica que los contenidos de la localidad de memoria con la dirección 10_{16} se suman al número contenido en el acumulador. El contenido leído de la localidad con dirección 10_{16} es tratado, entonces, como un dato. De este modo, se considera que el contenido del primer byte representa un código de operación, mientras que el contenido de la dirección de memoria se considera un dato. En seguida se explica cómo la computadora puede ir siguiendo la secuencia de instrucciones.

La computadora cuenta con un registro de un solo bit (en realidad, un flip-flop) conocido como registro de búsqueda-ejecución (*fetch-execute register* o FER). Su función es indicar a la computadora si se encuentra buscando una instrucción de

la memoria o ejecutando una instrucción (durante lo cual la información que se extrae de la memoria son datos). Si el valor contenido en FER es 1, entonces la computadora considera que la información que se extrae de la memoria es una instrucción. Si el contenido de FER es 0, entonces la información tomada de la memoria se trata como dato. Antes de describir con mayor detalle la operación del FER, es necesario tomar en cuenta el contador de programa (PC), el cual contiene la dirección de la siguiente instrucción a ejecutar. Cada vez que una instrucción se ejecuta, se debe pasar a través de una secuencia de pasos. Además de los pasos propios para realizar la instrucción, hay pasos adicionales que llevan el orden secuencial de la ejecución, entre los cuales está incrementar el número (dirección) almacenado en el PC en una unidad.

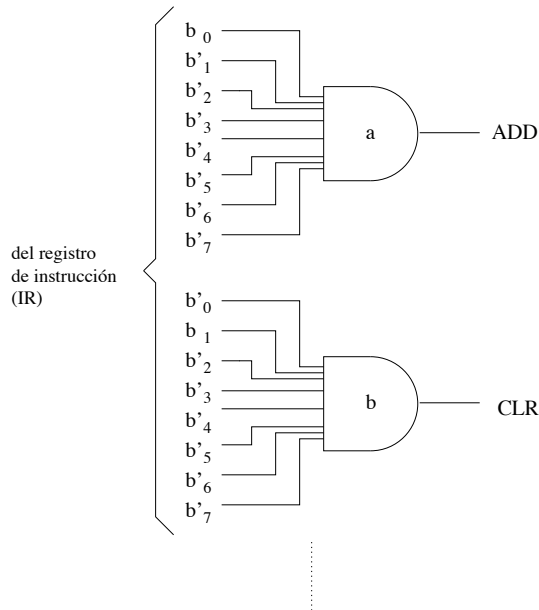


Figura 3.15: Parte de un decodificador de instrucciones.

Considérese ahora la ejecución de un programa. Cuando se inicia la operación, el contenido del FER tiene valor 1. Además, el contenido del PC toma el valor de la dirección del inicio del programa, por ejemplo 00000001_2 (se trabaja aquí con direcciones de memoria de 8 bits). Ya que el contenido del FER es 1, la información

binaria traída de la memoria es tratada como una instrucción. Como el contenido del PC es 00000001_2 , entonces se busca el contenido de la localidad de memoria con la dirección 01_{16} . Los 8 bits más significativos se colocan en el IR, mientras que los 8 bits menos significativos se ponen en el MAR. En este momento, la computadora puede ejecutar la instrucción, de modo que el contenido del FER toma el valor 0. Por otro lado, el contenido de IR se proporciona al decodificador, y así, se generan las señales de control apropiadas para la realización de la instrucción. La dirección almacenada en el MAR se lee, y la instrucción descrita por el código de operación almacenado en IR se lleva a cabo. Así, se ejecuta la instrucción. Al terminar la ejecución, el contenido del FER toma el valor de 1 de nuevo, y el contenido de PC se incrementa, de modo que toma el valor $00000010_2 = 02_{16}$. Ahora, se busca la información binaria contenida en la localidad de memoria con dirección 02_{16} , tratándola como instrucción. Y el proceso se repite una y otra vez por cada instrucción del programa. Nótese que la secuencia de pasos para la ejecución de instrucciones descrita aquí puede tener variaciones, como por ejemplo, el contenido del PC podría incrementarse mucho antes.

En la descripción del programa sencillo, el PC se ve afectado por una operación de incremento que aumenta su contenido binario en una unidad; sin embargo, las computadoras digitales cuentan con otras operaciones que afectan el contenido del PC, no necesariamente para solo incrementarlo, sino para cambiar su contenido completamente. Esto significa que el programa no necesariamente se ejecuta como una secuencia rígida de pasos, sino que puede incluir ciclos o selecciones, lo que aumenta grandemente la versatilidad de la programación de la computadora.

Recuérdese que la computadora no distingue la diferencia entre instrucciones y datos. Por ejemplo, supóngase que la tercera instrucción del programa se cambiara por:

$$0101_{16}$$

En tal caso, el contenido de la localidad de memoria con dirección 01_{16} , que es el código de operación de la instrucción para limpiar el acumulador, se suma al contenido del acumulador. Esto no tiene sentido, pero la computadora no tiene manera de interpretarlo. La computadora funciona como se le instruye. Así, dado que el valor en la dirección 01_{16} es 0200_{16} , tras la ejecución de esta instrucción el contenido del acumulador es:

$$0200_{16} + 0003_{16} = 0203_{16}$$

Después de la ejecución, este valor se almacena en la dirección 12_{16} .

En este ejemplo, se lee una instrucción y se utiliza como dato. Puede haber serias consecuencias por cometer un error así. Supóngase que en lugar de leer una instrucción, accidentalmente se escribe un dato en una localidad de memoria de una instrucción que todavía no ha sido ejecutada. Cuando se llega a la dirección de memoria de tal instrucción, muy probablemente no se encuentra un código de operación válido. En tal caso, el decodificador de instrucciones no genera ninguna señal de control. Esto hace que se detenga el cómputo, indicando un error difícil de detectar. Más aún, si por casualidad se tiene un código de operación válido almacenado, el resultado del cómputo no será el esperado, por lo que la situación de error persiste de una manera más sutil.

3.6. La computadora digital completa

En esta sección, y tras las descripciones y consideraciones anteriores, se puede discutir por fin una computadora digital completa. Se describe una computadora digital de 16 bits a la que se ha hecho referencia anteriormente, notando que las teorías de operación pueden aplicarse a una computadora con cualquier tamaño de palabra. La Figura 3.16 muestra un diagrama de bloques de una computadora muy pequeña y sencilla. Aun cuando se muestran sólo líneas interconectando los varios componentes de la computadora, recuérdese que tales líneas representan buses, compuestos de varios alambres individuales. En seguida, se discuten los varios componentes de la computadora. Solo se mencionan brevemente aquéllos que han sido discutidos previamente con mayor detalle.

La unidad lógico-aritmética (ALU) se relaciona con el registro acumulador, y se constituye de varios circuitos lógicos para obtener las operaciones deseadas. Se muestra también un registro adicional: el registro B. Este puede ser utilizado para almacenar algún número que se opera conjuntamente con el contenido del acumulador. En general, la ALU puede acceder otros registros, lo que le permite realizar operaciones más complejas, como multiplicación y división. Nótese que la salida del acumulador puede ser redirigida a la unidad de control, y a partir de aquí, a varias otras partes de la computadora, como podría ser la unidad de memoria.

La unidad de memoria (MU) contiene a la memoria principal. Ya que se suponen 8 líneas para el direccionamiento de memoria, se utiliza una pequeña memoria de 256 palabras. Las memorias de computadora, por lo general, utilizan muchísima

más memoria, por lo que ésta se hace disponible mediante multiplicar las líneas de direccionamiento; sin embargo, el tamaño de 256 palabras es sólo ilustrativo de este ejemplo de computadora sencilla. Por otro lado, el control de memoria se discute más a fondo en el Capítulo 6. Una dirección proveniente de la unidad de control se coloca en el MAR; si se da una instrucción de lectura de memoria, entonces el contenido en la localidad direccionada por el MAR se pasa al MBR; si se tiene una instrucción de escritura, entonces el contenido del MBR se copia a la localidad de memoria direccionada.

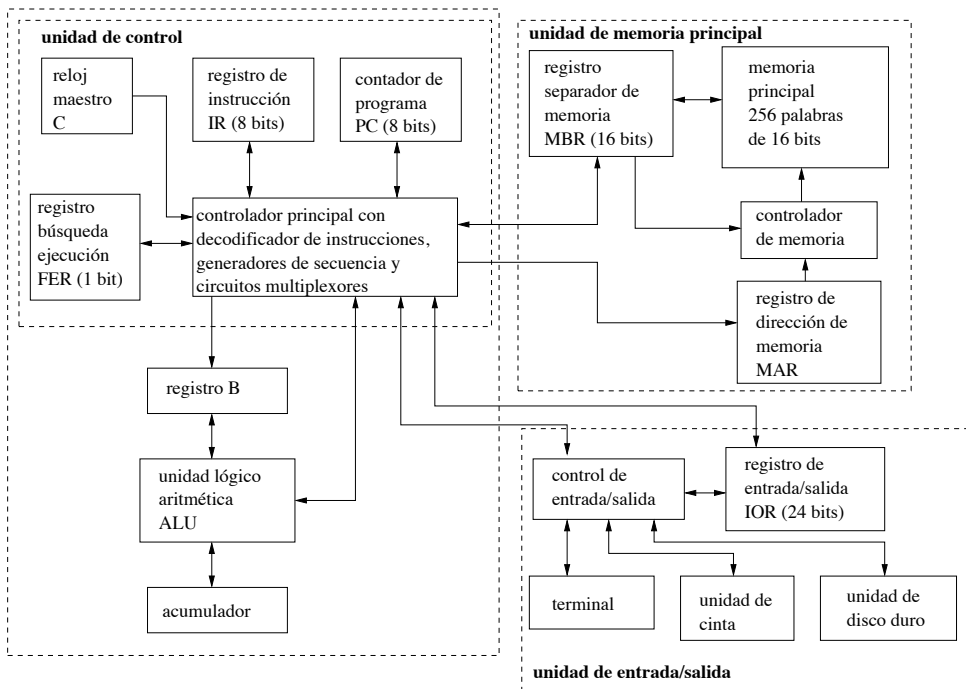


Figura 3.16: Diagrama de bloques de una pequeña computadora digital.

La unidad de control (CU) dirige la operación de toda la computadora. Contiene el decodificador de instrucciones (Figura 3.15), el cual traduce el código de operación relativo a la instrucción en señales de control que manipulan a la ALU, la memoria, los registros, y demás componentes de la computadora. También en

la unidad de control se encuentran el IR, el FER y el PC, que se discuten en la sección anterior.

El reloj maestro es también parte de la unidad de control. Previamente, se ha descrito cómo el reloj maestro mantiene la sincronización entre las diferentes partes de la computadora. En realidad, dentro de una computadora existen más de una secuencia de pulsos de reloj, debido a que ciertas partes de la computadora tienden a funcionar a velocidades muchos mayores que otras partes. En general, la memoria es mucho más lenta que la unidad central de proceso. Normalmente, se realiza una instrucción de lectura de una sola palabra de la memoria, seguida muchas otras operaciones que no involucran a la memoria. Sería un desperdicio de tiempo si todas las operaciones se realizaran a la velocidad de un ciclo de la memoria, y por tal razón, el reloj maestro normalmente se utiliza para generar varios conjuntos de pulsos a diferentes frecuencias. Por ejemplo, la Figura 3.17 muestra dos conjuntos de pulsos, unos más rápidos que otros; sin embargo, la relación entre ellos se mantiene constante. Así, C_1 podría utilizarse como reloj de las partes más veloces de la computadora, mientras que C_2 podría utilizarse para la memoria. Esto presupone, por supuesto, que hay varias operaciones de las partes más veloces de la computadora por cada ciclo de memoria. La unidad de control, por tanto, debe contener los circuitos necesarios para asegurar que la memoria no sea llamada antes de que las otras operaciones hayan concluido.

La unidad de entrada/salida contiene todos los dispositivos de entrada/salida. En el diagrama de bloques de la Figura 3.16 sólo se muestran una terminal, una unidad de cinta, y una unidad de disco duro. En realidad, puede haber un gran número de estos tipos de dispositivos u otros dispositivos, como impresoras, graficadores, escaners, etc. Más aun, pueden utilizarse otros dispositivos de entrada/salida diseñados para comunicar información más especializada, como sensores para medir la temperatura, y a la vez, actuadores que la computadora controla.

Aun cuando la Figura 3.16 solo muestra un registro asociado con la unidad de entrada/salida, tal unidad puede contar en realidad con una mayor capacidad de almacenamiento. Los dispositivos de entrada/salida son normalmente mucho más lentos que las demás partes de la computadora. Por ejemplo, la computadora puede producir datos binarios a una velocidad mucho mayor de que la pantalla puede presentarlos. Los datos, en este caso, se almacenan en una memoria del dispositivos de entrada/salida, conocida como *buffer*. De este modo, la pantalla toma los datos de su buffer, a su propio paso. De hecho, el buffer puede ser una sección de la RAM. Si no hubiera *buffer*, cada vez que se produjeran datos de salida, la computadora

tendría que hacer una pausa hasta que la pantalla terminara de presentar todos los datos. El uso del *buffer* de entrada/salida elimina la necesidad de muchas de este tipo de pausas.

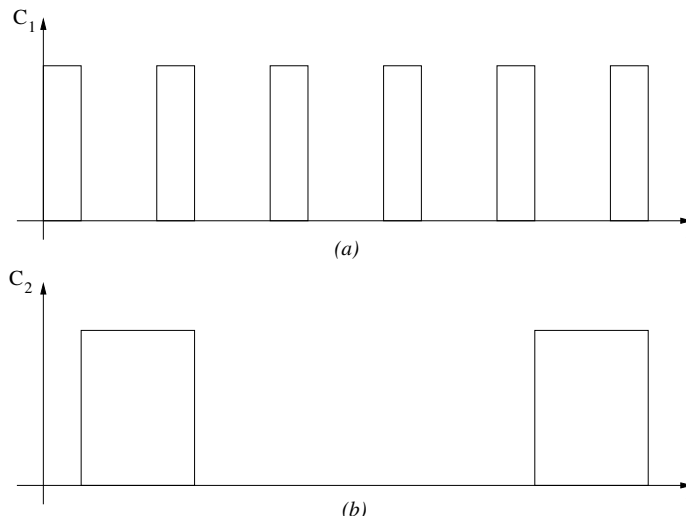


Figura 3.17: Pulsos de reloj (a) con temporización normal; (b) con temporización más lenta, para usarse con la memoria.

Cuando se coloca datos en el *buffer*, tales datos ocupan espacio de almacenamiento. Cuando los datos se leen del *buffer* y se despliegan por una pantalla, este espacio de almacenamiento se vacía, y el *buffer* puede recibir más datos. Si el *buffer* se llena, entonces no puede recibir más datos, y la computadora debe hacer una pausa.

3.7. Programación en lenguaje de máquina

En esta sección se presenta un pequeño lenguaje de máquina, y se muestra su utilización en el desarrollo de programas muy sencillos. Los lenguajes de máquina no representan un estándar, y el que se presenta aquí no es un lenguaje para una computadora en particular; sin embargo, las ideas que se describen a continuación son aplicables a todas las computadoras. El lenguaje de máquina propuesto

aquí se basa en mucho en las instrucciones que se dan a la ALU de la sección 2.8. Además, se añaden otras instrucciones útiles que se incluyen en la mayoría de las computadoras; sin embargo, inicialmente no se especifican instrucciones de entrada/salida. Estas se discuten más adelante.

Recuérdese que, en esta computadora sencilla, la palabra de una instrucción contiene 16 bits. Los 8 bits más significativos toman valores binarios de las instrucciones de la siguiente tabla. Los 8 bits menos significativos normalmente contienen direcciones de memoria. Para algunas instrucciones, la dirección de memoria no resulta un dato apropiado, y se le da un valor de 00000000₂. En realidad, podría dársele cualquier valor, ya que para efectos prácticos la computadora lo ignora.

instrucción	código binario	código hexadecimal
suma el contenido de la localidad de memoria dada con el acumulador	00000001	01
limpiar el acumulador	00000010	02
AND Lógico	00000011	03
OR Lógico	00000100	04
XOR Lógico	00000101	05
corrimiento a la derecha	00000110	06
corrimiento a la izquierda	00000111	07
complemento	00010000	10
incremento	00010001	11
verificación negativa	00010010	12
verificación de cero	00010011	13
colocar en la dirección de memoria dada	00010100	14
salto incondicional	00100000	20
salto si cero	00100001	21
salto si negativo	00100010	22
continua	00000000	00
detener el cómputo	01110111	77

Las primeras once instrucciones se discuten en la sección 2.8, en términos de operaciones de la ALU, y se detallan en las Secciones 3.4 y 3.5. Además, la instrucción para almacenar el contenido del acumulador en una dirección dada de memoria (14₁₆) y la instrucción para detener el cómputo (77₁₆) también se discuten en la sección 3.5. En seguida, tras describir brevemente las nuevas

instrucciones, se muestra la programación en lenguaje de máquina mediante algunos ejemplos de programas muestra.

A la penúltima instrucción, 00_{16} , se le llama continua. Esta instrucción no hace nada, es decir, al llegar a ella la computadora simplemente procede a la siguiente instrucción. Aun cuando esta instrucción no parece tener ninguna aplicación práctica, resulta muy útil, sobre todo cuando es necesario introducir algunos tiempos de espera en el programa. Otras instrucciones nuevas son todas aquellas que involucran saltos. El salto incondicional causa que el contenido del PC cambie. Recuerdese que el PC contiene la dirección de la localidad de memoria en la cual la siguiente instrucción debe buscarse. Normalmente, el PC sólo se incrementa en una unidad; sin embargo, ocasionalmente el programador puede requerir repetir algún segmento de instrucciones del programa, o evitarlo. El salto incondicional permite esto. Por ejemplo, considérese la instrucción:

$$0010000001000000_2 = 2040_{16}$$

Esta instrucción hace que el PC termine conteniendo el valor 40_{16} . De no haber otra instrucción de salto en esta dirección, el PC sigue incrementándose en una unidad, como es normal. Así, la siguiente instrucción a ser ejecutada es aquella en la dirección 41_{16} .

Las otras dos instrucciones de salto se les conoce como saltos condicionales. Funcionan en forma similar al salto incondicional, excepto que la ocurrencia del salto depende del valor almacenado en el acumulador. Por ejemplo, supóngase que $PC = 04_{16}$ y que se ejecuta la instrucción:

$$2142_{16}$$

Esta instrucción representa un salto si el valor almacenado en el acumulador es igual a cero. Si el contenido del acumulador es 00_{16} , entonces la siguiente dirección almacenada en el PC es 42_{16} . Así, la siguiente instrucción a ejecutar se toma de esta dirección. Por otro lado, si el valor almacenado en el acumulador no es cero, entonces el salto condicional es ignorado. Esto es, el PC toma el valor de 05_{16} , donde se busca la siguiente instrucción a ejecutar.

La instrucción salto si negativo trabaja esencialmente igual al salto si cero, excepto en que ahora el salto ocurre si el contenido del acumulador es negativo.

3.7.1. Programas simples

Para mostrar las ideas que se han discutido hasta ahora, a continuación se escriben dos programas en lenguaje de máquina. El objetivo no es propiamente enseñar a programar, sino más bien explicar el lenguaje de máquina y la operación de la computadora digital.

En el primer programa se realiza la resta de un número N_1 de otro número N_2 . Si el resultado es negativo, entonces se suma 10_{10} al número, mientras que si el resultado es positivo o cero, se le suma 5_{10} . Después de hacer todo esto, al resultado se le añade otro número N_3 . El programa y sus direcciones de memoria se presenta en la tabla siguiente.

dirección	contenido binario	contenido hexadecimal	comentario
1	0000001000000000	0200	limpiar el acumulador
2	0000000110010001	0191	lee sustraendo
3	0001000000000000	1000	complementar sustraendo
4	0001000100000000	1100	sumar 1 al sustraendo (Complemento a 2)
5	0000000110010010	0192	sumar minuyendo
6	0010001001010000	2250	salto si negativo
7	0000000101010111	0157	suma el contenido de 57_{16} al acumulador
8	0000000000000000	0000	continua
9	0000000110010011	0193	suma el contenido de 93_{16} al acumulador
A	0111011100000000	7700	detener
50	0000000101100010	0162	sumar el contenido de 62_{16} al acumulador
51	0010000000001000	2008	salto incondicional a la dirección 08_{16} ($PC = 08_{16}$)
57	0000000000000101	0005	dato $5_{16} = 5_{10}$
62	0000000000001010	000A	dato $A_{16} = 10_{10}$
91	0000000000000001	0001	dato $1_{16} = 1_{10}$
92	0000000000000111	0007	dato $7_{16} = 7_{10}$
93	0000000000001001	0009	dato $9_{16} = 9_{10}$

Considérese la operación de tal programa. Los números N_1 , N_2 y N_3 se almacenan en las direcciones de memoria 91_{16} , 92_{16} y 93_{16} , respectivamente. Los valores de estos números son los siguientes:

$$N_1 = 1_{16} = 1_{10}$$

$$N_2 = 7_{16} = 7_{10}$$

$$N_3 = 9_{16} = 9_{10}$$

En este punto, se revisa la operación del programa paso a paso. La instrucción en la dirección de memoria 1_{16} causa que el acumulador se limpie, de tal modo que el valor inicial del acumulador es cero. La segunda instrucción (en la dirección de memoria 2_{16}) hace que el contenido de la dirección de memoria 91_{16} se sume al contenido del acumulador. Este valor es N_1 , el cual debemos restar de N_2 . Para hacer esto, se obtiene el complemento a 2, mediante complementar y sumarle una unidad. Las siguientes dos instrucciones en las localidades de memoria 3_{16} y 4_{16} realizan tales operaciones. De este modo, hasta este punto, el acumulador almacena el valor equivalente a $-N_1$.

La siguiente instrucción (en la dirección de memoria 5_{16}) provoca que el contenido de la dirección de memoria 92_{16} se sume al acumulador. Esto suma N_2 con $-N_1$, de modo que se obtiene la resta deseada $N_2 - N_1$. Es en este punto donde se decide si el resultado es negativo y se salta con la instrucción en la dirección 6_{16} . Esto significa que si el valor almacenado en el acumulador es negativo, entonces el flujo del programa cambia (salta) a la dirección 50_{16} . Si el número almacenado en el acumulador no es negativo, entonces la siguiente instrucción que debe ejecutarse es aquella contenida en la dirección 7_{16} . Tal instrucción causa que el contenido de la dirección de memoria 57_{16} se sume al acumulador. Como se ha almacenado un valor de 5_{10} en tal dirección, entonces este valor se suma al contenido del acumulador. Así, se tiene ahora en el acumulador el valor $N_2 - N_1 + 5_{10}$.

La instrucción en la dirección 8_{16} es continua. La computadora la obtiene, y prosigue a ejecutar la siguiente instrucción. Más adelante, se detalla que en realidad esta instrucción se requiere como un punto seguro para saltar de vuelta al cuerpo principal del programa.

La siguiente instrucción a ejecutar es aquella contenida en la localidad de memoria con la dirección 9_{16} . Esta instrucción hace que el contenido de la localidad de memoria 93_{16} se sumen con el contenido del acumulador. Hasta este punto,

entonces, se tiene que en el acumulador se ha realizado la operación $N_2 - N_1 + 5_{10} + N_3$. El cómputo finaliza, y la computadora debería detenerse. En realidad, un programa más práctico requiere que se presente el resultado al usuario, por lo que sería necesario añadir instrucciones para tal fin. Pero tales instrucciones no se han incluido en este ejemplo, dado que todavía no se ha discutido la salida de datos.

Ahora bien, considérese de nuevo el salto en la dirección 6_{16} . Supóngase que se tienen valores diferentes almacenados en las direcciones 91_{16} y 92_{16} , de modo que la diferencia $N_2 - N_1$ resulta negativa. Al realizar el salto, resulta que la siguiente instrucción a ejecutar es aquella almacenada en la dirección 50_{16} . Esta instrucción suma el contenido de la dirección 62_{16} , que es un 10_{10} , al contenido del acumulador. Por lo tanto, si $N_2 - N_1$ es negativo, se realiza la instrucción para obtener en el acumulador el valor $N_2 - N_1 + 10_{10}$.

Nótese que la última instrucción se encuentra en la dirección de memoria 50_{16} . Después de ejecutarla, la siguiente instrucción a ejecutar es aquella contenida en la dirección 51_{16} , que es un salto incondicional que coloca la dirección 8_{16} en el PC. La instrucción siguiente a ejecutar es, entonces, aquella contenida en la dirección de memoria 8_{16} . Esta es la instrucción continua, que en realidad no hace nada, pero que sirve aquí como un punto seguro de retorno. Así, la siguiente instrucción a ejecutar es aquella en la dirección 9_{16} , que como se ha dicho anteriormente, se encarga de sumar el valor N_3 al contenido del acumulador, y el cómputo se detiene. De este modo, el resultado almacenado en el acumulador si $N_2 - N_1$ es negativo es el valor $N_2 - N_1 + 10_{10} + N_3$.

Supóngase ahora que se desea cambiar los datos y volver a ejecutar el programa. Como los datos se almacenan en las direcciones 91_{16} , 92_{16} y 93_{16} , se requiere entonces solamente cambiar los valores que se encuentran almacenados, sin necesidad de modificar el resto del programa. Esto es muy conveniente, particularmente si el programa es muy largo.

En seguida se considera otro ejemplo de un programa simple en lenguaje de máquina. El programa anterior, aun cuando realiza lo estipulado, no hace algo que se pudiera considerar útil. De este modo, se propone realizar un programa con un objetivo mejor definido. Supóngase que se desea realizar la suma de los primeros 100 números enteros:

$$1 + 2 + 3 + 4 + \cdots + 98 + 99 + 100$$

Un programa que hace esta operación directamente tendría muchos pasos, y sería muy tedioso para programar; sin embargo, la siguiente tabla presenta un programa en lenguaje de máquina que utiliza los saltos para acortar el programa y eliminar la repetición.

Dirección	Contenido hexadecimal	Contenido binario
1	0200	0000001000000000
2	0181	0000000110000001
3	1100	0001000100000000
4	1481	0001010010000001
5	1000	0001000000000000
6	1100	0001000100000000
7	0180	0000000110000000
8	2190	0010000110010000
9	0200	0000001000000000
A	0181	0000000110000001
B	0182	0000000110000010
C	1482	0001010010000010
D	2001	0010000000000001
80	0065	0000000001100101
81	0000	0000000000000000
82	0000	0000000000000000
90	7700	0111011100000000

La operación de este programa comienza limpiando el contenido del acumulador con la instrucción en la dirección 1_{16} . La siguiente instrucción en la dirección de memoria 2_{16} provoca que el contenido de la localidad de memoria con dirección 81_{16} se sume con el acumulador. Al principio, tal número tiene valor cero. El contenido de esta localidad de memoria es el siguiente número a ser sumado. La siguiente instrucción (en la dirección 3_{16}) le suma 1 al contenido del acumulador. Con la siguiente instrucción, en la dirección 4_{16} , el contenido del acumulador ahora se almacena en la dirección 81_{16} . El valor anterior en esta dirección se pierde.

El número que se acaba de almacenar en la dirección 81_{16} eventualmente se añade a la suma, pero antes es necesario verificar si no excede el valor de 100_{10} . Para hacer esto, el contenido del acumulador se complementa (con la instrucción

en la dirección 5_{16}) y se incrementa (con la instrucción en la dirección 6_{16}). Así, se ha calculado el complemento a 2 del número a verificar. La instrucción en la dirección 7_{16} hace que el número almacenado en la dirección 80_{16} se sume al complemento a 2 obtenido. Nótese que el valor almacenado en esta dirección es el valor 101_{10} . Si después de la operación el contenido del acumulador es cero, no se debe continuar sumando el contenido de la localidad de memoria 81_{16} con la suma. La siguiente instrucción, en la dirección 8_{16} , es un salto si se verifica un valor cero en el acumulador. De hecho, si el número almacenado en la dirección 81_{16} es 101_{10} , el resultado de la diferencia es cero, por lo que la siguiente instrucción a ejecutarse después del salto se encuentra en la dirección 90_{16} . La instrucción almacenada en esta dirección es 77_{16} , que significa detener el proceso. Así, solamente si el número que se almacena en la localidad de memoria 81_{16} tiene el valor 101_{10} , el cómputo se completa. Pero si el valor almacenado en tal dirección es menor que 101_{10} , entonces el proceso prosigue.

Regresando a la instrucción de salto en la dirección 8_{16} , supóngase que el salto falla, es decir, el contenido del acumulador no es cero. La siguiente instrucción a realizarse es aquella contenida en la dirección 9_{16} , y ésta causa que el acumulador se limpie.

Siguiendo con el programa, la instrucción en la dirección A_{16} hace que el contenido de la dirección 81_{16} se sume con el contenido del acumulador. Nótese que el valor en esta dirección es 1_{10} . La instrucción en la dirección B_{16} hace que el contenido de la dirección 82_{16} se sume al contenido del acumulador. Tal contenido en la dirección 82_{16} es todavía cero.

Ahora bien, la instrucción de la dirección C_{16} hace que el contenido del acumulador se coloque en la dirección de memoria 82_{16} . Este valor es 1_{10} , de modo que ésta es la primera suma.

La siguiente instrucción en la dirección D_{16} hace que se salte incondicionalmente a la dirección 1_{16} . Por lo tanto, el programa regresa a la primer instrucción, haciendo un ciclo. Si se va a través del programa de nuevo, en esta ocasión se almacena un valor 2_{10} en la dirección de memoria 81_{16} . Después de verificar que este número es menor que 101_{10} , tal valor se suma al número almacenado en la dirección 82_{16} ($2_{10} + 1_{10} = 3_{10}$).

Si se va a través del programa varias veces, en ciclos, es posible observar lo siguiente: en cada ciclo se van sumando los valores $1, 2, 3, \dots$ a la suma. La suma acumulada se va almacenando en la dirección 82_{16} . Antes de sumarse, el número se almacena en la dirección 81_{16} , de modo que antes de que se realice la suma,

se verifica si tal valor no es mayor que $65_{16} = 101_{10}$, mediante realizar la resta usando el complemento a 2. Cuando el resultado de la resta sea cero, el proceso termina.

En caso de que se desee cambiar el programa para obtener la suma de los primeros N números ($1 + 2 + 3 + \dots + N$), es necesario tan solo cambiar el número almacenado en la dirección 80_{16} de 101_{10} al valor $N + 1_{10}$.

Como es el caso en los dos ejemplos anteriores de programas simples en lenguaje de máquina, antes de considerar los detalles para escribir un programa, es necesario entender primero la idea básica que debe realizarse, es decir, el algoritmo que realiza. Por ejemplo, se podría expresar la idea básica del último programa de la siguiente forma: se desea obtener el resultado de la suma de los 100 primeros enteros. Al resultado se le puede dar el nombre de **SUMA**, y a un número entero, el nombre de **N**. Así, al inicio, se puede hacer que ambos, **SUMA** y **N** tengan valor 0. En seguida se incrementa **N** en una unidad. Es necesario verificar ahora que el valor de **N** sea menor que 101_{10} . Si este es el caso, entonces **N** se suma a **SUMA**, y el proceso se repite. Por otro lado, si **N** es igual a 101_{10} , entonces se detiene el proceso, y el valor de **SUMA** representa el resultado.

La idea detrás de un programa puede representarse de varias maneras. Un ejemplo utilizado tradicionalmente en varios cursos de programación, es el diagrama de flujo (*flow chart*). En general, se trata de una figura gráfica que sirve para representar y entender de manera más sencilla al programa, en lugar de utilizar solamente una descripción verbal. La Figura 3.18 muestra un ejemplo de la descripción del programa anterior en términos de un diagrama de flujo.

Nótese cómo el diagrama describe el flujo de control del programa. El bloque con forma de diamante se utiliza cuando debe tomarse una decisión (es decir, si hay un salto condicional). Si $N - 101 = 0$, entonces se sale del bloque de decisión por la salida marcada con **si**, y el proceso termina. En caso contrario, si $N - 101$ no tiene valor 0, se sale del bloque de decisión por la salida con etiqueta **no**, se incrementa **N** en una unidad y la operación se repite.

3.7.2. Salida de datos

En esta sección se considera la salida de datos de la computadora mediante el ejemplo de un programa sencillo. Cada dispositivo de entrada/salida se conecta a una tarjeta de interface (*interface board*), la cual a su vez se conecta con el *bus* de la computadora. La tarjeta de interface convierte la información digital del

bus a alguna forma de información que pueda ser usada por el dispositivo de entrada/salida (y viceversa). Cada dispositivo de entrada/salida generalmente tiene su propia tarjeta de interface, pero frecuentemente una sola tarjeta puede usarse para muchos dispositivos. Es en las tarjetas de interface donde se encuentran los *buffers* (memoria) asociados con los dispositivos de entrada/salida, y se utilizan para almacenar datos a enviar o recibir. En algunas computadoras, los buffers son parte de la memoria principal. El *bus* conecta todas las tarjetas de interface, cada una de las cuales se conecta con su dispositivo de entrada/salida respectivo. Los dispositivos de entrada/salida, como terminales, impresoras, unidades de cinta, disco, etc., son llamados también con el nombre genérico de periféricos (*peripherals*).

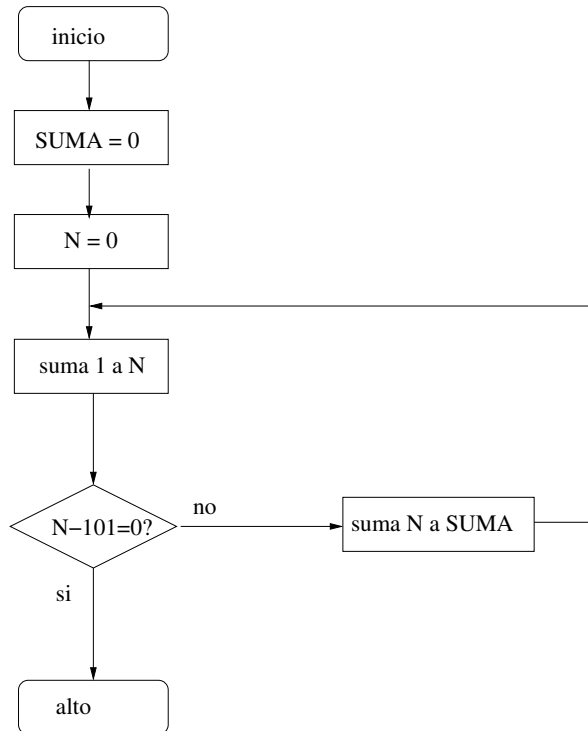


Figura 3.18: Diagrama de flujo del algoritmo que suma los primeros 100 números enteros.

Supóngase que en la computadora sencilla que se ha descrito hasta ahora, todos los datos a salir deben colocarse en el acumulador y entonces, mediante la instrucción adecuada, se transfiere a las terminales del *bus* reservadas para la entrada/salida. Para hacer esto, se utiliza la siguiente instrucción en lenguaje de máquina:

$$00110000_2 = 30_{16}$$

La instrucción 30_{16} sólo especifica que hay datos por salir; sin embargo, la instrucción también debe indicar cuál dispositivo debe utilizarse, y qué función debe realizar tal dispositivo. Por ejemplo, supóngase que se tiene una unidad de cinta. La información es normalmente almacenada en la cinta en forma de grupos de datos llamados bloques. Se puede, entonces, hacer la instrucción de tal modo que se le ordene a la unidad de cinta recorrer toda la cinta para comenzar con el primer bloque. Alternativamente, se puede hacer que la instrucción indique a la unidad de cinta que salte uno o varios bloques de datos, de modo que nuevos datos puedan escribirse sin afectar los datos ya contenidos en la cinta.

Como se considera para la computadora una palabra de instrucción con 16 bits, entonces la forma de la instrucción de salida debe ser:

$$00110000d_3d_2d_1d_0f_3f_2f_1f_0$$

Los primeros 8 bits indican que se trata de una instrucción de salida. Los bits d_i y f_i son adicionales, e indican el tipo y función del dispositivo. De hecho, los bits d_i se utilizan para indicar un código del dispositivo de salida, mientras que los bits f_i indican si el primer bloque debe o no saltarse. Se propone el siguiente código para diferentes dispositivos:

dispositivo	binario	hexadecimal
terminal	0000	00
impresora	0001	01
unidad de disco	0010	02
unidad de cinta	0011	03

Para las funciones de la unidad de cinta del ejemplo, se tienen las siguientes funciones:

función	binario	hexadecimal
recorrer la cinta al inicio	0001	01
saltar al siguiente bloque	0010	02

Así, si por ejemplo se desea desplegar datos en la terminal, la instrucción sería:

$$0011000000000000_2 = 3000_{16}$$

En este caso, los últimos dos bits no tienen un significado definido, por lo que pueden ignorarse.

O bien, supóngase que se desea colocar un resultado en una cinta magnética, y se desea que la cinta inicie desde el principio y se escriba sobre cualquier otra información que ya se encuentre en ella. Entonces, la instrucción sería:

$$0011000000110001_2 = 3031_{16}$$

Por otro lado, si se desea saltar en la cinta hasta el último bloque de datos y escribir nuevos datos después, la instrucción tendría la forma:

$$0011000000110010_2 = 3032_{16}$$

Para el caso de la unidad de disco habría instrucciones similares, en cuyo caso la pista donde puede escribirse la salida puede especificarse.

Todas las interfaces de entrada/salida se conectan a las mismas terminales del *bus*, que utilizan un decodificador similar al que se presenta en la Figura 3.15, causando que la interface del dispositivo apropiado se active y opere de forma adecuada. El decodificador recibe los bits:

$$d_3d_2d_1d_0f_3f_2f_1f_0$$

Esto envía una señal de instrucción a la interface apropiada, de modo que el dispositivo de entrada/salida correcto opere y permita la salida o entrada de datos.

Cuando se da una salida de datos, primero se coloca una instrucción de salida en las terminales del *bus* de entrada/salida. En seguida, los datos numéricos binarios se colocan en las mismas terminales del *bus*. Nótese que estos números de proveen en secuencia. Supóngase que, en esta computadora sencilla, hay una

instrucción que causa que el contenido del acumulador se coloque en las líneas de entrada/salida del *bus*. Tal instrucción puede ser:

$$01110000_2 = 70_{16}$$

De este modo, primero se requiere la instrucción de salida, y enseguida, la instrucción 70_{16} . Después de esto, se coloca un dato de salida en el acumulador, y a continuación, se da de nuevo la instrucción 70_{16} . Por ejemplo, supóngase que se desea imprimir el número almacenado en la dirección de memoria 91_{16} en la terminal, y además, almacenar el número $0011000000000000_2 = 3000_{16}$ en la dirección de memoria 26_{16} (nótese que esta es la instrucción que opera sobre la terminal). Entonces, la salida de datos se realiza mediante ejecutar las siguientes instrucciones (en hexadecimal):

```
0200
0126
7000
0200
0191
7000
```

La primera instrucción limpia el acumulador. Enseguida se suma el contenido de la dirección 26_{16} al contenido del acumulador, lo que coloca la instrucción de salida por terminal en el propio acumulador. La siguiente instrucción (7000_{16}) causa la salida del contenido actual del acumulador sobre las líneas del *bus*. El acumulador se limpia, y entonces se le colocan los datos numéricos de salida. Ya que el acumulador se limpió anteriormente, ahora contiene los datos numéricos de salida. Enseguida, este número se provee a las líneas de entrada/salida. Nótese que este procedimiento de salida de datos puede variar de computadora a computadora; sin embargo, entender este procedimiento debe permitir comprender los procedimientos en diferentes computadoras. De hecho, los detalles de salida de datos en una computadora dada pueden involucrar diferentes instrucciones, pero las ideas básicas son esencialmente las mismas.

Los dispositivos de entrada/salida son comúnmente mucho más lentos que la operación de la computadora. De hecho, es muy posible que los datos se provean en una tasa más rápida de lo que son impresos. El buffer de entrada/salida (véase

la sección 3.6) intenta resolver este problema; sin embargo, si los datos se proveen lo suficientemente rápido, el buffer puede llenarse. Es por esto que la unidad de entrada/salida cuenta con un registro de un bit llamado bandera (*flag*). Si el buffer se llena, la bandera toma valor 0; si el buffer no está lleno y puede recibir más datos, la bandera tiene valor 1. Cuando el valor de la bandera es 0, esto actúa como una señal para la unidad de control y el proceso de salida entra en una pausa; sin embargo, la salida de datos prosigue. Una vez que el registro de entrada/salida puede recibir más datos, la bandera toma el valor de 1 y el proceso de salida continua.

Algunas computadoras tienen otras banderas más sofisticadas, conocidas como interrupciones (*interrupts*). Estas señales causan que el proceso de la computadora se detenga y prosiga después si alguna acción o problema predecible ocurre en un dispositivo de entrada/salida. Por ejemplo, si la impresora se queda sin papel, el proceso se detiene.

Se pueden escribir programas que reciban datos de entrada en momentos específicos. En tal caso, los datos de entrada se colocan en el registro de entrada/salida, y cuando se da la instrucción apropiada de entrada, los contenidos de este registro se copian al acumulador. Así, se puede dar entrada de datos cuando se requiera.

3.8. Lenguaje ensamblador

Programar utilizando lenguaje de máquina resulta muy tedioso. El programador debe recordar los códigos de instrucción que corresponden a cada instrucción, o al menos, debe continuamente revisar su definición. Por otro lado, en casi todas las computadoras reales la lista de instrucciones es mucho mayor que la definida para la computadora sencilla. Esto hace que la labor de utilizar las instrucciones sea muy difícil. Más aún, el programador debe recordar en todo momento las direcciones de memoria donde se encuentran almacenadas las variables y las instrucciones, lo que es particularmente dificultoso cuando entre las instrucciones se involucran saltos. Como un ejemplo de estos problemas, supóngase que se requiere realizar la siguiente suma:

$$x = a + b + c + d$$

Debe haber una localidad de memoria reservada para cada una de las variables x , a , b , c y d , y las direcciones de esas localidades deben en todo momento

recordarse. Si se tiene un número relativamente pequeño de variables, la tarea es más o menos simple; sin embargo, en un programa grande, con muchas variables, el programador debe entonces crear una tabla para recordar la posición de las variables, lo que normalmente consume mucho tiempo y esfuerzo.

Para hacer la labor de programación menos tediosa, se han desarrollado otros tipos de lenguajes para el uso del programador, en lugar de utilizar lenguaje de máquina. En esta sección, se discute el lenguaje ensamblador (*assembly language*), el cual está muy cercanamente relacionado con el lenguaje de máquina, pero resulta mucho más fácil de utilizar por parte de los programadores.

Siempre que una computadora realiza alguna clase de proceso, un programa en lenguaje de máquina (binario o ejecutable) dirige su operación. Consecuentemente, siempre que otra forma de lenguaje se utilice, ésta debe ser traducida a lenguaje de máquina. Un programa conocido como ensamblador (*assembler*) dirige la acción de la computadora, de modo que traduce un programa del lenguaje ensamblador a lenguaje de máquina. El lenguaje ensamblador es, entonces, un conjunto de instrucciones que se utilizan para escribir un programa en ensamblador. A continuación, primero se describe un lenguaje ensamblador simple, y luego se comenta cómo la computadora lo usa.

Las instrucciones en lenguaje de máquina se realizan en términos de códigos binarios. En lenguaje ensamblador, los códigos se reemplazan por pnemónicos (*pnemonics*), que comúnmente son más fáciles de recordar que los códigos binarios. Por ejemplo, en lugar de escribir $01_{16} = 00000001_2$ para la operación de suma, simplemente se escriben las letras ADD (la palabra inglesa para suma). Nótese que ADD es mucho más fácil de recordar que 00000001. Por supuesto, se considera aquí que se puede dar una secuencia de letras como entrada a la computadora. Esto implica el uso de los códigos de detección de errores, que se mencionan en la sección 4.7.

Enseguida, se presenta un lenguaje ensamblador sencillo. Nótese que, como es el caso de los lenguajes de máquina, los lenguajes ensambladores no tienen un estándar. De este modo, por convención se utilizan tres letras para designar una instrucción en lenguaje ensamblador. De hecho, es conveniente utilizar un número fijo de letras para las instrucciones en lenguaje ensamblador, pero esto no siempre es necesario. En general, se puede pensar que una instrucción en ensamblador corresponde directamente con una instrucción en lenguaje de máquina. Nótese que los pnemónicos del lenguaje ensamblador son mucho más fáciles de recordar que los códigos binarios que representan. Otra ventaja de utilizar lenguaje ensamblador

es que el programador no tiene que recordar las direcciones en memoria de los datos almacenados. Tales valores se les da el nombre de variables, y como parte del lenguaje, puede dárseles un nombre. Comúnmente, tal nombre puede constar hasta de seis caracteres alfanuméricos.

instrucción en lenguaje de máquina	instrucción en lenguaje ensamblador
01	ADD
02	CLE
03	AND
04	ORR
05	XOR
06	SHR
07	SHL
10	COM
11	INC
12	NEC
13	ZEC
14	PIM
20	BRU
21	BRZ
22	BRN
70	CON
77	STP

Como ejemplo de un programa escrito en lenguaje ensamblador, a continuación se reescribe el programa del primer ejemplo de la sección 3.5, ahora en lenguaje ensamblador:

```
CLE
ADD DATA1
ADD B
PIM ANS
STP
DATA1:3
B:4
ANS:0
```

DATA1, B y ANS son nombres de variables. DATA1 es equivalente a especificar la dirección de memoria 10_{16} . Similarmente, se escribe B en lugar de la dirección 11_{16} y ANS en lugar de 12_{16} . Así, el programador no requiere recordar las direcciones de memoria. En cualquier momento en que se hace referencia a DATA1, se utiliza la dirección adecuada de memoria.

Ahora bien, a continuación se describe cómo la computadora trabaja sobre un programa escrito en lenguaje ensamblador para producir un programa en lenguaje de máquina. Se utiliza el programa ensamblador previamente mencionado. Este programa se escribe solo una vez, y la mayoría de los programadores no necesitan hacerle modificaciones. El ensamblador se carga en memoria de la misma forma en que se carga cualquier otro programa que ejecute la computadora. Enseguida, el programa en lenguaje ensamblador se le provee al ensamblador como datos de entrada. El ensamblador traduce el programa en lenguaje ensamblador a un programa en lenguaje de máquina, y sólo entonces se cuenta con un programa que puede ser ejecutado por la computadora; sin embargo, no fue necesario que el programador escribiera código en lenguaje de máquina. Solo es necesario escribir un programa en ensamblador.

Se puede utilizar alternativamente el siguiente procedimiento: el ensamblador traduce el programa en lenguaje ensamblador a un programa en lenguaje de máquina que se almacena en memoria, de modo que éste último puede a la vez almacenarse en, por ejemplo, un disco. Después de esto, es posible cargar el programa en lenguaje de máquina a partir del disco, y ejecutarlo. De nuevo, el programador no requiere escribir su programa en lenguaje de máquina.

En este punto surge la pregunta: ¿cómo trabaja un ensamblador? Recuérdese que cada enunciado en un programa en lenguaje ensamblador se le provee como datos de entrada. Si se utiliza una terminal, cada letra genera un código (véase sección 4.7), de modo que cada instrucción en lenguaje ensamblador puede representarse por diferentes secuencias de ceros y unos. Cuando el ensamblador se carga en la computadora, cada código se almacena en una dirección diferente de memoria. De manera similar, cada código que corresponde al programa en lenguaje ensamblador se almacena en otras direcciones de memoria. Ahora, cuando un enunciado del programa en lenguaje ensamblador se proporciona al ensamblador, su código se compara con los códigos de las instrucciones en lenguaje ensamblador. Cuando se encuentra un código igual al código de la instrucción, la instrucción correspondiente en lenguaje de máquina se obtiene y añade al programa en lenguaje de máquina que se está generando.

Respecto a la transformación de variables a direcciones de memoria, se sabe que todas las variables utilizadas en un programa en lenguaje ensamblador se escriben siempre después de una instrucción. Así, cualquier información alfanumérica que siga después de las tres letras de una instrucción puede identificarse como variable. El ensamblador, entonces, requiere tomar en cuenta el número y tipo de variables, a fin de apartar las direcciones necesarias para almacenar las variables del programa.

Durante la formación del programa en lenguaje de máquina a partir de un programa en lenguaje ensamblador, es común que este último se lea dos veces (aunque en algunas computadoras se lee únicamente una sola vez, y ya almacenado en memoria, se revisa dos veces). La primera lectura o pasada se utiliza para identificar todas las variables y establecer localidades de memoria para cada una de ellas. El programa se va revisando, y conforme se van hallando variables, se almacena su código. El nombre usado en el programa para la variable se almacena en una tabla de símbolos. Esta tabla eventualmente contiene también las direcciones en memoria de cada variable. Una variable puede utilizarse muchas veces dentro del programa, pero puede aparecer solo una vez en la tabla de símbolos.

En el espacio de memoria del programa, debe haber localidades de memoria asignadas tanto para cada variable como para cada instrucción del programa. Durante la primera pasada, una localidad de memoria se reserva para cada instrucción; las localidades de memoria para las variables se reservan después de reservar localidades para las instrucciones. Considérese por ejemplo el programa anterior en lenguaje ensamblador. La primera instrucción es CLE. Una palabra de memoria se ha reservado para esta instrucción. Es importante notar que las localidades de memoria se reservan en orden estrictamente secuencial. La primera instrucción recibe la dirección de memoria con menor valor numérico, por ejemplo, $00000001_2 = 01_{16}$. A continuación, se encuentra la instrucción ADD DATA1. Se reserva la dirección de memoria 02_{16} para ADD, mientras que se da entrada a DATA1 en la tabla de símbolos, pero todavía no se ha reservado una localidad de memoria para ella. Similarmente, al encontrarse ADD B se reserva la dirección 03_{16} para ADD, y se da entrada a B en la tabla de símbolos. La instrucción siguiente, PIM, se coloca en la dirección 04_{16} , y se da entrada a la variable ANS en la tabla de símbolos. Cuando se encuentra la instrucción STP, se le asigna la dirección 05_{16} . Ahora bien, se encuentra la cadena DATA1:3. Los dos puntos indican que DATA1 es una variable. Es hasta ahora que se le asigna la dirección en memoria 06_{16} a DATA1. Tal información se añade a información

sobre DATA1 en la tabla de símbolos. En forma parecida, se reserva la dirección 07₁₆ para B, cuyo valor es 4, y finalmente, se reserva la dirección 08₁₆ para ANS, cuyo valor es 0.

Como es notorio, se da un valor inicial a cada dato mediante los dos puntos (:). Este caracter indica que el número que le sigue es el valor que debe almacenarse en la dirección de memoria de la variable adecuada. Nótese que debe haber una inicialización para cada una de las variables, aunque esto no sea del todo cierto para todo ensamblador. Por ejemplo, después de que la suma se ha realizado, se coloca el resultado en la variable llamada ANS. Se requiere, entonces, que ANS tenga un valor conocido desde un principio, por lo que se hace la indicación ANS:0 para ello. Esto es necesario para reservar memoria para la variable ANS. Nótese que el valor 0 es arbitrario. Se puede utilizar cualquier valor. Una vez que ANS se coloca en memoria, su valor se reescribe una y otra vez. Se supone aquí que las localidades de memoria se asignan en orden, partiendo de una dirección inicial 01₁₆; sin embargo, varios ensambladores pueden variar en tal procedimiento.

En la primera pasada del ensamblador se reservan todas las localidades de memoria necesarias para el programa. Durante la segunda pasada los pnemónicos se convierten a sus códigos equivalentes en lenguaje de máquina, y se almacenan en las direcciones de memoria reservadas para las instrucciones. Esta es la primera parte de la palabra en lenguaje de máquina. Si el nombre de una variable está asociado con una instrucción, entonces se le busca en la tabla de símbolos, de modo que la dirección de la variable se vuelve la segunda parte de la palabra en lenguaje de máquina. Este proceso se repite paso a paso, instrucción por instrucción, hasta que todo el programa en lenguaje ensamblador se ha convertido en una programa en lenguaje de máquina, cuyas instrucciones se almacenan en secuencia.

Después de ejecutar el programa ensamblador, el programa generado se escribe generalmente en disco. La memoria principal se limpia, y el nuevo programa está listo para ejecutarse.

Puede ser que la memoria no sea lo suficientemente grande para almacenar el ensamblador, la tabla de símbolos y el programa en lenguaje de máquina. Pero si la memoria es lo suficientemente grande para completar la primera pasada, entonces se puede hacer lo siguiente: se puede iniciar la segunda pasada, pero si la memoria se llena, entonces la segunda pasada entra en una pausa. La memoria que contiene la parte del programa en lenguaje de máquina se puede transferir a disco, después de lo cual se libera parte de la memoria, y se continua con la

segunda pasada. Este proceso se puede repetir mientras la memoria se llene, y hasta que el programa en lenguaje de máquina termine de generarse.

3.9. Lenguajes de alto nivel

Aun cuando el lenguaje ensamblador es mucho más fácil de utilizar que el lenguaje de máquina, resulta todavía muy tedioso de programar. Por ejemplo, supóngase que se desea escribir un programa que evalúe la siguiente ecuación:

$$x = (a + b)(c - d)/(a - b)$$

Para realizar esta simple ecuación se requiere escribir muchas líneas de código en ensamblador, lo que requiere un gran esfuerzo para el programador. Es por esto que se han desarrollado otro tipo de lenguajes que permiten al programador trabajar con elementos de un nivel de abstracción más alto que lo que puede expresarse utilizando lenguaje ensamblador. Existen en la actualidad un gran número de lenguajes de programación de alto nivel que pueden utilizarse para expresar operaciones cada vez más complejas. Ejemplos de estos lenguajes son Fortran, Pascal, y C.

A continuación, se muestra cómo la expresión anterior puede evaluarse utilizando lenguaje C.

```
int a = 3, b = 4, c = 5, d = 3, x = 0;  
x = (a + b)*(c - d)/(a - b);  
printf(x);
```

Nótese que el asterisco (*) es el símbolo utilizado en C para representar la multiplicación.

El programa que realmente se ejecuta en la computadora es un equivalente de este programa, pero en lenguaje de máquina. Cuando se utiliza un lenguaje de alto nivel, el programa fuente (escrito en el lenguaje de alto nivel) se provee como datos básicos para la construcción de su equivalente en lenguaje de máquina, de manera muy similar a como un programa en lenguaje ensamblador son datos para el ensamblador; sin embargo, el programa que convierte un programa fuente a su equivalente en lenguaje de máquina es mucho más complejo y sofisticado que el ensamblador, ya que una sola instrucción en lenguaje de alto nivel equivale a una secuencia de instrucciones en lenguaje de máquina. Los programas que

convierten un programa en lenguaje de alto nivel a lenguaje de máquina pueden ser compiladores (*compilers*) o intérpretes (*interpreters*). Aun cuando permiten una mayor flexibilidad en la labor de programación, el uso de lenguajes de alto nivel implica el seguimiento de reglas rigurosas y restrictivas para el programador; sin embargo, los lenguajes de alto nivel siguen siendo la opción más adecuada para el desarrollo de programas, comparativamente con el lenguaje ensamblador y el lenguaje de máquina.

Fuentes útiles de información

Ars Technica LLC

Ars Technica: The Ars Technopaedia

<http://arstechnica.com/paedia>

Karbo, M.B.

Hardware Guides

<http://www.karbosguide.com>

PC Tech Guide

<http://www.pctechguide.com>

Evaluación

Preguntas de opción múltiple

1. Supóngase que el campo de código de operación de una instrucción contiene 4 bits. Más aun, supóngase que las instrucciones se definen exclusivamente por su código de operación. El conjunto de instrucciones en este caso contiene un máximo de:
 - a) 4 instrucciones.
 - b) 8 instrucciones.
 - c) 16 instrucciones.
 - d) 32 instrucciones.
 - e) 64 instrucciones.

-
2. La ALU realiza la siguiente función:
 - a) Comunica con los dispositivos de entrada/salida.
 - b) Realiza operaciones aritméticas y lógicas.
 - c) Realiza operaciones de transferencia de memoria.
 - d) Almacena datos.
 - e) Busca instrucciones.
 3. Las banderas asociadas con la ALU almacenan:
 - a) Información acerca del resultado de la última operación de la ALU.
 - b) Los resultados de la última operación de la ALU.
 - c) Las señales de control para otros componentes del procesador.
 - d) Las direcciones de memoria de los datos de la ALU.
 - e) El país de origen de la computadora.
 4. El número de bits de la señal de control necesario para un ALU con 25 operaciones es:
 - a) 25.
 - b) 4.
 - c) 5.
 - d) 1.
 - e) 0.
 5. Dentro del procesador, el papel de la Unidad de Control es:
 - a) Manejar la ejecución de programas.
 - b) Realizar operaciones lógico-aritméticas.
 - c) Enviar señales de control al procesador.
 - d) Enviar señales de control a los periféricos.
 - e) Almacenar datos.

6. Los *buffers* triestado se usan para:
 - a) Interconectar localidades de memoria.
 - b) Interconectar dispositivos de entrada/salida.
 - c) Interconectar dispositivos conectados a los *buses*.
 - d) Almacenar números ternarios.
 - e) Proveer un estado lógico extra.
7. El *bus* de entrada/salida conecta:
 - a) El procesador con la memoria principal.
 - b) Los dispositivos de entrada/salida con el procesador y la memoria.
 - c) La memoria con los dispositivos de entrada/salida.
 - d) La entrada con la salida.
 - e) La E con la S.
8. El *bus* de datos es:
 - a) Unidireccional.
 - b) Bidireccional.
 - c) Uniciclo.
 - d) Biciclo.
 - e) Entero.
9. El *bus* de direcciones es:
 - a) Unidireccional.
 - b) Bidireccional.
 - c) Uniciclo.
 - d) Biciclo.
 - e) Entero.
10. La siguiente memoria se especifica por el número de palabras multiplicado por el número de bits por palabra. ¿Cuántas líneas de dirección necesita una memoria de $2G \times 8$?

- a) 30.
 - b) 31.
 - c) 21.
 - d) 20.
 - e) 36.
11. La siguiente memoria se especifica por el número de palabras multiplicado por el número de bits por palabra. ¿Cuántas líneas de datos necesita una memoria de $2G \times 8$?
- a) 2G.
 - b) 8.
 - c) 2.
 - d) 30.
 - e) 1.
12. ¿Cuántos bytes se almacenan en una memoria de $16M \times 32$?
- a) 2^{20} .
 - b) 2^{22} .
 - c) 2^{24} .
 - d) 2^{26} .
 - e) 2^{30} .
13. La memoria de acceso aleatorio es un tipo de memoria que:
- a) Solo puede ser escrita.
 - b) Solo puede ser leída.
 - c) Puede leerse y escribirse.
 - d) Tiene contenidos aleatorios.
 - e) Provee salida aleatorias.
14. En una interfaz de entrada/salida, el registro de estado se utiliza para:
- a) Almacenar las instrucciones de los periféricos dados por el procesador.

- b)* Almacenar los datos que recibe el procesador.
 - c)* Dar aviso al procesador si un dispositivo de entrada/salida está listo para transferir datos.
 - d)* Preparar los datos a transferir del dispositivo de entrada/salida al procesador.
 - e)* Preparar los datos a transferir del procesador al dispositivo de entrada/salida.
- 15. El ciclo de búsqueda implica entre otras cosas:
 - a)* Decodificar la instrucción.
 - b)* Obedecer la instrucción.
 - c)* Ejecutar la instrucción.
 - d)* Decodificar y obedecer la instrucción.
 - e)* Escribir el resultado.

Respuestas a las preguntas de opción múltiple

- 1. *c*
- 2. *b*
- 3. *a*
- 4. *c*
- 5. *a*
- 6. *c*
- 7. *b*
- 8. *b*
- 9. *a*
- 10. *b*
- 11. *b*

12. *d*

13. *c*

14. *c*

15. *a*

Capítulo 4

Sistemas de memoria

Resumen

Se presentan las características de diferentes dispositivos de memoria, junto con las selecciones y discusiones durante el diseño de un sistema de memoria y la administración de memoria requerido por sistemas grandes de cómputo.

Objetivos del capítulo

Al final del capítulo, el estudiante debe ser capaz de:

1. Explicar los conceptos de memoria central y memoria auxiliar.
2. Listar las propiedades de las memorias central y auxiliar.
3. Justificar la necesidad de una jerarquía de memoria en sistemas de cómputo.
4. Explicar el papel de varios componentes de memoria dentro de la jerarquía de la memoria de un sistema de cómputo.
5. Describir los varios tipos de dispositivos de memoria, tales como memorias semiconductoras, discos y cintas.

4.1. Generalidades

Todas las computadoras digitales tienen una memoria que se usa para almacenar tanto programas como datos. En este capítulo se discuten diferentes formas de memoria que se utilizan para almacenar diversas cantidades de datos. En la sección 2.6 ya se menciona un tipo muy sencillo de memoria: el registro.

La memoria que almacena un programa en ejecución o los datos para tal programa se conoce como memoria principal o en ocasiones memoria de trabajo. Además, hay otro tipo de memoria conocida como secundaria, auxiliar o periférica, que permite almacenar programas y datos para su uso futuro. En general, este tipo de memoria consiste de cintas magnéticas y discos. Cuando se desea ejecutar un programa almacenado en memoria secundaria, es necesario leerlo de la cinta o el disco, y cargarlo en la memoria principal.

Los registros, como un tipo de memoria que se ha discutido anteriormente, almacenan datos binarios; sin embargo, hay otras formas de memoria creadas a partir de elementos semiconductores, como por ejemplo, la memoria de sólo lectura (*read-only memory* o ROM), que almacena permanentemente sus datos y no pueden ser (fácilmente) cambiados.

Actualmente, las memorias semiconductoras se fabrican mediante tecnología de circuitos integrados, y son generalmente rápidas y pequeñas. Otros tipos de memorias utilizan materiales magnéticos y ópticos, y son más lentas que las memorias semiconductoras; sin embargo, las memorias magnéticas y ópticas tienen ventajas sobre las memorias semiconductoras: (*a*) tienen una mayor capacidad de almacenamiento de información binaria, y (*b*) en el evento de una falla de corriente, no pierden la información que almacenan. A esta capacidad se le conoce como no volatilidad, en contraste con las memorias semiconductoras, que al fallar la fuente de alimentación eléctrica sí pierden sus contenidos. Son memorias volátiles.

Las memorias se clasifican por la forma en que se les escribe o lee información binaria. De hecho, los contenidos binarios almacenados en una memoria se organizan en grupos de bits llamados palabras. Cada palabra se almacena en una localidad de memoria llamada dirección. Cuando cada palabra se almacenan una tras otra, se dice que el almacenamiento es de tipo secuencial. En tal caso, para poder acceder (leer o escribir) la n -ésima palabra, es necesario pasar por todas las $n-1$ palabras anteriores. En general, las memorias con acceso secuencial tienden a ser muy lentas.

En contraste, la forma más rápida de acceso (lectura o escritura) de memoria es el acceso aleatorio (*random access*), que es característico de la memoria RAM (*random access memory*). En este tipo de memoria, cualquier localidad de memoria puede direccionarse sin considerar ninguna secuencia u orden específico. Ya que la lectura y escritura se hace velozmente, la RAM se utiliza siempre cuando la velocidad de acceso a los datos es importante.

Este capítulo describe brevemente todos estos tipos de memoria. La discusión se inicia con las memorias semiconductoras, particularmente con las memorias de acceso aleatorio (RAM).

4.2. La jerarquía de la memoria

Esta sección explica el concepto de jerarquía de la memoria. Se presentan las propiedades clave de los dispositivos de memoria y los compromisos necesarios para construir un sistema de memoria eficiente.

4.2.1. El sistema de memoria

A nivel sistema, se pueden distinguir dos clases de memoria en las computadoras:

- la memoria central, principal o primaria, que es la memoria de trabajo de la computadora;
- la memoria periférica, auxiliar o secundaria, que funciona como almacenamiento de respaldo.

Ambos tipos de memoria comparten las siguientes propiedades:

1. Velocidad: expresada en términos de el tiempo de acceso (el tiempo promedio requerido para alcanzar una localidad de memoria y obtener su contenido) y el tiempo de ciclo (que determina qué tan rápido se puede acceder la memoria continuamente, incluyendo el tiempo necesario para recuperarse después de un acceso).
2. Tipo de acceso: si los datos pueden accederse directamente o solo después de pasar otros datos.
3. Densidad y capacidad: indica cuántos bits pueden almacenarse por unidad de memoria.

4. Volatilidad: representa el tiempo que puede la memoria es capaz de retener los datos en una forma legible. Un disco magnético, por ejemplo, no es volátil mientras que una memoria semiconductora es comúnmente volátil, es decir, puede perder su contenido si se elimina la alimentación eléctrica.
5. Costo por componente: generalmente se mide en costo por bit; como regla general, mientras más veloz es un dispositivo de memoria, es más caro.
6. Disipación de energía: qué tanta energía requiere el dispositivo para funcionar, y qué tanto calor produce; esto tiene implicaciones de confiabilidad y costo.

En conjunto, el desempeño de un sistema de cómputo se relaciona directamente con su tiempo de ejecución, dado por:

$$t = IC \times CPI \times CP \quad (4.1)$$

donde IC es el número de instrucciones realizadas, y CPI es le tiempo promedio de ciclos de reloj por instrucción (este concepto se discute más en el Capítulo 5). CP representa el valor del periodo del reloj, relacionado con su frecuencia de operación. El desempeño de una computadora depende de la interfaz entre el procesador y su memoria. Si la interfaz no es la correcta, se incrementa significativamente el CPI . Dos parámetros caracterizan la interfaz procesador-memoria.

Ancho de banda, que es la tasa con la que el sistema de memoria puede servir solicitudes del procesador multiplicado por el ancho (en bits) de la memoria.

Latencia, que es el tiempo entre el inicio de de una solicitud de memoria y su terminación.

La velocidad de una computadora depende fuertemente del desempeño de su memoria.

La siguiente tabla resume las diferencias entre la memoria central y la memoria auxiliar, respecto a sus ventajas y desventajas.

	ventajas	desventajas
memoria central	rápida físicamente pequeña directamente direccionable	volátil cara pequeña capacidad
memoria auxiliar	barata persistente gran capacidad	físicamente grande lenta

4.2.2. El compromiso desempeño/precio

Mientras más rápido sea el procesador, más rápido deben accesarse y entregarse los datos e instrucciones. Un sistema de memoria perfecto sería aquél que puede proveer inmediatamente cualquier dato o instrucción solicitado por el procesador; sin embargo, las tres características principales de las memorias (capacidad, velocidad y costo) se encuentran en oposición directa. Un compromiso de rapidez-costo se aplica cuando se diseña un sistema de cómputo.

Los sistemas modernos de cómputo resuelven el compromiso entre rapidez y costo mediante implementar una **jerarquía de memoria** organizada en varios niveles, cada uno más pequeño, rápido y caro por byte que el siguiente. El objetivo es proveer un sistema de memoria con un costo casi tan bajo como el nivel más barato, y velocidad casi tan rápida como el nivel más rápido.

El concepto de jerarquía de memoria aprovecha dos principios:

1. El principio de localidad de referencia en el tiempo: las palabras de memoria que han sido accesadas tienen muchas posibilidades de accesarse de nuevo rápidamente.
2. El principio de localidad espacial de estados: si una palabra se accesa, las palabras vecinas son muy probables de ser accesadas pronto y después.

A un nivel de tecnología dado, las memorias más rápidas tienen una densidad y costo por bit substancialmente más bajo. Las jerarquías de memoria de las computadoras de propósito general modernas suelen contener lo siguiente:

- registros: la memoria más rápida y pequeña;
- memoria caché;
- memoria principal;
- memoria virtual;
- memoria auxiliar: la memoria más grande y lenta.

Un acceso de memoria se maneja mediante la memoria más rápida disponible, que guarda los datos requeridos. Los principios de localidad aseguran que el acceso a memorias más lentas y grandes sea relativamente poco frecuente. La Figura 4.1 muestra los componentes de una jerarquía de memoria con otros subsistemas.

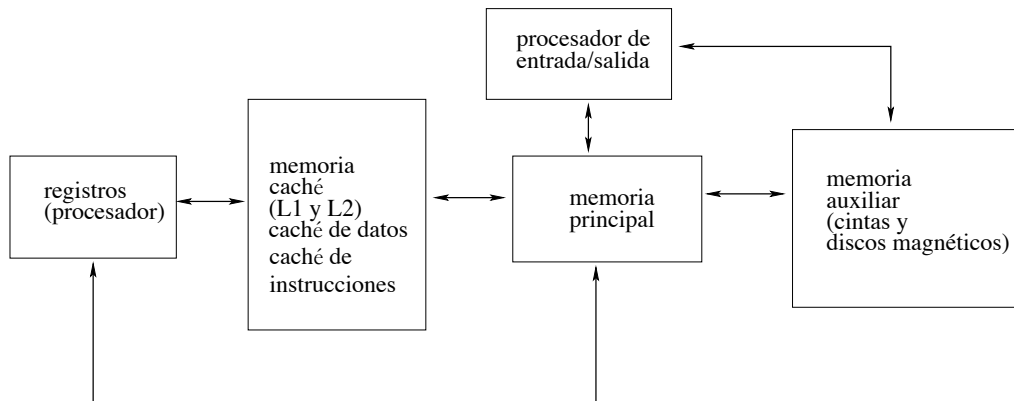


Figura 4.1: Una jerarquía de memoria.

Los componentes de la jerarquía de memoria se describen a continuación.

4.3. Registros y memoria caché

Los **registros** son un almacenamiento semiconductor pequeño y rápido contenido en el procesador. El compilador (que se discute en mayor detalle más adelante) es responsable de administrar su uso, decidiendo qué valores deben mantenerse en los registros disponibles a cada paso de la ejecución de un programa.

La **memoria caché** es una memoria semiconductora pequeña, de muy alta velocidad, localizada cerca del procesador (algunas veces, dentro del mismo), que guarda las instrucciones y datos más recientemente accedidos. Esta memoria incrementa la velocidad de procesamiento al hacer que los programas y datos estén rápidamente disponibles para el procesador.

Insertando la memoria caché entre el procesador y la memoria, y usándola para almacenar de forma temporal los datos e instrucciones más frecuentemente necesarios en las operaciones actuales y programas en ejecución, la velocidad de procesamiento se puede incrementar considerablemente. La memoria caché puede tener un tiempo de acceso de 5 a 10 veces más pequeño que el tiempo de acceso de la memoria principal, y su velocidad está más aproxima a la de los componentes del procesador. El éxito de la caché en acelerar la actividad del procesador se debe a la propiedad de localidad de referencia.

La operación de la memoria caché es como sigue:

1. Cuando el procesador necesita acceder a la memoria, se revisa primero a la memoria caché.
2. Si la información necesaria (datos o instrucciones) se encuentran en la memoria caché, el procesador la lee.
3. Si la información no se halla en la memoria caché, se accesa a la memoria principal.
4. Un bloque de palabras, entre las cuales se encuentra la palabra necesitada, se copia se la memoria principal a la memoria caché.

Cuando el procesador encuentra la información necesaria en la memoria caché, se le llama un acierto. Cuando no sucede, se le llama una pérdida, y la información debe traerse de la memoria principal. La razón de el número de aciertos entre el número de accesos del procesador a la memoria se conoce como razón de aciertos. Algunas computadoras tienen una razón de aciertos cercana al 0.9.

Las memorias caché varían ampliamente en términos de su tamaño y organización, y puede haber más de un nivel de caché en la jerarquía. Muchas de las cachés actuales tienen una estructura de dos niveles, con la más cercana al procesador llamada L1 y el siguiente nivel llamado L2.

4.4. Memoria principal y virtual

La **memoria principal** ocupa una posición central, ya que puede comunicarse tanto con el procesador como con la memoria auxiliar mediante un procesador de entrada/salida. Es relativamente rápida, y se utiliza para almacenar datos e instrucciones durante la operación de la computadora. La memoria principal satisface la demanda de los cachés y sirve, junto con el procesador de entrada/salida, como interfaz con la memoria auxiliar.

La **memoria virtual** es un mecanismo de *hardware* y *software* mediante el cual los dispositivos de almacenamiento auxiliar pueden hacerse aparecer como memoria principal adicional, en lo que toca a la programación.

La memoria virtual se utiliza para dar la apariencia de una memoria más grande que la memoria principal. Esto se logra mediante crear un espacio virtual de direcciones, y cada programa solo percibe una memoria del tamaño del espacio virtual de direcciones. Dentro de este espacio virtual de direcciones, la memoria principal actúa como una memoria caché de una imagen de la memoria almacenada en la memoria auxiliar. La memoria virtual depende de soporte tanto de *hardware*, de la Unidad de Administración de la Memoria (*Memory Management Unit* o MMU), y de *software*, que mapea cada dirección virtual de cada programa en una dirección física real. Más información al respecto se da en el Capítulo 7.

Cuando son necesarios por el procesador los datos o instrucciones que no se encuentran en la memoria principal, se les trae desde la memoria auxiliar. La memoria auxiliar almacena aquellas partes de los programas que no se usan en el momento presente por el procesador. Debe notarse que el procesador no tiene acceso directo a la memoria auxiliar. La memoria auxiliar es mucho más lenta, con tiempos de acceso 1000 veces mayor que la memoria principal.

Los dispositivos de memoria auxiliar más comúnmente utilizados en sistemas de cómputo son los discos y cintas magnéticos. Otros componentes muy utilizados son la memoria de burbuja magnética y los discos ópticos. La memoria auxiliar se organiza normalmente en registros o bloques.

4.5. Tipos de memorias

Esta sección detalla la implementación física de varios dispositivos de memoria cuyas propiedades lógicas se presentan en la sección anterior.

4.5.1. Memorias semiconductoras

Los dispositivos de memoria semiconductora son circuitos integrados como los procesadores. Dependiendo del tamaño total del sistema de memoria de una computadora, puede haber apenas algunos o muchos de estos circuitos integrados. En seguida, se introduce el uso de dos clases de memoria semiconductora, como se han mencionado en el Capítulo 3.

4.5.2. Memoria de acceso aleatorio (RAM)

Las memorias de acceso aleatorio (RAM), construidas mediante circuitos semiconductores, son utilizadas como la memoria principal de la mayoría de las computadoras digitales. Las memorias se organizan en palabras. Cada palabra consta de un número fijo de bits, que pueden tomar los valores binarios de 0 y 1. Por ejemplo, una palabra de memoria puede ser de 8 bits de longitud. Una palabra puede representar un número dentro de un cálculo, o puede representar una instrucción

en un programa. De hecho, una operación típica que involucra a la memoria es la suma de dos valores numéricos contenidos en dos palabras con diferentes direcciones en memoria. El resultado puede a la vez almacenarse en una tercera dirección.

La localidad de una palabra en una memoria se conoce con el nombre de dirección. Cuando se almacenan datos en cada bit de una palabra, se dice que los datos almacenados se escriben en memoria. Similarmente, cuando los datos almacenados en cada bit de una palabra se extraen de la memoria, se dice que se leen de la memoria. La palabra que ha sido escrita o leída de la memoria, para tal efecto, se dice que ha sido direccionada, es decir, se encuentra su localidad en la memoria ya sea para modificarse (escribirse) o inspeccionarse (leerse).

Aun cuando hay variaciones al respecto, resulta conveniente suponer que cada bit de una palabra se almacena en un flip-flop, es decir, que hay un flip-flop por separado cuyo estado determina el almacenamiento de cada bit en la memoria. En computadoras pequeñas, las palabras comúnmente contienen 8, 16 o 32 bits. En grandes computadoras, las palabras contienen 16, 32, 64 o más bits. En general, se intenta que la memoria sea capaz de almacenar un número grande de palabras.

De hecho, el almacenamiento de un solo bit involucra más que un simple flip-flop. Debe haber previsiones indicando si la información binaria debe ser escrita o leída. En general, existe una terminal del circuito de memoria etiquetado *read/write*. Cuando se coloca un valor de 1 en tal terminal, la memoria almacena el dato binario (se escribe en memoria); cuando se coloca un valor de 0 en la terminal, la memoria provee el dato binario (se lee la memoria). También debe tomarse en consideración el direccionamiento de cada palabra; es decir, que al leer o escribir una palabra en memoria, debe proveérsele una dirección de memoria.

Una celda binaria es la unidad básica de memoria, y se utiliza para almacenar un solo bit. Además de contener el valor binario, también considera señales binarias para su lectura y escritura, así como para su direccionamiento. La Figura 4.2 muestra el diagrama de bloque de una celda binaria. Nótese que cuenta con cinco terminales: una entrada (para la escritura del bit), una salida (para la lectura del bit), una terminal *read/write*, una terminal de direccionamiento y una entrada de reloj.

La Figura 4.2 también muestra el circuito digital de una celda de memoria usando un flip-flop RS. Considérese su operación. Para que el circuito funcione, debe tenerse un valor 1 en la terminal A (de direccionamiento). De no ser así, entonces al menos una entrada de las compuertas AND **c**, **d** y **e** tiene un valor

de 0, y por lo tanto, $R = 0$ y $S = 0$, por lo que el flip-flop no puede cambiar su estado: no puede escribirse en la celda binaria. Además, la salida tiene valor 0. Por lo tanto, el contenido de la celda de memoria tampoco puede leerse.

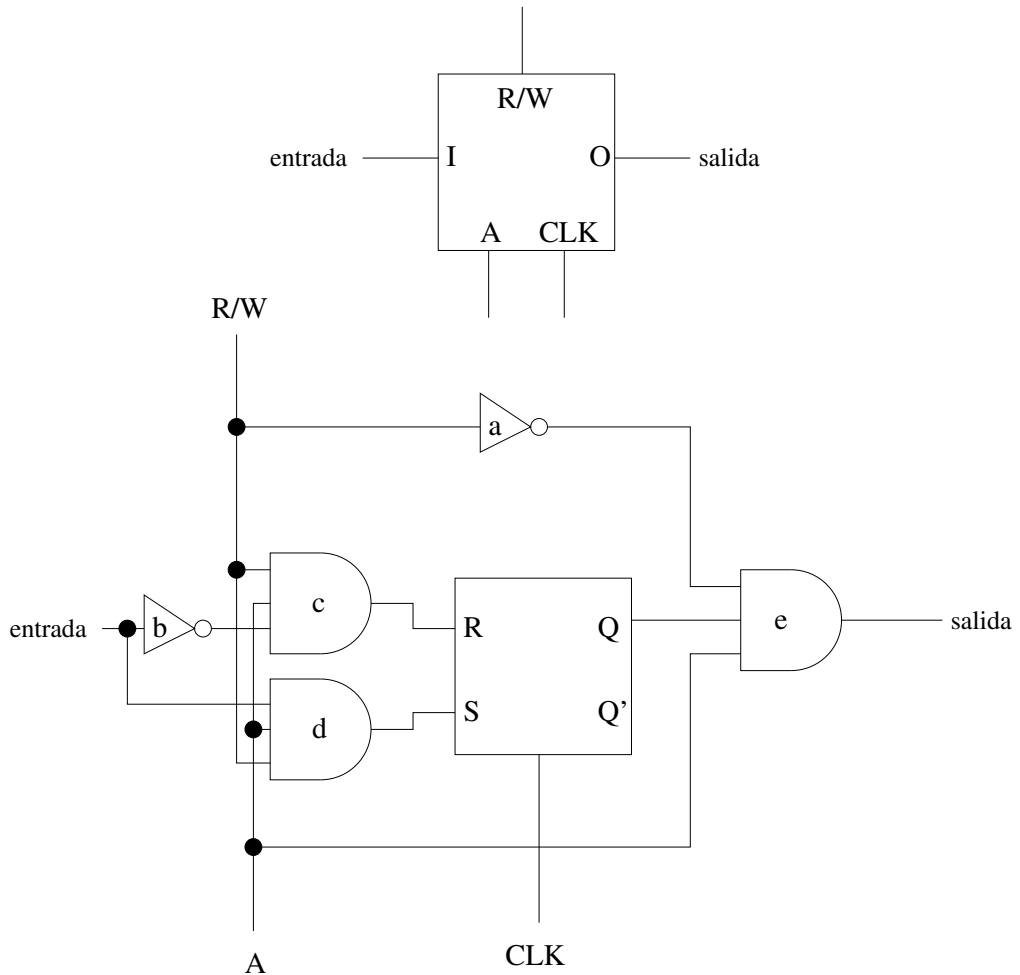


Figura 4.2: Celda binaria en diagrama de bloque e implementación usando flip-flop RS.

Ahora bien, supóngase que la celda ha sido direccionada, es decir, hay un valor de 1 en su terminal A. También supóngase que se desea escribir un bit en ella, de modo que se coloca un valor de 1 en su terminal R/W. De este modo, dos de las entradas a las compuertas AND **c** y **d** tienen valor 1. Supóngase que se desea almacenar (escribir) un 1. Entonces, se coloca tal valor en la terminal de entrada. De este modo, las tres entradas de la compuerta AND **d** tienen valor 1, lo que hace que $S = 1$; sin embargo, al menos una entrada a la compuerta AND **c** tiene valor de 0, por lo que $R = 0$. Dado que $R = 0$ y $S = 1$, el flip-flop cambia cuando ocurra el pulso de reloj, poniéndose en estado $Q = 1$. De este modo, se ha almacenado un 1 en la celda.

De manera similar, si se tienen valores de 1 en las terminales A y R/W, y la entrada tiene valor 0, entonces $R = 1$ y $S = 0$, por lo que se almacena un 0 en la celda de memoria.

Supóngase ahora que se desea leer el bit almacenado en la celda de memoria. Hay un valor de 1 en la terminal A, pero un valor 0 en la terminal R/W, por lo que al menos una entrada de las compuertas AND **c** y **d** tiene valor 0, y entonces $R = 0$ y $S = 0$, por lo que la información almacenada no cambia; sin embargo, dos entradas a la compuerta AND **e** tienen valor de 1, por lo que la salida tendrá el valor Q , que corresponde al bit almacenado en la celda binaria.

Una memoria real consiste de muchas celdas binarias organizadas en palabras. La Figura 4.3 muestra una memoria compuesta de cuatro palabras de cuatro bits cada una (en general, el número de palabras y el número de bits por palabra no necesariamente son iguales). Nótese la notación usada. Se han añadido subíndices a la terminal A, de dirección, de cada celda binaria. Estos subíndices indican el número de palabra y la posición del bit en la palabra que la celda ocupa en la memoria. Por ejemplo, los subíndices 1,0 indican la palabra número 1, bit número 0. Del mismo modo, 2,1 indica palabra 2, bit 1.

Cada palabra se direcciona utilizando una terminal de dirección. Nótese que todas las terminales A de las celdas de memoria pertenecientes a la misma palabra están conectadas juntas. Así, si un valor 1 se coloca en la terminal de dirección 0 y se colocan valores de 0 en las demás terminales, entonces se direccionan todas las celdas de la palabra 0. Ninguna otra de las celdas de memoria se direccionan. De manera similar, si se coloca un valor 1 en la terminal de dirección 2 con valores 0 colocados en las demás terminales de dirección, entonces todas las celdas de la palabra 2 se direccionan y ninguna otra celda en la memoria.

Recuérdese que cuando una celda de memoria no se direcciona, no pierde su información almacenada.

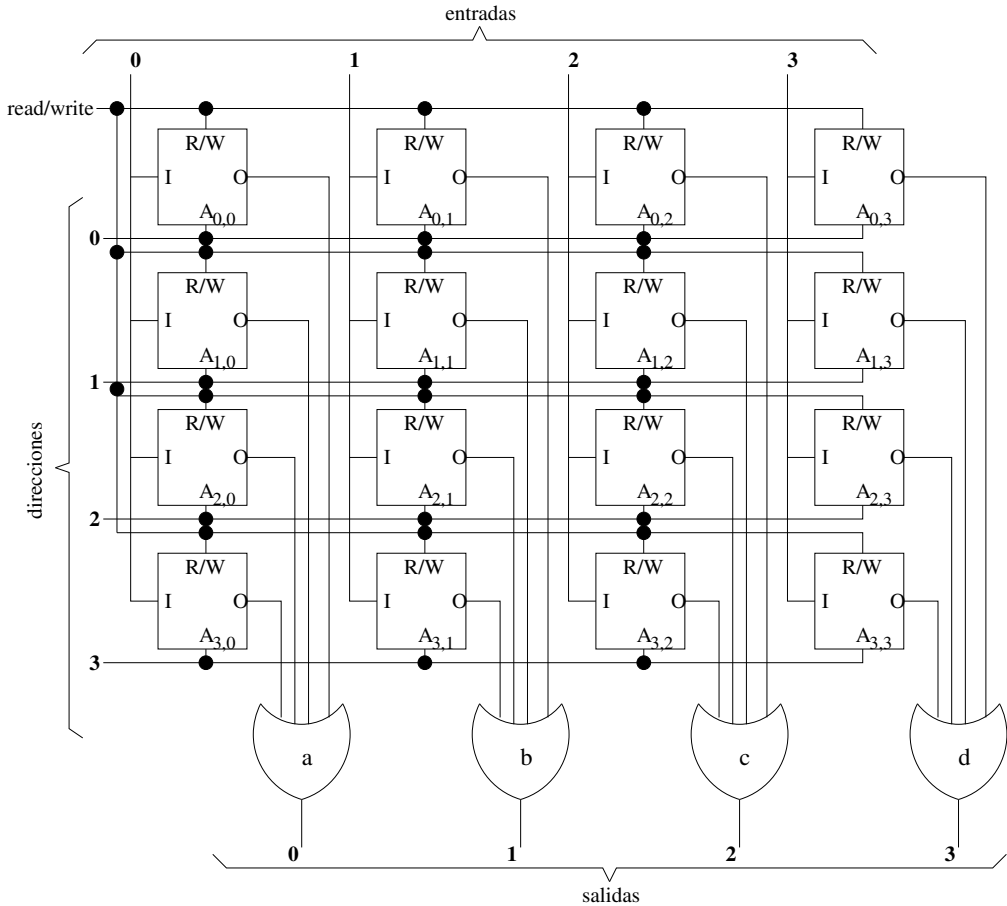


Figura 4.3: Una memoria de cuatro palabras de cuatro bits. La señal de reloj se omite por claridad.

Nótese que el número de terminales de entrada es igual al número de terminales de salida, y al número de bits en una palabra. Ya que se utilizan palabras de cuatro bits, hay cuatro terminales de entrada. Las terminales *read/write* de todas las celdas están conectadas. Si se desea almacenar una palabra (de cuatro bits)

en una localidad de memoria cuya dirección es 2, entonces debe colocarse un 1 en la terminal *read/write* y en la terminal de dirección número 2 (se debe colocar un valor de 0 a todas las demás terminales de dirección). Después del siguiente pulso de reloj, el estado de todos los flip-flops en la palabra 2 será igual a los valores colocados en sus respectivas terminales de entrada. Por lo tanto, la palabra de cuatro bits de entrada ha sido almacenada en la dirección 2. Nótese que todas las terminales de reloj deben conectarse a un reloj maestro. Tal conexión se omite en el diagrama de la Figura 4.3, para evitar sobrecargar el diagrama.

En seguida se describe cómo se lee una palabra de una localidad de memoria. Hay una terminal de salida por cada bit de todas las palabras. Así, para esta memoria, hay cuatro terminales de salida. La salida de una celda de memoria será 0 a menos que haya tanto un valor de 1 en su terminal de dirección, como un 0 en la terminal *read/write*. Supóngase que se coloca un valor de 1 en la terminal de dirección 3 y 0 en todas las demás terminales de dirección. Además, un 0 se coloca en la terminal *read/write*. Obsérvese la compuerta OR **a**. Tiene cuatro entradas, cada una de las cuales corresponde al bit 0 de cada palabra en la memoria; sin embargo, las salidas del bit 0 de las celdas de memoria 0, 1 y 2 tienen un valor de 0, ya que estas palabras no están direccionadas. Por lo tanto, la salida de la compuerta OR **a** es igual al valor almacenado en el bit 0 de la palabra 3. Así, cuando se coloca un valor 0 en la terminal *read/write*, las terminales de salida tendrán los valores de los bits almacenados en la palabra que haya sido direccionada.

Este tipo de lectura se conoce con el nombre de no destructiva ya que no remueve o borra la información almacenada en la memoria. La palabra permanece almacenada hasta que se sobreescribe. Una palabra almacenada permanece sin cambios a menos que se coloque un valor de 1 en la terminal *read/write*, y que tal palabra sea direccionada. En tal caso, los datos binarios colocados en las terminales de entrada reemplazan a la palabra almacenada.

En la memoria de la Figura 4.3 se tiene una terminal de dirección separada por cada palabra en la memoria. Si esta memoria fuera fabricada en un circuito integrado, debería tener una terminal por cada dirección. Ya que las memorias normalmente contienen una cantidad relativamente grande de palabras, se requeriría de igual número de terminales para su direccionamiento. Esto es, en la realidad, indeseable. El circuito integrado tendría que fabricarse demasiado grande para albergar todas las terminales necesarias, introduciendo a la vez otros problemas relacionados, como aquéllos debidos a la conectividad y al costo de fabricación.

Es por ello que se hace necesario hallar una forma en que se pueda realizar el direccionamiento de una gran cantidad de localidades de memoria, pero sin incluir un gran número de terminales.

Para reducir el número de terminales externas, la dirección puede darse como un número binario. Por ejemplo, en el caso de haber tres terminales de dirección, éstas pueden usarse para indicar hasta ocho direcciones: 000, 001, 010, 011, 100, 101, 110 y 111. En forma general, si hay N terminales de dirección, entonces se pueden direccionar hasta 2^N localidades de memoria. Para la memoria de la Figura 4.3, se requieren entonces dos terminales de dirección. Aunque esto no parece mucho, para cantidades grandes de palabras en memoria, el ahorro es más dramático. Por ejemplo, con 16 terminales para direccionamiento, se puede direccionar hasta 65,536 palabras (o simplemente, 64k palabras).

Dentro del circuito integrado, cada palabra debe tener una terminal de dirección separada como se indica en la Figura 4.3, de tal modo que es necesario contar con una lógica integrada en el circuito tal que convierta las direcciones de entrada en binario a señales individuales para las terminales de dirección. En la Figura 4.4 se muestra un circuito que controla cuatro terminales internas a partir de dos entradas de dirección binarias.

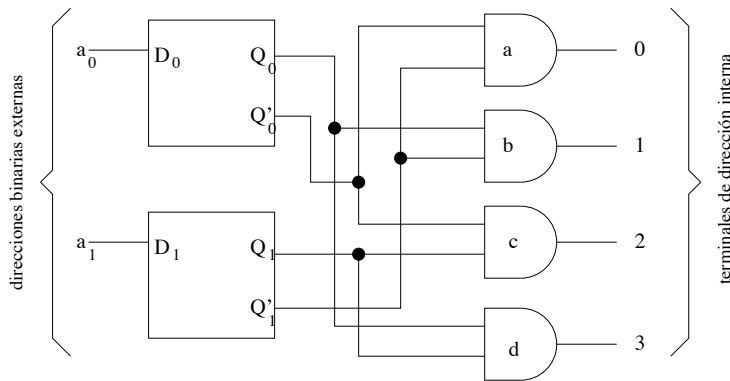


Figura 4.4: Un decodificador de dirección. Las señales de reloj se omiten.

La dirección binaria se introduce por la terminales a_0 y a_1 . La forma del número binario es a_1a_0 . Si $a_1 = 1$ y $a_0 = 0$, entonces la dirección binaria es 10_2 . Nótese que estas entradas controlan el estado de los flip-flops mediante conectarse

a las entradas D_0 y D_1 . Después de un pulso de reloj, se tiene que $Q_0 = a_0$ y $Q_1 = a_1$. Tales valores se pasan a las compuertas AND. Por ejemplo, la compuerta AND **a** sólo puede tener una salida con valor 1 si $Q_0 = Q_1 = 0$, es decir, si $a_0 = a_1 = 0$ durante el pulso anterior. En el caso de este ejemplo, se tiene que la compuerta AND **c** es la única que tiene una salida 1 para $Q_1 = 1$ y $Q_0 = 0$, lo que coloca un valor de 1 en la terminal interna de dirección 2 ($10_2 = 2_{10}$). Todas las demás terminales de dirección interna tienen un valor 0. Por tanto, el circuito de la Figura 4.4 se conoce como decodificador de dirección.

De hecho, el decodificador de la Figura 4.4 no requiere necesariamente contener flip-flops. Esto se utilizan con el sólo propósito de almacenar la dirección, dado que en ocasiones, la dirección binaria puede estar presente por un periodo corto de tiempo, aun cuando se requiera contar con la dirección por un tiempo mayor. En tal caso, los flip-flops resultan útiles.

Casi invariablemente, las computadoras usan RAM para su memoria principal de alta velocidad (y usan memorias más lentas para su memoria auxiliar). La RAM se utiliza para almacenar las instrucciones y datos que están sujetos a cambio, y su contenido se pierde si se apaga la computadora.

Los dispositivos de RAM pueden clasificarse en:

- estática (*Static RAM* o SRAM), que generalmente se utiliza para sistemas de caché;
- dinámica (*Dynamic RAM* o DRAM), que se utiliza para los sistemas de memoria principal.

La SRAM consiste esencialmente en flip-flops que almacenan la información binaria, y la información permanece disponible mientras se aplique corriente eléctrica a la unidad. La DRAM almacena la información binaria en forma de cargas eléctricas, y necesita refrescarse periódicamente en términos de milisegundos. Tiene una capacidad de almacenamiento mayor por circuito integrado, pero la SRAM tiene ciclos de lectura/escritura más cortos, y necesita menos *hardware* adicional de apoyo para sus circuitos integrados.

4.5.3. Memoria de solo lectura (ROM)

La memoria de solo lectura (*Read Only Memory* o ROM) es de acceso aleatorio, pero con la diferencia que solo almacenan información binaria. Una vez escrita,

permanentemente, no puede reescribirse por la computadora donde se encuentra instalada.

Las ROM se usan para almacenar instrucciones que permanentemente residen en la computadora, junto con otros datos que no cambian de valor durante su uso normal. Un uso típico de la ROM es almacenar el cargador inicial (*bootstrap loader*, también conocido como *Basic Input/Output System* o BIOS), que es un programa que inicializa las funciones de la computadora y persiste cuando la computadora se apaga. Este cargador inicial trae una porción del sistema operativo del disco a la memoria principal, da el control al sistema operativo y prepara a la computadora para uso general.

La memoria RAM descrita en la sección anterior se diseña para poder escribir y leer información binaria de ella; sin embargo, en muchas ocasiones no es necesario cambiar la información binaria almacenada una vez que ésta se introduce a la memoria. Es decir, la información binaria se escribe una vez, y no se cambia ni reescribe. Para este tipo de comportamiento de la memoria, resulta complicado el uso de la RAM. Es por ello que para aplicaciones en las que el contenido de la memoria no cambia durante su funcionamiento, se prefiere el uso de memoria ROM. La memoria ROM puede utilizarse para almacenar información binaria que se utiliza repetitivamente, como tablas de datos o instrucciones comúnmente utilizadas. La memoria ROM es no volátil: la información que se almacena en ella no desaparece cuando se le deja de suministrar energía eléctrica.

Estructura de la ROM básica

Para describir la operación de una memoria ROM, se utiliza la estructura de conexiones que se muestra en la Figura 4.5. Las terminales de salida se conocen como bits de salida. Por el momento, considérese que las terminales de palabra son entradas, y que solo se permite colocar un valor de 1 en solo una de las terminales de entrada a la vez. El valor lógico de las demás terminales se mantiene en 0. En respuesta de un valor 1 en una terminal de palabra, una palabra se produce en las terminales de salida.

Para describir su operación, supóngase que la ROM en la Figura 4.5 debe realizar la siguiente función: como tiene 10 terminales de palabra, cada una corresponde a un dígito decimal. Cuando un valor de 1 se coloca en una terminal de palabra, su valor binario aparece en las terminales de salida. Por ejemplo, si se coloca un 1 en la terminal de palabra 1, entonces la salida es $a_3 = 0$, $a_2 = 0$

$a_1 = 0$ $a_0 = 1$ ($0001_2 = 1_{10}$). De forma similar, si se coloca un 1 en la terminal de palabra 6 se tiene que $a_3 = 0$, $a_2 = 1$ $a_1 = 1$ $a_0 = 0$ ($0110_2 = 6_{10}$).

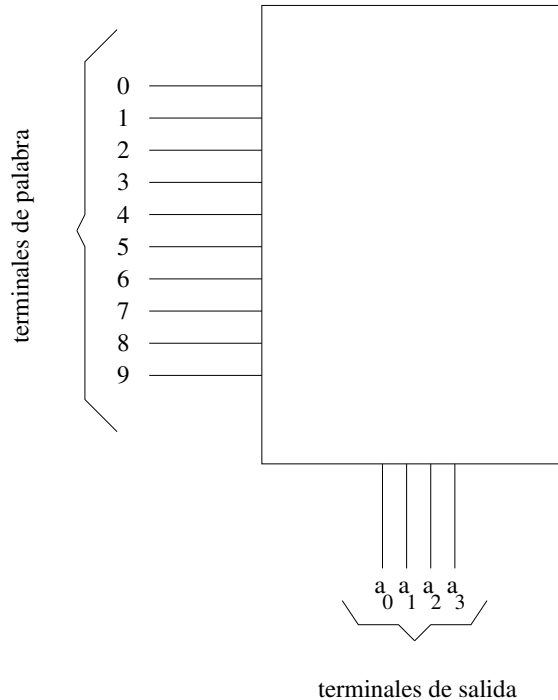


Figura 4.5: Conexiones de una ROM.

Los circuitos en el interior de la ROM conectan las terminales de salida con las terminales de palabra apropiadas. Estas conexiones no se hacen de forma ordinaria con alambre, sino más bien se utilizan circuitos semiconductores que previenen de la conexión directa entre circuitos de entrada y salida.

Una entrada real de una ROM consiste de un decodificador de dirección como el descrito en la Figura 4.4. Ahora, cada terminal de dirección del decodificador se conecta a la terminal de dirección correspondiente de la ROM. De tal forma, la ROM se utiliza de la misma manera que una RAM ordinaria, es decir, se le provee de una dirección a la ROM y ésta produce una palabra como salida (Nótese que puede considerarse que la ROM es una forma de RAM).

Tipos de ROM

Hay varios tipos de memorias ROM. En la ROM ordinaria, la información binaria se almacena durante la fabricación del circuito. El usuario provee al fabricante de la información binaria que requiere, y la ROM se fabrica bajo tal especificación. Una vez escrita, la información binaria no puede perderse ni modificarse. Está literalmente grabada en piedra.

La ROM programable (*programmable ROM* o PROM) es otra forma de ROM en la cual, después de su fabricación, el usuario puede escribir información binaria una vez, y a partir de ello, la información binaria queda permanentemente almacenada, y resulta muy difícil cambiarla de nuevo.

Cuando una PROM sale de la fábrica, viene con todas las conexiones cerradas entre sus entradas y sus salidas. Si se coloca un valor de 1 en cualquier terminal de entrada, se genera un 1 en todas las terminales de salida, es decir, la PROM no contiene información, y todos sus bits se encuentran con valor 1. Para poder programarla, la PROM cuenta con conexiones fundibles. Una conexión fundible actúa como un fusible común. Cuando se pasa suficiente corriente eléctrica por él, se derrite, abriendo el circuito. El usuario de la PROM, entonces, debe fundir aquellas conexiones donde la conexión entre entrada y salida no se desea. Esto se logra mediante pasar una corriente eléctrica relativamente grande entre las terminales de entrada y salida donde no se desea la conexión, lo que representa incorporar 0's de modo que la PROM se va llenando con la información binaria requerida por el usuario.

Como ejemplo, considérese que la ROM de la Figura 4.5 fuera una PROM. Durante su programación, todas las conexiones fundibles entre la terminal de dirección 0 y las salidas tendrían que romperse, de tal modo que no hubiera salida en las terminales fundidas cuando a la entrada se coloca un 1 en la terminal de dirección 0. De forma similar, la conexión entre la terminal de dirección 3 y las salidas a_3 y a_2 deben romperse, ya que al aplicar un valor de 1 a la terminal de dirección 3 se espera que a la salida $a_3 = a_2 = 0$ y $a_1 = a_0 = 1$ ($0011_2 = 3_{10}$). Procediendo de esta forma, toda la PROM puede programarse. Una vez programada, no puede volverse a programar. Se pueden romper algunas conexiones adicionales, pero aquéllas que han sido rotas no puede restaurarse.

La ROM programable y borrable (*erasable PROM* o EPROM) es un tercer tipo de ROM, que tiene la posibilidad de ser modificada después de su fabricación, y borrarse al ser afectada por una corriente eléctrica (*electrically erasable PROM*

o EEPROM) o por su exposición a rayos ultravioleta (*ultraviolet erasable PROM* o UVEPROM); sin embargo, los procesos de escritura y borrado son extremadamente lentos comparados con la velocidad de lectura y escritura en memoria RAM. Por lo tanto, todos los tipos de ROM se utilizan casi siempre como almacenamiento permanente de información binaria, excepto en los raros casos en que se requiere su reprogramación.

La conexión entre las salidas y entradas en una PROM borrable es un circuito semiconductor que inicialmente no provee de tal conexión, sino que se modifica su estado por efecto de una corriente eléctrica intensa (o en ocasiones un voltaje alto) aplicado entre las terminales de entrada y salida. Estas terminales quedan conectadas en una forma (casi) permanente.

Las PROM borrables se programan mediante atrapar una carga en áreas de aislamiento eléctrico. Cuando se aplica la corriente intensa, la región de aislamiento se rompe temporalmente. En tal circunstancia, una carga se atrapa en la región. Cuando se quita la corriente, la capa aislante se reestablece, pero ahora cuenta con la carga eléctrica atrapada. La presencia de tal carga se utiliza para encender los dispositivos electrónicos que conectan las terminales de entrada y salida de la PROM.

Una PROM de este tipo puede borrarse mediante su exposición a luz ultravioleta (UVEPROM). Esto causa que la capa aislante se rompa temporalmente, liberando la carga atrapada. Los niveles de luz ultravioleta necesarios para borrar una memoria son tan altos que la exposición a una luz ordinaria no es capaz de borrarla. No se requiere introducir corrientes intensas cuando se borra este tipo de memorias. Así, la PROM puede volverse a programar con el procedimiento normal. Nótese que borrar y programar son dos operaciones muy lentas, comparadas con la escritura y lectura de una RAM. Es por esto que las PROM no son frecuentemente reprogramadas.

Existen algunos circuitos especiales controlados por microprocesador que se utilizan para programar PROMs. En los más sofisticados, la información binaria puede almacenarse primero en una RAM, y luego escribirse en la ROM para programarla.

4.5.4. Paralelización de dispositivos de memoria

El número de palabras, o el número de bits por palabra, que se almacenan en una memoria semiconductora dada es muy comúnmente insuficiente para almacenar

todos los datos e instrucciones binarias de un programa dado; sin embargo, es posible incrementar la capacidad de la memoria semiconductora mediante componer una memoria semiconductora más grande a partir de acumular memorias pequeñas. De hecho, se tienen disponibles en el mercado algunos dispositivos de memoria que permiten incrementar la cantidad de memoria de una computadora. Al incremento de memoria para utilizarse como una sola memoria semiconductora en un sistema de cómputo se le llama paralelización de dispositivos de memoria. Esta técnica es muy utilizada por quienes cuentan con computadoras pequeñas para poder acceder a memorias de trabajo mayores. En tales casos, los dispositivos de memoria consisten de tarjetas impresas que cuentan con varias memorias y otros circuitos para su conexión. Alternativamente, el tamaño de la memoria puede incrementarse mediante añadirle memorias en forma de circuito integrado. A esto se le conoce como la paralelización de circuitos integrados de memoria.

La terminal *Chip Select* (CS)

Muchos dispositivos de memoria diseñados para paralelizarse frecuentemente tienen una terminal externa de direccionamiento llamada *chip select* (selección de circuito o CS). Esta terminal se utiliza para apagar el decodificador de dirección interno del dispositivo de memoria. La terminal CS funciona de la siguiente manera: si se le coloca un valor de 1, entonces el decodificador de dirección funciona como se muestra en la Figura 4.4, es decir, utiliza direcciones binarias externas para colocar un valor de 1 en la terminales de dirección internas. Si se coloca un valor de 0 en la terminal CS, entonces el decodificador de dirección no funciona. No importa qué valores se encuentren en las terminales externas de dirección, el decodificador no tendrá como salida ningún valor de 1.

El decodificador de dirección de la Figura 4.4 puede modificarse como se muestra en la Figura 4.6 para incluir una terminal CS. En tal caso, se añade una terminal extra de entrada a cada compuerta AND **a**, **b**, **c** y **d**, las cuales se conectan a la vez a la entrada CS. Cuando el valor de entrada en CS es 1, entonces el circuito de la Figura 4.6 funciona esencialmente como el circuito de la Figura 4.4. Por otro lado, cuando el valor en la terminal CS es 0, las salidas de todas las compuertas AND toman el valor de 0, por lo que ninguna palabra de memoria se direcciona.

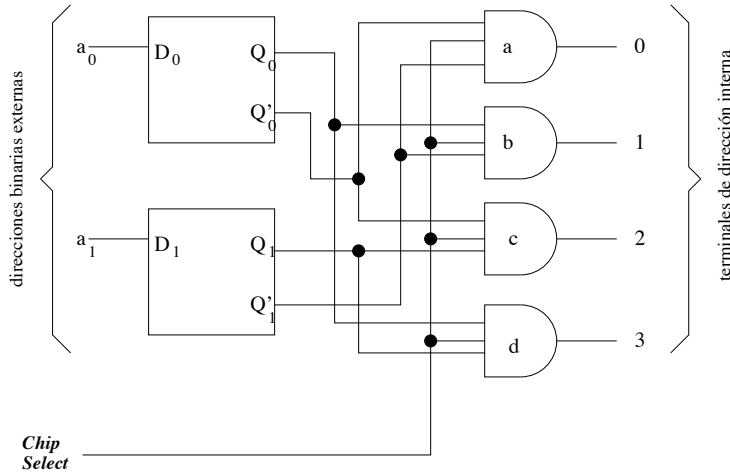


Figura 4.6: Modificación de un decodificador de dirección para contar con una terminal *chip select*.

4.5.5. La terminal *Output Enable* (OE)

Los dispositivos de memoria diseñados para la paralelización frecuentemente tienen una terminal de control de direcciones llamada *output enable* (habilitación de salida u OE). Cuando se coloca un valor de 1 en esta terminal, la memoria funciona de manera normal. Por otro lado, si se coloca un valor de 0, entonces la salida se desconecta efectivamente de las terminales de salida del dispositivo de memoria. Esto se conoce con el nombre de circuito triestado. En la Figura 4.3 se muestra cómo la salida de varias celdas binarias se conectan mediante compuertas OR a las terminales de salida de la memoria. El uso de OE elimina la necesidad de tales compuertas OR cuando las salidas de la memoria se paralelizan. Algunos dispositivos de memoria combinan las dos terminales, CS y OE, en una sola terminal *chip enable* (habilitación de circuito o CE).

En seguida, se describe cómo dispositivos de memoria pueden paralelizarse para incrementar el número de palabras en la memoria de una computadora. En la Figura 4.7 se muestra la paralelización de tres dispositivos de memoria. En realidad, se supone que cada dispositivo de memoria cuenta con una estructura similar a la memoria de la Figura 4.3, conjuntamente con un decodificador de

dirección con una terminal CS (Figura 4.6). Se supone que cada dispositivo de memoria tiene una terminal OE que se ha combinado con CS para contar con una sola terminal CE.

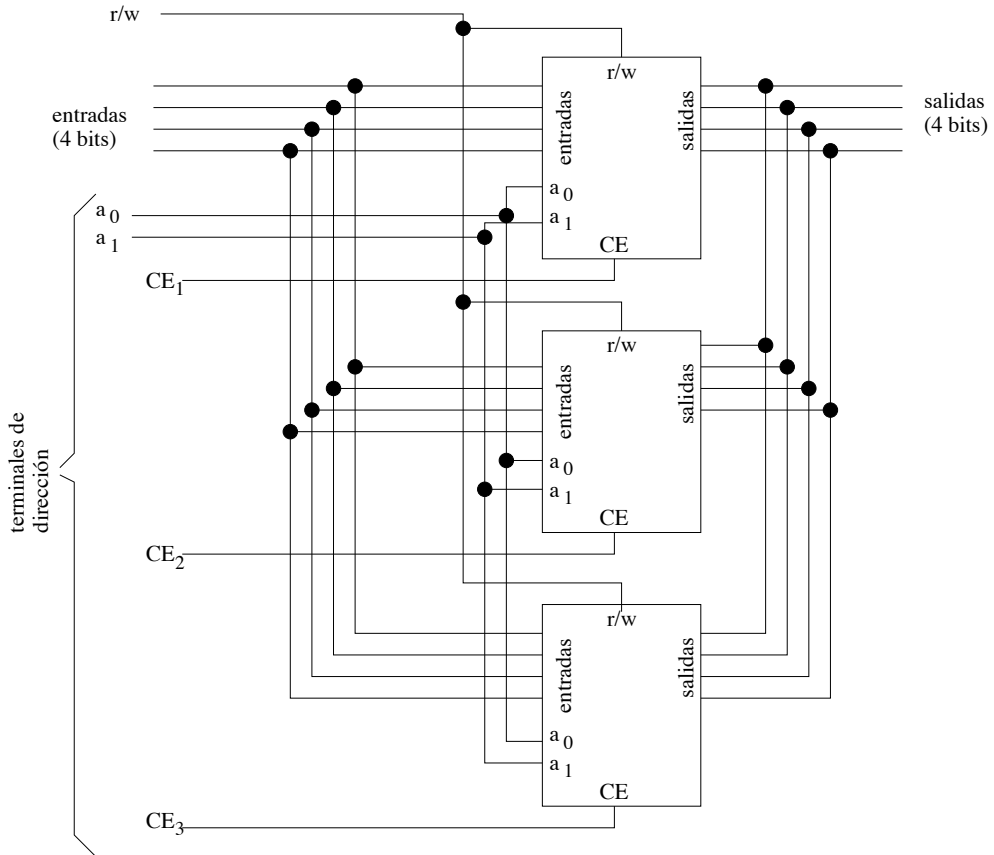


Figura 4.7: Una memoria de doce palabras construida mediante la paralelización de tres dispositivos de memoria de cuatro palabras.

Cada dispositivo de memoria contiene cuatro palabras de cuatro bits cada una. Nótese que las terminales de dirección a_0 y a_1 de todos los dispositivos de memoria se han conectado entre sí. De este modo, el mismo valor binario aplicado a estas dos terminales direcciona una palabra dentro de cada uno de los tres dispositivos

de memoria; sin embargo, nótese que las terminales CE de cada dispositivo no se encuentran conectadas entre sí. De tal modo, solo aquel dispositivo de memoria cuyo CE tenga un valor de 1 será direccionado. Por ejemplo, si se desea direccionar la palabra 3 del dispositivo de memoria 2, entonces se requieren los siguientes valores:

$$\begin{aligned}a_0 &= 1 \\a_1 &= 1 \\CE_1 &= 0 \\CE_2 &= 1 \\CE_3 &= 0\end{aligned}$$

Las terminales de entrada correspondientes se conectan juntas. Por ejemplo, todas las terminales de entradas para el bit 0 se conectan juntas. De forma similar todas las terminales para los bits 1, 2, etc., así como las terminales de entrada para la selección de lectura o escritura (*read/write*). Ahora bien, supóngase que se desea escribir en la palabra 3 del dispositivo de memoria 2. Los valores dados anteriormente se deben colocar en las terminales de dirección, mientras que un valor de 1 debe colocarse en la terminal *read/write*. Solo entonces, la palabra 3 de la memoria 2 puede direccionarse. Así, los valores de las terminales de entrada a la memoria se pueden almacenar en la localidad deseada.

En el caso de las terminales de salida, también es necesario conectar las terminales juntas. Por ejemplo, todos los bits 0 de las terminales de salida se conectan. Recuerdese que cuando el valor de la terminal CE de una memoria es 0, las terminales de salida se desconectan. Por tanto, si se desea leer una palabra almacenada en la dirección 3 de la memoria 2, se aplican los valores anteriores para el direccionamiento y se coloca un valor de 0 a la terminal *read/write*. Ya que CE_1 y CE_3 tienen valor 0, las salidas de sus respectivas memorias se encuentran desconectadas. Igualmente, dado que CE_2 tiene valor 1, las salidas de esta memoria se conectan a las terminales de salida. De este modo, la palabra deseada aparece en las terminales de salida.

En la Figura 4.7 se muestra la paralelización de tres memorias de cuatro palabras cada una. En general, las memorias que se paralelizan contienen muchas más palabras, y esas palabras pueden contener más bits. Nótese que entonces hay una buena cantidad de variantes. Por ejemplo, la operación de las terminales de control como CE también puede variar de fabricante a fabricante. Así, los detalles

reales para la conexión de memorias a fin de formar una memoria más grande y compleja puede variar respecto a los detalles que se han discutido hasta aquí, por lo que los detalles específicos de las memorias deben obtenerse de los manuales que proveen los fabricantes. Aun cuando no se ha mencionado, las memorias pueden paralelizarse para formar palabras con un mayor número de bits.

4.6. Dispositivos de almacenamiento auxiliar

Esta sección introduce varios tipos de dispositivos de almacenamiento auxiliar, cómo trabajan y para qué se utilizan.

4.6.1. Generalidades

Todos los tipos de memorias que se han discutido hasta este punto han sido memorias de acceso aleatorio. En tal tipo de memorias, cualquier dirección de memoria puede leerse o escribirse sin relación con ninguna otra dirección. Como se ha discutido anteriormente, esta es una memoria muy rápida. En esta sección, se mencionan otros tipos de memoria cuyo tiempo de respuesta es mayor, es decir, son más lentas para responder, pero que pueden proveer de mucho más espacio de almacenamiento a un costo mucho más bajo.

En realidad, estas formas más lentas de memoria pueden almacenar enormes cantidades de información binaria. Por ejemplo, pueden utilizarse para almacenar todos los registros de los cuentahabientes de un banco. Normalmente, esta cantidad de información es muy grande para almacenarse en la memoria principal, pero fácilmente puede guardarse en algunas cintas o discos. Cuando un programa que usa estos datos debe ejecutarse, sólo se requiere que los datos se transfieran de la cinta o disco a la memoria principal para ser procesados. El resultado puede ser de nuevo recolectado y almacenado en la cinta o disco.

Dado que se trata de información binaria, los programas también pueden almacenarse en cinta o disco. Cuando deben ejecutarse, se copian a la memoria principal. Las técnicas de almacenamiento masivo (y lento) que se discuten en esta sección resultan muy útiles para preservar datos y programas. También es común que este tipo de memoria se utilice como memoria principal, lo que hace la operación mucho más lenta, pero provee de una gran cantidad de espacio de memoria barata.

En la mayoría de los casos, las técnicas de almacenamiento masivas se basan en el uso de películas ferromagnéticas que almacenan datos. Su operación difiere al funcionamiento de la memoria principal en que aquí se efectúa realmente un movimiento, en el que la película ferromagnética se mueve respecto a un dispositivo o cabeza de lectura o escritura.

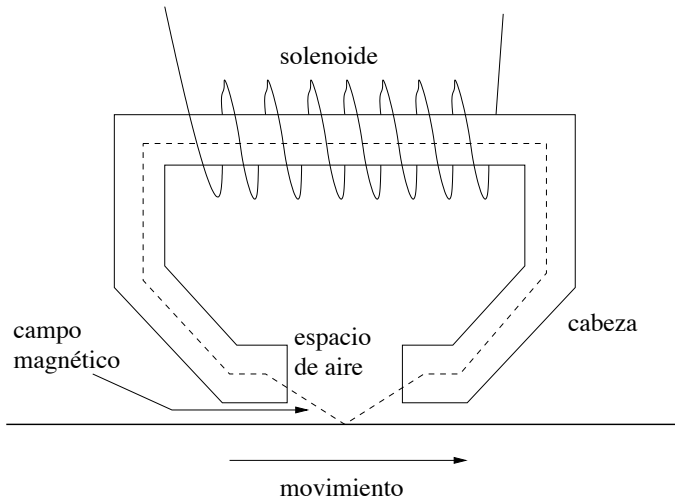


Figura 4.8: Diagrama de grabado en una película ferromagnética.

La Figura 4.8 muestra un dispositivo electromagnético que permite la grabación sobre superficies ferromagnéticas. En todos estos tipos de dispositivos, la película de material ferromagnético se monta sobre un mecanismo de soporte como una cinta o un disco, que se encarga de mover el material bajo la acción de una cabeza. Esta se fabrica de material magnético, y comúnmente cuenta con un solenoide conductor que se enreda en uno de sus extremos. En el otro extremo, se tiene un espacio de aire colocado muy cercanamente al material ferromagnético. Nótese que el diagrama de la Figura 4.8 es en realidad mucho más grande que una cabeza magnética verdadera. El espacio de aire, así como la distancia entre la cabeza y la película, son muy pequeños. Cuando una corriente eléctrica se pasa a través del solenoide, se genera un campo magnético en la cabeza. En el punto del espacio de aire, el campo toca la superficie ferromagnética de la película, magnetizándola. Dado que la película se mueve bajo la acción del campo magnético, los

pulsos de corriente que se apliquen al solenoide se reflejan como regiones cuyas moléculas tienen un tipo de arreglo magnético y regiones sin tal arreglo. La información, entonces, se ha escrito sobre la película. Nótese que se puede cambiar la dirección de la corriente, lo que invierte la polaridad de cada región magnética.

Ahora supóngase que no se aplica una corriente al solenoide, sino que solamente se mueve la película ferromagnética bajo la cabeza. Cuando una región magnetizada pasa bajo la cabeza, se genera un campo magnético en la cabeza. Cuando una región no magnetizada pasa bajo la cabeza, no se genera ningún campo. Por tanto, conforme la película se mueve bajo la cabeza, ésta cambia su característica magnética, lo que genera un voltaje en las terminales del solenoide. De este modo, la información almacenada en el medio ferromagnético se lee como un cambio de voltaje en las terminales del solenoide.

4.6.2. Interfaces con dispositivos de almacenamiento

Los dispositivos de almacenamiento necesitan transferir rápidamente grandes cantidades de datos hacia y desde la computadora. Por ello, usan comúnmente interfaces paralelas y bloques DMA para transferir datos. Existen actualmente dos interfaces estándar en uso: la interfaz integrada de dispositivos electrónicos (*Integrated Device Electronics* o IDE), también conocida como ATA, y la interfaz de sistemas de cómputo pequeños (*Small Computer Systems Interface* o SCSI), que permite el uso de otros tipos de dispositivos, y tiende a usarse en grandes computadoras y servidores.

Los dispositivos de almacenamiento utilizan dos tipos de tecnologías para su implementación: magnética u óptica, y ocasionalmente, una combinación de ambas.

El **grabado óptico** utiliza rayos láser para escribir series de puntos que codifican los datos a ser almacenados, o pequeños huecos en el medio de registro. Los puntos pueden leerse utilizando un láser para seguir la pista de los puntos. Si se utiliza una fotocelda para registrar la densidad de la luz reflejada por el medio en forma de corriente, ésta varía dependiendo si hay un punto o no, generando una señal que puede ser decodificada para restaurar los datos originales.

Los puntos o huecos pueden generarse de diferentes maneras. Pueden imprimirse en el medio mediante técnica de estampado (obviamente, esto solo puede hacerse en una fábrica), o el medio puede contener un decolorante que cambia de color con el calor de un láser, o contener una capa delgada de una sustancia que puede quemarse por el láser. En este último caso, una segunda aplicación del láser puede usarse para recuperar el medio, permitiendo que sea reutilizado. La grabación óptica puede lograr grandes densidades de datos, y es robusta, pero tiende a ser lenta.

El **grabado magnético** se basa en un medio cubierto con material ferromagnético, susceptible a magnetizarse. Utilizando un pequeño electromagneto, conocido como cabeza, el material puede magnetizarse, y alternando polaridades, es posible codificar patrones de datos. Es inherentemente reúsable.

Cuando el patrón de datos se pasa bajo una cabeza lectora similar, le induce una corriente eléctrica que se utiliza para reconstruir los datos. Otra técnica utiliza grabado optomagnético. Aquí, el material magnético se activa solo si se expone a la luz. Se usa un rayo láser para activar una región más pequeña que la propia cabeza, lo que permite registrar una cantidad mayor de datos. Los componentes optomagnéticos (sustancias cuya reflectividad depende de campos magnéticos) permiten la lectura de datos a altas densidades.

La grabación magnética puede ser muy rápida, pero no es muy robusta. La grabación de altas densidades depende de una escrupulosa exclusión de polvo y particular extrañas. La grabación optomagnética puede ser más robusta. Los discos y las cintas son los dispositivos de memoria auxiliar más comunes que se usan en una computadora digital.

4.6.3. Discos

En los dispositivos de almacenamiento auxiliar basado en disco, la película ferromagnética se coloca sobre un disco. La información se almacena en pistas sobre la superficie del disco. Nótese que las pistas ahora son círculos concéntricos, y no en forma de espiral como los discos fonográficos. Se hace girar el disco sobre su centro a una alta velocidad. Para leer o escribir del disco, la cabeza se coloca

sobre la pista deseada. Por tanto, para leer diferente información, la cabeza debe moverse sobre el radio del disco.

El almacenamiento en discos es mucho más rápido que en cintas, ya que se requiere pasar secuencialmente por las pistas en lugar de pasar por todos los datos almacenados. Así, en promedio, un disco requiere hacer media revolución antes de que la información deseada se encuentre bajo la cabeza, haciendo el tiempo de acceso mucho menor. El tiempo de acceso a disco se hace tan rápido como pueda la cabeza lectora moverse de una pista a otra, lo que representa un operación muy precisa y que toma algún tiempo. Para acelerar la operación, frecuentemente se colocan varias cabezas sobre un mismo disco. De este modo, la distancia en promedio que la cabeza debe moverse es menor, por lo que se disminuye el tiempo de acceso. El tiempo de acceso de un disco típico se encuentra en el orden de los milisegundos. Esto, por supuesto, es un promedio, ya que el tiempo de acceso depende del movimiento de las cabezas de una pista a otra. Este tiempo es mucho más veloz que una cinta, pero también resulta muy lento al compararlo con el tiempo de acceso de una RAM. El almacenamiento en disco no es secuencial, ya que no se lleva una secuencia de los datos del disco como se lleva en una cinta. Es por ello que a los discos se les conoce como almacenamiento de acceso directo.

Los discos magnéticos proveen de alta capacidad de almacenamiento a velocidades de operación moderadas. Almacenan información binaria en uno o más superficies circulares cubiertas de material magnético. Los discos se encuentran continuamente girando (a velocidades entre 5400 y 10,000 rpm), y se apilan con un espacio entre ellos. Se dividen en pistas, regiones circulares concéntricas al disco. Las pistas se dividen a su vez en sectores. Comúnmente, hay una cabeza lectora/escritora por cada superficie de disco, y en conjunto, las cabezas se montan en un brazo común. Esto para que la cabeza pueda colocarse con precisión y velocidad sobre la pista seleccionada.

Un sistema de disco se direcciona mediante bits de dirección que especifican (a) el número del disco, (b) la superficie del disco, (c) el número de sector, y (d) el número de la pista dentro del sector. Después de que las cabezas lectoras/escritoras se posicionan sobre la pista especificada, el sistema tiene que esperar hasta que el disco giratorio alcanza el sector específico bajo la cabeza. Una vez que se alcanza el principio de un sector, la información se transfiere velozmente.

En general, el dispositivo físico común para discos duros es conocido como el disco Winchester. El medio se construye como parte integral del disco y no es removible, permitiendo que el disco se encuentre sellado por fabricación, en una

atmósfera extremadamente limpia y controlada. Esto significa que se pueden utilizar cabezas muy pequeñas, y por tanto, la densidad de datos almacenados puede ser muy alta. El diámetro de los discos estándar son 3.5 pulgadas para computadoras de escritorio, y 2.5 pulgadas para computadoras portátiles. Su capacidad va desde unos cuantos Gigabytes a cientos de Gigabytes.

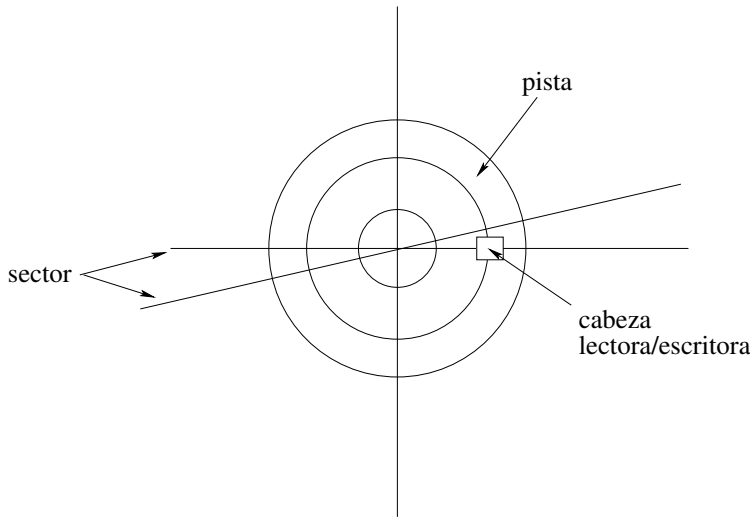


Figura 4.9: Un disco magnético.

Actualmente, el disco para computadoras digitales se conocen como disco duro (*hard disk*), y representa la forma original de los discos para almacenamiento masivo, y comúnmente se encapsulan en contenedores metálicos cerrados. Tienen capacidad de varios millones de bits, que año con año va en aumento.

Los dispositivos ópticos de almacenamiento se usan para grabar y recuperar video, audio y datos de imágenes, así como texto y datos binarios. Tienen una capacidad de grabado de extrema alta densidad, permitiéndoles almacenar varios Gigabytes de datos sobre un disco. Los dispositivos ópticos de almacenamiento tiene un tiempo de vida mucho más largo (decenas de años) comparados con el tiempo de vida de los dispositivos magnéticos (alrededor de 5 años).

Hay varios tipos de discos ópticos, incluyendo los siguientes:

- disco compacto (*Compact Disk* o CD): es un disco de 120 mm de diámetro

que almacena entre 650 y 700 Mbytes. Se utiliza más eficientemente para almacenamiento óptico preregistrado, para guardar archivos, y puede obtenerse como de sólo lectura (CD-ROM), de una sola escritura (CD-R), o reescribible (CD-RW). Los datos se organizan en sectores que pueden leerse independientemente, como un disco duro;

- disco digital versátil (*Digital Versatile Disk* o DVD): una versión más moderna que el CD, con la diferencia de que las pistas en el DVD son más angostas. Los DVD se leen con rayos láser con menor longitud de onda, lo que permite incrementar la capacidad de almacenamiento. Los datos se escriben en dos capas: una capa externa dorada semitransparente, que permite la lectura sobre una capa plateada inferior. Como los CD, hay varios tipos de DVD, incluyendo los DVD-ROM (de sólo lectura), los DVD-R (de una sola escritura, almacenando hasta 3.95 Gbytes por lado) y DVD-RAM (reescribible).

4.6.4. Cintas

Una forma de almacenamiento magnético involucra que la película ferromagnética se coloque sobre una cinta de plástico delgado. Esta cinta se enrolla sobre rieles, lo que permite que pueda manipularse y almacenarse fácilmente.

En una cinta magnética, los bits se graban como puntos magnéticos en pistas a lo largo (7 o 9). La información binaria se agrupa en bloques, llamados registros. Hay zonas de cinta no grabada entre los registros, lo que permite encontrar una información binaria particular mediante contar las zonas no grabadas y moverse hacia adelante, hacia atrás, y detenerse. Cada registro tiene un patrón de bits que lo identifica al principio de la cinta, de modo que los controles de la cinta puedan encontrar el registro solicitado, y el final de la cinta. Una cinta se direcciona mediante especificar el número de registro y el número de caracteres en el registro. Con cabezas lectoras/escriptoras montadas por cada pista, los datos binarios pueden leerse o escribirse en la forma de registros completos. Los registros pueden ser de longitud fija o variable. La lectura es completamente serial.

Frecuentemente, se utilizan varias cabezas en paralelo y perpendicularmente al movimiento de la cinta, lo que la divide en pistas o *tracks* a lo ancho de la cinta. Cada pista se utiliza para almacenar diferente información. Algunas pistas proveen a la computadora sobre información acerca de la localización o posición de datos en la cinta. Frecuentemente, las cintas tienen información grabada en su inicio que

se conoce como directorio (*directory*). Este sirve como un índice, informando a la computadora dónde en la cinta se almacenan ciertos datos. En general, las cintas almacenan diferentes conjuntos de datos llamados archivos (*files*). Por ejemplo, un archivo puede contener los registros de las cuentas de los clientes de alguna tienda; otro archivo puede contener información sobre inventarios, etc. Si se desea leer un archivo en particular, es necesario saber su posición en la cinta. Supóngase, por ejemplo, que una pista puede almacenar hasta 100,000 bits. Entonces, un archivo puede comenzar en una posición particular, en el bit 17,325. La pista que se utiliza para determinar posiciones puede contener un valor de 1 por cada bit. Así, mediante contar el número de valores 1 a partir del inicio de la cinta, la computadora puede determinar la posición de un punto en la cinta que contiene los datos requeridos, y colocarlo bajo la cabeza lectora.

Las cintas se escriben y leen mediante una unidad de cinta (*tape drive*). Esta contiene componentes para la grabación y elementos mecánicos para el movimiento de la cinta. Las cintas puede removerse fácilmente o insertarse en la unidad de cinta, de modo que una cinta diferente puede usarse cada vez. Por tanto, la cantidad de información que puede almacenarse en cintas es casi ilimitado.

Una sola cinta puede almacenar gran cantidad de información. Por ejemplo, una cinta con 6 cabezas para almacenamiento de datos puede almacenar hasta 100 millones de bits. Por otro lado, la cantidad de tiempo que se necesita para leer o escribir un dato en memoria se conoce como tiempo de acceso de esa memoria. La principal desventaja de las cintas como almacenamientos masivos es precisamente su gran tiempo de acceso. Una cinta debe ser reembobinada completamente antes de que se pueda al menos buscar un dato. Algunas unidades de cinta pueden mover las cintas a gran velocidad (algo así como 50 o 400 cm/s); sin embargo, el tiempo de acceso sigue siendo muy grande comparado con otros sistemas de almacenamiento. De hecho, las cintas se conocen como sistemas de acceso secuencial, ya que los datos se almacenan en una secuencia sobre la cinta, y es necesario mover la cinta secuencialmente por toda su longitud entre la posición actual y la posición deseada para obtener cualquier dato particular.

Las unidades de cinta utilizan la cinta contenida en una caja para facilitar su carga en la unidad. Las cintas de video, usualmente de 8 mm, o cintas de audio digital también pueden utilizarse. Su capacidad de almacenamiento es del orden de varios cientos de Gigabytes.

4.7. Códigos

La información almacenada en cualquier tipo de memoria digital es un número binario, una secuencia de unos y ceros. De hecho, todas las operaciones básicas de la computadora están diseñadas en términos binarios. Frecuentemente, es común que se desee almacenar información o trabajar con datos que contienen letras, números y otros caracteres útiles. Por ejemplo, una lista de nombres puede almacenarse para posteriormente ordenarse alfabéticamente. Este tipo de información se conoce con el nombre de datos alfanuméricos. Para almacenar datos alfanuméricos como secuencias de unos y ceros, se utilizan códigos. Un código es una cierta secuencia de números binarios que representan letras y símbolos. La computadora debe ser programada para procesar estos datos apropiadamente. Por ejemplo, un monitor funciona de modo que cuando en el teclado se escriben ciertas letras o números, estos aparezcan en la pantalla del monitor. Cuando el programa requiere que ciertas palabras se escriban, la computadora debe enviar las secuencias correctas de unos y ceros al monitor.

Los códigos de mayor uso en computadoras digitales son el código ASCII (*American Standard Code for Information Interchange*) y el código EBCDIC (*Extended Binary Coded Decimal Interchange Code*). La siguiente tabla muestra los valores binarios o códigos de algunos de los datos alfanuméricos más comúnmente utilizados para representar letras, números y el espacio en blanco.

Nótese que en ambos casos, cada código provee de un valor binario único para cada letra y cada número. Tales valores son de gran utilidad al comunicar a la computadora con el ambiente exterior. Cuando se escribe en un teclado de la computadora, por ejemplo, éste se encarga de convertirlos en su representación binaria. En general, es conveniente que todos los símbolos se representen mediante secuencias de ceros y unos con una misma longitud. Así, lo que se transmite son los valores binarios de los códigos de las letras, los números y otros símbolos (ortográficos, de puntuación, etc.).

Nótese que los códigos de la tabla anterior se ordenan numéricamente: el valor binario del código representante de la A es menor que el valor binario de la B. Esto resulta útil en programas de computadora que permiten ordenar listas alfabéticamente. En la tabla solo se muestran las letras mayúsculas; sin embargo, también hay códigos para las letras minúsculas. También es interesante que los tres bits más significativos del código ASCII para las letras mayúsculas son 100 para las letras de la A a la O, y 101 de la P a la Z. Si se reemplaza el 100 por

110 y el 101 por 111, se tienen los códigos ASCII para las letras minúsculas. Tal cambio es similar en el caso del código EBCDIC. En tal caso, los cuatro bits más significativos de los códigos cambian a minúsculas, en particular, si el 1100 se reemplaza por 1000, el 1101 se reemplaza por 1001, y el 1110 se reemplaza por 1010.

	ASCII	EBCDIC		ASCII	EBCDIC
A	100 0001	1100 0001	0	011 0000	1111 0000
B	100 0010	1100 0010	1	011 0001	1111 0001
C	100 0011	1100 0011	2	011 0010	1111 0010
D	100 0100	1100 0100	3	011 0011	1111 0011
E	100 0101	1100 0101	4	011 0100	1111 0100
F	100 0110	1100 0110	5	011 0101	1111 0101
G	100 0111	1100 0111	6	011 0110	1111 0110
H	100 1000	1100 1000	7	011 0111	1111 0111
I	100 1001	1100 1001	8	011 1000	1111 1000
J	100 1010	1101 0001	9	011 1001	1111 1001
K	100 1011	1101 0010	blanco	000 0000	0100 0000
L	100 1100	1101 0011			
M	100 1101	1101 0100			
N	100 1110	1101 0101			
O	100 1111	1101 0110			
P	101 0000	1101 0111			
Q	101 0001	1101 1000			
R	101 0010	1101 1001			
S	101 0011	1110 0010			
T	101 0100	1110 0011			
U	101 0101	1110 0100			
V	101 0110	1110 0101			
W	101 0111	1110 0110			
X	101 1000	1110 0111			
Y	101 1001	1110 1000			
Z	101 1010	1110 1001			

4.7.1. Códigos de detección de errores

Normalmente, las comunicaciones hacia una computadora y de la computadora hacia el exterior se hace mediante cables, alambres, o algún otro medio de transmisión de datos. La mayoría de estos medios funcionan en ambientes donde existe ruido, que se refiere a señales espurias que pueden llegar a afectar la transmisión. De hecho, por efecto de ruido, dentro de una comunicación digital un 0 puede aparecer después de la transmisión como un 1, y viceversa.

Para evitar este tipo de problemas, se han desarrollado códigos que permiten detectar si algún bit de una transmisión digital ha cambiado durante la misma. Uno de tales códigos se conoce como código de bit de paridad. En este código, se añade un bit extra al mensaje binario, que toma el valor de 0 o 1 dependiendo de si el número de unos en la representación binaria de una letra tiene paridad par o impar, respectivamente. De tal modo, al recibir una secuencia que representa un símbolo, la computadora puede verificar la paridad del número total de unos en la representación. Si el número total de unos no corresponde con la paridad expresada por el bit de paridad, entonces el símbolo se rechaza y se puede enviar un mensaje de error solicitando la retransmisión del símbolo; sin embargo, es notorio que si el medio de transmisión está sujeto a ruido, éste puede afectar a cualquier bit en el símbolo, pudiendo cambiar dos bits, o hasta al propio bit de paridad. En ambos casos, el código de bit de paridad resulta inútil para detectar errores, por lo que se han desarrollado otros tipos de códigos de detección y corrección de errores más sofisticados que permiten una transmisión más segura de los datos binarios.

Fuentes útiles de información

SCSI trade association

Technical Information on Storage Devices.

<http://www.scsita.org>

Evaluación

Preguntas de opción múltiple

1. ¿Cuál de los siguientes enunciados es verdadero?

- a) La memoria central es el almacenamiento de respaldo de la computadora.
 - b) La memoria auxiliar es la memoria de trabajo de una computadora.
 - c) La memoria auxiliar es veloz y cara.
 - d) La memoria central es la memoria de trabajo de la computadora.
 - e) La memoria central es grande y lenta.
2. Comenzando con los componentes más veloces y de pequeña capacidad, la jerarquía de la memoria es como sigue:
- a) Memoria caché, memoria virtual, registros, memoria principal, memoria auxiliar.
 - b) Registros, memoria caché, memoria principal, memoria virtual, memoria auxiliar.
 - c) Memoria auxiliar, memoria virtual, registros, memoria caché, memoria principal.
 - d) Memoria virtual, registros, memoria caché, memoria auxiliar, memoria principal.
 - e) Registros, memoria principal, memoria caché, memoria virtual, memoria auxiliar.
3. ¿Cuál de los siguientes enunciados sobre la memoria caché es verdadero?
- a) Contiene el código y datos más recientemente accedidos.
 - b) Se dedica exclusivamente de la transferencia de información entre la memoria auxiliar y el procesador.
 - c) Se dedica a la transferencia de información entre Entrada/Salida y el procesador.
 - d) Sirve como interface para la memoria auxiliar.
 - e) Se usa para dar una apariencia de gran cantidad de memoria.
4. ¿Cuál de los siguientes enunciados sobre la memoria auxiliar es verdadero?
- a) Contiene los datos y programas actualmente utilizados por el procesador.

- b) El procesador puede acceder directamente a la memoria auxiliar.
 - c) La memoria auxiliar es muy veloz.
 - d) La memoria auxiliar es volátil.
 - e) Contiene los datos y programas no actualmente utilizados por el procesador.
5. ¿Cuál es la principal ventaja de la SRAM sobre la DRAM?
- a) La SRAM tiene un consumo de energía menor y una capacidad mayor de almacenamiento.
 - b) La SRAM es volátil y cara.
 - c) La SRAM es más fácil de usar, tiene ciclos de Lectura/Escritura más cortos y no requiere *hardware* adicional.
 - d) La SRAM es más fácil de usar, tiene ciclos de Lectura/Escritura más largos y no necesita *hardware* adicional.
 - e) La SRAM requiere refrescarse frecuentemente.
6. Las memorias ROM:
- a) Almacenan los resultados de las operaciones de la ALU.
 - b) No mantienen su contenido cuando se les elimina la alimentación eléctrica.
 - c) Mantienen su contenido cuando se les elimina la alimentación eléctrica.
 - d) Son del tipo Lectura/Escritura aleatoria.
 - e) Son internamente dispositivos lógicos secuenciales.
7. La memoria de discos magnéticos provee:
- a) Rápida velocidad de operación.
 - b) Moderada velocidad de operación.
 - c) Lenta velocidad de operación.
 - d) Velocidad de operación tan lenta como los registros.
 - e) Velocidad de operación tan rápida como la memoria caché.

8. IDE es acrónimo de:
- a) *Interface for disk equipment.*
 - b) *Internal disk equipment.*
 - c) *Integrated disk equipment.*
 - d) *Internal delivery element.*
 - e) *Integrated device electronics.*
9. Una computadora utiliza circuitos RAM de 512×1 de capacidad. ¿cuántos circuitos se necesitan para proveer una capacidad de memoria de 1024 bytes?
- a) 8.
 - b) 16.
 - c) 2.
 - d) 24.
 - e) 32.
10. Una computadora emplea circuitos de RAM de 256×8 y circuitos de ROM de 1024×8 . La computadora requiere 2 kbytes de RAM y 4 kbytes de ROM. ¿Cuál es el número total de circuitos necesarios?
- a) 12.
 - b) 8.
 - c) 24.
 - d) 16.
 - e) 6.
11. ¿Cuál de las siguientes es una tecnología disponible de almacenamiento masivo?
- a) Magnético.
 - b) Holográfico.
 - c) Molecular.
 - d) Biónico.

- e) Cuántico.
12. ¿Qué tecnología de grabado utiliza el DVD?
- a) Cuántico.
 - b) Holográfico.
 - c) Magnético.
 - d) Optomagnético.
 - e) Óptico.
13. ¿Cómo se llama a los dispositivos de grabado y reproducción de un disco duro?
- a) Sensores.
 - b) Brazos.
 - c) Solenoides.
 - d) Cabezas.
 - e) Láasers.

Respuestas a las preguntas de opción múltiple

- 1. d
- 2. b
- 3. a
- 4. e
- 5. c
- 6. c
- 7. b
- 8. e
- 9. b

10. a

11. a

12. e

13. d

Capítulo 5

Desempeño

Resumen

Este capítulo examina cómo el diseño de un procesador y el sistema circundante afecta el desempeño, introduce algunas medidas básicas de desempeño, y discute su significado y limitación.

Objetivos del capítulo

Al final del capítulo, el estudiante debe ser capaz de:

1. Describir los parámetros que afectan el desempeño de una computadora y explicar las limitaciones en la mejora de la velocidad de procesamiento asociadas con cada factor.
2. Describir las restricciones para que los procesadores logren su velocidad potencial, y las medidas que deben tomarse para contrarrestar estas limitaciones.
3. Describir varias diferentes medidas de velocidad del procesador, y seleccionar cuál medida es aplicable a diferentes casos.
4. Entender la naturaleza del *benchmarking* y seleccionar el *benchmark* adecuado para describir el desempeño para diferentes casos.

5.1. Factores que afectan el desempeño

Hay varios elementos que juntos contribuyen a la velocidad de una computadora. Esta sección presenta cuáles son, y cómo se interrelacionan.

5.1.1. ¿Qué es la velocidad de una computadora?

Las computadoras se utilizan para una gran variedad de propósitos, y algunas medidas de velocidad se ajustan a algunas aplicaciones más que a otras.

Considerando a la computadora como un todo, se puede juzgar su desempeño en cuatro aspectos diferentes:

- desempeño del procesador: ¿qué tan rápido la computadora ejecuta el código en los programas?;
- desempeño en datos: ¿qué tan rápido puede la computadora procesar grandes conjuntos de datos contenidos en la memoria principal?;
- desempeño en entrada/salida: ¿qué tan rápido puede la computadora mover datos de la memoria principal a los dispositivos de entrada/salida?;
- desempeño de dispositivo: ¿qué tan rápido pueden los dispositivos periféricos, tales como discos, proveer y consumir datos?

Los primeros dos desempeños anteriores tienen que ver con el diseño del procesador y su soporte inmediato: la interface de *bus*, la administración de la memoria, el propio sistema de memoria principal, incluyendo cachés.

El factor determinante en el desempeño de entrada/salida depende de los dispositivos y la arquitectura de entrada/salida de la computadora. En computadoras actuales, esto puede ser altamente complejo. Una computadora personal moderna tiene al menos cuatro *buses* de entrada/salida:

- la interface de conexión a periféricos (*Peripheral Connect Interface* o PCI), un *bus* que conforma las conexiones de entrada/salida estándar;
- el *bus* AT (nombrado así por la computadora personal original de IBM), que se mantiene para permitir el funcionamiento de los periféricos antiguos de la IBM PC;

- un puerto de gráficos avanzados (*Advanced Graphics Port* o AGP) que es un conector para el *hardware* de la pantalla;
- un *bus* serial universal (*Universal Serial Bus* o USB) para una conexión simple de varios dispositivos periféricos.

El correcto servicio y secuenciación de los diferentes dispositivos puede hacer una diferencia considerable en cuanto a la velocidad con que los datos pueden alimentarse a los dispositivos periféricos, con un efecto grande particularmente en el desempeño de la computadora para la interacción con el usuario. Frecuentemente, tal desempeño se determina por la forma en que el sistema operativo actúa, o cómo es configurado. Los componentes de computadoras provenientes de diferentes proveedores, que se construyen con *hardware* idéntico, pueden tener desempeños muy diferentes.

5.2. Velocidad de reloj y ciclos de instrucción

Para una computadora simple, la velocidad de su reloj determina la velocidad de las operaciones o la velocidad de la ejecución de instrucciones, y por tanto, el desempeño de la computadora.

La velocidad del reloj de entrada a la unidad de control del procesador se conoce como **velocidad de reloj del procesador**.

Desafortunadamente, esto no dice nada respecto a la velocidad real de ejecución de programas por el procesador a menos que se sepa qué sucede en cada *tick* de reloj, cuando el secuenciador ejecuta otra operación en el procesador. El secuenciador genera las señales digitales que cargan valores en los registros internos del procesador, produciendo lo que se conoce como el paso de una instrucción.

Diferentes arquitecturas de procesador requieren diferente número de pasos para realizar una instrucción completa, y algunos requieren un número diferente de pasos para llevar a cabo diferentes tipos de instrucción. Muchas computadoras inicialmente usaban cuatro pasos para una instrucción:

- paso 1: buscar la instrucción;

- paso 2: buscar el operando;
- paso 3: realizar la operación;
- paso 4: escribir el resultado.

Tales computadoras requieren cuatro *ticks* de reloj para ejecutar cada instrucción, de modo que la velocidad de ejecución de instrucciones es un cuarto de la velocidad de reloj.

El número de pasos que se toman para realizar una instrucción individual provee el número de ciclos de reloj por instrucción (*Clocks per Instruction* o **CPI**) del procesador. La velocidad de ejecución de una instrucción indica qué tan rápido un procesador ejecuta sus instrucciones. Se calcula mediante dividir la velocidad del reloj (en Hertz, o Hz) entre el CPI del procesador para obtener el número de instrucciones por segundo (*Instructions per Second* o IPS). La medida normal son los millones de instrucciones por segundo (*Million Instructions per Second* o **MIPS**).

Un procesador con instrucciones muy simples normalmente requiere muchas de ellas para implementar un programa dado, que un procesador con instrucciones más complejas.

5.3. Aumentando el desempeño del procesador

Hay dos formas obvias de aumentar la velocidad de un procesador: (a) incrementar la velocidad de su reloj y (b) reducir el CPI.

5.3.1. Aumentando la velocidad de reloj

La velocidad máxima del reloj depende del tiempo que los circuitos lógicos en el procesador les toma para realizar un solo paso de una instrucción. Si le toma $1\mu s$ para propagar el acarreo de un bit menos significativo al bit más significativo de la ALU, una velocidad de reloj mayor a 1 MHz corre el riesgo de arrojar resultados

aritméticos incorrectos. La velocidad máxima posible de reloj depende del paso más lento dentro de una instrucción. Puede ser posible incrementar la velocidad de reloj mediante dividir tal paso en dos.

Por ejemplo, el procesador simple de cuatro pasos descrito anteriormente puede tener los siguientes tiempos mínimos para cada paso:

- paso 1: buscar la instrucción: 100 ns;
- paso 2: buscar el operando: 180 ns;
- paso 3: realizar la operación: 80 ns;
- paso 4: escribir el resultado: 100 ns.

La velocidad de reloj más rápida posible para este procesador está dada por:

$$\frac{1}{180 \times 10^9} = 5.56MHz$$

De modo que puede ejecutar instrucciones a una velocidad de:

$$\frac{5.56}{4} = 1.39MIPS$$

Supóngase que se divide el paso 2 en dos partes:

- paso 1: buscar la instrucción: 100 ns;
- paso 2a: calcular la dirección del operando: 80 ns;
- paso 2b: buscar el operando: 100 ns;
- paso 3: realizar la operación: 80 ns;
- paso 4: escribir el resultado: 100 ns.

La velocidad más rápida de reloj ahora es:

$$\frac{1}{100 \times 10^9} = 10MHz$$

Lo que le permite ejecutar instrucciones a una velocidad de:

$$\frac{10}{5} = 2MIPS$$

Lo que representa un incremento de cerca del 50%.

5.3.2. Reduciendo los ciclos por instrucción (CPI)

Si se contara con un puerto de memoria por separado para buscar una instrucción, se podría buscar la siguiente instrucción mientras se escribe el resultado previo. Por ejemplo:

- paso 1: buscar el operando: 180 ns;
- paso 2: realizar la operación: 80 ns;
- paso 3: escribir el resultado/buscar la siguiente instrucción: 100ns.

La velocidad de reloj posible en este ejemplo sigue siendo de $5.56MHz$, pero la velocidad de ejecución es ahora:

$$\frac{5.56}{3} = 1.85MIPS$$

Lo que refleja un aumento del 25 %.

Decrementar el número de pasos de ejecución disminuye el CPI, mientras que incrementarlos permite una mayor velocidad de reloj, a costa de un mayor CPI. Es labor de los diseñadores de procesadores encontrar la relación óptima para una arquitectura particular y una tecnología de fabricación.

5.3.3. Pipelines

Hay una forma de resolver el compromiso de la sección anterior entre CPI y velocidad de reloj, que implica diseñar el procesador mediante la técnica de *pipeline*. Cada paso de ejecución de una instrucción cuenta con su propia sección del procesador que la realiza, y alimenta sus resultados a la siguiente sección del siguiente paso. Un ejemplo de *pipeline* para un procesador de cinco pasos se muestra en la Figura 5.1.

A cada ciclo de reloj, se busca una nueva instrucción, se calcula la dirección de un operando, se busca un operando, se realiza una operación, y se escribe un resultado, de modo que 5 instrucciones se encuentran en progreso, una instrucción por cada etapa del *pipeline*. La Figura 5.2 ilustra esto.

Una nueva instrucción se completa por cada ciclo de reloj, de modo que el CPI es 1, y por lo tanto, la velocidad de ejecución de instrucciones es 10 MIPS, que representa un incremento de siete veces. El CPI permanece en 1 sin importar el

número de pasos en el *pipeline*, por lo que se puede hacer la división a pasos muy pequeños si se desea, utilizando una velocidad de reloj muy alta, y en consecuencia, un alto desempeño; sin embargo, hay otros factores que no permiten incrementar este desempeño desmedidamente. Por ejemplo, en cada ciclo de reloj, el *pipeline* realiza tres operaciones sobre la memoria, lo que requiere un sistema de memoria tres veces más rápido, o con tres puertos por separado.

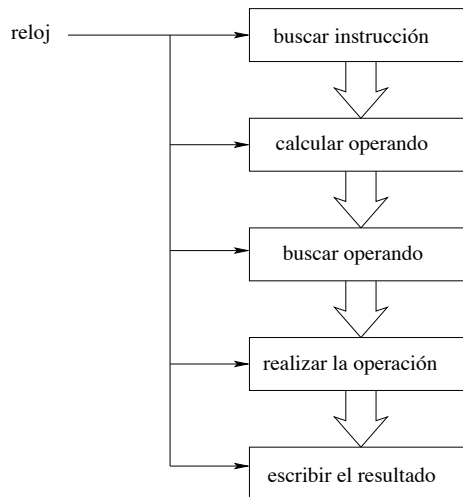


Figura 5.1: Un ejemplo de *pipeline* de cinco etapas.

Cuando una instrucción causa un salto, se modifica la dirección de la siguiente instrucción a ejecutar, que se tiene disponible en la etapa final del *pipeline*, para cuando el propio *pipeline* ya ha buscado y comenzado con la ejecución de otras 4 instrucciones. Estas ejecuciones deben detenerse, y todos los registros del *pipeline* deben reestablecerse antes de que se busque la siguiente instrucción. A esta acción se le denomina descargar (*flushing*) el *pipeline*. Además de la pérdida de ciclos de reloj, el *hardware* que se requiere para esto es complejo, lo que tiende a hacer lenta la operación del *pipeline*.

Ya que cada operación del *pipeline* se realiza cada ciclo de reloj, esto requiere de un *hardware* dedicado, lo que incluye una ALU por separado por cada etapa del *pipeline* que realice operaciones lógico-aritméticas.

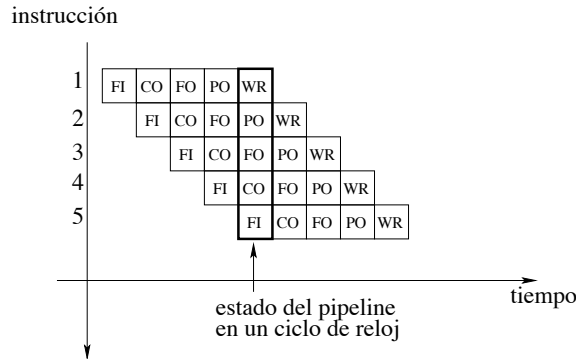


Figura 5.2: 5 instrucciones en un *pipeline*.

Finalmente, la operación en *pipeline* supone que todas las operaciones tienen el mismo formato, número de pasos y tiempo en cada paso, lo que no siempre es el caso. Diseñar un procesador en *pipeline* con suficiente flexibilidad para conjuntos de instrucciones complejas es una tarea difícil.

5.3.4. Computadoras de conjunto reducido de instrucciones (RISC)

Los procesadores en *pipeline* pueden simplificarse si su conjunto de instrucciones se diseña adecuadamente, esto puede hacer al conjunto de instrucciones más difícil de programar, pero como los programas ejecutables generalmente se producen mediante compilación, la programación no resulta un problema. Esta filosofía se conoce por el nombre algo engañoso de computadora de conjunto de instrucciones reducido (*Reduced Instruction Set Computer* o RISC). El nombre resulta engañoso ya que la mayoría de los procesadores RISC tienen más instrucciones que su contraparte, la computadora de conjunto de instrucciones complejo (*Complex Instruction Set Computer* o CISC), ni el poder de una instrucción en RISC difiere significativamente de una instrucción en CISC. El diseño RISC elimina características del diseño CISC que dificultan la operación del *pipeline*:

- se evitan las instrucciones de longitud variable y los ciclos de instrucción. Mediante usar instrucciones de una sola longitud y ciclos de instrucción de doble longitud, se hace que el diseño del *pipeline* sea más simple, por las razones descritas anteriormente;

- la mayoría de las operaciones suceden entre registros del procesador, por lo que se provee de una gran cantidad de registros, esto elimina la necesidad de sincronizarse con sistemas de memoria externos y mucho del cálculo de direcciones;
- se proveen instrucciones *load* y *store* para transferir datos entre la memoria y los registros del procesador. Estas instrucciones son generalmente de doble longitud, para permitir tiempo para acceder a la memoria externa;
- el efecto de una instrucción de salto se retrasa hasta después que se ha ejecutado la etapa final del *pipeline*, de modo que el *pipeline* no requiere descargarse.

Cuando se introdujeron inicialmente, los procesadores RISC sobrepasaron el desempeño de los procesadores CISC debido a las ventajas de velocidad del *pipeline*. Ahora, es posible diseñar implementaciones *pipeline* para las máquinas CISC más complejas, y la ventaja de desempeño de RISC ha desaparecido. Su simplicidad las hace adecuadas cuando el costo y consumo de energía son importantes.

El CPI de un procesador RISC bien diseñado o un procesador CISC en *pipeline* es mayor a 1, debido a las instrucciones de doble longitud y a la pérdida de instrucciones después de un salto. Son comunes los valores en el rango de 1.25 y 1.5.

5.3.5. Procesadores súperescalares

Imagínese que se diseña un procesador con dos *pipelines* como se muestra en la Figura 5.3.

A este arreglo se le conoce como procesador súperescalar. Cada ciclo de reloj, este procesador puede procesar dos instrucciones, siempre y cuando la etapa de búsqueda de instrucción pueda buscar dos instrucciones conjuntamente, lo que reduce a la mitad al CPI. Es poco probable que tal desempeño se logre. Frecuentemente, una instrucción hace uso de un resultado obtenido por una instrucción anterior, por lo que ambas no pueden ejecutarse al mismo tiempo. Un procesador súperescalar tiene circuitería que permite la detección y retraso de las instrucciones. Muchas veces, uno de los *pipelines* no se utiliza, reduciendo su ventaja.

Los procesadores súperescalares pueden diseñarse con más de dos *pipelines*, y los *pipelines* individuales pueden especializarse para diferentes tipos de instrucciones; sin embargo, conforme el número de *pipelines* incrementa, la aceleración

(*speedup*) disminuye, debido a las dependencias. Circuitería especializada para la planificación de instrucciones puede reordenar las instrucciones, y los *pipelines* desocupados se usan para ejecutar especulativamente los códigos que siguen ambas rutas a partir de un salto condicional, lo que permite un procesamiento ininterrumpido cualquiera que sea el resultado de la condición.

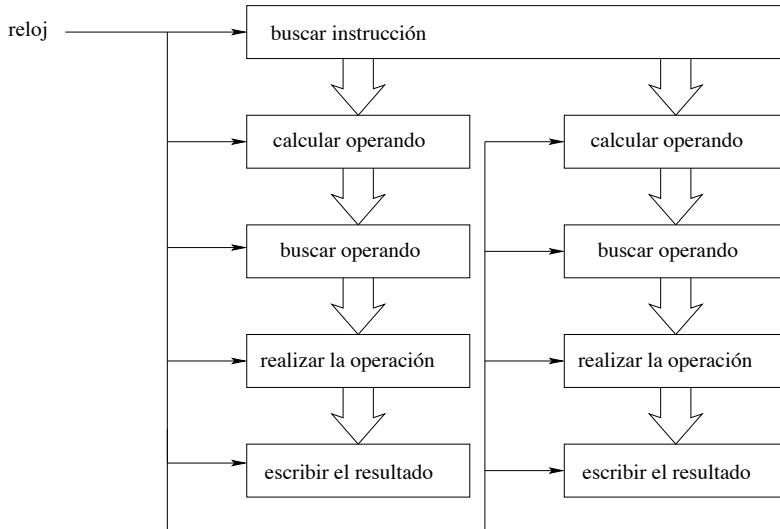


Figura 5.3: Procesador con dos *pipelines*.

5.3.6. El papel del compilador

La mayoría de los programas de computadora se escriben en lenguajes de alto nivel, como C o Java, que se traducen a instrucciones de máquina por un programa conocido como **compilador**.

Cuando se mide el desempeño de un programa, el compilador utilizado se considera como un elemento importante. Más aún, los compiladores se utilizan como complemento del diseño arquitectónico de una computadora. Los compiladores tradicionales se diseñaron alrededor de una pila de ejecución (*execution stack*),

un área de la memoria que contiene, entre otras cosas, las variables locales de un procedimiento. Los procesadores RISC requerirían tres instrucciones para cada acceso a la pila (un *load*, un *operate* y un *store*). Parte de la arquitectura RISC se basa en un compilador que utiliza los registros del procesador para las variables locales, en lugar de acceder a la pila, evitando el problema de acceso a la memoria. En este sentido, se considera al compilador como parte de la arquitectura del procesador.

Si el compilador genera instrucciones en pequeños grupos que no son interdependientes, entonces un procesador súperescalar puede mantener llenos sus *pipelines*. Esto tiene un gran efecto si el compilador cuenta con información del procesador.

Tomando esta idea al extremo, el compilador puede tomar el papel de asegurar que no hay interdependencias entre instrucciones ejecutándose concurrentemente, enviando paquetes de instrucciones, uno por cada *pipeline* en el procesador. A esto se conoce como computadora con palabra muy larga de instrucción (*Very-Long Instruction Word* o VLIW). Esta computadora puede lograr un mayor grado de paralelismo, y por tanto, un CPI más bajo que un procesador súperescalar. La labor de un planificador complejo lo hace el compilador, lo que potencialmente aumenta las velocidades de reloj. Dos ejemplos de procesadores VLIW son el Intel IA64 y el Transmeta Crusoe PC.

5.4. MIPS y FLOPS

Los millones de instrucciones por segundo (*Million Instructions per Second* o MIPS) y las operaciones de punto flotante por segundo (*Floating Point Operations per Second* o FLOPS) son dos medidas ampliamente utilizadas del desempeño de un procesador. Esta sección discute lo que significan, y cómo interpretar los valores numéricos de MIPS y FLOPS.

5.4.1. La medida de desempeño MIPS

A pesar de las desventajas descritas anteriormente sobre los MIPS, esta medida se ha vuelto de uso común para el desempeño de computadoras. Depende de un desempeño comúnmente acordado que representa 1 MIPS, y provee de un *benchmark* contra el cual el desempeño de otras computadoras puede ser comparado. Una de tales comparaciones es la hecha con respecto a la velocidad de

la computadora VAX11/780 de la Digital Equipment Corporation (antes DEC, ahora Compaq Corporation), que ha sido una computadora de 1MIPS popular. Frecuentemente, los MIPS se llegaban a citas como VAX MIPS.

Para cómputo matemático, la medida clave es el desempeño cuando se opera utilizando la representación numérica de punto flotante. El desempeño de todas las operaciones entre cantidades en punto flotante involucra tanto la adición como la multiplicación, de modo que el desempeño puede ser simplemente juzgado en términos del número de operaciones de punto flotante realizadas.

Cuando se juzga un desempeño en FLOPS, es importante conocer la longitud de las variables de punto flotante que se utilizan, ya sea 32 bits (en precisión sencilla), 64 bits (en doble precisión), u 80 bits (en precisión extendida).

5.5. Ancho de banda y cuellos de botella

Para tener un buen desempeño, todas las partes de una computadora deben ser suficientemente rápidas para apoyarse entre sí. Esta sección discute dónde se encuentran los cuellos de botella en un sistema de cómputo, y cómo pueden evitarse.

El **ancho de banda**, como se considera en sistemas de cómputo, es la velocidad con la que tal subsistema requiere transferir datos si debe funcionar a toda su capacidad.

En seguida se presentan algunos cálculos de ancho de banda para los dos procesadores que se han mencionado anteriormente, haciendo las siguientes suposiciones:

- se utiliza una palabra de instrucción de 32 bits (4 bytes);
- el 20 % de las instrucciones ejecutan transferencias de datos desde y hacia la memoria.

5.5.1. El procesador de cuatro ciclos

Este procesador tiene una velocidad de ejecución de instrucciones (*instruction throughput*) de 1.39MIPS, de modo que el ancho de banda utilizado es:

$$\begin{aligned}
1.39 \times 4 \times 8 \text{ millones de bits por segundo por instrucciones} &= 44.5 \text{ millones de bits por segundo} \\
1.39 \times 4 \times 8/5 \text{ millones de bits por datos} &= 8.9 \text{ millones de bits por segundo} \\
\text{Ancho de banda total} &= 53.4 \text{ millones de bits por segundo}
\end{aligned}$$

Si se conecta este procesador con un sistema de memoria de 32 bits, el tiempo del ciclo de memoria debe ser:

$$32/53,400,000 \text{ segundos} = 600 \text{ nanosegundos}$$

5.5.2. El procesador súperescalar

Este procesador tiene una velocidad de ejecución de instrucciones de 16MIPS, de modo que el ancho de banda utilizado es:

$$\begin{aligned}
16 \times 4 \times 8 \text{ millones de bits por segundo por instrucciones} &= 512 \text{ millones de bits por segundo} \\
16 \times 4 \times 8/5 \text{ millones de bits por datos} &= 102 \text{ millones de bits por segundo} \\
\text{Ancho de banda total} &= 614 \text{ millones de bits por segundo}
\end{aligned}$$

Si se conecta este procesador con un sistema de memoria de 32 bits, el tiempo de ciclo de memoria debe ser:

$$32/614,000,000 \text{ segundos} = 52 \text{ nanosegundos}$$

Una manera de duplicar el ancho de banda disponible es incrementar el ancho de la memoria. En la práctica, se requiere normalmente una memoria más veloz que la que se indica por estos simples cálculos.

5.5.3. Cachés

Los subcircuitos dentro de un circuito integrado pueden accederse mucho más rápidamente que los circuitos fuera del propio circuito. Así, el procesador puede funcionar más rápidamente si cuenta con los datos de manera interna que si tiene que buscarlos en memoria externa. La solución es un memoria caché. Esta memoria está organizada como un arreglo de entradas o líneas. Cada entrada contiene no sólo los datos en memoria, sino también un registro de la dirección en memoria principal (y a veces, de la memoria virtual) de donde provienen los datos, en una sección de la entrada que se conoce como etiqueta (*tag*). Conforme se leen datos de la memoria por parte del procesador, el caché los almacena. Si el procesador hace una solicitud de datos en memoria, primero se busca en la memoria caché,

mediante comparar la dirección del dato solicitado con la etiqueta de sus entradas. Si se encuentra, se le llama acierto, y el dato se lee de la memoria caché en lugar de la memoria principal, y el procesamiento continúa; de lo contrario, se le conoce como fallo, y se debe acceder a la memoria principal en busca del dato; sin embargo, el fallo hace que las direcciones vecinas al dato en memoria principal ahora se copien a la memoria caché, de modo que en la siguiente lectura, se asegura que se trate de un acierto. Se puede afirmar que la memoria caché acelera los accesos a las direcciones que han sido recientemente usados.

El diseño de la memoria caché afecta la razón de aciertos (*hit ratio*) de la propia caché, que también afecta el ancho de banda requerido por la memoria. Considere el siguiente ejemplo.

Si el procesador súperescalar se conecta a la memoria principal mediante una memoria caché con una razón de acierto del 80 %, entonces una quinta parte de los accesos se hacen a la memoria principal, de modo que el ancho de banda requerido es ahora:

$$614,000,000 \times 1/5 = 52 \text{ millones de bits por segundo}$$

La caché reduce el ancho de banda de la memoria a una quinta parte. Algunas veces, la memoria caché puede ser descargada (*flushed*), generalmente por un programa que lee una estructura de datos muy grande de punta a punta. Si esto sucede, el procesador funciona a un quinto de su velocidad, limitado por el ancho de banda de la memoria. Prestar atención al diseño de los ciclos internos de los programas puede tener un gran efecto en el desempeño de los procesadores con memoria caché.

Fuentes útiles de información

Cragon, H.G.

Memory Systems and Pipelined Processors.

Jones and Bartlett, 1996.

Mahapatra, R.N., Venkatrao, B.

The Processor-Memory Bottleneck: Problems and Solutions.

<http://www.acm.org/crossroads/xrds5-3/pmgap.html>.

<http://developer.intel.com>

<http://www.amd.com/us-en/Processors/ProductInformation/>

<http://www.transmeta.com>.

Evaluación

Preguntas

1. Usando el procesador de cuatro pasos de la sección 5.3.1, ¿cuál sería el desempeño de un procesador que tanto divide el paso 2 como conjunta el paso de búsqueda de instrucción con el paso de escritura de resultado?
2. ¿Cuál sería el efecto en el desempeño del procesador de cuatro pasos original de dividir el paso 3, realizar la operación, en dos pasos, cada uno de 40 ns de duración?
3. ¿Cuál sería el desempeño en MIPS del procesador de cuatro pasos original si se rediseñara como *pipeline*?
4. ¿Cuál es el ancho de banda requerido para el procesador descrito en la sección 5.3.2, suponiendo una palabra de instrucción de 16 bits y una transferencia de datos de 32 bits cada 10 instrucciones?
5. Dado un sistema de memoria de 32 bits, ¿cuál es el tiempo de ciclo de memoria?
6. ¿Cuál sería si el sistema de memoria de 32 bits se conectara mediante una caché con 70 % de razón de aciertos?

Preguntas de opción múltiple

1. ¿Cuál de los siguientes conceptos no es un aspecto del desempeño de una computadora?
 - a) Dispositivo.
 - b) Procesador.
 - c) Marca.
 - d) Entrada/salida.
 - e) Data.

2. ¿Cuál de los siguientes no es un tipo de *bus* de entrada/salida?
- a) AT.
 - b) MP.
 - c) AGP.
 - d) USB.
 - e) PCI.
3. ¿Cuál de las siguientes palabras describe el hecho de que una computadora realiza operaciones durante una señal de reloj?
- a) Psicótico.
 - b) Sincopado.
 - c) Sinónimo.
 - d) Sincronizado.
 - e) Simplificado.
4. ¿Qué es un ciclo de instrucción?
- a) La secuencia de instrucciones en un ciclo de programa.
 - b) Reutilizar una instrucción en la memoria caché.
 - c) Una banda de transporte robot con dos ruedas.
 - d) La secuencia de pasos requerida para completar una instrucción.
 - e) Un conjunto de instrucciones que hacen casi lo mismo.
5. ¿Qué quiere decir la abreviación “MIPS”?
- a) Millones de instrucciones por segundo.
 - b) Millones de instrucciones de programa sistema.
 - c) Multi-instrucciones de programa serial.
 - d) Múltiple inversor proceso secuencial.
 - e) Múltiple impreso proceso salida.
6. Si un procesador tiene un ciclo de instrucción de seis pasos, siendo el paso más largo de $10ns$, ¿cuál es la velocidad de reloj máxima posible?

- a) 60 MHz.
 - b) 166 MHz.
 - c) 16.66 MHz.
 - d) 600 MHz.
 - e) 100 MHz.
7. ¿Cuál de los siguientes no incrementa el desempeño de un procesador?
- a) Aumentar la velocidad del reloj.
 - b) Reducir los ciclos por instrucción (CPI).
 - c) Usar un disipador mayor de calor.
 - d) Diseñar un procesador en *pipeline*.
 - e) Diseñar un procesador como súperescalar.
8. ¿Cuántas instrucciones por ciclo de reloj realiza un procesador con un solo *pipeline*?
- a) 1.
 - b) 1/2.
 - c) 2.
 - d) Tantos como pasos en el *pipeline*.
 - e) 10.
9. ¿Qué es un procesador súperescalar?
- a) Un procesador muy grande.
 - b) Un procesador muy pequeño.
 - c) Un procesador con múltiples *pipelines*.
 - d) Un coprocesador de propósito especial.
 - e) Un procesador de alto rango.
10. ¿Cuál es el efecto de la memoria caché?
- a) Duplica la velocidad del procesador.

- b) Reduce el costo del procesador.
- c) Incrementa la velocidad del reloj del procesador.
- d) Reduce el tiempo de ciclo de un procesador.
- e) Reduce el número de accesos a memoria principal por el procesador.

Respuestas a las preguntas

1. Los pasos son ahora:

- Paso 1. obtener la dirección del operando: 80 ns.
- Paso 2. buscar el operando: 100 ns.
- Paso 3. realizar la operación: 80 ns.
- Paso 4. escribir resultado/ buscar la siguiente instrucción: 100 ns.

La velocidad máxima de reloj es de 10 MHz, y la velocidad de ejecución de instrucciones es $10/4 = 2.5MIPS$, un incremento de casi el 80% sobre el valor original.

2. Los pasos son ahora:

- Paso 1. buscar la instrucción: 100 ns.
- Paso 2. buscar el operando: 180 ns.
- Paso 3a. realizar la operación 1: 40 ns.
- Paso 3b. realizar la operación 2: 40 ns.
- Paso 4. escribir el resultado: 100 ns.

La velocidad de reloj más rápida posible está dada por $1/(180 \times 10^{-9}) = 5.56MHz$. Se ejecuta a una velocidad de ejecución de instrucciones de $5.56/5 = 1.112MIPS$, un decremento de 20% respecto al original.

3. El *pipeline* es como se muestra en la Figura 5.4.

Paso más lento: 180 ns, velocidad de reloj: 5.56 MHz

Velocidad de ejecución de instrucciones: 5.56 MIPS

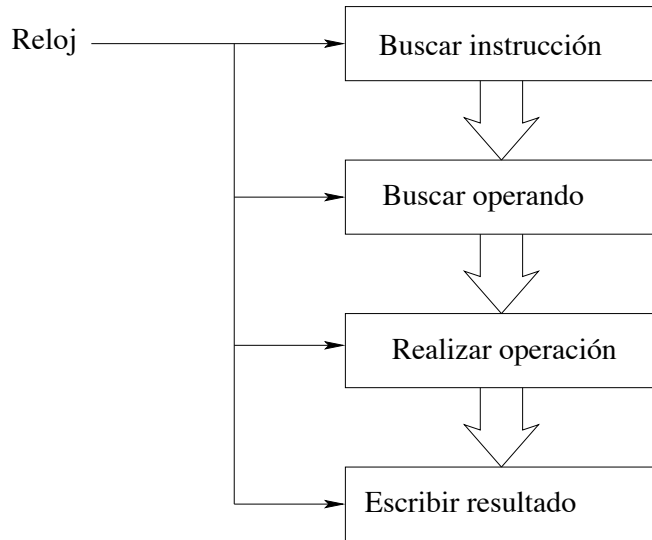


Figura 5.4: Procesador con un *pipeline* de cuatro pasos.

4.

$$\begin{aligned}
 \text{Instrucciones} &= 5560000 \times 16 = 88960000 \text{ bits por segundo} \\
 \text{Datos} &= 5560000 \times 32/10 = 17792000 \text{ bits por segundo} \\
 \text{Total} &= 106752000 \text{ bits por segundo}
 \end{aligned}$$

5. $32/106752000 = 299ns$

6. $\text{Acceso a memoria} = 106752000 \times 3/10 = 32025600 \text{ bits por segundo}$
 $\text{Tiempo de ciclo} = 32/32025600 = 1\mu\text{segundo}.$

Respuestas a las preguntas de opción múltiple

1. c

2. b

3. d

4. a

5. e

6. c

7. a

8. c

9. e

Capítulo 6

Periféricos

Resumen

Este capítulo introduce los diferentes tipos de periféricos, lo que hacen, y sus características.

Objetivos del capítulo

Al final del capítulo, el estudiante debe ser capaz de:

1. Describir la diferencia entre dispositivos carácter, bloque y arrastre, y clasificar diferentes dispositivos entre estas categorías.
2. Describir métodos paralelos y seriales para conectar un dispositivo al *bus* de la computadora, y discutir qué método es adecuado para que dispositivo particular.
3. Enumerar diferentes tipos de dispositivos periféricos, describir para que es cada uno de ellos, y contar con una visión de cómo se conectan a la computadora.

6.1. Tipos de periféricos

Se presenta una panorámica de los tipos de dispositivos periféricos de una computadora digital.

6.1.1. Clasificando periféricos

Una forma de clasificar los periféricos de una computadora es de acuerdo a lo que hacen. Otra forma es la forma como la computadora se comunica con ellos. En esta sección se clasifican los periféricos por la segunda forma, como dispositivos de carácter, bloque y escaneo, y si es un dispositivo de entrada, de salida, o de entrada/salida. Generalmente, la interface de *hardware* al dispositivo es bidireccional (entrada y salida), ya sea o no que el dispositivo se considere solo de entrada o de salida.

En un **dispositivo de carácter** los datos se transfieren en unidades pequeñas e individuales, usualmente de 8 bits, y frecuentemente conteniendo un código representando un carácter alfanumérico. El sistema de codificación más comúnmente utilizado es llamado ASCII, que se utiliza frecuentemente para almacenamiento interno de caracteres. Algunas veces, los datos que se transfieren se encuentran en algún otro código. Ya sea o no que la información se refiera a un solo carácter, se llama a cualquier dispositivo que transfiera datos en unidades pequeñas un dispositivo de carácter. Algunos periféricos usan o proveen de datos en unidades más grandes, de cientos o miles de bytes en un tiempo, generalmente para manejar datos a alta velocidad. Estos se conocen como **dispositivos de bloque**. Un **dispositivo de escaneo** permite producir o aceptar datos como un bloque bidimensional, generalmente utilizado para representar gráficos bidimensionales.

6.2. Conectando periféricos

Esta sección introduce las diferentes formas en que los dispositivos periféricos se conectan en la computadora.

6.2.1. Dispositivos paralelos

Desde el punto de vista de la computadora, los periféricos aparentan ser un registro o un conjunto de registros conectados al *bus*. En términos de *hardware*, esto simplemente significa que hay un registro conectado al *bus* y al decodificador de direcciones que temporiza la entrada y salida de datos al registro. Las entradas o salidas de este registro en el lado externo del *bus* proveen directamente un byte o más datos para el dispositivo conectado. Los dispositivos directamente conectados a un registro como este se conoce como dispositivos paralelos.

6.2.2. Dispositivos seriales

Un problema con las interfaces paralelas es que requiere varios cables para conectarse. Conforme la velocidad de transferencia de datos aumenta, el retardo entre las señales de cables individuales puede volverse significativo. Una forma de resolver esto es conectar el dispositivo periférico serialmente, utilizando un registro de corrimiento. Después de que cada byte de datos se carga, el registro de corrimiento envía, un paso de reloj a la vez, cada bit individual, lo que hace que parezca que los bits aparezcan secuencialmente a la salida, que se conecta mediante un solo cable al registro de corrimiento receptor. Tal registro funciona a la misma velocidad, causando que los bits de datos puedan reensamblarse en su forma paralela.

Una interface serial genérica se muestra en la Figura 6.1.

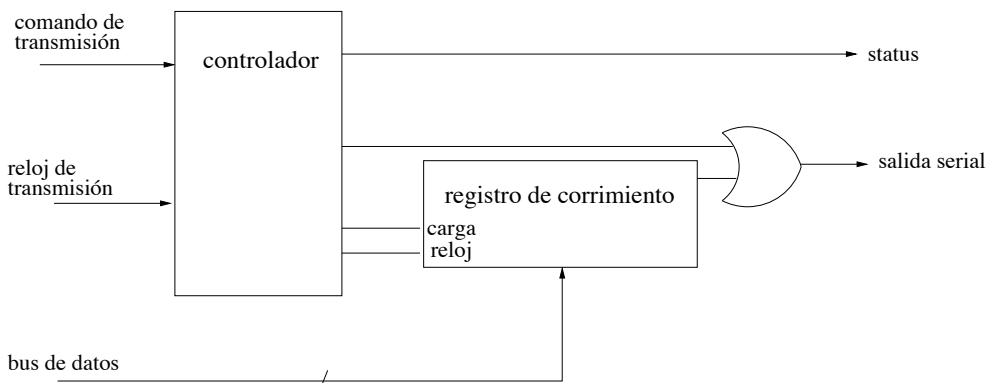


Figura 6.1: Una interface serial genérica.

La máquina de estados finitos dentro del controlador recibe un comando de transmisión y produce la siguiente secuencia:

1. Un carácter o byte se carga en el registro de corrimiento.
2. Suficientes señales de reloj al registro se registran como para correr el carácter por la salida serial.
3. La línea de status envía una señal que la interface está lista para transmitir otro carácter.

El receptor es similar, excepto que se utiliza un registro de corrimiento de serial a paralelo, y la secuencia de acciones se invierte. Tanto el transmisor como el receptor deben estar sincronizados, es decir, tener la misma frecuencia de reloj. Si tienen generadores de reloj independientes (lo que se conoce como una transmisión asíncrona) entonces solo pueden sincronizarse por periodos cortos de tiempo. Para permitir que el generador de reloj se inicie en el momento justo, el carácter a enviar se prefija con un bit de inicio, y termina con dos bits de detención (la compuerta OR de la salida serial permite que el controlador genere estas señales). En una transmisión síncrona, el reloj de recepción se genera a partir del reloj de transmisión, lo que garantiza la sincronía, por lo que los bits de inicio y detención no son necesarios.

El transmisor y receptor, junto con el generador de reloj y registros *buffer* sobre el *bus* de datos, generalmente se empaquetan juntos en un dispositivo conocido como receptor/transmisor asíncrono universal (*Universal Asynchronous Receiver/Transmitter* o UART) o un receptor/transmisor síncrono/Asíncrono universal (*Universal Synchronous/Asynchronous Receiver/Transmitter* o USART).

6.2.3. Dispositivos de interface con registro y memoria

Algunos dispositivos se organizan como un arreglo de registros, y pueden accederse utilizando operaciones de lectura y escritura de memoria. Esta memoria puede parecer ser de solo lectura para un dispositivo de entrada, o solo de escritura para un dispositivo de salida.

Un dispositivo de acceso directo a memoria (*Direct Memory Access* o DMA) contiene su propio procesador dedicado (a veces, compartido) que puede tomar el *bus* del sistema del procesador principal y transferir datos directamente de la

memoria a los registros de los dispositivos. El DMA se utiliza para dispositivos que requieren una transferencia de grandes cantidades de datos.

6.3. Dispositivos interactivos

Los dispositivos interactivos son aquellos que las personas utilizan para dar y recibir información de la computadora.

Un **dispositivo interactivo** se utiliza para la interacción de la computadora generalmente con un ser humano. Ambos forman parte de lo que se conoce como interface de usuario de la computadora.

6.3.1. Teclado

El teclado es generalmente el mayor medio por el cual el usuario ingresa información e instrucciones a la computadora. Las teclas tienen una organización como en una máquina de escribir, lo que se conoce como *Qwerty* dadas las cinco primeras letras de la parte superior izquierda del teclado. Un teclado normalmente tiene además otras teclas adicionales. Cada tecla tiene un interruptor por debajo, conectado en conjunto en forma de renglones y columnas. Al presionar una tecla, se conecta un renglón con una columna. Ambos se revisan mediante un pequeño procesador. Cuando se oprime una tecla, el procesador envía el código correspondiente (normalmente ASCII).

6.3.2. Ratones y dispositivos apuntadores

Un dispositivo apuntador permite controlar un punto (normalmente representado como una flecha) en la pantalla. La variedad más común de dispositivos apuntadores es el ratón (*mouse*).

El ratón es una pequeña caja con uno a tres botones. Conforme se mueve sobre una superficie, tal movimiento se detecta mediante esferas que giran por debajo del ratón, y es capturado por dos rodillos colocados en ángulo recto, que

se leen mediante un codificador óptico. Los datos que identifican la dirección del movimiento y qué botón se presiona se envían a la computadora mediante una conexión serial.

- un *tracker ball* es simplemente un ratón de cabeza, con una bola más grande que puede moverse directamente con los dedos;
- en las laptops, un borrador (*erasehead*) es un pequeño botón de goma en medio del teclado. Este botón tiene sensores de presión en dos direcciones. Los movimientos se envían a una velocidad proporcional a la presión ejercida;
- un panel de contacto (*touch panel*) es un pequeño cuadrángulo que puede detectar la presencia y posición de las puntas de los dedos colocados sobre su superficie, utilizando una técnica de revisión de conexiones de renglones y columnas, conectadas resistivamente o capacitivamente. Alternativamente, el panel puede medir presión en sus cuatro esquinas, y calcular la posición de las presiones relativas.

Los dispositivos de tipo ratón producen solo eventos de movimiento. El panel de contacto es capaz de sensar la posición absoluta. Si un panel de contacto se hace más grande, se le considera una tableta (*tablet*), un dispositivo apuntador de posición absoluta. Las tabletas operan normalmente con una pluma (*puck*), se se coloca directamente sobre una pantalla especial o *touchscreen*.

6.3.3. Sonido

El sonido se produce por ondas de presión en el aire, que se convierten en señales eléctricas por transductores: bocinas (de señal a sonido) y micrófonos (de sonido a señal). El sistema de sonido de una computadora personal generalmente permite dos canales, uno por cada oído. Las señales de sonido son analógicas, en donde el voltaje varía con la fuerza de la presión de aire. La computadora convierte esto a series de números que representan la intensidad, es decir, una señal digital, usando un convertidor analógico/digital (*Analog to Digital Converter* o ADC) para entrada de sonido, o un convertidor digital/analógico (*Digital to Analog Converter* o DAC) para salida de sonido. Para obtener una fidelidad respecto al sonido original, las señales de sonido deben muestrearse a una velocidad de al menos el doble de la frecuencia de sonido más alta, es decir, al menos 32 mil de muestras por segundo.

Una buena fidelidad se logra mediante usar muestras representadas en 16 bits, que proveen dos bytes de datos por cada muestra. Así, la velocidad de muestreo requerida por cada canal es de al menos 32000×2 bytes por segundo. El sistema de sonido de una computadora maneja normalmente hasta 44100 muestras por segundo a 16 bits por muestra (que es la velocidad de un disco compacto), o de 8000 muestras por segundo a 8 bits por muestra (que es la velocidad de transmisión de una línea telefónica). Los datos deben proveerse continuamente, de modo que las tarjetas de sonido hacen uso del DMA, y se trata de dispositivos en bloque tanto para entrada como para salida.

6.4. Pantallas

La pantalla es el dispositivo más visible de la interface de usuario de una computadora. Aquí se presentan varios tipos de despliegue, y sus características.

6.4.1. El tubo de rayos catódicos

El despliegue o pantalla es el mecanismo principal mediante el cual la computadora presenta información al usuario. La interface entre la computadora y la pantalla se basa en los requerimientos de un tubo de rayos catódicos (*Cathode Ray Tube* o CRT). La imagen se logra mediante proyectar un punto de luz pequeño y brillante a lo largo y ancho de la pantalla, tan rápidamente que parece una continua página de luz. El punto de luz llena la pantalla desde la posición superior izquierda, línea por línea, llenando la pantalla hasta llegar a la posición inferior derecha, desde donde regresa a la posición inicial. La brillantez del punto de luz va variando, partiendo de una señal de video, lo que dibuja en la pantalla la imagen requerida.

Un CRT a color usa tres puntos de luz y tres señales de video correspondientes, una para el rojo, una para el verde, y una para el azul, lo que se conoce como RGB.

El *hardware* del manejador de video divide cada línea en un número igual de elementos pequeños llamados pixels (*picture elements*) cuyo color y brillantez se codifica mediante un número que se coloca en un registro de corrimiento y se envía uno a la vez por cada línea de la pantalla. Esto se alimenta a una combinación de memoria de acceso aleatorio (RAM) y convertidor digital/analógico (RAMDAC), que lo convierte en una señal analógica RGB que se envía por el cable de video al monitor.

El número de pixels que se despliegan en una pantalla se conoce como la **resolución de la pantalla**, y usualmente se presenta como un producto $x \times y$, donde x es el número de pixeles a lo ancho de la pantalla y y es el número de pixeles a lo alto de la pantalla.

Tamaños típicos de pantallas en pixeles van del rango de VGA (*Video Graphics Array*) de 640×480 , a adaptadores especializados de gráficos de 1664×1280 .

El número de colores que pueden desplegarse depende de la RAMDAC, y muchos van de 8 bits (con 256 colores diferentes) a 32 bit (4,294,967,296 colores).

Otro parámetro que afecta la especificación de una pantalla es la **tasa de refresco**, que se refiere a qué tan frecuentemente se redibuja la pantalla de la computadora.

Si la tasa de refresco es muy lenta, el usuario puede ver el parpadeo de la pantalla, lo que es molesto y cansado. A una tasa de refresco de 50 Hz mucha gente puede percibir el parpadeo, pero a 75 Hz pocos pueden.

Una pantalla VGA, usando pixeles de 8 bits y que se refresca a 50 Hz, requiere una velocidad de datos de $307200 \times 1 \times 50 = 15360000$ bytes/segundo (en realidad la tasa pico es más alta que esto por tal vez un 25 %, ya que el punto de luz toma un tiempo para reiniciar).

Una pantalla de 1280×1664 pixeles, usando pixeles de 32 bits y una tasa de refresco de 75 Hz, requiere 638 976 000 bytes/segundo.

Un acceso DMA constante a la memoria a tal velocidad por la pantalla bajaría la velocidad del procesador principal, por lo que el sistema de pantalla tiene su propia memoria dedicada, que puede ser de un tipo especializado de RAM conocido como VRAM (Video RAM) o WRAM (Windows RAM).

La imagen de la pantalla se modifica mediante simplemente modificar el contenido de la memoria de video. La mayoría de los adaptadores de pantalla tienen un procesador de propósito especial diseñado para realizar sus operaciones muy rápidamente. Memoria de video extra es disponible usando memoria caché para el procesador gráfico.

Usando un procesador gráfico, solo es necesario que el procesador principal le envíe las instrucciones de despliegue, para que las maneje y aplique a una gran velocidad.

6.4.2. Pantallas de cristal líquido (LCD)

La alternativa más popular al CRT es la pantalla de cristal líquido (*Liquid Crystal Display* o LCD), que es plana y muy compacta. El cristal líquido modifica la polarización de la luz cuando se le expone a un campo eléctrico. Un LCD, entonces, consiste de un emparedado de vidrio, con polarizadores entrecruzados de modo que no pase luz a través de ellos, y conteniendo el cristal líquido. Los vidrios se recubren con electrodos transparentes en un arreglo rectangular. Cuando se aplica un voltaje entre los píxeles, el cristal líquido tuerce la polarización, permitiendo el paso de la luz. Una lámpara plana detrás de la pantalla genera la luz, y los píxeles se filtran en rojo, verde y azul para dar la coloración.

Interruptores del tipo transistores de película delgada (*Thin Film Transistors* o TFT) en el punto de cada píxel controlan el voltaje del propio píxel, y pueden revisarse para modificar el voltaje del píxel apropiado en el momento apropiado. Cuando el interruptor se abre de nuevo, el píxel permanece cambiado, de modo que un TFT LCD no parpadea.

Es difícil fabricar un LCD perfecto, por lo que tienden a ser costosos y su tamaño limitado.

6.4.3. Pantallas de plasma

Una pantalla de plasma es un panel plano que puede fabricarse en grandes tamaños. Dos capas de vidrio, impreso con patrones de electrodos, encierran una tercera capa perforada con un arreglo de hoyos, uno por cada píxel, y se llenan con un gas a baja presión. Cuando un voltaje alto se coloca en los electrodos, el gas se ioniza, produciendo plasma que emite luz.

La superficie interna de cada píxel se cubre con fósforo, que absorbe luz de un color y emite luz de otro. Los paneles de plasma son costosos, debido tanto al costo de fabricación como a la electrónica de alto voltaje necesaria para controlar el arreglo de píxeles.

6.5. Impresoras

Las impresoras son un medio para producir copias en papel de lo expuesto en una computadora. En esta sección se presentan varios detalles de la tecnología de impresoras.

6.5.1. Tipos de impresoras

Hay dos tipos de impresoras: impresoras de carácter, que imprimen un rango fijo de caracteres alfanuméricos, e impresoras de escaneo, que imprimen un patrón de píxeles sobre el papel. Actualmente, la mayoría de las impresoras son de escaneo. Su calidad se mide por el número de puntos que imprime en el papel. 300 puntos por pulgada (*dots per inch* o DPI) es lo mínimo para una impresión de calidad.

La información a imprimirse se codifica en caracteres mediante un lenguaje de control de impresora, siendo los más comunes el Postscript y el lenguaje de control de impresora de Hewlett-Packard (*Hewlett-Packard Printer Control Language* o HPCL). Los caracteres pueden enviarse mediante una interface paralela, conocida como una interface Centronics o puerto de impresión de PC, o mediante una interface serial.

La impresora se equipa con un procesador de escaneo de imágenes (*Raster Image Processor* o RIP), que convierte la descripción de lenguaje de control de impresora a una página de datos en píxeles. El *hardware* normalmente es una impresión de inyección de tinta (*inkjet*) o una impresora láser.

6.5.2. Impresoras de inyección de tinta

La imagen impresa se compone de gotas de tinta lanzadas de una cabeza de impresión que cubre a lo largo el papel. Después de cada renglón, la impresora mueve el papel, de modo que la impresora pueda imprimir renglón por renglón.

La cabeza consiste de un tanque de tinte conectado a salidas de impresión. éstas son muy pequeñas como para que la tinta saliera por sí misma, de modo que la tinta debe empujarse, ya sea por un pequeño calentador eléctrico, creando un flujo de burbujas, o por diminutas bombas piezoeléctricas. Las impresoras a color replican la cabeza de impresión para el color negro y tres colores primarios sustractivos: amarillo, azul y magenta.

Las impresoras de inyección de tinta modernas producen imágenes de muy alta calidad, cercanas a la calidad fotográfica. Sin embargo, no son adecuadas para impresión en volumen.

6.5.3. Impresoras láser

En este tipo de impresora, un láser se modula con una señal de video para colocar patrones de puntos sobre un cilindro fotosensitivo. Los puntos sobre el cilindro tienen carga eléctrica, lo que atrae granos finos de toner, una cera coloreada. Cuando una hoja de papel se presiona sobre el cilindro y el toner se calienta, una imagen en toner se transfiere al papel. Las impresoras láser pueden producir imágenes de alta calidad rápidamente, pero resultan más caras y de mayor tamaño que las impresoras de inyección de tinta.

6.5.4. Impresoras especializadas

Hay varios tipos de impresoras especializadas, que incluyen impresoras de sublimación seca, las cuales producen imágenes de calidad fotográfica pero muy cara, grandes impresoras/*plotters* de inyección de tinta, *photoplotters* y *photosetters* utilizados en la industria de impresión gráfica.

6.6. Periféricos especiales

Existen otros tipos de periféricos que no entran dentro de las categorías anteriores. Se presentan en esta sección.

6.6.1. Modems

Modem es una contracción de las palabras modulador/demodulador, y es un dispositivo que permite convertir una señal serial lógica a una forma que puede transmitirse a otro medio de comunicación. Los modems sobre sistemas antiguos de telefonía (*Plain Old Telephone System* o POTS) tienen una velocidad de datos máxima de 56 bits/segundo. Modems de banda ancha y cable tienen anchos de banda mayores. Los modems se conectan a las computadoras mediante una interface serial estándar, y algunos tienen integrada una UART en su tarjeta.

6.6.2. Escaners

Los escaners se utilizan para digitalizar imágenes de entrada a los sistemas de cómputo. Trabajan mediante copiar la imagen con un arreglo de celdas sensibles a la luz, produciendo una representación escaneada de la imagen.

6.6.3. Controladores de juegos

Los juegos de video demandan una entrada de control especial, ya sea un *joystick* o un *game pad*, ambos son similares en su funcionamiento a un borrador junto con botones de control. Los *game pads* normalmente se conectan mediante una interface de propósito especial paralela, que se integra comúnmente a la tarjeta de sonido.

6.6.4. Tarjetas de TV

Una tarjeta de televisión (TV) incluye *hardware* para digitalizar entrada de video, y codificar salida de video a una forma adecuada para desplegarse en una televisión estándar.

Fuentes útiles de información

PC Tech Guide.

<http://www.pctechguide.com/>

Evaluación

Preguntas de opción múltiple

1. ¿Cuál es el término no asociado con los tipos de periféricos de una computadora?
 - a) Carácter.
 - b) Escaneo.
 - c) Paralelo.
 - d) Serial.

- e) Pedazo.
2. ¿Qué significa USART?
- a) *Universal System Archive Resource Technique.*
 - b) *Unsynchronized Serial Access Receiver/Transmitter.*
 - c) *Universal Synchronous/Asynchronous Receiver/Transmitter.*
 - d) *Untested Systems Are Really Troublesome.*
 - e) *Unsynchronized Sending and Receiving Technology.*
3. ¿Cuál de los siguientes es un dispositivo “interactivo”?
- a) Controlador de cinta.
 - b) Teclado.
 - c) Disco duro.
 - d) Disco flexible.
 - e) DVD.
4. ¿Cuál de los siguientes no es un dispositivo apuntador?
- a) *Mouse.*
 - b) Borrador.
 - c) Panel de contacto.
 - d) Tecla de cursor.
 - e) *Tracker ball.*
5. ¿Cuál es el dispositivo principal de salida de una tarjeta de sonido?
- a) Bocina.
 - b) Audífonos.
 - c) DAC.
 - d) DAT.
 - e) RAC.

6. ¿Cuál es la tecnología de pantallas más costosa?

- a) CRT.
- b) Plasma.
- c) TCR.
- d) AGP.
- e) VGA.

7. ¿Qué color de tinta no se espera encontrar en una impresora a color?

- a) Rojo.
- b) Negro.
- c) Azul.
- d) Magenta.
- e) Amarillo.

Respuestas a las preguntas de opción múltiple

- 1. e
- 2. c
- 3. b
- 4. d
- 5. c
- 6. b
- 7. a

Capítulo 7

Introducción a sistemas operativos

Resumen

El objetivo de este capítulo es proveer una introducción básica a los conceptos, funciones y componentes de sistemas operativos, y cómo se relacionan con los componentes de la arquitectura de una computadora. No se intenta explicar el funcionamiento interno de un sistema operativo, que es tema de otro curso completo específicamente sobre sistemas operativos.

Objetivos del capítulo

Al final del capítulo, el estudiante debe ser capaz de:

1. Explicar el significado del término sistema operativo.
2. Listar los componentes de un sistema operativo.
3. Explicar la función de la administración de la memoria.
4. Explicar el papel de los sistemas de archivos y los conceptos de archivo, directorio y espacio de nombres de archivos.

5. Definir qué significa un dispositivo en el contexto de un sistema operativo y describir las funciones de la administración de dispositivos.
6. Explicar el papel de los sistemas de interfaz de usuario y los tipos en uso en sistemas operativos modernos.
7. Explicar la necesidad de sistemas multitarea y describir su operación.

7.1. ¿Qué es un sistema operativo?

Esta sección localiza las funciones de los sistemas operativos dentro de la arquitectura de un sistema de cómputo.

7.1.1. La historia

Aquí se presentan las mayores familias de sistemas operativos en uso actualmente.

- Microsoft Windows: los sistemas operativos de Microsoft obtuvieron una atención comercial cuando su predecesor MS-DOS (*Microsoft Disk Operating System*) se adopta por IBM para su primera Computadora Personal en 1982. Provisto de un sistema de archivos en disco magnético, cuenta con medios para cargar y ejecutar programas del mismo. Como una respuesta al sistema de Macintosh (véase abajo), Microsoft integra una interface gráfica de usuario en una nueva capa del sistema operativo, llamándolo MS Windows, que se ejecuta como una aplicación sobre MS-DOS. Más tarde, se desarrolla Windows NT (*New Technology*), que eventualmente se convierte en Windows 2000. El antiguo MS Windows basado en MS-DOS se continúa como Windows 95, Windows 98, y subsecuentemente Windows Me (*Millennium edition*). El sistema operativo Windows es por mucho el más probable de encontrarse en sistemas de Computadoras Personales;
- Apple Macintosh: es la primera computadora personal disponible que provee una interface gráfica de usuario, a finales de los años 1970. El diseño de la Macintosh se deriva de un producto anterior menos exitoso llamado Lisa, que a la vez se inspira en el sistema de oficina Star, de Xerox, el cual inicia el concepto de WIMP (*windows, icons, menus, pointer*) en interfaces de usuario;

- **Unix y Linux:** Unix se desarrolla a finales de los 1970s por los Bell Laboratories (parte de AT&T) como un sistema operativo para computadoras de oficina, que soporta varios usuarios simultáneamente. En los 1980s, Unix se convierte en el estándar para computadoras gráficas de un solo usuario en ingeniería. Para el inicio de los años 90, los derechos de Unix se incluyen en un cuerpo de estándares llamado X-Open, y hoy Unix es la especificación de una interface de programación de aplicaciones (*Applications Programming Interface* o API), donde cualquier sistema operativo que se conforme a ella puede ser considerado como Unix, sin importar su código fuente. Linux es un sistema Unix de código abierto (*open source*) y libre, desarrollado por Linus Thorvalds. Linux se ejecuta en computadoras personales estándar, así como en muchos otros tipos de computadoras.

7.2. Componentes de un sistema operativo

La visión de la estructura de los sistema operativos que se presenta aquí permite entender la estructura y propósito de los sistemas operativos en general.

Los proveedores de diferentes sistemas operativos usan términos diferentes para la misma cosa. Aquí se utilizan los términos genéricos que se encuentran en la mayoría de los libros de texto; sin embargo, es necesario ser cuidadoso en conectar el término propietario con el término genérico.

Capas de abstracción: una descripción del diseño de los sistemas operativos es como proceso de abstracción – reemplazo de detalles complejos por una idea más simple que contiene la esencia sin el detalle. Un sistema operativo abstraer el detalle del *hardware* de la computadora, dejando solo los detalles necesarios para la tarea presente.

Por ejemplo, para controlar un circuito integrado para comunicaciones seriales se puede requerir un número de instrucciones para controlar el *hardware*, pero todo lo que al usuario del programa le interesa es enviar y recibir caracteres.

Un sistema operativo puede verse como una construcción de un número de capas de abstracción, cada una representando una vista de más alto nivel de las funciones del sistema que la capa bajo ella. Si se hace esto, se termina con una figura como un corte horizontal en una cebolla; de ahí el modelo de capas de cebolla (Figura 7.1).

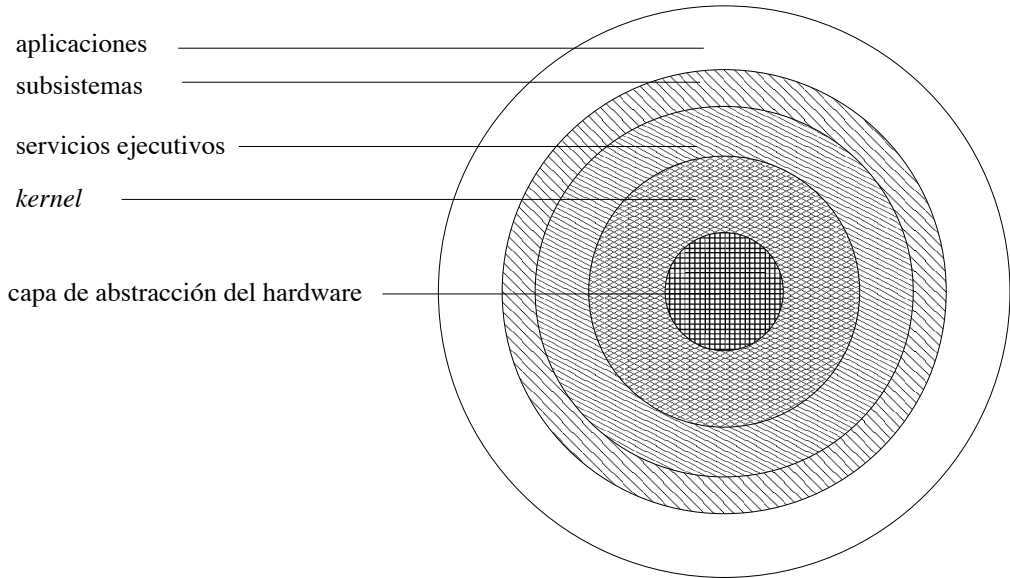


Figura 7.1: Modelo de capas de cebolla mostrando las capas de abstracción que Microsoft usa para su sistema operativo Windows NT.

Es importante no equivocarse con este modelo. Los componentes de *software* reales de un sistema operativo pueden no estructurarse en forma similar de un modelo de capas de cebolla; sin embargo, el diagrama y las capas definidas hacen más fácil entender la función de los sistemas operativos.

Las funciones del modelo de capas de Windows NT son como sigue:

1. Capa de aplicación: esta capa consiste en programas de aplicación que se ejecutan por parte del usuario.

2. Capa de subsistemas: esta capa provee de interfaces programadas de aplicación (*Application Programming Interface* o API). En el sistema NT se mantienen un número de APIs alternativas.
3. Capa de servicios ejecutivos: esta capa provee la mayoría de las funciones del sistema operativo tales como manejadores (*drivers*) administración de la memoria, administración de ventanas e interacción con los dispositivos gráficos.
4. Capa *kernel*: esta es una parte crucial del sistema operativo. Soporta la ejecución simultánea de programas dentro de la capa de servicios ejecutivos, que proveen los servicios del sistema operativo.
5. Capa de abstracción del *hardware*: esta contiene todo el código dependiente del procesador, dando una interface independiente del *hardware* para la capa del *kernel*.

API es un acrónimo de *Application Programming Interface*, y se trata de un conjunto de instrucciones del sistema operativo o llamadas al sistema, que un programador produciendo una aplicación para ejecutarse en un sistema operativo dado requiere conocer.

Los servicios que proveen la mayoría de los sistemas operativos actuales y se accesan vía una API son:

- administración de memoria;
- sistema de archivos;
- administración de procesos;
- interface de usuario.

7.3. Administración de memoria

Aquí se presenta el componente del sistema operativo que asegura que diferentes programas compartiendo una computadora tengan acceso a la memoria que necesitan, sin interferir entre sí.

Una de las labores del sistema operativo es compartir la memoria disponible entre diferentes programas. A esta labor se conoce como **administración de la memoria**.

La administración de la memoria simplemente consiste de dos tareas: alojamiento de memoria y mapeo de memoria.

El alojamiento de memoria es el proceso por el que la memoria disponible se comparte entre los programas. El primer programa que requiere alojarse en la memoria es el propio *kernel* del sistema operativo, y después otros programas que requieren de recursos de memoria cuando se cargan, además de otros recursos adicionales que usan mientras se ejecutan. Cuando un programa ha terminado su ejecución, sus recursos de memoria se hacen disponibles para alojar otros programas.

Un programa necesita los siguientes recursos de memoria:

1. La memoria del programa. En lo que concierne al programa, ésta es de solo lectura.
2. La memoria de datos. Esta puede tratarse como de solo lectura o de lectura/escritura, dependiendo del ambiente del programa.
3. La pila (*stack*) de memoria. Es parte de la memoria que se encarga de guardar la pila del programa. La pila normalmente contiene las direcciones de retorno de los procedimientos, valores de los parámetros usados cuando se llaman los procedimientos y las variables locales. ésta es de lectura/escritura.
4. La memoria de montón (*heap*). Es un recurso de memoria dinámica que se provee por algunos sistemas de programación. Es de lectura/escritura.

Frecuentemente, las memorias de datos y de montón se incluyen dentro de la memoria de pila, de modo que el programa se representa mediante dos localidades de memoria, el programa y la pila.

Un término usado frecuentemente para describir un trozo de memoria con un propósito particular, como pila o memoria de programa, es **segmento de memoria**.

Un segmento se describe frecuentemente usando dos números: un registro base, que da la dirección inicial del segmento, y un registro límite, que contiene la dirección más alta permisible que puede usarse en el segmento. Si la dirección de memoria es mayor que el límite, se previene que la instrucción accese a la memoria y se levanta una interrupción o excepción de memoria. Esto evita la modificación inadvertida de datos de otros segmentos por errores en el programa. En ocasiones, se utilizan registros base y tamaño.

Las tareas de alojamiento de memoria del *kernel* del sistema operativo son:

- llevar cuenta de la memoria de la computadora, ya sea que se encuentre libre o utilizada;
- alojar y desalojar memoria como se vaya necesitando por los programas;
- arreglar las tablas necesarias por el mapeo de memoria;
- resolver las condiciones cuando los programas tratan de acceder memoria fuera de su provisión;
- reorganizar los datos almacenados en la memoria para permitir un uso óptimo de la memoria del sistema.

Las **direcciones lógicas** son las direcciones de memoria tal y como aparecen en el código del programa. Las **direcciones físicas** son las direcciones utilizadas en el *hardware*. Hay un proceso de mapeo de memoria entre las direcciones lógicas y físicas.

Las tareas de mapeo de memoria se realizan para:

- traducir entre las direcciones lógicas del programa a direcciones físicas de la memoria;
- detectar accesos de memoria ilegales, que pueden ser una dirección fuera del segmento permitido o modos de acceso no permitidos para ese segmento;
- transferir el control al *kernel* del sistema operativo usando una interrupción o excepción de memoria si ocurre un acceso de memoria ilegal.

El mapeo de memoria depende del *hardware* de administración de memoria de la computadora, que generalmente es una parte integral del procesador. Puede seguir tres maneras: segmentada, paginada, o una combinación de ambas.

7.3.1. Mapeo en memoria segmentada

La administración de **memoria segmentada** divide el espacio de direcciones en algunos segmentos. Pueden ser de longitudes diferentes y corresponder a un uso particular del espacio de memoria, ya sea memoria de programa o de pila.

El registro de segmento contiene la dirección base del segmento, que se suma a la dirección del dato o segmento base (multiplicado por un número para incrementar el rango de direcciones de la computadora) para dar la dirección física para un acceso a memoria. La dirección física de los segmentos debe estar dentro del límite dado por el multiplicador. Puede haber un registro límite o tamaño para el segmento. La Figura 7.2 muestra estas operaciones.

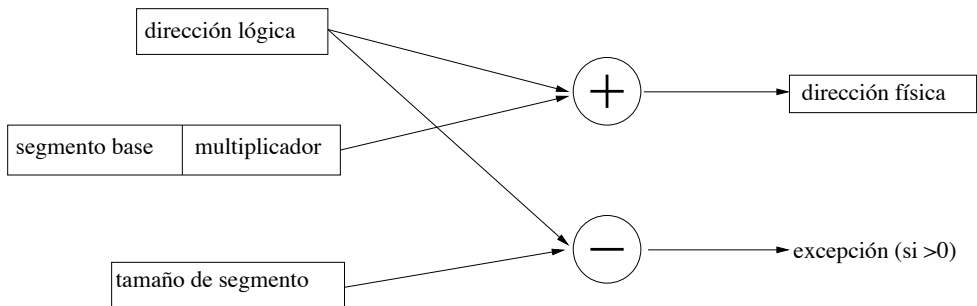


Figura 7.2: Un mapeo de memoria segmentada.

El registro de segmento puede ser seleccionado usando los bits más significativos de la dirección lógica, o puede ser implícito en la codificación de la instrucción, como por ejemplo, en el caso de la arquitectura 80X86.

7.3.2. Mapeo en memoria paginada

La administración de **memoria paginada** divide el espacio de direcciones en muchas páginas. Las páginas son todas del mismo tamaño (de 512 bytes a algunos kilobytes). El alojamiento se realiza mediante la administración de **tablas de páginas**.

Una tabla de páginas es simplemente una memoria pequeña y veloz. La dirección de esta memoria se alimenta con el número de la página, y su salida provee la dirección física de la página. La manera más simple de derivar la entrada a la tabla de páginas es seleccionar la parte más significativa de la dirección (Figura 7.3).

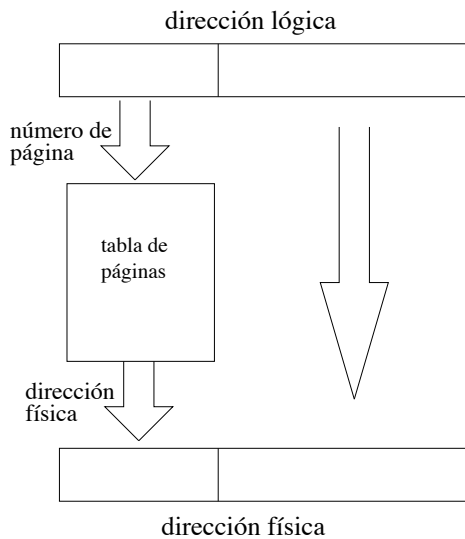


Figura 7.3: Un mapeo de memoria paginada.

Bits adicionales se pueden incluir en la tabla de páginas para dar mayor información acerca de la página, como por ejemplo, “en uso” o “solo lectura”. Cualquier violación causa una excepción de memoria. Una tabla de páginas puede hacerse para emular la mayoría de las características de la memoria segmentada:

- el segmento se determina por los bits más significativos de una dirección lógica;
- el límite de ese segmento se determina mediante marcar la siguiente página como “no en uso”.

Todo alojamiento de memoria debe hacerse en unidades de tamaño de página. La tabla de páginas también puede relocalizar páginas individuales, de modo que lo que parece ser un segmento continuo en memoria lógica puede realmente estar distribuido a lo largo de toda la memoria física, evitando una gran cantidad de copiado de un área de memoria a otro para liberar un área grande y continua.

Iniciar un nuevo segmento significa escribir un nuevo número a la tabla de páginas, mientras que en un esquema segmentado solo se requieren los registros base y límite para iniciarlo. Una tabla de páginas puede utilizarse como un segundo nivel de una administración de memoria segmentada, como se ha descrito anteriormente.

Una de las funciones de la capa de abstracción del *hardware* es traducir entre el modelo de memoria utilizado por el sistema operativo y aquel que se soporta por el *hardware*. Algunas veces, no es posible ocultar completamente las características del *hardware*, conduciendo a incompatibilidades entre versiones del mismo sistema operativo que se ejecuta en diferente *hardware*.

Almacenamiento de respaldo (*backing store*) significa que una memoria más lenta y de bajo costo, típicamente un disco magnético, se utiliza para guardar datos de la memoria principal, que pueden restablecerse o intercambiarse cuando sea necesario.

Los programas del almacenamiento de respaldo, que en total pueden utilizar más espacio que la memoria principal disponible, pueden ejecutarse juntos mientras haya suficiente almacenamiento de respaldo disponible para guardar los datos para todos ellos.

7.3.3. Memoria virtual

La memoria virtual depende de una pequeña modificación de los esquemas de administración de memoria descritos anteriormente. Si el procesador se diseña de

modo que una instrucción que causa una excepción pueda ejecutarse de nuevo o continuar desde el punto en que se causa la excepción, entonces el *kernel* puede intercambiar (*swap*) los datos faltantes del almacenamiento de respaldo a la memoria principal y reejecutar (o continuar) la instrucción, que opera como si los datos faltantes hubieran estado ahí desde un principio.

La memoria virtual puede operar con administración de memoria segmentada o paginada, pero el esquema paginado tiene la ventaja de que apenas una página de los datos necesarios debe estar en memoria principal en cualquier momento, mientras que en el esquema segmentado se requiere todo el segmento.

7.4. Sistemas de archivos

El sistema de archivos es un componente del sistema operativo que permite llevar cuenta del uso del almacenamiento secundario de una computadora (discos magnéticos, CDs, DVDs, y demás). La mayoría de los sistemas de archivos se estructuran alrededor de unos cuantos conceptos simples, que se introducen a continuación.

7.4.1. Lo que los sistemas de archivos hacen

El almacenamiento secundario de una computadora es tan grande que es impráctico accederlo usando direcciones numéricas simples. Es deseable proveer alguna abstracción de alto nivel para la estructura del almacenamiento de una computadora. A esto se le conoce como archivo (en analogía con los archivos de una oficina).

Un **archivo** es una colección de datos asociada con un nombre que se almacena en un medio de memoria secundaria, tal como un disco magnético. Además de su nombre, un archivo puede también tener un número de atributos que controlan su uso.

7.4.2. Bloques y bytes

Aún cuando los discos se organizan en bloques de datos, los sistemas operativos proveen de archivos como una secuencia con una longitud arbitraria de caracte-

res o bytes. Si una aplicación requiere de una estructura diferente (por ejemplo, un programa de base de datos que usa campos de datos), entonces depende del programador del sistema proveer de la estructura requerida.

7.4.3. Sistemas de archivos

Un **sistema de archivos** es el nombre dado al medio por el cual los archivos se crean, se les nombra, se leen y escriben, y se borran, y por el cual los recursos reales de almacenamiento se colocan en archivos.

Típicamente, un sistema operativo soporta un conjunto de de llamadas al sistema para actuar sobre los archivos, que generalmente incluyen las siguientes operaciones:

- crear un archivo; esta operación prepara todas las estructuras necesarias del sistema de archivos para el nuevo archivo;
- nombrar (o renombrar) un archivo;
- leer de un archivo;
- escribir a un archivo;
- borrar un archivo; remover el nombre de un archivo, borrar las referencias del sistema de archivos al archivo y regresar los bloques de almacenamiento que utilizaba para su reuso por nuevos archivos.

Frecuentemente, hay varias variantes de cada una de estas operaciones.

7.5. Administración de dispositivos

Un sistema operativo permite a los programas de aplicación trabajar con cualquier tipo de dispositivo que la computadora contenga, y permite también cambiar los dispositivos sin hacer al *software* obsoleto.

Un **dispositivo** es un nombre dado a cualquier pieza de *hardware* de la computadora fuera del procesador y la memoria principal.

7.5.1. ¿Porqué los dispositivos necesitan una administración?

Diferentes tipos de dispositivos, como se discute en el Capítulo 6, pueden tener diferentes requerimientos en términos de control y temporización de datos. Un sistema operativo debe proveer del *software* que controla un dispositivo en particular, dando al programador de aplicaciones un modelo más simple que el dispositivo real.

Independencia de dispositivos es una característica de una API que libera a los programadores de aplicaciones de tener que programar específicamente para un tipo o modelo de dispositivo. La API provee de definiciones de interface estandarizadas y genéricas, tales como de dispositivos de bloque o flujo, e interface gráfica adecuada. Esto permite a los programas de aplicación ejecutarse en una variedad de configuraciones de *hardware* sin cambiar el propio *hardware*.

El sistema operativo traduce los comandos estandarizados a los conjuntos de comandos requeridos por el dispositivo real de *hardware*. El mecanismo que utiliza el sistema operativo se conoce como manejador de dispositivo (*device driver*).

Un **manejador de dispositivo** es un módulo de *software* dentro del sistema operativo que traduce del modelo de dispositivo genérico provisto por la API a los requerimientos específicos de *hardware* del dispositivo real.

Hay muchas formas de proveer manejadores de dispositivos. Pueden estar incluidos en el *kernel* del sistema operativo, o ser externos al *kernel*. El sistema operativo normalmente se reconfigura con la inclusión del manejador requerido cada vez que el dispositivo se ajusta o cambia.

7.6. Administración de procesos y multitareas

Los sistemas operativos deben ser capaces de ejecutar varios programas concurrentemente, sin que tales programas se interfieran o detengan entre sí por su ejecución.

Un **proceso** o **tarea** es la modificación del estado de la memoria de la computadora, por la acción del procesador. Un programa es la especificación de uno o varios procesos. El programa puede ser de aplicación, o realizar alguna función de soporte del sistema operativo. En un sistema multitarea, el sistema operativo se encarga de crear la ilusión para cada proceso de que usa su propia computadora o **máquina virtual**.

Mantener una máquina virtual depende de dos cosas. La primera es la administración de la memoria, y la segunda es compartir (concurrentemente, en tiempo) a un solo procesador entre los procesos. Esto se logra mediante el tiempo compartido (*time slicing*), que significa que el procesador ejecuta una parte del código de cada proceso en turnos, como sigue. El estado de un proceso (que es los valores de todas las variables y la siguiente instrucción a ser ejecutada) se almacena en la pila de procesos (*process stack*). Para ejecutar un proceso, el apuntador de pila del procesador se carga con la dirección de la pila para ese proceso, siendo el primer elemento de la pila la dirección de la siguiente instrucción de cada proceso, que se obtiene y se carga en el contador de programa. Para detener un proceso (o suspenderlo) la dirección de la siguiente instrucción a ser ejecutada se vuelve a almacenar en la pila, y el control se pasa al planificador (*scheduler*).

Un **planificador** es el componente del sistema operativo multitarea, que se encarga de ejecutar o suspender los procesos.

El planificador mantiene un número de listas o colas de procesos, cada uno identificado con un apuntador de pila para un proceso. Las colas son:

- ejecutándose (*running*): esta cola contiene un proceso en una máquina de un solo procesador;
- listo para ejecutarse (*ready to run*): estos son los procesos que pueden ejecutarse, pero no tienen uso del procesador por el momento;

- esperando o suspendido (*waiting or suspended*): estos son procesos que no pueden ejecutarse debido a que esperan que suceda un evento, tal como que algún dato se reciba por un dispositivo.

El planificador mueve los procesos entre estas colas conforme se hacen disponibles para ejecutarse, se suspenden, o se ejecutan.

Los procesos se suspenden cuando no pueden ejecutarse, ya sea porque el proceso ha finalizado (en cuyo caso se suspende para ser después borrado) o el proceso necesita esperar por un dispositivo u otro proceso (en cuyo caso, se le coloca en la cola de espera). Muchos sistemas operativos operan con una planificación preventiva (*preemptive scheduling*), lo que significa que después de ejecutarse por un tiempo corto, un proceso se suspende, ya sea que pueda o no ejecutarse aun. Los procesos preventivos se colocan al final de la cola de “listo para ejecutarse”. La planificación preventiva comparte el tiempo disponible de procesador entre todos los procesos, y resulta en un sistema con mejores propiedades interactivas.

Los procesos necesitan comunicarse para cooperar, de modo que el sistema operativo debe contar con medios para la **comunicación interproceso**, que debe proveer de comunicación de datos y sincronización. Un modelo común utilizado es permitir el paso de mensajes entre procesos. En este modelo, la comunicación interproceso puede usar las mismas llamadas a sistema como las comunicaciones mediante dispositivos. Los *pipes* usados en Unix son un ejemplo de este tipo de comunicación interproceso.

Otro modelo es proveer un mecanismo específico para la propagación de eventos entre procesos. Los eventos encapsulan sincronización y datos, y pueden manejarse bien en sistemas *orientados a objetos*. Las comunicaciones basadas en eventos son típicamente utilizadas en sistemas WIMP.

Fuentes útiles de información

Patterson, D.A. and Hennesy, J.L.

Computer Organization and Design: The Hardware/Software interface.

Morgan Kaufmann, 2017

Stallings, W.
Operating Systems.
Prentice-Hall, 2000.

Tannembaum, A.S.
Modern Operating Systems.
Prentice-Hall, 2001.

Evaluación

Preguntas

1. El procesador 8086 original tiene un esquema de segmentación en el que los valores se almacenan en un registro de segmento de 16 bits (llamado *Code Segment* o CS, *Data Segment* o DS, *Stack Segment* o SS, y *Extra Segment* o ES), se multiplica por 16 y se suma a un valor de 16 bits en memoria arrojado por la instrucción. Suponiendo los valores del segmento de registro dados en la tabla siguiente (en hexadecimal), responda las preguntas siguientes:

CS	0100h
DS	8721h
SS	FFF0h
ES	0000h

- a) ¿Cuál es la longitud en bits de la dirección de memoria que arroja la operación del segmento?
- b) ¿Cuál es la dirección física dada por una dirección en memoria de 0500h en el CS?
- c) ¿Cuál es la dirección física dada por una dirección en memoria de 0000h en el DS?
- d) ¿Qué dirección de memoria en ES da la misma dirección física que un valor de 0100h en el CS?
- e) ¿Qué dirección de memoria en ES da la misma dirección física que un valor de 0000h en el SS?

2. Los procesadores de 32 bits x86 tienen un esquema de administración de la memoria paginada con dos niveles en la tabla de páginas. La dirección virtual se divide como sigue:

dirección virtual		
31	21	11 0
directorio	tabla de página	página

- El direccionamiento en la página utiliza los bits 0 a 11. ¿Cuál es el tamaño de la página?
- El direccionamiento de la tabla de páginas usa los bits del 12 al 21. ¿Cuántas páginas hay en cada tabla de páginas?
- ¿Qué cantidad de memoria representa una tabla de páginas?
- El direccionamiento del directorio usa los bits del 22 al 31. ¿Cuántas tablas de páginas hay en el directorio?

Respuestas a las preguntas

- 16 bits multiplicados por 16 (cuatro bits) es igual a 20 bits.
 - $(0100 \times 10) + 0500 = 01500h$.
 - $(8721 \times 10) + 0000 = 87210h$.
 - Dirección física en CS: $(0100 \times 10) + 0100 = 01100h$;
Base de ES: $(0000 \times 10) = 00000h$;
Dirección en ES: $01100 - 00000 = 1100h$.
 - Dirección física en SS: $(FFF0 \times 10) + 0000 = FFF00h$;
Base de ES: $(0000 \times 10) = 00000h$;
Dirección en ES: $FFF00 - 00000 = FFF00h$ (esta es una dirección inválida, ya que es mayor que 16 bits).
- El campo es de 12 bits. El número de bytes por página es $2^{12} = 4096$.
 - El campo es de 10 bits. El número de páginas en cada tabla es $2^{10} = 1024$.

- c) 1024 páginas de 4096 bytes es $2^{22} = 4194304$.
- d) El campo es de 10 bits. El número de tablas de páginas es $2^{10} = 1024$.

Preguntas de opción múltiple

1. ¿Qué son las diferentes capas en el modelo de cebolla?
 - a) Capas de *hardware*.
 - b) Capas de *software*.
 - c) Capas de grasa.
 - d) Capas de abstracción.
 - e) Capas de programación.
2. ¿Cuál de los siguientes servicios no se provee por la capa de servicios ejecutivos de Windows NT?
 - a) Manejadores de dispositivos.
 - b) Administración de la memoria.
 - c) Masaje ejecutivo.
 - d) Administración de ventanas.
 - e) Interacción de dispositivos gráficos.
3. ¿Qué significa API?
 - a) *Abstraction Programming Inicialization*.
 - b) *Applications Programming Interface*.
 - c) *Advanced Protocol Implementation*.
 - d) *Applications Protocol Interface*.
 - e) *Automatic Plug-In*.
4. ¿Cuál de los siguientes no se considera normalmente un dispositivo en un sistema operativo?
 - a) Impresora laser.
 - b) Disco magnético.

- c) CD-ROM.
 - d) Sistema de memoria.
 - e) Adaptador Ethernet.
5. ¿Qué es un manejador de dispositivo?
- a) Un especialista operador de computadoras.
 - b) Un tipo de panel de control.
 - c) Un sistema de diagnóstico para el *hardware*.
 - d) Un sistema de instalación de *hardware*.
 - e) Un módulo de *software* que conecta con *hardware* periférico.
6. ¿Cuál frase describe la habilidad de ejecutar más de un proceso?
- a) Multifunción.
 - b) Interactivo.
 - c) Multitarea.
 - d) 32 bits.
 - e) Nueva tecnología.
7. ¿Qué almacena una pila de procesos (*process stack*)?
- a) El estado del proceso.
 - b) Un sujetapapeles.
 - c) Código de programa.
 - d) Bytes desechados.
 - e) Memoria libre.
8. ¿Cuál de los siguientes no describe un posible estado de un proceso?
- a) Ejecutando.
 - b) Ilegal.
 - c) Listo para ejecutarse.
 - d) Suspendido.
 - e) Esperando.

Respuestas a las preguntas de opción múltiple

1. d
2. c
3. b
4. d
5. e
6. c
7. a
8. b

Bibliografía

- [1] Bartee, T. C. *Computer Architecture and Logic Design*. McGraw-Hill, 1991.
- [2] Gajski, D. D. *Principios de Diseño Digital*. Pearson Education, 1997.
- [3] Ghausi, M. S. *Circuitos electrónicos discretos e integrados*. Interamericana, 1987.
- [4] Hayes, J. P. *Diseño de Sistemas Digitales y Microprocesadores*. McGraw-Hill, 1984.
- [5] Mano, M. M. *Arquitectura de Computadoras*. Prentice-Hall, 1983.
- [6] Mano, M. M. *Diseño Digital*. Prentice-Hall, 1987.
- [7] Mano, M. M. *Lógica Digital y Diseño de Computadores*. Prentice-Hall, 1979.
- [8] Mano, M. M. *Computer System Architecture*. Prentice-Hall, 1993.
- [9] Mano, M. M. y Kime, C. R. *Logic Computer Design Fundamentals*. Prentice-Hall, 1997.
- [10] Murdocca, M. J. y Heuring, V. P. *Principios de Arquitectura de Computadoras*. Prentice-Hall, 2002.
- [11] Protopapas, D. A. *Microcomputer Hardware Design*. Prentice-Hall, 1998.
- [12] Stallings, W. *Organización y Arquitectura de Computadores*. Prentice-Hall, 2000.
- [13] Taub, H. y Schilling, D. *Digital Integrated Electronics*. McGraw-Hill, 1977.

- [14] Tocci, R. J. y Laskowski, L. P. *Microprocessors and Microcomputers*. Prentice-Hall, 1982.
- [15] Wiatrowski, C. A. y House, C. H. *Circuitos Lógicos y Sistemas de Microcomputadoras*. Limusa, 1987.

**Diseño de sistemas digitales
y arquitectura de computadoras**
editado por la Facultad de Ciencias de la
Universidad Nacional Autónoma de México,
se terminó de imprimir el 5 de octubre de 2019
en los talleres de Gáfica premier S. A. de C. V.
Cinco de febrero 2309, San Jerónimo Chicahualco
Metepéc. C.P. 52170. Estado de México

El tiraje fue de 500 ejemplares.

Está impreso en papel book creamy de 60 g.
En su composición se utilizó tipografía
Computer Modern de 11:13.5, 14:16 y 16:18 puntos de pica.

Tipo de impresión: offset.

El cuidado de la edición estuvo a cargo de
Mercedes Perelló Valls