

Estructura de Computadores y Periféricos

El conocimiento de la estructura de un computador es fundamental dentro de los estudios de Ingeniería Informática. El análisis de la interrelación existente entre los subsistemas que forman el computador, desde la unidad central de proceso, hasta los periféricos que hacen de interfaz hombre-máquina, es una disciplina obligada para poder comprender el funcionamiento global del sistema.

El texto nace de la experiencia de los autores en la enseñanza de estructura de computadores y periféricos en los primeros cursos de los estudios universitarios de Ingeniería Informática, y presenta los conceptos fundamentales de la estructura de computadores y de los periféricos más habituales. Debido a la amplitud de los temas tratados, el objetivo de los autores ha sido el de elaborar una guía docente, haciendo especial hincapié en el compromiso costo-prestaciones inherente en el diseño de un computador.

Los temas del libro han sido ordenados con la intención de presentar una descripción unificada de la estructura básica de un computador. Inicialmente se muestran las unidades funcionales básicas del procesador: unidad de control y unidad aritmético-lógica, describiendo su estructura y cómo se integran en el funcionamiento del computador. A continuación el texto aborda el estudio de la jerarquía de memoria y de los buses, que junto con el subsistema de entrada/salida son la base clásica de la estructura de computadores. Posteriormente se tratan los periféricos junto con las interfaces que permiten su interconexión al computador: los dispositivos de entrada de datos, de copia impresa, terminales de video y almacenamiento masivo. Por último, se realiza una introducción a sistemas informáticos con mayores prestaciones y arquitecturas más avanzadas. Cada tema incluye cuestiones que pueden ser útiles para repasar los contenidos, así como referencias bibliográficas.

ISBN 970-15-0690-1

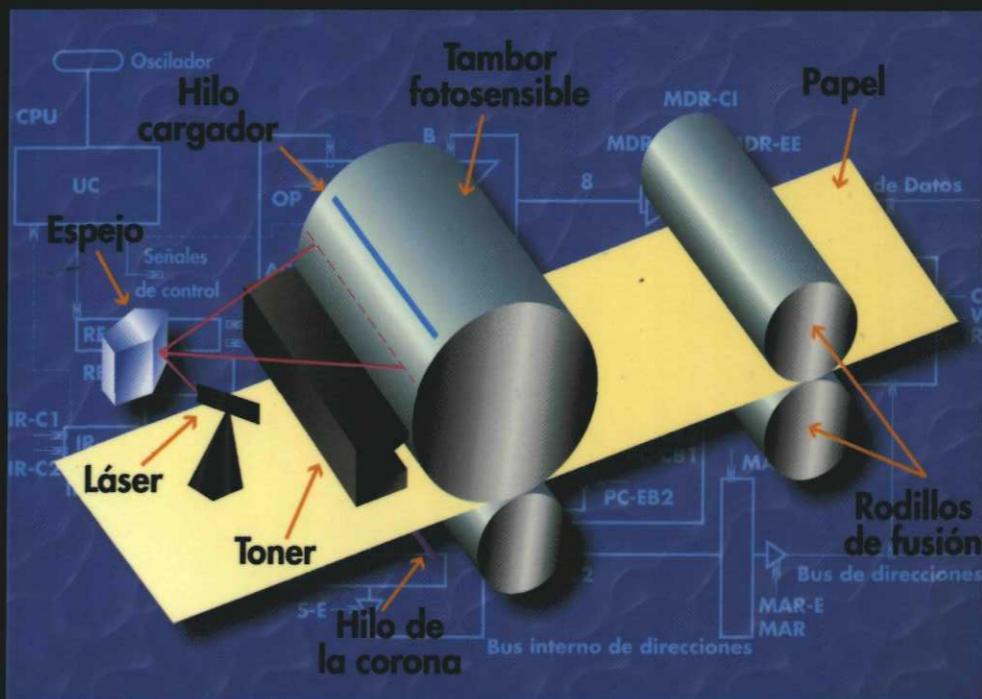


9 789701 506905



Estructura de Computadores y Periféricos

Estructura de Computadores y Periféricos



Rafael J. Martínez Durá
José A. Boluda Grau • Juan J. Pérez Solano

Estructura de computadores y periféricos



Estructura de computadores y periféricos

Rafael J. Martínez Durá,
José A. Boluda Grau,
Juan J. Pérez Solano

Departamento de Informática
Universitat de Valencia

Alfaomega  **Ra-Ma**

Estructura de Computadores y Periféricos
© Rafael J. Martínez Durán, José A. Boluda Grau, Juan J. Pérez Solana

**ISBN 84-7897-447-4, edición original publicada por RA-MA Editorial,
MADRID. España. Derechos reservados © RA-MA Editorial**

MARCAS COMERCIALES: RA-MA ha intentado a lo largo de este libro distinguir las marcas registradas de los términos descriptivos, siguiendo el estilo de mayúsculas que utiliza el fabricante, sin intención de infringir la marca y sólo en beneficio del propietario de la misma.

© 2001 ALFAOMEGA GRUPO EDITOR, S.A. de C.V.
Pitágoras 1139. Col. Del Valle, 03100 México. D.F.

Miembro de la **Cámara Nacional de la Industria Editorial Mexicana**
Registro No. 2317

Internet: <http://www.alfaomega.com.mx>
Email: ventas1@alfaomega.com.mx

ISBN 970-15-0690-1

© 2001 ALFOMEZA SA.
Transv. 24 No. 40-44
Bogotá, D.C., Colombia
E-mail: scliente@alfaomega.com.co
ISBN: 958-682-303-2

Derechos reservados.
Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario & los derechos del copyright.

NOTA IMPORTANTE
La información contenida en esta obra tiene un *S* exclusivamente didáctico; por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Edición autorizada para venta en México y todo el continente americano

Impreso en Colombia - Printed in Colombia

ÍNDICE

PRÓLOGO	xxi
CAPÍTULO 1. INTRODUCCIÓN A LA ESTRUCTURA DE COMPUTADORES	1
1.1 Conceptos preliminares	1
1.2 Arquitectura clásica de un computador: Modelo Von Neumann	4
1.2.1 Unidad central de proceso	6
1.2.2 Memoria	10
1.2.3 Entrada/Salida	12
1.2.4 Sistemas de interconexión: Buses	13
1.2.5 Periféricos	15
1.3 Ejecución de una instrucción	15
1.3.1 El sistema operativo	16
1.3.2 Lenguajes de alto nivel, ensamblador y código máquina	16
1.3.3 Flujo de datos	19
1.4 Tecnología de computadores	20
1.4.1 Tecnologías de circuitos integrados	21
1.4.2 Circuitos de memoria	24
1.5 Cuestiones	28
1.6 Bibliografía	29

CAPÍTULO 2. UNIDAD DE CONTROL: EJECUCIÓN DE INSTRUCCIONES 31

2.1	introducción	31
2.2	Repertorio de instrucciones	32
2.3	Modos de direcccionamiento	37
2.4	Formato de instrucciones	45
2.5	Arquitectura de un computadorelemental	50
2.5.1	Esquema del computador elemental	50
2.5.2	Operaciones con registros	56
2.5.3	Instrucciones	58
2.6	Ejecución de las instrucciones	60
2.6.1	Cronograma de ejecución de la instrucción ADD #456	61
2.6.2	Ejecución de la instrucción SW #456	66
2.6.3	Ejecución de la instrucción JMZ #456	67
2.7	Subrutinas	69
2.8	Excepciones	72
2.9	Diseño de la Unidad de Control	75
2.9.1	Unidad de control cableada	76
2.9.2	Unidad de control microprogramada	77
2.10	Evolución de los procesadores	83
2.10.1	Aumento de las prestaciones	83
2.10.2	Evolución de la arquitecturaintel iX86	85
2.10.3	Computadores RISC y CISC	86
2.11	Conclusiones	89
2.12	Cuestiones	89
2.13	Bibliografía	93

CAPÍTULO 3. UNIDAD ARITMÉTICO LÓGICA 95

3.1	Introducción	95
3.2	Estructura y operaciones de la ALU	97
3.3	Operaciones de desplazamiento	101
3.4	Operaciones lógicas	103
3.5	Operaciones de cambio y extensión de signo	104
3.6	Suma y resta	106
3.6.1	Sumador elemental binario	106
3.6.2	Sumadores w n acarreo adelantado	107

3.6.3	Resta de números enteros	109
3.6.4	Sumadores en BCD	110
3.7	Multiplicación y división de números enteros	111
3.7.1	Multiplicación de números enteros sin signo	112
3.7.2	Multiplicación de números enteros en complemento a 2: Algoritmo de Booth	117
3.7.3	División de números enteros sin signo	119
3.8	Representación de números en coma flotante	122
3.8.1	El estándar de precisión simple de IEEE	122
3.8.2	Operaciones en coma flotante	124
3.9	Cuestiones	126
3.10	Bibliografía	128

CAPÍTULO 4. JERARQUÍA DE MEMORIA **129**

4.1	Introducción	129
4.2	El principio de localidad	132
4.3	Memoriocache	135
4.3.1	Mapeado directo	137
4.3.2	Mapeado asociativo por conjuntos	140
4.3.3	Mapeado totalmente asociativo	141
4.3.4	Algoritmos de reemplazo	142
4.3.5	Manejo de los fallos en los accesos a la cache	142
4.3.6	Rendimiento de la cache	144
4.3.7	Diseño del sistema de memoria	146
4.4	Memoria virtual	148
4.4.1	Mecanismo de traducción de direcciones	149
4.4.2	Acceso a memoria	152
4.4.3	Diseño del sistema de memoria virtual	153
4.4.4	Memoria segmentada	154
4.5	Segmentación paginada	156
4.6	Conclusiones	156
4.7	Cuestiones	158
4.8	Bibliografía	164

CAPÍTULO 5. INTERCONEXIÓN ENTRE PROCESADOR Y PERIFÉRICOS 165

5.1	Introducción	165
5.2	Clasificación de los dispositivos de E/S	167
5.3	Módulos de Entrada/Salida	169
5.4	Programación de las órdenes en los dispositivos	170
5.5	La comunicación con el procesador. Sincronización	172
5.5.1	Sincronización por prueba de estado	173
5.5.2	Sincronización por interrupción	173
5.6	Transferencia de datos entre un dispositivo y memoria	175
5.6.1	Arbitraje del bus y transferencias de datos	178
5.7	Canales y procesadores de E/S	179
5.8	Conclusiones	180
5.9	Cuestiones	181
5.10	Bibliografía	182

CAPÍTULO 6. BUSES 185

6.1	Introducción	185
6.2	Características de un bus	186
6.3	Jerarquía de buses	190
6.4	Protocolos de bus	192
6.5	Arbitraje del bus	196
6.6	Interrupciones	199
6.7	Control de errores	201
6.8	Ejemplos de buses comerciales	202
6.8.1	Bus PCI	203
6.8.2	Bus VME	209
6.9	Conclusiones	213
6.10	Cuestiones	213
6.11	Bibliografía	214

CAPÍTULO 7. EL ENLACE EXTERIOR 215

7.1	Introducción	215
7.2	Tipos de interfaces	216
7.3	Interfaces serie	217

7.3.1	La interfaz RS-232C	218
7.3.2	El Bus Serie Universal (USB)	220
7.4	Interfaces paralelas	226
7.4.1	La interfaz Centronics	226
7.4.2	El estándar IEEE 1284	228
7.4.3	Small Computer Systems Interface (SCSI)	228
7.5	Conclusiones	235
7.6	Cuestiones	236
7.7	Bibliografía	236
CAPÍTULO 8. DISPOSITIVOS DE ENTRADA DE DATOS		239

8.1	Introducción	239
8.2	Interruptores	240
8.2.1	El problema de los rebotes	241
8.3	Teclados	243
8.4	Ratones	247
8.5	Displays gráficos interactivos	250
8.5.1	Lápiz óptico	250
8.5.2	Pantallas sensibles al tacto	251
8.5.3	Tabletas digitalizadoras	252
8.6	Digitalizadores de imágenes (escáneres)	253
8.6.1	Funcionamiento	254
8.6.2	Calidad del digitalizador	255
87	Conclusiones	257
8.8	Cuestiones	258
8.9	Bibliografía	259

CAPÍTULO 9. DISPOSITIVOS DE COPIA IMPRESA **261**

9.1	Introducción	261
9.2	Clasificación de las impresoras	263
9.3	Impresoras de matriz de puntos	264
9.4	Impresoras de transferencia directa	266
9.5	Impresoras láser	267
9.5.1	Operación	267
9.5.2	Impresoras láser en color	269
9.5.3	Lenguajes de descripción de páginas	270

9.6	Impresoras de inyección de tinta	272
9.7	Impresoras de burbujas	274
9.8	El color en las impresoras de inyección y de burbujas	274
9.9	Trazadores	276
9.10	Otros tipos de impresoras	277
9.10.1	Impresoras de tinta sólida	277
9.10.2	Impresoras de sublimación de tinte (<i>Dye-sublimation</i>)	277
9.10.3	impresoras de color térmicas (<i>Thermo Autocrome</i>)	278
9.10.4	Impresoras de cera térmicas (<i>Thermal Wax</i>)	278
9.11	Conexión con las impresoras	278
9.12	Conclusiones	279
9.13	Cuestiones	280
9.14	Bibliografía	281

CAPÍTULO 10. TERMINALES DE VÍDEO 283

10.1	introducción: Transductores ópticos	283
10.2	Pantallas de rayos catódicos	285
10.2.1	Señal de vídeo PAL	289
10.3	Terminales de video <i>raster</i>	290
10.4	Controladores de pantalla	291
10.4.1	Controladores de pantalla alfanumérica	291
10.4.2	Controladores de pantalla gráfica	297
10.5	Otras pantallas	299
10.5.1	Pantallas de cristal líquido	300
10.5.2	Pantallas de transistores de película fina	301
10.5.3	Pantallas de plasma	301
10.6	Cuestiones	302
10.7	Bibliografía	304

CAPÍTULO 11. ALMACENAMIENTO MASIVO 305

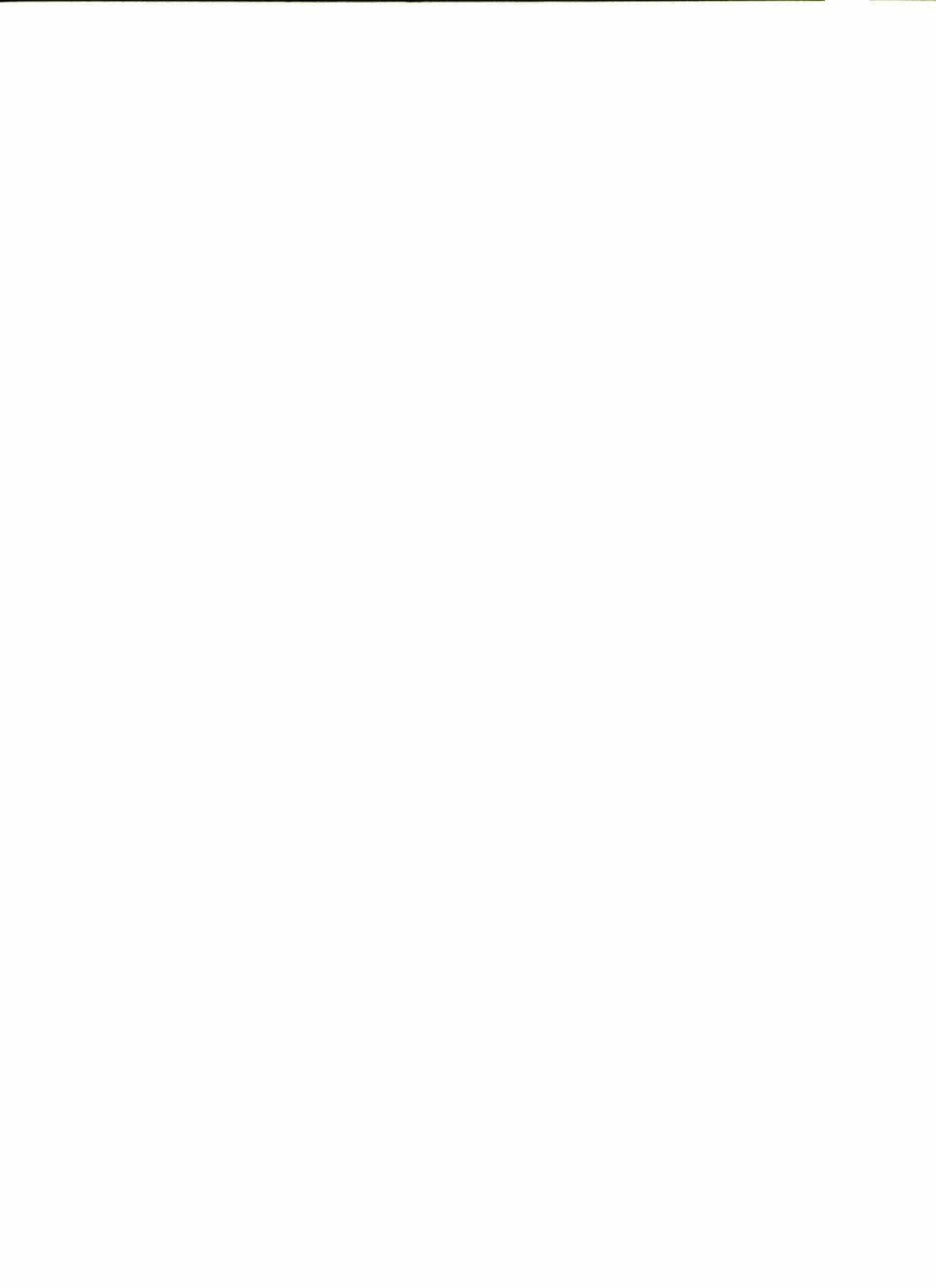
11.1	introducción	305
11.2	Grabación magnética	306
11.3	Códigos de grabación magnética	309
11.4	Discos magnéticos	313
11.4.1	Discos flexibles	315
11.4.2	Discos duros	317

11.4.3 Organización de los discos magnéticos	318
11.5 Cintas magnéticas	322
11.6 Discoóptico	327
11.6.1 CD-ROM (<i>Compact Disc Read Only Memory</i>)	328
11.6.2 DVD (<i>Digital Versatile Disc</i>)	331
11.7 Disco magneto-óptico	333
11.8 Conclusiones	334
11.9 Cuestiones	335
11.10 Bibliografía	335
CAPÍTULO 12. ARQUITECTURAS AVANZADAS	337
12.1 Introducción	337
12.2 Segmentación	339
12.3 Clasificación de las arquitecturas avanzadas	343
12.3.1 Clasificación de Flynn	343
12.3.2 Clasificación de Zargham	344
12.4 Multiprocesadores y Multicomputadores	349
12.4.1 Redes de interconexión	350
12.4.2 Multiprocesadores	353
12.4.3 Multicomputadores	356
12.5 El rendimiento	356
12.5.1 Sistemas con un solo procesador	357
12.5.2 Máquinas paralelas	359
12.6 Cuestiones	361
12.7 Bibliografía	362
ÍNDICE ALFABÉTICO	365



ÍNDICE DE TABLAS

2.1	<i>Ventajas y desventajas de los modos de direccionamiento</i>	45
2.2	<i>Evolución de los procesadores de Intel</i>	85
3.1	<i>Lista de indicadores de la ALU</i>	99
3.2	<i>Ejemplo de multiplicación secuencial</i>	115
4.1	<i>Tiempos de acceso y precio por Mbyte (1999)</i>	134
4.2	<i>Parámetros que describen una cache</i>	137
5.1	<i>Tabla comparativa de dispositivos de Entrada/Salida</i>	168
6.1	<i>Tamaño y posición de los datos en el bus VME</i>	211
7.1	<i>Señales del estándar RS-232C</i>	219
7.2	<i>Protocolo full-duplex</i>	221
7.3	<i>Señales que componen el estándar Centronics</i>	227
7.4	<i>Modos de transferencia de datos del Estándar IEEE-1284</i>	229
7.5	<i>Tipos de conectores del estándar ZEEE-1284</i>	230
10.1	<i>Tabla comparativa de las pantallas gráficas estándar</i>	298
11.1	<i>Código GCR</i>	312
11.2	<i>Código RLL_{2,7}</i>	312
11.3	<i>Clasificación de los discos por densidades</i>	316



ÍNDICE DE FIGURAS

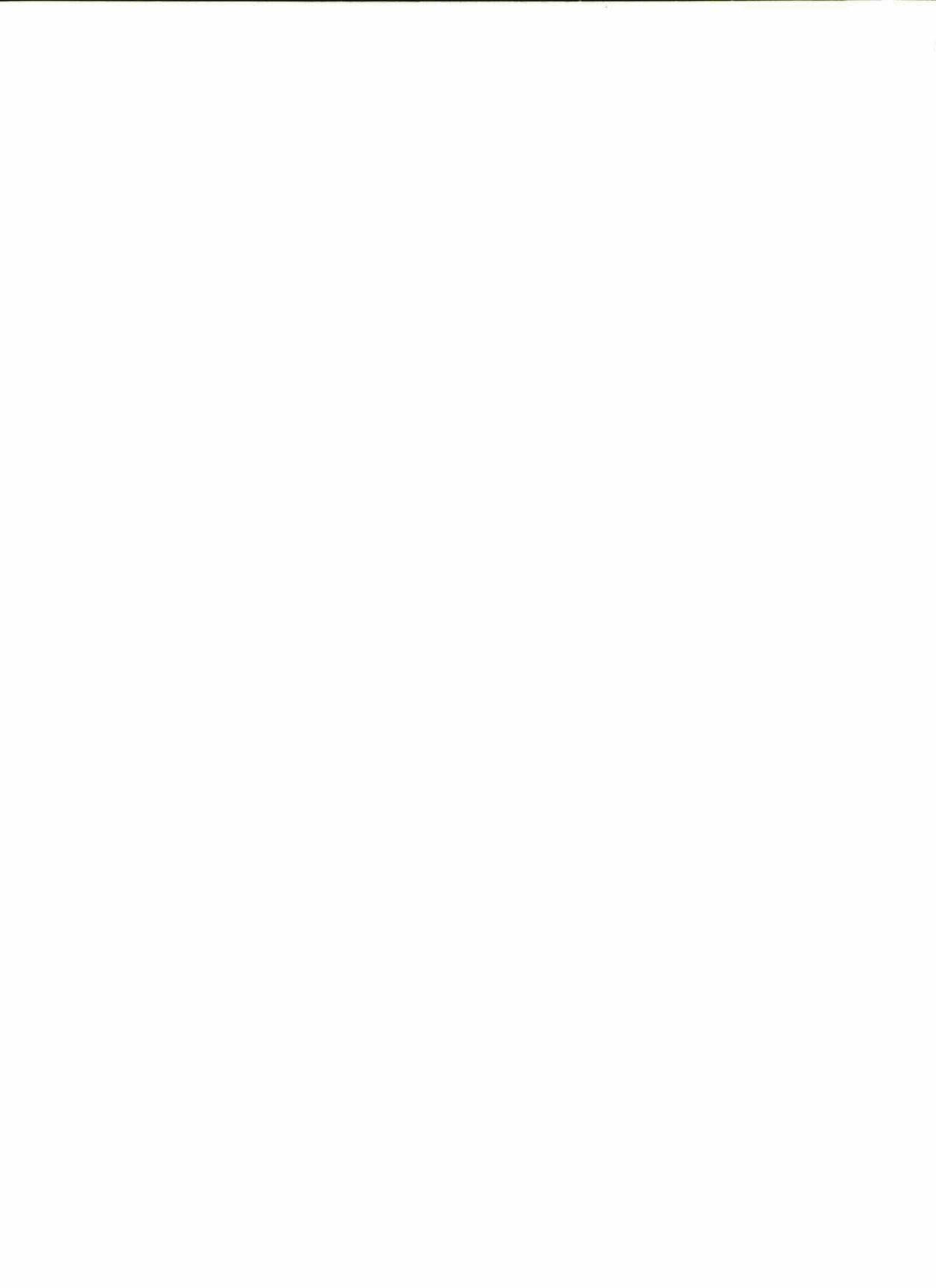
1.1	<i>Estructura básica de un computador</i>	5
1.2	<i>Estructura de la CPU y su conexión con la memoria</i>	7
1.3	<i>Ciclos de la máquina Von Neumann</i>	9
1.4	<i>Esquema</i> de una memoria de acceso aleatorio	11
1.5	<i>Esquema de una interfaz de Entrada/Salida</i>	13
1.6	<i>Señales en el bus de un computador</i>	14
1.7	<i>Código máquina y lenguaje ensamblador</i> equivalente a la <i>instrucción printf</i>	18
1.8	<i>Búsqueda de la instrucción LXI D</i>	19
1.9	<i>Ejecución de la instrucción LXI D (carga en HL de la dirección TABLA)</i>	21
2.1	<i>Esquema básico de la arquitectura Von Neumann</i>	32
2.2	<i>Direccionamiento inmediato</i>	39
2.3	<i>Direccionamiento directo</i>	40
2.4	<i>Direccionamiento mediante registro</i>	41
2.5	<i>Direccionamiento relativo a registro</i>	42
2.6	<i>Direccionamiento indirecto</i>	44
2.7	<i>Subcampos</i> en el campo de dirección	48
2.8	Campo de máscara de <i>condición</i> en las instrucciones <i>de salto</i>	49
2.9	<i>Estructura de un computador elemental Von Neumann</i>	51
2.10	<i>Transferencia elemental</i> mediante un bus	57
2.11	<i>Formato de la instrucción ADD #456</i>	62
2.12	<i>Cronograma</i> de la instrucción ADD #456	64

2.13	<i>Cronograma de la instrucción SW #456</i>	67
2.14	<i>Cronograma de la instrucción JMZ #456</i>	68
2.15	<i>Llamada a subrutina</i>	71
2.16	<i>Esquema de computador elemental</i>	72
2.17	<i>Formato del repertorio de instrucciones</i>	73
2.18	<i>UC cableada</i>	76
2.19	<i>Microprograma de la instrucción ADD #456</i>	78
2.20	<i>Agrupación en campos de las señales de control del computador elemental</i>	80
2.21	<i>UC microprogramada</i>	82
3.1	<i>Sumador elemental de 1 bit</i>	97
3.2	A) <i>Sumador combinacional paralelo</i> B) <i>Sumador secuencial serie</i>	98
3.3	<i>Esquema de una ALU genérica</i>	100
3.4	<i>Registro universal de 8 bits</i>	101
3.5	<i>Desplazamiento lógico</i>	102
3.6	<i>Desplazamiento circular</i>	102
3.7	<i>Desplazamiento aritmético</i>	103
3.8	<i>Operadores lógicos</i>	104
3.9	<i>Inversión de signo en diversos formatos</i>	105
3.10	<i>Extensión de signo en diversos formatos</i>	105
3.11	<i>Símbolo del sumador elemental</i>	106
3.12	<i>Circuito sumador restador</i>	109
3.13	<i>Ejemplo de suma en BCD</i>	111
3.14	<i>Multiplicación clásica de lápiz y papel</i>	112
3.15	<i>Matriz de multiplicación combinacional</i>	113
3.16	<i>Multiplicador secuencial</i>	114
3.17	<i>Términos producto en una multiplicación gendrica de 4x4 bits</i>	116
3.18	<i>Multiplicador combinacional básico y el multiplicador combinacional mejorado</i>	116
3.19	<i>Redistribución de los productos parciales para la interconexión con los drboles de Wallace</i>	117
3.20	<i>Reducción mediante un árbol de Wallace para $n = 4$</i>	117
3.21	<i>Ejemplo del algoritmo de Booth</i>	119
3.22	<i>Ejemplo de división con restauración</i>	121
4.1	<i>Ejemplo de mapeado directo en una cache</i>	137
4.2	<i>Funcionamiento de una cache con mapeado diwcto</i>	140

4.3	<i>Direccionamiento de una línea de cache genérica</i>	141
4.4	<i>Evolución de la tasa de fallos en función del tamaño del bloque</i>	145
4.5	<i>Ejemplos de diseño de sistemas de memoria</i>	147
4.6	<i>Mecanismo de traducción de direcciones en la memoria virtual</i>	150
4.7	<i>Estructura del TLB</i>	153
4.8	<i>Direccionamiento de páginas en la memoria virtual</i>	155
4.9	<i>Traducción de una dirección virtual con segmentación paginada</i>	157
5.1	<i>Estructura genérica de una interfaz de E/S</i>	171
5.2	<i>Estructura e interconexión del controlador de DMA</i>	177
6.1	<i>Esquema de la jerarquía de buses</i>	192
6.2	<i>Transferencia sincrona</i>	193
6.3	<i>Transferencia asíncrona</i>	194
6.4	<i>Transferencia en bloque</i>	1%
6.5	<i>Transferencia read-modify-write</i>	197
6.6	<i>Arbitraje daisy-chain</i>	198
6.7	<i>Mediante líneas individuales</i>	200
6.8	<i>Exploración secuencial</i>	200
6.9	<i>Daisy-Chain</i>	201
6.10	<i>Esquema de aplicación del bus PCI</i>	204
6.11	<i>Transferencia de lectura en el bus PCI</i>	208
6.12	<i>Transferencia de escritura simple en VME</i>	212
7.1	<i>Topología de una conexión USB</i>	223
7.2	<i>Formato del tipo de cable utilizado</i>	224
7.3	<i>Descripción de la interfaz SCSI</i>	232
7.4	<i>Fases en una transferencia SCSI</i>	233
8.1	<i>Diferentes tipos de interruptores</i>	240
8.2	<i>El problema de los rebotes</i>	241
8.3	<i>Circuito con una red RC para evitar rebotes</i>	242
8.4	<i>Circuito con un biestable RS que evita los rebotes</i>	242
8.5	<i>Lectura de la primera fila</i>	244
8.6	<i>Camino alternativo que sigue la corriente</i>	245
8.7	<i>Esquema de la conexión de un teclado</i>	246
8.8	<i>Esquema de un ratón optomecánico</i>	249
8.9	<i>Tableta digitalizadora con detección por ultrasonidos</i>	253

9.1	<i>Impresora de margarita</i>	262
9.2	<i>Detalle de una aguja del cabezal de una impresora matricial</i>	265
9.3	<i>Impresora de transferencia directa</i>	266
9.4	<i>Mecanismo de impresión de una impresora láser</i>	269
9.5	<i>Impresora de inyección de tinta</i>	273
10.1	<i>Display de 7 segmentos</i>	284
10.2	<i>Esquema de una pantalla de rayos catódicos o CRT</i>	285
10.3	<i>Varios tipos de agrupaciones del fósforo RGB en los monitores de color</i>	287
10.4	<i>Evolución de la señal de video PAL</i>	290
10.5	<i>Ejemplo de direccionamiento en una pantalla alfanumérica</i>	292
10.6	<i>Patrón de 12 filas x 9 columnas para la letra A</i>	294
10.7	<i>Estructura de contadores alfanuméricos de un controlador de pantalla alfanumérica</i>	2%
10.8	<i>Acceso a la paleta de colores en una pantalla gráfica</i>	299
10.9	<i>Estructura de una pantalla de cristal líquido</i>	301
11.1	<i>Lectura magnética</i>	308
11.2	<i>Códigos de grabación magnética</i>	313
11.3	<i>Estructura de un disco</i>	314
11.4	<i>División del disco</i>	315
11.5	<i>Discos flexibles de 5.25 y 3.5 pulgadas</i>	316
11.6	<i>Formatos con interleaving y zonning</i>	319
11.7	<i>Formato IBM 3740</i>	320
11.8	<i>Cinta con carretes</i>	324
11.9	<i>Formato de bloque</i>	324
11.10	<i>Cartucho de cinta</i>	325
11.11	<i>Cabezal de lectura</i>	326
11.12	<i>Cinta DAT</i>	327
11.13	<i>Esquema de funcionamiento del disco óptico</i>	328
11.14	<i>Escritura de un dispositivo magneto-óptico</i>	334
12.1	(A) <i>Esquema de un procesador secuencial (B) Evolución temporal</i>	340
12.2	(A) <i>Cauce segmentado de instrucciones (B) Evolución temporal</i>	340
12.3	<i>Clasificación de Flynn</i>	344
12.4	<i>Esquema de un Multiprocesador y un Multicomputador</i>	345
12.5	<i>Esquema de una máquina de flujo de datos</i>	346

12.6 <i>Esquema de un procesador matricial</i>	347
12.7 <i>Esquema de un procesador vectorial</i>	347
12.8 <i>Matriz sistólica bidimensional</i>	348
12.9 <i>Diagrama de una red neuronal</i>	349
12.10 <i>Topologías de red estáticas de 1 y 2 enlaces por nodo</i>	351
12.11 <i>Topologías de red estáticas de 3 y más enlaces por nodo</i>	352
12.12 <i>Topologías de red dinámicas</i>	353
12.13 <i>Multiprocesador a) débilmente acoplado y b) fuertemente aco- plado</i>	355



PRÓLOGO

El presente texto nace de la experiencia docente de los autores en el área de arquitectura y tecnología de computadores, de los estudios de Ingeniería Informática de la Universitat de València. El libro tiene como principal objetivo la descripción de la estructura del computador y su interacción con los periféricos más comunes. El contenido del libro coincide con las materias básicas de un curso de estructura de computadores.

En el presente texto se pretende realizar una descripción global de un sistema informático teniendo en cuenta tanto temas más tradicionales, por ejemplo la CPU, la memoria o los buses, como temas menos tratados, por ejemplo los periféricos de entrada de datos o las tarjetas controladoras de video. La intención de los autores ha sido describir los computadores y periféricos haciendo hincapié en la interacción entre éstos, así como de qué manera esta interacción afecta a las prestaciones que percibe el usuario.

En muchas ocasiones cuando se desea realizar un análisis comparativo de computadores, este análisis se queda en una mera comparación de las prestaciones de la unidad central de proceso o CPU. Un estudio más preciso tiene en cuenta, además, la jerarquía de memoria, la entrada-salida y la jerarquía de buses incluidos en el sistema. Sin embargo en pocos casos se realiza un análisis global del computador como un sistema informático en el que, tan importante como las unidades funcionales de la CPU o la jerarquía de memoria, es la interacción con los dispositivos periféricos. Para realizar este análisis es importante tener un conocimiento tecnológico de los periféricos y su interacción con el computador.

La organización del libro comienza con la exposición de la arquitectura del computador desde un nivel básico. En estos primeros capítulos se tratan las unidades elementales del procesador y el lenguaje máquina que éste es capaz de interpretar. A continuación se introducen los demás elementos que **forman** el computador junto con los periféricos, determinando las posibles **interacciones** entre **ellos** y su impacto sobre el rendimiento global del sistema. Para concluir se muestran otras posibles arquitecturas de computadores para que el lector se inicie en el estudio de arquitecturas **más** avanzadas y complejas. El texto se ha organizado en los siguientes 12 capítulos:

- 1. Introducción a la estructura de computadores:** En este capítulo se exponen los conceptos iniciales básicos de estructura de computadores y **periféricos**. Se describe el proceso de ejecución de una **instrucción** para introducir las unidades funcionales de un computador y la **interacción** entre éstas. **Análogamente** se realiza una breve descripción del proceso de fabricación de circuitos integrados y de la tecnología de memorias. Estos conceptos serán útiles para poder **justificar** cómo la mejor solución en términos de prestaciones no siempre es posible en términos de coste.
- 2. Unidad de control: ejecución de instrucciones.** Una vez introducidos los conceptos **básicos** se comienzan a detallar los elementos que constituyen el procesador. En particular en este capítulo se estudia la unidad de control, definiendo sus **funciones**, estructura y métodos de **diseño**, junto con el proceso de definición y ejecución del repertorio de **instrucciones** del **procesador**.
- 3. Unidad aritmético lógica:** Se presenta una introducción a la aritmética **digital**, mostrando diversas aproximaciones para realizar operaciones **binarias** de suma, resta, **multiplicación** y división. Se realiza un análisis comparativo de las diversas aproximaciones, indicando sus ventajas, desventajas y limitaciones.
- 4. Jerarquía de memoria.** En este capítulo se presenta la memoria, que **es** una de las partes elementales del computador, exponiendo cómo se divide dando lugar a la jerarquía de memoria. El objetivo de **esta** jerarquía es obtener un rendimiento alto con unos recursos de memoria limitados. De **esta forma** se comienza con la memoria **cache** exponiendo **su** estructura, **parámetros** característicos y las políticas de emplazamiento y sustitución de bloques. Para concluir se expone el mecanismo de traducción de direc-

ciones en tiempo de ejecución, las técnicas de segmentación y paginación de la memoria y la memoria virtual.

5. **Interconexión entre procesador y periféricos.** Este capítulo se centra en la interconexión entre el procesador y los elementos de entrada y salida. En él se explican los mecanismos de transferencia de datos entre el computador y los controladores de periféricos, definiendo las fases que tienen lugar en las operaciones de entrada y salida. También se explican técnicas avanzadas de acceso directo a memoria y procesadores de entrada y salida.
6. **Buses.** En el capítulo se introduce el concepto de bus como elemento de interconexión de todos los componentes del computador, definiendo sus propiedades y características. También se describe la organización interna de los buses, llamada jerarquía de buses, y cómo ésta afecta al rendimiento del sistema. Por último se muestran ejemplos de buses comerciales como son el PCI y VME.
7. **Enlace exterior.** El capítulo muestra las interfaces más comunes utilizadas para transferir información entre el computador y los elementos periféricos. Estas interfaces se dividen en dos grandes grupos dependiendo de si los datos se transfieren de forma serie o paralela. Para cada uno de estos grupos se detallan los estándares más utilizados en los computadores actuales.
8. **Dispositivos de entrada de datos.** Los dispositivos de entrada de datos permiten al computador recibir información de los usuarios. Estos periféricos no necesitan una velocidad de transferencia muy elevada, aunque sí que requieren una atención constante por parte del computador. Dentro de este grupo de dispositivos se encuentran los teclados, ratones, pantallas gráficas interactivas y scanners.
9. **Dispositivos de copia impresa.** En este capítulo se muestran los tipos de impresoras más utilizadas, exponiendo sus mecanismos de funcionamiento y características principales. Se presta especial atención a dos parámetros que definen sus prestaciones y la posibilidad de imprimir en color. También se detallan los lenguajes de descripción de páginas utilizados para codificar la información que se desea imprimir.
10. **Terminales de vídeo.** El capítulo se centra en los terminales de vídeo utilizados para mostrar información al usuario. Dentro de este tipo de

periféricos encontramos diversas tecnologías como los tubos de rayos **catódicos**, los LCD y los de plasma. Junto con los terminales se da **una** descripción de los **controladores** de pantalla utilizados como **interfaz** con el monitor.

11. **Almacenamiento masivo.** Se exponen los dispositivos de almacenamiento secundario haciendo referencia a su estructura física, la forma con la que se codifica la **información** y su estructura lógica. Los dispositivos explicados son las cintas y los discos, utilizando medios **magnéticos** u **ópticos**.
12. **Arquitecturas avanzadas.** En este último capítulo se introducen algunas arquitecturas de computadores más complejas y sofisticadas. En **primer** lugar se muestra la segmentación como primera **técnica** de paralelización de la ejecución de las instrucciones. A continuación se presentan las clasificaciones de computadores más usuales realizadas en función de su flujo de instrucciones y datos, para **concluir** mostrando las **arquitecturas multicámaras** y **multiprocesadoras** y las **técnicas** de medidas del rendimiento.

Junto a los contenidos citados en cada uno de los **capítulos** se incluye **bibliografía** y cuestiones.

Burjassot, a 27 de noviembre de 2000.

Los Autores.

CAPÍTULO 1

INTRODUCCIÓN A LA ESTRUCTURA DE COMPUTADORES

1.1 CONCEPTOS PRELIMINARES

Hay una gran variedad de sistemas diferentes susceptibles de recibir el nombre **de computador**; desde los simples **microcontroladores** que pueden gobernar los semáforos en un cruce, hasta los enormes supercomputadores encargados de realizar **cálculos** en simulaciones aeronáuticas, pasando por los sistemas de telecomunicaciones, sistemas de control y cálculo numérico, y los simples sistemas **ofimáticos**. Esta variedad se **manifiesta** tanto en la aplicación final, como en el tamaño, en el coste y en las prestaciones de estas máquinas en principio tan diferentes. A pesar de esta **gran** variedad de máquinas llamadas computadores, se aplican sistemáticamente ciertos conceptos fundamentales. En este capítulo se introducen los conceptos básicos y generales de los computadores **digitales**.

Un computador es un sistema secuencial **síncrono** complejo que procesa información. La información dentro del ordenador se trata de forma **binaria**, utilizando solamente los **dígitos** o valores lógicos '1' y '0', al contrario que el sistema métrico decimal habitual formado por **dígitos** del 0 al 9. Estos valores lógicos binarios **se** corresponden con niveles de tensión eléctrica, de manera que un '1' lógico corresponde a un nivel alto de tensión cercano a 5 voltios y un '0' lógico corresponde a un nivel bajo de tensión cercano a 0 voltios. Estos niveles eléctricos pueden ser distintos, dependiendo de la tecnología de los dispositivos **electrónicos** que forman el computador. En la **actualidad**, y con la intención de disminuir el consumo de la lógica **CMOS**, se está utilizando la lógica de baja

tensión, donde un '1' lógico viene definido por un nivel de tensión de 3.3 voltios y un '0' lógico por 0 voltios.

El procesador o núcleo **central** de un computador **está** formado por millones de transistores y componentes **electrónicos** de un tamaño microscópico. La duración de los eventos en el interior de un procesador es del orden de los **nano**-segundos, conmutando los transistores con frecuencias del orden de cientos de **MHz**.

La información binaria se introduce en el computador mediante **dispositivos periféricos** que hacen de **interfaz** entre el mundo exterior y el computador. En **definitiva**, van a *traducir* la información del mundo real a **señales** eléctricas, que serán interpretadas como unos o ceros. De la misma manera, los **periféricos** también permiten la comunicación del ordenador hacia el **exterior**, o bien hacia seres humanos u otras **máquinas**.

Un computador necesita de energía eléctrica para que se ponga en marcha toda la **circuitería** electrónica que **implementa** su funcionalidad. Cuando se interrumpe la alimentación del **computador** se **perderá** toda la información que no esté almacenada en un dispositivo de almacenamiento no-volátil.

La **única** forma de **analizar** un sistema tan complejo es basarse en una organización **jerárquica**. Un sistema jerárquico es un conjunto de sistemas **ínter**-relacionados, cada uno de los cuales se organiza a su vez en una estructura jerárquica, uno tras otro, hasta que se **alcanza** el nivel más bajo de subsistema elemental. La **naturaleza jerárquica** de los sistemas complejos es esencial tanto para su diseño como para su descripción. El **diseñador** necesita solamente tratar con un nivel del sistema a la vez. En cada nivel el sistema consta de un conjunto de componentes y la **interacción** entre ellos. El comportamiento en cada nivel depende **sólo** de una **caracterización** abstracta y simplificada del sistema que hay en el nivel inferior.

Para describir un sistema **jerárquico** se pueden utilizar dos aproximaciones: o bien se empieza por la parte más baja y se van construyendo los niveles de jerarquía basándose en el nivel inferior (modelo *bottom-up*), o bien se realiza una descripción desde el nivel más alto y se descompone el sistema en **módulos** de la jerarquía inferior (modelo *top-down*). Una posible clasificación de los niveles estructurales, ordenados de menor a mayor, sería:

1. Nivel de componente. En el nivel más inferior se trata con las leyes de la física de estado **sólido**, y la electrónica física. Los elementos de este nivel son difusiones de impurezas de tipo P y de tipo N en silicio, polisilicio cristalino y difusiones de metal que sirven para construir los transistores. Las **técnicas** de integración fotográficas (litografía) permiten integrar **millones** de polígonos de tamaños inferiores a la micra en unos pocos milímetros cuadrados. A partir de este nivel más bajo de abstracción se construyen los transistores que constituyen el siguiente nivel.
2. Nivel electrónico. En este nivel los componentes son transistores, resistencias, condensadores y **diodos** construidos con las difusiones del nivel anterior. Esta tecnología de muy alta escala de integración o **VLSI** (Very **Large** Scale of Integration) es la que se utiliza en la fabricación de circuitos integrados (**Clis**). Son de aplicación las leyes eléctricas y electrónicas y el comportamiento del sistema se describe en términos de tensión eléctrica y corriente. En este nivel se construyen las puertas lógicas a partir de transistores.
3. Nivel digital. El estado de los sistemas se describe mediante unos y ceros. Los elementos de este nivel son las puertas lógicas, biestables y otros módulos tanto combinacionales como secuenciales. En este nivel es de aplicación el álgebra Booleana y las propiedades de la lógica digital. Esta información binaria **representará** a su vez un carácter **ASCII**, un número en alguna forma de representación o una instrucción para el computador (acción).
4. Nivel RTL. El nivel de transferencia de registros **RTL** (**Register Transfer Level**) será el preferido para la descripción de los computadores en el presente texto. Elementos **típicos** en este nivel de abstracción son los registros y módulos combinacionales aritméticos. Los registros se encargarán de almacenar la información binaria y se construyen con puertas y elementos del nivel digital inferior. Los módulos combinacionales aritméticos serán los encargados de realizar transformaciones básicas de los datos.
5. Nivel **PMS**. Este nivel es el más alto en la jerarquía de descripción de computadores. Las siglas PMS provienen del inglés Processor Memory Switch. Son elementos de este nivel de jerarquía los **buses**, memorias, procesadores y otros módulos de alto nivel. La descripción del sistema se hace utilizando bloques que **intercambian** información por una serie de enlaces llamados **buses**.

En este texto se realizará una **descripción jerárquica** del computador utilizando básicamente el nivel RTL. Algunas veces será **necesario bajar** al nivel **digital**, pero en ningún caso se descenderá a los dos niveles más bajos de la jerarquía.

1.2 ARQUITECTURA CLÁSICA DE UN COMPUTADOR: MODELO VON NEUMANN

La conocida como arquitectura Von Neumann tiene sus orígenes en el trabajo del matemático John Von Neumann desarrollado conjuntamente con John Mauchly y John P. Eckert y divulgado en 1945 en la Moore School de la Universidad de Pensilvania, Estados Unidos, en el que se presentaba el EDVAC (*Electronic Discrete Variable Automatic Computer*). De este trabajo surgió la arquitectura de programa almacenado en memoria y **búsqueda/ejecución secuencial** de instrucciones.

Para empezar a describir la estructura básica de un computador siguiendo este modelo, cabe definir las funciones que se espera que pueda realizar una máquina denominada como tal. En términos generales un computador tiene que realizar tres funciones:

- Procesamiento de datos.
- Almacenamiento de datos.
- Transferencia de datos.

El computador debe **procesar datos**, transformando la información recibida. Así mismo debe **almacenar datos**, tanto resultados de **computaciones** intermedias, como el resultado final de éstas. De la misma manera el computador debe realizar **transferencia de datos** entre su entorno y él mismo. Para realizar estas tres funciones es necesario un mecanismo de **control** que gobierne el flujo de datos entre los diversos módulos.

Estas funciones se pueden realizar con arquitecturas de computadores muy avanzadas, como las que se analizan en el capítulo 12, o con la estructura de un computador básico que es la que se muestra en la figura 1.1. La arquitectura de

un computador hace referencia a la organización de sus elementos en **módulos con una funcionalidad** definida y a la **interacción** entre ellos.

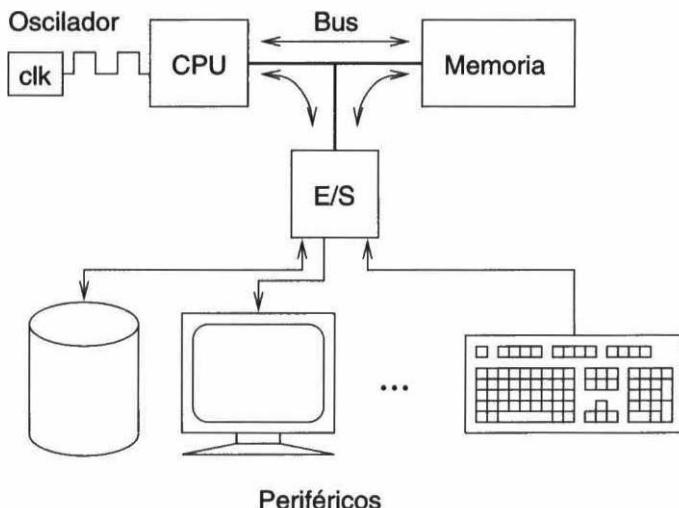


Figura 1.1: Estructura básica de un computador

- **CPU:** La Unidad **Central** de Proceso (*Control Process Unit*) es el corazón del computador. Controla el flujo de &tos, los procesa, y gobierna el **se-
cuenciamiento** & las acciones en todo el sistema. Para ello necesita un **oscilador** externo o reloj que **sincroniza** las operaciones y marca la velocidad de proceso. Este reloj no es más que una señal periódica **cuadrada** con una frecuencia constante. El reloj va marcando la evolución de la CPU y mide su velocidad de funcionamiento, por tanto siempre es deseable que las frecuencias de reloj sean lo más altas posibles. **Desafortunadamente** la frecuencia de reloj viene limitada por la tecnología de la CPU y del computador completo. Si se aumenta la frecuencia de reloj por encima de la frecuencia máxima de funcionamiento, el sistema no podrá seguir al reloj y se producirán **desfases** y **desincronizaciones**, dejando de funcionar **correctamente**. Un efecto colateral es que la **potencia disipada** por las circuitos de tecnología CMOS, (la mayoría de las **CPUs actuales**), es proporcional a la **frecuencia de funcionamiento**. Por lo tanto, una frecuencia excesiva puede provocar un sobrecalentamiento que **deteriore** la CPU.
- **Memoria:** Es la responsable del almacenamiento de datos. En un mismo computador hay muchos tipos de memoria que van desde las más masivas y baratas (cintas **magnéticas**), a las más **rápidas** y caras (memoria **SRAM**

cache). Estas diferencias provocan una distribución y organización por capas que se denomina jerarquía de **memorias**. El objetivo es disponer de la mayor cantidad de memoria posible combinando un coste reducido con una alta velocidad.

- **WS:** Transfiere datos entre el entorno exterior y el computador. Este bloque está formado por los elementos necesarios para realizar la comunicación del computador con el mundo exterior. En él se encuentran los controladores de **periféricos** que forman la **interfaz** entre los **periféricos**, la memoria y el **procesador**.
- **Sistema de interconexión: Buses:** Es el mecanismo que permite el flujo de datos entre la CPU, la memoria y los **módulos** de E/S. En los **buses** se propagan **señales eléctricas** que son interpretadas como unos y **ceros** lógicos.
- **Periféricos:** Los dispositivos **periféricos** son los que permiten la entrada de datos al computador, y la salida de información una vez procesada. Un tipo de estos **periféricos** puede entenderse como **grupo** transductores entre la información física externa y la información binaria **interpretable** por el computador. Un ejemplo son las tarjetas de **vídeo**, que conjuntamente con los monitores, transforman información **binaria** en información visual que percibe el **usuario**. En sentido **contrario** se **tendrían** los **escáneres**, que transforman la información visual en **códigos binarios** que la representan y el computador puede interpretar. **Otro** tipo de dispositivos **periféricos** sirven para almacenar datos o para que se comunique el computador con otros computadores o máquinas **digitales**. Ejemplos de dispositivos periféricos son el teclado, el monitor, el ratón, disco duro y las tarjetas de red.

En el presente texto se van a analizar los componentes de un computador clásico o máquina Von Neumann, así como en el capítulo 12 se van a introducir conceptos y tendencias en la estructura de computadores más avanzados.

1.2.1 Unidad central de proceso

La unidad central de proceso (**CPU**) controla el funcionamiento de todos los elementos del computador. El sistema, desde que es alimentado, empieza a

ejecutar instrucciones y no se detiene hasta que se corta la alimentación. Las instrucciones **forman** programas mediante los cuales se realiza una determinada acción y estos programas no son **sólo** los del usuario, sino también los de gestión del sistema completo. La CPU es la parte más importante del procesador, debido a que se encarga de controlar todas las partes del computador y de realizar los cálculos. La CPU tiene a su vez una estructura interna que se muestra en la figura 1.2.

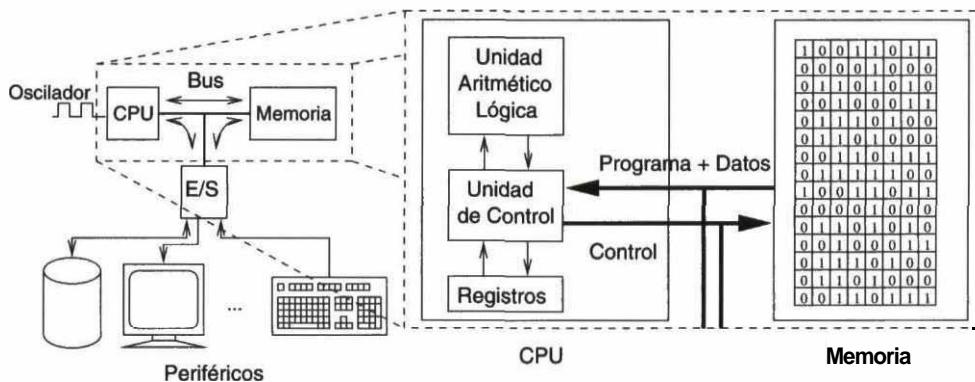


Figura 1.2: Estructura de la CPU y su conexión con la memoria

Una CPU básica consta principalmente de las unidades:

- **Unidad de Control (UC):** La unidad de control se encarga de leer de la memoria las instrucciones que debe ejecutar y de **secuenciar** el acceso a los datos y las operaciones a realizar por la unidad de proceso. Los datos y las instrucciones están en la misma memoria en el caso de la figura 1.2. La UC genera las señales de control que establecen el flujo de datos en toda el computador e interno a la CPU. Se encarga de acceder a memoria habilitando un camino de datos para que las instrucciones y los datos lleguen a la CPU.

Una instrucción no es más que una combinación de unos y ceros. Consta de un código de operación binario (que las diferencia entre sí) y **opcionalmente** de los datos necesarios para ejecutar la instrucción (o al menos indicación de dónde encontrarlos). Una vez se ha accedido a la instrucción que se va a ejecutar, la UC la almacena en un registro especial (el registro de instrucción), interpreta su código de operación y ejecuta la secuencia de acciones adecuada. Es decir, **decodifica** la instrucción. Los diversos tipos de instrucciones y sus **formatos** se estudiarán con detalle en

el capítulo 2. En consecuencia la ejecución **de una** instrucción pasa siempre por las fases de búsqueda de la instrucción, **decodificación**, búsqueda de los operandos y ejecución.

En el ejemplo de la figura 1.2 se puede **observar** un programa en memoria Una vez se ha **leído** la **instrucción** (se guarda una copia de ésta **en** el RI de la CPU), la UC se encarga de generar **las** señales de control adecuadas para ejecutarla. La UC se estudiará con detalle en el capítulo 2.

- **Unidad Aritmético-Lógica (ALU):** La ALU **es** la parte de la CPU encargada de realizar las transformaciones de los datos. Gobernada por la UC, la ALU constará de un serie de **módulos** que realizarán operaciones aritméticas y lógicas. Generalmente la ALU no es **más** que **circuitería combinacional** con unos pocos registros que almacenan los datos de entrada y los de salida o resultados. La UC se encarga & seleccionar la operación a **realizar** habilitando los caminos de datos entre los diversos operadores de la ALU y entre los registros internos. La ALU se analiza con detalle en el capítulo 3.
- **Registros internos:** La ejecución de las instrucciones puede necesitar almacenar resultados parciales. El almacenamiento de estos resultados en la memoria principal sería lento e impondría un **tráfico** excesivo de datos en el sistema de interconexión con la memoria, con lo que el rendimiento bajaría. Es preferible, por tanto, que estos resultados parciales se almacenen en pequeñas celdas de memoria incluidas en la propia CPU. Estas pequeñas celdas son los registros. De la misma manera también **se almacena** en **los** registros **internos** la **configuración** interna de la CPU o **información** sobre la última operación realizada en la ALU (signo, desbordamiento, resultado nulo, etc.). Los registros principales de una CPU genérica son:
 1. **Contador de programa:** Se encarga de almacenar la dirección de la siguiente instrucción a ejecutar. Normalmente se va incrementando para ejecutar el programa de **manera** secuencial. Cuando se modifica el orden de la **ejecución** de instrucciones (**un** salto a una subrutina o un bucle) se **debe escribir** en él el valor de la dirección de la nueva instrucción a ejecutar. Se conoce también por sus siglas en inglés **PC (Program Counter)**.
 2. **Registro de Instrucción:** En este registro se almacena la instrucción que se ha capturado desde la memoria y que se está ejecutando. Se conoce también por sus siglas en inglés **IR (Instruction Register)**.

3. **Registro de Estado:** Está compuesto por una serie de bits que informan del resultado de la última operación realizada en la ALU. Típicamente hay un bit que indica si el resultado de la última operación ha sido cero (Z), si el resultado ha sido negativo (N) o si ha habido desbordamiento o *overflow* (O). En cualquier caso los bits de este registro tienen significado por separado, y carecen de significado conjunto como palabra.
4. **Registro acumulador:** Algunas CPUs realizan operaciones aritméticas sólo en un registro especial denominado acumulador, y disponen adicionalmente de registros de propósito general. Hay máquinas más simétricas que permiten realizar todas las operaciones sobre todos los registros de propósito general, y que carecen de este registro especial.

El ciclo para ejecutar cualquier instrucción se divide en **ciclo de búsqueda** y **ciclo de instrucción**. En el ciclo de búsqueda la unidad de control genera las señales adecuadas para acceder a la memoria y leer una instrucción. El ciclo de búsqueda para cualquier instrucción siempre es el mismo. La diferencia estriba en el ciclo de ejecución, que será función del código de operación de cada instrucción. Una CPU como la mostrada en la figura 1.2 siempre está, o bien en el ciclo de búsqueda, o bien en el ciclo de ejecución, tal como se muestra en la figura 1.3. La CPU está eternamente en uno de ambos ciclo hasta que pierde la alimentación o hasta que se ejecuta alguna instrucción especial de parada (HALT).

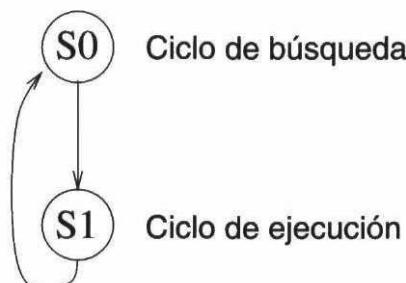


Figura 1.3: Ciclos de la máquina Von Neumann

1.22 Memoria

En la memoria se almacenarán el programa y los datos que, provenientes de un dispositivo de **Entrada/Salida** o dispositivos de almacenamiento secundario, va a **ejecutar** la CPU. **Las** instrucciones no son más que códigos binarios interpretados **por** la unidad de control, y los datos están **también** almacenados de forma binaria en **alguno** de los **formatos** de representación de la **información** binarios existentes. De esta manera el código **binario 1100001** almacenado en memoria puede representar una operación en código máquina, el carácter **ASCII** 'a', el número entero sin signo 97 o el número negativo -31 (en complemento a 2) o -33 (en signo magnitud).

Hay diversas formas y tecnologías de almacenamiento de la información binaria, desde el almacenamiento con transistores en los circuitos integrados de las **SRAM**, hasta el almacenamiento **magnético** de los discos duros de los computadores. En cualquier caso, el **proceso de acceso** a los programas y datos se realiza siempre a **través** de los distintos niveles de la memoria, siendo el más significativo la memoria principal o **main memory**.

Las diversas tecnologías de almacenamiento de la información imponen diversas características que se describen con una **serie de parámetros**. De todos ellos los más importantes son el tiempo de acceso, que sería el tiempo necesario para recuperar un dato almacenado, y el **precio** por bit. Ambas características suelen tener una relación inversa. De esta manera, dispositivos de memoria implementados con una gran velocidad de acceso suelen tener un alto coste y viceversa. Esto hace que los computadores no puedan tener toda la memoria **implementada** con el dispositivo más rápido existente cuando se diseñan. Un compromiso **prestaciones/coste** en necesario. Por lo tanto se suele realizar un **diseño jerárquico** de la **memoria** del sistema, dejando la memoria más **rápida** (y más cara) **directamente** accesible por la **CPU**, y la memoria **más** lenta en las capas de la jerarquía más externas. El principio de que sea la memoria más rápida la más cercana al procesador, combinado con el principio de localidad (que se mostrará en la sección 4.2), y con un flujo de datos eficiente entre las diversas capas, sirve para diseñar el sistema de memoria de un computador. En el capítulo 4 se analiza con detalle la jerarquía de memoria.

Otro aspecto interesante a tener en cuenta es la **volatilidad** de la información almacenada en las memorias. Es necesario que **toda** la información **imprescindible** se guarde en la memoria principal (o **RAM**) para que sea ejecutada por la CPU.

dible para que un computador **arranque** (se configure) correctamente y pueda responder al usuario no se pierda cuando se desconecte la máquina. De esta **manera**, el computador tendrá acceso a una zona de memoria no volátil en la que se almacenará toda la información necesaria para el **arranque** y la configuración. Cuando se conecta el computador automáticamente ejecutan unos programas, almacenados en memoria no volátil, que se encargan de hacer las **comprobaciones** iniciales (test), ponen en marcha los servicios de atención a los dispositivos **periféricos**, y la gestión de los recursos de la máquina, activando por último el interprete de órdenes que utiliza el usuario para ejecutar sus programas.

La memoria principal de los computadores tiene una estructura similar a la mostrada en la figura 1.4. Se puede considerar como una matriz de celdas de memoria a las que se puede acceder **aleatoriamente**, es decir, el tiempo de acceso no depende de la celda a la que se quiere acceder.

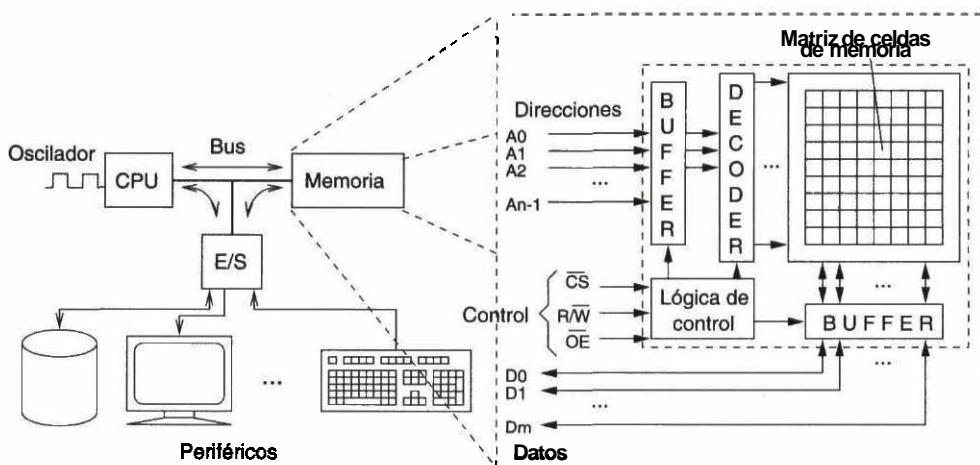


Figura 1.4: Esquema de una memoria de acceso aleatorio

La matriz de memoria está organizada en palabras, cada una de las cuales tiene asignada una dirección que indica su posición en la matriz. En el ejemplo de la figura 1.4 hay n líneas de direcciones, lo que da un tamaño de **la memoria** de 2^n direcciones posibles. Sólo con modificar la combinación binaria de las **líneas** A_0 a A_{n-1} se cambia la palabra a la que se quiere acceder. Cada palabra está **formada** por una **serie** de celdas a las que se accede en paralelo. En el caso de la figura 1.4 la palabra es de 8 celdas. En cada celda se almacena un bit de información, y estos bits son los que definen las instrucciones y los datos. La memoria de un computador se mide siempre en bytes (palabra de 8 bits), aunque se pueden disponer de módulos de memoria de otras longitudes de palabra.

En el capítulo 4 se **analizarán** la jerarquía de memoria y en el capítulo 11 se presentarán los dispositivos de almacenamiento masivo **más** comunes.

1.2.3 Entrada/Salida

Un computador tiene una gran cantidad de dispositivos de entrada y salida. Desde un teclado hasta un monitor de **vídeo**, pasando por un disco duro, todos son dispositivos que la CPU necesita para obtener datos o para almacenar o mostrar resultados.

El principal problema de estos dispositivos **periféricos** es su gran variedad. Muchos de ellos comunican a un operador humano con el computador y otros sirven de conexión con otros computadores. De manera adicional, cada dispositivo tiene una **tecnología** diferente, unas características de funcionamiento distintas, y unas necesidades de atención por la CPU distintas. Por ello, cada dispositivo no se conecta directamente con la CPU mediante un bus, sino que existe una interfaz que define cómo se van a entender el **procesador** y el controlador del periférico asociado para intercambiar datos. Los módulos de **E/S** son entidades que contienen uno o varios controles de **periféricos**.

Cuando la CPU ejecuta un programa que realiza operaciones con el mundo exterior a través de los periféricos, se dice que realiza transferencias u operaciones de **Entrada/Salida**. El objetivo de cada transferencia es llevar los datos desde el periférico a una zona concreta de la memoria y viceversa. Para ello la CPU programará las acciones a realizar en los controladores de **WS**. Posteriormente, cuando el control tenga listos los datos que lee del periférico, la CPU se podrá encargar de la **sincronización** y de la **transferencia a/desde** memoria. Una interfaz de entrada salida que controla uno o más **periféricos** tiene la **estructura** típica que se muestra en la figura 1.5.

Cada control de periférico tiene una **dirección** única en el sistema. La **interfaz** de **E/S** decodifica el bus de direcciones para detectar que la **CPU** se está dirigiendo a **él**. El **direccionamiento** es similar al **direccionamiento** de memoria **principal**. El bus de datos se utiliza para el paso de datos entre el periférico y la memoria. Las líneas especiales de control sirven para coordinar y **sincronizar** la transferencia.

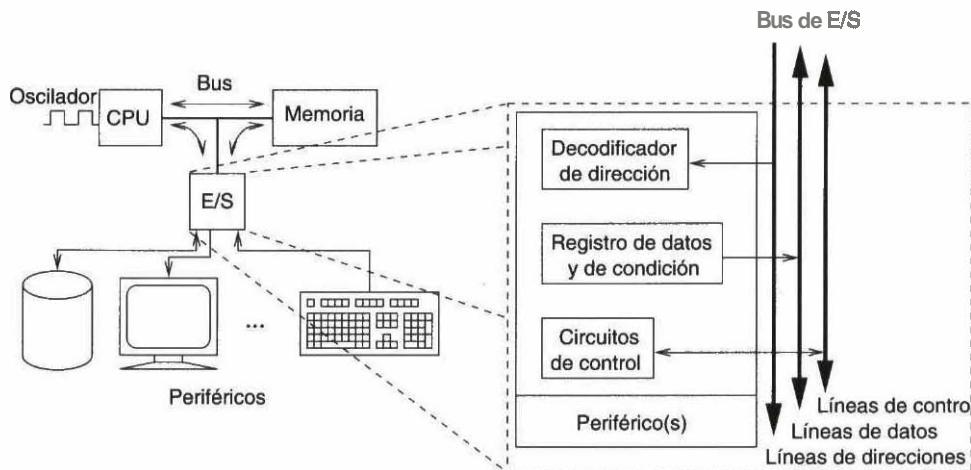


Figura 1.5: Esquema de una interfaz de Entrada/Salida

Claramente la velocidad de los periféricos, la jerarquía de memoria y el juego de instrucciones de la CPU influirán directamente en el diseño de la interfaz de **Entrada/Salida**. En el capítulo 5 se muestran las diversas estrategias para organizar el acceso de los periféricos a la memoria, así como para organizar la gestión de diversos periféricos simultáneamente por la CPU.

1.2.4 Sistemas de interconexión: Buses

La conexión entre los diversos componentes de un computador se efectúa físicamente a través de los **buses**. Un bus se define como un enlace de comunicación compartido que usa múltiples cables para conectar subsistemas. Cada **línea** es capaz de transmitir una tensión eléctrica que representa un '1' o un '0' binario. De esta manera, cuando hay varios dispositivos conectados a un mismo bus, habrá un dispositivo que podrá enviar una señal que será recogida por los demás módulos. Si varios dispositivos transmiten durante el mismo instante de tiempo, las señales se **solaparán** produciendo un error o una **contención de bus**, por lo que se tendrá que regularizar el acceso al bus.

Las señales de los **buses** están agrupadas siguiendo un criterio de **funcionalidad**, tal como se muestra en la figura 1.6 :

- **Bus de datos:** Son las líneas por las que se transmiten los datos entre los diversos dispositivos que forman el computador: CPU, memoria y controladores de Entrada/Salida.
- **Bus de direcciones:** Sirve para indicar a la memoria o al resto de dispositivos la **posición** del dato que se quiere acceder.
- **Bus de control:** Agrupa todas las señales de control que gobiernan el acceso al bus de datos y direcciones. Sirven para seleccionar el emisor y el receptor en una transacción de bus, **así** como el tipo de ésta. Agrupa señales de lectura, escritura, arbitraje, etc.
- **Bus de alimentación:** Diversos dispositivos pueden tener alimentaciones con tensiones distintas, pero todos ellos deben estar alimentados.

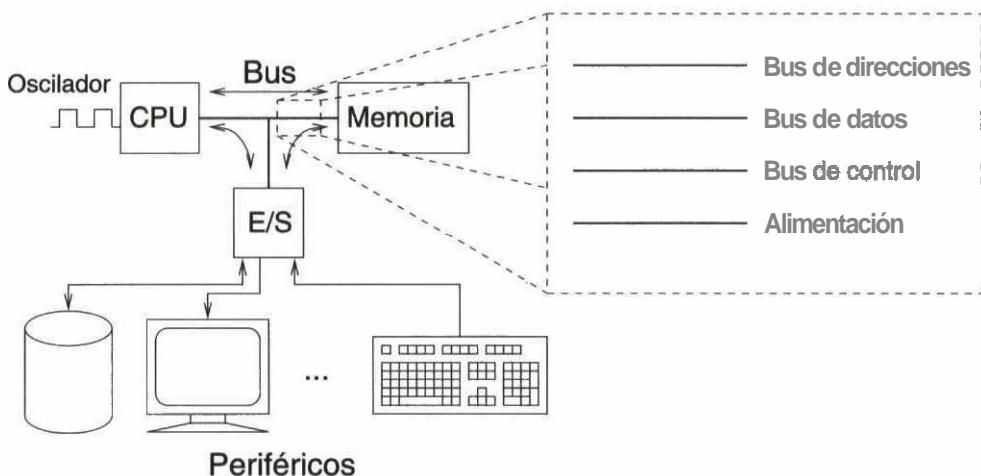


Figura 1.6: Señales en el bus de un computador

A su vez los diversos **buses** de un computador están organizados en una jerarquía que **optimiza** el acceso a los recursos y la comunicación entre los **periféricos** Y la **CPU**. En el capítulo 6 se estudiarán las generalidades de los **buses**, su jerarquía y los protocolos de **comunicación** y de arbitraje del bus.

1.2.5 Periféricos

Por periféricos se entenderán todos aquellos dispositivos que sin ser imprescindibles en la **estructura** de un computador, son necesarios para suministrar datos a la máquina o visualizar los resultados.

Como se ha descrito, el periférico se conecta mediante un bus especial (el enlace de **E/S**) a su controlador o al módulo de **E/S**. Debido a las notables diferencias de prestaciones entre los periféricos y la **CPU**, los **buses** de **E/S** son diferentes del bus que conecta al **procesador** con la memoria. En el capítulo 7 se analizarán con detalle estas características.

Los periféricos de entrada de datos son interruptores, teclados, ratones, pantallas **gráficas** interactivas y digitalizadores de imágenes (escáneres). Estos periféricos de entrada de datos se revisarán en el capítulo 8.

En la misma línea, se describen en el capítulo 9 los dispositivos periféricos de copia impresa que incluyen los diferentes tipos de impresora: de matriz de puntos, electrostáticas, de inyección de tinta y burbuja, **plotters** e impresoras láser.

Otros dispositivos periféricos fundamentales para la **interacción** del hombre con el computador son las terminales de vídeo y las tarjetas gráficas. En el capítulo 10 se muestra la tecnología de las pantallas de rayos **catódicos**, terminales de vídeo **raster**, controladores de pantalla, pantallas de cuarzo líquido y de plasma.

Finalmente, en el capítulo 11 se presentan la etapa más exterior de la jerarquía de memoria (los dispositivos de almacenamiento masivo) como periféricos. Dentro de esta categoría se incluyen las cintas magnéticas, los discos duros y flexibles, y los discos ópticos.

1.3 EJECUCIÓN DE UNA INSTRUCCIÓN

Cualquier sistema **informático** está realizado en base a unas especificaciones. Los usuarios son los que se benefician del uso del computador. Las tareas

que el usuario desea realizar se van efectuando de manera progresiva mientras se van ejecutando las **instrucciones** que forman un programa. No es sencillo programar un computador sin **ningún** tipo de ayuda externa, ya que, además de realizar el programa de usuario, se **debería** programar la **gestión** de los **recursos** del computador. Afortunadamente los sistemas **informáticos** disponen de unos programas que se van a encargar de gestionar los **recursos** de la máquina y atender al usuario, facilitando la ejecución los programas de aplicación.

1.3.1 El sistema operativo

Hace unos años un **sistema** operativo podría haberse definido como el *software* que controla al hardware. Actualmente esta definición ya no es totalmente cierta, ya que estas funciones de control están empezando a ser realizadas por el microcódigo o **firmware**, tal como se introducirá en el capítulo 2.

Un sistema operativo puede verse como los programas, instalados en software o **firmware**, que hace utilizable el hardware. El hardware proporciona la capacidad de cómputo y los sistemas operativos ponen dicha capacidad al alcance del usuario, administrando el hardware para lograr un buen rendimiento. Los sistemas **operativos** son ante todo administradores de recursos, siendo el hardware del computador el principal recurso que administran: el procesador, la memoria, los dispositivos de **Entrada/Salida** y todos los **periféricos**.

Los sistemas operativos realizan muchas funciones, como proporcionar una **interfaz** de usuario, permitir que los usuarios compartan entre sí el hardware y datos, evitar que haya conflictos de accesos simultáneos a un mismo recurso y localizar errores, advirtiendo de ellos a los usuarios. En cualquier caso la función del sistema operativo es vital para el correcto funcionamiento de un computador.

1.3.2 Lenguajes de alto nivel, ensamblador y código máquina

Es interesante, antes de analizar con detalle las partes de un computador, observar el proceso de ejecución de una instrucción. En primer lugar cabe aclarar el concepto de instrucción. Una instrucción de un lenguaje de alto nivel, como

pueda ser el lenguaje C o **Pascal**, es una construcción cercana al lenguaje humano pero alejada del lenguaje **binario** que entienden **las** máquinas. Un ejemplo es la siguiente instrucción de **código C**:

```
printf("Esta instrucción muestra este mensaje por la  
salida estándar");
```

La **instrucción** anterior es bastante autoexplicativa y simple, pero desencadena una secuencia de muchas acciones que a *priori* pueden aparecer ocultas.

En primer lugar la instrucción en C **printf** no se **corresponde** con una instrucción en **código máquina**, que es el único lenguaje que interpreta el computador. Para que el **computador** pueda interpretar esa instrucción debe ser traducida a código máquina. Para ello **se** realiza la **compilación**, en la cual la instrucción se traduce a la secuencia de unos y ceros que entiende el computador. Previamente a la obtención del programa de código **máquina** o ejecutable **se** tiene que realizar un **pre-proceso** o enlazado en el que se asignan direcciones a las etiquetas y se realizan otras tareas.

Las instrucciones en código **máquina** que **implementan** la instrucción de alto nivel se almacenan junto con los datos en memoria principal. En este caso la instrucción **printf** tiene wmo datos la frase que muestra en pantalla que se almacenará en una **zona** especial de memoria dedicada a los datos.

Las instrucciones de **código** máquina están codificadas como una serie de unos y ceros. En la figura 1.7 se muestra la secuencia de instrucciones máquina que corresponderían a la instrucción **printf**. Cada instrucción (código de operación + operandos) ocupa 8, 16 o 24 bits, siendo la longitud de palabra de la **memoria** de 8 bits. **Los** códigos **binarios** que codifican las instrucciones corresponden a la CPU de **INTEL** 8085, y han sido fijados con el **único** propósito de ilustrar esta sección. En el capítulo 2 se muestra el formato de las instrucciones máquina, así wmo los diferentes tipos de instrucciones.

Escribir un programa en código máquina es una tarea ardua y nada **sencilla**. Se requiere un conocimiento completo de la arquitectura del computador para el que se está desarrollando el programa. Es **más** sencillo escribir un programa en un lenguaje de alto nivel **más** o menos **estándar** y cuyo juego de instrucciones apenas cambia de una máquina a otra. **Posteriormente**, un programa llamado compilador, específico para cada máquina, se encargará de traducir las instrucciones de alto nivel a instrucciones máquina.

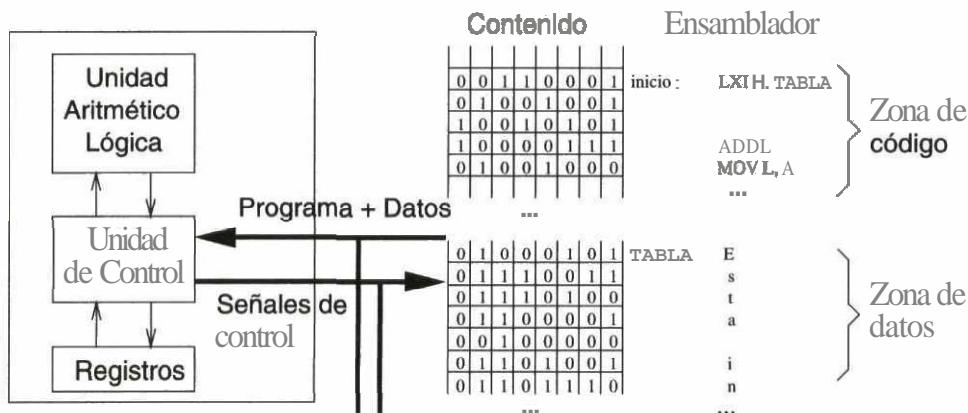


Figura 1.7: *Código máquina y lenguaje ensamblador equivalente a la instrucción `printf`*

Se puede hacer **más** agradable la escritura en **código** máquina asignándole un nemónico a cada instrucción y utilizando etiquetas para designar direcciones de memoria. Esto es lo que se conoce como código ensamblador. Este código tiene una relación directa y **biunívoca** con el **código** máquina, **pero** la utilización de **nemónicos** y etiquetas lo hace más legible que su equivalente máquina. La figura 1.7 ilustra el código máquina equivalente de la instrucción `printf` que está **ejemplificando** esta instrucción. Se incluye al lado de cada instrucción **máquina** (sólo están las primeras) el **nemónico** equivalente en ensamblador que lo hace mucho **más** legible. En la zona de **código** se incluyen las primeras instrucciones y en la **zona** de datos se almacena el código **ASCII** equivalente del **mensaje** "Esta in..."

1.33 Flujo de datos

El proceso de ejecución de la **instrucción print** consistirá en ejecutar cada una de las **instrucciones** del código máquina en las que ha sido traducida. Para ello, una vez que se ha terminado de ejecutar la última instrucción **máquina** anterior, **se** inicia el ciclo de **búsqueda** de la nueva instrucción. El ciclo de búsqueda consiste en realizar una copia de la **instrucción** a ejecutar en un registro interno de la CPU denominado **registro de instrucción**. Esto se consigue gracias a que la CPU almacena la dirección de la siguiente **instrucción** a ejecutar en el **registro contador de programa**. La CPU fija esta dirección en el bus de direcciones y genera las señales del ciclo de lectura. Una vez transcurrido el tiempo de acceso a **memoria** el **dato** contenido en esa dirección estará en el bus de datos y se captura en el **registro de instrucción**.

Cuando se **tiene** almacenada la wpi de la instrucción se inicia la secuencia de acciones específicas para cada instrucción. Para ello se **decodifica** la **instrucción** y la unidad de control habilita los caminos de datos para que **se** produzca el flujo de datos, **capturando** los operandos necesarios para **realizar** la operación determinada en la instrucción. De hecho estos datos fuente para la operación pueden estar situados en lugares diversos del computador: en la misma **instrucción**, en registros internos de la máquina o en memoria principal. La manera en que **se** establece el lugar de almacenamiento de los operandos fuente o resultado constituyen los **modos de direccionamiento**.

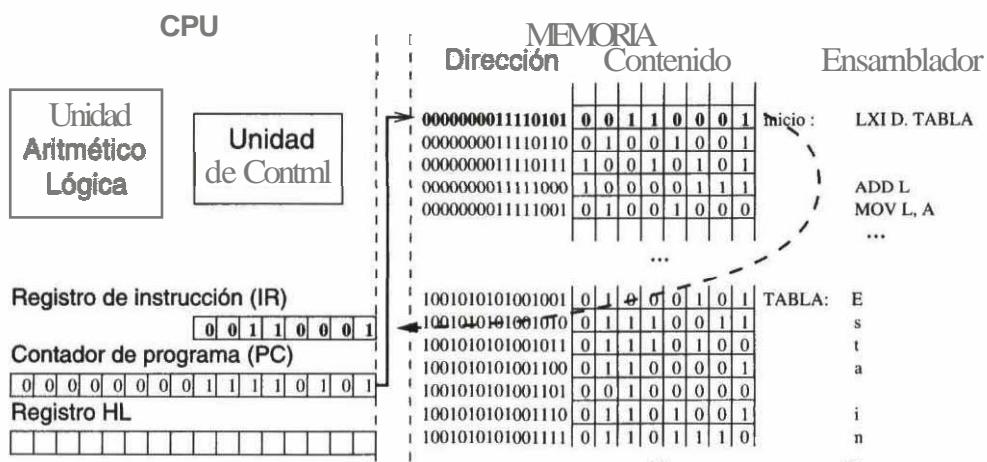


Figura 1.8: Búsqueda de la instrucción LXI D

Como ejemplo se va a **considerar** el caso del flujo de datos del código que se ha mostrado en la figura 1.7. La instrucción en **código máquina** a ejecutar será la instrucción contenida en la dirección a la que apunta el contador de programa de 16 bits (**PC=0000000011110101**). Esta **instrucción** es **00110001** (**LXI D**) que carga en el registro D el byte contenido en la dirección apuntada por el registro **HL**. Al **decodificarse** el código de operación de este primer byte la unidad de control constata que la instrucción está formada por 2 bytes más que son la dirección absoluta que se debe cargar y se accederá a ellos. Esta dirección almacenada en 2 bytes está **descrita** en **ensamblador** como la etiqueta TABLA. De esta **manera**, utilizando **nemónicos** y etiquetas se hace más cómoda la programación a bajo nivel del computador. Realizar esta programación en **código máquina** puede llegar a ser inabordable, siendo los compiladores los que se van a encargar de realizar la transformación de lenguaje de alto nivel a código máquina.

En la figura 1.8 se muestra cómo se accede a la dirección de **memoria principal** apuntada por el PC iniciando el ciclo de búsqueda. Dicho dato (1^{er} byte de la instrucción) se carga en el registro de instrucción, finalizando el ciclo de búsqueda e **iniciándose** el ciclo de ejecución de la instrucción con la **decodificación** de la instrucción por la unidad de control. Este ciclo de ejecución fuerza a su vez que se carguen los siguientes bytes (que son la dirección de la tabla) en el registro **HL**. Esto se muestra en la figura 1.9. Se supone que en esta máquina el registro funcionará como puntero a tabla utilizando el **direcccionamiento relativo** a registro base que se expondrá en el capítulo 2.

1.4 TECNOLOGÍA DE COMPUTADORES

La **afirmación** de que siempre es deseable resolver una tarea lo más rápidamente posible se traduce en **términos informáticos** en que siempre es deseable utilizar los circuitos digitales más rápidos. Esto no es siempre posible, ya que habitualmente las mayores prestaciones en términos de velocidad van ligadas a un mayor coste económico. Muchas veces la mayor velocidad en la ejecución de una tarea sistemática sobre una serie de bits (**por ejemplo**, una inversión lógica) viene delimitada por el tipo de circuito **digital** que se utilice. Realizar la operación **simultáneamente** sobre todos los bits es más **rápido** que realizar la operación **secuencialmente** sobre cada bit. En el primer caso se realiza la **opera-**

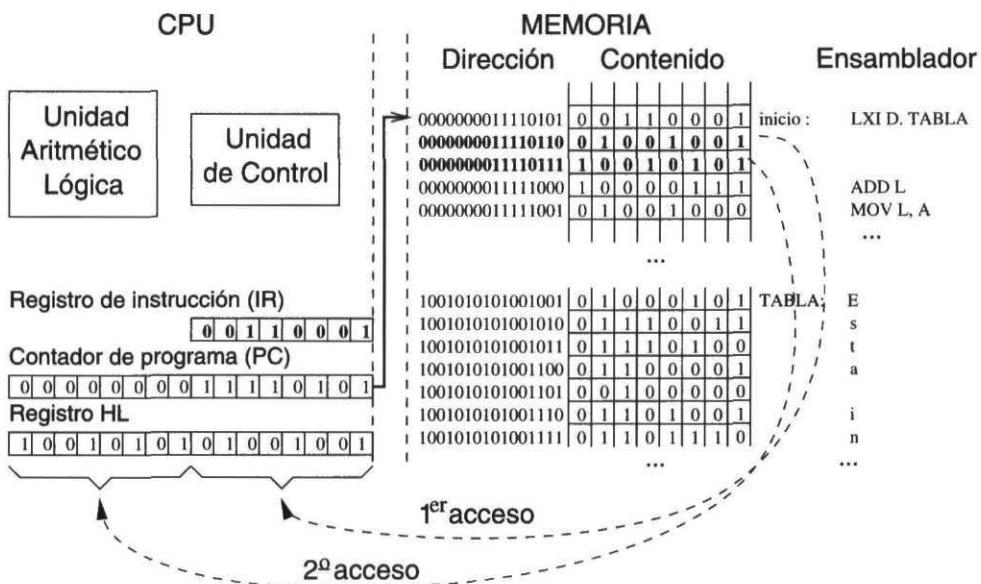


Figura 1.9: Ejecución de la instrucción LXI D (carga en HL de la dirección TABLA)

ción de forma paralela, y es necesaria una circuitería más compleja que replique la funcionalidad en todos los bits. En el segundo caso es necesaria una circuitería más sencilla, y se trata, por el contrario, de una solución más lenta. Una circuitería más compleja implica un mayor coste económico, ya que el aumento de la complejidad se traduce en un mayor tamaño de los circuitos. Para poder entender por qué el coste aumenta casi de forma exponencial con el tamaño es interesante realizar una rápida revisión de los fundamentos de la fabricación de circuitos integrados (CIs).

1.4.1 Tecnologías de circuitos integrados

Los ladrillos con los que se edifica el computador son los CIs. Estos pequeños componentes con pines metálicos y cuerpo plástico o cerámico implementan los bloques básicos que transforman la información en el computador. De hecho la CPU suele ser el CI de mayor tamaño y con mayor cantidad de pines de la tarjeta base de un ordenador. Las propiedades básicas que caracterizan un CI son:

- **Tecnología:** El tipo de transistores que utiliza el CI para **implementar** la **circuitería digital** definen la tecnología del CI. De esta manera los **CIs** basados en transistores **bipolares** o **BJT (Bipolar Junction Transistor)** originaron diversas familias tecnológicas, entre ellas la popular **TTL (Transistor Transistor Logic)**. Esta tecnología *ha* tenido como ventajas su facilidad para suministrar corriente y su rapidez, **apareciendo como** desventajas su alto consumo y su poca capacidad de integración en comparación con la tecnología **CMOS (Complementary Metal Oxide Semiconductor)**. Esta segunda tecnología se basa en la utilización de transistores de efecto campo, y es la elegida en la actualidad para **fabricar** la mayoría de **CPUs**. La gran **sencillez** de esta lógica y su *gran* capacidad de integración ha propiciado su utilización de forma masiva en los computadores actuales. Como desventaja básica se tiene que su consumo es lineal con la frecuencia y cuadrático con la tensión de alimentación, lo que provoca una **gran** disipación de potencia en los procesadores actuales que funcionan con una frecuencia de reloj de varios cientos de **MHz**. Esta **limitación** ha propiciado la utilización de la tecnología **CMOS** con menor tensión de alimentación (3.3 y 3 voltios), redefiniéndose los niveles lógicos, ya que el '1' lógico coincide con la tensión **más** alta del circuito. Otra tecnología como la **BiCMOS** combina en un solo proceso tecnológico transistores **BJT** y **CMOS**, intentando **combinar** las ventajas de ambos tipos de tecnología.
- **Velocidad:** La velocidad de los **CIs** **hace** referencia al tiempo de respuesta y los **retrasos** inevitables que aparecen en su funcionamiento. Esto en el caso de los **CIs más** sencillos depende directamente de la tecnología utilizada, pero en el caso de **CIs** más complejos (en el caso extremo los procesadores) depende de la **arquitectura** y organización de los módulos que lo forman. De esta **manera**, la frecuencia de funcionamiento no es una medida válida para procesadores donde hay que tener en cuenta factores como los ciclos por instrucción o el repertorio de instrucciones, tal como se justificó en la sección 12.5. En cualquier caso, la ejecución de una tarea se acelerará si se realiza en paralelo sobre la mayor cantidad de información posible respecto a la misma tarea realizada de forma **secuencial**. El problema estribará en que la **ejecución** paralela requerirá más circuitos que realizarán la misma tarea, lo que provocará que el tamaño del circuito sea mayor.
- **Escala de Integración:** Los **CIs CMOS** se construyen a partir de un proceso **litográfico**, en el que de forma **secuencial** se van aplicando **diver-**

sas máscaras que proyectan las siluetas de los polígonos que forman los transistores sobre un material fotosensible depositado sobre una superficie circular (oblea) de silicio. El tratamiento químico de esta oblea y la aplicación ordenada de las máscaras provoca la aparición y crecimiento de los diversos tipos de difusiones que forman los transistores. El proceso de impresión de las **máscaras** sobre la **superficie** de silicio se realiza con una óptica muy compleja que reduce los segmentos **mínimos** hasta **llegar a las micras**. Este tamaño mínimo **debe** garantizar que las difusiones y **polígonos** que forman los transistores van a estar bien definidos y que no se van a **destruir** en los múltiples procesos químicos a los que se somete la oblea. Obviamente, cuanto mejor y más preciso sea el proceso **tecnológico** de **creación** de las difusiones, los **tamaños** podrán ser menores, y por tanto en una misma **superficie** de silicio se **podrá** incluir más lógica

- **Tamaño:** Si se pretenden realizar **CIs** muy complejos éstos **incluirán** una gran cantidad de componentes lógicos y por lo tanto transistores, lo que se traducirá en una **superficie** mayor de silicio. La *impresión* de un circuito en una oblea de silicio tiene unos costes que se pueden considerar como fijos. Si el circuito es muy grande cabrán pocas copias en la misma oblea, con lo que el coste de producción se amortizará entre unas pocas muestras. Sin embargo si el circuito tiene una menor complejidad cada copia ocupará poco silicio, con lo que en una oblea de **tamaño** constante se **podrán poner** muchas muestras, lo que abaratará los costes de producción por ejemplar. Adicionalmente es inevitable la existencia de **imperfecciones** en la oblea, lo cual va a generar ejemplares defectuosos. Si el CI es **pequeño** y por tanto en una oblea caben muchas muestras, la **influencia** porcentual de los ejemplares defectuosos será **pequeña**. Sin embargo si el **CI** es grande y caben pocas muestras la misma cantidad absoluta de ejemplares defectuosos será más importante **porcentualmente**, lo cual encarecerá **más** aun el precio por **CI**. Es por tanto interesante observar el fuerte crecimiento del coste de un **CI** con la superficie que ocupa (y por tanto la complejidad que tiene). De esta manera no siempre las soluciones más veloces son siempre posibles, ya que requieren de aproximaciones paralelas con **CIs** muy complejos, lo que requiere una gran **superficie** de silicio que muchas veces tiene un coste prohibitivo.

1.4.2 Circuitos de memoria

El almacenamiento de la información se realiza en dispositivos de memoria, capaces de mantener la información **binaria** que se ha escrito con la **posibilidad** de recuperarla. En el **capítulo 4** se describirá de qué manera se debe organizar la memoria de un ordenador para combinar rapidez y capacidad de almacenamiento. Los diversos dispositivos & memoria se distribuyen en una jerarquía en la que están más cerca de la CPU los dispositivos **más rápidos** y en los niveles más alejados los más lentos. Los dispositivos de memoria más externos están encargados de realizar el **almacenamiento** masivo, y **se** revisarán con detalle en el capítulo 11. En esta sección **se** van a introducir los diversos tipos de circuitos de memoria habitualmente utilizados en un computador.

Los **parámetros** más importantes que nos van a permitir medir la bondad de los diversos circuitos de memoria son:

- **Tiempo de acceso** o tiempo necesario para recuperar una información previamente almacenada en memoria. Hay que distinguir entre memorias de acceso aleatorio, donde el tiempo de acceso es el mismo para todos los datos contenidos en la memoria y no depende de la posición que ocupa, y memorias de acceso **secuencial**, donde el tiempo de acceso a una dato dependerá de la posición que ocupa en la memoria.
- **Densidad de información** impuesta por la tecnología utilizada. Las diversas tecnologías que permiten almacenar información ocupan un espacio distinto por bit. Es deseable poder almacenar gran cantidad de información en el menor espacio posible, teniendo de esta manera memorias de **una gran capacidad**.
- **Volatilidad** de la memoria o pérdida de la información si no se mantiene una **alimentación** constante en el circuito. Es importante **que** la información de **configuración** del computador o **programas iniciales** de arranque permanezcan siempre almacenados, recuperándose de forma automática al conectarse la **alimentación** e iniciarse **el** funcionamiento del computador.

RAM estática asíncrona

Una RAM estática **asíncrona** o SRAM (*Static Random Access Memory*) es una memoria volátil, de acceso aleatorio muy rápido en las que se puede almacenar y leer **información** en el sistema. Esta característica las hace ideales para su utilización como memoria principal en computadores. Lamentablemente su alto coste hacen **prohibitiva** su utilización de forma masiva como memoria **principal** directamente; **direccional** en muchos computadores, quedando relegada su utilización a registros y memoria **cache** (el nivel más cercano al procesador de la jerarquía de memoria, tal como se explicará en el capítulo 4).

La celda de **almacenamiento** SRAM contiene 4 transistores MOS que almacenan un 1 o un 0 mientras se mantenga la **alimentación del** circuito. La escritura y lectura de la memoria SRAM se hace con niveles de **tensión** generados en el mismo computador, con lo que no es necesario ningún dispositivo especial para poder escribir la memoria.

RAM estática síncrona

Esta RAM utiliza la misma tecnología que las SRAM **asíncronas**, con lo que son **volátiles** y de un rápido acceso. La diferencia estriba en que existe una **señal** de reloj que **sincroniza** el proceso de lectura y escritura. Esto puede ser **útil** cuando se van a realizar accesos a direcciones consecutivas en memoria: en primer lugar se suministra la dirección base y después, en cada flanco, la memoria proporciona el dato (lectura) o se le suministra el dato a escribir. Este es el tipo de acceso a memoria conocido como **ráfaga** o **burst**.

Las memorias **cache** externas de algunos microprocesadores son de este tipo para facilitar el acceso de datos en modo ráfaga y acelerar el proceso de acceso a bloques de memoria.

RAM dinámica

La RAM dinámica o DRAM (*Dynamic Random Access Memory*). Son memorias volátiles y de acceso aleatorio al igual que las SRAM. Los elementos de memoria son capacidades a las que se accede con un solo transistor, en vez de

celdas con **varios** transistores, con lo que se consigue una muy alta escala de integración. El inconveniente principal de esta tecnología es que las capacidades se descargan mediante la comente de pérdidas de los transistores. Esto obliga a tener que **realizar** un refresco periódico de las capacidades si se quiere evitar que se pierda la información. En los ciclos **normales** de lectura y **escritura** habrá que intercalar ciclos de refresco, de manera que será **necesaria** una lógica adicional que se encargue de **realizar** estos ciclos. De manera adicional una desventaja de este tipo de memorias es su lentitud si **se** comparan con las SRAM.

La memorias DRAM tienen una estructura en forma de matriz, estando las **direcciones multiplexadas** en filas y columnas. Para acceder a una posición de memoria de una DRAM (tanto para **leer** como para escribir) se debe en primer lugar depositar la parte alta de la dirección en el bus de direcciones (fila), validarla para que la DRAM la capture, y posteriormente hacer lo mismo con la parte baja de la dirección (columna). A pesar de que esta forma de acceso **ralentiza más** aún el acceso a las memorias DRAM, este tipo de memorias son las utilizadas como memorias principales de los **ordenadores**, ya que **se** consigue una alta densidad de información a bajo coste.

Las memorias DRAM tienen modos de acceso más rápidos en los que se suministra la parte alta de la dirección y posteriormente se suministra solamente la parte baja de posiciones de memoria consecutiva. Este modo de acceso se denomina modo **página** y acelera el acceso a las DRAM al no tener que suministrar para cada acceso la dirección completa.

Análogamente a las SRAM **síncronas** se han **desarrollado**, y están teniendo una **gran** introducción en el mercado de ordenadores personales, las **SDRAM** o (*Synchronous Dynamic Random Access Memory*). Estas memorias DRAM **síncronas**, al igual que las SRAM **síncronas**, permiten las **transacciones** en modo ráfaga.

Memorias ROM

Las memorias **ROM** (*Read Only Memory*) son memorias no volátiles de acceso **aleatorio** y de sólo lectura. Una vez han sido escritas o **programadas** sólo se puede leer el contenido de las celdas. Se suelen utilizar para almacenar el **código** que permite arrancar a los sistemas una vez **se** suministra la **alimenta-**

ción o las **rutinas** del sistema, Las ROM se **fabrican** para aplicaciones masivas con máscaras de silicio. Si la **aplicación** es un prototipo o la producción va a ser baja no es viable la inversión en mascaras de silicio para la **fabricación** de las ROM. Afortunadamente hay tres tipos de ROM que pueden ser programadas en el laboratorio, algunas de ellas incluso pueden ser borradas.

- **Memoria PROM:** Las PROM o (*Programmable Read Only Memories*) son memorias **ROM programables eléctricamente** mediante un programador especial que genera picos de tensión altos, generando corrientes que funden físicamente unos fusibles grabando el dispositivo de forma permanente. Tienen el inconveniente de que no pueden ser borradas y de que el proceso de escritura se debe realizar con una tarjeta especial.
- **Memoria EPROM:** Las EPROM o (*Erasable Programmable Read Only Memories*) se programan también con un dispositivo de **programación** especial conectado a un ordenador. La diferencia con las PROM es que las **EPROM** sí que se pueden borrar. El borrado se realiza mediante rayos **UV**. Para ello las EPROM tienen un pequeña ventana de **cuarzo** transparente a los UV mediante la cual se realiza la exposición de la matriz de celdas. Una vez están programadas deben cubrirse con una etiqueta para evitar el borrado accidental de la memoria
- **Memoria EEPROM:** Las EEPROM o (*Electrically Erasable Programmable Read Only Memories*) son memorias **programables y borrables** mediante un dispositivo especial que se conectará a un ordenador. Este **programador/borrador** de **EEPROMs** genera picos de tensión para la **programación** y borrado de las EEPROM.

Memoria FLASH

Las memorias Flash son memorias de **lectura/escritura** de acceso aleatorio no volátiles. Su comportamiento para lectura es exactamente igual al de una **SRAM**, pero para escritura es diferente, deben ser primero borradas y después **escritas**. Además la especial **configuración** interna de las Flash hace que deban ser borradas completamente o al menos el sector adecuado para escribir aunque **sólo** sea un byte. Una memoria Flash tiene internamente un registro de instrucción y una máquina de estados que genera las señales de control internas necesarias para **borrar/escribir** en un bloque o toda la memoria.

Estas memorias están desplazando a las EEPROM y EPROM como almacenamiento de programas de arranque reprogramables debido a que no necesitan alimentaciones altas para reprogramarlas, al contrario que las EEPROM. Sólo con la alimentación de +5V se puede realizar la reprogramación de estas memorias, con lo que se pueden incluir en un computador como ROM del sistema, reprogramándolas sin extraerlas.

1.5 CUESTIONES

1.1 *Representar mediante un diagrama de bloques la estructura de un computador básico o modelo Von Neumann.*

1.2 *¿Qué es el reloj de un computador? ¿Por qué es necesaria esta señal? Razonar qué ocurre si esta señal deja de aplicarse. ¿Qué pasará si se aumenta la frecuencia de reloj?*

1.3 *Explicar brevemente cómo se representa la información en un computador.: ¿Cómo es posible representar caracteres alfanuméricos y números simultáneamente?*

1.4 *Enumerar los principales registros internos de la CPU. ¿Cuál es la utilidad básica de cada uno de ellos?*

1.5 *¿Qué diferencias hay entre el código de alto nivel como el C, el ensamblador y el código máquina? ¿Puede ejecutarse programa en código máquina en CPUs diferentes? ¿Cuál es el proceso que se debe seguir para ejecutar un mismo programa escrito en C en CPUs diferentes?*

1.6 *Enumerar las funciones básicas de los buses de direcciones, datos, control y alimentación.*

1.7 *¿De qué material están realizados los circuitos integrados? ¿Cómo se consigue integrar millones de transistores en circuitos de unos pocos milímetros cuadrados?*

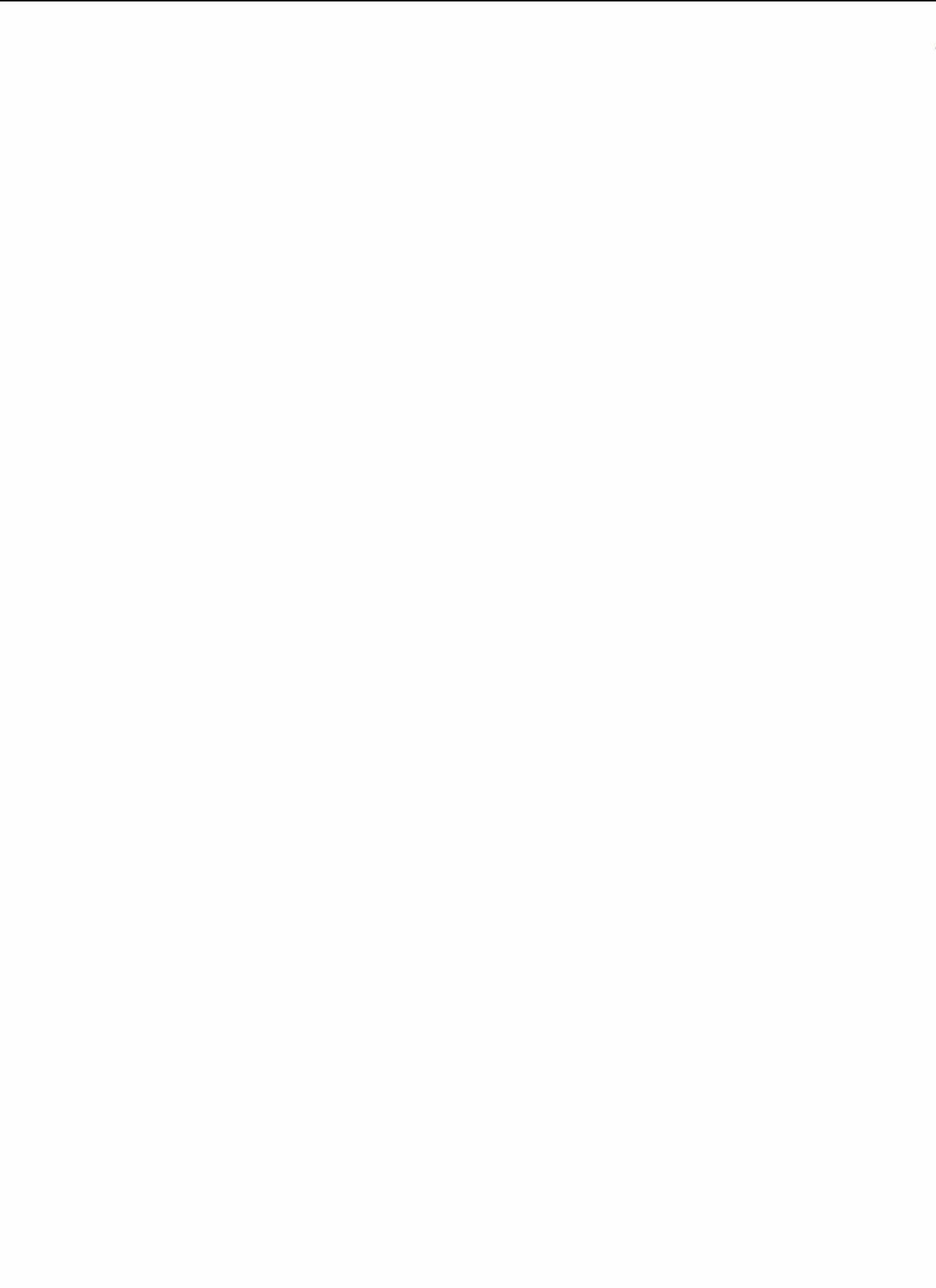
18 *¿Qué es un transductor? Indicar un ejemplo de transductores de sonido.*

19 *Enumerar las partes en las que se divide una CPU básica. ¿Cuál es la función de cada una de ellas?*

1.10 *¿Cuando se dispone de un computador que está alimentado, pero que no está realizando ningún programa de usuario, está parado el procesador? ¿Qué está haciendo?*

1.6 BIBLIOGRAFÍA

- Fundamentos de los **Computadores**. Estructura, funcionamiento **interno**, software de sistemas. *P. de Miguel*. Editorial Paraninfo. Madrid, 1999. ISBN 84-283-1790-9.
- **Estructura y Tecnología de Computadores I**. *C. Cerrada. V. Feliu*. Universidad Nacional de Educación a Distancia. Madrid, 1994. ISBN: 84-362-2963-0.
- **Estructura y Tecnología de Computadores II**. *S. Dormido, M. A. Canto y J. Mim, A. E. Delgado*. Universidad Nacional de Educación a Distancia. Madrid, 1994. ISBN 84-88667-06-X.



CAPÍTULO 2

UNIDAD DE CONTROL: EJECUCIÓN DE INSTRUCCIONES

2.1 INTRODUCCIÓN

En el capítulo 1 se ha introducido la estructura general de un computador. A partir de esta noción básica, el objetivo del presente capítulo es describir de una forma más detallada todos los elementos que lo constituyen, introduciendo el esquema de un computador elemental genérico. Este esquema es útil para estudiar el funcionamiento interno de la CPU a nivel de transferencia de información entre registros. Para iniciar este estudio se introduce el concepto de instrucción máquina desde el punto de vista de la información que debe contener y cómo se divide en los diversos campos que definen su formato. También se analiza brevemente el funcionamiento de los registros, pieza clave en la construcción del computador, mostrando cómo se realiza el transvase de la información desde unos registros a otros. Para representar gráficamente el secuenciamiento de las instrucciones se utilizan los **cronogramas** de ejecución, en los que se muestra la secuencia de activación de las señales de control de todos los elementos internos de computador.

Posteriormente se profundiza en el estudio de la unidad de control (UC), incluyendo sus componentes, la forma en que ejecutan las instrucciones en el procesador y su diseño. Para implementar la UC se tendrán en consideración dos métodos, el primero de ellos utiliza lógica cableada y el segundo utiliza la técnica de microprogramación. Para cada una de las dos alternativas se realizarán ejemplos y se comentarán sus ventajas e inconvenientes.

Por último, se repasa la evolución de los procesadores comerciales más populares, viendo sus características y tendencias de diseño. Como inicio del desarrollo del tema se muestra la figura 2.1. donde se puede observar la arquitectura básica de un computador Von Neumann. Dicha arquitectura está formada por los siguientes elementos:

- Registros generales
- Unidad **aritmético** lógica
- Unidad de control
- **Buses** del sistema
- Registros de comunicación con los **buses** del sistema
- **Buses** internos

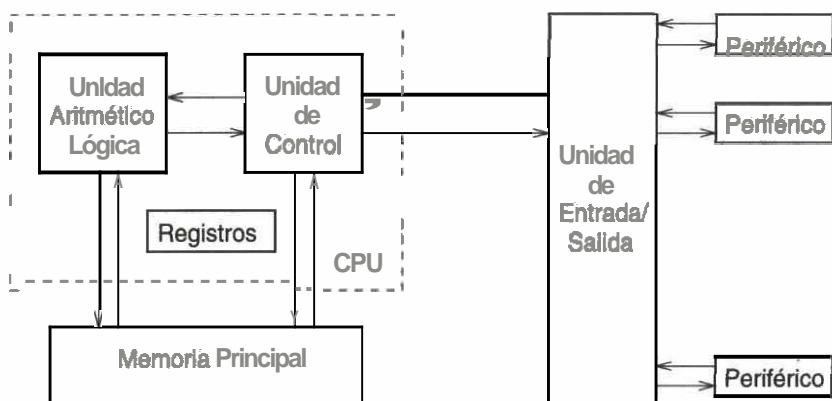


Figura 2.1: Esquema básico de la arquitectura Von Neumann

2.2 REPERTORIO DE INSTRUCCIONES

Para programar las tareas que se desea que ejecute el procesador, los usuarios a menudo construyen programas utilizando lenguajes de alto nivel como son el C o el **Pascal**. Estos programas no son directamente **interpretables** por el computador, sino que están a medio camino entre el lenguaje humano y el

lenguaje de los computadores. Un computador **sólo** entiende el **código binario**, por tanto una descripción de un programa en lenguaje de alto nivel debe ser **traducido** o **compilado** a este lenguaje de unos y ceros interpretable directamente por el **procesador**.

Las instrucciones que interpreta y ejecuta el **procesador** se denominan **instrucciones máquina** o **instrucciones del computador**. La **CPU** realiza diferentes funciones que son un **reflejo** de la variedad de las instrucciones que tiene **definidas**. Todo el **conjunto** de **instrucciones** distintas que **puede ejecutar** la CPU se denomina **repertorio de instrucciones**. El diseño de un **repertorio de instrucciones** es complejo, ya que define las funciones que realiza la CPU y es el medio que tiene el programador de controlarla. En consecuencia **deben** considerarselas necesidades del programador a la hora de **realizar** su **diseño**.

En la construcción de computadores comerciales, lo que interesa es que sean rápidos y económicos, por lo que **habrá** que dotarles de un conjunto bien seleccionado de instrucciones que permitan al usuario formular cualquier tarea de procesamiento de datos. Un programa escrito en alto nivel debe **traducirse** a lenguaje máquina para ser ejecutado, para ello el repertorio de instrucciones máquina debe ser suficientemente amplio como para expresar, con una o varias instrucciones, cualquier instrucción de **un** lenguaje de alto nivel.

El juego de instrucciones de un computador debe cumplir dos condiciones:

- **Completitud:** Debe poder calcular, en un tiempo finito, cualquier tarea computable.
- **Eficacia:** Debe permitir una alta velocidad de cálculo, sin exigir a cambio una excesiva complejidad de la unidad de control ni de la unidad aritmética.

Además, cada una de las instrucciones **máquina** que compone el repertorio de instrucciones debe cumplir unas características generales que son las siguientes:

1. Las instrucciones realizan una **función** única y sencilla, por lo que su **descodificación** por la CPU es **sencilla**.

2. Una misma instrucción emplea siempre un número fijo de operandos. Diferentes instrucciones pueden tener diferente número de operandos.
3. La **codificación** de las instrucciones es sistemática, puesto que facilita su **decodificación**.
4. Las instrucciones son **autocontenidas** e independientes, es decir, contienen toda la **información** necesaria para ejecutarse. Deben expresar: el tipo de operación a realizar, el valor o la posición donde se hallan los **operandos**, el lugar donde **se** tiene que depositar el resultado y la **ubicación** & la siguiente instrucción. La **CPU** ejecutará la instrucción siguiente almacenada en el **código** del programa si no **se** indica explícitamente la dirección de la nueva **instrucción** a ejecutar.

Dependiendo de la operación que realice una determinada instrucción, el **procesador** para ejecutarla puede necesitar **algún** tipo de **información** adicional, que incluye: cuántos operandos **necesita** la operación, dónde están almacenados o dónde debe almacenar el resultado. Los elementos constitutivos de una instrucción máquina con los que se indica al **procesador** toda esta información son:

- **Código de operación: Específica** mediante un **código binario** la operación a realizar. También **se** conoce por su **acrónimo CO** o el de las siglas en inglés *opcode*.
- **Referencia a operandos fuente:** La instrucción contiene los **operandos** de la instrucción o en **caso contrario** indica dónde se encuentran en memoria.
- **Referencia al operando resultado:** Si la instrucción produce **algún** resultado indica dónde se almacena.

Aunque las instrucciones máquina son cadenas de unos y ceros, para identificarlas más fácilmente, **se emplearán** sus nombres más comunes (en inglés) a modo de **nemáticos**. Los códigos nemáticos empleados por los ensambladores permiten a los programadores no trabajar directamente con los códigos binarios, lo que facilita la programación. Una vez realizado el programa **ensamblador** se encarga de traducir la secuencia de **nemáticos** a códigos binarios **interpretables**.

por el computador. Aunque cada procesador implementa un conjunto particular de instrucciones **máquina**, todos los repertorios de instrucciones de distintos computadores guardan una similitud. A continuación se detallan los tres principales grupos de instrucciones que puede presentar un computador.

Instrucciones de transferencia de datos

Las instrucciones de transferencia de datos permiten copiar, en el operando destino, la información almacenada en el operando origen, quedando este último sin modificar. En las instrucciones de transferencia junto al código de operación se debe especificar información referente al dato que se ha de transferir. En primer lugar debe especificar el origen y el destino de la transferencia que pueden ser una posición de memoria o un registro. En segundo lugar, debe indicarse la longitud del dato a transferir. Un ejemplo de este tipo de instrucciones es la instrucción **MOVE *destino, origen*** que transfiere el dato desde la posición origen a la destino.

Instrucciones aritméticas y lógicas

Este tipo de instrucciones efectúan alguna de las cuatro operaciones aritméticas básicas de suma, resta, multiplicación y división, o operaciones lógicas bit a bit como son: AND, OR o XOR. Cada instrucción se aplica sobre unos operandos de un tipo concreto. Estas instrucciones están siempre disponibles para números enteros con signo (punto fijo), existiendo a menudo su equivalente para números en punto flotante. Un ejemplo es la instrucción **ADD *destino, operando1, operando2*** que realiza la suma de los dos operandos almacenando el **resultado** en la posición **destino**.

Instrucciones que modifican la secuencia del programa

Cuando el procesador ejecuta un programa, va leyendo de memoria las instrucciones situadas en las posiciones a las que va apuntando el registro PC de forma sucesiva. Cuando se termina de ejecutar una instrucción, el PC se **incrementa** y se procede a cargar la instrucción siguiente. Las instrucciones de salto permiten modificar la secuencia normal de ejecución del programa. En estas

instrucciones se especifica la dirección de la siguiente instrucción a ejecutar. El procesador al ejecutarlas copia la nueva dirección en el PC, lo que produce un salto en el secuenciamiento de las instrucciones del programa. Un ejemplo de este tipo de instrucciones es **JUMP dirección**, que carga en el registro PC el contenido del campo **dirección**. Existen distintos tipos de bifurcaciones, que se detallan a continuación:

1. **Bifurcaciones incondicionales.** Este tipo de bifurcaciones cambian el secuenciamiento del programa siempre que se ejecutan, modificando el valor del registro PC con el valor contenido en su campo de operandos.
2. **Bifurcaciones condicionales.** En las instrucciones de salto condicional se hace la bifurcación (se actualiza el contador de programa con la dirección especificada en el operando) sólo si se cumple una condición dada, que habitualmente depende del resultado de la ejecución de la instrucción anterior. En caso contrario, se ejecuta la instrucción inmediatamente posterior de la secuencia (se incrementa el contador de programa de la forma habitual). La condición se basa en el estado de los bits del registro de estado, que se actualizan al ejecutarse algunas operaciones. Este registro se llama **registro de estado** e indica, por ejemplo, si en la ejecución de la última operación se produjo un resultado negativo o un desbordamiento.
3. **Bifurcaciones con retorno.** Otro tipo de instrucciones que modifican la secuencia de un programa son las bifurcaciones con retorno. En este grupo se distinguen las instrucciones de salto a subrutina y las de retorno de subrutina. Las primeras producen un salto incondicional a una dirección especificada en el operando, pero con la salvedad de que de forma automática el procesador almacena en memoria, en una zona reservada previamente llamada pila, la dirección siguiente a la instrucción de salto. La pila es una estructura de datos que se organiza según la filosofía **último en entrar, primero en salir**, de ahí el acrónimo **LIFO (Last In, First Out)**. Para manipular una pila, la CPU posee uno o varios registros punteros de la pila llamados **SP (Stack Pointer)** que apuntan a la dirección de memoria donde se encuentra la cima de la pila, es decir, el último dato introducido. El segundo tipo de bifurcaciones con retorno produce un salto incondicional a una dirección que se lee de la pila.

Otras instrucciones

Este grupo contiene instrucciones que no pueden ser incluidas en ninguno de los conjuntos anteriores. El primer ejemplo es la **instrucción NOP**, que no hace nada y **normalmente** se utiliza cuando es necesario un tiempo de espera. Otra instrucción de este grupo es **HALT**, que para la ejecución de instrucciones. Por último, la instrucción **INT** activa una **interrupción** software y rompe la secuencia actual de ejecución de instrucciones. Esta última **instrucción** se explicará con mayor **detailed** en secciones **posterior**.

2.3 MODOS DE DIRECCIONAMIENTO

Las **diversas** formas con las que se puede hacer referencia, tanto a los operando fuentes como a los resultados, **originan** los **modos de direccionamiento**. Este **procedimiento** sirve para **determinar** la **ubicación** del operando que dependiendo del tipo de **instrucción** puede estar en:

- **La propia instrucción:** Parte de los bits que componen una instrucción sirven para almacenar el valor del operando.
- **Memoria principal:** Los datos para realizar la **operación** están **almacena**dos en la memoria **principal**, y parte de los **bits** de la **instrucción** contienen la **dirección** donde se almacena el **operando** en memoria. A esta **dirección** se le llama **dirección efectiva del dato**.
- **Registros internos de la CPU:** Los datos ya se encuentran en los registros de almacenamiento interno de la **CPU** como resultado de la ejecución de operaciones anteriores.

Las diversas técnicas de **direccionamiento** deben presentar un compromiso entre el rango de direcciones al cual **permiten** acceder y la flexibilidad de **direc**cionamiento, así como entre el **número** de referencias a memoria y la complejidad del **cálculo** de las direcciones. Los programas **utilizan** normalmente varios modos de direccionamiento que facilitan:

1. El **ahorro de espacio**. Mientras más cortas sean las instrucciones, menos espacio de almacenamiento **necesitarán** los **programas** y menos bits se

tendrán que leer de memoria principal para ejecutarlo. En **este** sentido, serán convenientes direccionamientos que ocupen poco espacio.

2. **La generación de código reubicable.** Cuando el sistema operativo carga un programa en memoria para ser ejecutado, es deseable que el programa se pueda almacenar en diversas **zonas** de memoria.
3. **El manejo de estructuras de datos.** El manejo de estructuras de datos, tales como: tablas, **matrices**, colas o listas, se simplifican con el empleo de algunos modos de direccionamiento relativos, tal como se mostrarán en las siguientes secciones.

A continuación se analizan con **detalle** los modos de direccionamiento que **utilizan** la mayoría de computadores **actuales**.

Direccionamiento inmediato

La forma más **sencilla** de direccionamiento es el direccionamiento inmediato, en el que el operando este presente en la propia instrucción. El uso de este direccionamiento implica que la instrucción opera con un valor constante, que se **define** a la hora de crear el programa. La principal ventaja del **direcciónamiento** inmediato es que una vez cargada la instrucción, no se requiere una nueva referencia a memoria para obtener el operando, ahorrándose de esta manera un ciclo de memoria en la ejecución de la instrucción. La mayor desventaja es que el rango de valores de un operando inmediato queda limitado por el tamaño del campo empleado por la instrucción para **almacenarlo**, que en la mayoría de los repertorios de instrucciones es pequeño comparado con la longitud de palabra.

Hay procesadores que **permiten** distintos tamaños de **operandos** inmediatos, adaptando la instrucción al tamaño de dato deseado. Al poder tener instrucciones con **formatos** de una sola palabra o de varias palabras, no se desperdicia memoria en los **operandos** con un valor **pequeño**. Además, las instrucciones con este tipo de modo de direccionamiento son **más rápidas** de procesar, puesto que el operando se encuentra ya en el registro de instrucción IR al final de la fase de búsqueda de instrucción. El caso del formato con dos palabras de memoria, donde la segunda palabra contiene el operando inmediato, requiere una **lectura**

de memoria adicional antes de disponer del operando. La figura 2.2 muestra la disposición del operando con este tipo de direccionamiento.

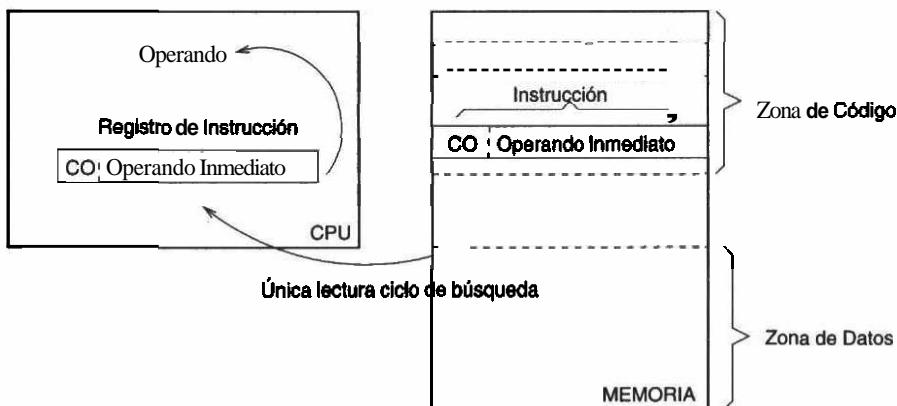


Figura 2.2: *Direccionamiento inmediato*

Direccionamiento directo

Un modo de direccionamiento se denomina **directo** cuando la instrucción contiene la dirección en memoria del operando. El operando puede estar en cualquier zona de la memoria siempre que se encuentre dentro del rango de posiciones direccionables. La principal ventaja es su sencillez, puesto que no necesita ningún cálculo previo para conocer la **dirección** final de memoria del dato. La limitación obvia es que dependiendo del número de **bits** de la instrucción que se utilicen para representar esta dirección se tendrá un rango de posiciones de memoria más o menos amplio. Si éste fuera el único direccionamiento de que dispone el procesador el CD o **código de dirección**, debería tener la suficiente longitud como para direccionar todo el mapa de memoria. Se puede conseguir mayor capacidad de direccionamiento utilizando **formatos** de dos palabras, si bien esto añade una lectura de memoria adicional. En la figura 2.3 se muestra un esquema con el método de obtención del operando con el **direcciónamiento directo**.

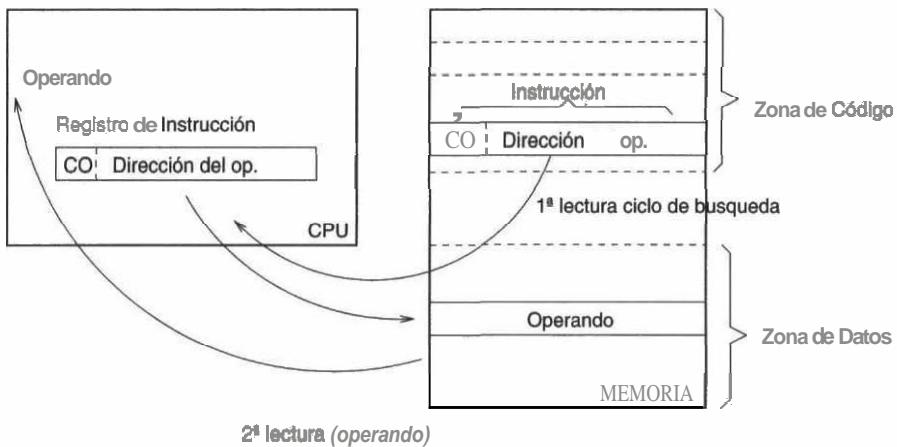


Figura 2.3: Direcciónamiento directo

Direcciónamiento mediante registro

En este tipo de direcciónamiento el operando se encuentra almacenado en uno de los registros internos de la CPU. En la instrucción existe un campo que sirve para identificar el registro que contiene el dato. Una ventaja que presenta este modo de direcciónamiento es que no precisa ningún ciclo de lectura adicional para acceder al dato. Además, la lectura en los registros **internos** es mucho **más** rápida que en memoria principal, con lo que se acelera la ejecución de la instrucción, y **sólo** es necesario un campo pequeño de direcciones en la instrucción.

Para conseguir la **implementación** de programas con buenas prestaciones es necesario **utilizar** este modo de direcciónamiento de **forma** extensiva y eficiente, ya que eliminan accesos a la memoria. Por ejemplo, las variables que utilice un programa es bueno mantenerlas en registros y no en memoria principal. La figura 2.4 **muestra** un esquema de cómo se obtendría el operando.

Direcciónamiento relativo a registro

En el direcciónamiento relativo a registro el dato al cual se desea acceder reside en **memoria**. La dirección efectiva del dato la calcula el **procesador** sumando una cantidad (llamada desplazamiento) al valor contenido en un registro.

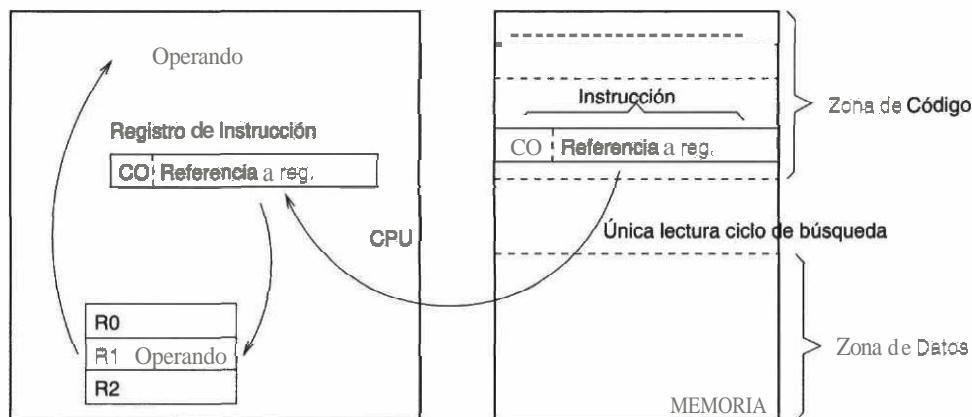


Figura 2.4: Direccionamiento mediante registro

En la **instrucción** se debe especificar el desplazamiento y el **identificador** del registro (aunque este último a veces viene determinado por el tipo de instrucción).

Este modo de **direccionamiento** facilita el acceso a un conjunto de posiciones de memoria especificadas a partir de una dirección **considerada** como referencia. Este modo de direccionamiento permite acceder a todo el espacio de **direcciónamiento** del **procesador**, empleando un número de bits en el campo CD muy inferior al necesario si se utilizase direccionamiento absoluto. En contraposición se requiere realizar la operación de **suma** del puntero w n el desplazamiento antes de obtener la dirección del operando, lo que supone un retraso adicional sin embargo pequeño, **en** comparación con un ciclo de lectura a memoria. por lo que en general el tiempo de ejecución de la instrucción no se **ve** penalizado.

La mayoría de los computadores también permiten desplazamientos **negativos** en tomo a la dirección de referencia, 0 en complemento a dos. Existen varios tipos de **direccionamientos** relativos a registro en función del registro que se **utiliza** como puntero (PC, pila o registro base), pero todos ellos siguen el método que se describe en la figura 2.5.

El direccionamiento relativo al registro contador del **programa** es un caso particular de direccionamiento relativo a **registro** que emplea como puntero el registro contador del programa. Ya que el contador de programa se **incrementa** cada vez que se lee una instrucción, la posición de **referencia** es la de la instrucción siguiente.

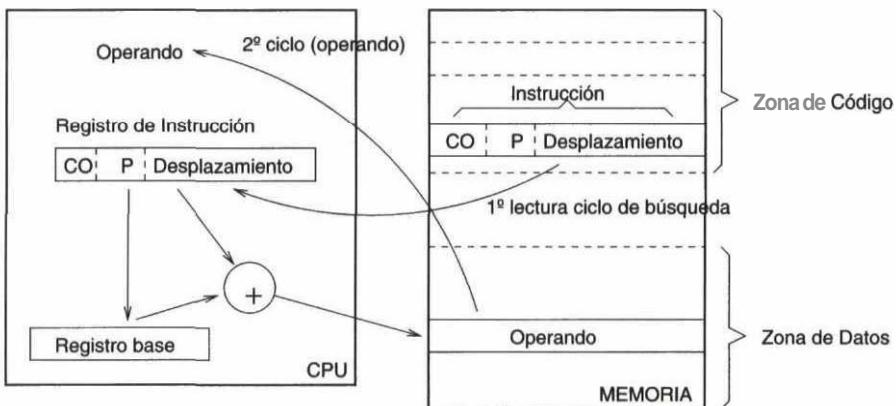


Figura 2.5: Direcciónamiento relativo a registro

Se suele emplear para **direcciónar instrucciones** cercanas a la instrucción en curso y para ejecutar **saltos** relativos. En estos **últimos**, al ejecutarse la instrucción, la nueva dirección calculada se almacenará en el mismo **contador** de programa.

El **direcciónamiento relativo** al **registro base** emplea como puntero un registro de propósito general o dedicado exclusivamente para este fin que se designa **registro base**. Dado que los computadores disponen de varios registros que pueden actuar como base, la instrucción **deberá** contener un **identificador** del mismo, además del desplazamiento, para calcular la dirección efectiva del dato tal **como** indica la figura 2.5.

Este direcciónamiento es muy conveniente cuando se dispone de una zona de datos (como un conjunto de variables). Cargando en el registro base la primera posición de esta zona, se pueden alcanzar los distintos datos **sólo** con conocer su posición relativa en la zona de datos.

El **direcciónamiento relativo a pila** suma el desplazamiento contenido en la **instrucción** con el valor del registro SP para obtener la **dirección** del objeto. En ocasiones ni siquiera existe tal campo de desplazamiento y las instrucciones de **direcciónamiento** de la pila se **refieren implícitamente** al elemento que se encuentra en la cima, siendo las operaciones de introducir un **dato** y extraer un dato las únicas permitidas de manejo de pila. Este direcciónamiento permite instrucciones muy compactas, puesto que la instrucción no requiere ningún desplazamiento.

Direccionamiento indexado

El mecanismo de direccionamiento es exactamente el mismo que el visto para el relativo a registro base, con la diferencia significativa de que el valor del registro índice se suele modificar con frecuencia en la ejecución de un programa, mientras que en el caso de direccionamiento relativo al registro base no.

El direccionamiento indexado se utiliza habitualmente como mecanismo eficiente **para** recorrer estructuras de datos tipo vector o tabla. Para recorrer este tipo de estructuras, el procesador realiza incrementos o decrementos del registro de forma automática tras cada referencia. Esto se denomina auto-indexado y da lugar a cuatro variantes de este modo de direccionamiento:

- Preautoincremento y Preautodecremento: El registro índice es **incrementado** (o **decrementado**) y seguidamente se obtiene la dirección como suma del registro índice más el desplazamiento.
- Postautoincremento y **Postautodecremeno**: **Primero** se calcula la dirección sumando el registro índice más el desplazamiento y seguidamente se incrementa (decrementa) el registro índice.

Direccionamiento indirecto

El problema del direccionamiento directo es que la longitud que tiene el campo de direcciones en la **instrucción** es normalmente menor que el número de **bits** que utiliza el procesador para especificar las direcciones, por lo que se limita el rango de direcciones accesible. El direccionamiento indirecto soluciona este problema, ya que el campo de direcciones contiene la dirección de una palabra de memoria que no contiene el dato, sino la dirección de memoria donde se encuentra éste. Un esquema de este modo de direccionamiento se muestra en la figura 2.6.

La ventaja obvia de esta aproximación es que ya no existe la limitación en el conjunto de direcciones accesible. La desventaja es que la ejecución de la instrucción requiere dos referencias a memoria para obtener el operando: una para obtener su dirección, y otra para obtener su valor.

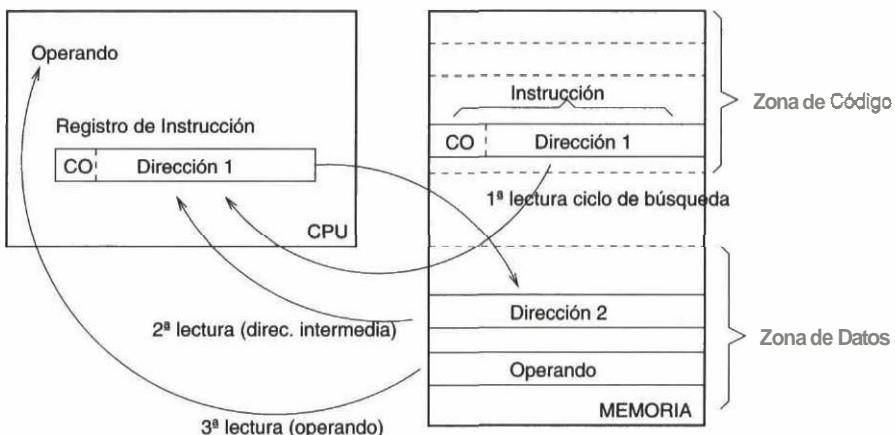


Figura 2.6: Direccionamiento indirecto

Este tipo de **direccionamiento** **está** especialmente indicado en aplicaciones que utilizan datos situados en posiciones distantes en la memoria, por ejemplo, la transferencia de **parámetros** de un programa principal a **subrutinas**.

El direccionamiento indirecto se puede combinar con todos los tipos de **di**-recionamiento relativos vistos anteriormente. Así mismo, se pueden plantear direccionamientos directos de más de un nivel, aunque esto no tenga una gran utilidad, puesto que exige varios accesos a memoria y ocupa varias posiciones de memoria.

Una variante del direccionamiento indirecto es el **direccionamiento indirecto con registro**, en el cual el campo de direcciones, en lugar de hacer referencia a la **dirección** de una palabra de memoria, hace referencia a un registro que contiene la **dirección** en memoria del operando. **Nótese** que el **direcciona**-miento indirecto con registro emplea una referencia menos a memoria que el diicionamiento indirecto.

En la tabla 2.1 se muestran, de forma comparativa, las principales ventajas y **desventajas** de los diversos modos de direccionamiento descritos en el **capítulo**. En dicha tabla los corchetes [Dir] deben leerse como **contenido de la dirección Dir** y **Dir(Op)** como **dirección del operando**.

Modo	Algoritmo	Principal ventaja	Principal desventaja
Inmediato	$Op=A$	No referencia a mem.	Operandos limitados
Directo abs.	$Dir(Op)=A$	Sencillo	Espacio de dir. limitado
Mediante reg.	$Op=R$	No referencia a mem.	Espacio de dir. limitado
Relativo a reg.	$Dir(Op)=[R]+A$	Flexibilidad	Complejidad
Indexado	$Dir(Op)=[R]+A$	Flexibilidad	Complejidad
Indirecto	$Dir(Op)=[[A]]$	Espacio de dir. grande	Referencia extra a mem.

Tabla 2.1: Ventajas y desventajas de los modos de direccionamiento

2.4 FORMATO DE INSTRUCCIONES

Como se ha visto **anteriormente**, cuando se crea un programa se almacena en un conjunto de posiciones en la memoria como una serie de 1's y 0's. El código **binario** de cada instrucción se debe interpretar para diferenciar entre los diferentes campos que contiene la instrucción (código de operación, modo **direcciónamiento, etc**). El formato de la instrucción indica los campos y el tamaño de los mismos para cada instrucción. En el diseño del formato de las instrucciones la primera decisión a considerar es el tamaño de las instrucciones. Para tomar esta decisión se tienen que considerar los siguientes aspectos.

- El tamaño de la memoria
- La organización de la memoria
- La estructura de **buses**
- La complejidad de la **CPU** y su velocidad

El diseño del formato define la riqueza y flexibilidad del procesador desde el punto de vista de su programación. El compromiso más obvio está entre el deseo de disponer de un repertorio de instrucciones máquina potentes y la necesidad de simplificar su **decodificación** y **ejecución**. Con la idea de simplificar su decodificación, la instrucción se divide en una serie de campos (cadenas de bits contiguos), estando referido cada campo a **un tipo de información** específico. Los dos campos básicos en un formato son el código de operación (CO), que indica la operación a realizar, y el campo de **dirección** (CD), que determina la

dirección de un dato, resultado, o **instrucción** a la que hay que bifurcar. El campo de **dirección**, dependiendo del tipo de direccionamiento, se divide en **subcampos**. Además de estos dos campos básicos, se suelen encontrar campos de extensión del código de operación, campos de longitud de operandos, campos de modo de direccionamiento, etc.

Las **características** generales que deben reunir los **formatos** de instrucciones son:

1. Un computador tiene uno o unos pocos **formatos** de **instrucción**. Cuantos menos **formatos** se tengan, **más** sencilla será la unidad de control que deba decodificarlos.
2. Los **formatos** son **sistemáticos**. Los campos del mismo tipo tienen la misma longitud y deben ocupar las mismas posiciones dentro de las instrucciones. De **esta** manera, se simplifica la **decodificación**. El primer campo suele **ser** el código de operación.
3. Para acortar el **tamaño** se emplean ampliamente las **técnicas** de **direccionamiento implícito**. **Así**, exceptuando las instrucciones de bifurcación, no existe campo de instrucción siguiente. Se sobreentiende que **está** en la **dirección** siguiente. **Así** mismo, no suele existir campo para especificar la representación de los **operandos**, el código de **operación** implica el tipo de operandos.
4. Los **tamaños** de los **formatos** encajan fácilmente en la palabra del computador. Lo **más corriente** es que el formato ocupe una palabra, **pero también** son frecuentes los **formatos** de **1/2**, de **1 + 1/2**, y de 2 palabras.
5. Cuando el computador tiene varios formatos, el código de operación diferencia entre ellos.
6. Los **tamaños** de los campos que expresan direcciones, ya sean del banco de registros o de memoria principal, deben corresponder a los mapas de direccionamiento del **procesador**.

Campo de **código** de operación

Este campo especifica la operación que realiza cada instrucción. Hay tantos códigos de operación distintos como instrucciones diferentes posea el **computa-**

dor. El **número** de instrucciones que constituyen el repertorio de un computador vendrá limitado por la longitud de este campo. Todas las instrucciones contienen el campo de **código** de operación, a diferencia del campo de **dirección** que es opcional. El código de operación implica **normalmente** el tipo de operando. Por **ello** se emplean códigos de operación distintos para diferenciar una misma operación con distintos tipos de datos. Cuando las instrucciones pueden tener **más** de un formato, el código de **operación**, que siempre es el primer campo del formato, debe indicar el formato de la instrucción **actual**. De esta manera, la **unidad** de control puede **saber** a partir de la primera **palabra** leída si ha de **hacer** más accesos a **memoria** o no para completar la **instrucción** en curso.

Si el tamaño del campo de código de operación es fijo para todos los posibles **formatos** de instrucciones de un **procesador**, y este campo tiene **a bits**, se podrán tener 2^a códigos diferentes cada uno de **ellos** asociado a una **instrucción** diferente. Dado que la frecuencia de utilización de las **instrucciones** difiere mucho de unas a otras, para **optimizar** el espacio ocupado por los programas, se puede emplear una **codificación** que utilice menos bits para las instrucciones más frecuentes. **Pero** la utilización de esta técnica complica mucho el diseño de la unidad de control. Algunos **procesadores** emplean como solución la extensión del campo del código de operación. Esta **técnica** consiste en el uso de un campo de código de operación fijo y reducido, con el que se codifican las operaciones más utilizadas. Para la codificación del **resto** de instrucciones se usa, junto con un **código** de **operación** especial que no se corresponde con ningún código perteneciente a instrucciones anteriores, un campo de extensión del código de operación que permite identificar las instrucciones restantes.

Campo de dirección

Las instrucciones pueden contener varios campos de dirección para identificar las posiciones de memoria o los registros internos donde se encuentran sus operandos, así como la posición del resultado. Cada instrucción puede tener un número diferente de operandos. Hay instrucciones sin operando, como **HALT** o **NOP**, o con un solo operando, como las instrucciones de salto incondicional o de llamada a **subrutina**. Las instrucciones de dos operandos son las que realizan operaciones aritmético lógicas diádicas, que guardan el resultado en uno de los operandos origen, perdiéndose su contenido anterior. De tres operandos son también las instrucciones que realizan operaciones aritmético lógicas diádicas,

pero que almacenan el resultado en un lugar distinto de cualquiera de los dos operandos origen. El número de campos de dirección que deben tener los **formatos** para minimizar la memoria **requerida** es difícil de evaluar, ya que depende de muchos factores: la arquitectura considerada, el tipo de algoritmo a ejecutar, la forma de programación, etc.

Los campos de dirección pueden dividirse a su vez en subcampos tal como se muestra en la figura 2.7. El motivo de esta división es que para **determinar** la dirección del operando, es posible utilizar varias técnicas o modos de direccionamiento, pero cada una de ellas tiene distintas necesidades de codificación.

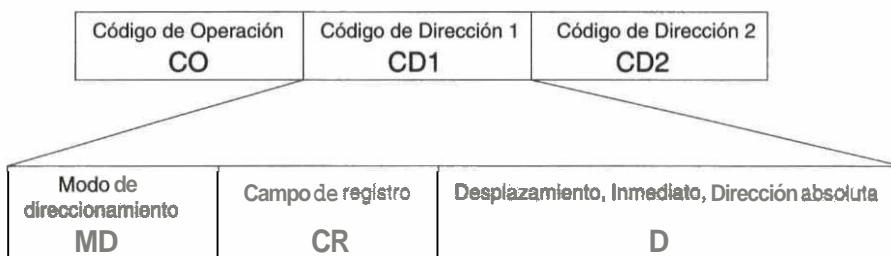


Figura 2.7: Subcampos en el campo de dirección

Los subcampos más significativos contenidos en el campo de direcciones son:

- e Modo de **direccionamiento (MD)**: Es un campo donde se **codifica** el modo de direccionamiento a emplear para localizar ese dato.
- e Campo de **registro (CR)**: Especifica un registro de la CPU.
- e Campo de **dirección (D)**: Dependiendo del modo de direccionamiento empleado, contiene la dirección absoluta, el desplazamiento relativo, el operando inmediato, etc.

Los tamaños de los campos que expresan direcciones absolutas, ya sean de los registros internos o de la memoria, deben abarcar todo el **mapa de direcciones**. Sin embargo, los campos de desplazamiento no tienen esta necesidad.

Campo de condición

En las instrucciones de salto condicional se ha de codificar la condición necesaria para que se produzca el salto, para ello se emplea un campo específico para indicar esta condición. A continuación se mostrarán dos métodos para codificar la condición.

1. El campo de **condición** consiste en una máscara binaria de condición. La **máscara**, tal como se muestra en la figura 2.8, contiene un bit por cada uno de los bits del registro de estado que pueden utilizarse en la condición. **También** puede contener un bit para cada indicador complementado. Se **fijarán** a 1 los bits de la máscara correspondientes a los bits del registro de estado que se deberán examinar, dejando a 0 los demás. La **CPU** utiliza la máscara para realizar ciertas operaciones lógicas con los valores actuales del registro de estado. **Así**, si el resultado de la operación lógica es cierto se producirá el salto, en caso contrario se seguirá la secuencia ordinaria de ejecución de instrucciones.

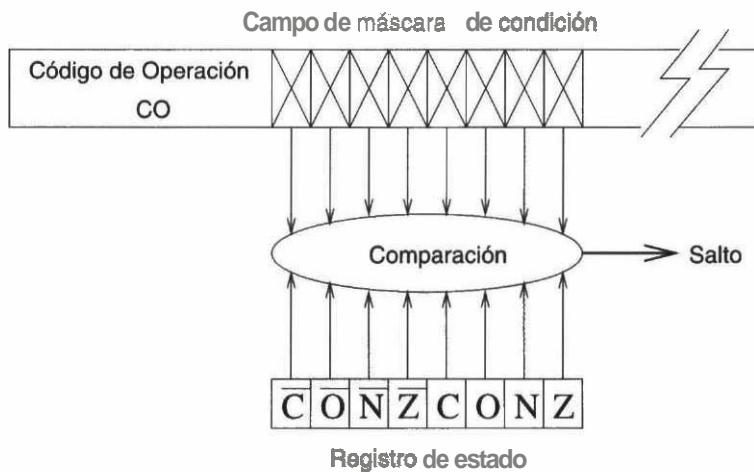


Figura 2.8: Campo de máscara de condición en las instrucciones de salto

2. **Otra** alternativa consiste en la codificación **directa** de la condición de salto en el **campo** de condición. En este método se emplea un código **binario** distinto para cada una de las condiciones contempladas por las **instrucciones** de salto. La unidad de control necesita **realizar** una **decodificación** previa para determinar los bits de estado implicados en cada condición.

2.5 ARQUITECTURA DE UN COMPUTADOR ELEMENTAL

Como base para la definición y explicación de los conceptos del presente capítulo, se va a definir un computador elemental genérico, basado en la arquitectura Von Neumann. Aunque es un ejemplo con una arquitectura muy elemental, incluye las unidades funcionales que todo computador debe poseer, por lo que **servirá** para sentar las bases que **permitan** el **estudio** de arquitecturas más complejas y de mayores prestaciones.

2.5.1 Esquema del computador elemental

En una primera división el computador elemental se compone de procesador (CPU) y memoria. Los elementos & entrada salida y los **periféricos** no se incluyen, con el fin de simplificar el funcionamiento general. Dentro de la CPU se encuentra la Unidad de Control (UC), la Unidad Aritmético-Lógica (ALU) y los registros internos. Tanto los módulos internos de la CPU como los externos, para poder intercambiar información entre ellos necesitan estar **interconectados** por medio de **buses**. Estos **buses** son simplemente hilos metálicos que conducen las señales eléctricas desde un elemento hasta otro. El estudio **detallado** de los **buses** se realizará en el capítulo 6. La figura 2.9 muestra el esquema completo del computador.

A continuación se **revisarán** cada uno de los elementos que componen el computador elemental ejemplo, definiéndolas especificaciones funcionales y las **señales** necesarias para su control. Todas estas señales son activadas de **forma** secuencial por la UC para **realizar** la lectura y ejecución de las **instrucciones** que forman el **programa** y que se encuentran almacenadas en la memoria.

Registros internos

En el capítulo 1 se introdujo el concepto de registro interno. Estas celdas de memoria internas a la CPU guardan datos provenientes de la memoria, **resultados** parciales de las operaciones o información sobre el estado actual de la CPU. Cada procesador tiene un número determinado de registros internos. Un aspecto

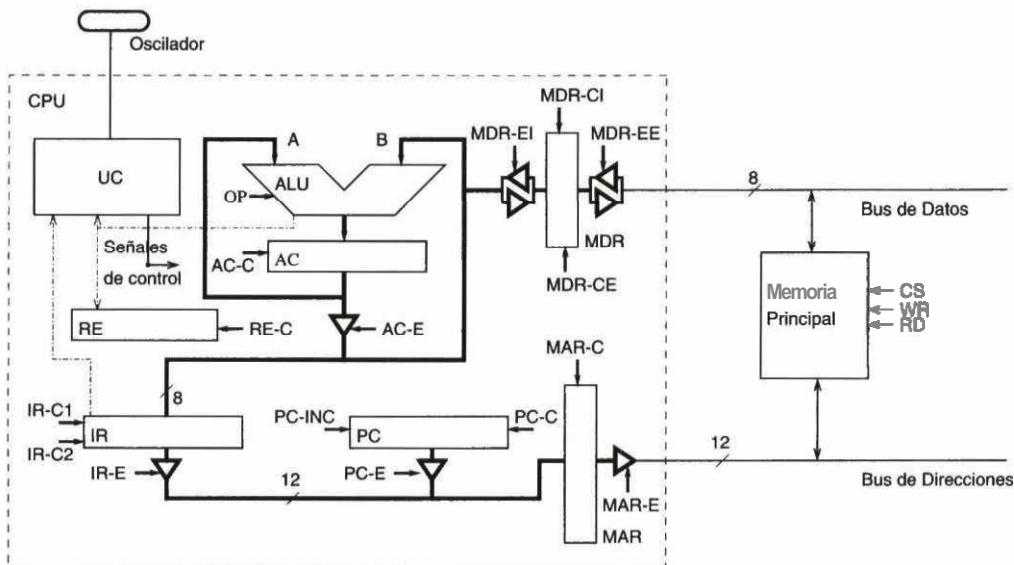


Figura 2.9: Estructura de un computador elemental Von Neumann

importante a tener en cuenta es el tamaño de los registros internos de datos, que indica cuál **es** el **número** máximo de bits que puede tener un operando, y con cuántos **bits** la CPU puede realizar operaciones en paralelo. A este valor se le llama Longitud de palabra del **procesador**. Los registros internos más usuales son:

- Registro de **Instrucción**: Es el registro donde se **almacena** la instrucción a ejecutar una vez que la CPU la ha leído de memoria. Este registro suele estar dividido en campos, cada uno de los cuales tiene un número de bits variable con cada arquitectura. Los campos que forman el registro vienen dados por el formato de las instrucciones del procesador, y dividen el registro en los campos del **código de operación** (CO), direcciones de operando (CD), etc. En la CPU ejemplo se considera que **este** registro tiene **16** bits, por lo que para completar la lectura de una instrucción se deben leer de memoria dos palabras de **8 bits**. Los primeros 4 bits son el campo de **código de operación** y con ellos se indica de qué **instrucción** se trata. A continuación **hay** 12 bits para el campo de **operando**. Este operando se especifica **siempre mediante** **direcciónamiento directo**, en el cual el **campo de la instrucción** contiene la **dirección** de memoria donde se **encuentra el** operando.

Para poder almacenar las instrucciones que se leen de la memoria, el IR utiliza dos señales. La primera, llamada (**IR-C1**), carga los 8 bits más bajos del registro (**IR[7:0]**). La segunda, llamada (**IR-C2**), carga los 8 bits **más** altos (**IR[15:8]**). Para habilitar la salida al bus de direcciones del contenido del campo CD (**IR[4:15]**) se **utiliza la señal (IR-E)**.

- Registros de propósito general: Son registros internos a la CPU que sirven para almacenar datos o resultados intermedios. En el **procesador** ejemplo se dispone de un Único registro, llamado acumulador (AC) , que almacena el resultado de la última **operación** realizada en la ALU. Este **registro** posee dos señales, una de carga (AC-C) y otra de habilitación de la salida de datos al bus interno (AC-E). En otros **procesadores** el número de registros de este tipo suele ser muy **superior**, permitiendo guardar en la propia CPU datos que se quieren modificar o utilizar de forma frecuente.
- **Registro** de estado: Almacena en **cada uno** de sus bits de **condición**(también llamados **flags**), **información** sobre los resultados de operaciones anteriores. En la CPU del computador ejemplo se incluyen los siguientes indicadores, por ser los **más** frecuentes:
 - Z: Es el indicador de cero. Almacena un 1 cuando el resultado de la última operación **realizada** en la ALU **es** cero.
 - N Es el indicador de negativo. Almacena un 1 cuando el resultado de la última operación realizada en la ALU es menor que cero.
 - C: Es el indicador de acarreo. Almacena un 1 cuando el resultado de la última **operación** de la ALU produjo un acarreo en el último bit.
 - O: Es el indicador de desbordamiento. Almacena un 1 cuando el resultado de la última **operación** realizada en la ALU y almacenado en el registro AC es incorrecto. Esta **situación** se produce debido a que el resultado no se puede representar con los bits que tiene el AC.
 - P: Es el indicador de paridad. Almacena un 1 cuando el número de unos del resultado de la última operación realizada **en** la ALU es par.
 - CP: Es el indicador de acarreo parcial. Almacena un 1 cuando el resultado de la **última** operación la **última** operación **en** la ALU produjo un acarreo entre el **cuarto** y el **quinto** bit.
- Contador de programa PC. Este registro contiene la **dirección** en **memoria** de la próxima **instrucción** a ejecutar por el computador. Debe ser

actualizado por la unidad de control cada vez que se carga una **instrucción**. El tamaño del registro PC de la arquitectura ejemplo es de 12 bits, y **se** incrementa dos veces en **cada** lectura de una instrucción, ya que las instrucciones son de 16 bits y ocupan dos posiciones de **memoria** de 8 bits cada una. El registro PC es un contador y se **incrementa** su valor cada vez que se activa la **señal** (PC-INC).

- **Registros de interfaz con el bus.** Son registros internos a la CPU que almacenan datos que el procesador envía o **captura** del bus **externo** de direcciones o del de datos. El registro MAR almacena la dirección que se pretende poner en el bus externo de direcciones. En el registro MAR la transferencia es **unidireccional**, por ello sólo hay una **señal** de habilitación **triestado** para depositar la dirección en el bus externo (MAR-E), y una **señal** de carga para tomar los datos del bus interno (MAR-C). Por otro lado, el registro MDR contiene el dato que el procesador va a enviar o a leer del bus externo de datos. La transferencia de información del registro es **bidireccional**, ya que los datos que **se** cargan en **él** provienen del bus interno del procesador y se dirigen hacia el bus externo o viceversa. En el caso del MDR hay dos señales de carga, una para datos procedentes del bus interno (MDR-CI) y otra para los datos del bus externo (MDR-CE), igualmente las **señales** de **habilitación** se duplican en interna (MDR-CI) y externa (MDR-CE).

Unidad Aritmético-Lógica: ALU

Como ya se ha indicado en el capítulo 1, la ALU es la parte de la CPU encargada de **realizar** todas las **operaciones** aritmético-lógicas dentro del computador. En el caso del computadorejemplo se tiene una ALU de dos entradas, la entrada A recibirá los datos directamente del AC y la entrada B recibirá los datos provenientes del registro interno MDR. Antes de realizar una operación se tendrá que cargar el registro **MDR** con uno de los operandos y habilitar la salida de datos de este registro al bus interno de datos. El otro operando deberá estar almacenado en el registro acumulador (AC).

La ALU recibe la **información** sobre el tipo de operación que debe realizar, mediante la activación de las entradas de operación (OP). En el computador ejemplo estas operaciones quedan restringidas a las 6 siguientes:

- e **OP(001)=A+B**: suma de los valores de las entradas.
- **OP(010)=A+1**: suma del valor de la entrada A y 1.
- **OP(011)=A-B**: resta de los valores de las entradas.
- **OP(100)=A and B**: AND lógico bit a bit de los valores de las entradas.
- **OP(101)=A or B**: OR lógico bit a bit de los valores de las entradas.
- e **OP(110)=B**: muestra por la salida el valor de la entrada B.

Debido a que son 6 operaciones distintas se tendrán que codificar utilizando 3 líneas de operación. Las entradas a la ALU serán por tanto **(OP[3:1])**. También se **supondrá que** el tiempo necesario para realizar **cualquiera** de las operaciones no supera la mitad de un periodo de la señal de reloj que gobierna el computador.

Unidad de control: UC

Su función principal es el gobierno interno del computador. Se encarga de leer la instrucción adecuada de la memoria, de decodificarla y de ejecutarla. La ejecución se consigue mediante la generación de la secuencia de órdenes adecuadas aplicadas a cada elemento del computador. Por tanto la UC tendrá que activar todas las señales de control necesarias para:

- e Realizar la transferencia de información entre registros. En cada transferencia de datos sólo puede haber un registro origen, por lo que no deben activarse las salidas triestado de dos registros a la vez. En cuanto a la carga de estos datos en el registro destino se realizará mediante la señal de carga que será activa por flanco. Esta **señal** se activa a mitad del periodo de la señal de reloj para permitir que los datos provenientes de otro registro estén estables en **el** bus.

- Generar las señales de operación de la **ALU**, dependiendo de si la **instrucción** que se quiere ejecutar realiza alguna operación aritmético-lógica. Anteriormente se deben situar los datos en sus entradas correspondientes habilitando los registros de datos AC y **MDR**.

Memoria principal

La memoria principal es la parte del computador en la que se almacenan instrucciones y datos. La memoria se implementa físicamente mediante un conjunto de circuitos integrados, y se comporta como un conjunto ordenado de registros. El contenido de estos registros se puede leer y modificar de forma aleatoria activando un conjunto de señales (líneas de dirección), que los identifica individualmente de forma unívoca. Además la memoria posee el siguiente conjunto de señales de control:

- CS: Selección de la memoria.
- RD: Lectura.
- WR: Escritura.

Para realizar un ciclo de lectura de memoria, la UC debe de llevar la dirección de la posición de memoria que desea leer al registro **MAR**. Este registro tiene unos **drivers** que permiten activar el bus de direcciones externo del procesador que conecta con la memoria. A continuación el procesador activa las señales CS y RD, y espera a que la memoria deposite los datos en el bus. Este periodo de espera es el tiempo de acceso de la memoria, definido como el tiempo que necesita la memoria para almacenar o sacar los datos, medido desde la activación de las señales de control. Se supone que en el computador ejemplo este tiempo de acceso es de medio periodo de la señal de reloj del procesador. Para finalizar el acceso, los datos provenientes de la memoria se cargan en el registro MDR. Para ello la UC debe de activar la señal de carga de este registro. Esta secuencia de activación de señales se **cumplirá** tanto en los ciclos de lectura de instrucciones como en los de datos.

Los ciclos de escritura comienzan de igual forma con la carga de la dirección en el registro MAR y del dato a **escribir** en el registro **MDR**. Cuando estas señales están estables en los **buses** externos se activan las **señales** CS y WR.

2.5.2 Operaciones con registros

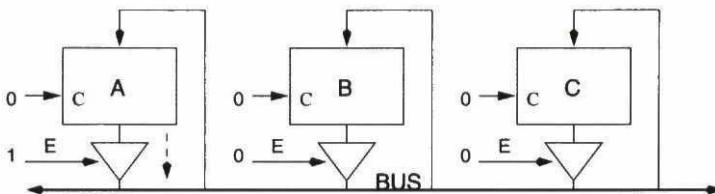
Un registro se puede considerar como una celda de memoria formada por **biestables** en los que **se** almacena un dato o una palabra de **información**. La operaciones básicas que se pueden realizar sobre él son las siguientes:

- Operación de carga: Almacena un nuevo valor en el registro. Para realizarla será necesario poner el nuevo valor en la entrada y posteriormente activar la **señal** de carga del registro.
- Operación de **l e .**: Permite la salida del dato contenido en el registro. Habitualmente todos los registros internos del **procesador** comparten un mismo bus para realizar la **transferencia** de información, pero **sólo** uno de ellos podrá usar el bus en un determinado momento. La **señal** de salida o **habilitación** actuará sobre puertas **triestado** para conectar el registro al bus y por tanto **permitir** que los datos contenidos en el registro pasen al bus.
- Operación de **reset**: Inicializa el valor contenido en el registro a un valor conocido.
- Operación de **incremento**: Algunos registros pueden incrementar su valor de forma autónoma, estos registros se denominan contadores.
- Operación de desplazamiento: Hay registros que presentan operaciones de desplazamiento de sus **bits** hacia la **izquierda** o derecha. Este tipo de operación se verá con más detalle en el capítulo 3.

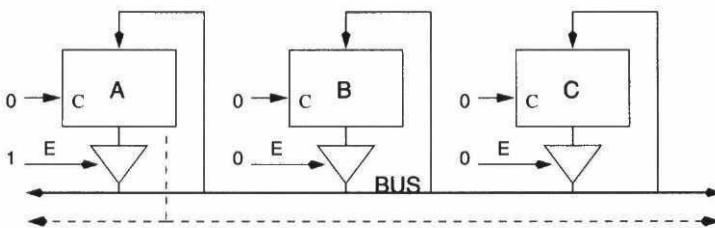
Las operaciones de transferencia sobre registros son útiles para transferir el valor contenido en uno de **ellos** hacia otro. **Siempre** debe haber un **registro** origen, en el cual se encuentra el dato original, y un registro destino en el cual se almacena. Además debe **haber** una conexión entre los dos registros para que se produzca el intercambio de información, que puede ser un **bus** de datos compartido por más de dos elementos. Primero hay que permitir volcar los datos desde el registro inicial **al** bus mediante la habilitación de sus puertas **triestado**. Simultáneamente hay que deshabilitar las salidas **triestado** el resto de los elementos con acceso al bus, para evitar la generación de valores **erróneos** o incluso averías físicas. Una vez que los datos se han **estabilizado** en el bus hay que indicar al registro destino que los datos ya son correctos y que los puede

capturar. Esto se realiza mediante la **activación** de la línea de carga de este registro. Un esquema del proceso a seguir en la transferencias se observa en la figura 2.10.

Paso 1: Habilitación del registro origen y deshabilitación de otras salidas al bus



Paso 2: Estabilización de los datos en el bus



Paso 3: Captura de los datos ya estabilizados

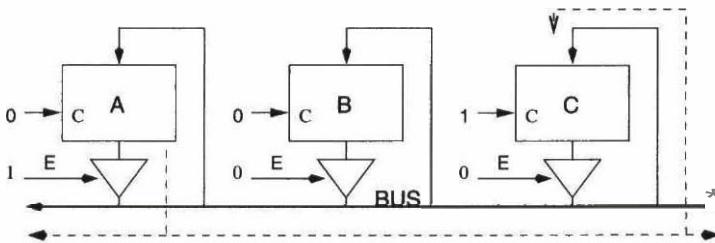


Figura 2.10: Transferencia elemental mediante un bus

Otro modo de realizar la selección del camino de conexión entre el registro origen y el registro destino es mediante la utilización de **multiplexores**. La salida del multiplexor se conecta a la entrada del registro destino, y las entradas del multiplexor a la salida de cada posible registro origen. Para realizar la transferencia hay que seleccionar el origen de los datos mediante las señales de control, esperar a que se estabilice el dato, y habilitar la señal de carga del registro destino.

25.3 Instrucciones

El repertorio de instrucciones de un computador está formado por el conjunto de **todas** las instrucciones que puede **ejecutar**. En el computador ejemplo se considera que el **microprocesador** es capaz de ejecutar **16** instrucciones, aunque hay **destacar** que en los **procesadores comerciales** actuales este número es mucho mayor. Una vez que la UC ha cargado la instrucción a ejecutar en el registro **IR**, **decodifica** el campo **CD** para saber de **qué** instrucción se trata. En el computador ejemplo este campo tiene **4 bits**, con los que se pueden codificar **16** instrucciones distintas. Estas instrucciones se pueden agrupar en 4 tipos:

- instrucciones **aritméticas** y lógicas: Son instrucciones que realizan una **operación** en dos datos contenidos, uno en la posición de memoria cuya dirección esta contenida en el campo **CD** y otro en el acumulador. El resultado de las operaciones se guarda en el AC. La UC es la encargada de transferir los valores iniciales a las entradas de la ALU, de activar el **código** de la operación en las **señales OP**, y de guardar el resultado en el registro AC. Se han **implementado** 5 instrucciones diferentes:
 - **ADD CD**: Suma los dos operandos y guarda el resultado en el AC.
 - **INC AC**: incrementa el valor del acumuladore en una unidad.
 - **SUB CD**: Resta los dos operandos y almacena el resultado en el AC.
 - **AND CD**: Instrucción lógica and bit a bit de los operandos almacenando el resultado en el AC.
 - **OR CD**: Instrucción lógica or bit a bit de los operandos almacenando el resultado en el AC.
- instrucciones de transferencia de datos: Son instrucciones que copian el dato contenido en el registro interno AC de la CPU a memoria y viceversa. En el **campo** de dirección de la instrucción se tiene que **indicar** la posición de memoria donde se desea escribir o leer el dato.
 - **LW CD**: Lee el valor contenido en la posición de memoria **CD** y lo almacena en el AC.
 - **SW CD**: Escribe el valor contenido en AC en la posición de memoria indicada por **CD**.

- Instrucciones de salto: Son instrucciones que alteran la secuencia natural de ejecución de un programa. El campo CD de la instrucción contiene la dirección de la nueva instrucción a ejecutar. La instrucción sustituye el contenido del registro PC que contiene la dirección de la próxima instrucción a ejecutar con el valor del campo CD.
 - **JMP CD:** La próxima instrucción a ejecutar es la contenida en memoria en la posición CD. Se produce un salto de **forma** incondicional.
 - **JM- CD:** Instrucción de salto condicional. Dependiendo del valor del indicador de estado correspondiente se rompe la secuencia de ejecución de instrucciones. Si se cumple la condición la CPU saltará a ejecutar la instrucción contenida en la posición CD, en caso contrario se ejecuta la instrucción siguiente a la actual. Las condiciones de salto están almacenadas en el registro de estado de la CPU. Se pueden distinguir varias instrucciones distintas dependiendo del indicador considerado. En cada caso el procesador saltará si el indicador adecuado está activo.
 - * **JMZ CD:** indicador de cero.
 - * **JMN CD:** indicador de negativo.
 - * **JMC CD:** indicador de acarreo.
 - * **JMO CD:** indicador de desbordamiento.
 - * **JMA CD:** indicador de paridad.
 - * **JMCP CD:** indicador de acarreo parcial.
- Otras instrucciones: Dentro de este grupo se incluyen instrucciones de carácter general no contenidas en ningún **grupo** anterior. En el procesador ejemplo se han **implementado** dos instrucciones:
 - **NOP:** Instrucción que indica no hacer nada.
 - **STP:** Instrucción de paro del procesador. Al ejecutarla el procesador para la secuencia de lectura y ejecución de instrucciones.

Todas las instrucciones presentan el mismo formato, donde los 4 bits menos significativos contienen el código de operación, y los 12 bits siguientes el código binario del campo de dirección, que permite direccionar 4 **Kbytes** de memoria. En el caso de la instrucción **INC**, que no precisa de campo de dirección,

el contenido del mismo no se tiene en cuenta. Ya que todas las instrucciones tienen la misma longitud, el ciclo de búsqueda de las instrucciones consta de 2 accesos a memoria, con lo que se **simplifica** el **diseño** de la UC al no haber casos especiales. En los **procesadores** reales esta circunstancia puede cambiar al haber instrucciones de diferentes tamaños, que necesitan un **número** diferente de accesos a memoria para su lectura.

2.6 EJECUCIÓN DE LAS INSTRUCCIONES

Los computadores son sistemas **síncronos** que **están gobernados** por una **señal** de reloj. Se denomina **operación básica** a las operaciones de transferencia que se realizan en el interior de la CPU. Estas operaciones básicas tienen una duración mínima determinada por el tiempo que transcurre entre 2 flancos consecutivos de la señal de reloj. El **computador** para ejecutar una determinada **instrucción** realiza una serie de operaciones **básicas** siguiendo un determinado orden. En un mismo periodo de la señal de reloj **pueden realizarse** varias operaciones básicas en paralelo, si éstas no **interfieren** al usar un mismo bus o algún elemento interno del procesador.

Las instrucciones típicas que ejecuta un computador suelen tener un periodo de ejecución dividido en cuatro fases. En general cada fase durará más de un periodo de la señal de reloj, realizándose en cada fase una serie de operaciones básicas que pueden ejecutarse en paralelo o en serie. Las fases que componen una instrucción son:

- **Lectura de la instrucción (fetch):** Ocurre al comienzo de la ejecución de cada instrucción y ejecuta la búsqueda de la instrucción en memoria. El primer paso en esta fase consiste en cargar el MAR con el contenido del registro PC. **Posteriormente** el **procesador** tiene que activar las señales de lectura de la memoria, guardar el dato en el registro MDR y por último transferirlo al registro IR. Este proceso de lectura se **realizará** en el computador ejemplo dos veces debido a que las instrucciones son de 16 bits, y los **buses** externos de datos y la memoria se consideran con una longitud de palabra de 8 **bits**.
- **Decodificación de la instrucción y lectura de los operandos:** El siguiente paso es la **decodificación** de la instrucción en la unidad de control. Jun-

to con la **decodificación** se tienen que buscar los operandos que utiliza la instrucción. Como se explicó en las secciones anteriores los modos de **direcciónamiento** indican la ubicación del operando, bien sea en memoria, en la **propia** instrucción o en algún registro del procesador. En el procesador ejemplo **sólo** se considera un modo de direcciónamiento en el cual la instrucción tiene un campo que contiene la dirección del dato situado en memoria. En las operaciones que necesiten dos datos el segundo se supondrá siempre contenido en el registro AC.

- **Ejecución de la operación:** En esta fase la UC activa las señales de control de los registros internos, para transferir la **información** necesaria entre ellos y **realizar** las operaciones adecuadas con la **ALU**. Las 2 fases anteriores realizan la misma secuencia de órdenes básicas, pero en esta fase por el contrario, esta secuencia depende de la **instrucción** que se esté **ejecutando**.
- **Almacenamiento de los resultados:** En esta fase se almacena en el lugar adecuado, dependiendo del modo de **direcciónamiento** empleado en la instrucción, el dato que resulta de la ejecución de la fase anterior.

Es posible que algunas instrucciones simples no necesiten todas las fases, **aunque** la de lectura y la de **ejecución** siempre deben estar presentes.

A continuación, se van mostrar algunos ejemplos de ejecución de instrucciones. Para ello se **utilizarán** cronogramas que representan el estado de las señales de control en cada uno de los ciclos que dura la ejecución de la instrucción.

2.6.1 Cronograma de ejecución de la instrucción ADD #456

Esta instrucción **realiza** la suma de sus operandos y el resultado lo almacena en el registro AC. El **primer** operando está situado en el registro AC y el segundo se encuentra en la posición de memoria que especifica la propia **instrucción**.

$$AC \leftarrow AC + M(456) \quad (2.1)$$

La instrucción consta de un total de 16 bits, organizados tal como se indica

en la figura 2.11. Como la longitud de palabra del bus de datos es de 8 bits, se necesitan 2 **lecturas** de memoria para acceder a toda la **instrucción**. En el caso de tener un bus de datos de 16 bits y un sistema de memoria con esta longitud de palabra solamente se necesitaría una lectura.

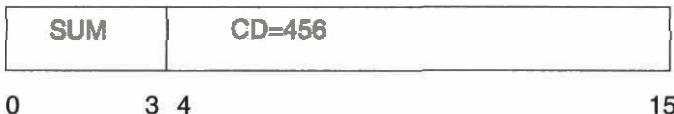


Figura 2.11: Formato de la instrucción ADD #456

Para empezar a ejecutar la instrucción se va a suponer que inicialmente el contador de programa (PC) contiene la dirección de dicha instrucción. La ejecución se divide en 4 fases. Cada una de ellas tendrá un determinado **número** de periodos de reloj y en cada una de ellas se realizarán unas determinadas operaciones básicas mediante la activación de la correspondientes señales de control.

1. **$MAR \leftarrow PC$.** Se transfiere **a** el contenido del registro PC (que es la dirección del primer byte de la instrucción), al registro de **direcciones MAR**.
2. **$PC \leftarrow PC + 1$.** Se incrementa el contenido del registro PC, para apuntar a la siguiente posición de memoria.
3. **$MDR \leftarrow M(MAR)$.** Se ejecuta un ciclo de lectura de memoria principal, cargando el registro MDR con la parte baja de la instrucción.
4. **$IR \leftarrow MDR$.** Se transfieren los 8 bits de la parte baja de la instrucción contenidos en el MDR a los 8 bits menos significativos del registro de instrucción IR. Igualmente se podrían considerar arquitecturas donde el primer byte leído fuera la parte alta de la instrucción, este primer byte iría a los bits **más** significativos del registro IR. Al primer tipo de ordenación de los datos en memoria en el que la parte menos significativa de la instrucción o del dato ocupa **posiciones de memoria** más bajas se le denomina **little-endian**. Por el contrario, cuando es la parte más **significativa** la que ocupa posiciones **más** bajas la ordenación **se denomina big-endian**.
5. **$MAR \leftarrow PC$.** Se transfiere el contenido del registro PC (que es la dirección del segundo byte de la **instrucción**), al registro de direcciones MAR.
6. **$PC \leftarrow PC + 1$.** Se **incrementa** el contenido del registro PC para apuntar a la siguiente instrucción.

7. $MDR \leftarrow M(MAR)$. Se ejecuta un ciclo de **lectura** de **memoria principal**, cargando en el registro MDR la parte alta de la instrucción.
8. $IR \leftarrow MDR$. Desde el registro MDR se **transfieren** los 8 bits a los bits $IR[8..15]$ del registro de **instrucción IR**.
9. $MAR \leftarrow IR[4..15]$. Se transfiere la **dirección** del dato al registro MAR, para iniciar la **lectura** del mismo en la **memoria**.
10. $MDR \leftarrow M(MAR)$. Se realiza *la* lectura de la posición de memoria cuya dirección está contenida en el campo de **dirección** del registro IR, en los bits $IR[4..15]$.
11. $AC \leftarrow AC + MDR$. Una vez capturado el **dato** en el registro MDR, se conecta el registro con la entrada B de la ALU mediante la señal de **habilitación** de la salida y se activan las señales de **selección de operación** OP con el **código** de la suma. Cuando **termina** la **operación de suma**, que supondremos que tendrá un retraso de medio ciclo de reloj, se activa la **señal de carga** del registro AC.

La **activación** de todas las **señales** se representa de **forma gráfica** en el **cronograma** de la figura 2.12.

Fase de búsqueda de la instrucción

En primer lugar se realiza la transferencia $MAR \leftarrow PC$. Para ello se activa la señal (**PC-E**) que habilita la **salida triestado** del registro y permite que el contenido del registro PC salga al bus de direcciones **interno** de la CPU. A continuación la UC activa la señal (MAR-C) **sincronizada** con el flanco de subida de la señal de reloj, que carga el registro **MAR** con el valor transmitido por el bus de direcciones interno. Todo esto se realiza en el primer periodo de la **señal** de reloj.

Durante el periodo 2 se completa el acceso a memoria. Primero se activan las **señales CS** y **RD** para realizar el ciclo de **lectura**. Pasado el tiempo de acceso de la memoria, los datos ya **están** estables en el bus de datos **externo** y se procede a cargar el contenido de la celda de memoria leída en el registro **MDR**, activando la señal (**MDR-CE**) en el último semiperiodo de la **señal** de reloj. En

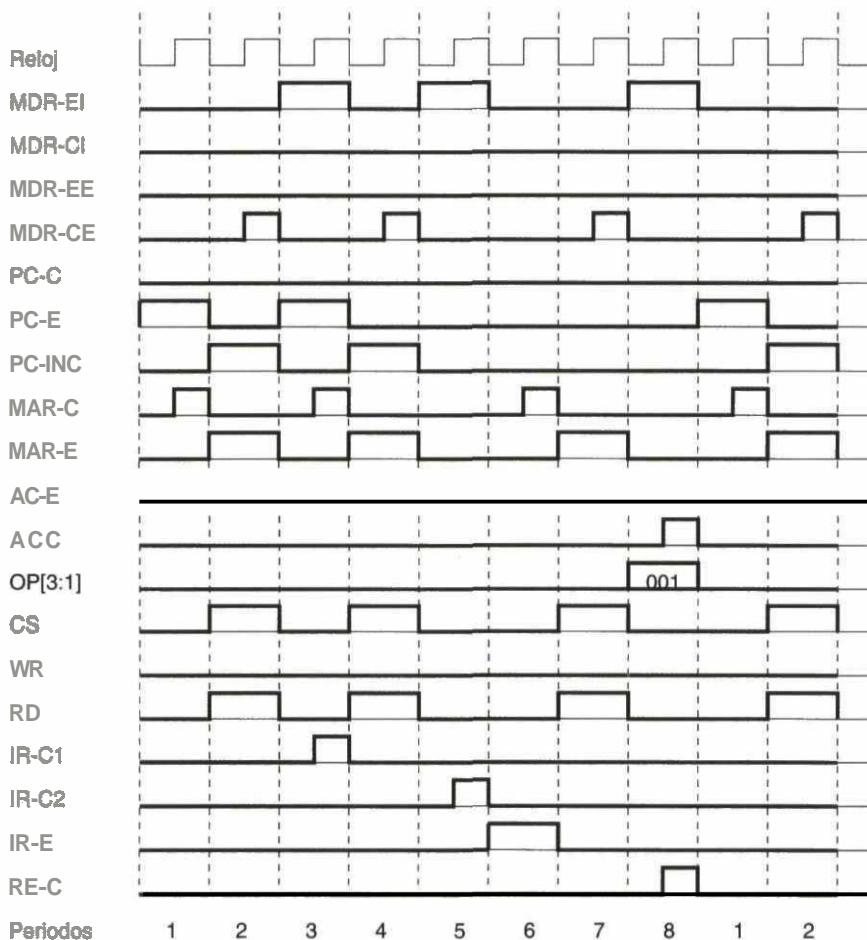


Figura 2.12: Cronograma de la instrucción ADD #456

este periodo también se activa la señal (PC-INC) de incremento del registro PC, para que apunte a la siguiente posición de memoria que contiene la parte alta de la **instrucción**.

En el periodo 3, la **parte** menos significativa de la **instrucción** contenida en el registro MDR se transfiere al IR. Para ello se activan la señales (MDR-EI), para que los datos del registro **MDR** salgan al bus interno del **procesador**, y la señal (**IR-C1**) para cargar los 8 bits más bajos del registro **IR**. Además se realiza la transferencia **MAR** \leftarrow **PC**, para comenzar un nuevo ciclo de lectura y acceder a la parte **alta** de la instrucción.

En el periodo 4 **se** repite la secuencia de **señales** del periodo 2, leyendo de memoria la **parte** más significativa de la instrucción e incrementando de nuevo el PC.

En el **último** periodo de la fase de búsqueda, se carga en en el registro IR la parte alta de la instrucción contenida en el registro **MDR**. Se activa la señal (MDR-EI) para **transferir** los datos al bus interno y la señal de carga (IR-C2). También se produce la **decodificación** de la **instrucción** en la UC, puesto que los 4 bits del código de operación que residen **en** la parte **más** baja de la instrucción ya están **almacenados** en el registro IR.

Estos 5 primeros periodos de reloj son comunes a todas las instrucciones, puesto que la fase de **búsqueda** no depende de la **instrucción** leída.

Fase de búsqueda de **operандos**

En el momento **en** que la UC decodifica la instrucción cargada **en** el registro IR, comienza la búsqueda de operandos. En esta fase se almacena en el registro MDR el operando que se encuentra en la **posición** de memoria, cuya dirección está contenida en **IR[15..4]**.

En el primer periodo de esta fase de la instrucción, se transfiere la dirección desde los bits (**IR[15..4]**) hasta el registro MAR, (**MAR** \leftarrow **IR[15..4]**). Para ello se habilita la salida de estos bits al bus de direcciones interno del **procesador** con la **señal** IR-E, y a continuación se capturan con la señal de carga (MAR-C).

En el siguiente **periodo** **se** activan las señales de lectura de la memoria CS y RD. Una vez transcurrido el tiempo de acceso de la memoria, el dato se carga en el MDR activando la señal (MDR-CE). Con este periodo termina la fase de captura de operandos.

Fase de ejecución y almacenamiento del resultado

Estas dos fases se realizan en el mismo periodo de la señal de reloj. Para realizar la **instrucción** $AC \leftarrow AC + MAR$ en el ciclo 8 la CPU tiene que:

1. Conectar el registro **MDR** a la entrada **B** de la **ALU** activando la señal **MDR-EI**.
2. Indicar a la **ALU** la operación a realizar mediante la activación de las señales **OP**.
3. Por último, sincronizada con el flanco de subida de la señal de reloj, activar la señal de carga del acumulador (**AC-C**).

Tras la ejecución de la instrucción se actualiza el registro de estado que contiene los bits de condición. Para ello, se activa la señal de flanco (**E-C**) en el instante en que la **ALU** finaliza la operación de suma, en el periodo 8.

2.6.2 Ejecución de la instrucción SW #456

Esta instrucción realiza una escritura del dato contenido en el acumuladoren la posición de memoria principal indicada por el campo **CD** de la instrucción.

$$M(456) \leftarrow AC \quad (2.2)$$

La instrucción **está** compuesta por los 4 bits del código de instrucción, y los 12 bits necesarios para indicar la dirección de memoria donde se guarda el **AC**. El cronograma que describe la ejecución de esta instrucción se muestra en la figura 2.13

La fase de búsqueda de la instrucción es similar a la del ejemplo anterior. Sin embargo, en la presente instrucción no hay fase de búsqueda de operandos ni almacenamiento de resultados, por lo que directamente se pasa a la fase de ejecución. La ejecución se realiza en los ciclos 6 y 7.

En el ciclo 6 se lleva la dirección de memoria desde los bits (**IR[4..15]**) hasta el registro **MAR** y el contenido del **AC** al registro **MDR**. En el siguiente periodo se realiza la escritura de memoria activando (**CS**, **WR**, **MDR-EE** y **MAR-E**). Los siguientes ciclos representados en el cronograma pertenecen al comienzo de la siguiente instrucción.

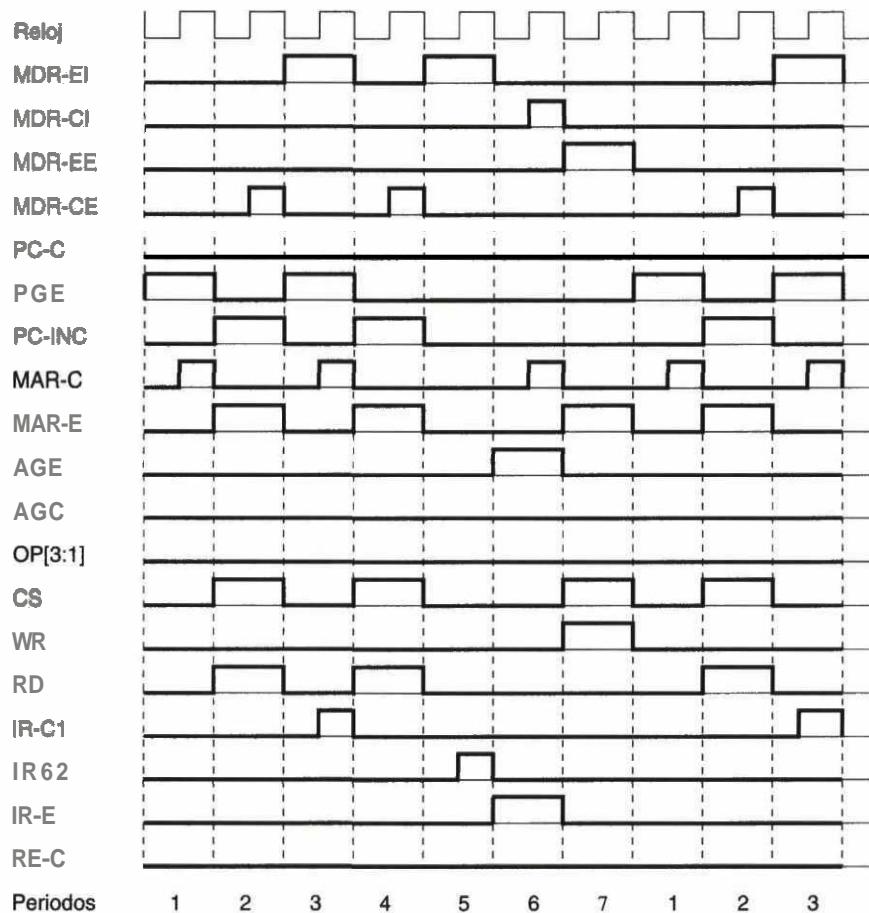


Figura 2.13: Cronograma de la instrucción SW #456

2.6.3 Ejecución de la instrucción JMZ #456

Esta instrucción realiza una bifurcación condicional utilizando direccionamiento directo. Se salta a la dirección contenida en el código de dirección si el indicador de cero del registro de estado está activo.

$$\left\{ \begin{array}{l} \text{Si } Z=1 \text{ entonces } PC \leftarrow \#456 \\ \text{Si } Z=0 \text{ entonces } PC \text{ no se } ifica. \end{array} \right. \quad (2.3)$$

El cronograma de ejecución de esta instrucción se muestra en la figura 2.14. **Después** de capturar la instrucción en el registro **IR**, comienza en el periodo 6 la fase de ejecución de la instrucción. Si no se cumple la condición de salto ($Z=0$) se inicia la ejecución de una nueva **instrucción**, ya que el **PC** ya se ha actualizado. Sin embargo si se cumple la **condición** se realiza la carga del **PC** con el valor efectivo #456 en el periodo 6, y posteriormente se inicia la ejecución de una nueva instrucción.

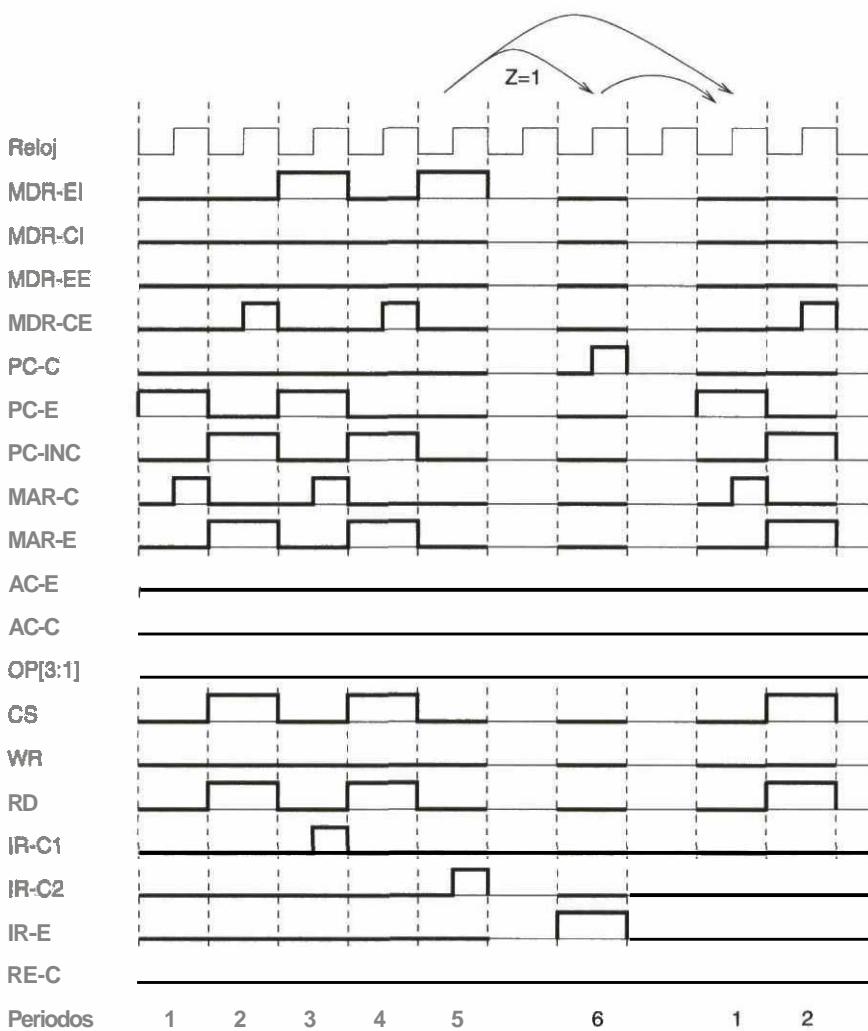


Figura 2.14: Cronograma de la instrucción JMZ #456

2.7 SUBRUTINAS

Los programas que ejecuta la **CPU** están formados por una secuencia de instrucciones. Para **estructurarlos** se dividen en módulos independientes y separados lógicamente denominados subnitinas. La utilización de subnitinas permite que los programas sean más fáciles de depurar, más **portables** y se facilita la **reutilización** de código. Con esta división la estructura de un programa está formada por un módulo principal, que realiza llamadas a diversas subnitinas. Cuando se produce un salto a subnitina el procesador detiene la ejecución del **módulo** principal y comienza la ejecución de la **subrutina**. Ésta **realizará** una **determinada** función y al finalizar el procesador vuelve a retomar la ejecución del módulo principal.

Cuando se salta a una subnitina el procesador debe almacenar en algún lugar el valor actual de PC, que servirá como dirección de retorno cuando se termine la ejecución de la subnitina. En una primera aproximación se utiliza alguna posición de memoria o algún registro interno para almacenar el PC. El problema que presenta esta técnica aparece cuando se realiza una segunda llamada a subnitina desde la primera subnitina, ya que se pierde el valor de retorno para la primera llamada. Una segunda solución para almacenar las direcciones de retorno es utilizar posiciones de memoria distintas para cada **llamada**. En este caso se puede usar la primera posición de memoria que ocupa la subnitina como espacio para guardar la dirección de retorno. Así, cada subnitina posee una posición distinta para guardar el PC. Pero de nuevo se presenta el mismo problema cuando se realizan llamadas recursivas. Para solucionarlo, es necesario que en cada llamada a subnitina se utilicen posiciones de memoria distintas para almacenar las direcciones de vuelta. La implementación de este método utiliza una pila, que apunta a nuevas posiciones de memoria a medida que se van almacenando datos.

Al saltar a una subnitina es normal que el programa principal tenga que pasar a la subnitina una **serie** de argumentos para que pueda realizar su función, y que ésta devuelva algún resultado. Estos **parámetros** pueden pasarse utilizando algún registro interno o alguna posición de memoria, pero en general se presentarán los mismos problemas que en el caso del almacenamiento del PC. El método más general de pasar los argumentos a la subnitina será también mediante la pila.

Para permitir la **utilización** de **subrutinas** en el **procesador** ejemplo, se van a implementar los elementos necesarios para gestionar una **pila**. Para ello se **tendrá** que dotar al **procesador** de un nuevo registro interno: el puntero a pila (SP). Este registro apunta a la siguiente posición **vacía** dentro de la pila. Despues de apilar un nuevo valor hay que decrementar el puntero y antes de desapilar hay que **incrementarlo**. De esta forma la pila irá creciendo hacia posiciones de memoria inferiores. También hay que ampliar el repertorio de instrucciones con cinco nuevas:

1. La instrucción **MSP CD** carga el SP con el valor contenido en el campo CD, esto **permite** situar la pila en la zona de memoria que se deseé.
2. La instrucción **PUSH R** apila el contenido del registro R en la posición apuntada en ese instante por el SP, **decrementándolo** posteriormente.
3. La **instrucción POP R incrementa** el SP y desapila un valor, **almacenándolo** en el registro R.
4. **Instrucción JSUB CD** es una instrucción de salto a **subrutina** que antes de modificar el **PC** apila su valor.
5. instrucción **RET** indica el final de la subrutina y desapila el valor inicial que contenía el PC antes de haberse producido la llamada.

En el esquema de la figura 2.15 se pueden observar los valores almacenados en la pila cuando se produce una llamada a **subrutina**.

A continuación se mostrarán los cambios que se deben realizar en la **estructura** del computador ejemplo para implementar estas nuevas instrucciones. En primer lugar aparece el nuevo registro SP con las señales de carga **SP-C**, de salida al bus de direcciones **SP-E**, incremento **SP-INC** y decremento **SP-DEC**. También hay que cambiar la **estructura** del registro de estado para permitir que se pueda cargar y acceder al bus de datos interno. La señal **RE-E** sirve para habilitar la salida al bus y la señal **RE-CB**, para cargar los datos desde el bus. Igualmente hay que permitir el acceso desde el **PC** al bus de datos. Las señales de habilitación, (**PC-EB1** y **PC-EB2**), activan la salida de la parte alta y baja del **PC**, y las dos señales de carga, (**PC-CB1** y **PC-CB2**), capturan datos del bus. Como el **PC** tiene 12 bits, para **almacenarlo** en memoria se divide en dos partes de 6 bits, siendo necesario realizar dos escrituras. En **ellas** dos de los ocho bits

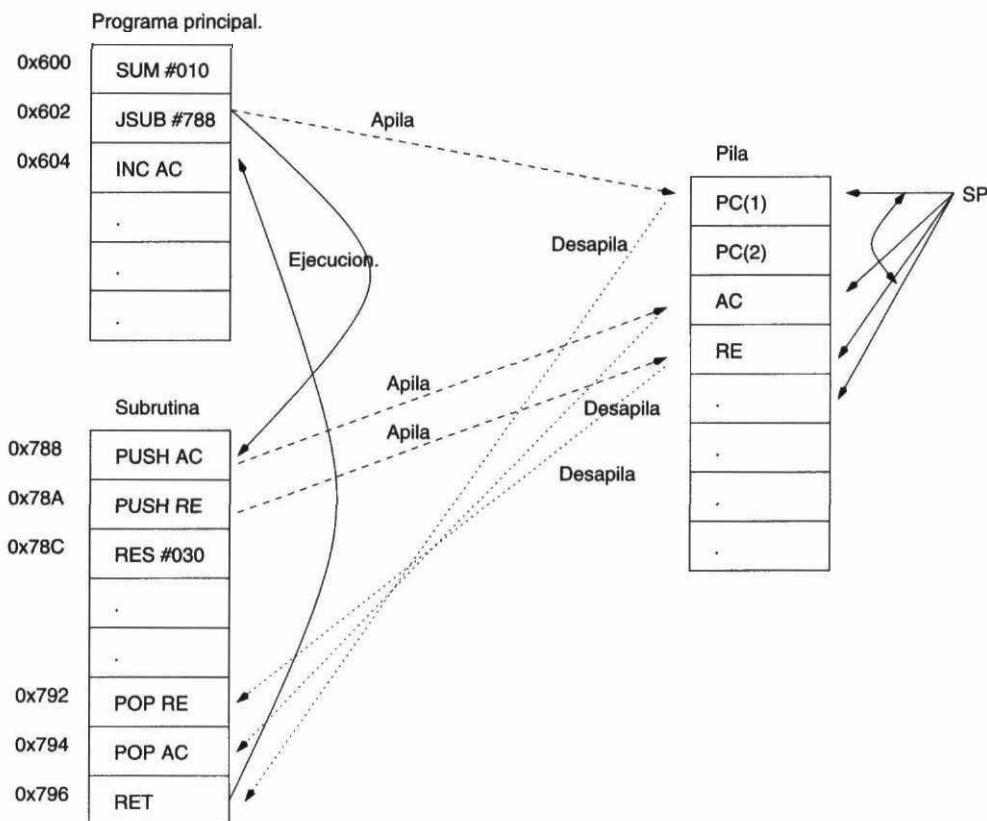


Figura 2.15: Llamada a subrutina

que se pueden almacenar tendrán un valor fijado por la UC, ya que no se utilizan, y el resto de los bits se corresponden con los del PC. Cuando se salva el PC en la pila cada uno de los dos bytes utilizados están en posiciones consecutivas de memoria. Al desapilarlos se accede en orden inverso, primero se accede al byte que fue apilado en segundo lugar, ya que la pila accede primero a la última posición utilizada, y a continuación se accede al primer byte apilado.

A parte de los cambios realizados en los registros internos para implementar estas nuevas instrucciones, también hay que modificar la UC y cambiar el formato de las instrucciones. Con la incorporación de las 5 nuevas instrucciones existen un total de 21, que supera el máximo de 16 que se pueden diferenciar con los 4 bits reservados para el código de operación. En vez de modificar este campo se ha optado por la solución de codificar todas las instrucciones que

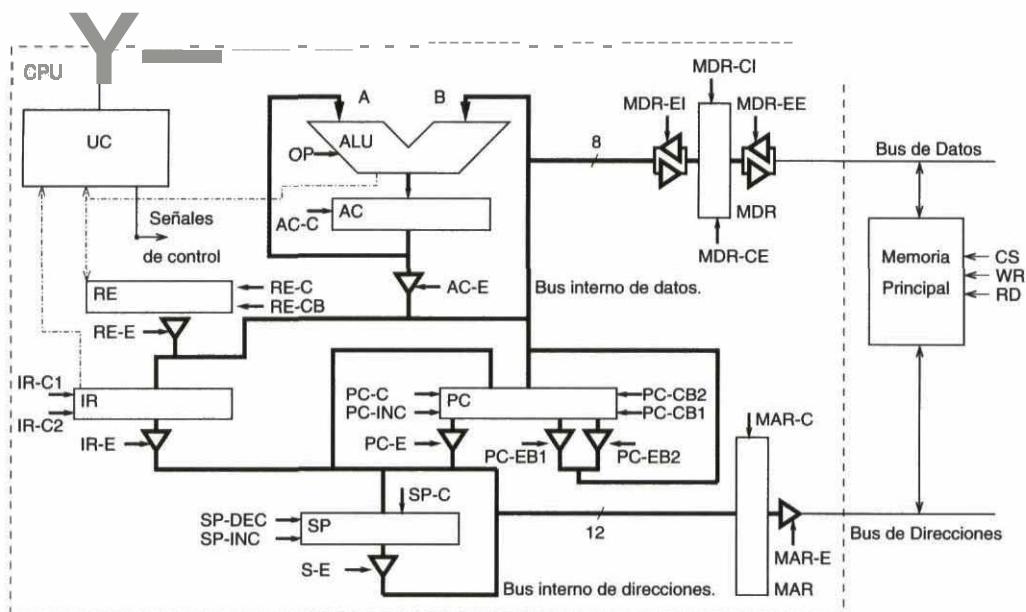


Figura 2.16: Esquema de computador elemental

no necesiten **campo de operandos** con un mismo código de operación de 4 bits. Posteriormente, para que la CPU pueda distinguirlas se utiliza un campo de expansión, tal como indica la figura 2.17.

2.8 EXCEPCIONES

Las excepciones son eventos que rompen la secuencia normal de ejecución del programa en curso, dejando el control a alguna **subrutina específica** de tratamiento de dicho evento. Las causas de una excepción pueden ser las producidas por algún acontecimiento anormal en el **procesador** durante la ejecución de una **instrucción**, por la ejecución de alguna instrucción **específica**, denominada *trap*, o como consecuencia de algún acontecimiento externo, denominada **interrupción**. Los tipos más comunes de excepciones son:

- Se ha producido un error de cálculo **aritmético** como, por ejemplo, una división por cero o un desbordamiento (*overflow*). (Excepción).
- Mediante una **instrucción** específica. Hay instrucciones cuya ejecución es una excepción. (Tmp).

Instrucciones Formato

SUM CD	0000	CD
RES DC	0001	CD
AND CD	0010	CD
	0011	CD
LEE CD	0100	CD
ESC CD	0101	CD
SAL CD	0110	CD
SAZ CD	0111	CD
SAN CD	1000	CD
SAC CD	1001	CD
SAO CD	1010	CD
SAP CD	1011	CD
SACP CD	1100	CD
MSP CD	1101	CD
JSUB CD	1110	CD
XXX	1111	CD

Instrucciones con expansión del código de instrucción. Formato.

INCAC	1111	000
NOP	1111	001
STP	1111	010
PUSH R	1111	011 0/1
POP R	1111	100 0/1
RET	1111	101

Figura 2.17: Formato del repertorio de instrucciones

- Se lee un código de operación inexistente, la instrucción no existe. El código de operación leído no se corresponde con ninguna instrucción. (Excepción).
- Un periférico requiere la atención de la CPU. Activación de un pin de entrada al procesador. (Interrupción).

Las excepciones están priorizadas, cuando se producen dos o más excepciones distintas en el mismo instante de tiempo se atenderá primero **aquella** a la que se le haya dado mayor prioridad. Dentro de la CPU debe existir **algún registro**

interno en el que se actualice la prioridad de cada excepción. Las interrupciones pueden ser enmascarables, cuando pueden **deshabilitarse** por software, o no enmascarables, cuando no es posible su **deshabilitación**. **Las** no enmascarables son indicativas de algún error **catastrófico** como es la caída de la alimentación o un error en **la memoria**, y tienen **la** mayor prioridad. La submtina de tratamiento de las interrupciones no enmascarables está situada en alguna posición fija en memoria, lo que se denomina **autovectorizada**, teniendo el procesador almacenada su localización de antemano. Las enmascarables, o interrupciones de usuario, necesitan un ciclo de reconocimiento durante el cual el **procesador** recoge el vector de interrupción, que indica la posición de memoria donde se encuentra la **subrutina** de **tratamiento** de la interrupción.

El **procesador**, **para** detectar la activación de una **excepción**, **comprueba** todos **los indicadores (flags)** de activación de excepciones, **almacenados** en algún registro interno especial dedicado a tal fin. Dependiendo del tipo de excepción la comprobación de su activación se produce en un determinado instante.

1. Final del ciclo de reloj: Se compmeban las interrupciones no **enmascara-**
bles.
2. Final del ciclo de bus: instrucción ilegal, violación de privilegios (excep-
ciones).
3. Final de la ejecución de instrucción: Interrupción enmascarable.
4. Durante la ejecución de la **instrucción**: División por cero (excepción),
instrucción de excepción (**trap**).

Una vez detectada la excepción, lo primero que se produce es el almac-
namiento en la pila del valor actual del contador de programa y del valor del
registro de estado. Dependiendo de si la interrupción es **autovectorizada** o no,
puede ser necesario un ciclo de reconocimiento para obtener la posición de me-
moria en **la cual** está situada **la subrutina**. A continuación se realiza el salto a
esta **subrutina** de tratamiento, la cual debe finalizar con una instrucción RETI,
que realiza el **retorno desapilando** los valores del PC y del registro de estado.
Además esta instrucción habilita el procesamiento de nuevas interrupciones que
tengan una prioridad menor y **estén** a la espera de ser ejecutadas.

2.9.1 Unidad de control cableada

Este tipo de unidad de control utiliza lógica combinacional (**lógica cableada** equivalente a puertas lógicas), para implementar las funciones lógicas de las señales de **control**. Un esquema de la UC cableada del computador elemental se muestra en la figura 2.18. En el esquema se puede apreciar cómo la UC es **básicamente un** circuito combinacional, pues la salida (señales de control) depende únicamente de las entradas (código de operación, indicadores de estado, registro contador e **interrupción**). Cada una de las salidas de la figura 2.18 es una señal de control del computador y **será** una función lógica de las entradas actuales.

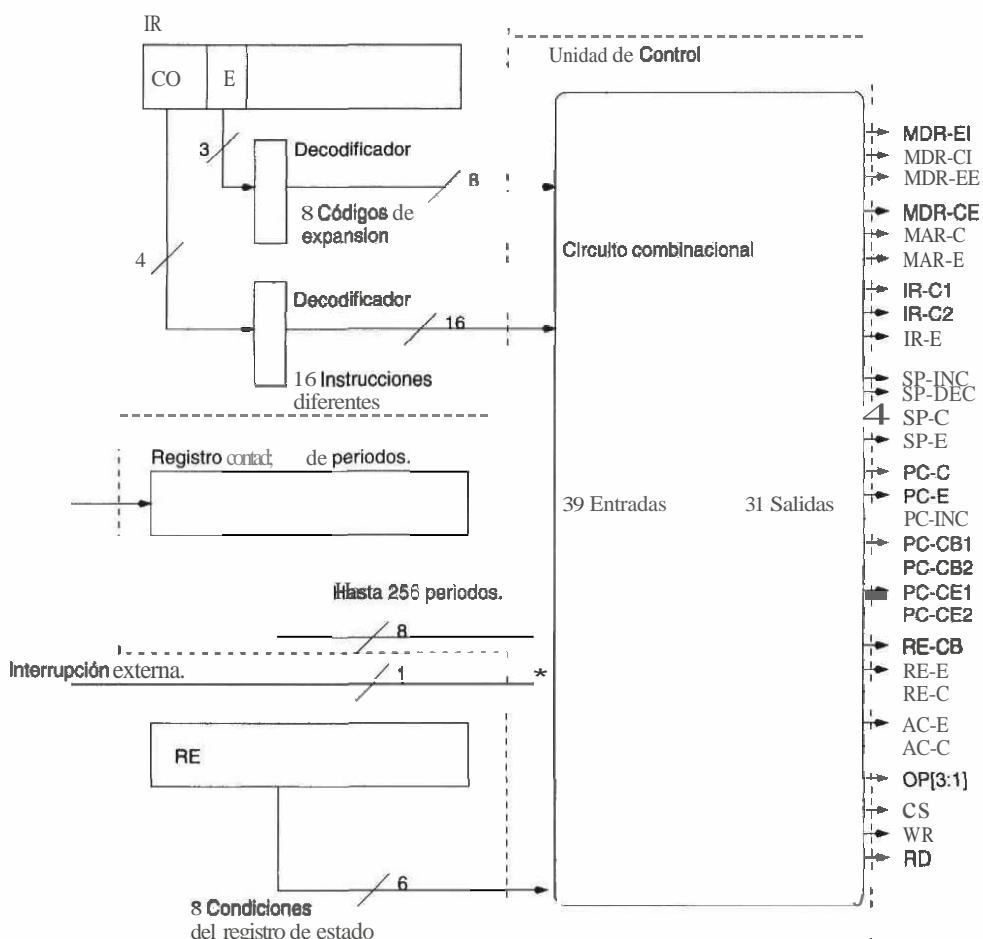


Figura 2.18: UC cableada

Para la **implementación** de interrupciones de usuario en el procesador ejemplo, es necesario dotarle de la **instrucción RETI** y de un pin adicional de entrada, así como modificar la unidad de control para que compruebe la activación de las excepciones. También habrá que dotar al procesador de un registro interno donde se almacenen los valores de prioridad de cada excepción. Por **sencillez** consideraremos que todas las interrupciones son **autovectorizadas**, con un **vector** de interrupción fijo y almacenado en la unidad de control. De esta forma no **será** necesario un ciclo de reconocimiento de interrupción para que el dispositivo solicitante **suministre** su vector.

2.9 DISEÑO DE LA UNIDAD DE CONTROL

La Unidad de Control es la encargada de generar todas las **señales** de control en cada uno de los periodos que forman el ciclo de ejecución de una instrucción. Para generar estas señales la unidad de control debe tener entradas que le **informen** del estado actual del procesador, de **cuál** es la instrucción a ejecutar e información sobre posibles eventos (excepciones). Además, la UC debe contar los periodos de reloj, para **realizar** el secuenciamiento de activación de las señales de control dentro de la **ejecución** de una instrucción. Esta cuenta la realiza utilizando un registro interno que incrementa en cada pulso de la **señal** de reloj. Cuando finaliza la ejecución de una **instrucción** la cuenta se pone a cero.

A partir de la secuencia de señales de entrada, la unidad de control genera las señales de control necesarias para la ejecución de la instrucción. Se puede considerar la unidad de control como un circuito digital que a partir de las entradas genera una secuencia de señales de salida que se corresponden con las señales de control. La secuencia y **definición** de **todas** estas **señales** debe realizarse en el momento de **diseño** de la CPU, utilizando las diversas **metodologías** que existen a la hora de **diseñar** un circuito digital. Existen dos formas fundamentales de abordar el diseño de una unidad de control: unidad de **control cableada** y **unidad de control microprogramada**.

Las principales características de una UC **genérica implementada** con lógica cableada se pueden resumir en:

- La realización de estas funciones **lógicas** es muy costosa. Aunque en la **actualidad**, con la utilización de lenguajes de especificación **hardware** de alto nivel, la tarea de **implementación** a nivel de función lógica está automatizada.
- Cualquier pequeña modificación (**por ejemplo, añadir** una instrucción) **requiere un rediseño prácticamente completo**. Este esfuerzo de **rediseño** se ha reducido **drásticamente** con la **utilización** de las técnicas de CAD que automatizan la generación del hardware.
- Presenta la ventaja de la rapidez. Se pueden **optimizar** los retrasos producidos en las puertas que forman el circuito para que la unidad funcione con **señales** de reloj de alta frecuencia.
- Se suele utilizar unidad de control cableada cuando se requiere una gran velocidad o el diseño es muy sencillo.

2.9.2 Unidad de control microprogramada

La unidad de control realizada con lógica cableada presenta como inconvenientes fundamentales la gran cantidad de funciones hardware necesarias para su **implementación**. Como alternativa a la unidad de control cableada se presenta la **unidad de control microprogramada**.

Este tipo de unidad contiene una memoria, llamada **memoria de control**, que almacena todos los valores que deben tomar las señales de control en cada periodo de reloj dentro de la ejecución de una instrucción. Al conjunto de los valores de todas las señales de control en un periodo determinado se le denomina **microinstrucción**. Cada una de ellas se almacena en una posición dentro de la memoria de control interna a la UC. Dentro de la UC hay una **circuitería** que secuencialmente accede a la memoria de control para leer los valores adecuados de las señales de control y activarlas en consecuencia. En la figura 2.19 se muestra un ejemplo de microprograma.

Contenido de la memoria de control.

MDR-EI	0	0	1	0	1	0	0	1
MDR-CI	0	0	0	0	0	0	0	0
MDR-EE	0	0	0	0	0	0	0	0
MDR-CE	0	1	0	1	0	0	1	0
PC-C	0	0	0	0	0	0	0	0
PGE	1	0	1	0	0	0	0	0
PC-INC	0	1	0	1	0	0	0	0
MAR-C	1	0	1	0	0	1	0	0
MAR-E	0	1	0	1	0	0	1	0
AGE	0	0	0	0	0	0	0	0
ACG	0	0	0	0	0	0	0	1
OP[3:1]	000	000	000	000	000	000	000	001
CS	0	1	0	1	0	0	1	0
WR	0	0	0	0	0	0	0	0
RD	0	1	0	1	0	0	1	0
IR-C1	0	0	1	0	0	0	0	0
IR-C2	0	0	0	0	1	0	0	0
IR-E	0	0	0	0	0	1	0	0
RE-C	0	0	0	0	0	0	0	1
	1	2	3	4	5	6	7	8

Microinstrucciones

Figura 2.19: Microprograma de la instrucción ADD #456

Para realizar una **instrucción** completa la UC debe ejecutar varias **microinstrucciones**. El conjunto de todas las **microinstrucciones** que forman una **instrucción** máquina recibe el nombre de **microprograma**.

Formato de las microinstrucciones

El **formato** de las **microinstrucciones**, al igual que el de las instrucciones máquina, indica cuántos **bits** forman una **microinstrucción** y qué información contiene cada uno de ellos. Las **microinstrucciones** pueden utilizar muchos **formatos** diferentes. La **elección** de este formato afecta y se ve afectado por todos los elementos que componen la CPU. Por ejemplo, el número de **señales** de

control que es necesario activar para cada **microinstrucción** limita el **número** mínimo de bits que forman una **microinstrucción**, que a su vez limita la longitud de palabra de la memoria de control que se utiliza para **almacenarlas**.

En una primera **aproximación** se puede proponer un formato en el cual cada bit de la **microinstrucción** se corresponde con una señal de control. Este tipo de formato se denomina **microprogramación horizontal**, y en **él** no se necesita una **traducción** para generar las señales de control, sino que, por el contrario, presenta la desventaja de que se trata de un formato con una gran cantidad de bits, por lo que requiere mucha más memoria para implementarlo.

Para limitar el tamaño de la memoria se pueden agrupar las señales de control en campos. Cada uno de ellos contiene bien aquellas señales de control que pertenezcan al mismo elemento interno de la CPU, o bien aquellas que no se pueden habilitar de forma **simultánea**. Ejemplos de señales que se pueden agrupar en un campo son las de **habilitación** de salida al bus interno del **procesador**, o las de gobierno de la ALU. El valor de cada una de las señales de control pertenecientes a un campo se obtiene al **decodificar** el valor correspondiente contenido en la **microinstrucción**. Este tipo de formato se denomina **microprogramación vertical**, y tiene la ventaja de la reducción del número de bits de la **microinstrucción**, y por tanto de la memoria de control. La desventaja que presenta es que es necesaria la **decodificación** de los valores contenidos en la **microinstrucción**. El grado de **codificación** puede ser mayor o menor. Una formato presenta un alto grado de **codificación** cuando las señales de control **están** todas agrupadas en un número reducido de campos. El formato es más vertical cuanto mayor sea el grado de **codificación**, hasta llegar al extremo de la codificación **total**. En la **figura 2.20** podemos ver la **agrupación** en campos de las señales de control del computador elemental.

Además de los campos que agrupan **señales** de control debe **definirse** un **campo de secuenciamiento**. En este campo se codifica **cuál** es la **próxima microinstrucción** que se va a leer, y puede presentar tres casos:

- La siguiente **microinstrucción** **está** contenida en la siguiente **posición** de memoria.
- La siguiente **microinstrucción** pertenece al comienzo de una nueva instrucción.

Campos	
Control RAM.	CS WR RD MAR-E MDR-EE MDR-CE
Acceso al bus da direcciones	SP-E IR-E PC-E
Carga desde el bus de direcciones	SP-C PC-C MAR-C
Carga desde el bus de datos	IR-CI IR-C2 RE-CB PC-CB1 PC-CB2 MDR-CI
Acceso al bus de datos	RE-E AC-E MDR-EI PC-CE1 PGCEP
Incremento de registros	PC-INC SP-INC
Control ALU	AC-C OP[3:1] RE-C

Figura 2.20: *Agrupación en campos de las señales de control del computador elemental*

- Es necesario un salto a una posición de memoria sin **determinar**.

Hay diseños donde el secuenciamiento de las microinstrucciones se realiza sólo mediante los dos primeros casos. En este **tipo de secuenciamiento** todas las microinstrucciones que forman un **microprograma** deben estar ordenadas una a continuación de la otra en la memoria de control. Otra condición necesaria es que el código de operación de la **instrucción** contenida en el **IR** debe indicar la posición de la primera **microinstrucción** de su microprograma **correspondiente**. Con estos dos requisitos el campo de secuenciamiento de cada **microinstrucción** se reduce a un bit, que indica cuándo hay que acceder a la posición de **memo-**

ria siguiente, porque todavía no se ha completado el **microprograma**, o por el contrario cuándo hay que acceder a la posición indicada por el **código de operación** del IR, para comenzar la **ejecución** de una nueva instrucción. Este tipo de secuenciamiento se denomina **implícito**, y presenta la limitación de que no es posible **implementar microprogramas** con **saltos internos** que puedan seguir una secuencia de **microinstrucciones** u otra dependiendo de alguna condición, como por ejemplo en el caso de las instrucciones de salto **condicional**.

El **secuenciamiento explícito**, por el contrario, permite los **tres** casos y, por tanto, como a priori no se conoce la **dirección**, es necesario codificar en el campo de **secuenciamiento** la dirección de la próxima **microinstrucción**. Con este modo ya se permiten saltos condicionales dentro de un **microprograma**, pero se aumenta la **longitud** de palabra del formato de las **microinstrucciones** y por tanto de la memoria de control.

Estructura de la UC **microprogramada**

La unidad de control **microprogramada** contiene un memoria de control en la que se almacenan las **microinstrucciones**. Además, en el caso de que éstas estén **codificadas**, debe contener una **circuitería** de decodificación de los campos y una unidad de **secuenciamiento** para obtener la dirección de la **próxima microinstrucción** a leer. En el caso de que **se permitan** saltos **condicionales**, el circuito debe considerar los bits correspondientes del registro de estado, para saltar a **una** posición u otra dependiendo de su valor. En la figura 2.21 se muestra el esquema de una **UC** microprogramada con **secuenciamiento explícito**.

Ventajas y **desventajas** de la **microprogramación**

La **microprogramación** presenta desventajas cuando el procesador que se diseña es muy simple o cuando se requieren unas prestaciones elevadas. En el primer caso, el **esfuerzo** en el diseño de una **UC microprogramada** puede ser excesivo si el procesador es relativamente simple (pocas instrucciones y poca circuitería **interna**) siendo más **sencillo** abordar el **diseño** con lógica **cableada**. Por el contrario, si el procesador es complejo y requiere altas prestaciones, puede **ocurrir** que la ejecución del **microcódigo** sea excesivamente lenta.

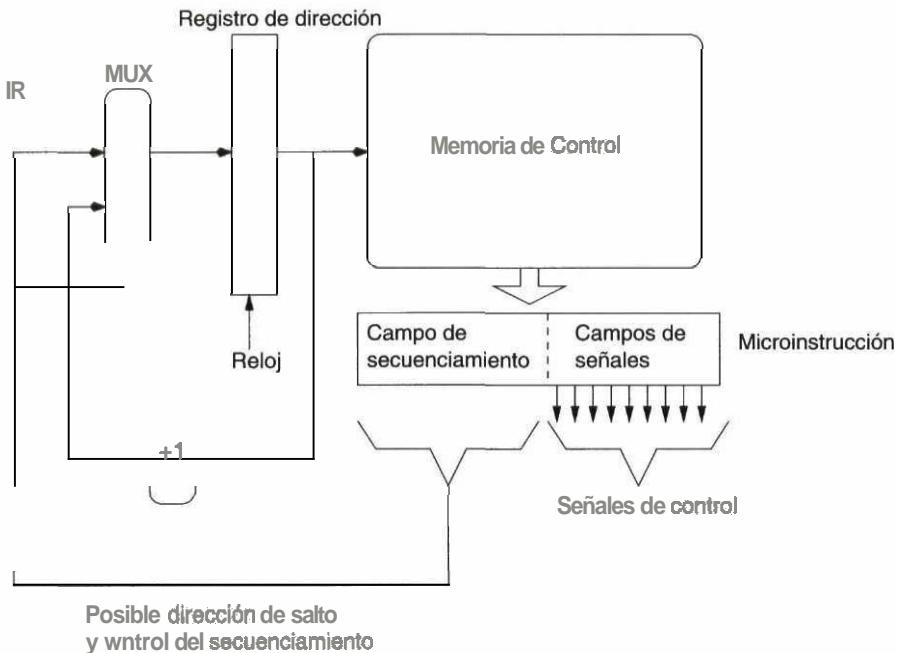


Figura 2.21: UC microprogramada

La **microprogramación** se basa en el hecho de que las **microinstrucciones** están almacenadas en un **memoria**, lo que presenta las siguientes ventajas:

- La UC **microprogramada** es más fácil de **reprogramar** o **rediseñar**, ya que sólo es necesario cambiar los valores de algunas posiciones de la memoria para corregirla.
- e Se pueden incluir, sin grandes dificultades **hardware**, instrucciones complejas que tengan muchos períodos de ejecución.
- e Permite **implementar** a este nivel algunas instrucciones complejas que son muy usadas por el computador, instrucciones de alto nivel que no es necesario dividir en **varias** instrucciones máquina.
- e Permite **implementar** procesadores con diversos juegos de instrucciones, ya que si la memoria que almacena el **microcódigo** es de tipo RAM se pueden cambiar los valores almacenados. Esto también permite copiar en un computador un juego de instrucciones de otro computador para ejecutar sus programas. es decir se pueden emular otros computadores.

- Las **microinstrucciones** permiten hacer **rutinas de diagnóstico** del computador mucho mejores **que** las hechas en código **máquina**. Es en este **nivel** (el **más bajo** del computador) donde se tiene un mayor control sobre los registros **y las** partes **internas** del computador.

2.10 EVOLUCIÓN DE LOS PROCESADORES

Durante el desarrollo de este capítulo el computador elemental diseñado ha servido como ejemplo para la explicación de muchos conceptos básicos de estructura de computadores. Hay que destacar que este computador tiene muy poca complejidad en comparación con los procesadores comerciales actuales, ya que el formato de las instrucciones y el número de instrucciones y de modos de direccionamiento es muy reducido. Los computadores reales presentan un número de instrucciones superior en algunos casos a las 400, así como múltiples modos de direccionamiento. Además el formato de las instrucciones no tiene por **qué** ser de 16 bits y regular, sino que dependiendo del procesador, este valor puede llegar a los 64 bits y ser diferente para distintas **instrucciones**. Por ello, se van a resumir ahora las arquitecturas de procesadores comerciales más utilizadas viendo su **evolución** y características. En el capítulo 12 se muestran adicionalmente aproximaciones distintas al modelo de máquina Von Newman presentado en este capítulo, entre ellas procesadores que usan la segmentación para aumentar las prestaciones.

2.10.1 Aumento de las prestaciones

El primer procesador comercial fabricado por **Intel** en 1971 tenía unos registros internos capaces de trabajar con datos (longitud de palabra) de hasta 4 bits, funcionaba con una frecuencia de reloj de 0.5 Mhz y podía acceder a 16 **Kbytes** posiciones de memoria (anchura del bus de direcciones). Desde entonces ha habido un incremento constante en las prestaciones de los **microprocesadores** como consecuencia de:

1. **Aumento** de la longitud de palabra del procesador: Al ser mayor el tamaño de los datos internos del procesador, se incrementa la velocidad de procesamiento al operar de forma paralela con un mayor número de **bits**.

2. Aumento de la **frecuencia** de reloj: Los **procesadores** son sistemas secuenciales **síncronos**, en los que, cuanto mayor es la frecuencia del reloj mayor es su velocidad de ejecución de instrucciones.
3. Aumento del nivel **de integración**: Aunque ya se ha mostrado que la arquitectura básica del procesador está formada por una unidad de control, la **ALU** y varios registros internos, se tiende a integrar junto con estas unidades elementos externos al procesador para aumentar el rendimiento del sistema. Ejemplos de estas unidades son las memorias **cache** de primer nivel y las unidades de gestión de memoria (**MMU**) .
4. Aumento de la **anchura** del **bus** externo: Ya que el procesador debe estar cargando continuamente datos e instrucciones de memoria, la anchura de **los buses** que conectan ambos elementos es fundamental. Cuanto mayor sea la anchura del bus, mayor será el número de datos e instrucciones a los que se **podrá** acceder en un solo ciclo, aumentando **así** la velocidad de acceso.
5. **Avances en la arquitectura:** Internamente el **microprocesador** puede ser paralelizado para ejecutar varias instrucciones a la vez. Estas técnicas reducen el número de ciclos de reloj necesario para ejecutar una instrucción. Ejemplos de estas arquitecturas son los procesadores segmentados.

Para medir la repercusión que estos avances tienen en los procesadores se debe considerar algún **método** para medir su rendimiento. Un método común de medida son los **MIPS** (millones de **instrucciones** ejecutadas por segundo) que un procesador alcanza para una determinada tarea. Se define **MIPS** como:

$$\text{MIPS} = F/CPI$$

Donde **F** es la **frecuencia** de reloj en **MHz** y **CPI** es la media de los ciclos de reloj que utilizan las instrucciones al ser ejecutadas. Es posible medir el rendimiento de forma equivalente mediante la utilización del tiempo **por** tarea, que mide el tiempo necesario para ejecutar una **determinada** tarea, sin incluir el tiempo de espera en la entrada y salida de datos. En la siguiente fórmula **N** es el número de instrucciones y **C** es el tiempo del periodo de la **señal** de reloj.

Procesador	Registros internos	Bus de datos/direc.	Reloj Max.
8086	16bits	16bits/20bits(1Mb) a 10Mhz	10MHz
80286	16bits	16bits/24bits(16Mb) a 20Mhz	20MHz
80386/486	32bits	32bits/32bits(4Gb) a 40Mhz	120MHz
Pentium	32bits	64bits/32bits(4GB) a 66Mhz	200MHz
PentiumII	32bits	64bits/36bits(64Gb) a 66Mhz	450MHz
PentiumIII	32bits	64bits/36bits(64Gb) a 100Mhz	1,13GHz

Tabla 2.2: Evolución de los procesadores de Intel

$$\text{Tiempo por tarea} = NI * CPI * C$$

2.10.2 Evolución de la arquitectura Intel iX86

Estos procesadores son los más utilizados en los computadores personales (PC). Desde la aparición del 8086 hasta el Pentium III, han sufrido una evolución constante que se muestra en la tabla 2.2. En ella se muestra el incremento de la longitud de la palabra interna (registros internos), de las longitudes de palabra del bus de datos y direcciones y de la frecuencia de la señal de reloj.

La evolución desde el punto de vista de sus unidades funcionales internas, también ha sufrido grandes cambios. El 8086 y el 80286 no integran ninguna unidad especial, a parte de las básicas ya conocidas. Sin embargo el 80386 integra una unidad de gestión de memoria (MMU) para gestionar la paginación y la segmentación de la memoria. En el 80486 se encuentra integrada una unidad de operación en punto flotante (FPU) y la **cache** de primer nivel de 8Kb. Además, comienzan los desarrollos de unidades de ejecución segmentadas, que en el caso de 486 alcanza 5 etapas. El Pentium es el primer procesador superescalar, con dos unidades de ejecución en paralelo segmentadas de 5 etapas. La arquitectura de la cache es de tipo **Harvard** con datos e instrucciones separados en memoria, se le dotó además de una unidad de predicción de saltos y se desarrolló un juego de instrucciones multimedia denominado MMX. En el **PentiumII** se introdujo una unidad de ejecución basada en la arquitectura P6, compartida con el Pentium-pro, con tres unidades segmentadas de 12 etapas. Las principales

características de esta arquitectura son que permite la **ejecución especulativa** y fuera de orden de las instrucciones, evitando dependencias y bloqueos en la unidades segmentadas. Incluye la **cache** de segundo nivel en el mismo encapsulado con una capacidad de 512 Kb, funcionado a la mitad de la velocidad del procesador. Por **último**, el **Pentium III** no tiene muchas diferencias con respecto al **Pentium II**, en **él** se introducen 70 nuevas instrucciones **multimedia** y se aumenta la velocidad del bus de memoria.

En el diseño de **procesadores** se pueden diferenciar dos tipos diferentes de tendencias, dando lugar a lo denominados **microprocesadores RISC y CISC**.

2.10.3 Computadores RISC y CISC

Si se consideran los procesadores comerciales desde el punto de vista de su repertorio de instrucciones, se pueden encontrar dos modos diferentes de diseños. La tendencia general en la **arquitectura y organización** de computadores durante muchos **años** ha sido incrementar la complejidad de la CPU: más instrucciones, más modos de **direccionamiento**, más registros especializados, **etc.** Esta filosofía de diseño es conocida como CISC (Computadores de Juego de instrucciones Complejo). Por otra **parte**, la tendencia RISC (Computadores de Juego de Instrucciones Reducido) representa una ruptura fundamental con la **filosofía** que hay detrás de los **procesadores CISC**. Con esta filosofía se consigue diseñar máquinas muy rápidas. que necesitan sin embargo del orden de un 30% más de instrucciones que una máquina CISC para un programa tipo.

Los argumentos empleados para justificar que, mientras **más rico** y variado es el **juego** de instrucciones, mejor es la arquitectura, son los siguientes:

- Un amplio juego de instrucciones facilita el diseño de compiladores.
- Los juegos de instrucciones amplios **reducen** el tamaño de los programas y al **tardarse** menos en leer el programa, la máquina es **más rápida**.
- incluir muchas funciones en forma **microprogramada** hace la máquina **más rápida**, al ser las **microinstrucciones** más rápidas que su equivalente en código máquina.

La **filosofía RISC**, por el contrario, establece una serie de principios de diseño que toda máquina **debe** poseer:

- e **Formato uniforme de las instrucciones:** El diseño de una unidad segmentada que pueda interpretar instrucciones con **formatos** muy diferentes y de longitud variable es muy complejo. Por ello, **todas** las instrucciones han de tener la misma longitud, los **códigos de operación** de todas las instrucciones han de tener la misma longitud y los campos de la **instrucción** que **especifican** los registros a leer han de estar siempre en la misma posición, para poder **simultáneamente** su lectura con la **decodificación** de la **instrucción**. Típicamente una máquina **RISC** tiene una longitud de **instrucción** de 32 bits y tiene como máximo 4 **formatos de instrucción** diferentes.
- **Pocas instrucciones y sencillas:** La sencillez del juego de instrucciones posibilita que la unidad de **control** sea cableada y muy rápida. **Así** mismo, hace posible que todas las instrucciones tengan la misma **duración**, lo que se simplifica el diseño de la unidad de control **segmentada**. En general un máquina **RISC** típica tiene un número de instrucciones menor o igual a 128 y los tipos de instrucciones se reducen a: movimiento de datos, aritméticas y **lógicas sencillas** y bifurcaciones.
- e **Pocos tipos de datos:** Por simplicidad se **utilizan** pocos **formatos** de datos. **Generalmente** un computador **RISC** empleará 4 **formatos** de datos de **64, 32, 16 y 8 bits**.
- e **Modos de direccionamiento sencillos:** Los modos de direccionamiento complejos y que requieren vados accesos a memoria tardan mucho en ejecutarse y paran la cadena de instrucciones. Una máquina **RISC** tiene como **máximo** 4 modos de **direcciónamiento**, entre los que se encuentran el inmediato y el relativo a registro.
- **Modo de ejecución registro-registro:** S610 realiza operaciones sobre **datos** contenidos en registros internos, lo que **permite** que **estas** instrucciones tarden menos en **ejecutarse** al no tener que acceder a memoria. Esta forma de **realizar** las operaciones se denomina modo de ejecución de registro a registro (**Rg-Rg**). Para intercambiar los datos desde la memoria y los registros internos se **utilizan sólo** dos instrucciones denominadas **Load** y **Store**. Esto **permite** que el **repertorio** de instrucciones sea muy regular, **ya** que la mayoría de instrucciones opera **sólo** sobre registros internos.

- Muchos registros: Para tratar de limitar el impacto del modo de ejecución Rg-Rg, las arquitecturas **RISC** suelen tener **más** registros que las **CISC**. Una máquina **RISC** suele tener un mínimo de 32 registros.

Bifurcaciones retardadas: El diseño de las máquinas **RISC** se ha de basar fuertemente en la segmentación, por lo que se incluye en su arquitectura la bifurcación retardada. Vdase capítulo 12.

- **Utilización de arquitectura Harvard:** Ya que estos procesadores necesitan muchos más accesos a memoria para leer instrucciones, se separan los **buses** y las memorias **cache** de datos y direcciones de forma que se puedan captar instrucciones y datos en paralelo.
- **Unidad de control segmentada y microcableada:** Dado que el juego de instrucciones es sencillo y regular, se permite el uso de unidades de control microcableada, que son más **sencillas** y **rápidas**. Por otro lado se introduce la segmentación que permite la división de la ejecución de una instrucción en **pequeño** pasos independientes entre sí, permitiendo que antes de que una instrucción complete todos los pasos, otra instrucción comience su **ejecución**, aumentando **así** el número de instrucciones ejecutadas simultáneamente en un determinado periodo de tiempo.

Hoy en día no está tan clara la diferencia entre procesadores **RISC** y **CISC**. Estas **filosofías** representan dos aproximaciones al diseño del **procesador** que pueden ser utilizadas en cualquier arquitectura. Un buen diseño debe saber **elección** cuáles son los atributos de **CISC** o **RISC** que se deben combinar, de esta forma se converge hacia diseños **híbridos** que explotan las mejores características de cada **método**. **Ejemplo** de esta tendencia son los **microprocesadores** de **Intel** considerados **CISC** y que actualmente incorporan unidades de ejecución **segmentadas**, arquitectura *Harvard* y paralelismo interno. Por otro lado los procesadores **RISC** cada vez **incluyen más** unidades funcionales complejas como la **MMU**, unidades de operación en punto flotante e instrucciones no tan simples.

2.11 CONCLUSIONES

La Unidad de Control es la parte del procesador que se encarga de gobernar todos los elementos internos del procesador. Las acciones que puede realizar un procesador vienen definidas en su repertorio de instrucciones. Estas instrucciones contienen toda la información necesaria para su ejecución, y se realizan mediante una serie de operaciones básicas secuenciadas con la señal de reloj del sistema. La unidad de control se puede diseñar como un circuito combinatorial digital (lógica cableada) o como un conjunto de microinstrucciones almacenadas en una memoria de control (micropogramación). Cada una de las dos alternativas permiten construir una UC, cuyas salidas son las señales de control que activan los elementos internos adecuados a cada pulso de la señal de reloj para ejecutar una instrucción. Además del gobierno interno del computador, la UC es la encargada de gestionar los eventos (excepciones) tanto internos como externos que llegan a la CPU. Todos estos conceptos han sido introducidos utilizando un computador ejemplo elemental que aun siendo muy sencillo permite comprender cuáles son las bases de funcionamiento de los procesadores comerciales.

2.12 CUESTIONES

2.1 Enumerar todos los modos de direccionamiento indicando cuáles son sus ventajas.

2.2 Se tiene una memoria con los siguientes valores almacenados:

Dirección	Contenido
100	101
101	104
102	100
103	103

Si un procesador ejecuta las siguientes instrucciones LOAD que carga un operando en el registro R2 con los siguientes modos de direccionamiento:

<i>LOAD R2,#100</i>	<i>inmediato</i>
<i>LOAD R2,\$101</i>	<i>directo</i>
<i>LOAD R2,[\\$100]</i>	<i>indirecto</i>
<i>LOAD R2,\$103</i>	<i>directo</i>
<i>LOAD R2,[\\$102]</i>	<i>indirecto</i>

¿Qué valor se almacena en dicho registro en cada una de las instrucciones?

23 Supongamos un computador con palabra de 32 bits. La CPU tiene 64 **instrucciones** diferentes, en las que se pueden especificar dos operandos, 32 registros internos de 32 bits y la posibilidad de dos modos de direccionamiento directo o relativo a registro. **Diseñar** el formato de las instrucciones. ¿Cuál es la máxima dirección de memoria que se puede alcanzar con el desplazamiento directo y con el relativo a registro? Diseñar el circuito necesario para calcular la dirección efectiva en el direccionamiento relativo a registro.

24 Diseñar una expansión de **código** de operación que **permita codificar** en una instrucción de 36 bits lo siguiente: 7 instrucciones con dos direcciones de 15 bits y una dirección de 3 bits. 500 instrucciones de una dirección de 15 bits y una de 3 bits. 50 instrucciones de 0 direcciones

25 Dibujar el **cronograma** de la instrucción *SUB #125*, siguiendo el esquema de computador elemental ejemplo.

26 ¿Qué valor estará almacenado en el acumulador y en la posición de memoria 401, cuando el computador elemental ejemplo termine de ejecutar las siguientes instrucciones?

Dirección	Contenido
400	10
401	5

bucle: **LW #401**
INC AC
SW #401
SUB #400
JMZ fin (salto condicional a línea indicada por la etiqueta **fin**)
JMP bucle (salto incondicional a línea indicada por **bucle**)
fin: **STP**

27 Realizar un programa en el lenguaje **ensamblador** del computador elemental **ejemplo que sume dos números** contenidos en **las** posiciones de memoria **100** y **101**, **guardando el resultado en la 102**.

28 Realizar un programa en el lenguaje **ensamblador** del computador elemental **ejemplo que busque un carácter en array situado** a partir de **la** posición de memoria **100**. **El final del array se indica con el valor 'O'**.

29 ¿Por qué es necesario implementar un pila para poder realizar **llamadas a subrutina recursivas?**

210 Enumerar los tipos de excepciones y explica **cuáles son sus causas.**

211 Indicar las **microinstrucciones** que **debería ejecutar una UC microprogramada** para completar la **instrucción AND DC**, en el computador elemental **ejemplo**.

212 Calcular el **tamaño** de memoria que **necesitaría** un **UC microprograma** para implementar el repertorio de instrucciones del computador elemental **ejemplo**.

213 Enumerar los tipos de excepciones y cuáles son sus causas.

2.14 Supongamos un computador de tamaño de palabra 32 bits y 16 registros, también de 32 bits cada uno. Dentro de estos registros tenemos el contador de programa (PC), y un puntero a pila (SP), siendo los demás de propósito general. El juego de instrucciones de esta máquina se reduce a dos instrucciones:

- **MOVE** origen, destino
- e **ADD** destino, operando1, operando2

y los modos de direccionamiento son:

- inmediato
- directo
- relativo a registro
- relativo a registro índice con predecremento, postdecremento, preincremento y postincremento, pero la cantidad incrementada/ decrementada es siempre 1.
- indirecto

En esta máquina cada campo de operando debe tener un campo indicador del modo de direccionamiento de dicho operando. Este campo, junto con el código de operación, es el que distingue entre los diferentes formatos de instrucción. Diseñar los formatos de instrucciones. Programar la función de salto con retorno y la función de retorno con las anteriores instrucciones, suponiendo que la dirección de retorno se almacena en:

1. Un registro general.
2. El principio de la subrutina.
3. En la pila.

2.13 BIBLIOGRAFÍA

- **Fundamentos de los Computadores. Estructura, funcionamiento interno, software de sistemas.** *Pedro de Miguel Anasagasti*. Editorial Paraninfo. Madrid, 1999.
- **Computer Organization & Design. The Hardware/Software Interface.** *David A. Paterson, John L. Hennessy*. Morgan Kaufmann Publishers, Inc. ISBN 1-55860-281-X. 1994.
- **Computer Arquitecture: A Quantitative Approach.** *John L. Hennessy, David A. Paterson*. Morgan Kaufmann Publishers, Inc. ISBN 1-55860-069-8. 1990.
- **Computer Organization and Architecture. Designing for performance.** *William Stallings*. 4 Edición. Prentice-Hall International, Inc. 1996. ISBN 0-13-394255-4.
- **Design of Microprocessor Based Systems.** *Nikitas Alexridis*. Prentice-Hall International, Inc. ISBN 0-13-588567-1. 1993.
- **The Indispensable PC Hardware Book.** *Hans-Peter Messmer*. 3^a ed. Addison Wesley Longman, Inc. 1997. ISBN 0-201-40399-4.



CAPÍTULO 3

UNIDAD ARITMÉTICO LÓGICA

3.1 INTRODUCCIÓN

En el capítulo 2 se ha descrito cómo la unidad de control secuencia las operaciones en las que se divide la ejecución de una instrucción. El procesador, después de acceder a memoria principal para copiar la instrucción en el registro de instrucción, inicia la secuencia de acciones propias de cada instrucción. Muchas de estas instrucciones consisten en la transformación de datos mediante la realización de operaciones lógicas o aritméticas. Para realizar estas operaciones existe un bloque especial de la CPU denominado Unidad Aritmético Lógica (ALU). En este capítulo se va a tratar de qué manera se pueden realizar las diversas operaciones lógicas y aritméticas, los circuitos lógicos necesarios, así como el **compromiso** entre velocidad de cálculo y complejidad de la circuitería de diversas aproximaciones útiles para realizar una misma tarea.

Gobernada por la unidad de control, la ALU tiene como entradas los datos almacenados en los registros internos de la CPU. Su salida (los datos transformados) se almacenan a su vez en los registros de la CPU. Adicionalmente, la ALU tiene como salida los indicadores de estado que señalan características del resultado de la última operación (por ejemplo, si el resultado ha sido cero, negativo, etc.)

La ALU a su vez está compuesta por diversos circuitos especializados en realizar una transformación particular de los datos, típicamente una operación

aritmética o lógica. Como ejemplo una ALU puede disponer de una unidad de **suma/resta** de enteros, una unidad de multiplicación de enteros, una unidad de división de enteros y su equivalente en punto flotante.

Estos módulos en los que se divide la ALU se pueden clasificar en función de diversos **parámetros**. Los criterios más habituales que van a caracterizar los módulos de la ALU son:

- Módulo **combinacional** o **secuencial**. En el caso de módulos **combinacionales** se tiene un circuito **digital** combinacional, sin elementos de memoria. Si se modifica uno de los operandos el resultado se modifica con un retraso que vendrá dado por la suma de los retrasos de todas las puertas que **intervienen** en la operación.
- Número de **operandos** del módulo. Hay módulos que solamente emplean **un** operando, como es el de la negación, pero **habitualmente** la mayoría de módulos **realizan** operaciones que utilizan dos operadores, como son la suma, división o AND lógico.
- incorporación de **parallelismo** al módulo. Si el módulo realiza la operación bit a bit se dirá que es serie. sin embargo, si se transforma toda la palabra simultáneamente se dirá que es paralelo. El primero es de tipo **secuencial** y **requiere** tantas fases como dígitos tengan los operandos, mientras que el segundo es de tipo paralelo.
- Operación **aritmética** o **lógica**. La operación realizada puede ser de tipo lógico (AND, OR, etc) o de tipo aritmético (suma, multiplicación, etc).
- **Integración** en la CPU. Puede ocurrir que parte de los módulos de la ALU estén integrados en la **CPU** (típicamente los que **realizan** operaciones lógicas o **aritméticas** con números enteros), y otros sean externos a la **CPU** debido a que son muy complejos y ocupan mucha **superficie** de silicio. Es el caso de los **coprocessadores** matemáticos que se **utilizaban** en **procesadores**.

Un ejemplo de módulo de la ALU es el sumador elemental de 1 bit de la figura 3.1. Este operador es aritmético y utiliza dos operandos. Suponiendo que una palabra tiene una anchura de 8 bits, si se conectan 8 sumadores elementales se obtiene un **sumador** paralelo. Sin embargo, si la unidad de control suministra

de manera secuencial los bits a una sola unidad de suma elemental y **almacena** el resultado parcial en un registro, se trata de un sumador serie.

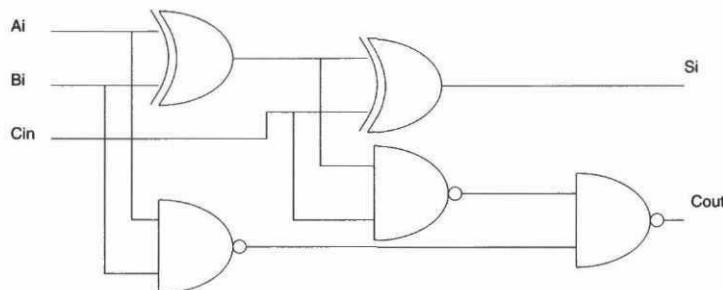


Figura 3.1: Sumador elemental de 1 bit

La principal ventaja del operador serie es que es más **pequeño** y por tanto ocupa menos superficie de silicio. La desventaja de este módulo es que es más lento que el operador paralelo. Cabe destacar **cómo** el operador serie ocupará más de 1/8 parte del módulo paralelo, al incluir también la UC módulos que **secuencian** las operaciones. En la figura 3.2 A se muestra un operador de suma paralelo de 8 bits **construido** a partir de 8 sumadores elementales de 1 bit de la figura 3.1. En la figura 3.2 B se muestra también un sumador de 8 bits, pero esta vez construido con un solo sumador de 1 bit más **circuitería** combinacional y secuencial. El secuenciador de la figura es un simple contador y emite las **señales** de control de los **multiplexores**, el **demultiplexor** y del registro de 1 bit que almacena el **acarreo** parcial.

3.2 ESTRUCTURA Y OPERACIONES DE LA ALU

La **ALU** está formada por **un** conjunto de operadores, un conjunto de registros que van a almacenar los operandos fuente y resultados parciales y unos **biestables** de estado. El **órgano** secueuciador (si hay operadores secuenciales) se integra en la UC.

Las operaciones más complejas no se **implementan** con circuitos combinacionales, ya que **requerirían** una **gran superficie** de silicio. Será la **unidad** de control la que se encarga de generar la ejecución secuencial de los algoritmos de las operaciones complejas. Un ejemplo típico son las operaciones de mul-

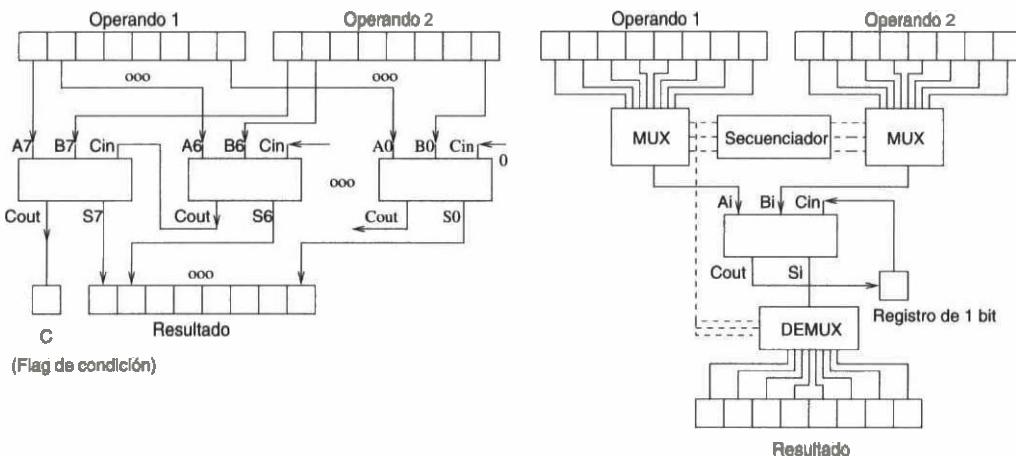


Figura 3.2: A) *Sumador* combinacional *paralelo* B) *Sumador* secuencial serie

tipificación y división. Este tipo de operaciones en procesadores pequeños no suele implementarse con lógica combinacional dentro del operador, sino que su ejecución se basa en sumas y restas elementales realizadas en módulos **combinacionales**.

Las **ALUs** suelen tener varios operadores que pueden funcionar independientemente de los demás y en algunos casos en paralelo. En el ejemplo de la figura 3.3 se puede seleccionar el módulo de la ALU que va a realizar la operación mediante unas señales específicas de selección. El banco de registros de **propósito general** sirve para almacenar resultados de operaciones intermedias, típicamente una ALU de tamaño intermedio suele tener de 8 a 16 registros. En algunas CPUs existe un **registro** especial llamado acumulador que recibe los resultados del operador y ciertas operaciones sólo pueden realizarse sobre el contenido del acumulador.

El registro de estado del **procesador** consta de una serie de indicadores o **flags** que almacenan información sobre el resultado de la última **operación** realizada. Estos indicadores, tal como ya se mostró en la sección 2.2, y definidos por IEEE, son los que se enumeran en la tabla 3.1.

Las operaciones más frecuentes que implementan los operadores de la ALU son las siguientes:

- Desplazamientos: Lógicos, circulares y **aritméticos**.

Nombre	Nemónico	Condición
ZERO	Z	Resultado cero
NOT ZERO	NZ	Resultado distinto de cero
POSITIVE	P	Resultado positivo
NEGATIVE	N	Resultado negativo
EQUAL	E	Igual
NOT EQUAL	EN	Distinto
GREATER THAN	GT	Mayor que
GREATER THAN OR EQUAL	GE	Mayor o igual que
LESS THAN	LT	Menor que
LESS THAN OR EQUAL	LE	Menor o igual que
CARRY	C	Hubo acarreo
NOT CARRY	NC	No hubo acarreo
OVERFLOW	V	Hubo desbordamiento
NO OVERFLOW	NV	No hubo desbordamiento
PARITY EVEN	PE	Paridad par
PARITY ODD	PO	Paridad impar

Tabla 3.1: Lista de indicadores de la ALU

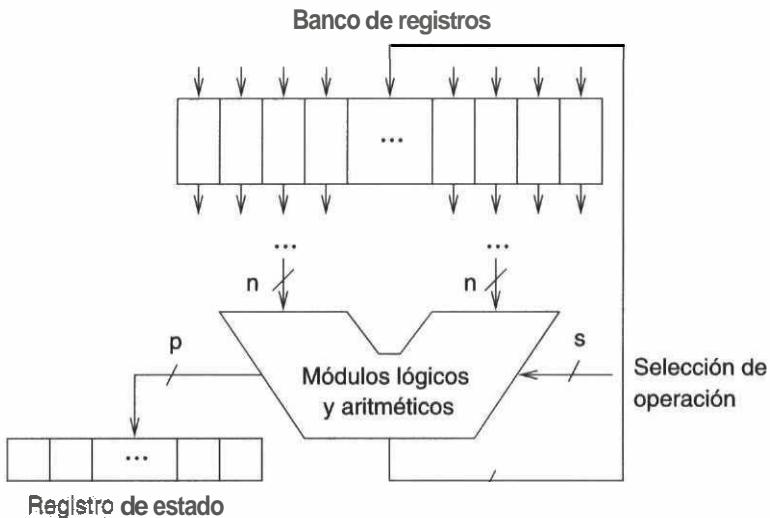


Figura 3.3: Esquema de una ALU genérica

- **Operaciones lógicas:** NOT, AND, OR, XOR.
- **Operaciones aritméticas:** Suma, resta, multiplicación y división.

Estas operaciones se pueden ejecutar de diversas maneras, dependiendo del paralelismo que se implemente en cada una de ellas. A mayor paralelismo mayor rapidez, pero habrá más unidades funcionales, con lo que la ALU ocupará más superficie de silicio y por tanto será más cara.

Las operaciones básicas y más frecuentes (**suma/resta**, desplazamientos y operaciones lógicas) siempre se implementan con una unidad **combinacional** específica. Los procesadores más potentes incorporan unidades específicas para realizar las operaciones de multiplicación y división en coma flotante, quedando para coprocesadores aritméticos y programas específicos operaciones como raíces y operaciones **trigonométricas**.

En las secciones siguientes se va a explicar de **qué** manera se implementan las operaciones habituales que realiza la ALU.

3.3 OPERACIONES DE DESPLAZAMIENTO

En este tipo de operaciones se **realiza** un desplazamiento, hacia la izquierda o hacia la derecha de los bits de un dato. El desplazamiento puede ser de n bits, aunque en los computadores **más** sencillos **sólo** se puede realizar un desplazamiento de 1 bit.

En cualquier caso una unidad de desplazamiento de n bits requiere una gran cantidad de **lógica** combinacional. Lo más habitual es que el registro origen coincida con el registro destino, y se reduzca la unidad a un simple registro de **desplazamiento**. En la figura 3.4 se muestra la **construcción** de un registro que permite desplazamientos de 1 bit a la izquierda, a la derecha y carga en paralelo. **Este registro** universal se realiza con biestables D y multiplexores.

El comportamiento de este registro **está** gobernado por las **señales** de control C_1 y C_0 . De esta manera con $C_1C_0=00$ se mantiene el estado presente, con $C_1C_0=10$ se produce un desplazamiento a la derecha de 1 bit, con $C_1C_0=01$ se desplaza 1 bit al a izquierda, y con $C_1C_0=11$ se realiza una carga en paralelo.

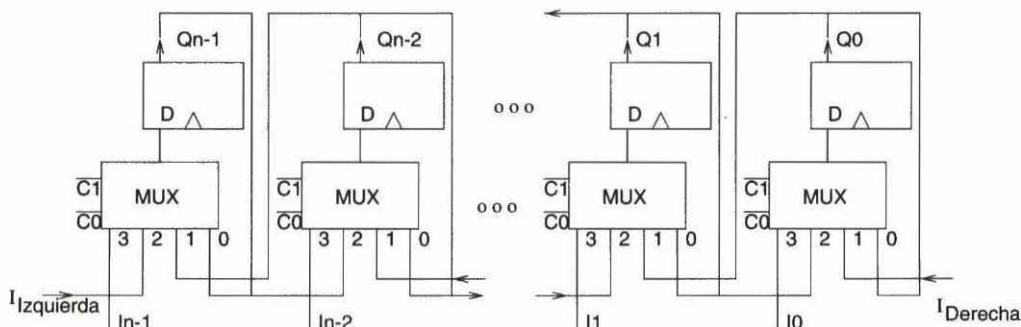


Figura 3.4: Registro universal de 8 bits

Dependiendo del tratamiento de los bits de los extremos, los desplazamientos se pueden clasificar en **lógicos**, circulares o aritméticos.

Desplazamientos lógicos

En este tipo de desplazamientos los valores de los extremos se rellenan con ceros o unos. En la figura 3.5 **se** muestra este tipo de desplazamiento para $n=2$.

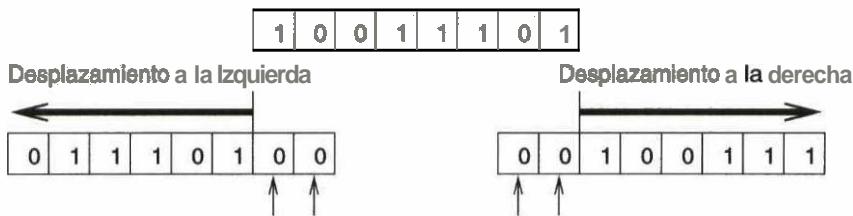


Figura 3.5: Desplazamiento lógico

Desplazamientos **circulares**

En este tipo de desplazamientos por uno de los extremos se introducen los valores que han salido por el otro extremo, tal como se muestra en la figura 3.6 para $n=2$.

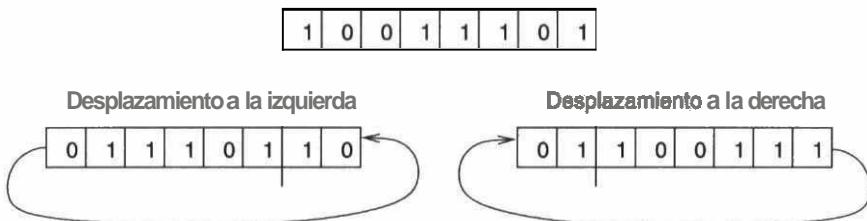


Figura 3.6: Desplazamiento circular

Desplazamientos **aritméticos**

Este tipo de desplazamientos produce una división o una multiplicación del operando por una potencia de dos. Cabe destacar que en este tipo de desplazamientos debe tenerse en cuenta el sistema de representación numérica del operador, ya que se debe tener en cuenta el signo para realizar correctamente el desplazamiento. El caso de la **representación de números** negativos en complemento a dos va **servir** de ejemplo en esta sección, aunque el desplazamiento sería análogo para el resto de representaciones.

Un desplazamiento a la derecha de 1 bit produce una división por dos. Para el caso de números positivos hay que introducir un cero, para el caso negativo (**supóngase** $-A$) su representación en complemento a 2 es el **número** $2^n - A$. Una división por 2 nos daría el resultado $\frac{2^n - A}{2} = 2^{n-1} - A/2$. Como en complemento

a 2 se cumple que $-A = 2^n - A$, **reescribiendo** la anterior expresión se obtendría $2^{n-1} + \frac{2^n - A}{2}$, que consiste simplemente realizar un desplazamiento a la derecha introduciendo un 1 en la **casilla** situada más a la izquierda.

En un desplazamiento a la izquierda debe comprobarse si los dos bits más significativos tienen un mismo valor para que no haya una pérdida de signo al perderse el **último** bit. En este caso se produciría una desbordamiento o **overflow**.

Los desplazamientos a la izquierda consisten en una multiplicación por una potencia de 2. En este caso se **desplazarían** a la izquierda los bits de la palabra **introduciendo un** cero por la derecha. Un ejemplo de desplazamiento **aritmético** de números representados en complemento a 2 se puede observar en la figura 3.7.

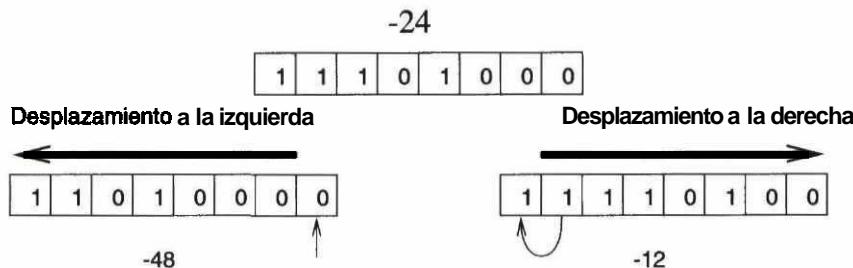


Figura 3.7: Desplazamiento aritmético

3.4 OPERACIONES LÓGICAS

Las operaciones lógicas que puede realizar un computador son:

- AND lógico
- OR lógica
- NOT o negación
- XOR lógico

Estas operaciones suelen implementarse mediante operadores paralelos **combinacionales** de **carácter** general, ya que no hay dependencia entre los diferentes

bits del operando. Esto permite que estas operaciones se realicen con **gran** rapidez. Como ejemplo, en la figura 3.8 se puede observar la ALU lógica de 1 bit. Este circuito es puramente combinacional. Con las señales O P se seleccionaría la operación a realizar (**AND**, **OR** o **XOR**). Construir una **ALU** lógica de n bits consistiría simplemente en poner en paralelo n veces este circuito básico.

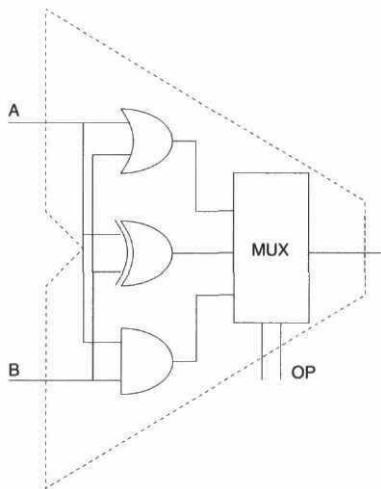


Figura 3.8: Operadores lógicos

3.5 OPERACIONES DE CAMBIO Y EXTENSIÓN DE SIGNO

Muchas veces, para implementar **una** operación de resta se opta por cambiar el signo al sustraendo y utilizar la unidad de suma. El cambio del signo de un número dependerá del sistema de representación de números negativos utilizado.

En el caso de que se utilice la representación con signo-magnitud solamente **habrá** que complementar el bit más significativo (que es el de signo). Por el **contrario**, en el caso de que se utilice representación en complemento a uno habrá que invertir todos los bits del número en cuestión. Finalmente, en el caso de que se tenga un número en complemento a 2 habrá que invertir todos los bits y sumar un 1 al resultado. Al ser el segundo operando un 1 fijo, se puede realizar la suma con un circuito combinacional **más** sencillo que un **sumador** completo.

La figura 3.9 ilustra la inversión de signo para los números enteros en estos tres formatos.

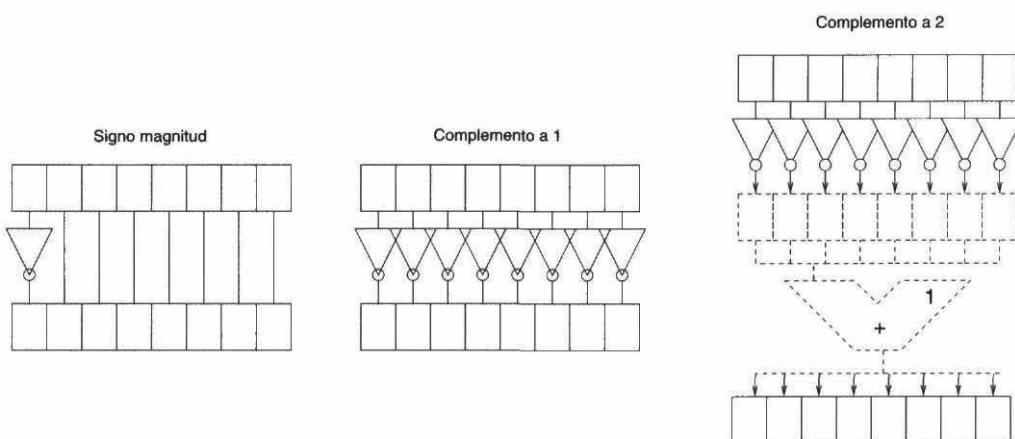


Figura 3.9: Inversión de signo en diversos formatos

Cuando **se** tiene un número **binario** representado con un cierto número de bits y se pretende representarlo con un mayor número de bits, hay que **realizar** una **extensión de signo**. Esto se consigue en el caso de signo magnitud haciendo que el número ocupe los bits menos significativos de la nueva **representación** y el bit de signo el más significativo.

Para complemento a 1 y complemento a 2 el número ocupará los bits menos significativos de la nueva representación, **mientras** que el bit de signo se replicarán para ocupar el resto de nuevos bits, **tal** como muestra la figura 3.10.

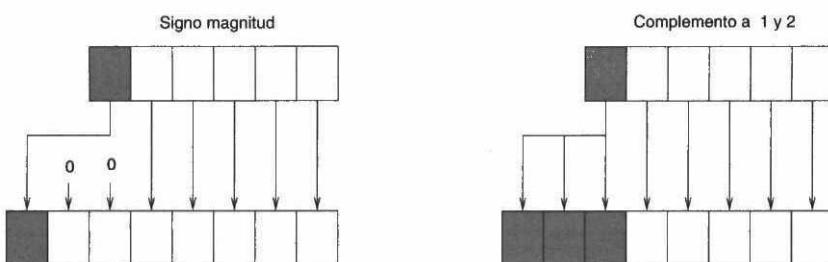


Figura 3.10: Extensión de signo en diversos formatos

3.6 SUMA Y RESTA

La operación aritmética fundamental a la que se pueden reducir todo el resto de operaciones es la suma. De hecho la resta no es más que una suma con uno de los operandos negativo, la multiplicación se puede reducir a una **serie** de sumas sucesivas y la división a una serie de sumas y restas.

3.6.1 Sumador elemental binario

El **sumador elemental** o **de un bit** se ha mostrado en la figura 3.1, y su símbolo para construir sumadores de mayor orden será el que se muestra en la figura 3.11.

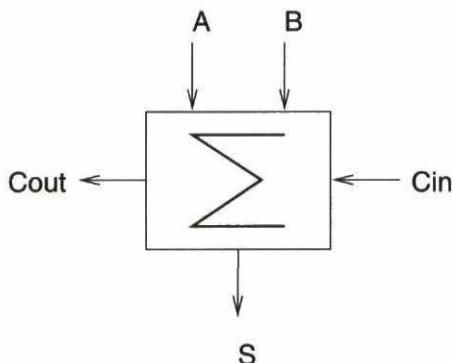


Figura 3.11: Símbolo del sumador elemental

A partir del sumador elemental de 1 bit se puede construir el **sumador paralelo** y el **sumador serie** que se muestran en la figura 3.2.

En el sumador binario serie la **operación** se realiza bit a bit. Para ello se necesitan dos registros de desplazamiento o lógica de **selección** para decidir en cada instante el bit que se debe sumar. **También es** necesario un registro de 1 bit, que inicialmente **estará** a cero, para almacenar el acarreo parcial. En el caso de la figura 3.2 B el secuenciador será un contador binario que con su estado seleccionará el bit que se debe sumar. En el caso del sumador paralelo de la figura 3.2 A se tienen n sumadores elementales con sus acarreos conectados en cascada, de forma que el acarreo de salida de un sumador sirve como acarreo de entrada del siguiente.

En este tipo de sumadores cabe destacar que:

- El acarreo del primer bit en principio debe estar a 0, aunque en operaciones de resta puede utilizarse.
- El acarreo del bit más **significativo** se conectará al bit del registro de estado **C (carry)**.
- El **retraso** total del sumador de **orden n** será $n \cdot \tau$ donde τ es el retraso en el sumador de 1 bit. Esto se debe a que el acarreo debe **propagarse** a través de toda la cadena para calcular el valor del último bit.

Esta última característica es un serio obstáculo en el diseño de ALUs suficientemente rápidas. Para evitar este problema se han desarrollado métodos de suma que anticipan el cálculo del acarreo. **Estos** son los **algoritmos** de suma conocidos como **suma con acarreo adelantado**.

3.6.2 Sumadores con acarreo adelantado

En el caso del sumador paralelo de la figura 3.2 **A** el acarreo tiene que atravesar los **n** bits para dar el resultado correcto. En los circuitos de acarreo adelantado se anticipa el acarreo mediante dos funciones **lógicas G y P llamadas** generador de acarreo y propagador de **acarreo** respectivamente. Estas funciones se definen para cada bit de la siguiente manera:

$$\begin{aligned} g_i &= a_i \cdot b_i \\ p_i &= a_i + b_i \end{aligned} \tag{3.1}$$

A partir de estas ecuaciones se puede reescribir el acarreo c_i y el resultado de la suma para cada bit s_i :

$$\begin{aligned} c_i &= c_{i-1} \cdot p_i + g_i \\ s_i &= p_i \oplus g_i \oplus c_{i-1} \end{aligned} \tag{3.2}$$

Aplicando estas ecuaciones de forma recursiva se puede obtener:

$$\begin{aligned}
 c_0 &= g_0 + c_{-1} \cdot p_0 \\
 c_1 &= g_1 + g_0 \cdot p_1 + c_{-1} \cdot p_0 \cdot p_1 \\
 c_2 &= g_2 + g_1 \cdot p_2 + g_0 \cdot p_1 \cdot p_2 + c_{-1} \cdot p_0 \cdot p_1 \cdot p_2 \\
 &\vdots \\
 c_i &= g_i + g_{i-1} \cdot p_i + \dots + g_0 \cdot p_1 \dots \cdot p_i + \dots + c_{-1} \cdot p_0 \cdot p_1 \cdot p_2 \dots \cdot p_i
 \end{aligned} \tag{3.3}$$

El resultado interesante de esta expresión del acarreo es que la función c_i no depende del acarreo del bit anterior c_{i-1} de manera que se evita el retraso de la conexión en cascada. La expresión de s_i depende de los valores de p_i y c_{i-1} . La expresión de c_{i-1} depende de los diversos valores de \mathbf{g} y \mathbf{p} , dependiendo éstos a su vez de los valores de los operandos.

Para realizar una suma de 2 operandos, en primer lugar se calcularán \mathbf{g} y \mathbf{p} en paralelo, con un tiempo de retraso de una puerta τ , para cada uno de estos valores. A partir de estos valores se calculan los acarreos c_i , con un tiempo de retraso de 2τ , ya que el árbol de lógica combinacional tiene dos puertas de profundidad. Una vez ya se tienen los retrasos se tardará un tiempo de 2τ para calcular s_i , lo que da un retraso de 5τ , independiente del número de bits.

Estos resultados presentan el problema del gran número de entradas necesarias en las puertas lógicas si el número de bits es alto. Claramente siempre se pueden construir puertas más grandes a partir de puertas pequeñas, pero en la práctica puertas lógicas con más de 5 entradas imponen un gran retraso.

La función e , requiere de una AND y una OR de $i+2$ entradas, por lo que al limitar el número de entradas de las puertas lógicas a 5, con la estrategia del acarreo adelantado en la práctica sólo se construyen sumadores de 4 bits.

3.6.3 Resta de números enteros

Los sumadores descritos hasta ahora pueden ser utilizados sin mayor problema para la operación de resta **sólo** con realizar el complemento a 2 del sustraendo. De hecho es lo que se realiza en la gran mayoría de ALUs, obviando la construcción de un circuito restador. La figura 3.12 muestra un circuito que cumple estas características. Cuando la señal \bar{s}/r vale 0 se realiza una suma del operando B. Sin embargo si \bar{s}/r vale 1 se realiza una suma del complemento a 2 del operando B, es decir se suma $-B$. Cabe destacar cómo para ello se ha calculado el complemento a 2 de B con las puertas **XOR** y con $C_{in} = 1$.

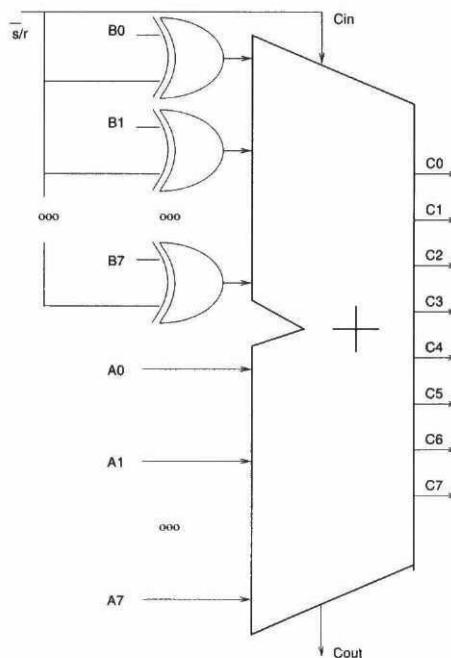


Figura 3.12: Circuito sumador restador

Análogamente se realizaría la resta en el caso de que los números negativos estén representados con signo-magnitud. La resta se realizará con el mismo circuito de suma de la figura 3.12 **sólo** con colocar el sustraendo en la entrada B del sumador restador.

En el caso de que se tengan números en complemento a 1 se utilizará un circuito similar. La única diferencia será que la señal C_{in} vale 0 para la resta y si $C_{out}=1$ hay que corregir el resultado añadiendo un 1 posteriormente.

Es interesante destacar la diferencia entre acarreo y desbordamiento en el resultado de una operación. Si se suman **números** sin signo la condición de desbordamiento se reduce a que $C_{out}=1$ ($C_{out} = C_n$). Sin embargo la condición de desbordamiento, en el caso de que se tengan números en complemento a 2, es **más** sutil. Si se suman dos números de signo contrario (o se restan dos números del mismo signo) no se puede tener desbordamiento. Sin embargo, si se suman dos números del mismo signo (o se restan dos números de signo contrario sí que se puede producir un desbordamiento). Esto va a ocurrir en la suma de números positivos representados en complemento a 2 **sólo** cuando se produzca que $C_{n-1} = 1$ (número negativo). Análogamente si se suman dos números negativos y $C_{n-1} = 0$ se ha producido un desbordamiento. La condición de desbordamiento en el caso de que se sumen números positivos o negativos del mismo signo representados en complemento a 2 D sería $D = C_{out} \oplus C_{n-1}$ o equivalentemente $D = C_n \oplus C_{n-1}$.

Como ejemplo, si se suman los **números** representados en complemento a 2 01111000 y 00100000 (+120 y +32 en decimal) el resultado de la suma seré en 8 **bits** 10011000 (con acarreo 0), cantidad que en complemento a 2 expresa el número -104 decimal. Claramente el resultado de la operación ha sido **incorrecto** y se ha producido un desbordamiento. En este caso la condición de desbordamiento D sería $D = C_n \oplus C_{n-1} = 0 \oplus 1 = 1$.

3.6.4 Sumadores en BCD

Los sumadores de código BCD se diseñan de manera análoga a los **sumadores binarios** pero **añadiéndoles** circuitos correctores para prevenir cuando la suma sea mayor que 10 y los sumadores binarios suministren un valor incorrecto. Este valor incorrecto se puede rectificar si se suma un 6, tal como indica el ejemplo **de** la figura 3.13.

En este ejemplo se realiza la suma de **1000 + 0100 = 1100** que es un resultado incorrecto en BCD. Para compensar este resultado se comege con la suma de 0110.

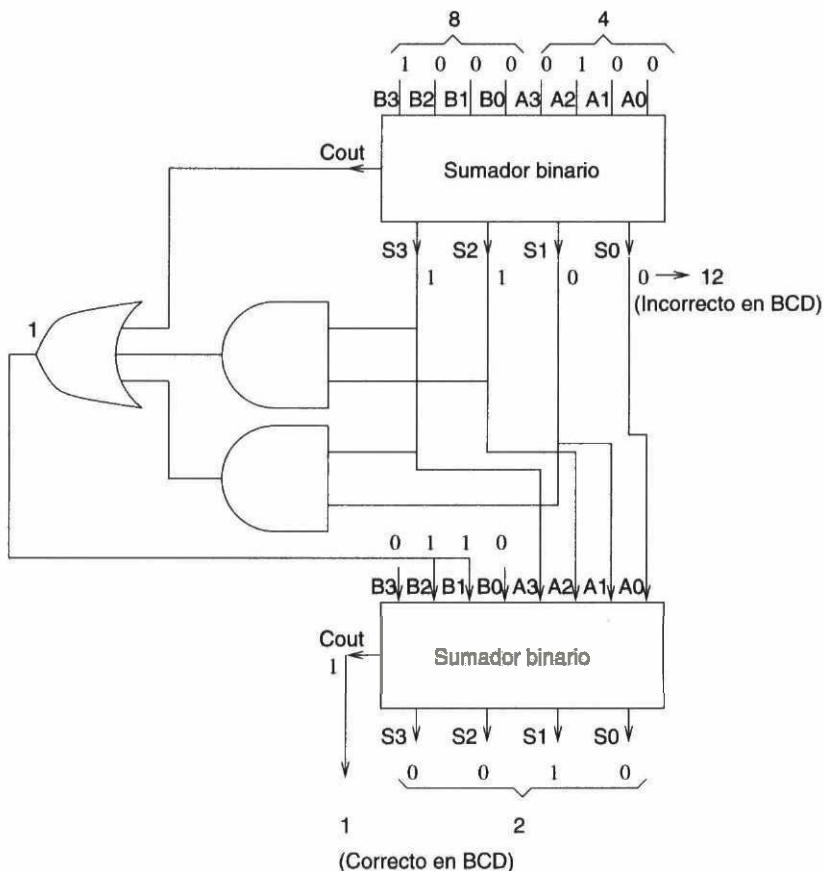


Figura 3.13: Ejemplo de suma en BCD

3.7 MULTIPLICACIÓN Y DIVISIÓN DE NÚMEROS ENTEROS

La multiplicación es una operación **más** costosa que la suma, pero no mucho **más compleja**. De hecho se puede reducir una multiplicación a una serie de sumas sucesivas con un desplazamiento. Este algoritmo de multiplicación se conoce como **multiplicación clásica de lápiz y papel**, y es la base para el algoritmo de multiplicación de números enteros **sin signo**.

3.7.1 Multiplicación de números enteros sin signo

La aproximación clásica de lápiz y papel de multiplicación decimal también se puede utilizar en el caso binario. En la figura 3.14 se muestra un ejemplo utilizando **este método** clásico de multiplicación. Si se utilizan operandos de n y m bits se obtendrá un número de $n + m$ bits.

$$\begin{array}{r}
 00011010 \longrightarrow 26 \\
 \times 0111 \longrightarrow \times 7 \\
 \hline
 00011010 \\
 00011010 \\
 00011010 \\
 \hline
 00000000 \\
 \hline
 00010110110 \longrightarrow 182
 \end{array}$$

Figura 3.14: Multiplicación clásica de lápiz y papel

La multiplicación utilizando el método de lápiz y papel es muy sencilla. Si en el multiplicador (el operando que está debajo en la operación) hay un 1, simplemente se copia el multiplicando (el operando que está arriba), en caso contrario, si en el multiplicador hay un cero, se copia una **fila** de ceros. Claramente cada vez que se realiza una multiplicación parcial hay que realizar un desplazamiento para después efectuar la suma de todos estos resultados parciales. Esta metodología se puede realizar de forma **directa** con dos aproximaciones distintas: una combinacional y otra **secuencial**.

Aproximación combinacional

A partir de la metodología del lápiz y papel, y utilizando sumadores elementales de 1 bit, se puede construir una matriz de multiplicación combinacional como la que se muestra en la figura 3.15. El truco en este diseño estriba en habilitar que se sume el multiplicando $A_{n-1}...A_0$ o una fila de ceros mediante una puerta AND que va al bit correspondiente del multiplicador $B_{m-1}...B_0$. El desplazamiento de los **resultados** parciales se realiza desplazando la conexión de las salidas de los **sumadores** a la siguiente fila de sumadores. Los acarreos se propagan a través de toda la red de **multiplicadores**.

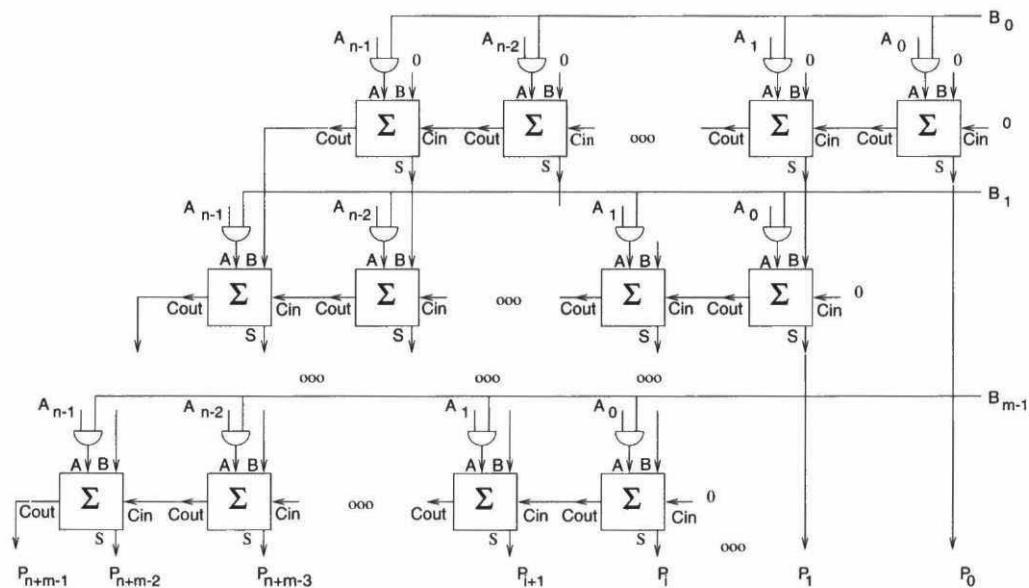


Figura 3.15: Matriz de multiplicación combinacional

Esta aproximación no se suele utilizar debido a la **gran cantidad de lógica combinacional** que utiliza, lo que se traduce en una gran superficie de silicio que acaba encareciendola ALU. A pesar de que esta aproximación es más rápida que **realizar** una circuito secuencial que ejecuta el algoritmo del **lápiz y papel**, se suelen utilizar unidades de multiplicación secuenciales. **Sólo las CPUs** muy potentes y **especializadas** incorporan unidades de multiplicación **combinacionales**.

Aproximación secuencial

La **matriz** de multiplicación de la figura 3.15 presenta el problema de que **utiliza** muchas puertas lógicas. Es por esto por lo que se han desarrollado otros **algoritmos** que, a costa de ser **más** lentos, **reutilizan** la **circuitería** de suma y **multiplicación**, y por tanto ocupan menos espacio. Un ejemplo es el esquema del sumador secuencial que se muestra en la figura 3.16.

En este circuito se supone que el multiplicando es $A_{n-1}..A_0$ y el **multiplicador** es $B_{m-1}..B_0$. El circuito consta de un **sumador** de n bits, un registro **acumulador** S de desplazamiento de n bits (con **señales** de carga en paralelo

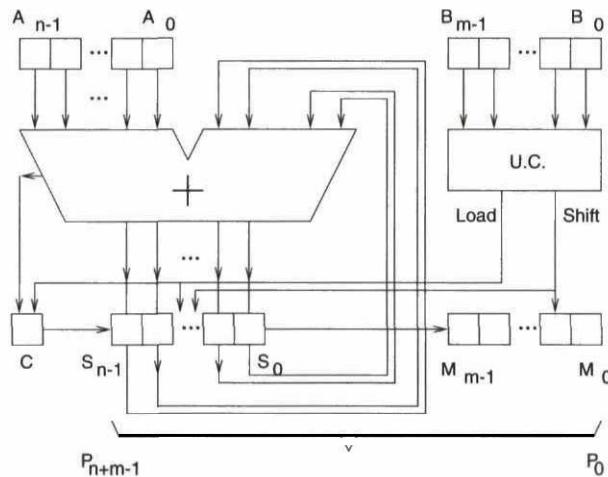


Figura 3.16: Multiplicador secuencial

Load y desplazamiento *Shift*), un registro de desplazamiento M de m bits y un registro C de 1 bit para almacenar el desbordamiento. La unidad de control no es más que una **máquina** de estados que realiza el algoritmo:

1. Se activa la señal de **reset** de todos los registros. $i=0$;
2. Mientras $i < m$ hacer
 - Si $B_i=1$ entonces activar la señal de carga **Load**, cargando en S el resultado de la suma y en C el acarreo.
 - Desplazamiento a la **derecha** (Activar la señal **Shift**), cargando en M_{m-1} el bit saliente de S_0
 - $i \leftarrow i + 1$

Con este sencillo algoritmo se realiza la multiplicación de los números enteros sin signo. Cabe destacar que cuando se realiza la carga ($B_i=1$) se actualiza tanto el registro de n bits que almacena la suma como el registro de 1 bit que almacena el desbordamiento. **Después**, independientemente del valor de B_i , se realiza el desplazamiento.

En el caso de que $B_i=0$ se desprecia la suma parcial, ya que no se carga el **valor** que aparece en la salida del sumador en el registro acumulador. Esta aproximación es más sencilla, ya que la **circuitería** es **más** simple. El resultado

Operación	i	B_i	C	Acumulador S	M
Reset	0	1	0	000000	000
Carga	0	1	0	101010	000
Desplazamiento	0	1	0	010101	000
Actualización de i	1	0	0	010101	000
No se carga	1	0	0	101010	000
Desplazamiento	1	0	0	001010	100
Actualización de i	2	1	0	001010	100
Carga	2	1	0	110100	100
Desplazamiento	2	1	0	011010	010
Actualización de i	3	-	0	011010	010
Fin del algoritmo				011010	010

Tabla 3.2: Ejemplo de multiplicación secuencial

final queda almacenado en el registro de $n + m$ bits que forman el acumulador de n bits y el registro de desplazamiento de m bits.

Como ejemplo se va a suponer que $A_{n-1..}A_0 = 101010$ (**42** decimal) u $B_{m-1..}B_0 = 101$ (**5** decimal). Claramente $n = 6$ y $m = 3$, con lo que al **inicializar** los registros se tendría 000000 en el acumulador S, 0 en el registro de 1 bit, y 000 en el registro M. **Las** operaciones y los distintos resultados se resumen en la tabla 3.2. Para seguir el ejemplo **sólo** hay que tener en cuenta que el acumulador, si $B_i = 1$, se carga con 101010 + S. Al final el valor correcto se almacena en el registro acumulador encadenado con el registro M. En este caso $011010\ 010 =$ (210 decimal), que es el resultado correcto.

Multiplicadores de alta velocidad

Tanto la aproximación combinacional como la secuencial derivadas del algoritmo del lápiz y papel son extremadamente lentas. El retraso en la aproximación combinacional descrita en la figura 3.15 tiene su **origen** en la propagación del acarreo a lo largo de todas las etapas. Este retraso se puede disminuir **significativamente** si en lugar de esperar a que el acarreo se propague a lo largo de

todos los sumadores binarios que se encuentran en su misma fila a su izquierda, se suma a la entrada del acarreo del **sumador** inmediatamente inferior a la izquierda. En realidad lo que se está haciendo es posponer la suma de los acarreos calculando previamente los **términos producto** P_{ij} . La multiplicación de la figura 3.17 se puede realizar con las dos aproximaciones que se muestran en la figura 3.18. En esta figura se muestra la propagación del **acarreo** en los caminos más latgos para el multiplicador combinacional básico y el multiplicador combinacional mejorado.

$$\begin{array}{r}
 A_3 A_2 A_1 A_0 \\
 \times B_3 B_2 B_1 B_0 \\
 \hline
 P_{03} P_{02} P_{01} P_{00} \\
 P_{13} P_{12} P_{11} P_{10} \\
 P_{23} P_{22} P_{21} P_{20} \\
 P_{33} P_{32} P_{31} P_{30} \\
 \hline
 S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0
 \end{array}$$

Figura 3.17: Términos producto en una multiplicación genérica de 4x4 bits

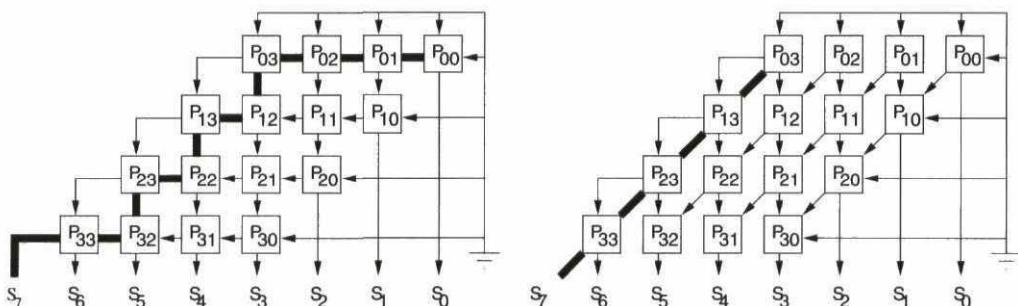


Figura 3.18: Multiplicador combinacional básico y el multiplicador combinacional mejorado

Se puede **mejorar** aún más la propagación del acarreo mediante la disposición inteligente de los productos parciales y utilizando sumadores binarios. El principio del árbol de Wallace consiste en interconectar la suma de los productos parciales utilizando sumadores binarios & 1 bit para reducir tres bits de igual peso a dos bits, uno de suma y otro de acarreo. La figura 3.19 muestra la disposición inicial de los productos parciales.

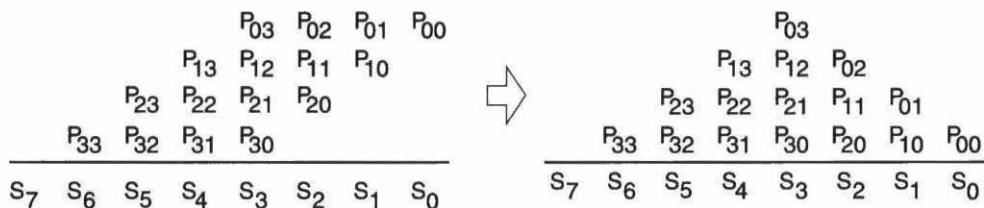


Figura 3.19: *Redistribución de los productos parciales para la interconexión con los árboles de Wallace*

La columna con altura máxima de n bits se divide en grupos de tres bits, reduciéndolos simultáneamente a dos bits. De esta manera se obtiene una columna de $\frac{2}{3}n$ que se vuelve a dividir de forma sucesiva hasta llegar a obtener una columna de dos bits. En el último nivel se utiliza un **sumador** con acarreo adelantado. La figura 3.20 muestra esta estrategia para acelerar la propagación de acarreos en el caso de $n = 4$.

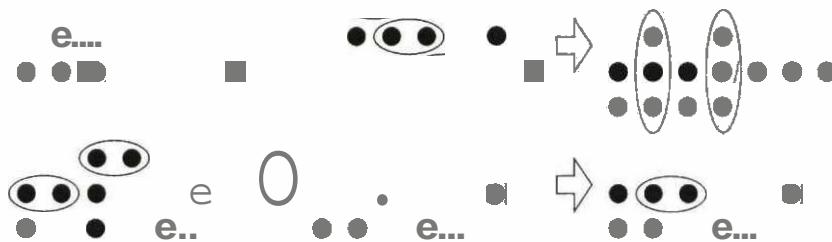


Figura 3.20: Reducción mediante un árbol de Wallace para $n = 4$

3.7.2 Multiplicación de números enteros en complemento a 2: Algoritmo de Booth

Para resolver el problema de la multiplicación de números con signo se va a analizar **sólo** el caso del complemento a 2. El caso de signo magnitudes idéntico al caso de enteros sin signo, salvo en que hay que hacer una función XOR con los signos de los operandos. El caso del complemento a 1 se considerará que se puede reducir a complemento a 2.

Para **realizar** la multiplicación de números en complemento a 2 se utiliza el algoritmo de **Booth**. Este algoritmo se conoce como el algoritmo de unos y ceros, ya que **optimiza** el número de sumas y restas para conseguir el resultado **final**.

Supóngase un multiplicando de n bits y un multiplicador de m **bits**. Para almacenar el resultado total serán necesarios $n + m$ **bits**. El algoritmo de **Booth** es un algoritmo secuencial que necesita de los siguientes elementos:

- Dos registros de n bits: A y M . En M se almacenará el multiplicando.
- Un registro S de m **bits**. En S inicialmente se almacenará el multiplicador.
- Un registro S_{-1} de 1 bit. Se situará a la derecha de S . La cadena de registros encadenada $A \rightarrow S \rightarrow S_{-1}$ debe poder **realizar** desplazamientos **aritméticos** a la derecha.
- Una unidad sumadora-restadora de **n bits**.
- La unidad de control del algoritmo debe poder comparar los bits de S y S_{-1} para detectar cadenas de unos y ceros. De la misma manera **tendrá** un contador C **módulo** m para llevar la cuenta del ciclo del algoritmo.

Con estos elementos el algoritmo de **multiplicación de números** enteros en complemento a 2 se puede **describir así**:

1. Se almacena en M el multiplicando y en S el multiplicador. **Los** registros A , S_{-1} y el contador C se **inicializan** a cero.
2.
 - Si $S_0 S_{-1} = 10$ (S_0 es el bit menos significativo de S) entonces $A \leftarrow A - M$. Se efectúa un desplazamiento aritmético a la derecha de 1 bit encadenando los registros $A \rightarrow S \rightarrow S_{-1}$. Se actualiza el contador $C \leftarrow C + 1$.
 - Si $S_0 S_{-1} = 01$ entonces $A \leftarrow A + M$. Se efectúa un desplazamiento aritmético a la derecha de 1 bit encadenando los registros $A \rightarrow S \rightarrow S_{-1}$. Se actualiza el contador $C \leftarrow C + 1$.
 - En cualquier otro caso ($S_0 S_{-1} = 11$ ó 00) se efectúa un desplazamiento **aritmético** a la derecha de p bits encadenando los registros $A \rightarrow S \rightarrow S_{-1}$, hasta que $S_0 \neq S_{-1}$ ó $C + p = m$, siendo p el número de desplazamientos de 1 bit. Adicionalmente se actualiza el contador con $C \leftarrow C + p$.

3. Si $C < m$ volver al paso 2. Si $C=m$ fin del algoritmo, estando el resultado correcto de $n+m$ bits almacenado en la cadena formada por los registro A y S.

En la figura 3.21 se muestra un ejemplo numérico de la aplicación de este algoritmo. Cabe destacar la reducción de operaciones de suma y resta gracias a la detección de unos y ceros consecutivos.

$$\begin{array}{r}
 -14 \\
 \times 7 \\
 \hline
 -98
 \end{array}
 \quad
 \begin{array}{l}
 1111\ 0010 \longrightarrow M-Multiplicando (n=8 bits) \\
 \times\ 0111 \longrightarrow S-Multiplicador (m=4 bits) \\
 \hline
 1111\ 1001\ 1110 \longrightarrow A encadenado con S-Produoto (12 bits)
 \end{array}$$

	C	A	S	S_1	
Paso 1	0	0000 0000	0111	0	Inicialización
Paso 2	0	0000 1110	011110	$S_0 S_{-1} = 10$	$A < A-M$
	1	0000 0111	0011	1	Desplazamiento de 1 bit. $C=1$
Paso 3	1	0000 0111	0011	1	$C < m$
Paso 2	3	0000 0001	1100	1	Desplazamiento de 2 bits. $C=3$
Paso 3	3	0000 0001	1100	1	$C < m$
Paso 2	4	1111 0011	1100	1	$S_0 S_{-1} = 01$ $A < A+M$
	4	1111 1001	1110	0	Desplazamiento de 1 bit. $C=4$
Paso 3	4	1111 1001	1110	0	$C=m$ Fin del Algoritmo

Figura 3.21: Ejemplo del algoritmo de Booth

3.7.3 División de números enteros sin signo

La operación de división es bastante más compleja que la de multiplicación. De nuevo existe la posibilidad de realizar una aproximación basada en el método de lápiz y papel, aunque también existen varios métodos combinacionales.

Los métodos combinacionales de división entera se basan en matrices regulares de celdas a través de las cuales se propaga el acarreo de diversas operaciones elementales de suma. Estas matrices tienen el problema de que los lazos de

realimentación son largos, y si se quieren realizar divisores de muchos bits el retraso puede llegar a ser inaceptable.

Por este motivo, para realizar la división entera en la mayoría de casos **sólo** se **recurre** a **algoritmos** secuenciales. La división entera se basa en dos **algoritmos** de **división** de números sin signo: La división con y sin restauración.

Los algoritmos de división de números enteros con signo son más complejos y requieren de transformaciones previas de los operandos. El método elegido será representar los operandos en signo magnitud y **después** asignar el signo del resultado.

División con restauración

Se pretende realizar la división de un número entero de n bits, **P**, entre un **número** entero de m bits **S**. El resultado será un cociente **Q** de n bits y un resto **R** de m **bits**. Es decir $\frac{P}{S} = Q + \frac{R}{S}$. Para poder realizar estas operaciones serán necesarios un registro **Q** de n bits donde inicialmente se almacenará el dividendo y al final estará almacenado el cociente, un registro **D** de m bits donde se almacenará el divisor, un registro **A** de m bits donde al **final** estará almacenado el resto, un módulo **sumador** restador de n bits y una unidad de control. El registro encadenado **A-Q** debe poder **efectuar** desplazamientos de un bit a la izquierda y será necesario un contador **C** para almacenar el número de la iteración. Con estos supuestos el algoritmo se ejecutará en n iteraciones y tendrá la forma:

1. **Inicializaciones:** $A \leftarrow 0$, $D \leftarrow$ divisor, $Q \leftarrow$ dividendo, $C \leftarrow 0$
2. Desplazamiento a la izquierda de 1 bit encadenando los registros **A-Q**.
3. $A \leftarrow A-D$.
4.
 - Si $A < 0$ entonces $Q_0 \leftarrow 0$ y se restaura **A** haciendo $A \leftarrow A+D$.
 - Si $A \geq 0$ entonces $Q_0 \leftarrow 1$.
5.
 - Si $C=n - 1$ entonces **fin** del algoritmo. **Q** contiene el cociente y **A** el resto.
 - Si $C < n - 1$ incrementar **C** ($C \leftarrow C+1$) y volver al paso 2.

El algoritmo toma su nombre de la operación de **restauración** que se realiza para el **registro A** si el resultado de **A-D** es negativo.

En la figura 3.22 se muestra un ejemplo del resultado de realizar la división entera de 10110011 (179) entre 0110 (6). El cociente es 00011101 (29) y el resto 0101 (5).

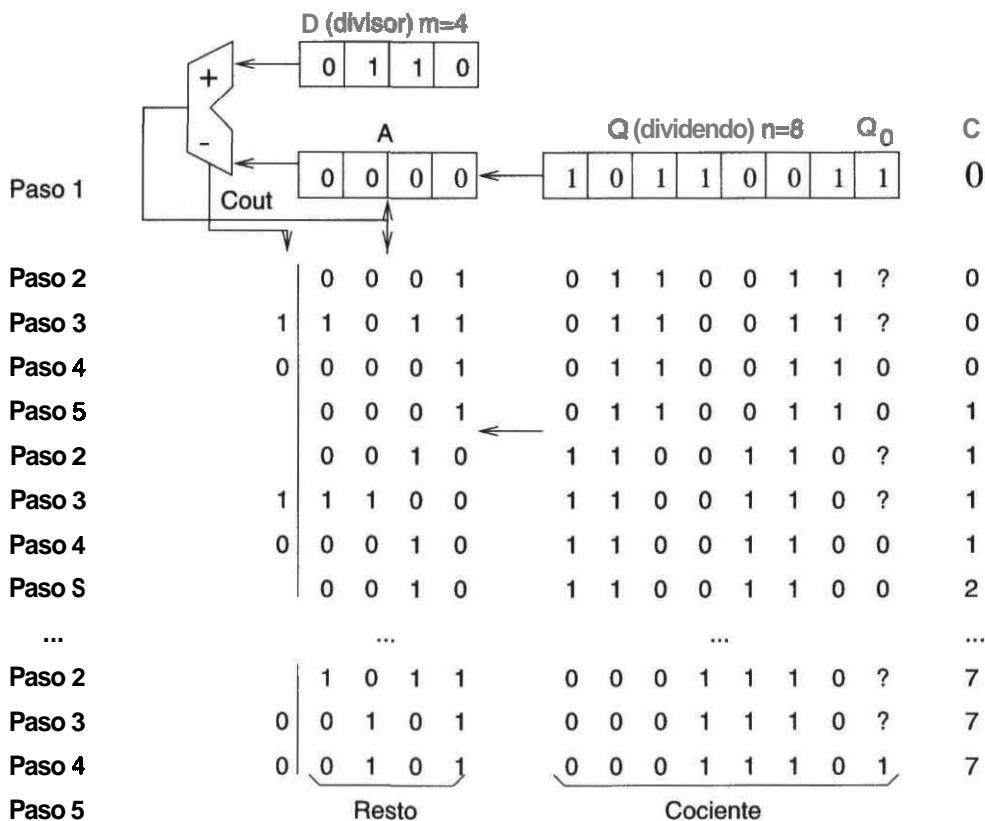


Figura 3.22: Ejemplo de división con restauración

División sin restauración

En el algoritmo de división con restauración se restaura A si A-D es negativo. Se puede realizar un algoritmo similar en el cual se evita este paso salvo al final, en el cálculo del resto. En esta versión se suma o se resta en función del valor de A. Este algoritmo se describe así:

1. **Inicializaciones:** A $\leftarrow 0$, D \leftarrow divisor, Q \leftarrow dividendo, C $\leftarrow 0$.
2. Desplazamiento a la izquierda de 1 bit encadenando los registros A-Q.
3. A $\leftarrow A-D$.
4.
 - Si $A < 0$ entonces $Q_0 \leftarrow 0$. Desplazar 1 bit A-Q a la izquierda. A $\leftarrow A+D$.
 - Si $A \geq 0$ entonces $Q_0 \leftarrow 1$. Desplazar 1 bit A-Q a la izquierda. A $\leftarrow A-D$.
5. C $\leftarrow C+1$
6.
 - Si $C < n - 1$ entonces volver al paso 4.
 - Si $C = n - 1$ entonces
 - Si $A < 0$ $Q_0 \leftarrow 0$. A $\leftarrow A+D$.
 - Si $A \geq 0$ $Q_0 \leftarrow 1$.
7. **Fin** del algoritmo. Q contiene el cociente y A el **resto**.

3.8 REPRESENTACIÓN DE NÚMEROS EN COMA FLOTANTE

Los computadores no utilizan solamente **formatos** numéricos de números enteros. Es fundamental para cualquier máquina de cálculo poder realizar operaciones con números **fraccionarios**. Es posible representar cualquier número racional o fraccionario no periódico mediante su **descomposición binaria** en potencias de 2. En el caso de **números** enteros la **descomposición** en sus dígitos binarios se realiza tras realizar divisiones sucesivas por potencias de 2 positivas. En el caso de números reales no enteros las divisiones sucesivas se extienden a potencias de 2 con el exponente negativo (2^{-1} , 2^{-2} , etc).

3.8.1 El estándar de precisión simple de IEEE

Antes de explicar cómo se realizan las operaciones en coma flotante es interesante explicar la representación de **números** reales en binario con el **estándar**

EEE. Para expresar cualquier número real en binario **sólo** hay que extender la representación en base 2 a los exponentes negativos. De esta manera se tendría por ejemplo:

$$3.5625_{10} = 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} = 11.1001_2 = 1.1001 \cdot 2^1 \quad (3.4)$$

La última representación corresponde al **estándar IEEE** de representación de números en coma flotante de precisión simple. Para la representación en binario se deja siempre en la mantisa un solo 1 a la izquierda de la coma modificándose el exponente de manera adecuada. En general se tendrá:

$$N = s \cdot M \cdot r^E \quad (3.5)$$

Los n bits que lo representan se dividen en p bits para la mantisa M y q bits para el exponente E y un bit de signo s. r es la base del exponente que se obvia al ser en el caso binario siempre 2.

El exponente se representa en binario con exceso. De esta manera en el caso del **estándar** de EEE se deberá sumar 127 al exponente E para representarlo, dando un margen de representación de números con exponente desde el -127 hasta el 128. El **estándar** de simple precisión de IEEE tiene un total de 32 bits, de los cuales el primero es para el signo, los 8 siguientes para el exponente con polarización de +127, y los 23 restantes para la parte fraccionaria de la mantisa

En el caso de la mantisa se elimina siempre el uno que está a la izquierda de la coma, por estar siempre presente, y se llenan los bits más significativos a partir de la coma hacia la izquierda. De esta manera se considerará la parte fraccionaria *f* como el número binario que corresponde a la mantisa menos 1. Como ejemplo se tiene el número:

1 01111001 01010000000

El bit más a la izquierda indica un signo menos. Los siguientes 8 bits son un número entero sin signo que representa el exponente. Como se ha apuntado **hay**

que restarle 127 para tener el valor correcto, por tanto $E=01111001_2 - 127 = -6$. El resto de **dígitos** expresan la parte **fraccionaria** f , que en este caso sería $0.0101_2 = 2^{-2} + 2^{-4} = 0.3125$. Para calcular la mantisa habría que sumarle 1, con lo que el resultado final, añadiendo el signo, quedaría $-1.3125 \cdot 2^{-6} = -0.0205078125$.

3.8.2 Operaciones en coma flotante

Para sumar (o restar) dos operandos representados en coma flotante se deben realizar las operaciones:

1. Seleccionar el número con menor exponente y desplazar su mantisa a la derecha tantos pasos como la diferencia en valor absoluto de los exponentes de los dos operandos.
2. Igualar el exponente del resultado al mayor de los exponentes de los operando~.
3. Sumar (restar) las **mantisas** y determinar el signo del operando.
4. **Normalizar** el resultado si es necesario.
5. Comprobar la **condición** de desbordamiento.

Como ejemplo se va a abordar la resta $1.5_{10} - 0.375_{10}$. Para ello hay que representar estos números en coma flotante, obteniéndose para el **estándar de IEEE** de simple precisión:

$$1.5_{10} = 1.1_2 = 1.1 \cdot 2^0 = 0\ 01111111\ 10000000000000000000000000000000 \quad (3.6)$$

$$-0.375_{10} = 0.011_2 = 1.1 \cdot 2^{-2} = 1\ 01111101\ 10000000000000000000000000000000 \quad (3.7)$$

Para realizar la resta se selecciona como exponente el mayor (0111111), se desplaza la mantisa del número menor tantas veces a la derecha como la

diferencia de los exponentes (2). Con lo que al final hay que realizar la resta $1.1002 - 0.011_2 = 1.001_2$, quedando de signo positivo.

Expresando el resultado en el estándar de IEEE se obtendrá:

$$0\ 01111111\ 00100000000000000000000000 = 1.001_2 = 1.001_2 \cdot 2^0 = 1.125_{10} \quad (3.8)$$

Análogamente la multiplicación (y división) de números en coma flotante es si cabe más sencilla. Los pasos a realizar en este caso son:

1. **Sumar** (restar) los exponentes.
2. Multiplicar (**dividir**) las **mantisas** y determinar el signo del resultado.
3. **Normalizar** el **valor** resultante si es necesario.

Como ejemplo se va a realizar la multiplicación de los **números** enteros 3.5 y 0.75. Estos números expresados en formato **IEEE** de coma flotante de precisión simple tendrán la forma 0 10000000 11000000000000000000000000000000 y 0 01111110 10000000000000000000000000 respectivamente. Si se suman los **exponentes** de 8 bits se obtiene el número 11111110, y restándole el exceso de 127, ya que se ha sumado este exceso 2 veces al estar representados ambos exponentes con exceso 127, se obtiene el exponente 01111111, lo que corresponde a un cero expresado con exceso 127.

Si se realiza la **multiplicación** de ambas **mantisas** 1.11 y 1.1 se obtiene 10.101, que corresponde al resultado previsto en decimal: 2.625. Normalizando la mantisa de nuevo para representarlo en el formato **IEEE** se obtendrá como resultado 1.0101 · 2¹. Si se añade este exponente al resultado del exponente anterior se tendrá un exponente expresado en exceso 127 igual a 10000000. La **mantisa** sería 0101 y el bit de signo no cambiaría al ser ambos números positivos. De esta manera la representación final del resultado será 0 10000000 010100000000000000000000.

Finalmente se puede realizar la división de ambas cifras de manera que 3.5 sea el dividendo y 0.75 el divisor. En este caso se restan los exponentes obteniéndose el resultado 00000010. Este resultado no está expresado en exceso

127, ya que se restan ambos excesos. Por tanto es necesario **añadir** 127 de nuevo, con lo que el exponente **será** 10000001.

Si se realiza la **división** de ambas mantisas 1.11 y 1.11 se obtendrá el resultado 1.001010101010101010101, que además no es exacto, ya que el resultado de la **división** es un número **periódico**. De esta manera, y teniendo en cuenta el exponente, el resultado obtenido es $1.001010101010101010101 \cdot 2^2 = 100.10101010101010101$, que expresado en decimal se corresponde con el número 4.666...6.

3.9 CUESTIONES

3.1 Indicar las diferencias entre un operador secuencial y un operador **combinacional**. ¿Qué ventajas presenta una **aproximación** frente a la otra?

3.2 En este capítulo se ha distinguido entre desplazamiento **lógico** y aritmético. ¿**Qué** diferencia fundamental existe entre ambos tipos de desplazamiento? ¿**Se** produce este efecto tanto en el desplazamiento a la izquierda como a la derecha? Indicar un par de ejemplos que demuestren las diferencias entre ambos tipos de desplazamiento.

3.3 **Diseñar** un circuito **lógico** que **realice** la inversión de signo de un número de 8 bits que puede estar almacenado en **formato** de signo magnitud, de complemento a 1 y de complemento a 2. El circuito debe tener 2 bits de selección con los que se **indicará** el tipo de **conversión** a realizar.

3.4 Indicar las ventajas y desventajas de los sumadores de acarreo adelantando frente a los **sumadores** realizados con **sumadores** binarios completos con el acarreo conectado en cascada.

3.5 Realizar la **multiplicación** de los **números** -20 y 5 representados con 8 bits y en **formato** complemento a 2 utilizando el algoritmo de Booth. ¿**Cuál** es el coste temporal de este producto (en **número** de ciclos de reloj)?

36 Se pretende realizar la **división** entera del número 25 (6 bits) entre el número 3 (5 bits). Indicar el tamaño de los registros implicados y el coste temporal en ciclos de reloj. Utilizar para ello el algoritmo de división entera con restauración.

37 Se pretende realizar la división entera del número 63 (8 bits) entre el número 20 (6 bits). Indicar el tamaño de los registros implicados y el coste temporal en ciclos de reloj. Utilizar para ello el algoritmo de división entera sin restauración.

38 Se propone completar el ejemplo de la figura 3.22 siguiendo el algoritmo de división con restauración, así como realizar el mismo ejemplo con el algoritmo de división sin restauración.

39 Representar los siguientes números *decimales* con el **formato de coma flotante IEEE** de precisión simple.

- 13.15625_{10}
- 2593.75_{10}
- -3124.0009765625_{10}

3.10 Expresar en **formato decimal** los siguientes números representados en el formato de **coma flotante IEEE** de precisión simple.

- $10110011010100000000000000000000$
- $00101011110110000000000000000000$
- $1011101110110110001000000000000$

3.11 Realizar la suma, producto y división de los números 21.53125_{10} y $-32.415.046875_{10}$. Utilizar para ello el formato IEEE de precisión simple.

3.10 BIBLIOGRAFÍA

- **Estructura y Tecnología de Computadores II.** *S. Dormido, M. A. Canto y J. Mira, A. E. Delgado.* Universidad Nacional de Educación a Distancia. Madrid, 1994. ISBN 84-88667-06-X.
- **Fundamentos de los Computadores. Estructura, funcionamiento interno, software de sistemas.** *Pedro de Miguel Anasagasti.* Editorial Paraninfo. Madrid, 1999. ISBN 84-283-1790-9.
- **Computer Arithmetic Systems.** *Amos R. Omondi.* Editorial Prentice Hall. Nueva York, 1994. ISBN 0-13-334301-4.
- **Digital Computer Arithmetic: Design and Implementation.** *Joseph J. E Cavanaugh.* Editorial Mc. Graw-Hill.USA, 1984. ISBN 0-07-010282-1.
- **Computer Arithmetic: Algorithms and Hardware Designs.** *Behrooz Parhami.* Editorial Oxford Univ Press. UK, 1999. ISBN 0-19-512583-5.
- **Division Algorithms and Implementations.** *Stuart E Oberman y Michael J. Flynn.* IEEE Transactions on Computers, vol. 46, nº 8, Agosto 1997.

CAPÍTULO 4

JERARQUÍA DE MEMORIA

4.1 INTRODUCCIÓN

La memoria principal (*main memory*) es la parte del ordenador donde residen los programas y los datos que éstos utilizan en el momento de su ejecución. Cuando se desea ejecutar un programa, como normalmente está almacenado en un dispositivo de almacenamiento secundario, lo primero que se hace es copiarlo en memoria. Este proceso lo realiza el Sistema Operativo (**S.O.**) teniendo también que gestionar las posiciones de memoria donde se va a cargar el programa. Una vez la copia ha finalizado, el procesador inicia la ejecución de las instrucciones leyéndolas de memoria a partir de la posición donde el S.O. ha cargado el programa.

El tamaño máximo que puede tener la memoria física de un ordenador viene determinado por el número de **líneas** de direcciones que posee el procesador. Un procesador con 16 líneas de direcciones será capaz de acceder al rango de posiciones de memoria desde la 0 hasta la $0FFFFh$. Se define el espacio de direccionamiento del procesador como la cantidad de posiciones de memoria a las que puede acceder. En el ejemplo anterior se puede decir que el espacio de direccionamiento del procesador es de 64 Kbytes. En algunos procesadores se diferencia el espacio de direccionamiento de las posiciones de memoria del espacio de direccionamiento utilizado para acceder a los recursos de los dispositivos de **Entrada/Salida**. Dependiendo de la anchura del bus de datos se define **la** longitud de palabra de la memoria como la cantidad de octetos que pueden ser

transferidos de **forma** simultánea en un ciclo de bus. Si el bus de datos es de 32 bits, la transferencia de una palabra de memoria constará de 4 **bytes**. Normalmente, la unidad de **información** menor que puede ser transferida es el byte, por lo tanto, cuando se tiene una dirección **ésta** hará referencia a un byte de la memoria. La **dirección** de dos palabras consecutivas, teniendo en cuenta el ejemplo anterior, diferiría en 4 unidades.

Cuanto antes **lleven** las instrucciones y los datos desde la memoria al **procesador** mayor **será** la velocidad de **ejecución** de los programas. Para medir la velocidad del sistema de memoria se consideran los siguientes **parámetros**:

- **Tiempo de acceso:** Tiempo mínimo que transcurre desde que las **direcciones** se depositan **en** el bus y se recogen los datos, en el caso de una lectura.
- **Tiempo de ciclo:** Tiempo mínimo que tiene que transcurrir entre dos operaciones de memoria consecutivas

Es muy importante hacer un buen diseño del sistema de memoria para minimizar estos tiempos y por tanto aumentar el rendimiento del ordenador al máximo. En este **diseño** hay que tener en cuenta que no **sólo** el **procesador** accede a la memoria, sino que **también** existen transferencias entre la memoria y los dispositivos de **entrada/salida**, haciendo que la memoria **sea** un dispositivo muy ocupado que pueda llegar a ser el cuello de botella del sistema.

Para implementar el sistema de memoria **se utilizan** memorias RAM (*Random Access Memory*), caracterizadas principalmente porque su tiempo de acceso es independiente tanto de la posición a la que se quiere acceder como de la secuencia de los accesos anteriores. Frente a estos dispositivos se encuentran las memorias de acceso secuencial, cuyo tiempo de acceso depende de la secuencia de operaciones realizadas. Hay que tener **en** cuenta que el coste por bit de **información** en este segundo tipo de dispositivos es mucho menor que en los dispositivos de acceso aleatorio, pero que debido a los tiempos de acceso tan elevados que presentan **sólo** son apropiadas como dispositivos de almacenamiento masivo para hacer copias de seguridad.

Para aumentar las prestaciones se va a incorporar una memoria **cache** construida a base de memoria SRAM de alta velocidad entre el procesador y la memoria principal del sistema. La política de acceso a esta **jerarquía** de memoria

va a **permitir** que la mayoría de los accesos **se** realicen **en** la memoria **rápida**, que contendrá por tanto una copia de un subconjunto de posiciones de la memoria principal.

Otro aspecto que influye notablemente en el **diseño** del sistema de memoria ha sido el aumento del espacio de **direcciónamiento** de los **procesadores** convencionales. A medida que las necesidades de **utilización** de los sistemas **informáticos** han ido creciendo, los programadores han realizado aplicaciones más complejas con unos requerimientos de memoria mayores. Como para ejecutar un programa, su **código** y sus datos deben estar en memoria principal, se llegó **rápidamente** al caso de que la memoria no era lo suficientemente grande como para satisfacer las necesidades de un programa.

Para aumentar el **tamaño** de la memoria se va a utilizar la memoria de almacenamiento secundario para guardar parte de los programas. Estos sistemas aparecieron en los años cincuenta, donde se **disponía** de una pequeña memoria RAM. Los programadores dividían sus programas en secciones independientes que alojaban en memoria secundaria. Las secciones **se** transferían a memoria y a medida que el programa evolucionaba se hacía un acceso secuencial a disco o a cinta **para** cargar una nueva sección que reemplazaba a otra que ya no se necesitaba en ese momento. El programador era el responsable de realizar estas tareas.

A medida que empezaron a **hacerse populares** los lenguajes de programación que **permitían** la **implementación** de programas **más** complejos y conforme el programador estaba cada vez **menos familiarizado** con el sistema **informático**, la eficiencia de los programas basados en *overlays* fue decreciendo. Empezó a emerger el problema de la asignación de memoria a los programas. Las soluciones adoptadas en la **época consistían** en realizar una asignación estática de la memoria basada en las predicciones de necesidad de memoria del programa que se **podían** hacer, y en realizar una asignación dinámica de la memoria que **crecía** y **decrecía** según las necesidades reales. **Rápidamente** resultó imposible realizar una asignación estática de la memoria debido a la complejidad de los programas.

A continuación se inventó la memoria **virtual**, que es una técnica que permite a la B_U generar direcciones **virtuales** que son **trasladadas** en direcciones físicas que se utilizan para acceder a la jerarquía de memoria. Esta técnica se **desarrolló**

por primera vez en la Universidad de Manchester con el **ATLAS Computer**, máquina con una memoria con dos niveles, una **RAM** de 16K y un tambor de 96K. Los programadores realizaban sus programas como si la memoria fuera de **96K**, y el ordenador trataba la memoria como si estuviera dividida en páginas de 512 palabras, cargando **32** páginas de programas y datos al principio. Si se accedía a una palabra de una página que no estaba en memoria, el **hardware** traía la página requerida a memoria que sustituía a otra que era pasada al tambor. Todo este proceso era transparente al usuario. Este sistema de memoria se sigue utilizando en la actualidad y recibe el nombre de memoria virtual, ya que la memoria principal parece de mayor tamaño de lo que realmente es.

En esta introducción se han **definido** los dos **parámetros** esenciales que se deben tener en cuenta en el momento de construir un buen sistema de memoria: las prestaciones y el **tamaño**. Ambos están en función del coste. y por lo tanto el objetivo va a ser **construir** una memoria de tamaño grande, de altas prestaciones, pero con un coste por bit pequeño.

Se va a diseñar un sistema de memoria jerárquico, compuesto por varios niveles de memoria que difieren entre sí en tamaño y prestaciones. Lo que se pretende es obtener unos resultados parecidos a un sistema en el que toda la memoria está compuesta por los dispositivos más rápidos de la estructura.

4.2 EL PRINCIPIO DE LOCALIDAD

A medida que el **tamaño** de las aplicaciones fue creciendo, se detectaron ciertas **características** de los **programas** en ejecución que se aprovecharon para **implementar** los sistemas de memoria de alto rendimiento con un coste bajo. La **observación** del comportamiento de los programas revela la fuerte tendencia de los accesos a memoria a estar agrupados en regiones pequeñas de memoria durante cualquier pequeño periodo de tiempo.

Los programas no necesitan acceder a su código ni a sus datos de una vez con la **misma** probabilidad. El principio de localidad asegura que los programas acceden únicamente a una porción relativamente pequeña de su espacio de **direcciónamiento** durante un corto espacio de tiempo. Por lo tanto existe una fuerte tendencia en los patrones de acceso futuros a ser similares a los patrones

ocurridos en el pasado cercano, es decir, que mirando la historia de los accesos se pueden predecir los accesos que ocurrirán en el futuro. Haciendo estas predicciones el sistema puede trasvasar bloques de datos entre los diferentes niveles de la memoria jerárquica para garantizar, con cierta probabilidad, que cuando se busca un dato en el nivel superior de la jerarquía se va a encontrar. El principio de localidad surge de la estructura propia de los programas. Existen dos tipos diferentes de localidad:

- **Temporal.** Si se hace referencia a un objeto, existe una cierta tendencia a volver a **referenciarlo** en un corto espacio de tiempo (bucles en un programa o llamadas a **subrutinas**).
- **Espacial:** Si se hace referencia a un objeto, también tenderán a ser **referenciados** los demás objetos que están ubicados en direcciones próximas a éste. (Acceso a vectores de datos o a la memoria de instrucciones).

Para aprovechar la localidad, se va a implementar la memoria como una memoria jerárquica de manera que existirán diferentes niveles de memoria con distintos tamaños y velocidades. La memoria más rápida constituirá el nivel supenor y se colocará próxima al procesador. El nivel inferior constituido por la memoria más lenta, que tiene un precio mucho menor, se colocará detrás del nivel supenor. Los programas se dividirán en bloques de tamaño fijo que se cargan en la memoria más rápida para aprovechar las ventajas del principio de localidad. Con esto, el objetivo que se persigue es que el usuario tenga la mayor cantidad de memoria posible con la tecnología más económica, pero con el tiempo de acceso ofrecido por la memoria más rápida. Las tecnologías que se usan para construir las memorias de las capas más cercanas de la jerarquía son las tecnologías de circuitos de memoria **SRAM** (*Static Random Access Memory*) y a continuación las **DRAM** (*Dynamic Random Access Memory*) con sus múltiples variantes de acceso síncrono, modo ráfaga, etc. Las capas más externas de la jerarquía de memoria las constituyen los dispositivos de almacenamiento masivo que se analizan en el capítulo 11. En la tabla 4.1 se muestran de forma comparativa los tiempos de acceso y el coste por **Mbyte** aproximados para estas tecnologías de memorias en el año 1999.

Debido al principio de localidad temporal, la mayoría de las veces se encuentra el dato en la memoria más rápida, ya que es bastante probable que el dato haya sido accedido **anteriormente**.

Tecnología	T_{ACC}	\$ por Mbyte en 1999
SRAM	8-25 ns	\$50-\$100
DRAM	25-40 ns	\$10-\$20
Disco	7-20 10^6 ns	\$0'1

Tabla 4.1: *Tiempos de acceso y precio por Mbyte (1999)*

La jerarquía de memoria puede constar de varios niveles, **pero** los datos siempre se van a copiar entre dos niveles adyacentes. Por simplicidad, aunque no se pierde generalidad, se va a considerar **únicamente** el nivel **superior** y el nivel inferior.

Cuando el **procesador** busca un dato en la memoria se pueden producir dos situaciones:

1. Acierto (**Hit**): Los datos están en el nivel superior. La tasa de aciertos (**hit rate**) define la **fracción** de los accesos a memoria en los que **se han** encontrado los datos en el nivel superior.
2. Fallo (**Miss o fault**): Los datos no están en el nivel superior. Por tanto se debe acceder al inferior para **traer** el bloque que los **contiene**. Se define la tasa de fallos (**miss mre**) como 1 menos la tasa de aciertos.

Ya que con la estructura jerárquica principalmente **se** persigue obtener un mayor rendimiento en el sistema, hay que tener muy en cuenta la velocidad con la que se procesan los aciertos y los fallos. Se define por tanto:

- **tiempo de acierto o Hit time:** tiempo necesario para acceder a los datos en el nivel superior, incluyendo el tiempo necesario para determinar si es un acierto o un fallo.
- **Tiempo de fallo o Miss time:** tiempo que se tarda en sustituir un bloque, incluyendo también el tiempo utilizado en pasarlo al **microprocesador**. Este **parámetro** es mucho mayor que el anterior.

4.3 MEMORIA CACHE

La idea de la memoria cache es similar a la de la memoria virtual en el sentido de que existe una pequeña porción de la memoria principal que está duplicada en una memoria especial (llamada memoria cache) de alta velocidad. El **término** cache se utiliza para denominar al nivel superior de la jerarquía de memoria. Cuando se genera una petición de memoria, la petición es presentada primero a la cache, y si ésta no proporciona el dato se le **presenta** entonces a la memoria principal.

Los conceptos de memoria cache y memoria virtual son similares, ya que se diferencian únicamente en la **implementación**, que será distinta debido principalmente a las diferencias de **velocidad** que existen entre los distintos niveles de la jerarquía de memoria. La memoria principal es de 4 a 20 veces más lenta que la cache y el disco es alrededor de 1000 a 10000 veces **más** lento que la memoria **principal**. El coste **tan** elevado de los fallos & página hace que se utilicen estrategias **distintas** para manejarlos, no siendo tan importante en el caso de la cache la generación de un fallo en algún acceso.

El funcionamiento de la cache es muy simple. Cuando la CPU **envía** por primera vez una solicitud de **lectura** de una **posición** de memoria, se transfiere a la cache un conjunto de palabras (**llamado** bloque) que contienen el dato **referenciado**. Sucesivos accesos a cualquiera de las posiciones del bloque hacen que su contenido **se** lea directamente de la cache. La correspondencia que existe entre los bloques de la cache y las posiciones de memoria principal la decide una función de mapa o mapeo . Cuando la cache se **llena** y se hace referencia a una palabra que no está en la cache, se debe decidir **qué** bloque se elimina para que el nuevo ocupe su lugar. El conjunto de reglas que se **utilizan** para tomar esta decisión constituye el algoritmo de reemplazo.

No es necesario que la CPU tenga conocimiento de la existencia de la cache. Al generar las **direcciones**, existe una **circuitería** que decide si el dato está en la cache o no. (Esta **circuitería** en los sistemas comerciales se implementa en un integrado especial llamado controlador de cache). En las lecturas, si el dato está en la cache, la memoria principal no interviene. Sin embargo en las escrituras se puede actualizar simultáneamente tanto la cache como la memoria (caso de una cache *write-through*) o **se** puede actualizar sólo la cache y marcarla como

modificada (caso de las caches *write-back* o *copy-back*). Más tarde, cuando se reemplace el bloque se actualizará la memoria principal. El primer **método** tiene como inconveniente que genera operaciones de escritura **innecesarias**, por ejemplo cuando se escribe varias veces sobre el mismo bloque. Sin embargo, cuando cada vez que se **modifica** la cache **también** se **modifica** la memoria principal, ambas copias son siempre iguales y por tanto evita problemas de incoherencias con los datos en los sistemas **multiprocesadores**.

Los datos en la cache se agrupan en estructuras llamadas líneas o bloques, cada uno de los cuales tiene asociada una etiqueta. El tamaño mínimo de un bloque es una palabra, sin embargo, para aprovechar el principio de localidad espacial se suelen utilizar bloques de varias palabras. Esto es debido a que si los bloques tienen un tamaño superior a una palabra, cuando ocurre un fallo en la cache se **traen** a la cache varias palabras adyacentes. Debido a este principio los bloques que se acaban de copiar tienen una gran probabilidad de ser necesitados en un **corto** espacio de tiempo, con lo que se van a mejorar las tasas de acierto en los accesos a la cache y por lo tanto el rendimiento del sistema de memoria será más elevado.

Los datos son copias de una parte de la memoria principal. Para poder averiguar de qué parte se trata se utiliza la etiqueta, que indica de alguna manera la dirección que ocupan en memoria estos datos. Cuando el procesador realiza una lectura, primero se buscan los datos en la cache utilizando la etiqueta. Si se encuentran se ha producido un acierto y el **procesador** lee los datos directamente de la cache. En caso contrario ocurre un fallo en la lectura y no hay **más** remedio que leer los datos de la memoria principal.

Las palabras de la cache y de la memoria principal se **van** a agrupar en bloques de tamaño constante. En las siguientes secciones se van a describir los **métodos** que se utilizan para asignar las posiciones en la cache donde se copia un bloque. Esta **posición** puede ser única, en el caso de tener una cache **mapeada** directamente, o bien puede ser variable en el caso de tener caches asociativas por conjuntos o caches totalmente asociativas. Para poder describir los algoritmos de **mapeado** con mayor facilidad se va a utilizar la nomenclatura definida en la tabla 4.2.

En este último caso, en donde un bloque de memoria **principal** se puede copiar en posiciones diferentes de la cache, se tiene que **utilizar** un **algoritmo** de

L	→ Número de bytes en un bloque
K	→ Número de bloques en un conjunto
N	→ Número de conjuntos en la cache
$M = K * N$	→ Número de bloques en la cache

Tabla 4.2: Parámetros que describen una cache

emplazamiento (o reemplazamiento en el caso de que haya que sustituir algún bloque) para determinar la posición de destino.

4.3.1 Mapeado directo

Este **método** se caracteriza porque cada bloque de la memoria se corresponde con un bloque único en la cache. En la figura 4.1 se muestra un ejemplo de este tipo de mapeado.

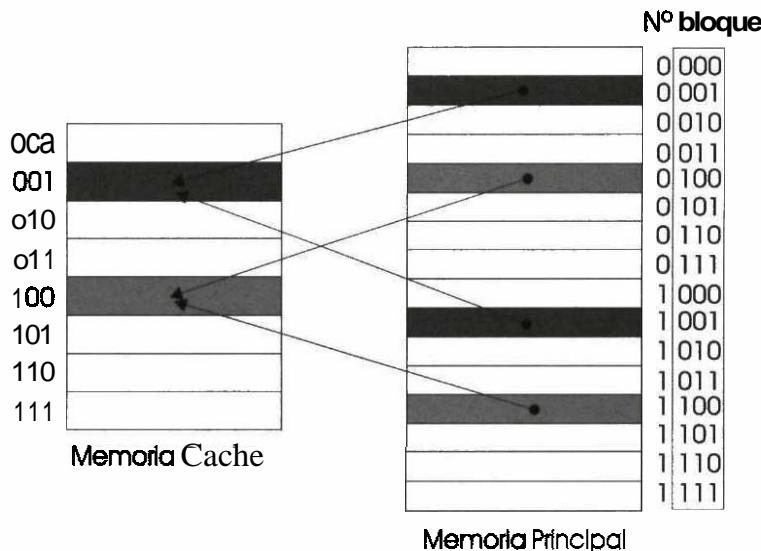


Figura 4.1: Ejemplo de mapeado directo en una cache

Cuando el **procesador** realiza un acceso a un dato se tiene que averiguar si existe una copia del mismo en la cache para evitar un acceso a memoria principal innecesario. Para ello, primero se calcula qué bloque de memoria contiene el

dato y la posición del bloque de la cache que tiene asignado. Seguidamente se compara la etiqueta con la que se obtiene de la dirección del dato y el bit de válido de dicho bloque para ver si se ha producido un acierto.

La **dirección** del bloque de memoria que contiene un dato determinado se calcula como:

$$(dirección\ del\ dato) \dividido\ (tamaño\ del\ bloque\ en\ bytes)$$

La posición que tiene asignada en la cache un bloque de memoria viene dada por la expresión:

$$(dirección\ del\ bloque) \bmod\ (tamaño\ de\ la\ cache\ en\ bloques)$$

La etiqueta que le **corresponde** al bloque que contiene el dato se calcula como:

$$(dirección\ del\ bloque) \dividido\ (tamaño\ de\ la\ cache\ en\ bloques)$$

Este método es atractivo si los **parámetros** que definen la **estructura** de la memoria cache (**véase** la tabla 4.2) son una potencia de 2. En este caso, se puede calcular la dirección del bloque que contiene al dato realizando un desplazamiento aritmético a la derecha de la dirección del dato $\log_2 L$ posiciones. De la **misma** manera los $\log_2 M$ bits menos **significativos** de la **dirección** del bloque indican la **posición** donde se copia este bloque en la cache (ver la **figura** 4.2). Este valor se calcula realizando la operación **binaria** (**dirección** del bloque en memoria) AND M. La etiqueta se **corresponde** con los bits restantes, es decir con el valor que resulta de realizar un desplazamiento **aritmético** a la derecha de la dirección del bloque $\log_2 M$ posiciones. Estas operaciones son muy sencillas y se realizan de forma muy rápida en la **ALU** de cualquier ordenador.

Cada posición en la cache puede contener varias posiciones de **memoria**, por tanto la cache **deberá** disponer de información adicional para poder diferenciar los distintos bloques de la memoria. Para ello es necesario añadir una etiqueta que identifica la dirección en memoria de cada bloque. Esta etiqueta sólo necesita guardar la parte alta de la dirección, la que no se utiliza para desplazarse sobre la cache, ya que los datos a los que apunta una dirección sólo pueden **almacenarse** en un bloque concreto & la cache. Por lo tanto el número de bloque que ocupan los datos en la cache junto con el contenido de la etiqueta de ese bloque, especifican la dirección de memoria del bloque.

De manera adicional, es necesario saber si la información contenida en un bloque de la cache es válida, para ello se añade un bit de válido a cada bloque.

En la figura 4.2 se observa un ejemplo de cache mapeada directamente con 4096 bloques de 32 bytes cada uno. Se muestra cómo se descomponen los bits de las direcciones del procesador en campos, cada uno de ellos utilizado para identificar, respectivamente y de derecha a izquierda, la posición que ocupa el byte referenciado dentro de la palabra, la posición que ocupa esta palabra dentro del bloque, la posición (número de bloque) donde se copia el bloque de memoria en la cache y por último el valor de la etiqueta que se copia en la zona reservada para almacenar la etiqueta del bloque. Como se puede observar, el bloque de datos no se compone de un único byte, sino que lo forman 4 palabras de 4 bytes cada una. Debido a que el bus de datos es de 32 bits se necesita un **multiplexor** para secuenciar la copia o la lectura del bloque de la cache.

Cuando el procesador quiere realizar un acceso a memoria, por el bus de direcciones saca la dirección de este byte. Para determinar si el dato está en la cache, la parte de la etiqueta se compara con la etiqueta contenida en el bloque de la cache cuya posición la definen los bits centrales de la propia dirección. Si son iguales y además el bit de válido del bloque está activado se produce un acierto y el procesador puede, en el caso de una lectura, leer los datos de la cache, o en el caso de una escritura escribir los datos en la misma. En caso contrario se produce un fallo y se tiene que acceder a memoria principal.

Debido a que se usan los bits menos significativos (LSB bits) de la dirección del bloque para acceder a la cache, ésta tendrá un tamaño potencia de 2. La cantidad de memoria necesaria para implementar una cache estará por tanto en función del número de bloques que se desee guardar en la cache y del tamaño del espacio de **direcccionamiento** del procesador, (que influirá sobre la longitud de las etiquetas). Como ejemplo es posible calcular la memoria necesaria para implementar una cache mapeada directamente de tamaño 2^n . Esta cache servirá para el procesador **MIPS**, que usa direcciones de 32 bits y con un tamaño bloque de 1 palabra (4 bytes).

$$\text{El tamaño de la cache sera } 2^n(32-n-2+32+1)=2^n(63-n)$$

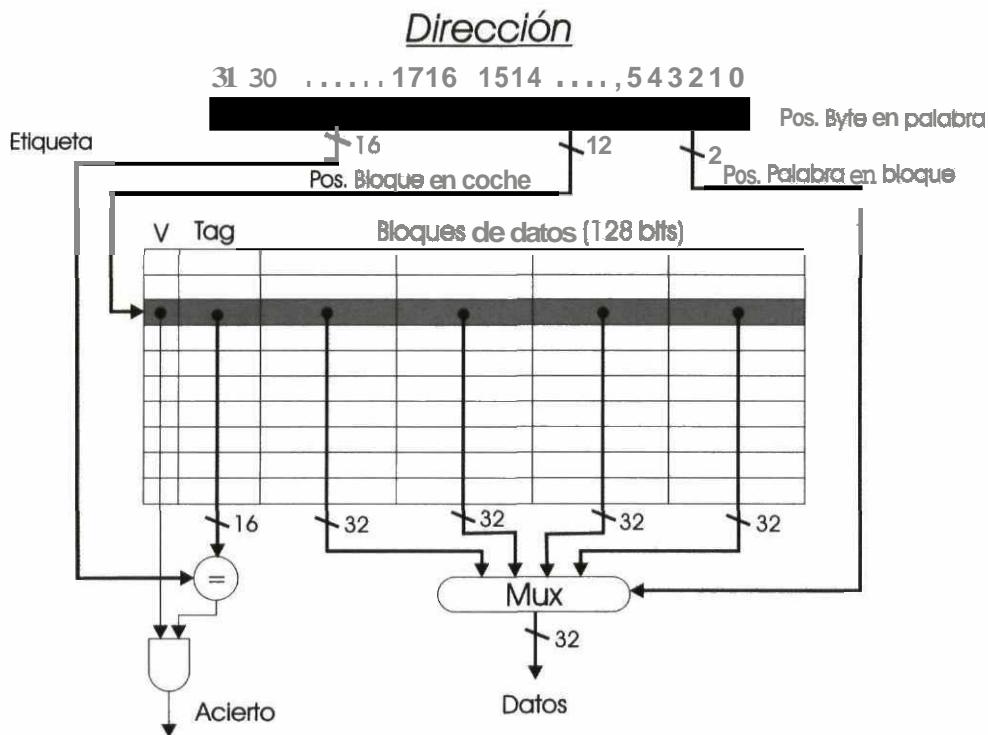


Figura 4.2: Funcionamiento de una cache con mapeado directo

4.3.2 Mapeado asociativo por conjuntos

La cache se divide en varios conjuntos (**N**) que contienen cada uno el mismo número de bloques (**K**). En este caso la cache se dice que es **K-way associative**. El número de bloques que contiene la cache se obtiene fácilmente multiplicando $N \times K$. Este algoritmo se **caracteriza** porque cada bloque de memoria principal tiene asignado un **único** conjunto dentro de la **cache**.

Para ello, se averigua primero **qué** bloque de memoria contiene el dato y **qué** conjunto le **corresponde** a ese bloque en la cache. A **partir** de este momento ya se **sabe** que si existe copia del bloque, **únicamente** se podrá encontrar en alguno de los bloques que pertenecen al conjunto que se acaba de calcular. A continuación se procede a comparar las etiquetas de estos bloques con la etiqueta que se obtiene a **partir** de la dirección del dato para ver si coinciden. En caso **afirmativo** los datos están en la cache.

La ventaja que presenta este método es que no es necesario verificar todo el directorio de la cache (la etiqueta de todos los bloques de la cache), sino **sólo** la de los bloques del conjunto.

Dada la dirección de un dato es posible calcular la **posición** que ocupa dentro de **la cache**. Como en un bloque siempre van a estar todos los bytes (**puesto** que el bloque es la unidad mínima de **información** que se transfiere **desde/a** memoria principal) no se tendrán en cuenta los $\log_2 L$ bits menos significativos de la dirección para el mapeado. Se utilizarán los $\log_2 N$ bits siguientes para **determinar** el conjunto dentro de la cache donde puede estar el bloque. El resto de la **dirección** es lo que se utilizará para compararlo con la etiqueta o **tag**. No hace falta utilizar el **número** de conjunto, ya que todos los bloques dentro del conjunto tienen el mismo **número** y por tanto es **información** redundante que no es necesario almacenar. La figura 4.3 muestra esta metodología.

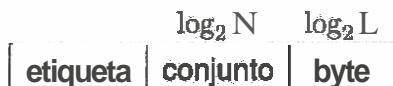


Figura 4.3: Direccionamiento de una línea de cache genérica

4.3.3 Mapeado totalmente asociativo

Si se utiliza este algoritmo de mapeado, los bloques de memoria principal pueden ocupar cualquier posición dentro de la cache. Básicamente es como si se tuviera una cache asociativa por conjuntos en donde **sólo** existe un conjunto que contiene todos los bloques de la cache ($N = 1$).

Para determinar si se ha producido un acierto o un fallo en la búsqueda de un bloque en la cache se deben comparar las etiquetas de todos los bloques con la etiqueta calculada a partir de la dirección del dato. Si alguna de ellas coincide y además su bit de válido asociado está activo se habrá producido un acierto.

La etiqueta que le corresponde al bloque que contiene el dato se calcula como:

$$(dirección\ del\ bloque)\ dividido\ (tamaño\ del\ bloque\ en\ bytes)$$

En el caso de que los parámetros que definen la **estructura** de la cache sean potencias de 2, esta operación se puede realizar de **forma** sencilla calculando el desplazamiento **aritmético** a la derecha de la dirección del bloque en $\log_2 L$ posiciones,

El mapeado totalmente asociativo se caracteriza por el **emplazamiento** flexible que tienen los bloques. Debido a ello, si se utiliza un **algoritmo** de reemplazamiento adecuado para elegir los bloques que son sustituidos en la cache se pueden conseguir tasas de acierto elevadas. El problema que presenta es que el tiempo de acierto es elevado, puesto que se tienen que **comparar** las etiquetas de todos los bloques de **la** cache y además el **tamaño** de la etiqueta es grande, puesto que la misma **coincide** con la **dirección** del bloque en memoria principal.

4.3.4 Algoritmos de reemplazo

Cuando la cache está llena, es necesario sustituir **algún** bloque para cargar otro nuevo. Se debe, por tanto, elegir una estrategia de reemplazo de forma que se mantengan en la cache los bloques que tienen **más** probabilidad de ser accedidos en futuros accesos. Para ello es útil basarse en el principio de localidad de los accesos a memoria, con lo que se reemplazaría aquel bloque que haya estado en la cache más tiempo sin haber sido accedido. Este algoritmo se denomina **LRU** (*Last Recently Used*) menos recientemente usado.

Existen **otros** algoritmos, como el aleatorio, del que se obtiene un mejor resultado en ciertos casos de lo que a priori cabría esperar.

4.3.5 Manejo de los fallos en los accesos a la cache

Cuando se produce un fallo de lectura en la cache, los datos se buscan en la memoria principal y se copian en la posición que les corresponde dentro de la cache, actualizando la etiqueta del bloque de esa posición y activando el bit de válido en el caso de que no lo estuviera. En paralelo se llevan también los datos al procesador. El conjunto de pasos que se realizan cuando se produce un **fallo** en la lectura de una **instrucción** que va a ser ejecutada (*fetch miss*) es el siguiente:

1. Se **decrementa** el contador de programa.

2. Se leen de memoria los datos.
3. Se escriben en la cache los datos (**instrucción** a ejecutar), se escriben los bits más significativos de la dirección en la etiqueta y se activa el bit de válido.
4. Se inicia de nuevo la búsqueda de la **instrucción** en la cache, produciendo en este caso un acierto o **fetch hit**.

En el caso de las escrituras pueden aparecer problemas tanto en los aciertos como en los fallos. Si se produce un acierto se puede crear una inconsistencia entre los datos que hay en la cache y los de la memoria principal. Para prevenir las **inconsistencias** no se va a permitir que el contenido de ambas memorias sea diferente, por lo que se va a usar un **protocolo** para las escrituras llamado **write through**. El protocolo consiste en que cada vez que hay que escribir en la cache un dato, se actualiza tanto la cache como la memoria central. Si se encuentra una copia de los datos que se quieren en la cache (es decir, que se ha producido un acierto en la **escritura**), simplemente se escriben los datos tanto en el bloque de la cache como en el de memoria principal.

En el caso de tener un fallo en la escritura, el procesador no puede escribir la palabra modificada directamente en la cache, en el bloque que le corresponde de acuerdo con la **función** de mapa, y actualizar la etiqueta. En este caso se produciría un error debido a que el contenido del bloque que no ha sido modificado obviamente no es una copia del bloque de memoria principal al que hace referencia la nueva etiqueta. Piénsese que al existir una nueva etiqueta se considera al bloque como copia de otro bloque diferente de memoria principal. Este problema lógicamente no existe para tamaños de bloque de una palabra.

Para solucionarlo previamente se tiene que copiar en la cache el bloque entero de memoria que se quiere modificar, garantizando de esta manera que al realizar la **posterior** escritura se producirá un acierto y por tanto no hay que modificar la etiqueta. Se puede observar que siguiendo esta estrategia cada escritura lleva consigo implícitamente una lectura de memoria.

Al usar el método **write through** el rendimiento del sistema decrece, ya que en las escrituras es como si no estuviera la cache (hay que esperar a que se escriban los datos en la memoria lenta). Para solventar este problema se usan

buffers de escritura de alta velocidad que **guardan** los **datos** para ser escritos **posteriormente** en la memoria. El procesador una vez ha escrito los datos en el **buffer** continúa ejecutando la **instrucción**. Si se llena el **buffer** el procesador debe esperar hasta que los **buffers** se escriban en memoria. El tamaño del **buffer** oscila entre 1 y 10 bloques.

4.3.6 Rendimiento de la cache

El **tamaño** del bloque afecta al rendimiento. Si se aumenta se aprovechan las ventajas de la localidad espacial y por tanto la tasa de fallos decrece. Pero si el bloque se hace demasiado grande, al tener **palabras** muy distantes entre sí no les afectará la localidad espacial y por tanto no se accederá a la mayoría de las palabras de ese bloque. Como a medida que crece el tamaño del bloque disminuye el **número** de bloques que se pueden ubicar en la **cache**, habrá gran competencia por ocupar la cache. Los bloques serán **desechados** antes de que se agoten las **posibilidades potenciales** que ofrece la localidad (se accederá sólo a unas pocas palabras **del** bloque). En consecuencia la tasa de fallos se **incrementará**. La **gráfica** de la figura 4.4 muestra la tasa de fallos que tiene un programa ejemplo en ejecución en función del tamaño del bloque. En la **gráfica** se representan las tasas de **fallo** de caches con diferente tamaño.

Otro problema que surge al aumentar el tamaño del bloque es que se incrementa el coste de un fallo (tiempo que se tarda en recuperar el bloque del nivel inferior y cargarlo en la cache). Si el tamaño del bloque aumenta, el tiempo de transferencia aumenta y con ello el tiempo que tiene que estar esperando el procesador hasta que se acabe de cargar el bloque completo. Se utilizan dos soluciones para solucionar este problema:

1. **Early Restart:** Cuando se ha cargado la palabra del bloque que produjo el **fallo**, el procesador continúa con su trabajo sin esperar a que se termine de cargar el bloque.
2. **Requested Word First:** La primera palabra de memoria que se **trae** es la que produjo el miss.

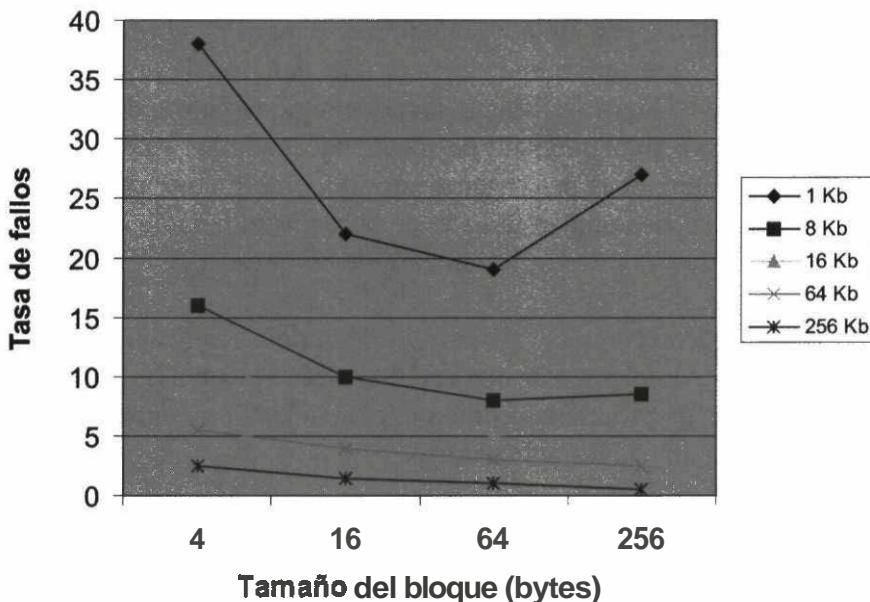


Figura 4.4: Evolución de la tasa de fallos en función del tamaño del bloque

Cálculo del rendimiento

El tiempo que tarda la CPU en ejecutar un **programa** se puede dividir en el tiempo que utiliza la unidad de control para ejecutar las instrucciones y el tiempo en el que ésta está esperando a que el sistema de memoria le suministre los datos:

$$\text{tiempo de CPU} = (\text{ciclos de reloj en ejecución} + \text{ciclos de reloj esperando a memoria}) * \text{tiempo de ciclo reloj}$$

Los ciclos de reloj en los que el **procesador** espera a la memoria se calculan como:

$$\begin{aligned} \text{ciclos de espera} &= (\text{nº accesos a memoria/programa}) * [\text{tasa de fallos} * \text{nº} \\ &\text{ciclos de memoria} + (1 - \text{tasa de fallos}) * \text{nº ciclos de cache}] \end{aligned}$$

Como se puede apreciar en esta fórmula, es vital a la hora de implementar el sistema de memoria, por un lado reducir los tiempos de acceso a los dispositivos de memoria y por otro aumentar la tasa de aciertos de la memoria cache. El primer objetivo se logra fácilmente incrementando el coste del sistema, pero para el segundo hay que sintonizar de forma adecuada los **parámetros** que determinan las características del sistema jerárquico de memoria. Es importante elegir valores adecuados para el **tamaño de** la cache, el **tamaño** del bloque y los algoritmos de **mapeo** y de reemplazamiento de bloques.

Dependiendo de la carga del sistema, es decir de las aplicaciones que vaya a realizar, **será mejor implementar** una cache u otra. Por medio de la simulación y utilizando **trazas de programas** comerciales se pueden obtener los **parámetros** óptimos para cada una de las aplicaciones.

4.3.7 Diseño del sistema de memoria

Si se diseña la memoria para transferir bloques grandes eficientemente se puede incrementar el **tamaño** del bloque y obtener un mayor rendimiento en la cache. A **continuación** se describirán los métodos que se utilizan para aumentar la velocidad de transferencia de los datos.

Aunque es difícil **reducir** el retraso en la **búsqueda** de la primera palabra de memoria, es posible reducir el coste en los fallos si se incrementa el ancho de banda (número de **bits** que se transfieren por unidad de tiempo) de la memoria central a la cache.

El tiempo necesario para acceder a la memoria se divide en:

- 1 ciclo de reloj para enviar las **direcciones**.
- 10 ciclos de reloj para iniciar el acceso a la **DRAM**.
- 1 ciclo para enviar una palabra de datos.

Entonces es posible calcular el ancho de banda para varias configuraciones, tal como se muestra en la figura 4.5. Si se desea transferir un bloque de 4 palabras de 32 bits cada una, el ancho de banda será el siguiente:

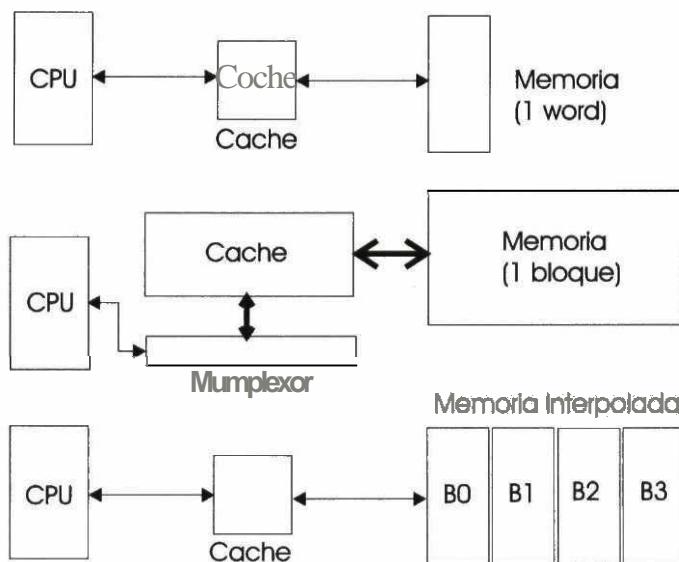


Figura 4.5: Ejemplos de diseño de sistemas de memoria

1. Anchura de memoria de una palabra: Los accesos **se** hacen **secuencialmente**.
Tiempo en cargar un bloque: $1 + 4*10 + 4*1 = 45$ ciclos.
Número de bytes transferidos por ciclo de reloj: $BW = 16 / 45 = 0.35$ bytes.
2. Anchura de memoria de 4 palabras: Se han ampliado los **buses** de **interconexión** para permitir la transferencia de todo el bloque en paralelo.
Tiempo en cargar un bloque: $1 + 1*10 + 1 = 12$ ciclos.
Número de bytes **transferidos** por ciclo de reloj: $BW = 16 / 12 = 1.33$ bytes.
3. Memoria interpolada (*interleaving*): Se **incrementa** el ancho de banda de la memoria pero no los **buses** de interconexión. Los **chips** de memoria se organizan en bancos para poder leer y escribir **múltiples** palabras en un solo acceso. Si se **envía** la dirección a todos los bancos a la vez se pueden leer simultáneamente (se **ahorra** la latencia de acceso).
Tiempo en cargar un bloque: $1 + 1*10 + 4 = 15$ ciclos.
Número de bytes **transferidos** por ciclo de reloj: $BW = 16 / 15 = 1$ byte.

Como se puede **observar**, el sistema de memoria **interpolada** aunque no **maximiza** el ancho de banda, simplifica bastante el **diseño** manteniendo una **velo-**

ciudad parecida al sistema con un bus de 4 palabras. El coste de este sistema y las probabilidades de fallo del mismo serán reducidas, por lo que habitualmente se utiliza para **construir** las memorias.

4.4 MEMORIA VIRTUAL

La cache es un medio de conseguir un acceso rápido a porciones de memoria que han sido usadas **recientemente**. De la misma manera, la memoria central puede actuar como una especie de cache para el almacenamiento secundario, implementado normalmente por medio de discos magnéticos. A este método se le llama memoria virtual. Los principios que gobiernan el comportamiento de la cache y de los sistemas de memoria **virtual** son los mismos:

- Mantener los elementos activos en la memoria de mayor velocidad.
- Enviar los datos a la memoria de menor velocidad en el momento que dejen de estar activos.
- a El rendimiento tenderá a estar cercano al rendimiento de la memoria de mayor velocidad y el coste tenderá a estar cercano al coste por bit de la memoria de menor velocidad.

La memoria física se divide en bloques del mismo tamaño **llamados** marcos de página (*frames*). La memoria lógica se divide también en bloques con el mismo tamaño llamados **páginas**. Éstas constituyen la unidad básica de información que se mueve entre la memoria principal y el dispositivo de almacenamiento secundario. Cuando un proceso va a ser ejecutado, sus páginas se cargan desde el dispositivo de almacenamiento secundario en los marcos de página libres de la memoria física. Cada página se puede cargar en cualquier marco de **página** de la memoria.

Con este método las direcciones que produce el procesador son direcciones vimiales. De la misma forma que el tamaño de la cache era inferior al de la memoria principal, ahora la CPU hace referencia a un espacio de **direcciónamiento** virtual independiente del espacio físico existente en memoria principal. El tamaño del espacio de **direcciónamiento virtual** no está en relación con la memoria física disponible y suele ser mucho mayor. A cada dirección virtual

se le asocia una **dirección** física en memoria central mediante el mecanismo de **mapeado** o de traducción de las direcciones. Si la pagina que contiene el dato no está copiada en la memoria, el resultado es un **fallo** de página que produce la copia

4.4.1 Mecanismo de traducción de direcciones

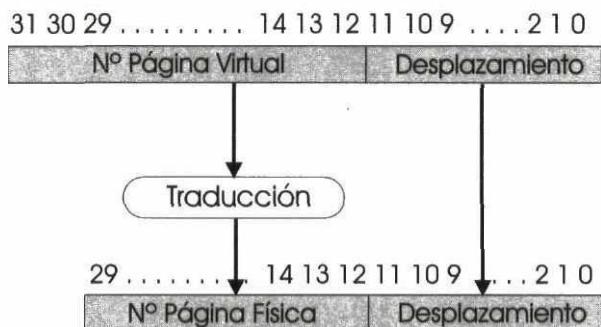
La memoria virtual necesita de la traducción de las direcciones lógicas en físicas. Para ello se suele utilizar una unidad que o bien puede estar dentro del propio procesador o bien puede ser un dispositivo especial, llamada **MMU** (*Memory Management Unit*) .

Las direcciones que genera la CPU se dividen en un número de página virtual y en un desplazamiento dentro de la página. Cada proceso utiliza una estructura en memoria llamada tabla de páginas que contiene la dirección base de cada página en memoria física. Esta tabla tendrá tantas entradas como páginas tiene el proceso. En cualquier instante de tiempo el Sistema Operativo del computador **lleva** cuenta de los procesos que se están ejecutando concurrentemente. Como consecuencia también existirán múltiples tablas de páginas en memoria y es por tanto necesario tener un **registro de tabla de página** para que apunte a la posición de comienzo de la tabla en memoria propia del proceso. Este registro se implementa dentro de la **MMU** .

Para calcular la dirección física se utiliza el número de página como índice en la tabla de páginas para calcular la dirección base de la página y sumarle el desplazamiento (que es el mismo tanto para la dirección virtual como para la física). El número de bits en el desplazamiento determina el tamaño de las páginas. **Observese** que no es necesario guardar etiquetas en la tabla de páginas, tal como se hacía en la memoria **cache**, puesto que hay una entrada en la tabla para cada posible página virtual. En la tabla de páginas existe un bit de **válido** que se activa cuando la página está cargada en memoria.

El tamaño de las páginas lo determina el hardware del computador y suele ser una potencia de 2 para facilitar la traducción de las direcciones. Si el tamaño de la página es de 2^n bytes, entonces los n bits menos significativos de la dirección lógica indican el desplazamiento dentro de la página y el resto de los bits más significativos indican el número de página. La figura 4.6 muestra un ejemplo de traducción de una dirección virtual en física.

Dirección virtual



Dirección física

Figura 4.6: Mecanismo de traducción de direcciones en la memoria virtual

Debido a la gran cantidad de páginas que puede tener un proceso, la tabla de páginas puede hacerse demasiado grande como para caber en memoria. Si por ejemplo se tienen páginas de **1K**, para un espacio de direccionamiento de 4G (direcciones de 32 bits) habrá 4M páginas. Una tabla en memoria con 4M entradas ocupa demasiada memoria. Para solucionar este problema se pagaña la tabla de páginas. En este caso el número de **página** virtual se divide en dos partes, la primera de ellas sirve para **desplazarse** sobre una tabla de tablas de páginas (que siempre están en **memoria**) que contiene la dirección base en **memoria** de las tablas de páginas donde se encuentra ya la dirección física. La segunda parte del **número** de página virtual se utiliza para desplazarse dentro de esta última tabla. En este caso un acceso a memoria conlleva incluso hasta tres lectoras.

Las ventajas que ofrece el método de traducción de direcciones en tiempo de ejecución de los programas son:

- Permite la **implementación** de la memoria virtual, en donde los **programas** pueden exceder el tamaño de la memoria física. Antes, si un programa era más grande que la memoria **física**, se tenían que usar **overlays**, que eran trozos de programas cargados por el programador cuando eran necesarios. El propio programador tenía que controlar que nunca se necesitara un **overlay** si no estaba cargado. Adicionalmente éstos nunca tenían que superar el tamaño de la memoria física.

- Aumenta el grado de multiprogramación, ya que la memoria física y el procesador pueden ser compartidos por los procesos y sólo se necesita tener en memoria una parte del programa. Esto es posible debido a que cada proceso utiliza su propio espacio de direccionamiento independiente, que es un rango separado de posiciones de memoria a los que **sólo** puede acceder el propio programa. Se tienen que implementar por tanto mecanismos de protección entre los diferentes espacios de direccionamiento de los procesos. Cada marco de página lleva asociado en la tabla de páginas unos bits de protección que informan de si la página es de **sólo-lectura** o si contiene código ejecutable. Cuando se realiza el proceso de traducción de la dirección, el **S.O. comprueba** estos bits para ver si se está realizando una operación no permitida. En ese caso se produce una interrupción debida a una violación de la protección de memoria.
- Los programadores ya no tienen que controlar el manejo de la memoria principal ni el del almacenamiento secundario, será el S.O. el encargado.
- Facilita la **relocalización**, ya que se puede cargar el programa en cualquier dirección física puesto que el espacio de direccionamiento del programa está definido a través de direcciones **virtuales**.
- Permite compartir trozos de código entre los diferentes procesos. Si el código de un programa es reentrant (también llamado código puro) significa que durante la ejecución del mismo no se modifica, por lo que diferentes procesos lo pueden ejecutar de forma simultánea. Únicamente se mantiene una copia del programa en **memoria** física. Las diferentes tablas de páginas de los usuarios asignan los marcos de página a esa misma página, utilizando páginas diferentes únicamente para los datos.

En este caso ya no se sabe dónde se va a cargar un programa en memoria, ya que como la memoria se comparte entre los programas de usuario y el Sistema **Operativo**, se deben proteger los programas entre ellos para que no se invadan los espacios de direccionamiento.

La memoria virtual presenta las desventajas de que el proceso de traducción de las direcciones ralentiza al ordenador. Los controladores utilizados en pequeñas aplicaciones de control en tiempo real no implementan estos mecanismos puesto que el procesador no es compartido entre diferentes procesos, no se

necesita protección de los espacios de **direcciónamiento** y la carga dinámica de los programas tienen lugar muy de vez en cuando.

4.4.2 Acceso a memoria

Cuando un programa desea **acceder** a una posición de memoria, primero se debe acceder a la tabla de páginas, en la posición que se obtiene de sumar el contenido del registro de tabla de páginas con el número de página de la dirección virtual, para leer el valor de número de marco de página. Este valor se **combina** con el desplazamiento de la página para producir la dirección física. Una vez que se sabe la posición en memoria donde se quiere acceder, se realiza un segundo acceso a memoria para leer o escribir el dato. Como se puede apreciar, para leer una palabra de memoria se necesitan dos accesos. Este incremento por un factor de 2 en el tiempo de acceso a la memoria es intolerable en la mayoría de circunstancias.

Para evitar este problema y debido a que el mecanismo de traducción de las direcciones también cumple el principio de localidad, se incluye una **cache** especial que recoge las entradas de la tabla de páginas más recientemente usadas. A esta cache se le llama **TLB (Translation Lookaside Buffer)** o a veces recibe el nombre de **registros asociativos**. Cada posición de la TLB contiene un **código** y un valor. En la etiqueta se **almacena** el número de página y en los datos el número de marco de página, es decir, la **dirección** base de la página en memoria **física**. La figura 4.7 muestra esta estructura.

En este caso para cada referencia se busca primero la página en la TLB. Si se encuentra, se utiliza el número de marco de **página** para acceder a memoria de forma casi inmediata. En caso contrario, se busca el número de marco de página en la tabla de páginas que **está** en memoria. Cuando se obtiene ya se puede utilizar para acceder a la memoria, pero además se **añade** el número de **marco** de página y el número de página en **los** registros asociativos para que sean utilizados en futuros accesos. Antes de realizar el acceso al dato **se comprueba** si la página está cargada en memoria. Si no lo está, el **microprocesador** ejecuta una excepción de fallo de página (**page fault**) que se encarga de copiar la página del disco en la memoria.

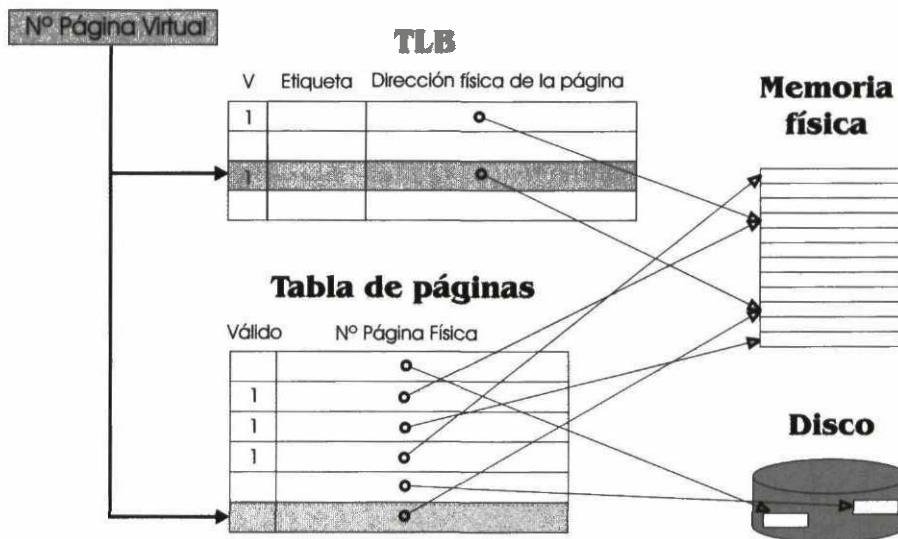


Figura 4.7: Estructura del TLB

4.43 Diseño del sistema de memoria virtual

Cuando se produce un **fallo** de página se utiliza mucho tiempo en volver a cargar la nueva página, por lo tanto en el **diseño** de estos sistemas se tienen en cuenta una serie de aspectos:

- Páginas con tamaño elevado (entre 4k y **16k**) para aprovechar **mejor** la localidad espacial.
- Emplazamiento flexible para reducir la tasa de fallos de página. Se permite emplazar cualquier página **virtual** en cualquier marco de página. De **esta forma** **cuando ocurre** un fallo se puede elegir sustituir cualquier página (**mapeado totalmente asociativo**).
- Los **fallos** de página se manejan **vía software**, puesto que mientras se responde a un **fallo de** página hay mucho **tiempo para realizar** funciones de manejo de la memoria que reduzcan futuros fallos. Si **antes** en la **cache** el **procesador** esperaba a que se cargaran los datos, ahora el **procesador** pasa a ejecutar otra tarea diferente. Cuando la **página** no está en memoria se produce una **interrupción** y se transfiere el control al Sistema Operativo. La instrucción que produjo el **fallo** se **debe** volver a ejecutar una vez

se haya cargado la pagina, por lo que se debe salvar el estado del proceso. El proceso que sigue el S. O. es el siguiente:

1. Se busca en la tabla de **páginas** la dirección de la pagina en el disco.
 2. Se escoge una **página** física para reemplazarla
 3. Se lleva la página del disco a memoria
 4. Mientras se cede el control a otro proceso.
 5. Cuando **se** termina de cargar la página se vuelve a ejecutar la operación que causó el fallo.
- El algoritmo que se utiliza para **reemplazar** las páginas es una versión simplificada del algoritmo LRU (*Least Recently Used*): Si todas las paginas físicas están ocupadas, el Sistema Operativo reemplaza la menos recientemente usada. Este algoritmo se basa en el principio de localidad temporal. Para determinar cuándo se usó la pagina por última vez se utiliza en la tabla de páginas un bit de **referencia** para cada página que **se** activa cuando **se** usa la **página**. **Periódicamente** el Sistema Operativo va desactivando poco a poco los bits de referencia.
 - Como la escritura en disco de una **página** tarda cientos de miles de ciclos de reloj, no se puede usar el **algoritmo write-through**. En este caso **se** usa un protocolo llamado **write-back** donde las escrituras individuales se acumulan en la pagina. Antes de reemplazar la página de memoria, se debe copiar en el disco si ha sido modificada. Existe un bit de sucio (**dirty** bit) en la tabla de páginas que se **actualiza** cuando se modifica (escribe por primera vez) la página.

4.4.4 Memoria segmentada

La segmentación es un esquema de manejo de la memoria que divide al espacio de **direcciónamiento** virtual en **segmentos** con distinto nombre de tamaño variable. El usuario, cuando hace referencia a un dato en la memoria, debe especificar el nombre de segmento y el desplazamiento dentro del segmento.

Para facilitar la **implementación** los segmentos no se identifican por su nombre sino que a cada uno se le asigna un número. Los compiladores y **ensambladores**, cuando construyen los programas, se encargan de **agrupar** los datos y el

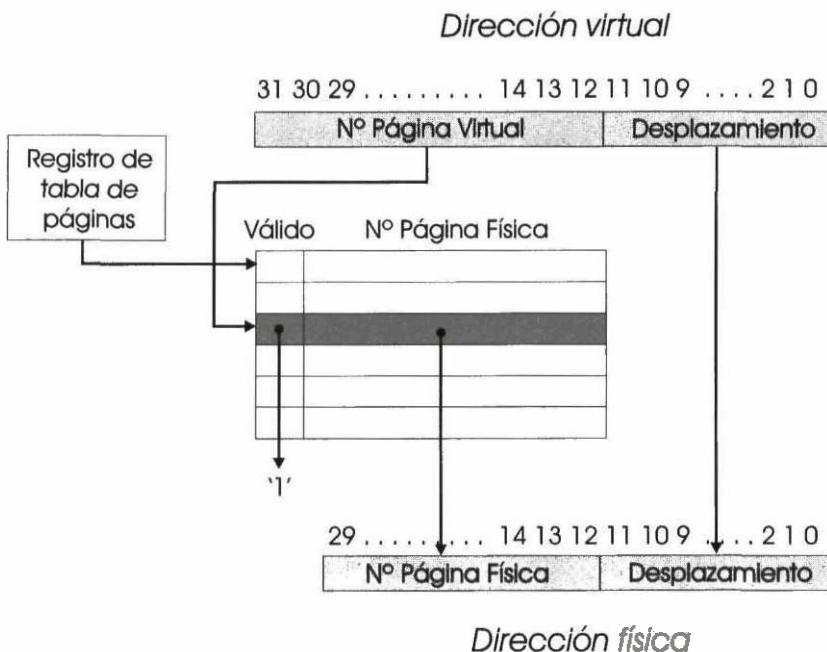


Figura 4.8: Direccionamiento de páginas en la memoria virtual

código en segmentos diferentes. Son habituales los segmentos para las variables globales del programa, para la pila y para el código del programa

Esta aproximación bidimensional que tiene el usuario del espacio de **direcciónamiento** del procesador tiene que ser traducida a un **direccionamiento** lineal para acceder a la memoria física. Para ello las direcciones lógicas se dividen en un **número** de segmento y un desplazamiento dentro del mismo. El número de segmento sirve para acceder a una tabla de segmentos que contiene para cada uno de los segmentos definidos una base de segmento y un límite de segmento. La dirección física se obtiene sumando el desplazamiento a la base del segmento. Para tener cierto control de **errores**, el controlador de memoria siempre **comprueba** que el desplazamiento de la **dirección** lógica sea menor que el **límite** de segmento. En caso contrario se ejecuta una excepción para indicar el error de acceso por encima del límite de segmento.

En los esquemas con memoria **segmentada** la tabla de segmentos también debe localizarse en la memoria y **también** se **puede** poner **parte** de ella en registros rápidos a modo de **cache** o de TLB. El procesador debe tener un registro base de

la tabla de segmentos para localizar la misma en la memoria y debido también a que un proceso puede utilizar múltiples segmentos, también debe existir un registro de longitud de tabla de segmentos para saber si existe el segmento que se está referenciando en la dirección.

Al igual que ocurre con la paginación, la segmentación también implementa métodos de protección de los accesos incorrectos a los segmentos y posibilita la **compartición** de un mismo segmento por parte de **varios** procesos.

4.5 SEGMENTACIÓN PAGINADA

Las dos alternativas anteriores (paginación y segmentación) se pueden combinar para aprovechar las ventajas de ambas. La solución consiste en utilizar segmentos de tamaño elevado pero que se almacenan en memoria de forma paginada.

Para **implementar** esta aproximación, la tabla de segmentos no contiene la dirección base de los mismos sino que contiene la dirección de la tabla de páginas asociadas a ese segmento. Las direcciones virtuales se dividen por tanto en un número de segmento, un número de **página** y un desplazamiento dentro de la página. En la **figura 4.9** se puede ver un diagrama de bloques que muestra cómo se realiza la traducción de una dirección virtual en una dirección física.

4.6 CONCLUSIONES

Durante este capítulo se ha tratado de mostrar la importancia que tiene la **relación coste/prestaciones** del sistema **informático** en la toma de decisiones sobre la arquitectura del mismo. Se han introducido los conceptos fundamentales relacionados con la estructura de la **memoria** de un ordenador: Acceso a memoria y a disco y ejecución de programas.

En las secciones que tratan sobre la memoria **cache** se ha justificado la necesidad y las ventajas de la misma, viendo además el principio de localidad como el pilar de la memoria **jerárquica**. Se ha tratado con detalle por un lado la

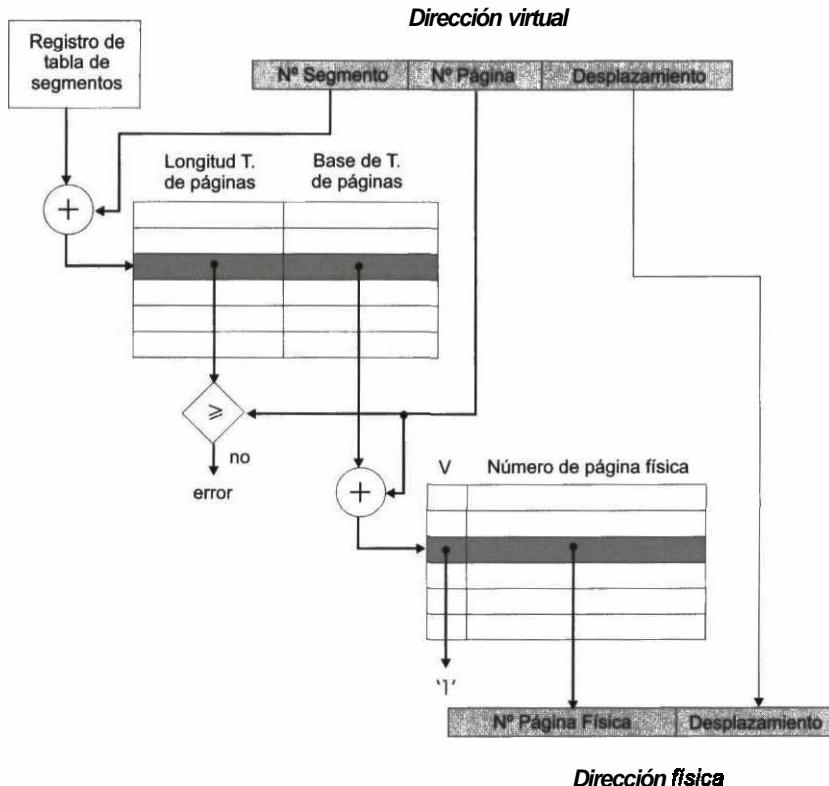


Figura 4.9: Traducción de una dirección virtual con segmentación paginada

política de emplazamiento de los bloques y por otro la política de sustitución de los mismos.

Otro aspecto importante ha sido el describir los parámetros que afectan al rendimiento de la **cache**, atendiendo a la tasa de aciertos, al número de accesos a memoria realizados y a los tiempos de transferencia de los bloques.

La evolución de la tecnología de los procesadores afecta tanto al **tamaño** de la memoria como a la complejidad y tamaño de los programas **informáticos** que se realizan. Las nuevas necesidades de memoria hacen que se implementen la paginación y segmentación como técnicas que se pueden solapar para resolver estos problemas. Utilizando estas técnicas se dispone de un espacio de **direcciónamiento** virtual mucho mayor que la memoria física disponible en el sistema. Al dividirse los programas en trozos, llamados páginas, no hace falta tener todo

el programa en memoria, sino que ésta puede estar compartida entre diferentes procesos. Se facilita además la compartición de código y la realización de programas reubicables.

4.7 CUESTIONES

4.1 *¿Qué es la memoria jerárquica?* Dibujar un esquema de un procesador conectado a una memoria de este tipo y explicar los niveles que podemos encontrar.

4.2 Explicar el **funcionamiento** de una cache, tanto en las operaciones de lectura como en las de escritura, suponiendo que es de tipo write-through. *¿Cómo* afecta al rendimiento del sistema el usar esta política para las escrituras? *¿Cómo* se puede mejorar este rendimiento? *¿Solventa* el problema de las inconsistencias?

4.3 Se quiere diseñar un sistema **microprocesador** que tenga una cache de datos de 256 **Kb**. El espacio de direccionamiento del procesador es de **16 Mb** y la **longitud** de palabra de **64 bytes**. La cache estará mapeada **directamente** y los bloques serán de **16 palabras**. *¿Cuánta memoria habrá* que comprar para implementar dicha cache?

4.4 Explicar las diferencias **existentes** en cuanto a su manejo entre un **programa** constituido por **Overlays** y un **programa paginado**.

4.5 *¿Cómo* se **aprovecha** el principio de localidad mostrado por los programas en ejecución para definir los **parámetros** de la cache.

4.6 *¿Cómo* influye en **tamaño** del bloque en el rendimiento de la cache? Diferenciar entre lecturas y escrituras.

4.7 *¿Qué* operaciones se tienen que realizar desde que el procesador genera una **dirección** virtual para leer una **página** de un programa que se encuentra en disco? Suponer el caso peor y que en el sistema existe una **TLB**.

4.8 Indicar los objetivos que se deben seguir a la hora de implementar un sistema jerárquico de memoria y **cómo** se consiguen.

4.9 ¿Por qué se utilizan tamaños potencias de 2 a la hora de implementar las caches?

4.10 Comparar el algoritmo de mapeado directo con respecto al asociativo, atendiendo tanto a la memoria que utilizan como al rendimiento que presentan.

4.11 La carga de un sistema **informático** se caracteriza porque existe un porcentaje elevado de escrituras, pero sin embargo, debido a que la cache es de tamaño reducido, la tasa de aciertos en escritura es muy baja. ¿Qué política en escrituras se debería utilizar? ¿Es conveniente modificar el tamaño del bloque? Razonar brevemente la respuesta.

4.12 ¿Qué tecnología de memoria se emplea para implementar las memorias caches?

4.13 Si se desea poder ejecutar en un sistema **informático** programas más grandes, razonar qué **parámetros modificarías** en el mismo: el **número** de líneas de direcciones y **datos del procesador**, el tamaño de la cache, el tamaño de la memoria **física**, el tamaño del disco duro.

4.14 Dada una cache con un **tamaño fijo** para los datos, indicar si la modificación de los siguientes **parámetros influiría sobre** el tamaño total de la **cache**: **tamaño del espacio de direcciónamiento del procesador**, **algoritmo utilizado** para el mapeado de los bloques, **tamaño del bloque**, **número** de bloques.

4.15 Indicar **cómo influye** en el rendimiento del **sistema de memoria**: el aumento del **tamaño** del bloque, la **utilización** del algoritmo Requested Word First (Primerola **palabra** pedida), utilizar en la cache de instrucciones **una** política Write-back, incremento del ancho de banda de la memoria principal.

4.16 Indicar las semejanzas y diferencias que existen entre cache y memoria virtual.

4.17 Algunas **memorias** caches utilizan buffers de escritura. Explicar brevemente lo que son, para qué sirven y cuándo es apropiado utilizarlos.

4.18 Dibujar un diagrama con todos los elementos necesarios **para implementar** un sistema con memoria virtual.

4.19 Indicar qué problema se produce cuando un proceso puede tener un **número considerablemente** alto de **páginas**. ¿**Cómo** se soluciona?

4.20 Siendo **h** el porcentaje de aciertos (**hit ratio**) que se produce en la traducción de direcciones de un **programa** en ejecución, **t_c** el tiempo de acceso a la memoria cache y **t_p** el tiempo de acceso a la memoria principal, calcular el tiempo medio de acceso **tm** del sistema **jerárquico** de memoria.

4.21 Indicar **para** qué **sirve** el algoritmo de reemplazamiento de una memoria cache y explicar por qué no tiene sentido hablar de este algoritmo en las memorias con mapeado directo.

4.22 Explicar las diferencias entre memoria cache y memoria virtual con respecto a:

- Traducción de direcciones:
- Tamaño de bloque/página:
- Política en escrituras:

4.23 Si se dispone de una **cache** write-back con 2 Mbytes de memoria para almacenar 512 bloques y el **procesador** es capaz de **direccionar** 4 Gbytes de memoria, indicar **cómo** podríamos saber cuántos **bits** tendrán las etiquetas en el caso de que: a) la **cache** sea con mapeado directo y b) la cache sea totalmente asociativa.

4.24 En un **sistema** con memoria virtual, calcular el **tamaño** de la tabla de páginas sabiendo que el espacio de direccionamiento virtual es de 1 Gbyte, **las páginas** son de 16 **Kbytes**, y que el sistema tiene 4 Mbytes de memoria **RAM**. Razonar **la** respuesta.

4.25 ¿Qué papel desempeñan en **el proceso de traducción** de las direcciones virtuales los siguientes elementos: Páginas, Registro de Tabla de Páginas, Tabla de Páginas, **TLB**? Poner **un** ejemplo.

4.26 ¿Qué parámetros se **utilizan** para medir las prestaciones del sistema de memoria?

4.27 ¿Qué parámetros deciden la cantidad de memoria que se necesita para implementar una cache **asociativa** en conjuntos de **k** (**k-way associative**)?

4.28 Se quiere **diseñar** un sistema **microprocesador** que tenga una cache de **datos** write-through de **256 Kb**. El espacio de **direcciónamiento** del procesador es de **16 Mb** y la **longitud** de palabra de **64 bits**. La cache estará **mapeada** directamente y los bloques serán de 16 palabras. ¿Cuánta memoria habrá que comprar para implementar dicha cache?

4.29 Si se tiene una **cache write-back** con 16 bloques y algoritmo de reemplazamiento **LRU**, indicar cuáles de **los** siguientes accesos **producen** un hit o un miss. Se supone que inicialmente la cache **está** vacía, que cada bloque contiene una **única** palabra y que el **procesador** tiene un bus de datos 16 bits y un bus de direcciones de **10 bits**. Considerar **dos** casos, uno suponiendo que la cache utiliza el **algoritmo** de emplazamiento **mapeado** directo y otro suponiendo que la cache es totalmente **asociativa**. **Calcular** también el tamaño en bytes de la cache.

Contenido de la memoria: **27A(00), 27B(01), 27C(02), 27D(03), 27E(04), 27F(05), ... , 39A(06), 39B(07), 39C(08), 39D(09), 39E(0A), 39F(0B)**.

Secuencia de accesos: Lectura palabra en **27E**, lectura palabra en **39E**, lectura byte en **39A**, escritura byte (FF) en **39B** y escritura byte (00) en **27E**

4.30 Sea un procesador con los **buses multiplexados** de direcciones y datos de **16** bits y una cache de **1kb** con **64** bloques. Calcular la cantidad de memoria que se necesita para implementar la cache, sabiendo que ésta utiliza mapeado directo y que es de tipo **write-back**. Además indicar cuál es la etiqueta y a qué bloque de la cache irían los datos que se corresponden con las direcciones: **34h**, **39eh** y **8ab3h**.

4.31 Si se tiene una cache **write-back** con **2 Mbytes** de memoria para almacenar **40%** bloques y el procesador es capaz de direccionar **4 Gbytes** de memoria, indicar cómo se podría saber cuántos bits **tendrán las** etiquetas en el caso de que la cache sea con mapeado directo o bien totalmente asociativa.

4.32 Considerar un sistema **informático** con un procesador que tiene un espacio & direcciónamiento de **4Kb**, un **bus** de datos de **32** bits y una cache write-through con **mapeado** directo de **32** bloques con **2** palabras por bloque. Calcular el **tamaño** en **bytes** de la cache y las tasas de aciertos y de fallos suponiendo que se acceden a las siguientes posiciones de memoria **expresadas** en **hexadecimal**: **320 (l)**, **420 (l)**, **352 (e)**, **426 (l)**, **422 (e)**, **320 (l)**, **255 (l)**, **422 (e)**, y que inicialmente la cache está **vacía**.

4.33 Supóngase un sistema informático que utiliza un espacio de direcciónamiento virtual de **1 Mbyte** y tiene una memoria **física** de **512 Kb**. Teniendo en cuenta el contenido de los registros de tabla de páginas para los procesos **P0**, **P1** y **P2**, y que las longitudes respectivas de las **tablas** de páginas **para** cada proceso son de **8** elementos. ¿Cuál será el **tamaño** de las páginas? Indicar para los siguientes accesos si se produce un acierto o un fallo de página al ejecutar las instrucciones que se indican en la tabla, añadiendo también la dirección **física** de memoria a la que se accede. Indicar también cuál será el **contenido final** de las tablas de páginas. Suponer que el algoritmo de reemplazo de páginas es el **LRU** (Least Recently Used).

T. páginas	Proceso	D. Virtual	Ac/Fa.	D. Física	Reg. T.pág.
0	P0	$280 \cdot 2^{10}$			P0 0
1	P1	$700 \cdot 2^{10}$			P1 4
2 0	P2	$130 \cdot 2^{10}$			P2 C
3	P0	$600 \cdot 2^{10}$			
4	P1	$130 \cdot 2^{10}$			
5					
6 $128 \cdot 2^{10}$					
7					
8					
9 $256 \cdot 2^{10}$					
A					
B					
C					
D $384 \cdot 2^{10}$					
E					
F					
10					
11					
12					
13					

4.34 Supóngase que un sistema informático con un espacio de direccionamiento de **1Kbyte** y una cache con 8 bloques ejecuta un programa que accede sucesivamente a las siguientes direcciones: **D3, 134, 318, 2AA, A6, 1F, 111, 13C** y **D1**. Completar la etiqueta y el bit de válido de las caches siguientes, suponiendo en primer lugar que la cache tiene mupeado directo y en segundo lugar que la cache es totalmente asociativa, sabiendo que en ambas el **tamaño** del bloque es de **8 bytes**. (Inicialmente la cache tiene todos los bits de válido a 0).

4.35 Si se tiene un procesador hipotético con un espacio de direccionamiento de 256 bytes, un bus de datos de 16 bits y además utiliza una memoria cache write-back con 8 bloques, siendo el **tamaño** de cada bloque de 4 bytes.

1. Calcular la cantidad de memoria necesaria para implementar dicha cache.

2. Indicar **cuál** será el contenido de la **cache** y de la **memoria central** después de ejecutar el programa que se muestra en la memoria de **la figura**. El contador de programa contiene la dirección **6AH** y se supone que el procesador se para **después** de ejecutar la cuarta instrucción.
3. Calcular el tiempo que **tardará** en ejecutarse el programa, sabiendo que el **Miss time** es de **200 ns**, el **Hit time** es **30 ns**, el **tiempo que** el procesador tarda en ejecutar una instrucción **aritmética** es de **50 ns** y **que** el **tiempo de ejecución** de una **instrucción de movimiento de datos** es de **30 ns**.

4.8 BIBLIOGRAFÍA

- Computer Organization & Design. The Hardware/Software Interface. David A. Paterson, John L. Hennessy. Morgan Kaufmann Publishers, Inc. ISBN 1-55860-281-X. 1994.
- Computer Arquitectura: A Quantitative Approach. John L. Hennessy, David A. Paterson. Morgan Kaufmann Publishers, inc. ISBN 1-55860-069-8. 1990.
- High-Performance Computer Architecture. Harold S. Stone. Addison-Wesley. ISBN 0-201-52688-3. 1987.
- The Cache Memory Book. Jim Handy. Academic Press, inc. ISBN: 0-12-322985-5. 1993.
- Computer Organization and Architecture. Designing for performance. William Stallings. 4^a ed. Prentice-Hall International, inc. 1996. ISBN 0-13-394255-4.
- Computer Systems Design and Architecture. Vincent P. Heuring, Farry E Jordan. Addison Wesley Longman, Inc. 1997. ISBN 0-8053-4330-X.
- Operating System Concepts. A. Silberschatz, J. Peterson, L. Galvin. 3^a ed. Addison Wesley Longman, inc. 1991. ISBN 0-201-51379-X.

CAPÍTULO 5

INTERCONEXIÓN ENTRE PROCESADOR Y PERIFÉRICOS

5.1 INTRODUCCIÓN

En cualquier ordenador, además del procesador y del subsistema de memoria, existe una parte muy importante, llamada el subsistema de **Entrada/Salida (E/S)**, que hace posible la comunicación con el mundo exterior. Este **sistema está formado** por varios dispositivos periféricos que proporcionan un medio para intercambiar datos con el exterior y que se comunican con el procesador a través de una serie de módulos llamados de **E/S**. Cualquiera de estos módulos contiene una serie de controladores que se encargan de manejar el funcionamiento de uno o varios periféricos.

Los módulos de **E/S** no deben conectar directamente el periférico con el bus del sistema, sino que tienen que poseer una cierta inteligencia para poder realizar la comunicación entre el periférico y el procesador de forma eficiente. Si se observan algunas de las características del subsistema de **Entrada/Salida** será posible darse cuenta de esta necesidad:

- Existe una **gran** diversidad de periféricos que utilizan métodos de operación diferentes. No sería **lógico** que la CPU tuviera que incorporar **toda** la lógica necesaria para controlar este rango de dispositivos.
- La velocidad de transferencia de los datos de los periféricos es a menudo mucho **más** lenta que la que tiene el procesador con el sistema de memoria, por lo tanto resulta poco práctico usar el bus del sistema de alta velocidad para comunicarse directamente con los periféricos.

- A menudo los periféricos utilizan **formatos** y longitudes de palabra de datos diferentes a **los que utiliza** el procesador. Debe haber por tanto algún mecanismo para adecuar las señales de ambos dispositivos.

Los módulos de **WS** establecen una serie de reglas (llamadas *interface*) que les **permite**, por un lado **conectarse** con la CPU y la memoria bien a través del **bus del sistema**, bien a través de el bus de expansión, y por otro lado conectarse con los dispositivos periféricos a través de enlaces dedicados para datos.

Cualquier operación de **WS** se descompone siempre en varias fases bien diferenciadas:

1. **Fase de direccionamiento:** En esta fase se identifica al controlador del dispositivo objeto de la transferencia y se indican las operaciones que se desean realizar escribiendo un código especial en alguno de los registros del controlador. Todos los registros de cualquier controlador ocupan una dirección única en el sistema que dependiendo del tipo de procesador será en el mapa global de direcciones o en el caso de que se distinga entre posiciones de memoria y de E/S en el espacio dedicado a la E/S. Cuando el procesador **escribe** una dirección en el bus, todos los **controladores** la leen para **determinar** si se quiere acceder a alguno de sus registros. En ese caso se leen **también** las **líneas** de control para saber si se trata de una lectura o una escritura y se contesta al procesador.
2. **Sincronización y coordinación de las operaciones de E/S:** En esta fase se decide el momento exacto donde debe comenzar la transferencia de datos.
3. **Transferencia de datos:** Desde el controlador del periférico seleccionado al procesador (o directamente a la memoria) en el caso de una operación de lectura y al **contrario** para las operaciones de escritura.

En los **controladores** siempre existen una serie de registros **estándar**: el registro de control **para poder programar** el tipo de **operación** a realizar, el **registro de estado** que informa sobre el **resultado** de la **última** operación o la **disposición** del controlador para **recibir/transmitir** los datos y el **registro** de datos **que** almacena

temporalmente los datos que van a ser transferidos. De esta manera se facilita la **sincronización** entre el procesador y el periférico.

Actualmente, para operar con los periféricos se utilizan controladores más avanzados, muy parecidos a los procesadores convencionales, llamados **coprocesadores**. Los controladores tratan de realizar la mayor parte de las operaciones en paralelo con el objetivo de minimizar el tiempo que dedica el procesador a las operaciones de E/S. También se utilizan microprocesadores **estándar** como controladores de periféricos, recibiendo en este caso el nombre de procesadores de **WS**.

5.2 CLASIFICACIÓN DE LOS DISPOSITIVOS DE E/S

Las operaciones de **E/S** se realizan a través de una serie de dispositivos externos que suministran un medio para intercambiar datos con el mundo exterior. El dispositivo externo se conecta con el ordenador a través de un enlace o bus que va a uno de los módulos de **WS**. Este enlace sirve para programar las operaciones, intercambiar datos y para comprobar el estado del periférico y el resultado de la operación.

Los dispositivos periféricos son muy diversos, por lo que para clasificarlos se tendrán en cuenta:

- Comportamiento: Entrada, salida o almacenamiento.
- Destino de la **comunicación**: El destino de la comunicación puede ser un humano o una máquina.
- Tasa de datos: Tasa máxima de transmisión de datos entre el dispositivo y memoria principal o entre el dispositivo y el procesador.

En la tabla 5.1 se muestra una relación comparativa de los dispositivos más comunes de **Entrada/Salida**.

Dispositivo	Comportamiento	Conexión	Velocidad(Kb(seg.)
Teclado	Entrada	Humano	0.01
Ratón	Entrada	Humano	0.02
Micrófono	Entrada	Humano	0.02
Digitalizador	Entrada	Humano	200.00
Altavoz	Salida	Humano	0.60
Impresora de líneas	Salida	Humano	1.00
Impresora láser	Salida	Humano	100.00
Display gráfico	Salida	Humano	30000.00
Terminal por red	Entrada/Salida	Máquina	0.05
Red de Área Local	Entrada/Salida	Máquina	200.00
Disco flexible	Almacenamiento	Máquina	50.00
Disco óptico	Almacenamiento	Máquina	500.00
Cinta magnética	Almacenamiento	Máquina	2000.00
Disco magnético	Almacenamiento	Máquina	2000.00

Tabla 5.1: Tabla comparativa de dispositivos de Entrada/Salida

5.3 MÓDULOS DE ENTRADA/SALIDA

El módulo de **E/S** es responsable de controlar a uno o más dispositivos externos y de intercambiar **datos** entre estos dispositivos y la **memoria** principal o alguno de los registros de la CPU. Por lo tanto este módulo consta de dos **interfaces**: una **interna** al ordenador (a la CPU y a la memoria) y otra externa (a los **periféricos**).

Las funciones principales o los requerimientos de un **módulo** de E/S se recogen en alguna de las siguientes categorías:

- **Control y temporización:** Como la CPU se comunica con los **periféricos** de forma impredecible, dependiendo de las necesidades de **E/S** del *programa* en ejecución, los recursos internos se comparten para realizar una serie de actividades diferentes. Las actividades de E/S deben coordinar el flujo de datos entre los recursos internos y los dispositivos externos. Por ejemplo la transferencia de datos entre la CPU y un dispositivo **periférico** se realiza siguiendo los pasos siguientes:
 1. La CPU pregunta al **módulo** de **E/S** el estado del **periférico**.
 2. El modulo de E/S devuelve el estado.
 3. Si el módulo está operativo y a punto para realizar la transferencia, la CPU le suministra una orden al módulo de WS para pedir la transferencia de datos.
 4. El **módulo** de **E/S** obtiene una unidad de datos del dispositivo externo.
 5. Los datos se transfieren desde el módulo de **E/S** a la CPU.
- **Comunicación con la CPU:** Del escenario anterior se desprende la necesidad de que el **módulo** de **E/S** se comunique con la CPU. Para esta comunicación se requiere:
 1. Decodificación de la orden: Las órdenes se envían al módulo escribiendo en alguno de sus registros y el **módulo** las envía al periférico mediante el bus de control.
 2. Datos: Los datos se **intercambian** a través del bus de datos.

3. **Notificación** del estado: Ya que los **periféricos** son muy lentos es importante conocer el estado del **módulo** de E/S. El **módulo** utiliza **líneas** de estado (Ocupado, Listo y Error) para indicar su disponibilidad para realizar la transferencia.
 4. Reconocimiento de la **dirección**: Cada dispositivo de **Entrada/Salida** debe tener una **dirección** que debe reconocer el **módulo** que los controla.
- **Comunicación con el periférico:** Recoge el envío de **órdenes**, información de estado y datos.
 - **Almacenamiento temporal de datos:** Los datos que suministra la memoria se envían al módulo de **E/S** mediante **una** transferencia en ráfaga. Los datos se almacenan en el módulo y poco a poco se van enviando al **periférico**.
 - **Detección de errores:** El módulo de **E/S** es responsable de la detección de errores y de la notificación de los mismos a la CPU. Se detectan errores mecánicos, eléctricos o cambios en el patrón de bits **transmitido**.

5.4 PROGRAMACIÓN DE LAS ÓRDENES EN LOS DISPOSITIVOS

Para realizar una operación de WS, el procesador debe generar una **dirección** que **especifique** el **periférico** al que se quiere acceder dentro de un módulo de E/S y por el bus de datos se enviará la orden que se quiere ejecutar. La figura 5.1 muestra la **estructura e interconexión** genérica de una *interface* de **E/S**.

Como existen muchos **módulos** de WS, cada uno de ellos con varios periféricos, se utiliza un identificador único (una **dirección**) para cada uno de ellos. Cuando la CPU, la memoria y la **E/S comparten** el mismo bus se pueden diferenciar dos modos distintos de **direccionamiento**: **E/S mapeada** en memoria y **E/S** aislada.

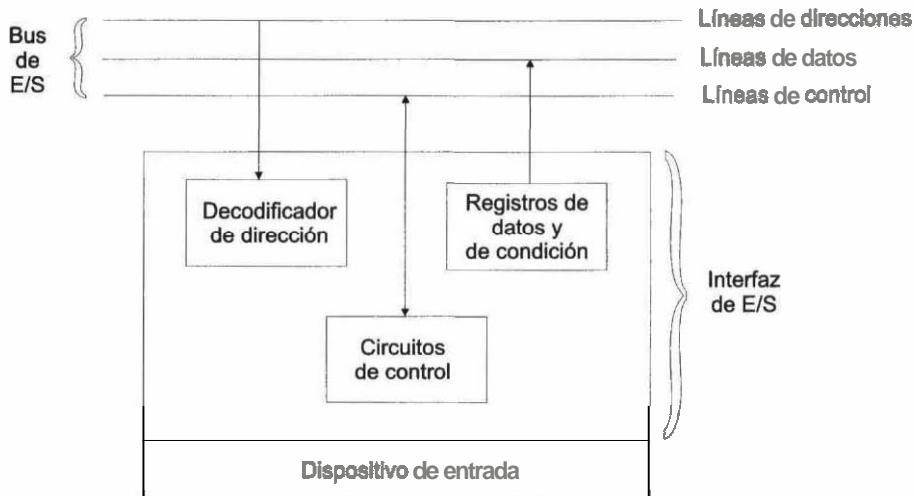


Figura 5.1: Estructura genérica de una interfaz de E/S

E/S mapeada en memoria

Existe un único espacio de direccionamiento que se comparte entre la memoria y los periféricos. Los registros de los controladores son tratados como si fueran posiciones normales de memoria. Las lecturas y escrituras a esas direcciones se interpretan como órdenes para el dispositivo.

La ventaja es que se reduce el juego de instrucciones, simplificando el diseño de la unidad de control, puesto que no se requieren instrucciones de E/S especiales. Además se puede utilizar todo el repertorio de instrucciones usado para acceder a la memoria con las operaciones de E/S, permitiendo una programación más eficiente. Las desventajas de este modo son que el diseño del mapa de memoria es más complejo, puesto que el circuito de decodificación de las **direcciones** utilizado para seleccionar a los distintos dispositivos y a la memoria requiere muchas más líneas. Esta técnica la utilizan los procesadores de la familia Motorola.

Para que el usuario no pueda acceder directamente a los registros de los controladores, el S.O. **prohibe** su acceso mediante los mecanismos de traducción de las direcciones. Si un usuario intenta acceder a uno de los registros se producirá un error de protección.

E/S aislada:

El bus del sistema dispone de líneas de lectura y escritura en memoria y de unas líneas adicionales para la lectura y escritura de los registros de los módulos de **E/S**. Por tanto se dispone de dos espacios de direccionamiento diferentes, el de memoria y el de WS.

Para poder acceder a un registro del controlador se utilizan instrucciones especiales del procesador que activan las líneas de WS. Para prevenir el acceso **directo** por parte del usuario a los controladores de periféricos, el S.O. prohíbe la ejecución de las instrucciones de WS cuando no se **está** en modo Kernel o modo Supervisor. Ejemplos de procesadores con instrucciones de E/S: son los de la familia INTEL 8086 y el IBM 370.

5.5 LA COMUNICACIÓN CON EL PROCESADOR. SINCRONIZACIÓN

La aparición de eventos del exterior es impredecible. El procesador necesita algún medio para **sincronizarse** con estos eventos y así poder administrar las transferencias de WS.

El método más simple consiste en que la CPU ejecuta un programa que espera a que el periférico tenga los datos listos o esté dispuesto a recibirlas, y entonces realiza una transferencia explícita de un dato. De este modo el **procesador** está esperando mucho tiempo, sobre todo cuando los periféricos son lentos. Este tiempo podría ser aprovechado por el procesador para hacer trabajo útil.

El siguiente método consiste en que el periférico avise al procesador de su disponibilidad para realizar la transferencia, enviándole una interrupción. De esta forma el procesador utiliza mejor su tiempo, pero sin embargo todavía se requiere la acción de la CPU para transferir los datos.

El método más eficiente es utilizar un dispositivo especial de acceso directo a memoria que se encarga de realizar una transferencia de un bloque de datos directamente entre el **periférico** y la memoria. El procesador **sólo** interviene al principio y al fin de la transferencia.

5.5.1 Sincronización por prueba de estado

Cuando el procesador está ejecutando un programa y encuentra una operación & WS, ejecuta esa **instrucción** enviando una orden con la operación a realizar al controlador del periférico apropiado. Si la **sincronización** es por prueba de estado, el controlador realizará la acción con el periférico y activará los bits correspondientes en el registro de estado una vez terminada, no realizando ninguna acción posterior para **avisar** al procesador. Por lo tanto es responsabilidad de la CPU el comprobar periódicamente, mediante la ejecución de un programa, el estado del **periférico** para determinar si el periférico está listo para aceptar o enviar los datos. De esta manera la **CPU** podrá dirigir la transferencia de los datos.

Cuando el dato se encuentre en el *buffer* de datos, el **procesador** lo leerá y lo guardará en uno de sus registros. Si el dato aún no está listo, el procesador deberá esperar e intentar de nuevo. La gran desventaja es que el procesador utiliza innecesariamente mucho tiempo leyendo los registros de estado en las consultas sin **éxito**. Cuando se completa la operación se deberá comprobar el registro de estado para detectar los posibles **errores**.

Si varios **periféricos** **pueden** estar activos al mismo tiempo, se deberá realizar un **escrutinio (polling)** sobre todos los registros de los **periféricos** para determinar cuál de ellos **está** listo. Los **manejadores** de dispositivo (*device handlers*) son rutinas que realizan **estas** tareas, existiendo una **distinta** para cada dispositivo.

5.5.2 Sincronización por interrupción

Para solucionar la sobrecarga del sistema debido al **escrutinio**, el **procesador** envía la orden de E/S a un **módulo** y a continuación pasa a realizar otra tarea útil. Cuando el módulo esté **listo** para **realizar** la operación **interrumpirá** a la **CPU** y entonces ésta realizará la transferencia de datos, como en el caso anterior, continuando con la tarea que realizaba después de terminar. De esta manera cuando el **procesador** desea **realizar** una transferencia **programada**, se evita el esperar a que el periférico esté listo para efectuar la transferencia. Las acciones que se realizan ante la llegada de una **interrupción** son:

1. El periférico envía la interrupción activando una línea de control del bus conectada a la CPU, la línea de petición de interrupción (INT o INTR).
2. La CPU finaliza la instrucción en curso.
3. La CPU comprueba si hay peticiones de interrupción pendientes, y en caso afirmativo, envía una señal de reconocimiento al dispositivo que mandó la interrupción.
4. La CPU detiene la tarea que estaba realizando y guarda la suficiente información para poder continuar después. Es decir el registro contador de programa y la palabra de estado del procesador.
5. La CPU ejecuta la rutina de servicio de interrupción que atiende al periférico cargando en el PC la dirección de comienzo de esta rutina. Si hay más de una rutina el procesador deberá determinar cuál ejecutar. Esta información se puede incluir en la señal original o tiene que ser el procesador el que se la pida al dispositivo que mandó la interrupción. Si se producen dos interrupciones al mismo tiempo, el procesador tratará primero la que tenga mayor prioridad, que normalmente estará asignada al dispositivo de mayor velocidad.
6. Dentro de la rutina el procesador salva el estado de todos los registros del procesador que vaya a modificar para pasar después a realizar la E/S.
7. Cuando termina la **rutina**, antes de retomar, se restaura el valor de los registros y se ejecuta la instrucción de retorno de **interrupción** que restaura a su vez el PC y el registro de estado. Como resultado, la siguiente instrucción que se ejecuta pertenecerá al programa que fue interrumpido.

Las ventajas que presenta este **método** son la respuesta rápida y que el programador no tiene que utilizar rutinas de **muestreo** del estado del periférico. El inconveniente es que a pesar de obtener una mayor velocidad en la respuesta aún no es un método lo suficientemente rápido, puesto que cada dato transferido desde la memoria al periférico aún tiene que pasar por la CPU.

Identificación del dispositivo que pidió la interrupción

Cuando hay varios **periféricos** conectados al computador y el procesador recibe una interrupción, antes de atender la petición debe identificar cuál es el

periférico solicitante. Además, si se realizan varias peticiones de interrupción en el mismo instante, siempre se deben atender de forma priorizada. Hay diversos métodos para determinar cuál ha sido el periférico fuente de la interrupción, y en caso de que haya **varias interrupciones** determinar la más prioritaria, los métodos más comunes son: mediante exploración secuencial, método (*daisy-chain*) y mediante **líneas** individuales. Estos métodos serán descritos con mayor detalle en el capítulo 6.

Enmascaramiento

De forma selectiva se pueden inhibir las peticiones usando una máscara de **bits**. El procesador tiene un registro de enmascaramiento donde están estos **bits**. Hay que tener en cuenta que hasta que la rutina de tratamiento no desactive la petición de la interrupción, habrá que **deshabilitar** las interrupciones para que no se detecte a sí misma. Esto es debido a que las interrupciones se pueden anidar, es decir, que dentro de una subrutina de tratamiento de interrupción se pueden producir más interrupciones. Entre los controladores integrados más populares está el 8259 de **Intel** que utiliza interrupciones vectorizadas.

5.6 TRANSFERENCIA DE DATOS ENTRE UN DISPOSITIVO Y MEMORIA

En los métodos analizados el procesador pasa los datos mediante transferencia programada, a través de instrucciones ya sea de WS o de memoria. Estas técnicas son buenas para dispositivos lentos en los que es interesante tener *interfaces* sencillas.

Sin embargo, para dispositivos con mayor rendimiento se utilizan mecanismos de transferencia de datos controlados por interrupciones. En este caso el S.O. puede trabajar en otras tareas mientras los datos se leen o escriben en el dispositivo. Estas transferencias permiten que el procesador no tenga que esperar a los eventos de **E/S**, por lo tanto no hay sobrecarga en ausencia de operaciones de **E/S**.

Algunos **procesadores** tienen instrucciones para mover bloques de datos que se pueden usar en las operaciones de **E/S** en lo que es conocido como transferencias de bloques de datos. Los datos se **pueden** transferir de esta manera de **forma** más **rápida que** en las **E/S programadas** o dirigidas por **interrupciones**. La CPU tiene que **sincronizar** el movimiento del bloque con el periférico y no puede realizar ninguna otra función mientras **progesa** el movimiento del bloque. Esta técnica no se suele utilizar normalmente y **sólo** es apropiada para **periféricos** muy rápidos en los **que** su velocidad sea **comparable** con la del **procesador**.

En consecuencia, el **procesador** tiene que perder mucho tiempo **realizando** la transferencia de datos. Sería más deseable evitar usar la **CPU completamente** para **realizar** operaciones de WS. El objetivo de toda transferencia **es** transferir los datos desde el **periférico** a la memoria para poder ser procesados posteriormente. Esto requiere la generación de las direcciones apropiadas de memoria y del controlador y activar las líneas de control del bus para realizar la transferencia. Para realizar esto, es posible **añadir** cierto **hardware** para **implementar** estas funciones de forma que es el controlador del dispositivo el que transfiere los datos **directamente** a la memoria sin que intervenga el **procesador**.

Este mecanismo se llama Acceso Directo a Memoria (**Direct Memory Access** o **DMA**). En este caso también se usan **interrupciones** para comunicarse con el procesador, pero sólo en caso de **error** o cuando se acaba la transferencia.

La transferencia DMA la realiza **un** dispositivo especial. Este controlador **debe** ser capaz de actuar como maestro **del bus del sistema** para poder realizar la transferencia. Cuando la CPU quiere realizar una transferencia, accede al controlador de DMA como si **fuerá** el controlador de un dispositivo típico de **E/S**. En la figura 5.2 se muestra la interconexión típica de un controlador de DMA. **Este** incorpora los siguientes registros:

- 『 R. de dirección de memoria: Almacena la posición en memoria donde va el primer dato.
- 『 R. contador: Indica la cantidad de datos a transferir.
- 『 **Buffer** de datos: Es una memoria de almacenamiento secundario. Puede **ser** un registro o una **fifo**.
- 『 R. de estado: Donde se almacena el estado de la operación.
- 『 R. de control: En **él** se escriben las **operaciones** a realizar.

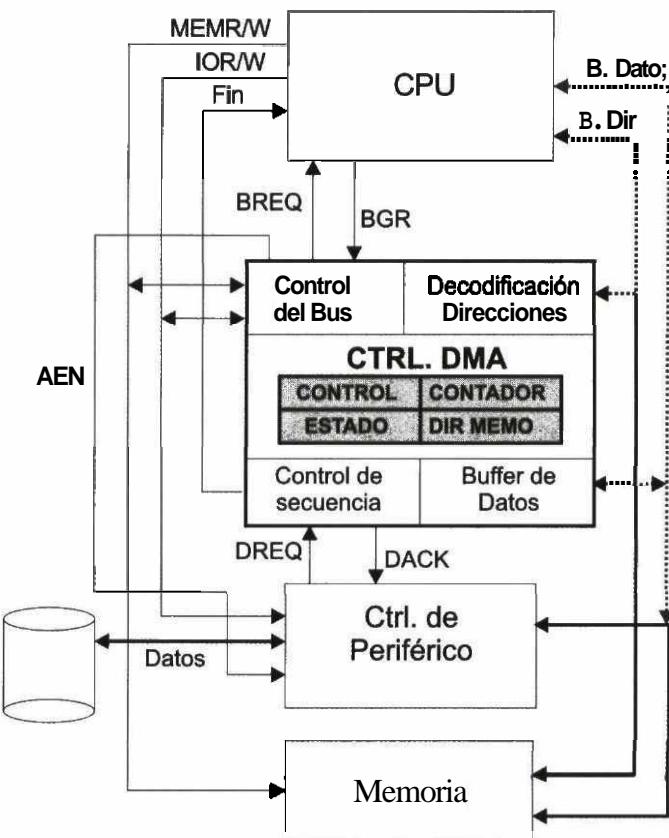


Figura 5.2: Estructura e interconexión del controlador de DMA

Toda transferencia se realiza mediante las operaciones:

1. La CPU inicializa el controlador de periférico para realizar la operación. También inicializa el controlador de DMA, indicando el número de bytes a ser transferidos, la dirección de memoria donde se inician los datos y el tipo de operación.
2. El controlador de DMA espera a que el **periférico** le indique que los datos están listos para realizar la transferencia (normalmente mediante una interrupción).
3. Cuando ya están listos los datos, el controlador de DMA inicia la operación y solicita el bus al procesador. (A diferencia de las interrupciones,

aquí la **CPU** puede responder inmediatamente, ya que no tiene que cambiar su estado interno).

4. Cuando el controlador de DMA posee el bus coloca la dirección de memoria de **comienzo** del bloque en el registro de direcciones, reconoce la petición del controlador del periférico y pide los datos al periférico.
5. El controlador de periférico activa las **señales** apropiadas de control sobre el bus y suministra los datos. Cuando los datos son capturados por la memoria el controlador de DMA se lo indica al **periférico**.
6. El controlador del periférico niega la **línea** de petición de transferencia y espera a que el siguiente dato del periférico esté disponible. El controlador de DMA devuelve el bus, **incrementa** el registro de comienzo de las direcciones y compara la dirección de comienzo con la de **fin**. Si son iguales, la transferencia **termina** y el controlador de DMA **interrumpe** a la CPU para que compruebe si todo ha ido bien. Si no son iguales, se espera a que **llegue** la siguiente petición de transferencia del controlador del periférico.

5.6.1 Arbitraje del bus y transferencias de datos

La CPU y el controlador DMA compiten por la **utilización** del bus **procesador-memoria**. El controlador DMA tiene que tener una prioridad muy alta en la petición de bus para evitar el tener que realizar esperas demasiado prolongadas. Existen tres tipos de esquemas diferentes:

1. Arbitraje por robo de ciclo: El controlador de **DMA** realiza un ciclo de acceso a memoria cada vez que quiere transferir un dato. Para cada dato se pide el bus, se **utiliza** y **después** se libera, de forma que la **CPU** y el controlador de DMA se **multiplexan** el bus en el tiempo. En las transferencias por bloques únicamente se pide el bus una vez, ya que el controlador de DMA no lo devuelve hasta que no ha terminado completamente con el bloque. Existe además un método de transferencia de datos bajo demanda que consiste en una transferencia en bloques supervisada por el **periférico**. El periférico puede avisar al controlador de **DMA** para que detenga la transacción **si** así lo requiere.

2. Acceso transparente sin arbitraje: En un ciclo de memoria se da el tiempo suficiente para que acceda tanto el procesador como el DMA. En primer lugar el procesador activa sus **drivers** de bus y después lo hace el controlador de DMA. Este método necesita de **circuitería** especial para poder ser implementado, puesto que al no haber arbitraje se pueden **producir** contenciones de bus si ambos dispositivos tratan de acceder a la vez.
3. **Utilización** de memorias **multipuerto**: La memoria principal del sistema es especial y tiene varios puertos de **direcciones**, datos y control. Cada dispositivo utiliza **buses** diferentes y por lo tanto se eliminan los **conflictos** en el acceso, pudiéndose hacer transferencias **simultáneas**.

El primer método es el más utilizado, puesto que el **diseño** es bastante sencillo y no requiere la utilización de **circuitería** especial ni de **memorias** de doble puerto.

5.7 CANALES Y PROCESADORES DE E/S

Para reducir la necesidad de interrumpir al procesador y ocuparlo en el manejo de **interrupciones** de **WS**, se pueden **diseñar** los módulos de **WS** de forma que exista un procesador en el módulo que ejecute instrucciones especiales para realizar operaciones de WS. Al módulo de **WS** se le llama canal de E/S. La CPU construye en memoria un **programa** de E/S, también llamado programa de canal, para este procesador. La transferencia se inicia indicando al canal que ejecute el programa. Éste especifica los dispositivos, las áreas de memoria para el almacenamiento, la prioridad y las acciones a realizar en caso de **error**. El canal va siguiendo estas instrucciones y controlando la transferencia. Hacen un uso intensivo del acceso directo a memoria, pudiendo compartir el controlador de DMA entre varios dispositivos.

Si se utiliza esta técnica la CPU **sólo** es interrumpida cuando se termina de realizar la secuencia de operaciones de **WS**, con lo que el procesador interviene muy poco en la realización de la operación.

Si además del procesador el módulo de **WS** contiene memoria local, se le llama procesador de E/S. Con esta arquitectura se puede controlar un *gran número* de dispositivos de E/S sin la **intervención** de la CPU.

A diferencia de los circuitos integrados de DMA que son **procesadores** especiales, los **procesadores** de **E/S** se **implementan** con procesadores normales de propósito general.

5.8 CONCLUSIONES

En este capítulo se han descrito los conceptos y las operaciones que tienen lugar cuando se realiza una transferencia de **Entrada/Salida**. No hay que olvidar que el **subsistema** de **Entrada/Salida** compone uno de los grandes pilares de la arquitectura del computador, junto con el procesador y la memoria, por **ello** es de vital importancia realizar un **diseño** eficiente del mismo. El ordenador se comunica con el mundo **exterior** a través de los **periféricos**. No es necesario disponer de un sistema demasiado eficiente para **cubrir** las necesidades de comunicación de los usuarios, pero sin embargo sí que es preciso de disponer de enlaces de alta velocidad para conectar unas máquinas con **otras**.

Si se analiza el **desarrollo** en los últimos años de los sistemas **informáticos**, los avances en **microeléctronica** han hecho posible la **implementación** de **procesadores** que funcionan a frecuencias muy elevadas (del orden del GHz). Este hecho supone que las necesidades de alimentación del procesador con **instrucciones** o con datos se han incrementado considerablemente, con lo que el subsistema de WS tiene que evitar que el procesador esté parado en espera de la llegada de la información. En los ordenadores personales el cuello de botella lo constituye el acceso al disco, y en los sistemas distribuidos cada vez es más necesario disponer de un mayor ancho de banda en las comunicaciones.

Después de describir las características de las operaciones de **Entrada/Salida**, en este capítulo se han clasificado los **periféricos** más importantes. Lo esencial es comprender las fases **por** las que pasa una operación de **Entrada/Salida**: Rítmico se programan las operaciones a realizar en los controladores escribiendo directamente sobre los registros de control, que tienen posiciones únicas en el mapa de memoria. Después, una vez que el controlador ya dispone de los datos del periférico, se debe **sincronizar** con el procesador para culminar la operación con la realización de la transferencia de los datos hacia alguna posición de memoria.

De vital importancia son los procesadores de E/S que hacen uso intensivo de los canales de acceso directo a memoria. Con los sistemas operativos multitarea y los grandes sistemas multiusuario lo importante es que el procesador no se detenga para que pueda ejecutar los procesos de otros usuarios que no están esperando datos. Con el uso de estos módulos, el procesador apenas interviene en la ejecución de estas operaciones.

5.9 CUESTIONES

5.1 ¿En qué consisten las transferencias programadas realizadas por el procesador? Indicar los tipos que se conocen.

5.2 En una transferencia de **entrada/salida**, indicar la **fase** en donde están activos: el **decodificador** de direcciones, el registro de orden, los registros de datos y el registro de control, indicando además cuáles son las funciones de cada uno de ellos.

5.3 ¿Para qué sirve el ciclo de reconocimiento de **interrupción** del procesador?

5.4 ¿Qué ventajas tiene usar **DMA** para realizar una transferencia de datos? ¿Qué es mejor; utilizar **DMA** o un canal? Razonar la respuesta.

5.5 Enumerar los métodos que el procesador puede emplear para transferir un bloque de datos desde un controlador de periférico a una zona de la memoria.

5.6 Calcular el ancho de banda que tiene un sistema de memoria cuando se transfiere un bloque de **8 palabras** de 16 bits cada una, suponiendo que se tarda un ciclo de reloj en enviar las direcciones, el tiempo de acceso de la memoria RAM es de 5 ciclos de reloj y el tiempo de transferencia de los datos por el bus es de 1 ciclo de reloj. Suponer tres configuraciones diferentes:

- Bus de datos de 8 bits

- *Bus de datos de 16 bits*
- *Memoria interpolada con dos bancos de 8 bits.*

5.7 *¿Cómo se programan las órdenes en los dispositivos de E/S?*

5.8 *¿Qué ventajas presenta la sincronización por interrupción frente a las sincronización por prueba de estado?*

5.9 *¿Por qué dispone un controlador de DMA de líneas de arbitraje de bus?*

5.10 *Cuando un dispositivo envía una petición de interrupción al procesador, ¿Qué métodos utiliza el procesador para identificar al dispositivo que ha producido la interrupción?*

5.11 *Explicar cómo se realiza una transferencia de datos utilizando DMA. ¿Qué registros debe tener el controlador de DMA?*

5.10 BIBLIOGRAFÍA

- **Computer Organization & Design. The Hardware/Software Interface.** David A. Paterson. John L. Hennessy. Morgan Kaufmann Publishers, Inc. ISBN 1-55860-281-X. 1994.
- **An introduction to Microcomputer Systems. Architecture & Interfacing.** John Fulcher. Addison Wesley Publishers Ltd. ISBN 0-201-41623-9. 1989.
- **The Indispensable PC Hardware Book.** Hans-Peter Messmer. 3^a ed. Addison Wesley Longman, inc. 1997. ISBN 0-201-40399-4.
- **Computer Arquitecture: A Quantitative Approach.** John L. Hennessy, David A. Paterson. Morgan Kaufmann Publishers, Inc. ISBN 1-55860-069-8. 1990.

- **High-Performance Computer Architecture.** *Harold S. Stone*. Addison-Wesley. ISBN 0-201-52688-3. 1987.
- **Computer Organization and Architecture. Designing for performance.** *William Stallings*. 4^a ed. Prentice-Hall International, Inc. 1996. ISBN 0-13-394255-4.
- **Computer Systems Design and Architecture.** *Vincent P. Heuring, Farry E Jordan*. Addison Wesley Longman, Inc. 1997. ISBN 0-8053-4330-X.



CAPÍTULO 6

BUSES

6.1 INTRODUCCIÓN

En los capítulos anteriores se ha expuesto la estructura básica de un computador, en este esquema junto a los elementos principales (CPU, memoria y E/S) aparecen caminos de comunicación a través de los cuales se intercambia información (instrucciones y datos). Este intercambio de información se produce de forma continua entre el procesador y la memoria, para acceder a las instrucciones y realizar las lecturas y escrituras de los datos. De manera adicional, el procesador también realiza un intercambio de información con el exterior a través de los dispositivos de **E/S** y los controladores de DMA acceden a la memoria para intercambiar bloques de datos con algún controlador de forma autónoma. Toda esta comunicación entre dispositivos se realiza mediante la transmisión de señales **eléctricas** a través de cables o de pistas de un circuito impreso. Cada línea es capaz de transmitir un nivel de tensión, que representa un '1' o un '0' binario. Esta estructura forma lo que se denomina bus, donde además de las características eléctricas y físicas también se definen un conjunto de protocolos que hacen posible la comunicación entre los diferentes subsistemas.

Habitualmente los **buses** están compartidos por los dispositivos. Es, por tanto, necesaria la **implementación** de mecanismos que aseguren que nunca va a haber más de un dispositivo controlando el estado (el nivel de tensión) de las líneas del bus. Si varios dispositivos tratan de transmitir información por el mismo bus en el mismo instante de tiempo, las señales se solaparían produciéndose

un error llamado contención del bus. En todas las transferencias siempre existe un elemento emisor, o dispositivo origen, que envía información a uno o varios elementos receptores o dispositivos destino.

Todos los elementos conectados a un mismo bus deben ser capaces de interpretar de manera correcta la información que se transmite por el bus. Para conseguir que diferentes módulos de distintos fabricantes puedan interactuar entre ellos sin problemas, todos los buses deben estar estandarizados, cumpliendo una serie de requisitos eléctricos (niveles de tensión en el bus), mecánicos (tipos de conectores) y de protocolo (cómo se envía la información). De manera adicional, la estandarización tiene otras ventajas como son la flexibilidad y el bajo coste, ya que si se define un único sistema de conexión con los mismos conectores para todos o para una conjunto grande de dispositivos, se podrá intercambiar dispositivos entre diferentes ordenadores o añadir nuevos.

Por otro lado, el principal inconveniente que plantea el uso de buses para la interconexión de los elementos de un computador es que puede convertirse en un cuello de botella que limite las prestaciones del sistema. Esta situación se produce si las necesidades que tienen los diferentes elementos conectados a un mismo bus (CPU, memoria y E/S) de transferir información, supera la capacidad de transferencia, produciéndose múltiples estados de espera que degradan el rendimiento del sistema. La máxima velocidad en las comunicaciones vendrá dada por el ancho de banda del bus que indica la cantidad de datos que se pueden transferir por unidad de tiempo (habitualmente en bytes/s). Si la velocidad del bus no es suficiente para evitar la degradación del rendimiento, se puede aumentar el número de buses del sistema construyendo una jerarquía que permita realizar transferencias simultáneas.

6.2 CARACTERÍSTICAS DE UN BUS

Todas las características que tiene un bus y que lo diferencian de otros buses se recogen en un documento llamado estándar del bus. Dependiendo del ámbito de aplicación, será conveniente elegir un bus con unas características determinadas que permita realizar mejor todas las transferencias. Las características básicas que define el estándar son el nivel físico, número y función de las líneas eléctricas y el modo de operación.

Nivel físico

La primera propiedad de los **buses** es el soporte físico que utiliza. Dentro de la estructura jerárquica de los **buses** que tiene un sistema informático, hay **buses** que conectan dispositivos rápidos cuyas líneas son de reducida longitud y están formadas por pistas del circuito impreso. Estas **líneas** conectan **directamente** pines de los circuitos integrados, ya sean del procesador, la memoria o de algún controlador. En contraposición, los **buses** que enlazan externamente con los **chips periféricos** se componen a menudo de múltiples hilos, recogidos en mangueras o en los denominados cables planos. En una zona intermedia se sitúan los **buses** de expansión que conectan diferentes placas de circuitos impresos. En estos casos se dispone de un circuito impreso especial que contiene las líneas del bus y diversos conectores dispuestos de forma regular. A estos conectores se les denomina ranuras o **slots**. Los circuitos impresos normalmente se colocan sobre una estructura metálica con guías que permite la inserción de otras tarjetas y la colocación de fuentes de alimentación. La estructura recibe el nombre de **rack** y todas las tarjetas conectadas a la misma estructura comparten el mismo bus. Cada uno de los componentes del sistema ocupa una o varias ranuras de esta estructura. Para que todas las tarjetas sean compatibles, el **estándar** del bus debe definir perfectamente la forma y el tamaño de los conectores, tanto de las ranuras como de las placas que se introducen en la estructura.

La longitud de las líneas de un bus y el número de dispositivos que se conectan a **él** está limitado. En función de los valores de estos parámetros el dispositivo que fija el estado de las líneas debe suministrar más o menos comente al bus. Para poder conectar cualquier dispositivo se utilizan circuitos integrados especiales llamados **drivers** de bus que convierten las señales eléctricas locales de los integrados en señales adecuadas al **estándar** del bus. Estos integrados suministran la suficiente intensidad de comente para que la señal se reciba correctamente en el otro extremo. También se caracterizan por consumir poca comente de entrada y tener márgenes de tensión apropiados para evitar el ruido cuando actúan como receptores. Un ejemplo es la familia **BTL** (**Backplane Transceiver Logic**) que utiliza niveles de tensión entre 1'9 y 2'1 voltios para transmitir un 1 lógico y entre 0'75 y 1'10 voltios para el 0 lógico.

La longitud de las pistas individuales que forman un bus en un circuito impreso puede no ser la misma. Este hecho no tiene importancia para bajas frecuencias, pero llega a ser un factor critico a medida que aumenta la frecuencia

de funcionamiento del sistema. **Las** pequeñas diferencias en la longitud de las pistas provoca pequeñas diferencias **temporales** de propagación de la señal. Este retraso de las señales se denomina **skew** y puede producir la pérdida de **sincronización** entre varios dispositivos. Este problema es especialmente crítico en la señal de reloj y para evitarlo se incluyen **drivers** especiales, que generan la misma señal de reloj pero con diferentes retrasos para que llegue al mismo tiempo a todos los dispositivos.

Por último, también es necesario **especificar** los niveles de tensión que acepta el bus y los tipos de salida de los **drivers** de bus. Esta **especificación** es necesaria, para que los dispositivos **conectados** no **interfieran eléctricamente** sobre el nivel de tensión de las señales del bus, únicamente aquel que posea permiso en un determinado instante debe controlar el nivel **eléctrico** de las señales. Esto se consigue utilizando los siguientes tipos de salida en los **drivers** de bus:

1. **Totem-Pole:** Es el tipo de salida que siempre está activa, tomando comente cuando está a nivel bajo y suministrando comente en el caso contrario. Se utilizan para las **señales que sólo** tienen conectado un **driver** por línea como son: las señales de **interrupción** o de arbitraje encadenadas **daisy chain** que se describirán a continuación.
2. OC (Colector Abierto): El driver toma comente cuando está **a nivel** bajo, pero no suministra cuando está a nivel alto. Debe incluirse en la línea **una resistencia de pull-up** para conseguir el estado alto **definido** cuando ningún driver está activo. Como el estado de la línea cambia cuando uno o **más** drivers se activan (enviando un nivel bajo) se dice que implementa una or-cableada. Se utilizan para **señales** que pueden ser activadas por varios dispositivos a la vez (líneas de petición de interrupciones o de petición de bus). Estas puertas se usaron mucho en **buses** antiguos, pero debido a su alto consumo de **corriente** es preferible usar puertas **triestado**.
3. TS (**Triestado**): Funcionan de forma similar a las **totem-pole**, pero se diferencian en que se pueden desactivar pasando la línea a un estado de alta impedancia. Para ello tienen un transistor que habilita o deshabilita la salida, dependiendo del estado de una entrada de control. Se utilizan para líneas que pueden ser activadas en varios puntos diferentes del bus (líneas de direcciones o de datos) pero nunca simultáneamente.

Líneas

Una de las propiedades que describe el **estándar** es el número de líneas eléctricas que posee el bus, que pueden ser desde solamente dos hasta a varios cientos. Cada una de las líneas tiene asignada una función o un significado en particular y en general se pueden agrupar en los siguientes cuatro tipos de líneas:

1. Alimentación: $\pm 12\text{ V}$, $\pm 5\text{ V}$, $\pm 3\text{ V}$ y tierra (GND).
2. Datos: Llevan información (pueden ser instrucciones, datos o paridad) entre los módulos del sistema. El número de líneas representa el ancho del bus de datos y por tanto define la cantidad de **información** que se puede transmitir en paralelo en un instante de tiempo **determinado**, influyendo enormemente en el rendimiento del sistema.
3. Direcciones: Se utilizan para designar el origen o el destino de los datos que hay en el bus. Sirven para habilitar dispositivos o posiciones de memoria y su anchura (número de líneas) determina la máxima capacidad de memoria física del sistema.
4. Control: Transmiten información sobre el tipo de ciclo que está teniendo lugar (lectura, escritura...), información de sincronización (información temporal que indica la validez de la información en las líneas del bus), **interrupciones** y señales de arbitraje. Entre las distintas líneas de control típicas están:
 - Escritura y lectura de memoria,
 - Escritura y lectura de E/S,
 - Reconocimiento de una transferencia.
 - Petición y concesión de bus.
 - Petición y reconocimiento de interrupción.
 - Reloj.
 - Reset.

Existen **buses** donde las mismas líneas físicas se utilizan para llevar información tanto de datos como de direcciones. El ciclo de bus se divide en dos,

en primer lugar se envían las **direcciones** y se validan. A continuación, por las mismas **líneas** se envían o se leen los datos dependiendo del tipo de ciclo. A los **buses** que presentan esta característica se les denomina **buses** multiplexados y tienen la ventaja de utilizar menos **líneas**, lo que abarata su coste. Como desventaja se tiene que el control del bus es más complejo.

Modo de operación

El **estándar** también regula el modo de **operación** del bus que indica la forma en la que se realiza una transferencia, incluyendo protocolo, **ordenación** y **temporización** de las **señales**. Se distingue entre **buses** **síncronos**, donde todas las acciones se realizan en instantes de tiempo determinados en **función** de la señal de reloj perteneciente al **bus**, y **asíncronos**, que utilizan **señales** de control para indicar la **realización** de determinadas acciones.

Dentro de los elementos conectados se pueden diferenciar los maestros y los esclavos de bus. Un dispositivo maestro es aquel capaz de iniciar una transferencia de datos en el bus, mientras que un esclavo **sólo** puede enviar datos como respuesta a una **peticIÓN** de un maestro. Hay **buses** en los que se permite la existencia de **varios** maestros de bus. Como dichos dispositivos **se** conectan en cualquier instante de tiempo, para evitar la **contención** del bus, **sólo** uno de ellos puede estar seleccionado en cada momento.

6.3 JERARQUÍA DE BUSES

Por regla general cada uno de los dispositivos conectados a un bus tendrá una determinada velocidad de funcionamiento. **También** es habitual que la necesidad de ancho de banda en las transferencias que realiza **cada** dispositivo sea diferente. Dado que la **incorporación** de elementos lentos **puede** **retrasar** las transferencias de los elementos más rápidos, necesitados de mayor ancho de banda, los dispositivos dentro de un **computador** se suelen **agrupar** por sus velocidades y necesidades de ancho de banda. Los dispositivos dentro de un **grupo** comparten un mismo bus y el conjunto de todos los **buses** forman una **jerarquía** dentro del sistema. Con la separación de los **buses** se consigue reducir los retrasos de las señales, al reducir la capacidad de las **líneas** debida a la conexión de

múltiples elementos. También se reduce el riesgo de que el bus pueda llegar a constituirse como un cuello de botella del sistema si la demanda de transferencia de datos alcanza el ancho de banda máximo del bus. Un esquema de la jerarquía de **buses** se muestra en la figura 6.1. Normalmente en una jerarquía de **buses** se encuentran los siguientes niveles:

- **Bus del procesador:** Se utiliza para conectar el procesador a la **cache** o a una serie de dispositivos muy específicos y rápidos. Tiene poca longitud y alta velocidad, y utilizan las propias señales del procesador, por lo que son específicos para cada sistema.
- **Bus local:** A medida que se incrementa la velocidad de los dispositivos de **E/S**, sobre todo los **terminales** de vídeo, se necesita un bus de alta velocidad estrechamente ligado al sistema para permitir transferencias a gran velocidad. Este bus es independiente del procesador, por lo que diferentes sistemas con procesadores distintos pueden usar las mismas especificaciones. Si el sistema de memoria está conectado a este bus también se le puede denominar bus de memoria.
- **Bus de expansión:** Conecta los **controladores** de E/S más lentos con el procesador. A este bus se conectan muchos dispositivos, por lo que suele tener un ancho de banda variable. Normalmente suelen estar **estandarizados** para que se puedan acoplar múltiples dispositivos distintos. A esta clase de **buses** también se les denomina **buses de E/S**.
- **Bus del sistema:** Conecta las diferentes tarjetas procesadoras y periféricas que forman un mismo sistema. Habitualmente un computador personal se caracteriza por estar compuesto de una placa base donde se dispone de ranuras de expansión para conectar periféricos. Estos sistemas tienen una flexibilidad limitada y no son fáciles de expandir. Otros sistemas más complejos (como **multiprocesadores**) están **compuestos** por múltiples tarjetas que se conectan entre si mediante un bus único, el bus de sistema. Estos **sistemas** son más flexibles y escalables y pueden estar compuestos por tarjetas de distintos tipos y de diferentes **fabricantes**.

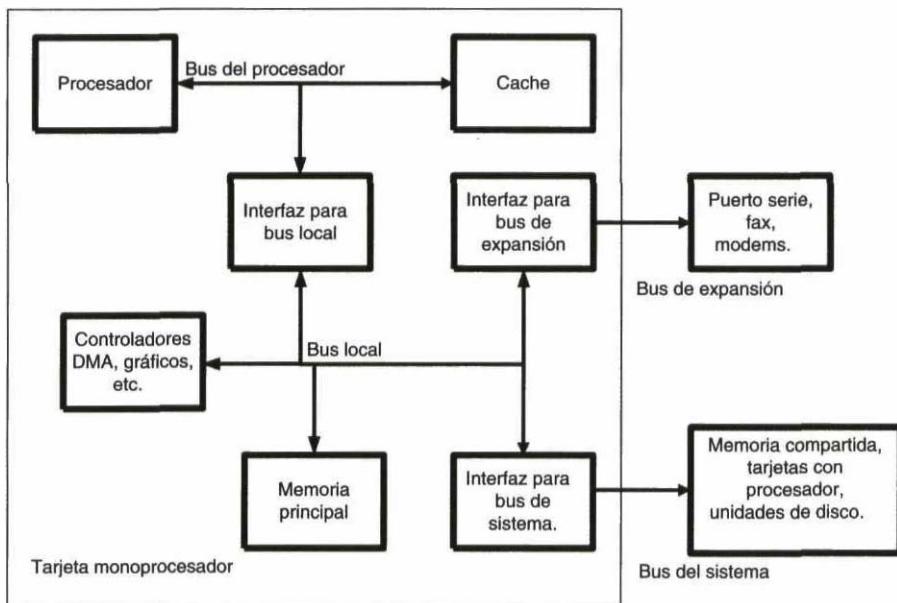


Figura 6.1: Esquema de la jerarquía de buses

6.4 PROTOCOLOS DE BUS

Un protocolo de bus describe el conjunto de reglas mediante las que **se** ponen de **acuerdo** el maestro de bus y **el** esclavo, **para realizar** una transferencia. Estas transferencias, atendiendo a la **sincronización** de los eventos en el bus, **se** pueden clasificar en:

Transferencia síncrona

En las transferencias **síncronas** todos los eventos tienen lugar en un instante de tiempo específico, **sincronizadas** con una señal de reloj incluida entre **las** líneas de control del bus. Cualquier transferencia consta de un número fijo de ciclos de reloj, por lo que los datos permanecen en el bus un determinado intervalo de tiempo. Este instante en el que los datos están en el bus es el momento en el cual deben de ser leídos por el dispositivo receptor. Un **cronograma** con este tipo de transferencia se muestra en la figura 6.2

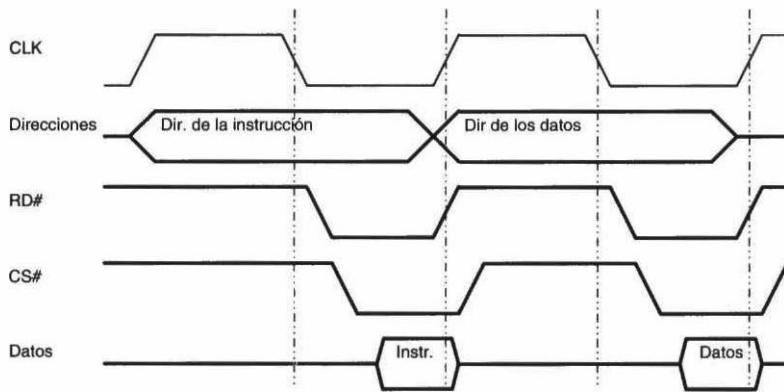


Figura 6.2: Transferencia síncrona

En el ejemplo se realiza una lectura de memoria. En el borde de subida de la señal reloj del primer ciclo se transmite la dirección. En el borde de bajada se supone que la dirección ya está estable en el bus, por lo que se activa la señal de lectura. Después del retraso de los **decodificadores** se activa la línea de selección de memoria que suministra el dato, que es capturado por el procesador en el segundo borde de subida.

- **Ventajas:** Este tipo de protocolo es fácil de interpretar, por lo que el bus puede funcionar a alta velocidad, ya que la **interfaz** es sencilla.
- **Desventajas:** El dispositivo debe funcionar a la misma frecuencia que la CPU, ya que si se conectan dispositivos de diferentes velocidades el bus deberá funcionar siempre a la velocidad del dispositivo más lento. Debido a que los tiempos son fijos, pueden haber problemas de sincronización debido a los diferentes retrasos de las señales si el bus tiene mucha longitud.

Transferencia asíncrona

En este tipo de transferencias la ocurrencia de un evento depende únicamente de la ocurrencia de un evento anterior. Por lo tanto no existe ninguna relación directa con el reloj del sistema. Para coordinar la transferencia es necesario un protocolo de intercambio de información (**handshake protocol**) que consiste en un conjunto de pasos donde **sólo** se llega a la siguiente etapa si el emisor y el receptor están de acuerdo.

Para **implementar** este protocolo el bus dispone de un conjunto de líneas adicionales y no utiliza la **señal** de reloj. En el siguiente ejemplo de lectura **asíncrona** de memoria **sólo** se utilizan 3 líneas de control:

- **Lectura:** Con ella el dispositivo que la activa indica que quiere realizar un **ciclo** de lectura de memoria.
- **Rdy:** Se utiliza para indicar que los datos **están estabilizados** en el bus. Esta línea la activa el **procesador** o la memoria.
- **Ack:** El dispositivo que la activa reconoce que ha detectado y procesado la **activación** de alguna de las líneas anteriores.

El **cronograma** correspondiente se muestra en la **figura 6.3**

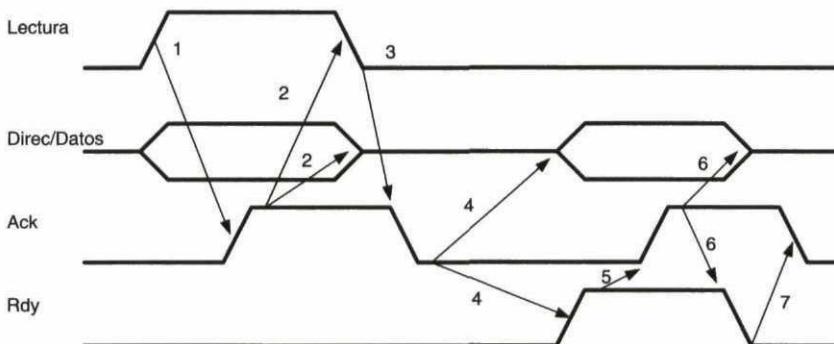


Figura 6.3: Transferencia asíncrona

La lectura tiene **lugar** en los siguientes pasos:

0. El maestro activa las direcciones y la **petición** de lectura.
1. El esclavo lee las direcciones y avisa activando la señal de reconocimiento.
2. El maestro detecta el **reconocimiento** y desactiva las direcciones y la **petición** del bus.
3. El esclavo detecta que cesa la petición y desactiva el reconocimiento.
4. Al transcurrir el tiempo de acceso de la memoria, el esclavo activa los datos y los valida.

5. El maestro lee los datos y los reconoce.
6. El esclavo detecta el reconocimiento y desactiva los datos y su validación.
7. El maestro elimina el reconocimiento para finalizar.

La principal ventaja que presenta esta sincronización es que acopla diferentes velocidades de CPU y periféricos. Los **buses** de **E/S** suelen ser de tipo asíncrono (a no ser que se deseé un bus **síncrono** de alta velocidad específico a algún dispositivo concreto). De manera adicional el bus puede tener grandes longitudes físicas, puesto que ya no va a haber problemas con los retrasos. Para evitar la parada del sistema debido a un error de protocolo, se utilizan **temporizadores de guardia** que activan una interrupción cuando la transferencia dura más de una determinada cantidad de tiempo.

Transferencia en bloque

Este tipo de transferencias se utilizan cuando es necesario aumentar el ancho de banda del bus. Se caracterizan porque se realiza la transferencia de más de una palabra (un bloque), especificando solamente la dirección de comienzo, ya que las demás posiciones se supondrán consecutivas. Hay **buses** en los que el número de ciclos que completa una transferencia es fijo y otros en los que puede ser variable, interrumpiéndose la transferencia a petición del maestro o del esclavo. Este **método** es muy utilizado en los **buses** de procesador para acceder a zonas de memoria y almacenarlas en la **cache**, existiendo versiones **síncronas** o asíncronas. La figura 6.4 muestra una transferencia en bloque asíncrona. En este caso se tienen 4 **líneas** para realizar la sincronización. Las señales AS y DS son de validación de dirección y de datos respectivamente y son activadas por el maestro. Las señales AK y DK son de reconocimiento de direcciones y datos y las activa el esclavo. En cada activación de la pareja de señales DS-DK, suponiendo activación por flanco, ya sea de subida o bajada, se envía un nuevo dato sin enviar una nueva dirección. El ciclo finaliza con la desactivación de AS y AK.

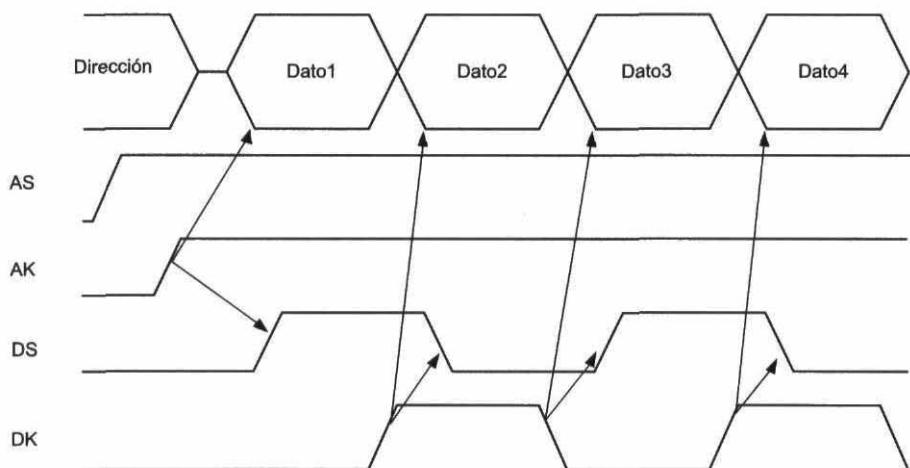


Figura 6.4: Transferencia en bloque

Transferencia *read-modify-write*

Este tipo de transferencia se utiliza para implementar una secuencia de eventos **ininterrumpible** en la que primero el maestro lee un dato, lo modifica y a continuación escribe el resultado en la misma posición. Este tipo de transferencia es **importante** en sistemas **multiprocesador** con memoria compartida, ya que **mientras** un procesador accede a un dato para **modificarlo**, los **demás** procesadores no deben acceder a esa posición de memoria hasta que el dato haya sido **actualizado**. En la figura 6.5 se muestra este tipo de **transferencia** en el caso de un bus **asíncrono**. **Primero** se produce un lectura **sincronizada** con DS-DK y posteriormente una escritura sin dar una nueva dirección. Durante toda la transferencia el bus permanece bloqueado hasta la desactivación de AS-AK.

6.5 ARBITRAJE DEL BUS

En un instante determinado, solamente un dispositivo debe transmitir información por el bus. Si varios dispositivos tratan de usar el bus simultáneamente se pueden producir **errores**, puesto que las **líneas** pueden estar en un estado indefinido. A este hecho se le denomina contención de bus, y para evitarlo debe existir un **método**, llamado **arbitraje del bus**, que impida que **varios** módulos utilicen el bus de forma simultánea. El arbitraje se caracteriza por:

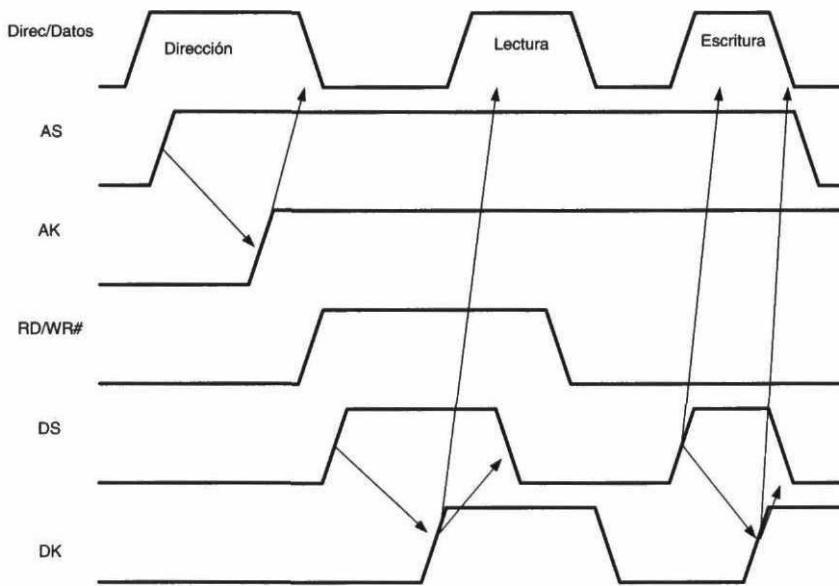


Figura 6.5: Transferencia *read-modify-write*

- 1. Priorización de peticiones:** El maestro de prioridad más alta será el próximo dueño del bus.
- 2. Juego limpio (*fairness*):** Cualquier dispositivo que quiera usar el bus tiene garantizado que tarde o temprano lo podrá usar.
- 3. Reducción al máximo del tiempo de arbitraje:** El proceso de selección se solapa con las transferencias por el bus de datos.

La forma más simple de funcionamiento del bus consiste en que no haya arbitraje, sólo hay un maestro que es el procesador que controla el bus en todo momento. Este caso no es atractivo porque el procesador se ocupa de forma innecesaria de realizar las transferencias de datos entre **dispositivos** veriféricos y memoria. La alternativa consiste en tener varios maestros de bus que puedan iniciar transferencias. Para ello habrá que implementar un mecanismo de arbitraje que decida **qué** maestro es el que va a usar el bus. Existen básicamente dos formas de arbitraje: Centralizado, donde un dispositivo, llamado controlador de bus o árbitro, es el responsable de asignar la posesión del bus, o distribuido, donde todos los módulos deciden entre si el próximo maestro que accede al bus.

En un esquema de arbitraje centralizado, el dispositivo que quiere usar el bus activa una **señal de petición (bus request)**. Después *de un* proceso de arbitraje, si resulta ganador se le avisa activando la **línea de concesión (bus grant)**. Cuando el dispositivo no desea usar más el bus activa la **línea de liberación (bus release)** para que el árbitro realice un nuevo arbitraje.

Los diversos tipos de arbitraje, tanto centralizado como **distribuido**, se pueden agrupar en 4 clases:

- **Arbitraje daisy-chain:** La **línea de concesión** va atravesando los dispositivos de mayor a menor prioridad (la prioridad la determina el orden en el que la **línea** las atraviesa). Cuando un dispositivo detecta la activación de esta línea puede utilizar el bus sin activar la **señal** hacia el siguiente **módulo**. En el caso de que no necesite acceder al bus, **propagará la activación** de la señal al siguiente **módulo** de la cadena. Se caracteriza porque es simple de **implementar**, pero el **proceso** puede llegar a ser lento si existen muchos dispositivos, ya que la señal debe **recorrer** todos los **módulos**. Un ejemplo de bus que utiliza este tipo de arbitraje es el VME, cuyas especificaciones se indican al final del **capítulo**. Un esquema de la estructura de este método se puede observar en la **figura 6.6**.

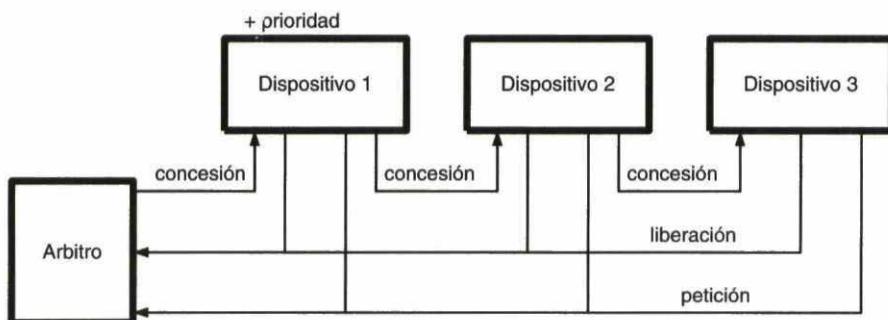


Figura 6.6: Arbitraje daisy-chain

- **Arbitraje centralizado-paralelo:** En este caso se usan múltiples **líneas de petición** independientes que son recogidas por un árbitro central, el cual elige el **próximo maestro** que accede al bus. El principal **problema** que presenta un esquema con un único árbitro es que éste se puede convertir en un cuello de botella en las tareas de arbitraje. Además, el funcionamiento de todo el sistema depende de este elemento, por lo que no es bueno en

los sistemas tolerantes a fallos. Por otro lado su principal ventaja parte del hecho de que la prioridad la decide el árbitro central, por lo que es flexible y reprogramable.

- Arbitraje distribuido por **autoselección**: Estos esquemas usan también múltiples líneas, **pero** son los propios dispositivos quienes determinan el ganador del bus. Cuando comienza el arbitraje cada dispositivo pone su **código** sobre las líneas de arbitraje. A continuación se examinan estas líneas para comprobar si su valor coincide con el código del módulo. En caso **afirmativo** significa que se ha ganado el arbitraje, puesto que no hay otro dispositivo con mayor prioridad. Presentan la ventaja de que cambiando el **código** de cada dispositivo se puede alterar la prioridad. El bus **NUBUS** y el Gespac utilizan este método.
- Arbitraje distribuido por detección de **colisión**: Cada dispositivo trata de utilizar el bus en el momento que detecta que está libre. Pero cuando se realizan peticiones simultáneas se produce una colisión, que debe ser detectable por cada uno de los dispositivos. En ese caso todos los dispositivos dejan de utilizar el bus y esperan un tiempo aleatorio hasta volver a intentar acceder al bus. La red **Ethernet** utiliza este método de arbitraje.

6.6 INTERRUPCIONES

Los **buses** como medio de conexión entre los distintos elementos de un sistema **informático** deben permitir que cualquier esclavo pueda producir interrupciones en el sistema. Dependiendo de la prioridad de la interrupción, el maestro parará la tarea actual **en un** instante determinado y la **atenderá**. Si se solicitan varias interrupciones a la vez, se debe determinar cuál es la más prioritaria y dejar las demás en espera. Los métodos más utilizados para la **implementación** de las interrupciones son:

- Mediante **Líneas** individuales. En este primer método cada uno de los esclavos tiene una línea individual que llega hasta el maestro. Con la activación de cada línea, el maestro reconoce el esclavo que ha producido la petición. Las ventajas del método son la velocidad y la flexibilidad a la hora de priorizar las interrupciones, ya que se pueden modificar las prioridades de cada esclavo reprogramando el maestro, **así** como la sencillez.

Sin embargo, tiene como desventajas la existencia de un gran número de líneas y el posicionado geográfico del maestro, que debe situarse siempre en el mismo *slot* del bus, al cual llegan todas las **líneas** de interrupción de los esclavos, tal como se muestra en la figura 6.7.

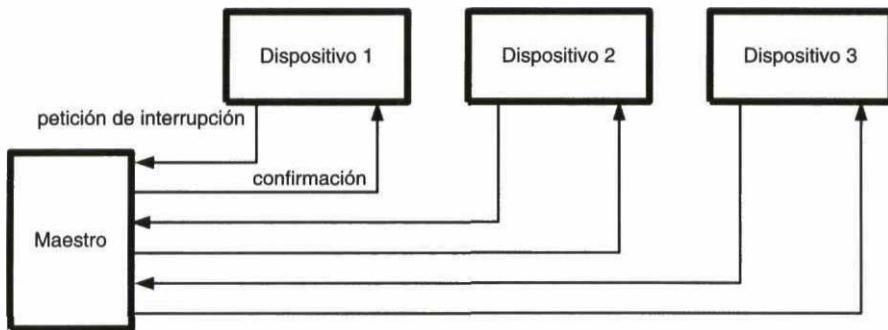


Figura 6.7: Mediante líneas individuales

- m **Exploración secuencial.** En este caso todos los esclavos **comparten** una misma línea de interrupción. El maestro tiene constancia de que se ha solicitado una **interrupción** pero no conoce la fuente. Mediante **líneas** dedicadas, el maestro **genera** una serie de **códigos**, cuando el generador de la **interrupción** reconoce su **código** en el bus, responde mediante otra **línea** destinada para **tal fin**. Es un método lento, pero tiene las ventajas de la flexibilidad y de que se trata de un método no **geográfico**. En la figura 6.8 se puede **observar** su esquema.

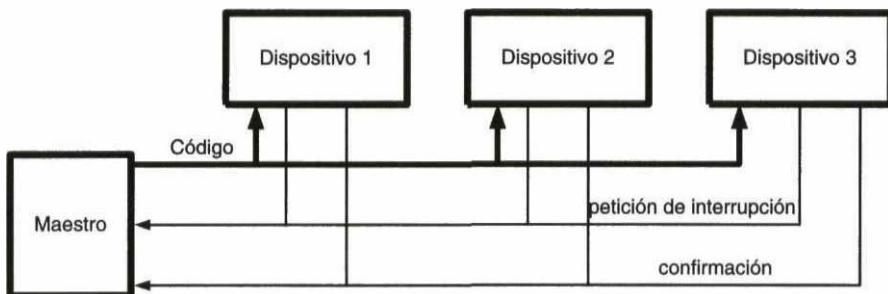


Figura 6.8: Exploración secuencial

- m **Daisy-Chain.** De nuevo hay una única línea de **interrupción** compartida por **todos** los esclavos. Junto con la línea de petición existe una línea de

identificación que **recorre** todos los dispositivos desde el maestro hasta el **último** esclavo en orden **geográfico**. Cuando el maestro activa esta **señal**, cada uno de los esclavos no peticionarios que la **reciben** propagan la **activación** hacia el esclavo que ocupa la siguiente **posición** en el bus. Cuando la **señal llega** al esclavo **solicitante**, éste para la **activación** hacia el siguiente esclavo y pone el vector de **interrupción** en el bus. Es un método lento, ya que la señal debe recorrer todos los maestros, y **geográfico**, puesto que la prioridad se codifica dependiendo de la **posición** que ocupe el esclavo en el bus, pero tiene la ventaja de la sencillez. La figura 6.9 muestra este último **método**.

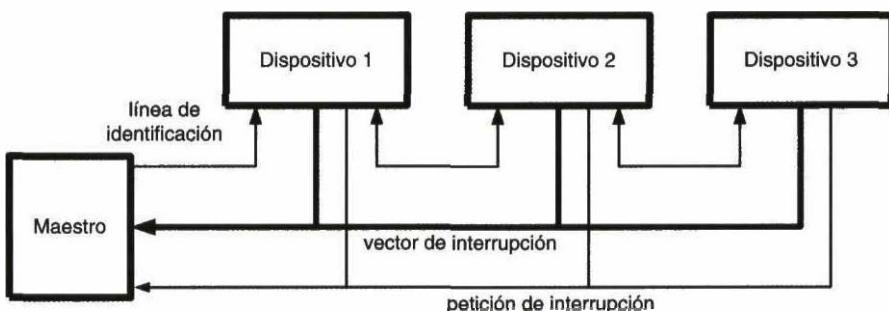


Figura 6.9: Daisy-Chain

6.7 CONTROL DE ERRORES

A continuación se describe una lista con los tipos de errores más comunes que se producen en los **buses**:

1. Transacciones a direcciones de **memoria/periféricos** no existentes.
2. Bloqueo del sistema por errores en los protocolos asíncronos.
3. Accesos a zonas protegidas por el Sistema Operativo.
4. **Errores** de naturaleza **eléctrica** en las señales:
 - Ruido externo (EMI) de carácter **electromagnético**: Estos problemas se solucionan utilizando mecanismos de **apantallamiento** (placa o **carcasa** de metal conectada a masa).

- Ruido que proviene de la red eléctrica: para evitarlo se filtra la entrada de la fuente de alimentación.
- Ruido cruzado (**crosstalk**): se produce por las comentes inducidas que se generan entre dos líneas muy juntas, cuando las comentes **conmutan** a muy alta frecuencia. Para solucionar el problema o bien se **trazan líneas** de masa entre las líneas, o se utilizan algoritmos de **rutado** que previenen el trazado de líneas paralelas por encima de una cierta longitud.
- Reflexiones de las señales de alta frecuencias (**ringing**): cuando la señal llega a su destino se refleja hacia el origen de la línea. Este problema se soluciona situando **terminadores** con una resistencia adecuada al final de cada **línea** para adaptarla y evitar las reflexiones.
- Variaciones en la alimentación de los integrados producidas por cambios bruscos de consumo: si de repente el integrado varía mucho su consumo de comente se puede producir una variación en la tensión de alimentación. Para evitarlo se utilizan **condensadores** de desacoplamiento, uno por cada par de entradas de alimentación y tierra, que se oponen a los cambios bruscos de tensión.

El sistema debe implementar mecanismos de **temporización** (*time-outs*) para detectar errores, principalmente los que dejan al sistema en un estado desconocido. Para **ello**, se utilizan los **temporizadores** de guardia (*watch-dog timers*), que contienen un contador que el **procesador** **inicializa** periódicamente. Si se bloquea el sistema, el **procesador** no podrá **reiniciar** el contador, por lo que se alcanzará el **final** de la cuenta. Cuando esto **ocurre**, el **temporizador** envía una interrupción de **alta** prioridad al procesador para intentar deshacer el bloqueo.

6.8 EJEMPLOS DE BUSES COMERCIALES

Los **buses** comerciales desarrollados en su mayoría por uno o **varios** fabricantes, **están** totalmente especificados por medio de un **estándar**, que como ya se ha explicado, recoge todas las características del bus. Esta **estandarización** permite que varios fabricantes puedan **desarrollar** productos para un mismo bus sin problemas de **incompatibilidades**. Dependiendo de las prestaciones que posean, se orientan hacia la interconexión de unos **determinados** elementos dentro

del computador, situándose dentro de **algún** grupo de la jerarquía de **buses anteriores** descrita. De esta forma podemos encontrar **buses** locales como el PCI, **buses de expansión** como el bus ISA y **buses** de sistema como el VME. Al igual que todo sistema **informático** los **buses** comerciales también sufren una evolución constante, con la aparición de nuevos **buses** y la desaparición de los obsoletos. A continuación se van a explicar las características **básicas** de dos **buses** comerciales el PCI y el VME.

6.8.1 Bus PCI

El bus **PCI** nació con la intención de **eliminar** el cuello de botella que se **había** producido entre el procesador y los dispositivos periféricos **gráficos** en los computadores personales PC, debido a la aparición de sistemas operativos como **Windows** y **OS/2**. En estos sistemas operativos hay una transferencia continua de información hacia la **memoria gráfica** que utiliza el controlador **gráfico** para refrescar la pantalla. Por ejemplo, cada vez que se activa un cuadro de diálogo hay que salvar y **reescribir** una parte de la pantalla. Con los **buses** de expansión **anteriores** a PCI, ISA y EISA con velocidades inferiores a 10 Mhz, esta tarea se hacía muy costosa, por lo que aparecieron los primeros **buses** locales. En un principio estos **buses** no fueron más que el propio bus del **microprocesador** al cual se conectaba la memoria **gráfica** para ganar velocidad. El problema es que este bus depende del procesador y no es **estándar**. Para evitar esta desventaja se desarrollaron dos **buses**, por un lado **Intel** desarrolló el **PCI (Peripheral Component Interconnect)** y por otro el comité VESA desarrolró el **Vesa local bus**, hoy en **día** casi desaparecido. Una aplicación típica del PCI se muestra en el diagrama de bloques de la figura 6.10.

Características básicas del bus

En el esquema de la figura 6.10 se observan tres tipos distintos de elementos PCI. El primero es el puente entre el bus del procesador y el bus local PCI, este puente permite el desacople de los dos **buses**. Internamente cuenta con **buffers** de **prebúsqueda** y de escritura, que le permiten convertir ciclos independientes de lectura o escritura de la **CPU** en ciclos de transferencia en modo bloque. En el caso de la memoria **gráfica**, la **CPU**, aunque accede a posiciones consecutivas

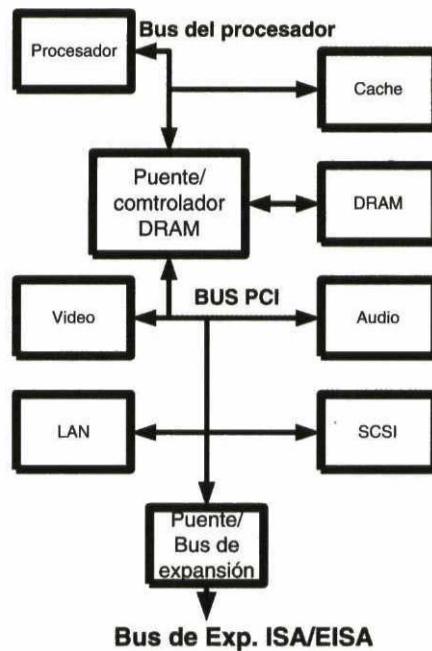


Figura 6.10: Esquema de aplicación del bus PCI

de memoria, no puede realizar la transferencia en modo bloque, ya que esta zona de memoria no tiene correspondencia en la **cache**. El puente **PCI** detectará que todas las transferencias son consecutivas y realizará una transferencia en bloque, siendo ésta la gran ventaja del bus **PCI**. La máxima velocidad de transmisión cuando el bus **realiza** transferencias en bloque utilizando líneas de datos de 64 bits es de **266Mbytes/s**.

Los dispositivos o agentes **PCI** se agravan al sistema mediante los **conectores** de expansión. Hay dos tipos distintos de conectores, que permiten el **funcionamiento** a tensiones de 5V y 3.3V. Los agentes permiten la conexión al bus de elementos como controladores **SCSI**, controladores **LAN** y tarjetas **gráficas**, entre otros. Tanto el esquema de arbitraje como el de interrupciones de estos elementos esclavos es centralizado con líneas independientes.

Con respecto al bus cabe destacar que tiene un funcionamiento **síncrono** con frecuencias de reloj que pueden llegar hasta los 33Mhz. Están multiplexadas las **d i i o n e s** y los datos, lo cual disminuye el **número** de líneas necesarias para

implementar el bus pero reduce la velocidad de funcionamiento. Para evitar en lo posible esta degradación, el modo de transferencia básico es el modo bloque, con un número cualquiera de datos en cada transferencia.

Además de los espacios de direcciones de memoria y **entrada/salida**, el bus PCI implementa un nuevo espacio, llamado espacio de direcciones de **configuración**. Se usa para **acceder** a los 256 bytes internos de los elementos PCI que forman los registros de configuración.

Señales

Las **señales** más importantes que utiliza el bus son:

Señales del sistema:

- **CLK**: Es la señal que **sincroniza** todas las transacciones en el bus.
- **RST#**: Es la señal de reset para todos los registros, contadores y señales del bus.

Señales de direcciones y datos:

- **AD[31:0]**: Las direcciones y los datos están multiplexados. Una transacción está compuesta por una **dirección** seguida de una o más fases de datos.
- **C/BE[3:0]**: Los comandos de bus y la habilitación de los byte (byte *enables*) están multiplexados en los mismos pines. Durante la fase de dirección se definen los comandos. Durante la fase de datos se usan como señales de habilitación de los **bytes**.
- **PAR**: Bit de paridad de AD y C/BE.
- **AD[63:32]#**: Proporciona 32 bits adicionales multiplexados de datos y direcciones.
- **C/BE[7:4]#**: Comandos del bus y habilitación de byte multiplexados sobre el bus.

- **REG64#:** Indica que se quieren enviar datos usando direcciones de 64 bits.
- **A CK W** Indica que se va a transferir un dato de 64 bits.
- **PAR64:** Es la paridad de las señales AD y C/BE de extensión.

Señales de control:

- **FRAME#:** Es utilizada por el maestro para indicar el principio y la duración de un acceso.
- **IRDY#:** Indica que durante la presente transacción el maestro está preparado para aceptar los datos (lectura) o ha puesto los datos (escritura).
- **TRDY#:** Indica que durante la presente transacción el esclavo está preparado para aceptar los datos (escritura) o ha puesto los datos (lectura).
- **STOP#:** Indica que el esclavo quiere parar la transacción en curso.
- **LOCK#:** Indica una operación atómica que requiere múltiples transacciones para completarse.
- **IDSEL:** Selección de dispositivos para configuración.
- **DEVSEL#:** Reconocimiento activado por el dispositivo seleccionado.

Señales de error:

- **PERR#:** Se activa por el elemento receptor de los datos cuando ha habido un error de paridad en los datos.
- **SERR#:** Se activa por el elemento receptor de los datos cuando ha habido un error de paridad en las direcciones o cuando hay un error catastrófico.

Señales de interrupción:

- **INTA#:** Hay 4 líneas de interrupción activas por nivel.
- **INTB#, C, D:** Son líneas utilizadas para dispositivos multifunción.

Órdenes

Las órdenes del Bus indican el tipo de transacción que se va a realizar:

- e **Acknowledge**: Es un ciclo de reconocimiento de interrupciones.
- **Special cycle**: Es un mensaje broadcast.
- e **I/O read command**: Lectura de un puerto.
- e **ZO write command**: Escritura de un puerto.
- e **Memory read command**: Lectura de memoria.
- m **Memory write command**: Escritura de memoria.
- e **Configuration read command**: Lectura del espacio de configuración de cada agente.
- m **Configuration write command**: Escritura en el espacio de configuración de cada agente.
- e **Memory Read multiple**: Indica que el maestro intenta acceder a más de una línea de cache antes de terminar la transacción.
- e **Dual address cycle**: El comando se usa para transferir direcciones de 64 bits.
- e **Memory write and invalidate**: Es igual que la escritura pero garantiza como mínimo la transferencia de una línea completa de la cache.

Transferencias en el bus

El mecanismo básico de transferencia es en modo bloque (**burst**), tanto a memoria como a WS. Una transacción siempre es iniciada por un maestro tras ser autorizado por el árbitro del bus. El esquema de arbitraje es centralizado con **líneas** independientes para cada dispositivo. Cada uno de los maestros de bus debe realizar el arbitraje para cada acceso. El algoritmo de arbitraje utilizado en cada caso depende del diseñador y es implementado por las señales REQ# y GNT#. A continuación, en la figura 6.11 se muestra una transacción de lectura.

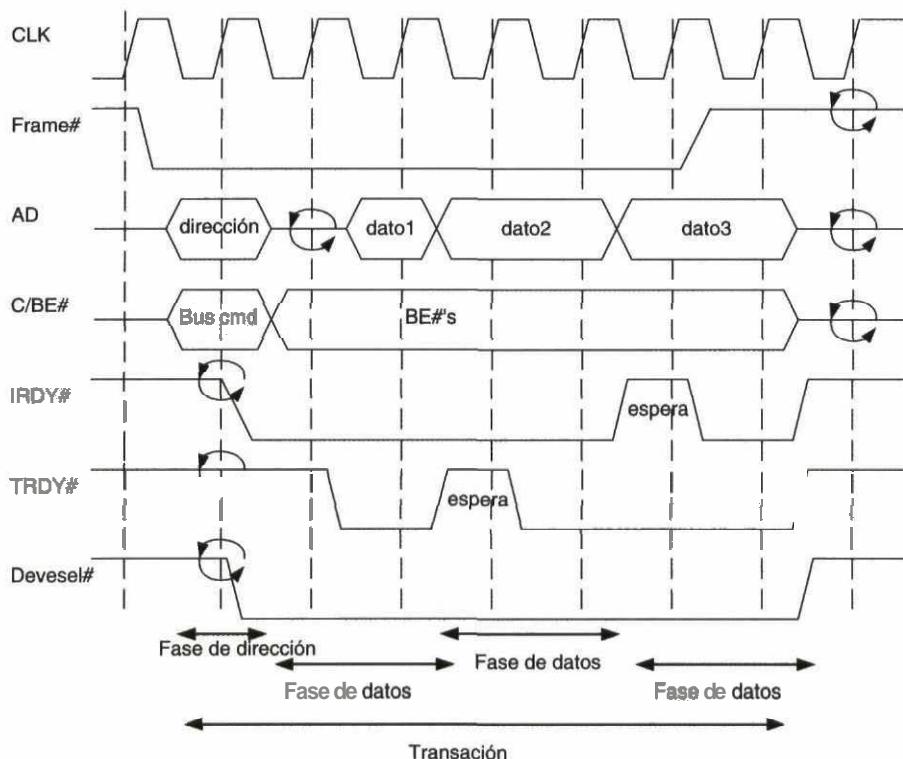


Figura 6.11: Transferencia de lectura en el bus PCI

La transacción se inicia con la activación de la línea **Frame#** al mismo tiempo que se da la dirección y el comando. Las líneas **IRDY#**, **TRDY#** y **Devesel#** por su parte tienen un periodo de contención. El siguiente ciclo presenta la contención en las líneas **AD**, debido a que es **una** lectura, para que el maestro deje paso al esclavo en la activación de dichas señales. En este mismo ciclo se produce la respuesta del esclavo **direccinado** mediante **Devesel#**. A continuación se **inician** los ciclos de datos. En cada uno de ellos, para que se pueda **producir** la transferencia, tanto la línea **IRDY#** como la **TRDY#** deben estar activas, y al ser transferencia en bloque cada nuevo dato corresponde a una dirección consecutiva a la anterior. La transacción **finaliza** cuando el maestro desactiva la señal **Frame#**, indicando que la próxima transferencia es la **última** y **esta** se completa. A continuación hay un ciclo de contención en las líneas **Frame#**, **C/BE#** y **AD**, antes de comenzar una nueva transferencia.

Conclusión

Hoy en día el bus **PCI** está presente en todos los computadores personales PC. Existen una gran cantidad de tarjetas periféricas que utilizan este bus, además la mayoría de dispositivos que utilizaban el bus de expansión ISA están siendo rediseñados para **PCI**, ya que ofrece mejores prestaciones. Por otra parte el bus PCI en computadores personales ha dejado de ser el bus de interconexión con la tarjeta **gráfica**, esto es debido a la aparición del bus AGP, que ha sido diseñado específicamente para ese fin y ofrece unas velocidades de transferencia superiores.

6.8.2 Bus VME

El bus VME (*Versa Module Europe*) aparece en 1981 como sucesor del **Versabus**, fue desarrollado conjuntamente por Motorola, **Signetics** y Mostek, siendo actualmente el bus de sistema más utilizado dentro de un conjunto en el que podemos encontrar **buses** como Fastbus, Futurebus y Multibus. Aceptado como **estándar** por **IEEE** en la norma 1014, es el más **sencillo** y el que menos prestaciones tiene de todos los **buses** de sistema, sin embargo su gran aceptación es debida a que es posible encontrar más de 200 módulos comerciales para este tipo de bus. Sus aplicaciones se centran en los campos de sistemas industriales en tiempo real, aplicaciones militares e investigación.

Características del bus

Es un bus **multiproceso**, lo que significa que múltiples procesadores pueden **compartir** el mismo bus. El chasis mecánico (**rack**) es del **tipo Eurocard** de 19" y tiene hasta 21 ranuras de expansión. Los módulos pueden ser simples con un solo **backplane** denominado **P1/J1** que maneja hasta 24 bits de direcciones y 16 bits de datos o de dos **backplane** **P1/J1 + P2/J2** con **32** bits de direcciones y datos. El bus tiene una adaptación dinámica del **tamaño** y acepta datos de 8, 16 o 32 bits con una justificación (*low-end byte*).

Es **asíncrono** y no multiplexado con velocidades máximas de transmisión de hasta 10 **Mbytes/s** para transferencia simple, 20 **Mbytes/s** con transferencia en

bloque y 40 **Mbytes/s** con transferencias en bloque con datos de **32 bits**. Las **líneas** de alimentación utilizadas son de +5V y ±12V utilizando tecnología TTL. Las interrupciones y el arbitraje del bus utilizan un esquema **daisy chain** y no disponen de **direcccionamiento geográfico**.

Señales

Las **señales** que utiliza el bus son:

- **AD(AD31-AD0)**: Son las **líneas** de direcciones que pueden utilizarse como líneas de 16.24 o 32 bits. Sin embargo sólo pueden utilizarse 8 bits para acceder a dispositivos de Entrada/Salida.
- **AM (AM5-AM0)**: Indican el **espacio** de direcciones al cual se pretende acceder, **código**, datos, modo **usuario** o privilegiado. Además indican cuál es el tamaño de la dirección contenida en AD.
- e **DB(DB31-DB0)**: Son las **líneas** de datos y pueden contener datos de 8, 16 o 32 bits con justificación baja.
- **WRITE#**: Señal de escritura o lectura.
- **DS1#, DS0#**: Indican un dato de 8 o 16 bits respectivamente.
- **LWORD#**: Indica una transferencia con un dato de **32 bits**.
- **AS#, DTACK#**: Señales de protocolo para transferencia **asíncrona**, el maestro activa AS# al indicar la **dirección** y el esclavo activa DTACK# al completar una transferencia.
- **BR#, BGIN#, BGOUT#, BBSY#** y **BCLR#**: Señales de arbitraje con BR#, petición del bus, daisy chain, BBSY#, BCLR# bus ocupado y libre respectivamente.
- **IACK#, JACKIN#, JACKOUT#**: Señales de ciclo de interrupción, y entrada salida de interrupción en daisy chain.
- e **SYSCLK#, SYSRESET#, SYSFAIL#, ACFAIL#**: Utilización de **señales** periódicas, **reset**, fallo del sistema y alimentación.

BD24-31	BD16-23	BD8-15	BDO-7	DS1#, DS0#, AI, LWORD#
BYTE(0)	BYTE(1)	BYTE(2)	BYTE(3)	0, 0, 0, 0
BYTE(0)	BYTE(1)	BYTE(2)		0, 1, 0, 0
	BYTE(1)	BYTE(2)	BYTE(3)	1, 0, 0, 0
	BYTE(1)	BYTE(2)		0, 0, 1, 0
		BYTE(2)	BYTE(3)	0, 0, 1, 1
		BYTE(0)	BYTE(1)	0, 0, 0, 1
			BYTE(3)	1, 0, 1, 1
		BYTE(2)		0, 1, 1, 1
			BYTE(1)	1, 0, 0, 1
		BYTE(0)		0, 1, 0, 1
ILEGAL				1, 0, 1, 0
ILEGAL				0, 1, 1, 0

Tabla 6.1: Tamaño y posición de los datos en el bus VME

Transferencias en el bus

El bus presenta cuatro tipos de ciclos:

- Ciclo simple de lectura o escritura.
- Ciclo de transferencia en bloque.
- Ciclo **read-modify-write**.
- Ciclo de reconocimiento de interrupciones.

El tipo más común es el simple, en el cual el dispositivo maestro envía un dato de 8, 16, 24 o 32 bits a un esclavo. El tamaño del dato y las líneas del bus por las que se transmite se indica mediante **DS0#, DS1#, LWORD#** y A1 siguiendo la tabla 6.1.

El **direcccionamiento** se realiza con las señales AD31-ADO, **AM5-AM0**, **DS1#-DS0#**, **IACK#** y **LWORD#**, validadas mediante la señal AS#. El tamaño de la dirección también puede ser variable con valores de 16 bits (**A15-A0**), 24

bits (**A23-A0**) y 32 bits (**A31-A0**), indicado mediante las señales AM. La señal **IACK#** inactiva indica que no es ciclo de reconocimiento de interrupción. La finalización del ciclo se produce cuando el esclavo **afirma** mediante la señal **DTACK#**, el maestro desactiva las señales **AS#**, **DS0#** y **DS1#**, y por último el esclavo desactiva **DTACK#**. A continuación en la figura 6.12 se muestra un ciclo de escritura simple para este bus.

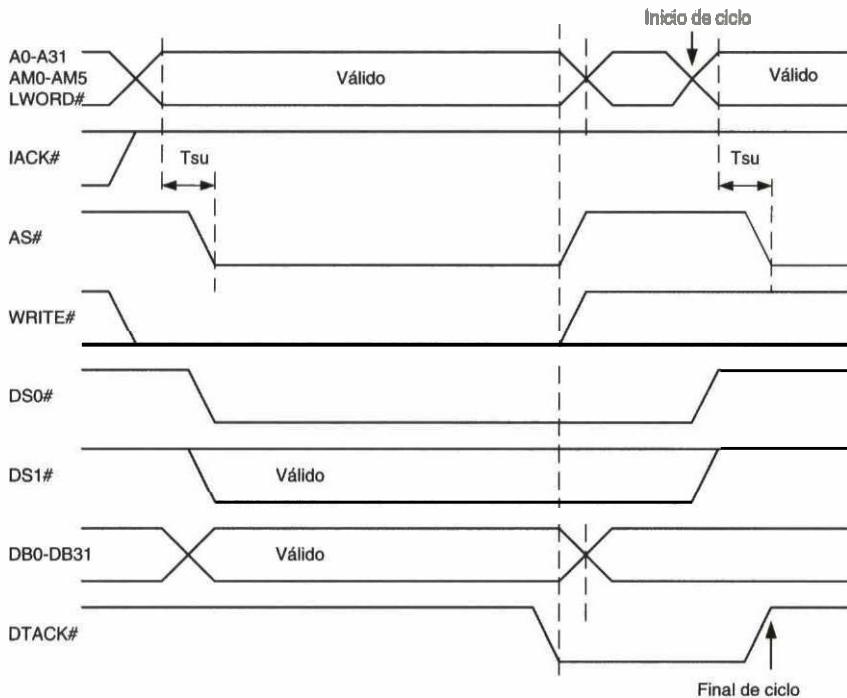


Figura 6.12: Transferencia de escritura simple en VME

Conclusión

Los **buses** de sistema no **están** presentes en computadores personales, sino que se reservan para aplicaciones informáticas **más sofisticadas** en las que es necesario añadir módulos específicos, ya sean de memoria o varias tarjetas **procesadoras**, que permiten construir sistemas con mayores prestaciones. Esta es la **razón** por la que no son **buses** muy populares y es más difícil encontrar módulos que los soporten.

6.9 CONCLUSIONES

Dentro de cualquier computador se encuentra una jerarquía de **buses** que interconecta todos sus elementos. Cada uno de los niveles que forman esta jerarquía engloba **buses** que están orientados a la interconexión de unos determinados elementos, con unos requisitos de ancho de banda y prestaciones distintas para cada nivel. La especificación de un bus se realiza mediante un **estándar** en el cual se recogen todas sus características, desde el número de líneas hasta los modos de operación, incluyendo la sincronización, los tipos de transferencia, el arbitraje y las interrupciones. Esta estandarización es necesaria para que todos los módulos que se conecten al bus puedan intercambiar información, y permite la construcción de sistemas de **forma** modular.

6.10 CUESTIONES

6.1 *¿Qué utilidad tiene el driver triestado mediante el cual los dispositivos acceden a las líneas de un bus?*

6.2 *¿A qué bus se conectarían las tarjetas de interfaz con la conexión RS-232 dentro de la jerarquía de buses?*

6.3 *¿Cuál es el mínimo intervalo de tiempo que debe esperar un maestro antes de enviar o recibir un nuevo dato, en una transferencia en bloque asíncrona?*

6.4 *Realiza el cronograma de una transferencia en bloque asíncrona, en la que las transacciones de datos se realicen solamente en los flancos de subida de las señales DS-DK. ¿Cuál sería el mínimo tiempo de espera entre dos datos en este caso?*

6.5 *Enumerar las ventajas y desventajas de cada uno de los métodos de arbitraje del bus.*

6.6 *¿Qué tipo de arbitraje tiene el bus VME? ¿Cómo funciona este tipo de arbitraje?*

6.7 Enumerar las ventajas y desventajas de cada uno de los métodos de **implementación** las interrupciones en un bus.

6.8 ¿Cuál **fue** la causa que impulsó la aparición del bus PCI? ¿Qué problemas solucionó?

6.9 ¿Tarda el mismo tiempo en producirse la lectura y la escritura de un **dato** en el bus PCI? ¿Por qué?

6.11 BIBLIOGRAFÍA

- a **Design of Microprocessor Based Systems.** *Nikitas Alexandridis.* Prentice-Hall International, Inc. ISBN 0-13-588567-1. 1993.
- a **The Indispensable PC Hardware Book.** *Hans-Peter Messmer.* 3^a ed. Addison Wesley Longman, Inc. 1997. ISBN 0-201-40399-4.
- a **Periféricos e Interfaces Industriales.** *José C. Campelo, Francisco Rodríguez, Vicente Torres.* Servicio de Publicaciones de UPV. ISBN 84-7721-474-3.1997.
- a **PCI Local Bus Specification. Revision 2.1.**
- a **Computer Organization and Architecture. Designing for performance.** *William Stallings.* 4^a ed. Prentice-Hall International, Inc. 1996. ISBN 0-13-394255-4.

CAPÍTULO 7

EL ENLACE EXTERIOR

7.1 INTRODUCCIÓN

En este capítulo se describe el enlace exterior como el conjunto de **interfaces** que conectan los **controladores** de **Entrada/Salida** con sus dispositivos periférico-asociados. Ejemplos de estos enlaces son los **buses** dedicados de **Entrada/Salida** que conectan la impresora con el ordenador (interfaz **centronics**), el cable del **módem** (interfaz **RS232**), los cables de los discos (interfaz **IDE** o **SCSI**), el cable del teclado e incluso el de un ratón.

El objetivo a la hora de diseñar estos elementos es doble. Por una parte, para los dispositivos que necesitan un ancho de banda pequeño se trata de diversificar lo más posible los protocolos para poder conectar un mayor abanico de dispositivos diferentes, como pone de manifiesto el bus **USB**. Sin embargo para otros dispositivos, como los de almacenamiento masivo, el objetivo principal del enlace exterior es el de incrementar el ancho de banda al máximo.

En capítulos anteriores se han visto ya las técnicas empleadas para conseguir estos objetivos. A lo largo de esta sección se van a describir principalmente los estándares más importantes y más comunes que se utilizan en la actualidad para conectar los periféricos al ordenador.

72 TIPOS DE INTERFACES

Dentro de la estructura jerárquica de **buses** que presentan los sistemas **informáticos**, destacan aquellos que son útiles para conectar el ordenador con el mundo exterior. Se caracterizan porque son los más lentos, tienen una menor longitud de palabra y menores velocidades de transferencia de datos. Su diseño se basa en un **estándar** para permitir la interconexión de dispositivos de diferentes fabricantes.

Los **buses** de **E/S** se utilizan en las operaciones de **E/S** para llevar los datos desde el periférico hacia su controlador. Como ejemplos **se puede** citar la conexión existente entre el disco duro y su controlador, la conexión con la impresora o con un **terminal serie** y también la **conexión** a una red informática.

Las operaciones que tienen lugar y los protocolos que se siguen en las transferencias se **unifican** en una serie de **estándares** que hacen más fácil la interconexión de dispositivos con características similares. A todos los elementos **hardware** y **software** que hacen posible la conexión entre dos unidades diferentes se les denomina interfaz. Esta interfaz debe estar diseñada según las características del periférico al que se accede para poder obtener el **mayor rendimiento** posible. **Normalmente** se distingue entre **interfaces** serie e **interfaces** paralelas.

1. **Interfaz serie:** Se utiliza una única línea para transmitir los datos.
2. **Interfaz paralela:** Se utilizan varias **líneas** de datos para **transmitir** múltiples **bits** de forma simultánea.

Se puede realizar otra posible clasificación si se tienen en cuenta el número de dispositivos y de controladores que se pueden conectar a través de un enlace de comunicación. Se pueden distinguir las conexiones punto a punto y las conexiones multipunto. Una conexión punto a punto dedica un enlace exclusivamente para conectar el módulo de **E/S** y el dispositivo externo. En los sistemas pequeños se utiliza este sistema para conectar teclados, **impresoras** y **módem**s. Un ejemplo típico de esta interfaz es la **especificación EIA-232 de conexión serie**.

De mayor importancia son las interfaces externas multipunto, utilizadas en los dispositivos de almacenamiento secundario y dispositivos multimedia. Estas interfaces son realmente **buses** externos y utilizan la misma lógica que el bus del sistema, permitiendo conectar múltiples dispositivos a través del mismo cable.

7.3 INTERFACES SERIE

La conexión a través de esta interfaz es muy importante debido a su gran flexibilidad. En los ordenadores personales la interfaz serie se utiliza para conectar multiples dispositivos como plotters, módems, ratones y también impresoras.

En la transmisión serie se van **transfiriendo los** bits de información uno a uno a través de una línea de datos. Si se utilizan señales adicionales (reloj o señales de petición y reconocimiento) para indicar cuándo el bit siguiente es válido, entonces se dice que la transmisión se realiza de forma síncrona. La principal ventaja de este tipo de transferencias es que el receptor puede funcionar a varias frecuencias de reloj (siempre que no se sobrepase su frecuencia máxima de funcionamiento). Simplemente bastará con retrasar el envío de la **señal** de reconocimiento para relentizar el protocolo. En las transferencias **asíncronas**, por el contrario, tanto el receptor como el transmisor deben funcionar a la misma frecuencia. En este caso se envía también información de sincronización a través de la línea de datos, que se corresponde con un bit de comienzo (**bit de start**), que indica el comienzo de una unidad de datos, un bit de fin (**bit de stop**) indicando su finalización y opcionalmente un bit de paridad para controlar los posibles errores.

El bit de paridad lo generan los controladores serie de forma automática, pudiendo configurarse entre las opciones de: sin paridad, paridad par (**odd**), paridad impar (**even**), siempre un nivel alto (**mark**) o siempre un nivel bajo (**space**).

Las tasas de transferencia de datos se miden en baudios. Los baudios indican el número de veces que puede cambiar una señal en la línea de transmisión por segundo. En una interfaz **serie**, las señales cambian siempre a la misma frecuencia y se realiza una codificación binaria de la información de forma que cuando se quiere enviar un '1' se pone la línea a nivel alto y cuando se quiere enviar un '0' se pone la línea a nivel bajo. En este caso los baudios coinciden

con el **número** de bits por segundo **transferidos** si se incluyen también los bits de comienzo, de fin y el de paridad.

Para poder **realizar** una transferencia **asíncrona**, tanto el emisor como el receptor se deben poner de acuerdo en la frecuencia de transmisión de datos y en la **forma** de las tramas que se van a enviar: número de bits de datos, de comienzo, de fin y tipo de paridad. **Después** de la recepción de una trama, se pueden producir los siguientes errores:

- **Error de trama (Framing error).** Se produce cuando el receptor detecta un bit de stop en un momento que no le **corresponde**, puesto que no se adapta a la forma de las tramas que se espera recibir.
- **Error de rotura (Break error).** Se produce cuando se detecta que la línea está a nivel bajo durante un tiempo mayor al que se tarda en enviar una trama completa. Esto es **así** ya que por defecto, cuando no se utiliza la **línea**, permanece a nivel alto.
- **Error de sobrepasamiento (Overrun error).** Se produce cuando la **CPU** aún no ha recogido los datos del controlador y de nuevo llegan nuevos datos que **sobreescreiben** los primeros.
- **Error de paridad (Parity error).** En **este** caso la paridad que calcula el receptor con los datos que le llegan no coincide con la paridad recibida.

7.3.1 La interfaz RS-232C

Este estándar lo incorporan todos los **ordenadores** personales y **está** definido por la **EIA (Electronic Industries Association)** aunque en Europa se le conoce como el estándar V24 definido por la **CCITT (Consultative Committee for International Telephone and Telegraph)**. En él se **definen** todas las características mecánicas, eléctricas y los protocolos necesarios para conectar un equipo terminal de datos (**DTE - Data Terminal Equipment**) con un equipo transmisor de datos (**DCE - Data Carrier Equipment**). Inicialmente se definió para realizar la comunicación entre un ordenador personal y un **módem**, aunque actualmente se **utiliza** con muchos otros propósitos para enviar datos de forma **serializada**.

En la tabla 7.1 se puede **observar** una tabla que recoge la **definición** de las **señales** tanto para los **conectores** de 25 patas **como** para los de 9 patas.

25 pin	9 pin	Señal	Significado	Dirección	Descripción
1	-	-	<i>Protective Ground</i>	-	Tierra de protección
2	3	* TD	<i>Transmitted Data</i>	DTE→DCE	Datos transmitidos
3	2	RD	<i>Received Data</i>	DCE→DTE	Datos recibidos
4	7	RTS	<i>Request To Send</i>	DTE→DCE	Petición para enviar
5	8	CTS	<i>Clear To Send</i>	DCE→DTE	Vía libre para el envío
6	6	DSR	<i>Data Set Ready</i>	DCE→DTE	Preparado para la conexión
7	5	-	<i>Signal Ground</i>	-	Señal de tierra
8	1	DCD	<i>Data Carrier Detect</i>	DCE→DTE	Detección de la señal portadora
20	4	DTR	<i>Data Terminal Ready</i>	DTE→DCE	DTE preparado para la conexión
22	9	RI	<i>Ring Indicator</i>	DCE→DTE	Indicación de llamada recibida
23	-	DSRD	<i>Data Signal Rate Detector</i>	DCE→DTE	Nueva tasa de transferencia

Tabla 7.1: Señales del estándar RS-232C

Cuando se conecta un módem al ordenador, la comunicación puede ser simple (**simplex**) en el caso de que sea únicamente el DTE el que envíe datos al DCE, funcionar en ambos sentidos de forma no simultánea (**half-duplex**) o realizarse una comunicación en **ambos** sentidos de forma simultánea, ya que existen dos líneas de datos, una de envío y otra de recepción (**full-duplex**).

Si se **realiza** una **comunicación** de tipo simple para transferir datos desde el DTE al DCE se utiliza la línea TD para transferir los datos, la línea DSR la **utiliza** el DCE para **sincronizarse** con el DTE y evitar por ejemplo los desbordamientos de datos y las demás líneas no se utilizan o están activas constantemente. Si los datos **se envían** en el sentido opuesto, el DCE utiliza la línea RD para transferir los datos al DTE. Puede utilizar también la línea DCD para indicarle al **DTE** que un dispositivo externo quiere realizar una **transferencia** y empezar la transferencia cuando el DTE le contesta que está disponible activando la línea **DTR**. Esta **misma línea** se **utilizará** para **sincronizar la transferencia**.

En una transferencia **half-duplex** se utilizan tanto la línea TD para transmitir datos como la RD para **recibir**, pero nunca se hará uso de las dos líneas de forma simultánea. Para **sincronizar la transferencia** se utilizan las líneas RTS y CTS. Cuando el DCE quiere enviar datos al DTE activa la línea DCD y espera a que el DTE le conteste activando la línea DTR.

En las transferencias **full-duplex**, **utilizadas** en la mayoría de las conexiones con los **módems**, se pretende que no haya esperas, por lo que a veces no se **utili-**

zan las líneas de **sincronización DTR, DSR, RTS y CTS**, que **permanecen** activas durante todo el tiempo (por ejemplo, en el caso de la **configuración null-modem**). En la tabla 7.2 se muestra un ejemplo del protocolo que se sigue cuando un dispositivo externo quiere conectarse con el host a través de un **módem**. Se ha implementado un protocolo de entendimiento **hardware** que utiliza las líneas de sincronización del estándar.

El **estándar** define **voltajes** que oscilan entre **+[3-15]V** para el nivel alto y **-[3-15]V** para el nivel bajo. Debido a la gran diferencia de voltaje que existe entre los niveles altos y bajos, se **permiten** tasas de transferencia de hasta 115.200 baudios si la longitud del cable es de unas pocas decenas de metros.

Si se utiliza este **estándar** para conectar otros **periféricos** diferentes de los **módems**, éstos se comportan como dispositivos **DTE** y por lo tanto las señales cambian de significado, e incluso las señales **DCD** y **RI** dejan de utilizarse formando una conexión llamada **null-modem** o **zeromodem**.

Si se conecta una impresora, el principal problema reside en que el PC puede transmitir los datos con gran rapidez y se puede desbordar el buffer de recepción de la impresora. La solución reside en conectar el pin 19 de la impresora, que indica que el buffer está **lleno** a la señal del ordenador **DSR**, indicando cuándo está preparado para recibir más datos.

7.3.2 El Bus Serie Universal (USB)

El USB es un **estándar** (1995) que define un bus utilizado para conectar periféricos al ordenador. La principal característica que tiene es que la conexión es muy sencilla, ya que utiliza un único **conector** para conectar a **través** de un bus serie todos los dispositivos. En él se definen los conectores y los cables, una topología especial tipo estrella para conectar hasta 127 dispositivos y protocolos que permiten la detección y configuración automática de los dispositivos conectados. USB soporta dos tasas de transferencia diferentes, una baja de 1'5 Mbps para la conexión de dispositivos lentos de bajo coste (joysticks, ratones) y otra alta de hasta 12 Mbps para la conexión de dispositivos que requieren un mayor ancho de banda (discos o CD-ROMS).

	← RI — -	← RI — A	← RI — -
	← DCD — -	← DCD — -	← DCD — -
X	— DTR → X	— DTR →	— DTR →
	← DSR — X	← DSR — X	← DSR — X
	← CTS — X	← CTS — X	← CTS — X
X	— RTS → X	— RTS →	— RTS →
	← RD —	← RD —	← RD —
	— m →	— TD →	— m →

1. Posición inicial de reposo

2. La unidad externa Intenta conectar con el PC. El módem activa RI.

3. El PC indica al módem que acepta la llamada activando DTR.

	← RI — -	← RI — -	← RI — -
	← DCD — -	← DCD — A	← DCD — A
A	— DTR → A	— DTR →	— DTR →
	← DSR — A	← DSR — A	← DSR — A
	← CTS — X	← CTS — A	← CTS — A
X	— RTS → A	— RTS →	— RTS →
	← RD —	← RD —	← RD — A
	— TD →	— TD →	— TD →

4. Cuando el módem termina de preparar la conexión activa DSR.

5. El módem activa DCD para indicar que la conexión está lista. El PC activa RTS para enviar datos y el módem responde con CTS.

6. Se transmiten datos hasta que se desactiva DCD, DSR o DTR.

Tabla 7.2: Protocolo full-duplex

La **especificación** de este **estándar** ha sido respaldada por las empresas líderes mundiales en el campo de la informática: **Intel, IBM, DEC, Microsoft, Compac, NEC** y **Northem Telecom**, empresas que garantizan la continuación y la utilización de este estándar.

Anteriormente los **periféricos** se conectaban **mapeados** directamente en **direcciones** de WS, se les asignaba una interrupción específica y en algunos casos un canal DMA. Esta **situación** conducía a tener conflictos en la **asignación** de estos recursos, puesto que siempre han estado bastante limitados en el ordenador. Además cada dispositivo tenía su propio puerto de conexión y utilizaba sus cables específicos, lo que daba lugar a un incremento de los **costea**. Debido a que a cada dispositivo se le tenían que asignar unos recursos **específicos** (sobre todo si es **plug-and-play**, es decir que el Sistema Operativo lo configura de forma automática) la detección del mismo debía hacerse a la hora de arrancar el sistema y nunca se podía incorporar un nuevo dispositivo cuando el sistema estaba en marcha.

Los dos aspectos fundamentales que motivaron la realización de este **estándar** fueron la necesidad de configurar de **forma** sencilla los periféricos conectados al ordenador y la necesidad de aumentar el **número** de puertos disponibles (habitualmente en los ordenadores personales de la década de los noventa **sólo** se disponía de 4 **ranuras PCI**, 4 ISA, 2 **puertos** paralelos y 2 serie).

Este **estándar** define una topología de conexión en estrella, tal como se muestra en la figura 7.1. por medio de la incorporación de varios concentradores (hubs) conectados en serie. Cada concentrador se conecta por un lado al computador (**llamado host** en el **estándar**) que contiene una o dos **interfaces** de este **tipo** en la **placa** base o a otro concentrador, y por otro lado se conecta a varios **dispositivos** o incluso a otro concentrador. De este modo **pueden** existir **periféricos** que vengan ya preparados con nuevos conectores **USB** para **incorporar** nuevos **dispositivos**. hasta un total de 127. todos ellos funcionando simultáneamente. Los hubs tienen la **misión** de ampliar el número de dispositivos que se pueden conectar al bus. Son **concentradores** cableados que permiten la **conexión** simultánea de múltiples dispositivos y lo **más** importante es que se pueden **concatenar** entre sí ampliando la cantidad de puertos disponibles para los **periféricos**. El **concentrador** detecta cuándo un **periférico** es conectado o desconectado a/de uno de sus puertos notificándolo de inmediato al controladordel USB. **También** realiza funciones de acoplamiento de las velocidades de los dispositivos más lentos.

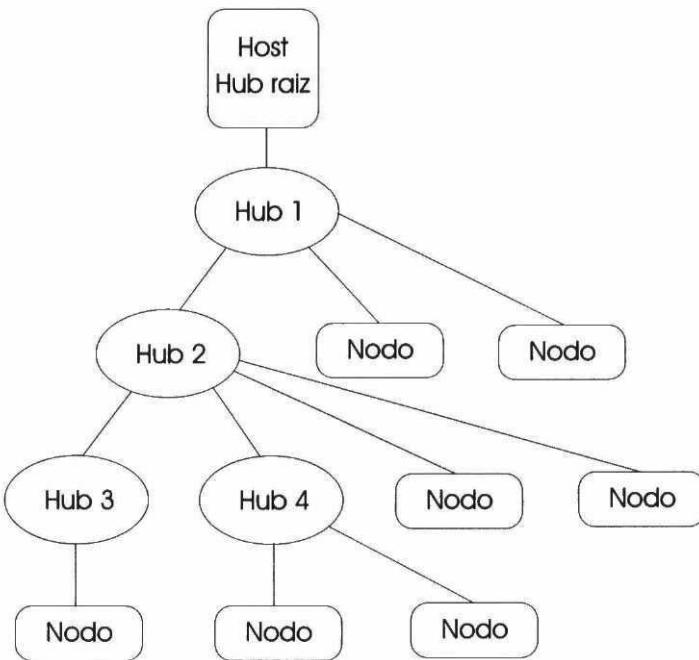


Figura 7.1: Topología de una conexión USB

Al conjunto de **dispositivos** que se **pueden** conectar al bus a excepción de los **concentradores** se les conoce en el **estándar** como **funciones**. Existen una gran variedad de dispositivos USB que se conectan todos al mismo bus: teclados, ratones, joysticks, impresoras, escáneres, cámaras digitales, video cámaras, altavoces, módems, adaptadores de red, discos duros y **concentradores**. La característica **más** importante es que todos los dispositivos utilizan el mismo tipo de cable y de **conector** y se conectan de la misma **forma** tan sencilla. El **host** decide qué dispositivo puede acceder al bus, utilizando un protocolo parecido al de paso de testigo. Este protocolo se caracteriza porque entre los **diferentes** dispositivos se va pasando un **identificador** a lo largo del tiempo que permite la **utilización** del bus.

El **host USB** **tiene** las funciones de:

- Detectar la **conexión/desconexión** de dispositivos y configurarlos.
- Controlar las transferencias de datos y de **control** que **tienen** lugar en el bus.

- Realización de **auditorías** sobre la actividad del sistema.
- **Servir** como fuente de alimentación a los dispositivos.

El USB define dos líneas para transmitir datos y otras dos para transmitir potencia (véase 7.2). Los datos se transmiten de forma balanceada a velocidades entre 12 **Mbps** y 1.5**Mbps**. La **señal** se transmite codificada en un código autoreloj de no retorno a cero invertido (NRZI) para poder incluir junto con los datos **información de sincronización**. (Se puede ampliar la información sobre el código **NRZI** en el capítulo 11). Las líneas de alimentación (**Vbus** y GND) evitan la necesidad de utilizar fuentes de alimentación externas. Tiene una tensión de **5V** y la corriente se limita a un máximo de 3 a 5 **amperios** por tazones de seguridad, siendo el consumo y la **configuración eléctrica totalmente transparente** al usuario. La distancia entre dos **periféricos** conectados al mismo cable no debe ser superior a los 5 metros para **evitar** problemas de caídas de tensión.

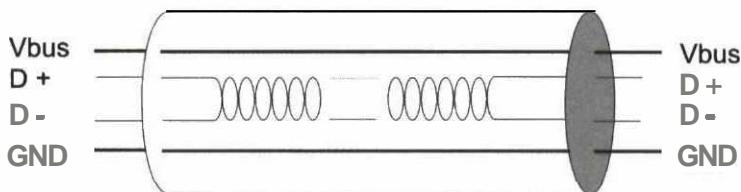


Figura 7.2: Formato del tipo de cable utilizado

El computador identifica automáticamente el dispositivo que se conecta mientras opera y lo configura sin tener que **instalar** drivers específicos del fabricante. Al **comienzo** se detectan los dispositivos conectados midiendo los niveles de voltaje de las líneas. Si un dispositivo está conectado, entonces el dispositivo envía información sobre el tipo o la clase de dispositivos a la que pertenece, qué modo de transferencia **utilizará** y cuáles son sus necesidades de ancho de banda. El host reconocerá el dispositivo buscando en la lista de drivers del sistema operativo y teniendo en cuenta los demás dispositivos conectados le asignará un ancho de banda determinado. De la misma forma también se pueden desconectar los dispositivos del sistema. El controlador USB del host asigna un número diferente de dispositivo a cada uno de los **periféricos** que se conectan a este bus. Para **empezar la transferencia**, éste **envía un paquete que** identifica al dispositivo objeto de la transferencia. El protocolo soporta cuatro tipos de transferencias:

- e **Control.** Son transferencias entre el host y el dispositivo que se utilizan para leer información de los descriptores en los registros de los dispositivos (llamados **end points**), interpretarla y poder **configurarlos**.
- **Interrupción.** Usado en los periféricos del tipo de los controladores de juegos, teclados y ratones, cuya comunicación es **unidireccional** y poco frecuente.
- e **Masiva (Bulk).** Son transferencias no periódicas que precisan de todo el ancho de banda disponible. Utilizado por las impresoras y los **scanners** y también para transferir imágenes.
- e **Isocrona.** Dedicada a las transferencias de telecomunicaciones, como voz o **vídeo**, que garantiza unas tasas de transferencias constantes. Se caracteriza porque el número de pulsos de reloj que transcurren entre la transmisión de dos caracteres es constante, por lo tanto se está enviando **información** constantemente entre el host y el dispositivo.

Intel ha implementado un conjunto de controladores de USB para **periféricos** de **PCs** (**8x930Ax** USB Peripheral Controller) y para los concentradores (**8x930-Hx** USB Hub Controller y el **8x931Ax** USB Peripheral Controller).

Los controladores USB constan de las siguientes partes:

- e Transceivers para adecuar las **señales** del bus.
- **Interfaz serie (Serial Interface Engine - SIE)** que serializa los datos y se encarga además de realizar la codificación NRZI, el control de errores, el control de los protocolos y el secuenciamiento de los paquetes.
- Unidad de **interfaz** (Function **Interface Unit - FIU**) que **monitoriza** el estado del controlador, las transacciones y los buffers de datos, e interrumpe también a la CPU.
- Buffers de almacenamiento temporal de datos que son memorias FIFO (primero en entrar, primero en **salir**) destinadas a la transmisión y a la recepción de datos, **así** como al control de interrupciones y a las transferencias tanto **isocrónicas** como masivas (bulk).

La **versión** 2.0 de este **estándar**, que está siendo desarrollada actualmente, multiplica la velocidad del bus por un factor de 30 o 40, utilizando los mismos cables, **conectores** e interfaces software.

7.4 INTERFACES PARALELAS

Los ordenadores personales **incorporan** tradicionalmente un puerto paralelo consistente en un **conector** DB25 de 25 **pines**. Este **tipo** de interfaz se caracteriza porque se **envían** simultáneamente **los** bits de datos por medio de diferentes **líneas**. **Desde** siempre se ha considerado **la interfaz** paralela como el puerto utilizado para conectar la impresora, pero desde comienzos de la **década** de los noventa se viene utilizando con otros **fines**, ya sea para comunicar diferentes sistemas **informáticos** o bien para conectar dispositivos de almacenamiento masivo. La clave para su **expansión** fue la utilización de **estándares** que permitían la **comunicación bidireccional** por las **líneas** de datos.

7.4.1 La interfaz Centronics

Inicialmente (mediados de los **años** sesenta) **se diseñó** una interfaz con 36 pines, que utilizaba la casa Centronics Data **Computer Corporation** en sus **impresoras**, y es por ello por lo que se le conoce como la interfaz **Centronics**. Sin embargo la interfaz Centronics de los ordenadores personales actuales fue diseñada por **Epson Corporation**.

La interfaz consta de 8 pines para datos **más** 5 **señales** que controlan la **impresora** y cinco que vienen de la misma. Se utilizan **voltajes TTL** con señales no balanceadas (en el destino el valor de la señal se obtiene en referencia a otra **señal** de tierra), por lo que son susceptibles de recibir ruido y producir **errores**. Si se utilizan cables normales de impresora, la longitud máxima del cable ronda los 4'5 metros, aunque se puede **aumentar** hasta 15 metros si se utilizan cables especiales. El bus soporta **tasas** de transferencia de datos de hasta 100 **kbytes/s**.

En la tabla 7.3 se pueden observar las **señales** que componen el **estándar Centronics**, su **descripción**, **dirección** y su **localización física**, tanto en el **conector DB25** del PC como en el **conector Centronics** de la impresora.

Número de pin		Nombre	Dirección		Función
DB-25	Centronics		PC	Impre	
1	1	/Strobe[a]	→		Cuando esta señal se activa indica que los datos que se envían a la impresora ya están estables en el bus desde hace 0.5 μ s. La impresora leerá los datos en el momento que detecte el flanko de bajada de esta señal.
2	2	Data bit 0	→		
3	3	Data bit 1	→		
4	4	Data bit 2	→		
5	5	Data bit 3	→		Datos hacia la impresora.
6	6	Data bit 4	→		
7	7	Data bit 5	→		
8	8	Data bit 6	→		
9	9	Data bit 7	→		
10	10	/Ack	←		Es la señal de reconocimiento que envía la impresora para indicar que ya ha procesado los datos y que puede recibir más.
11	11	Busy[a]	←		La impresora activa esta señal para indicar que está procesando los datos que acaba de leer. Hasta que no se desactiven tanto Busy como /Ack no podrá aceptar más datos.
12	12	PE	←		La impresora se ha quedado sin papel o hay un atasco del mismo.
13	13	Slct	←		Impresora seleccionada o activa (on-line).
14	14	/Auto Fd[a]	→		Cuando el PC activa esta señal le indica a la impresora que debe avanzar una línea cada vez que reciba un carácter de retorno de carro.
15	32	/Error or /Fault	←		La impresora indica una condición de error.
16	31	/Init	→		El PC pide la reinicialización de la impresora (vaciado del buffer y vuelta al principio de la línea).
17	36	/Select In[a]	→		El PC selecciona la impresora, por lo que ésta ya puede aceptar datos.
18-25	16, 19-30, 33	Ground	↔		Tierra.

Tabla 7.3: Señales que componen el estándar Centronics

Actualmente se han **diseñado** dos **estándares** que tratan de aumentar el ancho de banda de la interfaz Centronics sin perder la compatibilidad con el mismo, permitiendo **además** la comunicación bidireccional. Son las **interfaces** ECP (*Extended Capabilities Port*) y EPP (*Enhanced Capabilities Port*) que se definen en el **estándar** del IEEE 1284 (*IEEE Std. 1284-1994 Standard Signaling Method for a Bi-Directional Parallel Peripheral Interface for Personal Computers*). ECP se utiliza en las impresoras y **escáneres**, puesto que permite mayores tasas de transferencia con protocolos sencillos, mientras que EPP sirve para los demás dispositivos en donde se necesita un control de errores **más** exhaustivo.

7.4.2 El estándar IEEE 1284

Este nuevo **estándar** define 5 modos de transferencia de datos, desde el viejo Centronics hasta dos métodos que permiten la comunicación bidireccional entre el ordenador y el dispositivo. Debido a que los protocolos se implementan por hardware, EPP y ECP permiten tasas de transferencia de datos mucho mayores, llegando incluso al Megabyte por segundo. La tabla 7.4 describe los modos de transferencia de datos que define el estándar.

El **estándar** describe el formato de las **señales**, la asignación de pines y los mecanismos de detección y corrección de **errores**, sin embargo las funciones de la BIOS, la interfaz software y el control de los puertos están a cargo de los fabricantes. La tabla 7.5 describe los tipos de **conectores** que existen.

El puerto paralelo se configura inicialmente en el modo compatible. **Después** se establece un diálogo con el periférico para decidir el modo de funcionamiento final, aunque debido a la facilidad con la que se puede cambiar el modo, es posible **realizar** transferencias cambiando los modos de emisión y de recepción de datos de forma dinámica. Los modos byte, ECP y EPP son opcionales en el **estándar**.

7.4.3 Small Computer Systems Interface (SCSI)

La interfaz SCSI es una interfaz paralela, con 8, 16 o 32 líneas de datos que se utiliza para comunicar dispositivos **rápidos**, como discos CD-ROM, **dispositi-**

Modo	Características	Velocidad (kbytes/s)	Apropiado para
Compatible	Salida de 8 bits hacia la impresora. Monitorización restringida de la impresora. I/O programada basada en interrupciones. Sincronización del control de flujo.	100 a 200 sólo salida	PC y Centronics original.
4-bits	Entrada de 4 bits de la impresora. Modo compatible para las salidas. I/O programada basada en interrupciones.	40 a 60 de entrada	Impresoras y otras aplicaciones que pueden utilizar los 4 bits de estado para enviar datos al PC.
8 bits	Entrada de 8 bits de la impresora. Modo compatible para las salidas. I/O programada basada en interrupciones.	80 a 300 de entrada	Impresoras y otros periféricos con mayores necesidades de ancho de banda (discos o interfaces de red). Lo incorporan los PCs a partir del 1993. Existe un bit que controla la dirección del puerto.
ECP	8 bits de entrada y 8 de salida. I/O programada basada en interrupciones y DMA. Usan FIFOs de 16 bytes o más para acelerar la transferencia. Implementación del protocolo por hardware (El sw no se preocupa de activar/desactivar las señales de reconocimiento). Posibilidad de indicar si la transferencia es de datos o de órdenes mediante la activación de una línea.	Más de 2,000	Tasa de transferencia máxima. Soporta compresión RLE (en lugar de enviar muchos bytes iguales, se envía el byte original y el número de veces que se repite). Apropiado para conectar escáneres y CD-ROMS. Desarrollado por HP y Microsoft
EPP	8 bits de entrada y 8 de salida. I/O programada basada en Interrupciones (DMA NO). Incluye direccionamiento para soportar más de un periférico conectado por <i>daisy-chain</i> al cable. Implementación del protocolo por hardware (El sw no se preocupa de activar/desactivar las señales de reconocimiento).	Hasta 2000	Comunicaciones interactivas con controladores de red, CDROM, discos y cintas. Es el modo más potente y flexible.

Tabla 7.4: Modos de transferencia de datos del Estándar IEEE-1284

Tipo de Conector	Descripción	Distancia (m)	Comentarios
A	DB-25 estándar.	2	Puerto paralelo estándar del PC (hembra).
B	Conector centronics de 36 pines.	2	Puerto paralelo estándar en la impresora.
C	Conector centronics de 36 pines miniatura.	10	Nuevo estándar de conexión, con nuevos voltajes que permiten un mayor ancho de banda. Desarrollado por Intel, Xircom, and Zenith.

Tabla 7.5: Tipos de conectores del estándar IEEE-1284

vos de audio y dispositivos de almacenamiento **externo** de datos. Normalmente se considera a la **configuración** SCSI como un bus (conexión multipunto), sin embargo los dispositivos están encadenados entre **sí** formando una conexión **daisy-chain**. Cada dispositivo tiene dos conectores, uno de entrada y el otro de salida. El comienzo del bus se conecta con el **host** y el último dispositivo incorpora un **terminador** para evitar problemas de reflexiones de las **señales**. Los dispositivos funcionan de forma independiente y pueden intercambiar datos tanto entre sí como con el **host**.

Este bus puede soportar múltiples procesadores y múltiples dispositivos periféricos. Soporta hasta 8 dispositivos de los cuales cada uno puede tener 8 unidades lógicas, cada una de las cuales soporta 256 subunidades lógicas.

La especificación original se llamó SCSI-1 y usaba 8 líneas de datos a una frecuencia de **5MHz**, permitiendo una transferencia de datos de 5 Mb/seg. SCSI-1 soporta hasta 7 dispositivos que pueden ser encadenados al bus.

En 1991 surgió una extensión al estándar, el SCSI-2, que **incrementaba** el número de **líneas** de datos a 16 o 32 bits e incrementaba la frecuencia de reloj a **10 MHz**. Así se logran tasas de transferencia máxima de hasta **40 Mbytes/seg**.

Las transferencias en el bus siempre tienen lugar entre un iniciador (dispositivo que manda comandos) y un objetivo (dispositivo que ejecuta los comandos). Normalmente el **host** es el iniciador y el controlador del dispositivo es el objetivo, aunque puede haber algún dispositivo que sea ambas cosas a la vez. Las señales que se transmiten por el bus pueden estar implementadas utilizando un

solo cable cada una y compartiendo una masa común en el caso de un **single-ended SCSI** o utilizando dos cables cada una en el caso del **differential SCSI**. El primero se utiliza para distancias menores a 6 metros y el segundo para distancias menores a 25 metros. Los conectores son de 50 pines.

En la figura 7.3 se muestran las señales de este bus. Existen 9 **líneas** de datos y 9 señales de protocolo que coordinan la transferencia entre el controlador y el host. A continuación se presenta una lista que describe la **funcionalidad** de las **señales** de protocolo del bus.

- **Ocupado (BUSY).** Esta señal la activa el iniciador o el objetivo para indicar que el bus **está ocupado**.
- **Selección (SELECT).** Esta señal **la** utiliza el iniciador para seleccionar un objetivo. o el objetivo para reseleccionar al iniciador. La señal de E/S distingue entre selección y reselección
- **Control/Datos (CONTROL/DATA).** El objetivo indica si la **información** en el bus de datos es de control (orden, **estado** o mensaje) o de datos.
- **E/S (I/O).** La utiliza el objetivo para controlar la dirección del movimiento de datos.
- **Mensaje (MESSAGE).** Con ella el objetivo le indica al iniciador que la **información** que se transmite es un mensaje.
- **Petición (REQUEST).** La usa el objetivo para pedir una transferencia de datos. En respuesta a **la** activación de **esta** señal, el iniciador acepta datos del bus durante la fase de entrada de datos o pone datos **en** el bus durante la fase de salida de datos.
- **Reconocimiento (ACKNOWLEDGE).** La usa el iniciador para reconocer una petición del objetivo. Indica que el iniciador ha puesto los datos **en** el bus o que los ha recogido.
- **Atención (ATTENTION).** El iniciador **indica** al objetivo que tiene un mensaje listo para ser transferido.
- **Inicialización (RESET).** Utilizado para inicializar el bus

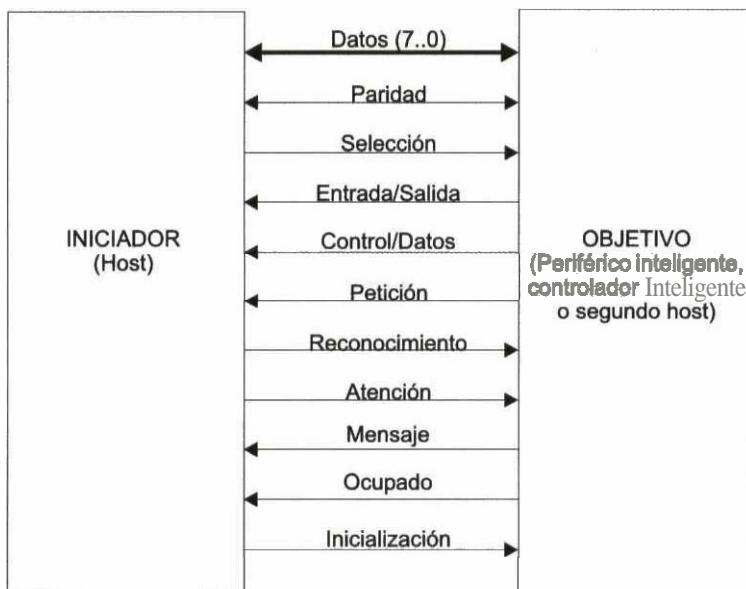


Figura 7.3: Descripción de la interfaz SCSI

Las transferencias que tienen lugar en el bus se dividen en varias fases, tal como muestra la figura 7.4. Como se puede observar no todas las transferencias tienen que pasar por todas las fases para completarse. A continuación se describen las fases por las que se pasa en el protocolo SCSI.

1. **Bus libre.** No es una fase propiamente dicha, ya que ningún dispositivo está utilizando el bus.
2. Fase de arbitraje. Antes de empezar o continuar con la **operación de E/S** en necesario ganar el control del bus, ya que puede haber múltiples iniciadores de la transferencia.
3. Fase de **selección**. El iniciador selecciona un objetivo. Éste contesta con los tipos de transferencia que es capaz de soportar: **entrada/salida** de datos, petición de comando, reconocimiento de estado o **entrada/salida** de mensaje.
4. Fase de **reselección**. El objetivo se vuelve a conectar con un iniciador con la intención de continuar con la operación que él mismo suspendió.
5. Fase de orden. El objetivo le pide al iniciador información sobre la orden.

6. Fase de datos. El objetivo le pide al iniciador que realice la transferencia de datos. Los dispositivos se comunican mediante un conjunto de comandos de alto nivel que se envían en los **bytes de descripción** de los paquetes.
7. Fase de estado. El objetivo pide permiso para poder enviar **información** de estado.
8. Fase de mensaje. El objetivo pide la transferencia de uno o más mensajes.

Cuando en una transferencia ya se ha arbitrado el bus y se han elegido tanto el iniciador como el objetivo se pueden realizar una o varias fases de transferencia de información (orden, datos, estado y mensaje). La última fase suele ser la de entrada de mensaje, y en ella se transmite un mensaje al iniciador de desconexión o de finalización de orden.

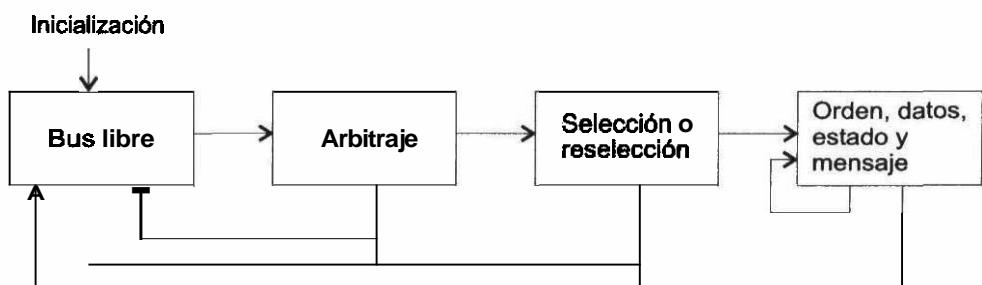


Figura 7.4: Fases en una transferencia SCSI

Una característica importante del SCSI es la capacidad de reselección. El protocolo permite que si la orden tarda mucho tiempo en completarse, el objetivo libere el bus y se reconecte después, como, por ejemplo, para formatear un disco. Con esta política, es posible que un dispositivo reciba varios comandos antes de ejecutar el primero, con lo que puede realizar una selección fuera de secuencia para optimizar el tiempo de respuesta.

Temporización del bus

Inicialmente ninguna señal está activa. La fase de arbitraje comienza cuando uno o más dispositivos activan la **línea BSY** y una de las líneas de datos

que indica su **identificativo** (0-7). El dispositivo con mayor prioridad es el de **identificativo** mayor y es el que gana el arbitraje.

El dispositivo que gana el arbitraje es el iniciador. Entra **en** la fase de **selección** mediante la **activación** de la **señal SEL**. En **esta** fase activa su **ID** y la línea de datos que **corresponde** con el W del objetivo, y después de un retraso desactiva la señal BSY. Cuando el objetivo detecta activada SEL, y desactivadas BSY e **I/O**, y **además detecta** su **identificativo** activado en la línea de datos, activa la señal de BSY. Cuando el iniciador detecta BSY, desactiva el bus de datos y niega **SEL**.

A **continuación**, el objetivo entra en la fase de orden mediante la activación de la línea **C/D** y pide el primer byte de la orden al iniciador mediante la **activación** de **REQ**. El iniciador, cuando deposita en el bus de datos el primer byte de la orden, activa **ACK**. **Después** de que el objetivo lee el byte, **desactiva** **REQ** y entonces el iniciador **desactiva** **ACK**. El primer byte de la orden contiene el **código** de **operación** de la orden que indica cuantos bytes quedan aún por transferir. Esta transferencia **se produce** del mismo modo que la anterior mediante las líneas de protocolo **REQ/ACK**.

Después de que el objetivo ha recibido y ha interpretado la orden coloca el bus **en** la fase de datos negando la línea **C/D** y activa la línea **U 0** para indicar **una** fase de Data **In** (la dirección de la transferencia es de objetivo a iniciador). La transferencia **se realiza** de la misma forma que **se transmitió** la orden.

Cuando ya se han transferido todos los datos que se pidieron, el objetivo pone al bus en la fase de estado y transfiere un byte de estado al iniciador indicando el éxito de la transferencia. En este caso la **línea C/D** se activa de nuevo y la **línea I/O permanece** activada. También aquí **se utiliza** el protocolo **REQ/ACK**.

Finalmente, para concluir con la **transacción** el objetivo pone al bus en la fase de mensaje activando la línea **MSG** y transfiere un byte que contiene el mensaje de orden terminada. Cuando este byte es recibido por el iniciador, el objetivo **desactiva** **todas** la líneas para dejar al bus en estado de no utilización.

Mensajes

Existen 3 tipos de **formatos** de mensajes: de un byte, de dos **bytes** y mensajes extendidos de tres o más **bytes**. Algunos ejemplos de mensajes son:

- Orden **completada**. El objetivo le indica al iniciador que ha terminado la orden y que se le ha enviado un mensaje de estado válido.
- Desconexión. El objetivo indica que se va a desconectar del bus y que realizará una reconexión posterior para completar la orden.
- **Error** detectado por el iniciador. El iniciador comunica que ha detectado un error.
- Terminación. El iniciador quiere terminar la transferencia actual.
- **Transferencia** de datos **síncrona**. Para establecer que la transferencia se va a realizar de forma **síncrona**.

7.5 CONCLUSIONES

En este capítulo se ha definido el enlace exterior como el medio de comunicación que interconecta el controlador de periférico con su periférico asociado.

Además se han clasificado las interfaces en función del número de líneas utilizadas para la transmisión de los datos y del número de dispositivos conectados y se han introducido las interfaces **serie** (RS232) y paralelo (IEEE 1284) como ejemplo de la gran diversidad de estándares que existen.

También se ha explicado de forma mas exhaustiva un bus exterior paralelo: el **SCSI**. El objetivo de este punto ha sido el de conocer las señales que lo forman, su funcionalidad y los protocolos que utiliza para realizar el arbitraje del bus, las transferencias de datos y el envío de mensajes.

Con la finalización de este capítulo, el lector debe ser capaz de representar el diagrama de bloques completo de un computador sencillo, incluyendo los periféricos, puesto que ya se han contemplado todos los bloques funcionales que forman la arquitectura de un computador.

7.6 CUESTIONES

7.1 *Se quiere enviar uno secuencia de 3 caracteres (15h, 25h y A0h) por una línea serie utilizando un protocolo asíncrono. Se configuran tanto el emisor como el receptor para enviar/recibir caracteres a una velocidad de 10 bits por segundo, con paridad par y un bit de stop. Realizar un esquema temporal que muestre el estado de la línea de transmisión incluyendo las celdas de bit. ¿Cómo sabe el receptor cuando debe muestrear la línea?*

7.2 *Explicar las diferencias existentes en cuanto a la metodología de conexión de un bus USB y de un bus SCSI.*

7.3 *¿Qué tipo de arbitraje utiliza el bus SCSI? Indicar cómo tiene lugar el proceso de arbitraje.*

7.4 *¿Por qué la señal de selección en el bus SCSI es bidireccional?*

7.7 BIBLIOGRAFÍA

- e **Computer Peripherals.** Bany M. Cook, Neil H. White. 3^a ed. Edward Arnold. ISBN 0-340-60658-4. 1995.
- e **An Introduction to Microcomputer Systems. Architecture & Interfacing.** John Fulcher. Addison Wesley Publishers Ltd. ISBN 0-201-41623-9. 1989.

- The indispensable PC Hardware Book. *Hans-Peter Messmer*. 3^a ed. Addison Wesley Longman, Inc. 1997. ISBN 0-201-40399-4.
- El Libro del RS232. *Joe Campbell*. Ediciones Anaya Multimedia, S.A. ISBN 84-7614-055-X. 1985.
- Conexiones en el IBM /PC/XT/AT. Teoría y práctica de periféricos, comunicaciones y configuraciones. *M. D. Seyer*. Ediciones Anaya Multimedia, S. A. ISBN 84-7614-119-X. 1986.
- Guía a las Comunicaciones del IBM/PC. *David Kruglinski*. McGraw-Hill, Inc. ISBN 84-7615-057-1. 1984.
- Conceptos Actuales Sobre la Tecnología de los Ordenadores. *Joseph C. Giarratano*. Ediciones Díaz de Santos, S. A. ISBN 84-86251-15-X. 1984.

CAPÍTULO 8

DISPOSITIVOS DE ENTRADA DE DATOS

8.1 INTRODUCCIÓN

Los medios a través de los cuales llega información al procesador son muy dispares. A lo largo de este capítulo únicamente se van a considerar aquellos dispositivos que utiliza el usuario para introducir información en el ordenador.

Debido a la condición humana estos dispositivos son bastante lentos y no requieren un ancho de banda demasiado elevado, por lo que el diseño de las *interfaces* con el procesador es bastante sencillo. Habitualmente se utilizan protocolos de comunicación de datos serie para transmitir la información de entrada, que puede variar desde un conjunto de caracteres alfanuméricos introducidos por el teclado, a una coordenada de la pantalla, o incluso a un conjunto de bits que describen la forma de una imagen **digitalizada**.

Todos estos dispositivos se conectan al sistema **informático** por medio de **buses estandarizados** para abaratar los costes de producción y facilitar el intercambio entre diferentes sistemas. Actualmente el bus **USB** va reemplazando a las *interfaces* tradicionales Centronics y **RS232**.

A continuación se van a describir los dispositivos más típicos, desde el teclado a los ratones, pantallas sensibles al tacto o digitalizadores de imágenes.

8.2 INTERRUPTORES

El dispositivo **más** simple de entrada es el **interruptor** o el pulsador. Son dispositivos electrónicos que están en estado abierto (no conducen la corriente eléctrica) o cerrado (conducen la **corriente** eléctrica). La concepción más simple de interruptor consiste en dos hilos que están en contacto o no lo están. Para activar el **interruptor** se unen y para desactivarlo se separan. Entre los diferentes tipos que existen destacan los siguientes:

- **Mecánicos:** Dos contactos mecánicos entran en contacto para cerrar un circuito, tal como se puede observar en la figura 8.1 (a). Al presionar el **interruptor** los contactos se tocan o se **separan** fijando un voltaie u otro. Estos **interruptores** tienen una pequeña **resistencia** del orden & 0'1 ohmios que suele aumentar con el tiempo. En los interruptores más modernos los contactos están bañados en oro para evitar su **oxidación** y por el mismo hecho de accionar el interruptor un mecanismo limpia los contactos.

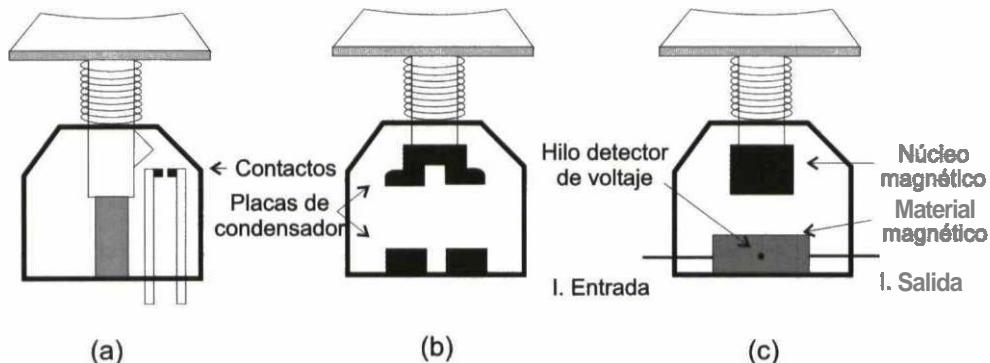


Figura 8.1: Diferentes tipos de interruptores

- De membrana: Tiene tres capas de un material formado a base de películas de **poliéster** o **policarbonato**, separadas por **espaciadores**. El conjunto se ensambla ocupando menos de un milímetro. La capa del fondo **es** fija, mientras que la **intermedia** y la superior son flexibles, teniendo esta **última** impreso en la cara exterior el dibujo del carácter que representa la tecla. Los contactos se localizan en las caras internas de la membrana inferior y la media. Al presionar la capa superior el nivel intermedio presiona al nivel inferior **formando** el contacto eléctrico. Presentan la ventaja de que

son muy delgados y baratos, y además son herméticos, lo que los hace más duraderos, pudiendo ser incluso hasta sumergibles.

- **Capacitivos:** Como se puede observar en la figura 8.1 (b), estos interruptores internamente tienen dos placas de condensador en la base del interruptor. Al presionar se acerca una tercera placa que produce una variación de tensión entre las dos placas de la base. La calidad de los interruptores de este tipo es bastante elevada, ya que no existen contactos mecánicos que se puedan estropear o desgastar con el tiempo.
- De efecto Hall: El efecto *Hall* produce una diferencia de potencial entre los extremos de una pieza hecha a base de un material semiconductor cuando fluye la corriente a través de la misma y el material sufre los efectos de un campo magnético perpendicular a la dirección del flujo de la comente. La figura 8.1 (c) muestra un interruptor de estas **características**. Cuando se presiona la tecla, al acercarse el núcleo magnético produce una variación del flujo magnético que atraviesa la pieza del fondo del interruptor. Esta variación de flujo produce un cambio en la tensión del semiconductor que sirve para detectar la pulsación de la tecla.

8.2.1 El problema de los rebotes



Figura 8.2: El problema de los rebotes

Desde el momento en que se dispone de un interruptor en un sistema informático, como, por ejemplo, para generar la señal de reset de un **microcontrolador** (véase la figura 8.2). se deben poner circuitos para evitar las oscilaciones que produce la señal debido a la acción de los contactos. El tiempo de respuesta del interruptor es varias órdenes de magnitud más lento que el ordenador, por lo tanto cada vez que se presiona el interruptor, las oscilaciones que produce pueden ser interpretadas como nuevas entradas. Para solventar este problema se pueden aplicar varias técnicas que se resumen a continuación:

- Filtro RC: Se utiliza un **integrador** con una constante de tiempo RC que determina la velocidad con la que se va cargando **asintóticamente** el condensador hacia Vcc. Se debe elegir el **filtro** para que un rebote en el **interruptor** no sobrepase nunca el nivel **bajo** & la **señal**. La figura 8.3 muestra el efecto que se consigue con el filtro en la **señal** que procede del interruptor.

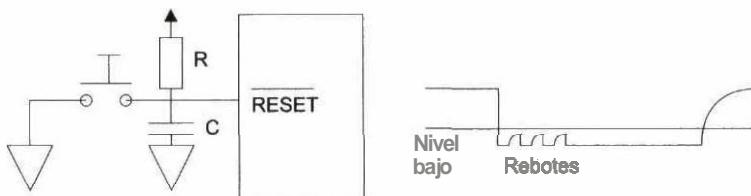


Figura 8.3: Circuito con una red RC para evitar rebotes

- Monoestable: Este dispositivo en el momento en que se detecta un flanco de bajada en la señal de entrada produce un pulso de amplitud fija. Mientras esté activo el pulso, subsiguientes flancos en la entrada serán ignorados, con lo que se eliminan por completo los rebotes.
- Biestable SR: Debido a la inclusión de un biestable SR **sólo** se responde al primer estado bajo que activa el Reset, como en el caso anterior. La figura 8.4 muestra un ejemplo de este circuito.

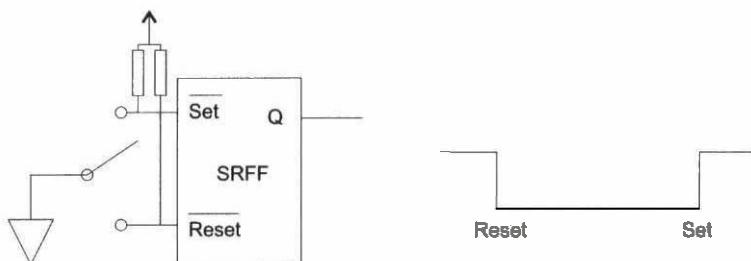


Figura 8.4: Circuito con un biestable RS que evita los rebotes

8.3 TECLADOS

La **mayoría** de los teclados suelen tener alrededor de 100 teclas. El sistema **informático** debe reaccionar a cada una de ellas independientemente. No sería práctico tener una línea dedicada para cada tecla, y en cualquier caso, es bastante raro que varias teclas se presionen a la vez, por lo tanto se emplea un canal compartido.

Las teclas **se** configuran en forma de matriz de m filas por n columnas. Para controlarlo **sólo** se necesitan conectar $m + n$ cables a un puerto del ordenador, pero hace falta la ayuda adicional del *software* para identificar la tecla pulsada.

Si se tiene una matriz de 3×3 , como muestra el ejemplo de la figura 8.5, se conectarán las 3 líneas de las **filas** a un puerto de salida, **con** lo que el **procesador** podrá cambiar el estado (alto o bajo) de cada una de ellas. Hay que destacar que la **línea** discontinua del dibujo indica el sentido que sigue la **comiente eléctrica**. Como el primer puerto de **salida** muestra un nivel bajo tiene que absorber corriente. Las columnas se conectarán a un puerto de entrada, de forma que el procesador **podrá** averiguar, leyendo del puerto, cuál es su valor. El procedimiento para interpretar las pulsaciones consiste en que a medida que se van escribiendo ceros en las **filas** se van leyendo de forma simultánea los valores de las columnas.

Cuando **se** escribe un '0' por la fila 1, siempre que se detecte un '0' en la columna 1 significa que se ha presionado la tecla **(1,1)**. En memoria se tendrá una tabla que indique **cuál** es la tecla que se tiene en esta posición. Existen 8 posibilidades para cada fila, indicando desde las tres teclas pulsadas simultáneamente (000) hasta ninguna tecla pulsada (111). Esta técnica de **decodificación** por *software* consume mucha memoria debido a la **gran** cantidad de posibilidades que se pueden obtener, por lo tanto normalmente no se tienen en cuenta las pulsaciones múltiples en este tipo de teclados cuando se aplican estos algoritmos & interpretación de las teclas pulsadas.

Además, este método puede presentar **errores** en la **codificación** de las teclas que se han pulsado cuando se activan varias a la vez. Para ilustrarlo se va a considerar la misma matriz **de** teclas del ejemplo anterior y se va a suponer que se pulsan de forma simultánea 3 teclas, las dos últimas de la primera fila y la tecla del medio de la segunda fila, tal y como muestra la figura 8.6.

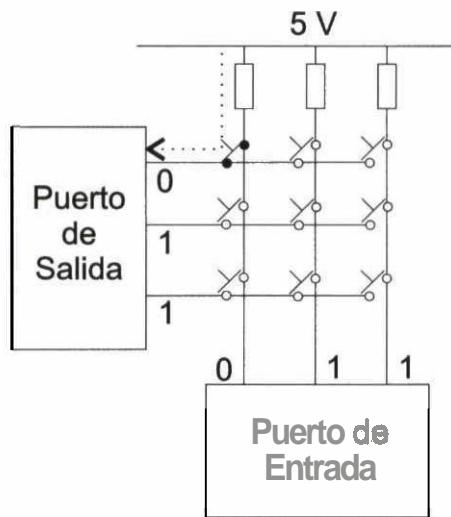


Figura 8.5: Lectura de la primera fila

Cuando el algoritmo pasa a **comprobar** la fda 2, como **únicamente está pulsada** la tecla del medio de dicha fila, en el puerto de entrada se debe leer la secuencia 1 0 1. Sin embargo se detectan **niveles bajos** en las dos últimas columnas (1 0 0) con lo que se interpreta que también **está pulsada** la ultima tecla de la fila y por lo tanto se produce un error. Hay un 0 no esperado debido al camino alternativo que sigue la comente desde la columna 3 hacia la fila 2. Si **se enumeran** los intemptores por pares de **filas** y columnas, desde el momento en que el puerto de salida activa un cero en la fila 2, empieza a adquirir corriente que es suministrada por **las** columnas 2 y 3, pasando respectivamente por los **interruptores (1,2)(2,2)** y por **(1,3)(1,2)(2,2)**. Debido al flujo de corriente por la columna 3 se produce una caída de potencial en la resistencia que produce la lectura del 0 **errónea** en el puerto de **entrada**. La **solución** consiste en **añadir diodos** en los **interruptores** para evitar la vuelta atrás de la comente desde las filas a las columnas cuando el pulsador está apretado.

Otro de los problemas que presenta esta técnica consiste en que el **procesador** tiene que estar todo el tiempo comprobando **las filas secuencialmente** para detectar las pulsaciones de las teclas. por lo que se pierde eficiencia. Para solucionarlo se emplean otros algoritmos en los cuales el **procesador** activa todas las filas a la vez y pasa a realizar alguna otra tarea en espera de la **pulsación** de alguna tecla (detectada por la presencia de **algún cero** en alguna columna). En

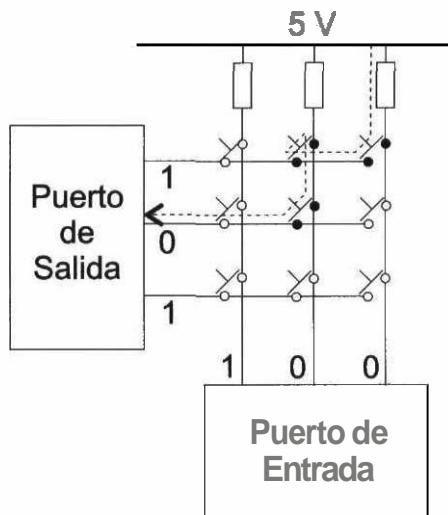


Figura 8.6: Camino alternativo que sigue la corriente

el momento de la detección el puerto de entrada es el que envía una interrupción al **procesador** para avisarle de la pulsación y para que ejecute el **driver** de teclado que se encarga ya de realizar la comprobación de todas las filas de forma secuencial para **identificar** la tecla pulsada.

El teclado de un ordenador se controla de forma diferente a las **vistas** anteriormente. En la figura 8.7 se puede observar un esquema de un teclado con su conexión a la placa del **procesador**. Éste es un dispositivo de entrada de datos que se conecta al controlador de teclado o a la **interfaz** de teclado de la placa base del ordenador mediante una **interfaz** serie. Los teclados del ordenador consisten en una matriz de teclas y un circuito integrado que se encarga de **supervisar** esta **matriz** y de detectar las pulsaciones. En el momento en que **se** detecta la pulsación o la liberación de una tecla se escribe su código correspondiente en un **buffer** interno del teclado y a continuación el teclado transmite este código al controlador que se encuentra en la placa base del **procesador** **vía** serie a través del cable del teclado. Las funciones del **controlador** incluido en el teclado se pueden resumir como las siguientes:

- Prevenir las falsas repeticiones (rebotes) y evitar la detección de pulsaciones incorrectas debidas a la vuelta atrás de la comente mediante la **inclusión** de **diodos** en las teclas.

- e Traducir la tecla pulsada, identificada por su posición en la matriz, en un código único, llamado ***scan-code***, que la **identifica**. Con un byte es suficiente para almacenarlo, ya que los teclados de ordenador suelen tener alrededor de 105 teclas.
- e Repetir un carácter si se ha pulsado durante cierto tiempo (normalmente del orden de un segundo).
- e Detectar la pulsación simultánea de varias teclas (***rollover***): Cuando se pulsa una tecla, se genera su código correspondiente. Si la primera tecla permanece apretada mientras se pulsa una segunda, se **generará** la salida correspondiente a la segunda tecla. Si se pulsa una tercera tecla mientras las anteriores están apretadas (o alguna de **ellas**) se genera el **código** de la tercera. Así sucesivamente hasta que se pulsan todas las teclas menos una.

El **conector** del teclado dispone de 5 hilos para transmitir las señales de reloj, datos, reset, tierra y alimentación, con lo que se puede establecer una comunicación **bidireccional** entre el ordenador y el teclado con propósitos de transferencia de datos o de configuración (como, por ejemplo, para programar la velocidad de repetición de la tecla pulsada, o para deshabilitar el teclado). La línea de reset sirve para inicializar el teclado.

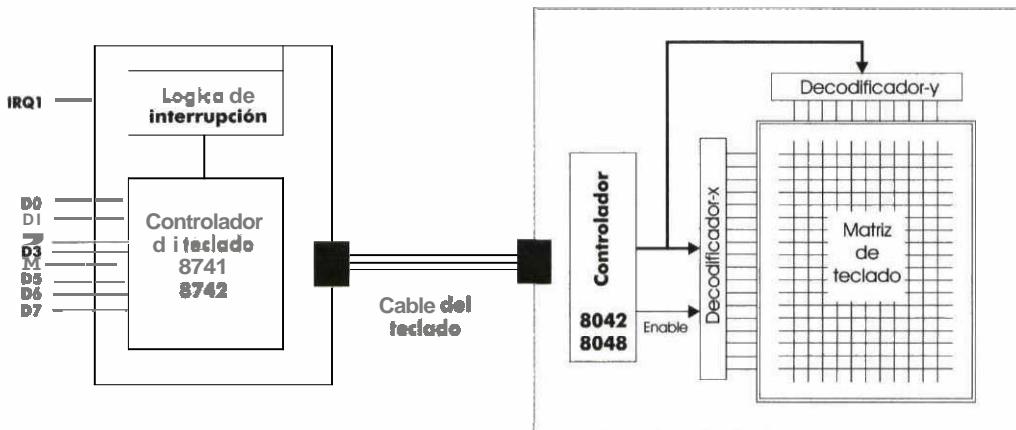


Figura 8.7: Esquema de la conexión de un teclado

Cuando se produce una pulsación, el controlador transfiere un código al ordenador (llamado ***make-code***) que genera una **interrupción** para que la **rutina**

de tratamiento lea el código enviado. Es el *driver* de teclado el que en esta **interrupción** se encarga de combinar los códigos para generar los caracteres (mayúsculas, minúsculas, teclas de función, control y cualquier combinación entre éstas). Cuando se suelta una tecla, se envía el código de la misma (llamado *break-code*) que equivale al *scan-code* + 128 y de nuevo se ejecuta la rutina de tratamiento de la **interrupción** del teclado.

8.4 RATONES

Los ratones son los dispositivos de entrada posicionales que más se utilizan. Otros dispositivos son el *joystick* y los *trackballs*. El ratón viene asociado normalmente con un cursor sobre la pantalla que sigue su movimiento. Se suelen conectar al puerto serie del ordenador, aunque existen versiones con sus propias tarjetas adaptadoras que se insertan en el bus de expansión. Estos últimos reciben el nombre de (*bus mice*).

Los ratones funcionan asociados con las **interfaces** de usuario gráficas, en las que la mayoría de las acciones vienen determinadas por la pulsación de un botón del ratón cuando **éste** se encuentra sobre una determinada área en la pantalla. Los botones diferencian eventos cuando se aprietan o cuando se sueltan. El *driver* de ratón es el programa que se encarga de convertir las señales recibidas en órdenes de la CPU.

Al mover el ratón, la distancia recorrida se envía en forma de coordenadas X-Y al ordenador. El ratón envía información posicional relativa que se utiliza para mover un cursor sobre la pantalla. Será este cursor el que lleve cuenta de la posición absoluta. El Sistema Operativo actualizará la posición del cursor sobre la pantalla en función del desplazamiento recorrido por el ratón. La comunicación entre el ratón y el ordenador consiste típicamente en mensajes de la siguiente forma:

- botón n apretado
 - a botón n soltado
- ratón desplazado en (X,Y) unidades

Los ratones se diferencian básicamente en el método empleado para detectar el movimiento, existiendo principalmente los sistemas mecánicos y los ópticos. El mecánico emplea una bola de goma que está en contacto con dos **pequeños** rodillos. El movimiento de la bola es transmitido a los rodillos que miden separadamente la **translación** en las direcciones X e Y (relativas al ratón). Al final de cada **rodillo** se encuentra un disco con **pequeños** agujeros realizados siguiendo un patrón **regular**. La rotación del disco se utiliza para medir el desplazamiento relativo del **ratón**. Ya que los rodillos **se** colocan en posición perpendicular, (constituyendo un sistema de coordenadas cartesiano), cualquier movimiento oblicuo del ratón se convierte en una combinación de desplazamiento vertical y horizontal. Los valores obtenidos del desplazamiento se transmiten **vía** serie al ordenador, por medio de un cable o más recientemente mediante señales **infrarrojas**. Existen dos métodos para medir el desplazamiento:

1. **Electromecánico:** Los discos tienen contactos configurados en dos **círculos**, existiendo un **pequeño** desplazamiento entre los contactos de un disco y otro. El número de contactos delimita la resolución del ratón, siendo 40 un número bastante común. El **ratón** detecta el movimiento usando una banda de metal para cada **círculo** de contactos y otra banda que está en contacto con los dos **círculos** a la vez. La banda central produce un voltaje que se detecta en las **otras** dos cuando se alcanza un contacto. Se graba una unidad de movimiento cada vez que las dos tiras detectan un voltaje. Como ambas **pestañas** están **desplazadas** ligeramente, existe un **pequeño** retraso en la recepción del pulso sobre las pestañas y el orden de los dos pulsos indica la dirección **del** movimiento.
2. **Optomecánico:** Como se puede **observar** en la figura 8.8, los discos tienen un conjunto de agujeros **distribuidos** a lo largo de todo el perímetro. En una parte del disco se coloca un par de **leds** que apuntan hacia el otro lado del disco donde se encuentran sus respectivos fotodetectores. La rotación del disco hace que el haz de luz emitido desde el **led** hacia el **fotodetector** sea interrumpido cada vez que se termina un agujero, con lo que se genera un tren de impulsos **eléctricos** en el fotodetector. Cada pulso sirve para medir unidades de movimiento y como se dispone de dos parejas de led-fotodetector cuya posición relativa con respecto a los agujeros está desplazada ligeramente se puede determinar la dirección del movimiento controlando cuál es el flanco que se detecta primero.

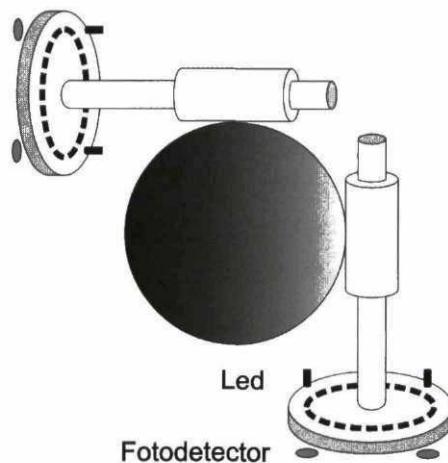


Figura 8.8: Esquema de un ratón optomecánico

Los ratones ópticos no utilizan la bola para **detectar** el movimiento. En la base del **ratón** existen dos ventanas por las cuales un par de **leds** alumbran la **superficie** sobre la que se mueve el ratón. La luz se **refleja** en esta superficie y vuelve a entrar por las mismas ventanas, siendo detectada por dos **fotodetectores**. Un **led** produce luz roja **normal** visible y el otro luz **infrarroja**. La almohadilla debe ser especial conteniendo líneas negras y azules pintadas de forma alternada tanto en la dirección X como en la Y.

La luz roja es absorbida tanto por las líneas negras como por las **azules**, mientras que la luz **infrarroja sólo** es absorbida por las líneas negras. Por lo tanto, el **fotodetector** de luz roja generará **pulsos** cada vez que el **ratón** se mueva por encima de alguna línea, **sea** cual sea su color, mientras que el **fotodetector** de **infrarrojos** generará pulsos únicamente al pasar por encima de las **líneas negras**. Para detectar la **dirección** del movimiento las ventanas tienen un **pequeño** desplazamiento tanto horizontal como vertical, con lo que se consigue una **pequeña variación** en los pulsos generados por los **sensores** que produce una **señal cuadrangular**.

Estos ratones no tienen partes móviles, por lo tanto son **más** fiables, pero necesitan una almohadilla especial. Los **mecánicos** tienen la desventaja de que la bola de goma y sobre todo los rodillos que la sujetan se suelen ensuciar, con lo que no se transmite bien el movimiento.

El **conector** del ratón sirve de manera adicional **para** alimentar al ratón. Cuando se mueve el **ratón** o se aprieta algún botón, el ratón transmite un paquete de datos a la **interfaz que produce** una **interrupción**. La **rutina** de tratamiento de la interrupción (es decir el driver del **ratón**) se encarga de leer el paquete de datos y de actualizar el estado de los pulsadores y de la posición del ratón. Además se debe encargar también de mover el puntero del **ratón** sobre la pantalla. Para ello se borra el puntero de la posición **actual**, se escriben los contenidos anteriores de esa posición, se leen los contenidos de la nueva posición y se dibuja el **puntero del ratón**.

8.5 DISPLAYS GRÁFICOS INTERACTIVOS

Los displays **gráficos** interactivos permiten al usuario retomar información sobre la posición en la pantalla de los objetos seleccionados, los objetos movidos o borrados. Esta información se puede introducir de muy diversas maneras.

8.5.1 Lápiz óptico

Estos dispositivos consisten en un elemento punzante que se conecta por medio de un cable al ordenador. Se utilizan para seleccionar objetos en la pantalla o para realizar **algún** trazo. En un extremo incluyen un botón para realizar la selección del objeto. Los **lápices ópticos** han sido poco a poco **reemplazados** por los ratones. Antiguamente se utilizaban por su facilidad de manejo, pero son dispositivos bastante complejos y con un **ámbito de aplicación** **también** reducido.

Se conectan al controlador del tubo de rayos **catódico** por medio de un cable. El lápiz tiene un fotodetector en la punta que detecta la luz emitida desde el píxel donde se posiciona sobre la **pantalla**. Al pulsar un botón, como **está** conectado al circuito de **barrido** de la pantalla, es posible determinar las coordenadas de la pantalla sobre las cuales está apoyado el **lápiz** cuando se detecta el refresco del píxel.

El problema se genera cuando el píxel sobre el cual se coloca el **lápiz** no está iluminado. El haz de electrones cuando pasa sobre un punto no iluminado

está **deshabilitado**, por lo que el fotodetector del lápiz es incapaz de detectar su presencia. Para evitar este caso se pueden tomar varias alternativas:

- Utilizar siempre el lápiz para seleccionar un objeto encendido, a modo de **menús** distribuidos sobre la pantalla.
- Usar el **lápiz** para controlar un **cursor** que le sigue como si fuera un ratón. Tiene el inconveniente de que no se puede separar el **lápiz** de la pantalla.
- iluminar la pantalla cada vez que se pulsa el botón del lápiz para evitar los puntos no iluminados.

8.5.2 Pantallas sensibles al tacto

Debido a su facilidad de uso cada vez más las podemos encontrar en aplicaciones domésticas o de uso general. El ejemplo más claro es el de los cajeros automáticos de los bancos. A simple vista son monitores de ordenador comunes **pero** que permiten **realizar** una selección de los objetos que se ven por la pantalla simplemente tocándolos con el dedo.

Cuando se toca la pantalla se puede detectar las coordenadas X, Y donde se ha realizado la pulsación. Frente a su facilidad de uso tienen la desventaja de que su resolución es muy baja, con lo que sólo **sirve** para detectar unos pocos objetos representados en la **pantalla**.

Existen dos tipos distintos:

- El primero usa una hoja transparente que se pega sobre la pantalla. **Las** hojas pueden ser de tipo resistivo o capacitivo. Las de tipo resistivo usan dos hojas, la primera con hilos conductores trazados horizontalmente y la segunda con los hilos trazados verticalmente. Si se presiona un punto de la pantalla los hilos hacen contacto suministrando la información de donde se ha tocado con el dedo. Las de tipo capacitivo utilizan una tecnología similar.
- El segundo tipo **utiliza** una **fila de sensores** de **infrarrojos** o de **ultrasonidos** que se montan en un extremo de la pantalla y una fila de receptores en el

otro lado. Los transmisores envían continuamente una **señal** a los receptores, mientras que el usuario interrumpe los haces de rayos produciendo la detección de la posición **pulsada**.

Para evitar la activación accidental de la pantalla, se incorpora una pequeña membrana sobre la pantalla que detecta cuándo se presiona ésta.

8.5.3 Tabletas digitalizadoras

Estas tabletas están formadas por un dispositivo apuntador móvil más un panel sensible de hasta 1 m^2 de tamaño. La resolución puede llegar hasta milésimas de milímetro. El dispositivo apuntador suele ser una mirilla o lápiz electrónico similar al **lápiz** óptico visto anteriormente que tienen cuatro botones que sirven entre otras funciones para recoger las coordenadas de la posición **actual**.

Para determinar la posición del puntero se pueden diferenciar tres métodos:

- Electromagnético: Debajo de la tableta existe una **matriz** de hilos muy densa. El **procesador** de la tableta envía pulsos eléctricos a través de la rejilla, recomendando todas las columnas para cada una de las filas. Cuando coinciden los pulsos de las **X's** y de las **Y's** se induce un voltaje en el puntero que sirve para determinar su **posición** en **función** del instante de tiempo en que se detectó. El espacio de la rejilla puede variar entre 0'25 y 10 milímetros.
- Por ultrasonidos: El puntero emite ultrasonidos que son recogidos por **micrófonos**. Un ejemplo de este tipo de tableta se puede observar en la figura 8.9
- Por presión: Existe una rejilla igual que en los teclados de membrana

Estos dispositivos tienen una resolución mucho mayor que la de los ratones, por lo que se usan en aplicaciones de CAD. Para la mayoría de estos programas, la tableta viene dividida en varias partes: una central para posicionar el **cursor**, y una **sección** en los bordes con diferentes símbolos. La pulsación del botón en la

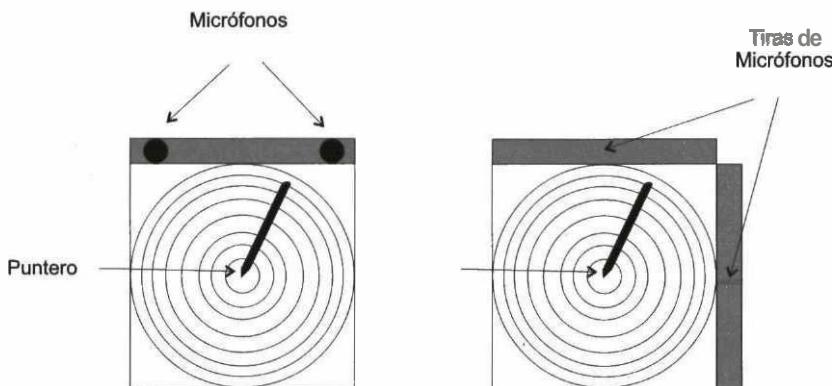


Figura 8.9: Tableta digitalizadora con detección por ultrasonidos

zona central posiciona un punto, mientras que en la sección **periférica** produce la activación de la función seleccionada.

8.6 DIGITALIZADORES DE IMÁGENES (ESCÁNERS)

Un digitalizador, en su nivel más básico, es un dispositivo más de entrada. La diferencia estriba en que toma sus **entradas** de forma **gráfica**. La imagen es **digitalizada** y convertida a un mapa de **bites** que se almacena para su posterior procesamiento. Los digitalizadores son útiles en un conjunto muy extenso de aplicaciones y es por ello por lo que se han diversificado para cumplir mejor las exigencias de cada una de **ellas**. Se pueden utilizar, por ejemplo, como **reconocedores ópticos de caracteres (OCR)**. Esta aplicación tienen como objetivo interpretar una imagen **digitalizada** de un texto y traducirla a código **ASCII**. A continuación se detallan los digitalizadores más comunes:

- m Digitalizadores a base de tubos de rayos **catódicos (CRT)**: Se utiliza un CRT para hacer incidir un haz de **luz** sobre la **superficie** a **digitalizar**. Un fotodetector mide la intensidad de luz reflejada y la convierte en tecnología **digital** para **almacenarla** en un fichero. Este tipo ha quedado ya obsoleto.
- m **Digitalizadores de impresora**: Se coloca un pequeño detector en el cabezal de impresión de una **impresora raster** capaz de imprimir en modo **gráfico**. La impresora se programa para que imprima una imagen en blanco so-

bre la zona que se quiere digitalizar y mientras se va imprimiendo, se va leyendo la **información**. Este escáner es de muy bajo coste.

- Digitalizadores de tambor: Son capaces de digitalizar tanto transparencias como **material** reflectivo. Pueden trabajar con tamaños desde los 35 mm hasta paneles de 16 pies por 20 pulgadas con resoluciones de más de **400 ppp (puntos por pulgada)**.
- Digitalizadores de documentos compactos: Se **diseñan** exclusivamente para aplicaciones de reconocimiento automático de caracteres (OCR) y de manejo de documentos.
- Digitalizadores para **fotografías**: Funcionan desplazando una fotografía sobre una fuente fija de luz.
- Digitalizadores de transparencias: Sirven para digitalizar elementos transparentes o translúcidos y su funcionamiento se basa en pasar luz a través de la transparencia en vez de reflejarla
- Digitalizadores de mano: Son portátiles. El escáner de mano que es capaz de leer una pequeña fila del documento en cada instante. Para recoger una página completa se pasa manualmente el escáner sobre toda su **superficie** y después se reagrupan las imágenes en un **único** fichero.
- **Digitalizadores** de tableta: Son los **más** utilizados por su versatilidad. Capturan **fotografías** y dibujos en color. **páginas** de libros y revistas e incluso son capaces de capturar películas fotográficas transparentes. Miden por tanto la intensidad y el color de una serie de puntos recogidos en un **rectángulo**.

8.6.1 Funcionamiento

El escáner convierte la luz en datos **digitales**. Todos **utilizan** el mismo principio de **reflexión** o de transmisión de la luz. La imagen se coloca debajo del cabezal lector, que consiste en una fuente de luz y en un **sensor** que mide la cantidad de luz reflejada o transmitida. Después la información **analógica** se convierte en **digital** utilizando un conversor **Analógico/Digital (conversor A/D)**.

El sensor de luz suele ser un dispositivo CCD (**Charge Coupled Device**) que convierte la intensidad de luz recibida en un voltaje proporcional. El cabezal monta una tira con miles de estos elementos. El escáner emite luz sobre tres filtros (rojo, verde y azul) y la luz reflejada en el documento a digitalizar es dirigida hacia la tira de sensores mediante un sistema de espejos y lentes. El CCD actúa como un fotómetro que produce un voltaje que posteriormente se convierte en información digital.

Actualmente se utiliza una nueva tecnología, llamada CIS, que utiliza bancos de **leds** de color rojo, verde y azul que producen luz y sustituye los espejos y las lentes por una fila de sensores ubicados en una zona muy cercana a la imagen. Como resultado se puede construir un digitalizador mucho más delgado y ligero, con un menor consumo y más barato, aunque de momento, la calidad que muestran no es tan buena como la de los **digitalizadores** convencionales.

8.6.2 Calidad del digitalizador

Los parámetros que determinan la calidad de un digitalizador son los siguientes:

- La resolución: Mide el grado de detalle que un digitalizador es capaz de distinguir. Se mide en puntos por pulgada y se necesita un dispositivo CCD para cada punto. Si el escáner tiene una resolución de 600 ppp y es capaz de capturar documentos de 8'5 pulgadas de anchura, entonces el cabezal necesita disponer de 5100 CCD. Este cabezal se monta sobre un soporte móvil que se desplaza a lo largo del documento en intervalos muy pequeños. El número de **CCD's** en el cabezal **determina** la tasa de muestreo en la **dirección** horizontal, mientras que el número de intervalos que se realizan en una pulgada **determina la tasa** de muestreo vertical. Habitualmente estos parámetros sirven para tener una referencia sobre la resolución del escáner, sin embargo la resolución verdadera la marca la capacidad del escáner para detectar los detalles de los objetos, y viene dada por la calidad de los elementos electrónicos, las lentes, los filtros, el control del motor paso a paso, además de la tasa de muestreo. Los **digitalizadores** actuales utilizan una técnica de **interpolación** para conseguir una mayor resolución, que emplea métodos tanto **hardware** como **software** para adivinar valores intermedios (realizando la media de los colores

que rodean al punto) e insertarlos entre los verdaderos. De esta forma se consiguen resoluciones de hasta **9600 ppp**, siendo sin embargo la resolución verdadera como máximo de **600 x 1200 ppp**.

- **El color:** Los digitalizadores en color utilizan tres fuentes de luz para cada uno de los colores primarios. Otros contienen un único tubo fluorescente y **tres** CCDs por **píxel** que filtran la luz. Los primeros necesitan dar tres pasadas al documento, mientras que los segundos, que son los más utilizados, digitalizan la imagen en una sola pasada. Los digitalizadores de una pasada utilizan dos **métodos** diferentes para leer la imagen. El primero pasa la luz a **través** de un prisma para separar los tres colores primarios, que son entonces **leídos** por tres CCDs distintos. La otra técnica, más barata, consiste en la utilización de **tres** CCDs, **recubiertos** con filtros para que únicamente puedan leer un color concreto. Esta **técnica** da casi los mismos resultados que la anterior.
- **Profundidad de bit (Bit-depth):** Esta **característica** indica la cantidad de información que se puede recoger por punto **digitalizado**. Habitualmente se almacena un byte de información por cada uno de los colores primarios, denominándose esta técnica “**digitalización con color verdadero**”. Actualmente existen digitalizadores de **30** y **36 bits**.
- **Rango dinámico:** Es similar al anterior, e indica los rangos de tonos que puede almacenar el escáner. Depende de la calidad de los convertidores A/D, de la pureza de la luz con la que se ilumina el papel, de la calidad de los **filtros** y de cualquier otro ruido eléctrico que pueda entorpecer el funcionamiento del digitalizador. Se mide en la escala de **0.0** (blanco perfecto) a **4.0** (negro perfecto). Dependiendo de la calidad del escáner se le asigna un valor que indica el porcentaje de ese rango que se puede distinguir. Los **escáneres** habituales tienen **dificultades** para distinguir los colores cuando está todo muy oscuro o demasiado iluminado, por lo que se les asigna un rango dinámico de **2.4**. Los **escáneres** profesionales obtienen rangos desde **2.8** a **3.2**, mientras que los de **tambor llegan** hasta los 3.8. Teóricamente, un escáner de 24 bits tiene un **rango** de 8-bits **para** cada uno de los colores primarios, pero hay que tener en cuenta que los bits menos significativos de cada uno de los colores se desprecian para evitar el ruido, por lo que se pierde calidad.

8.7 CONCLUSIONES

A partir de este capítulo comienza una **serie** donde se recoge la descripción de los dispositivos **periféricos** básicos utilizados en los sistemas informáticos.

En este capítulo se ha dado un repaso a los dispositivos de entrada de datos más importantes, comenzando por el más simple que es el interruptor. Se han contemplado los diferentes tipos de interruptores que existen y se han comparado entre si. Además se han presentado y solventado los problemas derivados de la utilización de los interruptores en los sistemas basados en microprocesador.

A continuación se han descrito los métodos que habitualmente se utilizan para la detección de las pulsaciones de un teclado. Se han analizado los problemas que presentan en cuanto a la detección de pulsaciones incorrectas y se han propuesto métodos para solventarlos. Dentro de este punto se insiste en la importancia que tiene el uso del procesador en las prestaciones del sistema y se analiza otro método de lectura que utiliza interrupciones para evitar que el procesador esté ocupado en todo momento. Para terminar con los interruptores, se han analizado los teclados de los ordenadores personales, se han presentado sus características y los enlaces de interconexión con el procesador.

También se describen otros dispositivos de entrada de datos, como son los ratones, el lápiz óptico, las pantallas sensibles al tacto, las tabletas **digitalizadoras** y los digitalizadores de imágenes. Los principales objetivos que se persiguen al describir un periférico consisten por un lado en que se comprenda cómo se detecta la actuación del periférico (sentido del movimiento, dirección, posicionamiento, pulsación, captura de información) y además que se puedan analizar las características que más influyen en las prestaciones de los mismos.

Por otro lado, con los avances tecnológicos, y con el auge de los sistemas multimedia y de realidad **virtual**, han salido al mercado multitud de dispositivos de entrada de altas prestaciones que no han podido ser incluidos en el presente texto. Como ejemplo se pueden citar micrófonos, guantes con sensores de presión, sensores de movimiento, ratones **tridimensionales** y pistolas inalámbricas.

8.8 CUESTIONES

8.1 *¿Qué es un escáner de teclado? Dibujar un teclado de 3 x 3 teclas, incluyendo los puertos y los **diodos** para evitar la vuelta atrás. Poner un ejemplo de pulsación de teclas donde se vea claramente la necesidad de utilizar estos diodos.*

8.2 *Explicar las funciones de los controladores de teclado.*

8.3 *Explicar el método empleado por el ratón para enviar la información al procesador, indicando además el tipo de información que se envía.*

8.4 *Diseñar mediante **biestables** D con entradas **asíncronas** de puesta a 0 y puesta a 1 un contador en **cuadratura** para poder ser utilizado en un ratón.*

8.5 *¿En qué consiste el problema de los rebotes en los dispositivos de entrada de datos? Indicar cómo afecta este problema a los teclados y describir alguna solución.*

8.6 *¿Cómo se averigua la dirección y el sentido del movimiento en un ratón optomecánico?*

8.7 *¿Para qué sirve el rollover en los scanners de teclado?*

8.8 *Describir los elementos que tiene un ratón óptico e indica para qué sirven.*

8.9 *Explicar las diferencias existentes entre un ratón y una tableta **digitalizadora** atendiendo a la información que recibe el procesador de dónde se encuentra el **cursor** en la pantalla y a la resolución que presentan.*

8.9 BIBLIOGRAFÍA

- **Computer Peripherals.** *Barry M. Cook, Neil H. White.* 3^a ed. Edward Arnold. ISBN 0-340-60658-4. 1995.
- **An Introduction to Microcomputer Systems. Architecture & Interfacing.** *John Fulcher.* Addison Wesley Publishers Ltd. ISBN 0-201-41623-9. 1989.
- **The indispensable PC Hardware Book.** *Hans-Peter Messmer.* 3^a ed. Addison Wesley Longman, Inc. 1997. ISBN 0-201-40399-4.
- **Conceptos Actuales Sobre la Tecnología de los Ordenadores.** *Joseph C. Giarratano.* Ediciones Díaz de Santos, S.A. ISBN 84-86251-15-X. 1984.

CAPÍTULO 9

DISPOSITIVOS DE COPIA IMPRESA

9.1 INTRODUCCIÓN

Sin duda, uno de los dispositivos que resulta imprescindible en todo sistema **informático** es la impresora. Cualquier sistema consta al menos de uno de estos dispositivos, desde los pequeños ordenadores personales que se instalan en casa, hasta los grandes supercomputadores. Debido a que cada día se va ampliando tanto el abanico de aplicaciones que ejecutan los ordenadores como su **expansión** por **entornos** no especializados, las impresoras resultan de vital importancia para poder escribir sobre el papel documentos de texto, fotografías, planos, esquemas de circuitos eléctricos e incluso **nóminas** y cheques. En este capítulo se describen los tipos de impresoras que se han venido utilizando hasta ahora, empezando por las antiguas impresoras **matriciales** y terminando por las sofisticadas impresoras láser en color.

La impresoras inicialmente utilizaban las mismas técnicas que se aplicaban a los teletipos. Un trozo de metal con la forma del carácter que se deseaba imprimir presionaba una cinta impregnada de tinta sobre la **superficie** de un papel para pasar la tinta al mismo. Estos caracteres se recogían en tambores o esferas.

Los desarrollos en esta tecnología dieron como resultado la impresora de margarita, de cuyo funcionamiento se muestra un esquema en la figura 9.1. Cada carácter se sitúa al final de un brazo flexible para formar un juego de caracteres

sobre un tambor. El tambor rota sobre **sí** mismo y un martillo golpea el carácter **específico**, que ya **está** situado en la posición central, el tiempo justo para que entre en contacto con la cinta. El mecanismo que opera sobre el tambor se coloca sobre un carro que **se** desplaza de izquierda a derecha por encima de la línea de texto. Estas **impresoras** están limitadas a la impresión de documentos de texto, con un único juego o repertorio de **caracteres**, denominado **fuente**.

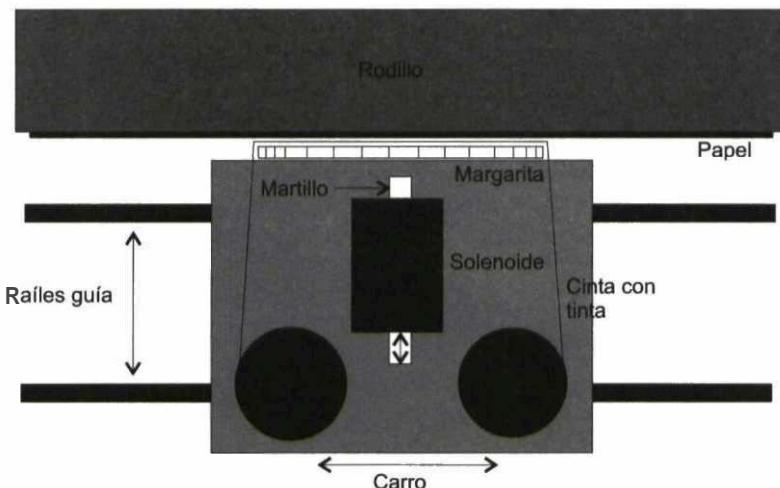


Figura 9.1: Impresora de margarita

El siguiente paso en el **desarrollo** de las impresoras tuvo lugar con la aparición de las impresoras de propósito general que podían producir tanto **gráficos** como caracteres. Los nuevos dispositivos permitían el control individual de un número relativamente grande de puntos situados **regularmente**. Esta flexibilidad permitía formar las **líneas** y **figuras geométricas** de los dibujos, así como definir de forma arbitraria el contorno de los caracteres para **formar** las fuentes. A estos dispositivos se les conoce generalmente como dispositivos de mapas de bit, ya que es posible controlar individualmente el estado de los puntos del papel a imprimir.

Para describir la información sobre una página completa **se define** un **patrón** de **bits** que representa el estado de cada uno de los puntos del dibujo. Hay que tener en cuenta que para que una impresora pueda representar los dibujos con un buen detalle se necesita del orden de 4 **Mbytes** de memoria para cada **página** completa que utilice **sólo** dos niveles de color (**on/off**). Un dispositivo en color de alta calidad necesita cerca de los 100 **MB**. Es por **ello** por lo que la necesidad

de memoria de las impresoras es bastante grande y **además** aumenta a medida que **se** requiere representar los dibujos con una mayor resolución.

Sobre los años ochenta, las **impresoras** que predominaban en el mercado **eran** las de matriz de puntos y las impresoras láser, sin embargo, a partir de los noventa empezaron a predominar las **impresoras** de inyección de tinta.

9.2 CLASIFICACIÓN DE LAS IMPRESORAS

Debido a la gran diversidad de **impresoras** que existen en la actualidad, resulta difícil realizar una clasificación completa. Entre los diferentes criterios que se pueden aplicar resulta interesante **realizar** una comparación atendiendo a **los** elementos comunes que poseen, que son los siguientes:

1. El mecanismo de impresión, que indica el elemento que se encarga de transferir la tinta al papel.
2. La parte de tracción del papel, encargada de desplazar el papel línea a línea para que el cabezal pueda imprimir a lo largo de todo el folio.
3. La **electrónica de control** utilizada para mover el papel, mover el cabezal e imprimir los **caracteres** o los puntos de tinta

Si se tienen en consideración el mecanismo de impresión, dependiendo de si el cabezal entra en contacto con el papel a la hora de realizar la **impresión** de una **página**, se pueden distinguir

- **Impresoras** de impacto. El mecanismo de impresión presiona una cinta de papel impregnada en tinta. Dentro de este tipo de impresoras antiguamente se **podía** distinguir entre:
 1. impresoras de líneas. Todos los caracteres iguales que existen en una misma línea se imprimen a la vez. En la impresora de tambor, filas enteras del mismo carácter aparecen sobre la superficie del tambor, con el juego completo de caracteres distribuido sobre toda la circunferencia. El tambor **se** rota para todos los caracteres y los martillos **se** activan cuando ese **carácter** en particular aparece sobre la **línea**.

De esta manera las páginas se pueden imprimir con una mayor velocidad que supera las 1000 **líneas** de 132 caracteres por minuto, que es una velocidad mucho mayor que los 700 **caracteres/seg** de las impresoras de caracteres.

2. Impresoras de caracteres. Imprimen un carácter tras otro de **forma** secuencial.
- **Impresoras de no impacto.** No existe contacto entre la cabeza de **impresión** y el papel; en las de inyección **se** emiten **pequeñas** burbujas de tinta cargadas **electrostáticamente** que **impactan** sobre el papel; las impresoras térmicas usan un papel especial que se quema aplicando un cierto voltaje sobre la **superficie** para producir los **gráficos**; las impresoras **electrosensibles/electrostáticas** producen una carga **electrostática** sobre la **superficie** del papel sobre la cual se adhiere el **tóner** cuando el papel pasa sobre **él**. Estas impresoras evitan el desgaste mecánico de los cabezales y además son silenciosas, pero normalmente requieren tratar el papel de alguna **forma** especial.

Si se atiende a la posibilidad de generar **gráficos** de alta **resolución** y de poder elegir entre multitud de fuentes diferentes se pueden distinguir las siguientes clases de **impresoras**:

- **Impresoras de texto conformado.** La forma de los caracteres está moldeada en los **cabezales**, como la impresora de margarita, donde el carácter deseado se posiciona delante de un martillo haciendo rotar la margarita. El martillo entonces golpea el carácter sobre la cinta de tinta que impregna el papel. Las ventajas son que produce caracteres de calidad **como** en una máquina de escribir y que reemplazando la margarita se puede cambiar el juego de caracteres.
- **Impresoras de matriz de puntos.** Forman los **caracteres** y **gráficos** gradualmente, **descomponiéndolos** en patrones de puntos muy pequeños.

9.3 IMPRESORAS DE MATRIZ DE PUNTOS

Estas impresoras van formando los caracteres de forma gradual por columnas de puntos. En el cabezal existen 9 guías distribuidas circularmente que

mantienen agujas **en su interior** en una posición lo **más** perpendicular posible al papel. Existe también un solenoide por cada **guía** que al **recibir** un cierto voltaje empuja la aguja en dirección **al** papel, haciendo que ésta presione la cinta impregnada en tinta para producir un punto sobre la superficie del papel. En la **figura 9.2** se muestra una de las agujas del cabezal. **Las** agujas se organizan en una columna vertical. El muelle de contención evita el movimiento excesivo si **los** cabezales están demasiado cerca del papel o **si** el papel **es** demasiado gordo.

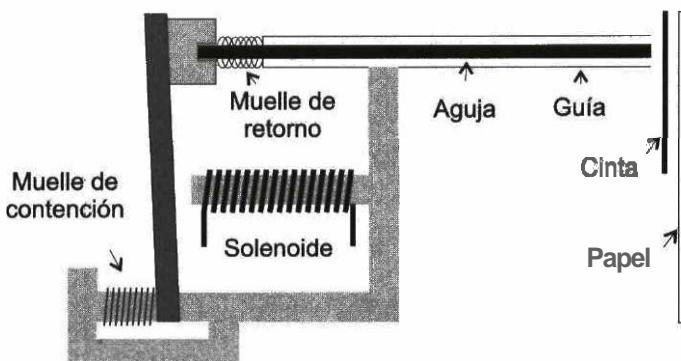


Figura 9.2: Detalle de una aguja del cabezal de una impresora matricial

De **esta forma** se van imprimiendo los caracteres columna a columna. El número de puntos usado para formar el carácter determina la **calidad** de la impresión. Las **impresoras** de alta calidad permiten escoger el espacio existente entre los puntos. **Realizando** varias pasadas sobre la misma línea con las agujas **en** diferentes posiciones **se** pueden formar caracteres con mayor resolución, incluso se pueden sobreponer para producir texto de calidad, con texto en negrita, itálica o subrayado. Estas **impresoras** pueden llegar a tener hasta 27 agujas.

Las **impresoras matriciales** tienen la ventaja sobre las que usan otras tecnologías de que el papel puede ser continuo y que además permiten la alimentación con hojas sueltas, utilizando **rodillos** o **ruedas** dentadas. Para la cinta con tinta es posible tener dos carretes o un cartucho que contiene una cinta continua. Sin embargo la velocidad es bastante lenta, oscilando entre los 50 y los 500 caracteres por segundo, dependiendo de la calidad de las letras que se quieran imprimir.

9.4 IMPRESORAS DE TRANSFERENCIA DIRECTA

Son impresoras **electrostáticas** que emplean papel **dieléctrico** sobre el que se generan zonas cargadas negativamente por medio de un conjunto de agujas dispuestas formando una tira. Después de cargar una **línea** en el papel se le espolvorea con **tóner** líquido que contiene partículas de carbón mezclado con **parafina** cargadas positivamente. Estas partículas se pegan en las áreas cargadas del papel y forman la imagen. El **tóner** restante se elimina y la tinta adherida se seca antes de que el papel pueda ser manipulado.

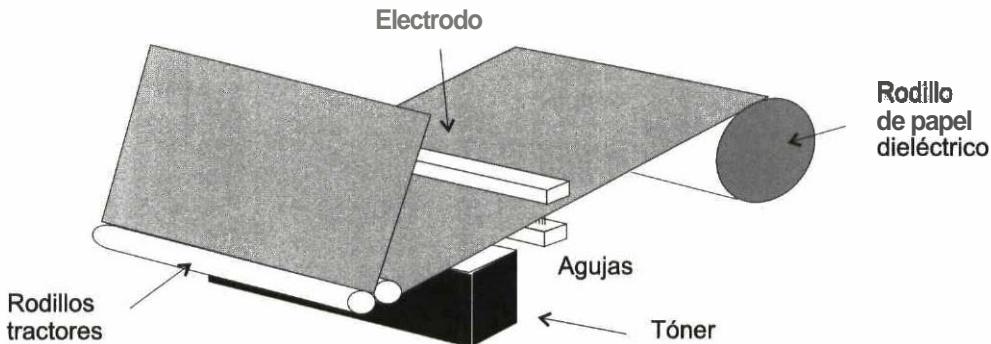


Figura 9.3: Impresora de transferencia directa

Existe un hueco de aire de **aproximadamente** 0'6 mm entre las agujas y el papel, mientras que en el otro lado un electrodo está en contacto continuo con el papel. Entre los electrodos se **aplica** un voltaje cuyo valor **oscila entre** los **550** y los **750** Voltios. Cuanto **mayor** sea este voltaje se **producirá** una imagen más oscura. La figura 9.3 muestra un esquema de una impresora de este tipo.

El espacio que **existe** entre las **agujas** es aproximadamente de 0'25 mm, por lo que la impresora tiene una **resolución** de **100** puntos por pulgada. Además es posible utilizar dos **filas** de agujas para doblar la **resolución**. También se puede **imprimir** en color, simplemente pasando el papel varias veces con **tóner**s de diversos colores.

Estas **impresoras** tienen un alto coste debido **principalmente** a la electrónica de alto voltaje que necesitan. Normalmente son reemplazadas por las **impresoras láser**, excepto cuando se necesita **utilizar** papel de **gran** anchura, puesto que se **pueden** utilizar tiras de agujas con una mayor **longitud**, que **permiten** **imprimir** mayor **número** de columnas.

9.5 IMPRESORAS LÁSER

Estas **impresoras** fueron pensadas originalmente para producir una **impresión** de alta calidad, a gran velocidad y con un **gran volumen** de producción. **Actualmente** se logran velocidades de **varios** cientos de páginas por minuto, se ha reducido su tamaño, se han abaratado los costes de producción y la resolución **mínima** suele ser de **600 puntos** por pulgada. Todos estos factores las han convertido en **impresoras** muy populares, siendo también muy utilizadas en aplicaciones domésticas.

Las impresoras **láser** fueron inventadas por **Hewlett-Packard** en 1984, utilizando una tecnología **desarrollada** por Canon **para** su fotocopiadoras. Se distinguen básicamente de éstas **en** que la fuente que se utiliza para iluminar el papel en la fotocopiadora es una luz brillante, mientras que en las impresoras la luz proviene de un láser, siendo el proceso de copia o de impresión del papel similar.

Las ventajas de **estas impresoras** frente a las de inyección de **tinta** son que producen mejor calidad en blanco y negro, que están pensadas para producciones altas y además permiten la utilización con facilidad de otros soportes como postales, sobres y cualquier otro tipo de material de **tamaño** no **estándar**.

9.5.1 Operación

En las impresoras láser, el componente más **crítico** es el tambor **fotosensible** o también conocido como cartucho orgánico fotoconductor (**OPC - Organic Photo-conducting Cartridge**). El tambor está cubierto por un material compuesto **por selenio** que es sensible a la luz. En la oscuridad tiene una alta capacidad y actúa como un condensador que se carga a un alto voltaje a medida que pasa por el hilo cargador. Un rayo láser se refleja sobre un espejo **poligonal** móvil que lo mueve sobre una línea del tambor. Ya que el tambor está **rotando** el haz lo cubre totalmente. La luz, al incidir sobre el tambor, reduce la capacidad del tambor en el punto del impacto, reduciéndose el efecto del condensador y descargándose, eliminando el alto voltaje. Modulando la intensidad del láser, se pueden descargar los puntos del tambor **selectivamente** o dejarlos cargados

de forma que se representa en el mismo la imagen de salida. Cada punto del tambor se corresponde con un punto en la hoja del papel. Tanto el tambor como el **tóner** tienen que reemplazarse periódicamente, **incrementándose** el coste del mantenimiento de la impresora.

El láser se enciende y se apaga a medida que va pasando **horizontalmente** sobre el tambor, cambiando de **estado** millones de veces cada segundo. El **tambor**, a su vez, rota de forma que se **construye** una línea horizontal cada vez. Cuanto menor sea el paso de **rotación**, mayor será la resolución vertical de la página, y cuántas más veces se apague y se encienda el láser en una línea, mayor **será** la resolución horizontal.

Al pasar el tambor sobre el **tóner**, cargado negativamente, las áreas que aún quedan cargadas atraen las partículas de tinta que se sitúan sobre su superficie. Las partes descargadas no atraen al **tóner** y por tanto **posteriormente** resultan en zonas blancas en el papel. A continuación una hoja de papel pasa entre el **tóner** y el hilo de la corona. El cable **produce una** descarga negativa sobre el papel que es suficiente para **desprender** el **tóner** del tambor, **transfiriéndose** de esta manera la imagen al papel. Un **rodillo calentado** a una temperatura aproximada de 260 °C funde las partículas del **tóner** sobre el papel para que, en **combinación** con la **presión** que realizan unos rodillos, queden sujetas **permanentemente** al papel. También es posible utilizar rodillos de gran **presión** sin aplicar calor para fijar la imagen en el papel.

El último paso consiste en limpiar el **tóner** remanente del tambor para iniciar de nuevo el proceso. Existen dos **formas** de hacerlo, la **física** y la eléctrica. En la primera el **tóner** se elimina de **forma mecánica** mediante la actuación de unos cepillos y se devuelve a su recipiente, en la segunda el cable de la corona carga el tambor para repeler y por tanto eliminar el **tóner**. En la figura 9.4 se muestra un detalle de los elementos más importantes que se encuentran en este tipo de impresoras.

En lugar de utilizar un rayo láser, existen impresoras que utilizan un conjunto de **diodos** emisores de luz (*leds*) dispuestos formando una tira para incidir sobre el tambor. Una impresora con una resolución de 300 puntos por pulgada (**ppp**) dispondrá de 300 *leds* por pulgada. A este tipo de **impresoras** se las conoce como **impresoras** de *leds*, que suelen ser más **económicas** que las **lásers**, pero cuya resolución horizontal sin embargo es **fija**. Las **impresoras** LCD trabajan de

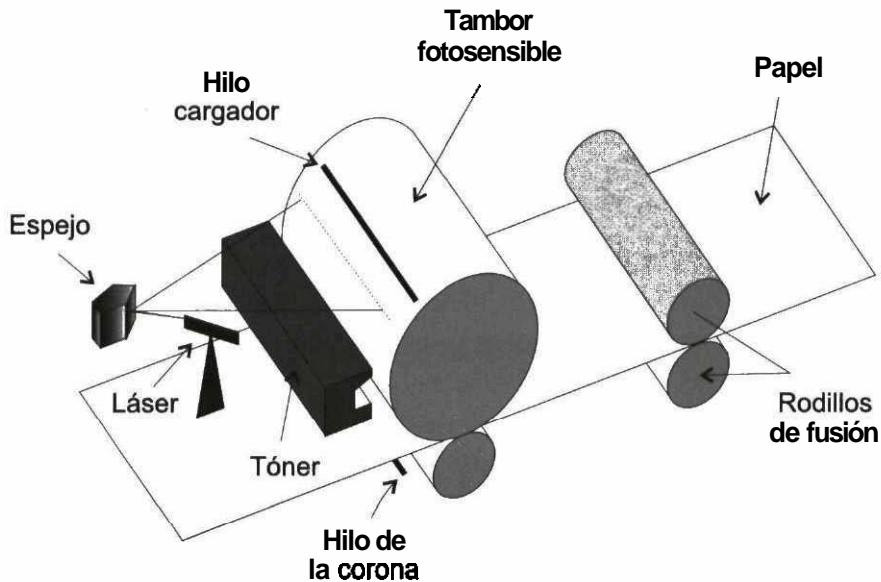


Figura 9.4: Mecanismo de impresión de una impresora láser

una forma similar, utilizando un panel de cristal líquido como la fuente de luz en vez de utilizar la **matriz de leds**.

9.5.2 Impresoras láser en color

Cualquier color se puede obtener con una mezcla de azul claro, morado, amarillo y negro (colores también llamados CMYK). Las impresoras **láser** en color realizan cuatro fases en el proceso electro-fotográfico. situando cada vez un **tóner** en la página o construyendo la imagen de cuatro colores en una superficie intermedia de transferencia. Las prestaciones no son tan buenas como las de las **impresoras** en blanco y negro, llegando a imprimir entre 3 y 5 páginas por minuto. Las nuevas impresoras tienden a procesar los cuatro **tóners simultáneamente** para ganar velocidad, para ello se dispone de 4 tambores y de 4 vectores de **leds**, uno para cada color. Los datos se envían simultáneamente a las cuatro cabezas y se van imprimiendo primero el color morado, luego el azul claro, después el amarillo y por último el negro. Otra de las grandes ventajas de estas impresoras es que el tóner se pega al papel, y no es absorbido por éste.

como ocurre en las **impresoras** de inyección de tinta, por lo que se puede imprimir en **multitud** de superficies sin que debido a la humedad se **arrugue** el papel. Se puede decir que éstas van a ser las **impresoras del futuro**, siendo tan usuales como las **fotocopiadoras**.

9.5.3 Lenguajes de descripción de páginas

Una impresora láser necesita tener toda la información sobre la página antes de empezar a imprimir. Para enviar las imágenes, el método **más sencillo**, pero menos práctico, es transferir la misma como un mapa de puntos. La impresora simplemente se dedicará a transferir estos puntos sobre el papel, por lo que no puede haber un procesado intermedio de la información para mejorar la imagen. Si se utiliza este método, drásticamente se reduce la portabilidad que tiene la **página** impresa entre diferentes impresoras. No es práctico describir el resultado deseado a nivel de bit, ya que diferentes impresoras trabajan con resoluciones diferentes, desde menos de **70 ppp** a más de **2000 ppp** y también con tamaños de páginas diferentes, por lo que el mismo mapa de bits no quedará representado de igual manera en el papel.

Sin embargo, si se conocen más datos sobre la imagen a imprimir, se puede incrementar la calidad de la misma. **Piénsese** que una imagen en un monitor tiene aproximadamente **760.000** puntos, mientras que en un folio a resolución de **600 ppp** tendrá más de **33 millones** de puntos. Es por tanto necesario proveer algún método para aumentar la calidad de las imágenes impresas.

Inicialmente se enviaba texto a la impresora junto con algunos caracteres de control para definir los tipos en negrita, subrayado, condensado o extendido. La fuentes las incorporaba la impresora y los gráficos se construían línea a línea de forma muy lenta. La transmisión de documentos era sencilla, pero sin embargo nunca se escribía la página exactamente igual a como se **veía** en la pantalla, ni todas las **impresoras** producían páginas iguales.

También existen un conjunto de **impresoras**, las llamadas "**impresoras windows**" que no tienen procesador, y que por tanto en el lugar de origen se debe crear ya el mapa de **bits**.

Hoy en día lo habitual es enviar la descripción de la página utilizando un **lenguaje especial** de programación de alto nivel, llamado lenguaje de descripción de páginas (PDL). El lenguaje define por medio de **vectores**, formas **geométricas**, fuentes y caracteres la información que **se** desea imprimir. El procesador interno de la impresora (**llamado RPI - Raster Image Processor**) se encarga de convertir esta descripción en una imagen de mapa de bits (**raster**), definida ya como un conjunto de **puntos/píxeles** en un formato de filas y columnas. Dependiendo de las **características** de la impresora se generará un mapa de bits u otro. La impresora puede imprimir las **líneas** y curvas tan finas como su resolución lo permita. La idea que se persigue es que una misma página pueda **enviarse** a dispositivos diferentes. Para los caracteres, aunque también son **líneas** y curvas, se utiliza un formato de fuente predeterminada, como los **formatos True-Type o Type-I**. Además de un emplazamiento preciso de los caracteres, **también** se pueden escalar y rotar, de forma que enviando poca **información** se pueden producir diferentes estilos y **tamaños**. A este proceso se le conoce en terminología inglesa con el nombre de **rendering**.

Uno de los lenguajes PDL más extendidos es el **Postscript** desarrollado por Adobe **Systems Incorporated**. Las impresoras que aceptan la impresión de documentos descritos a partir de este lenguaje tienen una memoria **ROM** donde se almacenan **los** diversos tipos de fuentes y todas las definiciones propias del **PostScript**. Los procesadores de textos y otros programas de diseño asistido por ordenador generan el **código** fuente **PostScript**. El Nivel 1 de **PostScript** permite que todas las impresoras escriban figuras en el mismo tamaño y posición **pero** con mayor o menor calidad dependiendo de la resolución de **la** impresora. El nivel 2 de PostScript permite utilizar colores y comprimir los datos **antes** de ser enviados a la impresora.

Otro lenguaje de descripción de páginas muy utilizado actualmente es el **PCL (Printer Control Language)**, diseñado por **Hewlett-Packard** para comunicar **PCs** con sus impresoras láser. Inicialmente se diseñó para las impresoras de **matriz** de puntos, pero con el PCL 5 ya se incluyeron **características análogas** al lenguaje PostScript, permitiendo descripciones vectoriales, el escalado de las fuentes y la compresión de datos. **Con** el PCL 5e se permite la comunicación **bidireccional** para que **la** impresora envíe información de estado. Actualmente se utiliza PCL 6 que supone un gran incremento en las prestaciones de impresión de **páginas**, ya que se utilizan algoritmos de compresión de datos que reducen la cantidad de información que se debe enviar a la impresora.

Otra alternativa que existe para enviar **páginas** supone usar GDI (*Graphical Device Interface*), donde el **procesador** en el origen realiza la **renderización** de la página, con lo que las impresoras son más simples pero se **incrementa** el esfuerzo que se realiza en el origen para enviar la página. Las **impresoras** que utilizan el sistema de impresión del Sistema Operativo **Windows** (*Windows Print System*) permiten la renderización de la imagen a medida que se va imprimiendo.

9.6 IMPRESORAS DE INYECCIÓN DE TINTA

A partir de los años noventa fue cuando empezaron a comercializarse las impresoras de inyección de tinta. Se empezó por la impresora tricolor que pronto fue desbancada por el modelo de cuatro colores. La ventaja que tienen sobre las **láser** es que son más económicas, ya que el mantenimiento de los cartuchos es mucho más caro y precisan de papel especial para lograr calidades aceptables.

En las impresoras basadas en un cristal piezo-eléctrico, la tinta pasa a través de un **pequeño** agujero para producir una comete de alta velocidad de pequeños puntos de tinta. En la figura 9.5 se muestra un diagrama de bloques de la **estructura** de una impresora de inyección de tinta. El tamaño y el espaciado de estos puntos es constante, y se logra haciendo vibrar el compartimento (como si fuera un altavoz) a una frecuencia ultrasónica mediante un cristal montado al final de la cavidad. La frecuencia de la vibración suele ser de 100 **kHz**, el diámetro del agujero de 0'06 mm y el espacio que existe entre dos puntos de tinta consecutivos de 0,15 mm.

A cada punto de tinta, después de abandonar la cavidad, se le aplica una descarga cuando pasa a través de un electrodo cargador localizado junto al expulsor. Los puntos son desviados verticalmente por un segundo electrodo y golpean el papel. La posición horizontal se controla moviendo todo el sistema de inyección. El grado de deflexión vertical lo determina la cantidad de carga del punto de tinta. Si no están cargados no existe deflexión y estos puntos son recogidos en un recipiente cercano al papel.

Las ventajas de este método son que se puede variar el grado de flexión del cristal para producir puntos de tinta más grandes o más pequeños. Como no es necesario calentar la tinta, ésta puede ser de mayor calidad y mejor absorbida por el papel.

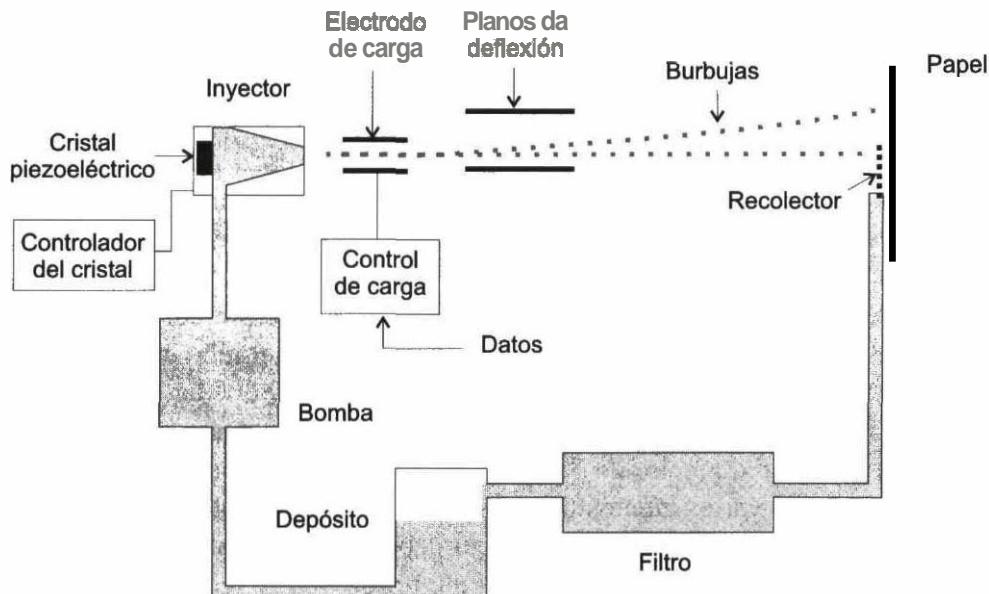


Figura 9.5: Impresora de inyección de tinta

Estos sistemas de flujo continuo están siendo **rápidamente** desplazados por los sistemas bajo demanda (DOD **Drop on Demand**). En ellos se utiliza un elemento **piezoeléctrico** que **sólo** expulsa puntos de tinta cuando es necesario y por lo tanto no existe la necesidad de utilizar el **recolector**. Si se **utiliza** una pila de expulsores en vez de uno solo, se puede producir una **línea** vertical sin la necesidad de los **deflectores**, **incrementando** la velocidad de impresión.

Las últimas impresoras de inyección de tinta tienen **cabezales** con **128** agujas para el negro y **64** para cada uno de los colores azul, morado y amarillo (**Cyan, Magenta, Yellow (CMY)**), obteniendo una resolución de **720 x 720 ppp**. Además si se hacen dos pasadas, como la precisión de los **cabezales** es excelente, se pueden lograr resoluciones de hasta **1440 x 720 ppp**.

Los caracteres se forman mediante matrices de puntos. **Para** una **impresión** de alta calidad se necesitan unos mil puntos por **carácter**. Si el **expulsor** genera **10^5** puntos por segundo (frecuencia & **100 kHz**), se puede imprimir a una velocidad de 100 caracteres por segundo.

9.7 IMPRESORAS DE BURBUJAS

Las impresoras de burbujas fueron **desarrolladas** por **Hewlett-Packard** y **Canon**. Su funcionamiento es muy similar al de las **impresoras** de inyección, se diferencia en que en lugar de controlar los puntos de tinta mediante un elemento **piezoelectrónico**, se utilizan medios térmicos. Un **pequeño** calentador en el **tubo** capilar de la cabeza de **impresión** vaporiza un poco de tinta produciendo **una** burbuja de gas. La presión que produce al expandirse dentro del **tubo** empuja un punto de tinta hacia el papel. El **vacío** creado dentro del tubo es llenado con tinta nueva del depósito. Existen **normalmente** entre 300 y 600 agujas, cada una con un diámetro de 70 micras (como el de un cabello humano), que expulsan gotas de tinta de un **diámetro** de 50 a 60 **micras**. Con esta **técnica** se logran velocidades de hasta varios cientos de caracteres por segundo. Los problemas de las impresoras de burbujas son que la tinta debe resistir el calor y que se debe esperar cierto tiempo a que se **enfrie** la tinta, por lo que se reduce la velocidad de impresión.

El cabezal **recorre** el papel horizontalmente y va formando la imagen imprimiendo tiras de la misma. Las impresoras actuales permiten resoluciones de hasta 1200 ppp y tienen una velocidad de 3 a 4 páginas por minuto (**ppm**) en color y de 4 a 8 **ppm** en blanco y negro, con lo que se acercan a las velocidades de las impresoras láser en color. Esto se logra proyectando la tinta a una frecuencia aproximada de 12 MHz e imprimiendo columnas de media pulgada.

9.8 EL COLOR EN LAS IMPRESORAS DE INYECCIÓN Y DE BURBUJAS

Las impresoras utilizan los colores **azul** claro, morado y **amarillo** para producir los colores. Para producir el blanco no se imprime ningún color. Existen impresoras de inyección que además incluyen un cartucho de tinta negra para imprimir en blanco y negro, pudiendo intercambiar este modo de funcionamiento con el de impresión en color.

La calidad de la impresión la determinan la resolución de la impresora y el **número** de niveles de color que se pueden imprimir por **píxel**. Para realizar **di-**

bujos es mejor tener muchos niveles de colores para poder obtener fotografías fidedignas, mientras que para aplicaciones **ofimáticas** prima la resolución. El modo más simple de imprimir en color es utilizar puntos con colores puros, sin ser posible la utilización de niveles intermedios. Si la impresora permite **unir** varios puntos de color, se imprimiría **únicamente** en morado, amarillo, rojo, **ver-**&, azul, negro y blanco, produciendo una calidad **insuficiente** para la mayoría de aplicaciones. La solución adoptada consiste en la utilización de un **tamaño** de punto variable, basado en una celda que contiene **varios** puntos elementales, de forma que combinando las proporciones de los puntos con los colores puros se Uega a producir sensación de color (*halftonning*).

Otra estrategia para producir color consiste en la incorporación de cabezales que **producen** 256 tonos diferentes por color, pudiendo por tanto imprimir hasta 16.7 **millones** de colores. Esta **técnica** se **llama** impresión de tono continuo. Sin embargo lo habitual son las impresoras que proporcionan únicamente varios tonos por punto (entre 4 y 16), proporcionando una paleta más completa de colores sólidos y tonos combinados **más** suaves. Estos dispositivos se llaman **impresoras contone**.

Recientemente se han introducido impresoras con seis colores (seis **cartu-**chos diferentes) **añadiendo** el color azul cielo **claro** y el morado **claro**, que producen más luminosidad en las hojas impresas. Las futuras tendencias apuestan por reducir el **tamaño** de las burbujas de tinta para poder proyectar **más** cantidad sobre un único punto en el papel, de forma que se pueda conseguir mayor gama de colores sólidos. Actualmente Hewlett-Packard utiliza impresoras que pueden proyectar hasta 29 puntos de tinta por punto (más de 3500 colores sólidos por punto).

La tinta que se utiliza **en** las impresoras a color debe secarse **rápidamente** para evitar que se mezclen unos colores con otros. Se utiliza una tinta especial que se seca en décimas de segundo, pero debido a que se basa en agua la humedad arruga el papel ligeramente, por lo que la tinta de las impresoras láser proporciona mejor calidad. Actualmente se investiga para poder utilizar **cualquier superficie** de impresión, sin embargo la calidad **fotográfica** únicamente se consigue con papel especial que no es muy económico.

9.9 TRAZADORES

Existe un tipo de impresoras especial apropiada para la realización de gráficos vectoriales, que utilizan arquitectos e ingenieros en sus aplicaciones de diseño asistido por computador (CAD) para cubrir sus necesidades de impresión. Este tipo de gráficos se caracterizan por estar formados por **líneas** continuas cuyo trazado puede ser realizado fácilmente con la utilización de plumillas o **rotuladores** de varios colores. Las impresoras que utilizan esta técnica se clasifican en función del **tamaño** del papel que soportan y de su capacidad para utilizar varias plumillas para producir colores. Estas impresoras reciben el nombre de **trazadores** o *plotters*.

Los trazadores poseen motores separados que mueven la plumilla de forma independiente sobre los dos ejes, incorporando también algún medio para levantar y bajar la plumilla. Para los dibujos de pequeño tamaño, la plumilla se desplaza a lo largo del papel, estando **éste** fijo, mientras que para los dibujos grandes, un rodillo de fricción mueve el papel hacia delante o hacia atrás de forma relativa a la posición del cabezal para cubrir toda la página.

En los trazadores en color, se pueden intercambiar las plumillas **automáticamente**, teniendo que realizar varias pasadas sobre la misma hoja para producir una copia en color.

La calidad, el tiempo de impresión y el tiempo necesario para cambiar las plumillas son parámetros que marcan la calidad del trazador. El tiempo de respuesta (velocidad y aceleración) desempeña un papel importante en el tiempo total del dibujo. La velocidad es crítica en el dibujo de líneas largas y en el relleno de polígonos de grandes dimensiones, mientras que la aceleración es más crítica en el dibujo de curvas y pequeñas **líneas** no conectadas.

Hoy en día se están sustituyendo los trazadores de plumilla por los de inyección de tinta, donde primero es necesario **transformar** el formato recibido, normalmente en un lenguaje específico de descripción de gráficos vectoriales como puede ser el **HP-GL** o el **HP-GL2**, a un mapa de **bits**. Tienen la ventaja de que pueden simular múltiples plumillas de múltiples colores y anchos, sin embargo la calidad no es tan buena. Son como si fueran impresoras de inyección pero con soporte para papel grande.

9.10 OTROS TIPOS DE IMPRESORAS

En esta sección se describen **impresoras** menos comunes, por su restringido **ámbito** de aplicación o simplemente porque resultan bastante caras y con menos prestaciones que las habituales.

9.10.1 Impresoras de tinta sólida

Este tipo de impresoras utiliza pequeños bastoncillos de tinta **sólida** que se calienta para ser repartida sobre un tambor que se encarga de **solidificarla** posteriormente sobre el papel. Son más económicas que las impresoras láser, **utilizan** cualquier tipo de **papel** y cuestan menos de mantener, sin embargo la calidad no es tan buena. La resolución **varía** desde los 300 ppp hasta un máximo de 850 por 450 ppp. La velocidad de impresión en color es del orden de 4 páginas por minuto.

9.10.2 Impresoras de sublimación de tinte (*Dye-sublimation*)

Estas impresoras se utilizan para aplicaciones **gráficas** y fotográficas que precisan de una alta calidad y resolución. La tinta se calienta hasta que se obtiene un gas que se expande **sobre** el punto de tinta. Dependiendo **de la** temperatura con la que se caliente se puede controlar la cantidad de tinta que se inyecta, permitiendo por lo tanto aplicar el color de una forma continua en lugar de por puntos como lo hacen **las impresoras** de inyección.

La tinta se almacena en folios del mismo tamaño del que se quiere imprimir y la imagen se realiza impregnando el folio en blanco cada vez con un color distinto (amarillo, azul claro, morado o negro), empezando **por** el **amarillo** y terminando por el negro. Estas **impresoras necesitan papel especial** para poder crear los colores de forma precisa y las velocidades de impresión son inferiores a una página por minuto.

Actualmente hay impresoras de inyección que **también** vaporizan la tinta, calentando los inyectores hasta los 500 °C. Estos dispositivos alcanzan una resolución de 1200 ppp.

9.10.3 Impresoras de color térmicas (*Thermo Autocrome*)

El proceso de impresión es más complejo que el de las impresoras de inyección o las láser. Se utiliza en las cámaras fotográficas digitales y utiliza un papel especial que contiene tres sustratos de pigmentos (azulclaro, **morado** y amarillo) sensibles a diferentes temperaturas. La impresora tiene cabezales térmicos y ultravioleta por los cuales el papel se pasa tres veces. En la primera pasada se calienta el papel con la temperatura adecuada para activar el pigmento amarillo y a continuación se fija utilizando el cabezal ultravioleta. Después se utiliza el mismo proceso para el color morado y el azul, con lo que resulta una impresión con colores bastante duraderos.

9.10.4 Impresoras de cera térmicas (*Thermal Wax*)

Se utilizan para imprimir transparencias. Utilizan rodillos con colores, del mismo tamaño que el papel, que contienen películas plásticas recubiertas de cera coloreada. La impresora **actúa** demitiendo los puntos de tinta (de forma binaria) sobre un papel térmico especial.

La resolución y la velocidad no son muy altas, típicamente alcanzan los 300 ppp y una velocidad de una página por minuto.

9.11 CONEXIÓN CON LAS IMPRESORAS

Normalmente el procesador puede transmitir los datos a una frecuencia que supera la velocidad con la que los pueden aceptar las impresoras. Para que no se pierdan los datos debe haber un método de detener la transmisión hasta que la impresora esté lista para seguir aceptando más datos. Este método se conoce como control de flujo. Además de los cables que envían los datos, debe haber una conexión en la dirección inversa que indique que la impresora está ocupada.

La mayoría de las impresoras escriben todos los caracteres que forman una línea de forma consecutiva teniendo que esperar por tanto a que se reciba totalmente la siguiente línea para poder imprimirla. En consecuencia es necesario almacenar de forma temporal los caracteres a medida que se van recibiendo en

el dispositivo. Cuando llega un carácter la impresora activa la línea de ocupado durante un corto espacio de tiempo hasta que almacena el dato en el **buffer**. Cuando ya se ha recibido la línea completa, la impresora está ocupada durante un tiempo mucho más largo hasta que la termina de imprimir.

Para ganar rapidez en la **recepción** de datos, la mayoría de las impresoras incluyen un **buffer** doble, el **primero** **recibe** los datos hasta que la línea está completa, y entonces la transfiere al segundo **buffer**. La información en este **buffer** sirve para controlar el mecanismo de impresión. Al mismo tiempo, el **primero** queda libre y dispuesto a recibir más datos. Si los datos no llegan demasiado rápidos, se **produce** una impresión continua. En la **práctica**, como la memoria es bastante **económica**, las **impresoras** incluyen **buffers** grandes **capaces** de almacenar una **página** completa.

9.12 CONCLUSIONES

Como se **habrá** podido **observar** con la lectura de este capítulo, los aspectos tecnológicos también **influyen** notablemente en el **desarrollo** de nuevos tipos de impresoras con mayores prestaciones. En un par de **décadas** se ha pasado de las ruidosas impresoras de **líneas** y de margarita alas potentes impresoras de **inyección** de tinta y a las impresoras **láser**. Resulta necesario advertir que cuando se analicen las prestaciones de los dispositivos objeto de este capítulo, no se deben considerar de forma absoluta los parámetros de resolución y velocidad que marcan la calidad de la impresora, ya que seguramente se **estarían** utilizando valores **obsoletos**. Es **más** interesante realizar una comparación entre los dispositivos, ya que las cantidades relativas se alteran **menormente** con los avances **tecnológicos**.

La impresión de páginas o documentos de cualquier tipo es una tarea cotidiana que **acompañía** a la utilización de los sistemas **informáticos**. Debido a ello han aparecido multitud de clases diferentes de impresoras Útiles para aplicaciones muy diversas. Sin embargo, actualmente se puede decir que los dos aspectos que marcan las diferencias entre las impresoras son la posibilidad de **imprimir** en color y la cantidad de páginas por **minuto** que es **capaz** de **imprimir** el **dispositivo**. Cuando se precisa una alta producción de **documentos** impresos, se **utilizan** **impresoras láser** en blanco y **negro compartidas** por multitud de **pues-**

tos. Las **impresoras láser** en color todavía presentan una velocidad y un coste que no las hace **operativas** en estos **entornos**. Tanto en el ámbito del usuario doméstico o incluso en algunos despachos se prefiere la **incorporación** de una impresora de inyección de tinta en color por puesto de trabajo. Estas **impresoras** son baratas y silenciosas, aunque todavía presentan el problema del alto coste del folio impreso en color.

En el capítulo se han considerado **otras impresoras** mucho más específicas utilizadas en aplicaciones muy particulares. El **propósito** ha sido el de mostrar la **gran** cantidad de periféricos de impresión que existen para preparar al lector para la llegada de nuevos dispositivos **más** complejos.

9.13 CUESTIONES

9.1 *Explicar brevemente los fundamentos de la **impresión láser**.*

9.2 *Explicar las ventajas y desventajas de la **impresión** de documentos utilizando dispositivos de mapa de **bites**.*

9.3 *¿Qué ventajas se obtienen de utilizar un lenguaje de **descripción** de páginas para enviar los datos a la **impresora**?*

9.4 *Indicar las diferencias que existen entre una impresora **electrostática** y una impresora **láser**.*

9.5 *¿En qué se diferencia un **trazador de inyección** de tinta de una impresora de inyección de **tinta**?*

9.6 *¿Cuál es el papel que desempeña el rayo **láser** en las impresoras **láser**?*

9.7 *¿Cuál es la utilidad del hilo cargador en una impresora **láser**?*

9.8 ¿En qué casos es más eficiente la utilización de un trazador frente a una impresora de mapa de bits?

9.9 Enumerar las ventajas y las desventajas de la introducción de los lenguajes de descripción de páginas.

9.14 BIBLIOGRAFÍA

- Computer Peripherals. *Barry M. Cook, Neil H. White.* 3^a ed. Edward Arnold. ISBN 0-340-60658-4. 1995.
- An Introduction to Microcomputer Systems. Architecture & Interfacing. *John Fulcher.* Addison Wesley Publishers Ltd. ISBN 0-201-41623-9. 1989.



CAPÍTULO 10

TERMINALES DE VÍDEO

10.1 INTRODUCCIÓN: TRANSDUCTORES ÓPTICOS

La visión es el sentido que aporta una mayor cantidad de información al ser humano. Por tanto, es interesante que el computador produzca señales luminosas como resultado del procesamiento de la información. Los transductores ópticos son los **periféricos** de salida de datos que convierten la información **digital** en luz.

Los transductores ópticos más sencillos son los leds (*light emitting diodes*). La mayoría de los computadores disponen de algunos leds para mostrar información básica del sistema: la alimentación de la CPU, alimentación del monitor o la utilización de la disquetera. Sin duda los transductores ópticos más populares en los computadores **domésticos** son los tubos de rayos catódicos o **CRT's** (*Cathode Ray Tubes*), conocidos por su utilización en los monitores de TV. En la actualidad se utilizan mucho otros transductores **ópticos**, como son los cristales de cuarzo líquido y las pantallas de plasma, debido principalmente a los avances impulsados por la introducción de los ordenadores portátiles.

Los transductores ópticos tienen en común que emiten luz para mostrar información **digital**. Este principio básico se consigue con sencillez en el caso de los leds, pero no se logra de forma tan inmediata en el caso de la pantalla **gráfica** de los computadores. Este dispositivo necesita una circuitería especial que codifique la información digital que se desea mostrar en las señales eléctricas necesarias para el transductor óptico.

Los *leds* están formados por una unión p-n de fósforo de galio (GaP) que cuando son atravesados por una corriente eléctrica emiten luz. La corriente típica que suele atravesar un *led* es del orden de unos pocos miliamperios y la caída de tensión entre sus extremos varía entre 0'5 y 3 voltios. La luz se emite desde los extremos de la unión y se refleja hacia el exterior por medio de un metal en el cátodo. Los leds se suelen empaquetar en plásticos coloreados formando lentes.

Varios leds se pueden agrupar para representar información numérica o alfanumérica. Estas agrupaciones toman el nombre de *displays*, siendo los *displays* de 7 segmentos los más populares para representar códigos decimales. Estos dispositivos utilizan siete leds en forma de barra llamados segmentos, tal como muestra la figura 10.1.

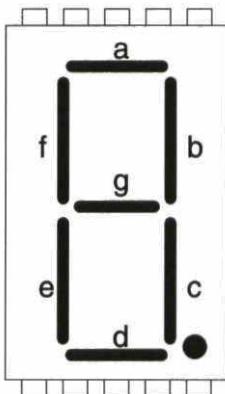


Figura 10.1: Display de 7 segmentos

Los displays de 7 segmentos suelen incluir lentes para aumentar la luz de los distintos segmentos. Este tipo de pantallas es adecuado para aplicaciones que requieren un número limitado de caracteres, por ejemplo para representar números en los equipos de prueba y medida. Habilitando combinaciones adecuadas de segmentos se pueden representar los números. El ordenador suministra la información del carácter a visualizar en código decimal codificado en binario 6 BCD, surgiendo la necesidad de implementar un decodificador de BCD a 7 segmentos para representar los caracteres.

Un alfabeto completo, con números y otros símbolos, se puede representar con un display de 16 segmentos, pero éstos no son excesivamente populares,

habiendo sido desplazados por las pantallas de matrices de **leds**, que pueden conseguir juegos de **caracteres** mucho **más** extensos. Normalmente se componen de 9 **filas** y de 7 columnas, pero requieren una lógica **más** compleja para representar los caracteres.

10.2 PANTALLAS DE RAYOS CATÓDICOS

Los **transductores** ópticos utilizados mayoritariamente en los computadores son las pantallas de rayos **catódicos** o CRT. La figura 10.2 muestra un esquema del corte transversal de un CRT típico. El funcionamiento de los **CRTs** se basa en la propiedad del fósforo de emitir luz cuando es bombardeado con electrones.

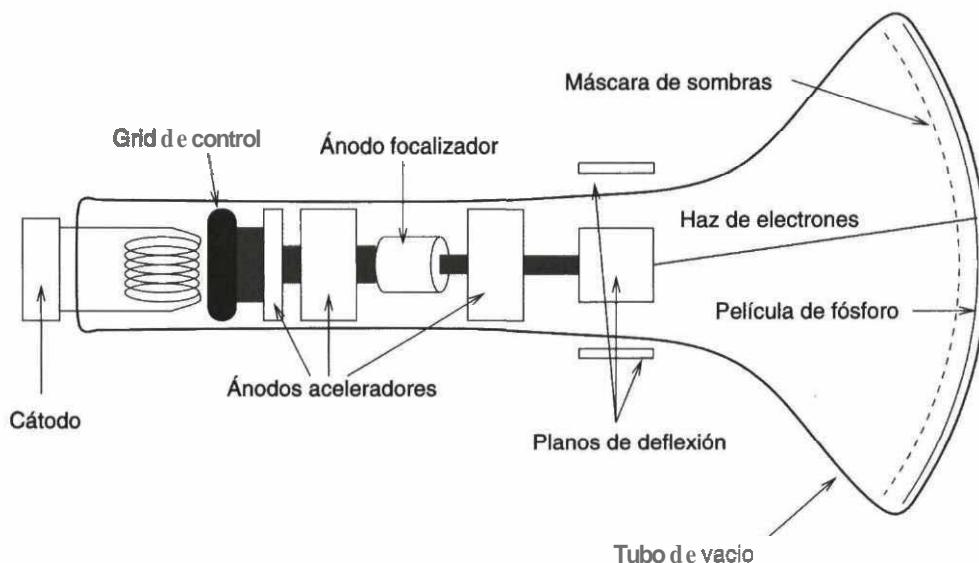


Figura 10.2: Esquema de una pantalla de rayos catódicos o CRT

Para producir una imagen los haces de electrones inician el bombardeo en la parte superior de la pantalla, **recorriéndola rápidamente** de izquierda a derecha dibujando una línea horizontal (observando la pantalla desde la parte exterior frontal). Al llegar al final del borde derecho se retorna a la posición original izquierda una línea **más** abajo y se vuelve a **recorrer** toda la **línea**, repitiendo este proceso hasta que se recorre toda la pantalla. El movimiento y la intensidad de los haces se controlan mediante la **información** suministrada por la **tarjeta** de

vídeo. La intensidad **del** haz de electrones en cada punto es la que controla la luminosidad del **píxel** en la pantalla. El proceso de bombardeo de la pantalla ocurre de forma suficientemente rápida **como** para que la pantaila completa se dibuje varias veces por segundo y no produzca efectos de parpadeo.

En el interior de un tubo de vacío se **calienta** un **cátodo**, a una temperatura de unos 800° C, de **forma** que emite un haz de electrones que incide sobre la pantalla de fósforo. Delante del **cátodo** se sitúa la rejilla de control que actúa sobre la intensidad del haz. Al aplicar un potencial negativo a la rejilla, ésta repele los electrones del haz (**puesto** que también están cargados negativamente) disminuyendo su intensidad. Si se aplica una tensión de -20 a -100 V se puede llegar a detener el haz completamente.

Después de la rejilla de control se encuentran un conjunto de ánodos cargados positivamente. El primer **ánodo** (cargado a unos 400 V) atrae al haz y lo acelera hacia la pantalla. El segundo está conectado con la **parte** interior de la pantalla y sirve para acelerar **más** aún el haz. Estos **ánodos** están cargados a un potencial entre **15 kV** y **25 kV**. En medio de estos dos existe otro ánodo, llamado **ánodo focalizador**, que tiene un potencial **variable**, entre 0 y 400 V, y es útil para enfocar el haz de modo que converja sobre un pequeño punto en la pantalla.

El haz debe orientarse para que llegue a cualquier punto de la pantalla. Esta orientación se logra por medios **electrostáticos** o por medios **magnéticos**. Los primeros utilizan dos planos para la **deflexión** horizontal y dos para la vertical, **formándose** un campo **electrostático** que atrae el haz hacia un plano y lo repele hacia el otro. **Este** sistema se utiliza en los **instrumentos** de medida, puesto que es fácil desviar el **haz** rápidamente. Los segundos utilizan cuatro bobinas que se **encuentran** fuera del tubo y que generan un campo magnético. Estos últimos se utilizan en aplicaciones **gráficas** donde la longitud del tubo debe ser reducida y no se precisa de una velocidad de **deflexión** muy alta.

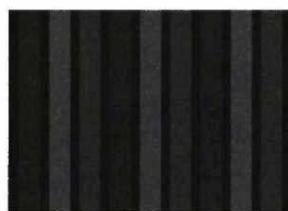
El **rápido** refresco de la pantalla (50 a 100 veces por segundo) consigue engañar al ojo humano, produciendo la impresión de una imagen fija. Dependiendo **del tipo** de fósforo, la imagen persiste iluminada un mayor o menor tiempo, compensando el parpadeo. Sin embargo, si la persistencia es demasiado grande, cuando se **refresca** de nuevo la pantalla con otra imagen se pueden producir sombras no deseadas debido a la retención de las imágenes previas.

La generación de las señales que controlan el **recorrido** del haz. llamadas señales de barrido horizontal y barrido vertical, y de la señal que define la intensidad del haz, se realiza mediante una **circuitería** especial que se encuentra dentro del monitor. Estas **señales** a su vez utilizan las señales de la tarjeta de video, la cual accede a la información **digital** que se desea representar en pantalla (típicamente localizada en una zona especial de memoria). La intensidad de los puntos o **píxeles** sobre la pantalla queda determinada por la señal de video que se traduce en un voltaje aplicado a la malla de control. Para que el **CRT** pueda generar las señales de barrido también se le deben de pasar las señales de sincronismo, tanto vertical como horizontal.

El funcionamiento de los tubos de rayos **catódicos** de color es muy similar al de los monocromos. En este caso se usan 3 haces de electrones para producir los colores primarios en la pantalla, rojo, verde y azul (de las siglas en inglés de estos colores **Red**, **Green** y **Blue** proviene el nombre de **monitores RGB**). En estos **monitores** fósforos de diferente color se agrupan en bloques de tres sobre la **superficie** de la pantalla formando un **patrón** determinado. A cada bloque se le llama **dot** (punto) y existen diversas formas de agruparse, tal como se muestra en la figura 10.3. Una separación **típica** entre puntos en el caso de 3 puntos circulares es de **0'25mm**.



Puntos circulares en
formación triangular



Malla de apertura



Máscara de ranuras

Figura 10.3: *Varios tipos de agrupaciones del fósforo RGB en los monitores de color*

Cada haz apunta a su **correspondiente** color y su intensidad está controlada por una señal diferente suministrada por la tarjeta de video. Los colores se forman regulando la intensidad de los haces sobre los **colores** primarios.

Para crear una imagen precisa, es necesario asegurar que el haz de electrones de cada color incide únicamente sobre el fósforo adecuado para ese color. Para

ello se utiliza una pequeña lámina de metal agujereada, llamada **máscara de sombras** (*shadow mask*), que hace que los haces de electrones pasen a través de los agujeros para evitar que incidan sobre los fósforos que no les corresponden.

Existe otro sistema similar, denominado malla de apertura (*aperture grill*), en el que en lugar de una máscara de sombras se utilizan cientos de tiras de metal que atraviesan la **superficie** de la pantalla verticalmente. Los fósforos se imprimen en la pantalla también en **tiras**, cada una de un color, de forma que el haz de un color **sólo** puede incidir sobre el fósforo de ese color. Las ventajas que tiene **éste** segundo método son que se permite pasar mayor parte del haz de electrones, con lo que se consiguen imágenes más **brillantes** y una pantalla plana verticalmente, con lo que se evitan **distorsiones**. Este sistema es el utilizado por los monitores **Trinitron** de **Sony**. La desventaja que tiene este tipo de pantallas es que para evitar que las **tiras** de metal vibren se utilizan una serie de finos hilos dispuestos horizontalmente, no apreciables en la imagen si no se observa muy de cerca. Puede ocurrir que la máscara de sombras se magnetice, dando lugar a colores incorrectos en la imagen. Con la intención de evitar este problema, cada vez que se conecta el monitor se le aplica un campo alterno de **desmagnetización**. A este proceso se le llama **degaussing** y las pantallas suelen incluir la posibilidad de realizar esta **desmagnetización** durante su funcionamiento normal mediante el uso de un pulsador colocado para tal fin.

Los monitores **CRTs** se utilizan tanto en pantallas que representen caracteres **alfanuméricos** como en **monitores gráficos**. De esta última clase existen dos tipos de monitores gráficos, dependiendo de cómo se controle el movimiento del haz de **electrones** sobre la **superficie** de la pantalla:

- a) **Gráficos vectoriales:** En este caso es posible desplazar el haz de electrones a cualquier zona de la pantalla con total libertad. La imagen a **visualizar** se traza repetidamente hasta que se necesita una nueva imagen. Sólo las partes de la pantalla necesarias para formar la imagen son activadas por el haz de electrones. Este tipo de pantallas poco comunes se utiliza para aplicaciones **gráficas**.
- **Gráficos raster:** La imagen se construye punto a punto por medio de una serie de líneas que se trazan sobre la pantalla, empezando por el borde superior izquierdo y moviéndose hacia abajo hasta la esquina inferior derecha. En este esquema el movimiento del haz de electrones es fijo y está controlado por un circuito de barrido, tal y como se ha explicado en la **sec-**

ción anterior. El haz **recubre** la pantalla lo suficientemente **rápido** como para que la imagen tenga una apariencia estacionaria y sin parpadeo. Toda la pantalla es alcanzada por el haz continuamente, aunque dependiendo del **tipo** de imagen a **visualizar** el haz no tiene la suficiente intensidad como para excitar el fósforo.

10.2.1 Señal de vídeo PAL

La representación de una imagen de televisión en blanco y negro se consigue asignando una cierta **luminancia** Y, o nivel de gris, a cada punto de luz. En Europa los CRT **raster** tienen 625 líneas, correspondientes con el **estándar de televisión EPAL (*European Phase Alternating Line*)**. Estas pantallas representan las **imágenes** de forma entrelazada, es decir, primero se trazan las líneas pares del monitor y a continuación las impares, en lugar de la representación secuencial de las 625 líneas. Se produce un refresco de cada imagen 25 veces por segundo, con lo que el cambio de cuadro (par o impar) se produce 50 veces por segundo. Por tanto la señal de sincronismo vertical tiene una frecuencia de 50 Hz. El entrelazado se usa en los monitores de televisión, pero no en los monitores de alta resolución de los ordenadores.

La duración del trazado de una línea horizontal es de $64 \mu s$. Este tiempo es la suma de un margen anterior de $1'5 \mu s$, un pulso de sincronismo horizontal de $4'7 \mu s$, un margen posterior de $5'8 \mu s$ y $52 \mu s$ de imagen de la línea, lo que da una frecuencia de barrido horizontal de $15'625 \text{ KHz}$. Los niveles de tensión son $0'020\text{V}$ para el nivel de sincronismo horizontal, $0'327\text{V}$ para el nivel negro y $1'041\text{V}$ para el blanco, tal como se muestra en la figura 10.4.

El sincronismo vertical se genera cada $312'5$ líneas con una duración de $2'5$ líneas, introduciéndose pulsos para mantener el sincronismo horizontal durante los cuales el haz se **inhibe**.

Las cámaras y los monitores RGB utilizan una codificación similar para mantener la compatibilidad con la señal PAL en blanco y negro, con este fin, a partir de las señales RGB se calcula la luminancia Y como una media ponderada de la intensidad de los colores que se sigue transmitiendo como la señal **PAL** en escala de grises. De manera adicional se transmite la señal R – Y y B – Y, durante el pórtico posterior de la línea PAL con una portadora de $4'43 \text{ MHz}$. De esta manera se pueden recomponer las señales R, G, y B.

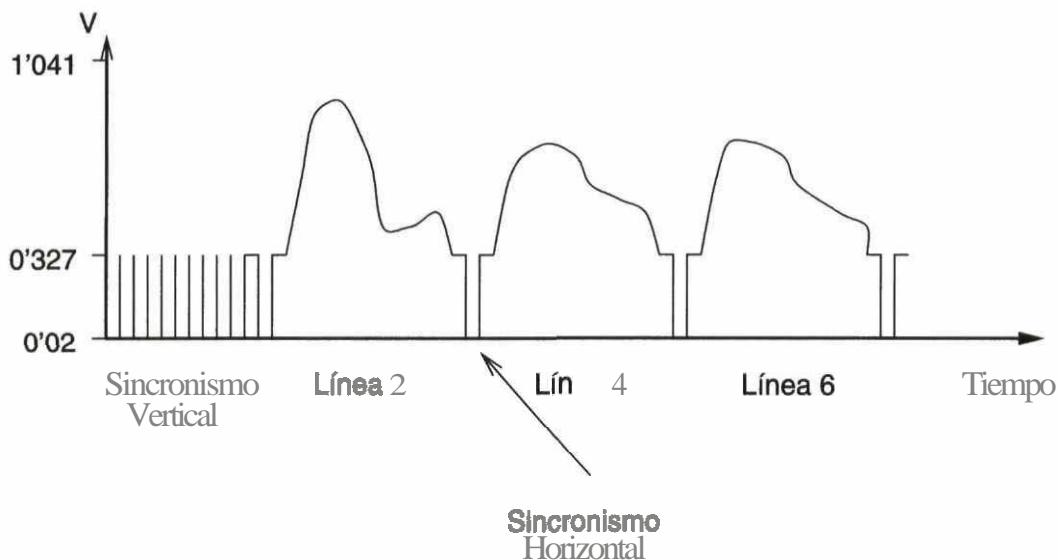


Figura 10.4: Evolución de la señal de vídeo PAL

10.3 TERMINALES DE VÍDEO RASTER

Los receptores de **televisión** pueden adaptarse para usarse como pantallas de ordenador, pero tienen bastantes limitaciones en **comparación** con las pantallas propias de **los** ordenadores (**VDT** o *Video Display Terminals*) que **presentan** las siguientes ventajas:

- El tamaño del punto del haz de electrones, cuando incide sobre la pantalla de **fósforo**, es considerablemente menor (normalmente **0'25 mm** de diámetro), por lo que puede aumentar la resolución de la pantalla
- Usan una **circuitería** de video de un ancho de **banda** especial mayor (el ancho de banda es la frecuencia máxima de la **señal analógica** que puede soportar el monitor), que **permite** resoluciones mayores.
- En el caso de monitores en color, se **usan** tres señales separadas para controlar los distintos haces de **electrones** y dos señales de sincronismo en vez de una señal de **vídeo** compuesto. Es por **ello** por lo que a este tipo de pantallas se les reconoce como monitores **RGB**.
- En vez de estar limitados a las frecuencias fijas de televisión de **15'625 KHz** horizontales y **50 Hz** verticales, los monitores **multisync** o **multifre-**

cuencia permiten variar estas frecuencias sobre un alto rango (**45 - 110 Hz** verticales). Esto se traduce en una mayor resolución, mayor frecuencia de refresco y por lo **tanto** menos parpadeo.

Las tendencias actuales incluyen la representación de los caracteres negros sobre fondo blanco, el **giro** de las pantallas 90° para representar mejor un folio **A4** y el incremento de la resolución sobre 1600 x 1200 para las aplicaciones gráficas.

10.4 CONTROLADORES DE PANTALLA

En un computador se almacena información (**gráfica** o alfanumérica) que se desea representar por la terminal de vídeo. Para generar la señal de vídeo y las señales de sincronismo de entrada al monitor es necesaria una circuitera especial que genere estas señales. Los dispositivos que implementan esta funcionalidad son los controladores de pantalla.

Un controlador de pantalla debe tener en cuenta 3 aspectos principalmente:

1. El acceso a la memoria de vídeo
2. La generación de la señal de video
3. La generación de las señales de sincronismo

10.4.1 Controladores de pantalla alfanumérica

Los **CRT** se pueden utilizar como pantallas alfanuméricas para representar caracteres. La información a representar en la pantalla se almacena en una memoria RAM, llamada memoria de vídeo, que contiene los códigos **ASCII** de los caracteres a visualizar. La pantalla tiene un formato de n filas por m columnas. Los caracteres, cuando son visualizados, se colocan en una fila y una columna determinada de la **pantalla**. Para visualizar nuevos caracteres **ASCII**, simplemente habrá que escribir en el área de memoria asignada al **CRT** los códigos **ASCII** de dichos caracteres.

Acceso a la memoria de vídeo

Existe una correspondencia entre las posiciones de la memoria de vídeo donde se almacena un **código ASCII** y la zona de la pantalla donde aparece **visualizado** el carácter correspondiente a ese código. Los métodos de **direccionamiento** de la memoria de **vídeo** definen esta correspondencia. La memoria de vídeo se **direcciona** usando dos métodos, tal como se **ejemplifica** en la **figura 10.5** para un monitor con 24 filas y 80 columnas:

0	1	...	79
80	81	...	159
...
1920	1921	...	1999

Direccionamiento
lineal

0	1	...	79
128	129	...	207
...
3072	3073	...	3151

Direccionamiento
por filas y columnas

A11	A7	A6	A0
		fila	columna

Figura 10.5: Ejemplo de direccionamiento en una pantalla alfanumérica

- Direccionamiento lineal:** El carácter que se ubica en la primera **fila** y columna de la pantalla ocupa la posición 0 y el **último** de la primera línea la posición 79. El primer **carácter** de la segunda línea se corresponde con la **posición 80** y así hasta el **último** de la **fila 24**. La ventaja de este método es que como **todas las** celdas son consecutivas, no se malgasta memoria. Sin embargo, es necesario un método complejo para averiguar la **posición** que ocupa un carácter en la memoria de vídeo dada su **posición** en la **pantalla**.
- Direccionamiento por filas y por columnas:** Se asegura que cada fila comienza en memoria en una posición **múltiplo** de una potencia de **2**. Para una pantalla de **80x25** se pueden utilizar 5 bits para indicar la fila y 7 bits para indicar la columna. Si se concatenan estos bits se obtiene que direcciones de **12** bits son suficientes para acceder a cada celda. El problema que tiene este método es que se malgasta mucha memoria.

Como ya **se** ha descrito, la memoria que almacena la información que se muestra por pantalla es la memoria de vídeo. Esta memoria de **vídeo** esta incluida en los controladores de pantalla, tanto alfanumérica como **gráfica**, y se **implementan** utilizando **ciertos** tipos **de** tecnología especiales:

- a **Video RAM (VRAM)**: Es un tipo especial de memoria dinámica RAM (DRAM) de doble puerto. Requiere **una** menor frecuencia de refresco y por tanto tiene un funcionamiento **más** óptimo que la DRAM habitual.
- **Extended Data Output DRAM (EDO DRAM)** : Es un tipo especial de DRAM que incluye un cauce **segmentado** en el puerto de datos, mejorando **&** esta manera el acceso a la memoria en modo página descrito en la sección 1.4.2. La segmentación consiste en dividir en etapas una tarea situando registros **intermedios** entre etapas. De esta manera se consigue una **aceleración** en la ejecución de las tareas, tal como se describe **en** la sección 12.2.
- **Synchronous DRAM (SDRAM)**: Es un tipo de memoria DRAM con señal de reloj que acelera el **acceso** en modo **página**.

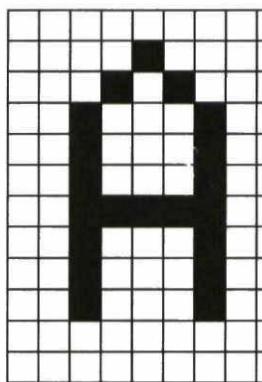
Tanto la CPU como el controlador de CRT acceden simultáneamente a la memoria de vídeo. El primero, para decidir qué caracteres se van a representar sobre la pantalla, y el segundo para realizar el refresco de la misma. Si no se utilizan memorias de doble puerto es necesario proveer de un mecanismo para arbitrar el acceso a la memoria de vídeo si no utilizamos memorias de doble puerto. Para ello se usan **varios** métodos:

- a Usar un controlador de DMA para manejar transferencias de bloques entre la CPU y la memoria de vídeo y entre la memoria de vídeo y el controlador (véase el capítulo 4).
- La CPU y el controlador comparten la memoria. Cada ciclo de reloj se divide en dos fases: en la primera accede la CPU y en la segunda el controlador.
- a La CPU sólo accede a la memoria mientras se está realizando el retorno horizontal o el vertical, que es cuando el controlador no escribe en la pantalla y por tanto no tiene que leer la información de la memoria.

Generación de la señal de vídeo

Cada carácter está representado por una **matriz** de puntos, llamada celda de **carácter** de **dimensión** $m \times n$, que indica la forma con la que aparecen los caracteres por la pantalla. Esta **información** se guarda en memoria ROM, en una tabla **llamada** generador de caracteres. Si se desea cambiar la forma de los caracteres, bastará con sustituir la ROM que guardan los patrones de bit del **carácter** de un tipo de **letra** determinado por otro.

En la figura 10.6 se muestra el **almacenamiento** en memoria de un carácter que utiliza una **matriz** de 12 x 9 **bits**. Cada bit de la celda de carácter se corresponde con un **dot (punto)** & la pantalla, que se ilumina si el bit correspondiente contiene un '1' y no se ilumina en caso contrario. Para representar el carácter por la pantalla se van leyendo del generador de caracteres las filas que lo constituyen y al mismo tiempo se van **serializando** los datos para ser enviados al circuito de control de video.



0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	0
0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0	0
0	0	1	1	1	1	1	0	0
0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Figura 10.6: Patrón de 12 filas x 9 columnas para la letra A

Para representar los caracteres en una pantalla **raster**, el controlador *de* video lee el **código ASCII** de una **fila** completa de caracteres de la memoria de vídeo. A **continuación** va generando la **señal** de vídeo a partir de la información que va obteniendo del generador de caracteres. La visualización de cada fila de caracteres se **realiza** generando en primer lugar la señal de vídeo para la primera línea de todos los caracteres, a **continuación** se genera la señal de la segunda línea y **así** sucesivamente hasta completar todas las líneas de la celda de **carácter**. Para refrescar toda la pantalla se **tendrá** que repetir la **operación** anterior con todas las **filas** de caracteres de la memoria de video.

Como se puede observar, éste es un proceso repetitivo cuya velocidad depende de la velocidad con que se leen los puntos de la celda de carácter. Es necesario un reloj, **llamado** reloj de punto (*dot clock*), que marca la frecuencia con la que se puede enviar la señal de vídeo de los píxeles que forman el carácter a la pantalla. A partir de este reloj, dividiendo su frecuencia mediante una cadena de contadores, se pueden obtener todas las frecuencias necesarias para el proceso de representación de los caracteres. Por tanto se tiene:

- **Frecuencia de carácter:** Velocidad con la que se representa una línea completa de celda de un carácter. Si los caracteres están representados en una matriz de 12 x 9 puntos, esta frecuencia será 9 veces menor que la frecuencia de punto.
- **Frecuencia de línea:** Velocidad con la que se representa una línea completa en la pantalla. Hay que tener en cuenta que se debe contabilizar el tiempo que tarda el haz en realizar el retorno horizontal y en trazar los márgenes, que suele ser normalmente el 20% del tiempo que se tarda en trazar una línea. Para una pantalla de 80 x 25 líneas, esta frecuencia será $80 + 16$ veces menor que la frecuencia de carácter.
- **Frecuencia de fila de caracteres:** Velocidad con la que se representa una fila completa de caracteres. Esta frecuencia en el caso del ejemplo de la matriz de 12x9 puntos será 12 veces menor que la frecuencia de línea.
- **Frecuencia de cuadro (pantalla completa):** Frecuencia con la que se representa toda la pantalla, incluyendo como en la frecuencia de línea, el tiempo que **tarda** el haz en pasar por los márgenes inferior y superior y en hacer el retorno. Este tiempo se aproxima al 10% del tiempo que se tarda en escribir las líneas. Por lo tanto la frecuencia de cuadro será $25 + 3$ veces menor que la frecuencia de fila de caracteres.

La estructura básica del controlador de pantalla alfanumérica se implementa mediante una cadena de contadores. El contador de fila y de carácter selecciona de la memoria de vídeo el carácter que se va a **visualizar**. El código ASCII del carácter sirve para acceder al generador de caracteres donde se encuentra la celda de carácter que define la forma que tendrá sobre la pantalla. El contador de las líneas de celda selecciona una de las filas de la celda del carácter actual que es enviada al registro de desplazamiento. El reloj de punto va desplazando los bits del **registro** de forma continua para ir generando la señal de vídeo. Este proceso

se repite para todas las **líneas** elementales de las celdas de todos los **caracteres de una fila** y para todas las **filas** de la pantalla. Cuando se termina con estas **filas** se repite todo el **proceso**. La figura 10.7 muestra de forma **esquemática** esta disposición encadenada de **contadores** en el caso de que se utilice **direcciónamiento por filas** y por **columnas**.

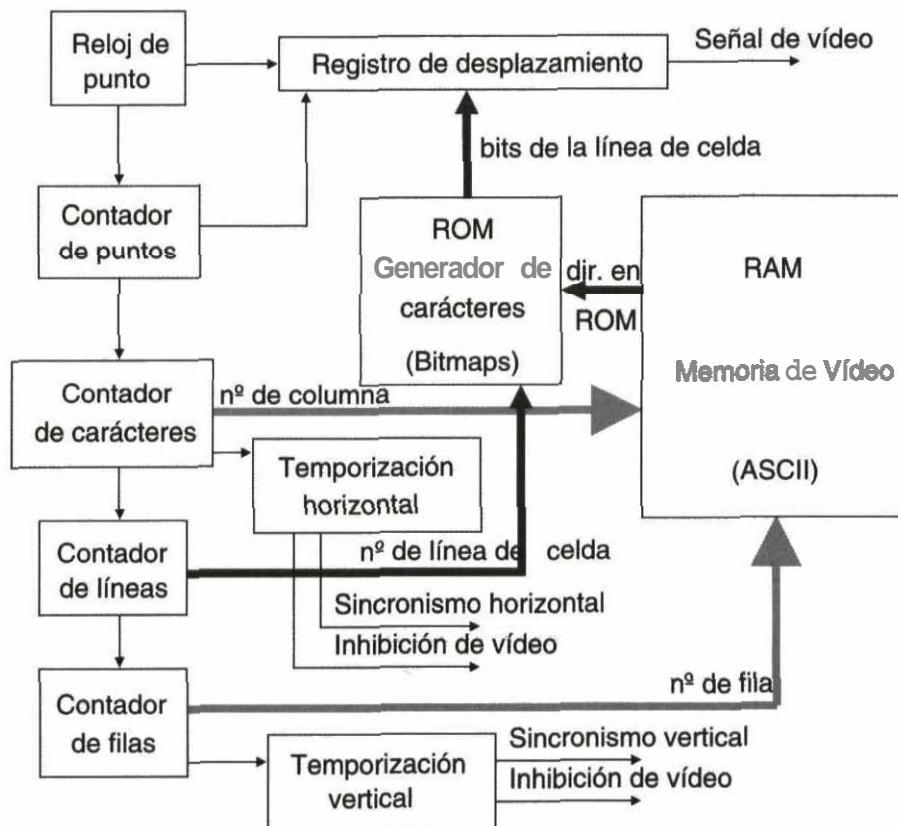


Figura 10.7: *Estructura de contadores alfanuméricicos de un controlador de pantalla alfanumérica*

Generación de las señales de sincronismo

Durante un **barrido horizontal** completo, se refresca la parte visible de la **línea**, se cubre el margen derecho, se **retorna** el haz y se **cubre** el margen izquierdo. Las duraciones relativas de estas fases suelen ser del **80%**, **4%**, **8%** y **8%** respectivamente del tiempo que se tarda en trazar los puntos visibles en una

línea. Lo mismo ocurre con el barrido vertical, **pero** en este caso suele ser el 10% del tiempo que **se** tarda en **trazar** una pantalla completa. En el controlador, **por** tanto, deberá haber una serie de **comparadores** para que en función de los valores de los contadores habiliten o deshabiliten la señal de **vídeo** (inactivo en los márgenes y en el retorno) y generen las **señales** de sincronismo.

10.4.2 Controladores de pantaila gráfica

El principal objetivo de las **pantallas** gráficas es el de producir dibujos en lugar de caracteres **alfanuméricos**. Por lo **tanto** será deseable una alta resolución, una amplia gama de colores y un refresco bastante rápido. Valores típicos para las pantallas gráficas pueden ser una **resolución** de **1000 x 1000** y 256 colores **simultáneos** de una paleta de **65.536**. El rápido **refresco** de la pantalla requiere para la memoria de **vídeo** el uso de RAM dinámica de alta velocidad.

Las **pantallas gráficas** no **se** suelen programar a nivel de bit, sino que se dispone de una serie de primitivas gráficas o **macros** que permiten dibujar **líneas**, **rectángulos**, círculos y otra serie de formas **geométricas**. Los circuitos integrados controladores gráficos empezaron a **desarrollarse** a partir & los **años** ochenta y hoy en día son parte fundamental en los controladores de pantalla **gráfica**.

En el controlador de pantalla **alfanumérica**, los códigos de los caracteres a **visualizar** **se** almacenan en la memoria de pantalla. Esta información se usa repetitivamente, **tantas** veces como líneas elementales tiene un carácter, para representar una fila de caracteres. Las posibilidades gráficas de estos **controladores** son por tanto muy limitadas.

Si se desea controlar **cada** punto de la pantalla por separado es necesario un controlador **gráfico**. Se utilizan controladores en los que al suprimir el generador **ROM** de caracteres, el usuario puede modificar la pantalla a nivel de punto. La información que describe el estado de cada **píxel** **se** guarda en una memoria RAM que se denomina 'plano de **bits**'. La memoria de **vídeo** no almacena el **código ASCII** y no existe un generador de caracteres, sino que la **información extraída** de la memoria de **vídeo** se almacena directamente en el registro de desplazamiento para ser **visualizada** por la pantalla.

Nombre	Carácter	Resolución	Frecuencia	Colores	Paleta
MDA	9x14		50Hz	2	
HGA	9x14	720x350	50Hz	2	
CGA	8x8	640x200	60Hz	16	
EGA	8x8	640x350	60Hz	16	64
VGA		640x480 320x200		16 256	$262144 \cdot 2^{18}$
SVGA				$256 \cdot 2^8$ $65536 \cdot 2^{16}$ $16777216 \cdot 2^{24}$	$16777216 \cdot 2^{24}$

Tabla 10.1: Tabla comparativa de las pantallas gráficas estándar

Si se almacena en el plano de bits un bit para cada píxel, se podrá representar una imagen en blanco y **negro**. Si se almacenan cuatro, se dispondrán de 16 niveles de gris o de 16 colores. Dependiendo del **estándar** utilizado se utilizará un número mayor o menor de **bites**. En la tabla 10.1 se comparan diversas tarjetas de **vídeo** consideradas como estándar. Las siglas de **MDA** corresponden a (*Monochrome Display Adapter*), **HGA** a (*Hercules Graphic Card*), **CGA** a (*Color Graphics Adapter*), **EGA** (*Enhanced Graphics Adapter*) y **VGA** a (*Video Graphics Adapter*).

Normalmente el plano de bits almacena 8 bits por píxel, que se **utilizan** para acceder a una tabla en memoria, llamada paleta de colores, donde cada entrada tiene 12 bits de **información** (cuatro para cada **color primario**), que son útiles para controlar la intensidad de los 3 haces de electrones del **CRT**. Los bits de **información** son previamente dirigidos a 3 convertidores **digital-analógicos** llamados **DACRAM**, que se encargan de generar la **señal de vídeo** que actuará sobre la rejilla de control de los cañones de electrones. La figura 10.8 muestra esta disposición con 8 bits por píxel y una paleta de colores de 12 **bits**. El número de planos de bits **determina** el número de colores simultáneos que se observan en la pantalla. De manera adicional, el número de bits de la paleta de colores indica el número de colores disponibles de los cuales se eligen los que se **visualizan** de forma simultánea.

Con el **estándar CGA** se pueden representar hasta 16 colores, puesto que se tienen 4 **señales**: Rojo, Verde, Azul e Intensidad. El **estándar EGA** usa 6

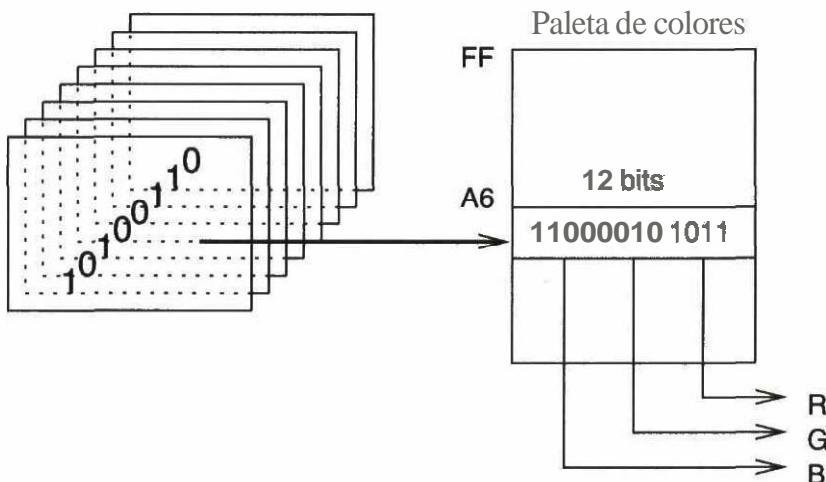


Figura 10.8: Acceso a la paleta de colores en una pantalla gráfica

señales: R, **r**, G, **g**, B y b, dando como resultado $2^6=64$ colores diferentes. En los monitores VGA se utiliza una paleta de colores con 6 bits para cada cañón (de un total de 3) con lo que se distinguen hasta $2^{(6+6+6)}=262144$ colores.

Si se desean cambiar los colores que se visualizan en el monitor, se puede utilizar otra paleta de colores, o bien se pueden cambiar entradas individuales en la tabla para definir colores propios.

En las **tarjetas gráficas** el contador de líneas elementales abarcará toda la pantalla, no teniendo sentido ya hablar del contador de caracteres. En estos **controladores**, el **direcciónamiento** lineal de la memoria de vídeo es imprescindible para no desperdiciar memoria, por lo que se debe incluir un contador de direcciones de memoria que funcione en paralelo con los contadores que generan el sincronismo. Este contador se incrementa cada vez que se accede a la memoria, a la frecuencia de punto. El incremento se **inhibe** cuando no se muestra **información** por la **pantalla**, es decir durante los retornos y los márgenes.

10.5 OTRAS PANTALLAS

Los **CRTs** son las pantallas que más se utilizan en los computadores debido a la alta calidad de las **imágenes** que pueden producir (número de colores, **re-**

solución y velocidad de refresco). Entre las desventajas más importantes de los tubos de rayos **catódicos** están el alto consumo y el tamaño, lo que limita su uso en los computadores **portátiles**. Hoy **en** día se están introduciendo otras tecnologías, impulsadas fuertemente por el **desarrollo** de los computadores portátiles. En esta sección se va a revisar la tecnología de este nuevo tipo de pantallas.

10.5.1 Pantallas de cristal líquido

Los cristales líquidos, descubiertos a **finales** del siglo **XIX**, son sustancias casi transparentes, que presentan **propiedades** combinadas de los sólidos y los líquidos. La luz que atraviesa los cristales líquidos se polariza en el sentido del cristal, siendo ésta una propiedad de los materiales sólidos. Posteriormente se **descubrió** que mediante campos **electrostáticos** cambiaba su alineamiento molecular, y de esta manera la **polarización** de la luz que los atraviesa.

La mayoría de los cristales líquidos son compuestos orgánicos formados por **largas** moléculas, que en su estado natural tienen sus ejes alineados **en** paralelo de manera aproximada. Es posible controlar el alineamiento de estas moléculas si se aplica un campo eléctrico. En este principio se basan las pantallas de cristal líquido o LCDs (*Liquid Crystal Display*).

Un LCD consiste **en** 2 filtros de **polarización** lineales con sus **líneas** dispuestas perpendicularmente tal como se **muestra** en la **figura** 10.9. Al incidir la luz natural (que tiene un eje de polarización aleatorio) sobre el primer plano **sólo** pasaría la luz polarizada en la dirección del primer plano. El segundo plano de polarización bloquearía el paso de la luz totalmente, ya que su eje de polarización es perpendicular al segundo. El cristal líquido situado entre ambos planos de polarización puede orientar su eje de polarización mediante la aplicación de un campo eléctrico. De esta manera se polariza la luz que atraviesa el primer plano en el sentido adecuado para que se pueda atravesar el segundo plano. Pegados en el interior del cristal existen unos **electrodos** de un material transparente conductor que polarizan una zona del cristal líquido. Para cada píxel deberá haber un control individual de los electrodos.

Los **LCDs** se pueden usar tanto en forma **reflectiva** como transmisible. En la primera se necesita luz ambiental para producir los **caracteres** legibles. Se utiliza un espejo en la parte posterior de la pantalla para reflejar la luz. En modo **transmisible** la pantalla debe ser iluminada desde la parte posterior.

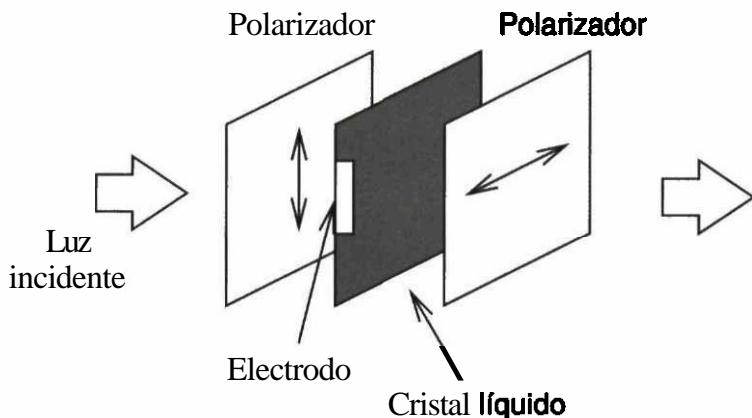


Figura 10.9: Estructura de una pantalla de cristal líquido

Este tipo de pantallas ofrece las ventajas de su tamaño reducido, no necesitan una fuente de alto voltaje para funcionar, la pantalla es plana y el acceso es lineal (ya que cada píxel se puede **direccionar** directamente en **términos** de filas y de columnas). Si se añaden filtros para colorear la luz es posible construir pantallas de matrices de puntos, siendo **éstos** incluso apropiados para aplicaciones de televisión.

10.5.2 Pantallas de transistores de película fina

Este tipo de pantallas LCD utilizan la tecnología **TFT** (*Thin-Film Transistor*) o de matriz activa en la que se añade una matriz extra de transistores al panel LCD (un transistor para cada color **RGB** de cada pixel). Estos transistores mejoran el tiempo de respuesta y el contraste de un LCD normal. Las **TFTs** pueden ser mucho más delgadas y luminosas que los LCDs comunes, siendo hoy en día las pantallas más utilizadas como monitores de computadores portátiles.

10.5.3 Pantallas de plasma

Una de las mayores desventajas de los LCDs es que tienen muy restringido el ángulo de visión. En las **pantallas** de plasma cada píxel es como un tubo de **neón** en miniatura consistente en un ánodo y un **cátodo** con un gas de plasma

entre ellos que suele ser una mezcla de **neón** y de **argón/xenón**. Si se aplica un voltaje entre los electrodos, el gas se **ioniza** y emite luz.

Los paneles de plasma incorporan **toda la circuitería** de interconexión para que puedan ser controlados directamente por la salida **estándar** de un monitor **RGB**. **Las** pantallas actuales llevan control sobre la luminosidad de los puntos pudiendo representar varios niveles de gris.

Las ventajas de estas pantallas son la **gran** luminosidad, la ausencia de parpadeo y que son muy compactas. Su principal desventaja es el gran consumo de **corriente** que tienen (sobre los 200V).

10.6 CUESTIONES

10.1 Describir de manera breve los principios de funcionamiento del tubo de rayos **catódicos**. ¿Cuál es la diferencia entre un tubo de rayos **catódicos monocromo** y uno en color? **Explicar** de qué maneras se consigue componer señales de colores.

10.2 La señal de vídeo **PAL** fue originalmente establecida para transmitir señales de televisión en blanco y negro. ¿De qué manera se introdujo la señal **PAL** en color manteniendo la compatibilidad de las señales?

10.3 Enumerar los **métodos** por los que se **arbitra** el acceso de la CPU y del controlador de la pantalla de rayos **catódicos** a la **memoria** de vídeo.

10.4 Un **CRT** funciona con 3 **señales**: vídeo, **sincronización horizontal** y **sincronización vertical**. Dibujar un esquema de un **CRT** indicando para qué se utilizan y dónde **actúan** cada una de estas seriales.

10.5 En una aplicación **CAD** es interesante disponer de una pantalla con una buena resolución y de un monitor que no **cance** demasiado la vista. Si se supone que la resolución de la pantalla es de 1000x1000 y que se tiene un monitor

no entrelazado de 100 Hz de frecuencia de **refresco** vemos. ¿Qué frecuencia elemental de punto (**dot-clock**) tiene que ofrecer la controladora **gráfica** para poder hacer funcionar al monitor con estos supuestos? **NOTA:** Considerese que el tiempo que tarda el haz en hacer el retorno **horizontal** y en recorrer los **márgenes** es el 20% del tiempo que se tarda en recorrer una **línea**. Para el retorno y los márgenes verticales se **supondrá** que el tiempo es del orden del 10% del tiempo que se tarda en recorrer todas las líneas de la **pantalla**.

10.6 ¿Qué elementos de un controlador **gráfico** se pueden **modificar** para **reducir** el parpadeo? Incluir también los relacionados con el monitor.

10.7 ¿Por qué es peligroso manipular los monitores?

10.8 Explicar la relación que existe entre la paleta de colores y el **número** de planos de bit de que dispone un controlador gráfico.

10.9 En un controlador de **vídeo**, ¿qué es el **dot-clock** o también llamado reloj de punto?

10.10 En el diseño de un **driver para** la controladora de video se quiere que se refresque la pantalla a **una frecuencia** de 70 veces por segundo. Además se va a utilizar una resolución de **1024 x 768 puntos** con la posibilidad de visualizar 2^{16} colores simultáneos de un rango de 2^{24} . Calcular la frecuencia de punto con la que se deberá enviar la **información** de video al monitor; la memoria de video necesaria y el **tamaño** de la paleta de colores.

10.11 ¿Qué es el generador de caracteres de un controlador **alfanumérico**?

10.12 ¿Qué son las **pantallas** de cristal **líquido**? Describir los **principios básicos** en los que se basa su **funcionamiento**.

10.7 BIBLIOGRAFÍA

- **Barry M. Cwk , Neil H. White.** *Computer Peripherals.* 3^a ed. Edward Arnold. ISBN 0-340-60658-4. 1995.
- e **John Fulcher.** *An Introduction to Microcomputer Systems. Architecture & Interfacing.* Addison Wesley Publishers Ltd. ISBN 0-201-41623-9. 1989.
- **Hans-Peter Messmer.** *The Indispensable PC Hardware Book.* 3^a ed. Addison Wesley Longman, Inc. 1997. ISBN 0-201-40399-4.

CAPÍTULO 11

ALMACENAMIENTO MASIVO

11.1 INTRODUCCIÓN

Los **primeros capítulos** del texto se centraron en el funcionamiento interno del computador y en el proceso de **ejecución** de las instrucciones almacenadas en memoria principal. Sin embargo, normalmente esta memoria del computador es volátil y tanto el sistema operativo, como los programas y los datos, desaparecen al **interrumpirse** la alimentación del sistema. Para evitar este problema, es necesario que el sistema operativo y los programas estén almacenados en alguna memoria no volátil a la que pueda acceder el **procesador** para cargarlos en memoria principal. Este espacio de memoria constituye el almacenamiento **secundario**, y permite salvar de forma masiva gran cantidad de programas y datos.

Con la invención de la memoria virtual, parte de la capacidad del almacenamiento secundario se utiliza para guardar páginas de memoria virtual, que **forman** parte de los programas en **ejecución**. De esta manera, el almacenamiento secundario debe tener capacidad para almacenar gran cantidad de &tos a un coste reducido y ser lo suficientemente rápido como para no degradar el rendimiento al acceder a estas **páginas**.

A lo largo de este capítulo se van a **describir** las características fundamentales de los dispositivos más representativos utilizados en el almacenamiento secundario. Todos los dispositivos tienen en común la utilización de medios

magnéticos u ópticos para el almacenamiento de información digital, codificada en base a conjuntos & 1's y 0's. Para almacenar un 1 o un 0 se utiliza una pequeña posición del medio magnético u óptico, que se modifica para poder diferenciar su contenido. Los dispositivos de almacenamiento secundario más comunes son los discos, tanto magnéticos como ópticos, y las cintas, que empezaron como una evolución de la tecnología de **audio**. Hoy en día se comparten los dispositivos para ambos propósitos, utilizándose *Compact Disc* y cintas de audio **digital** (DAT) para almacenar archivos.

11.2 GRABACIÓN MAGNÉTICA

El material magnético está formado por un gran número de pequeñas áreas, compuestas por **billones** de **átomos**, con un campo magnético propio. Si estos campos **se** orientan de manera **aleatoria**, sus efectos se compensan y **macroscópicamente** el material no parece magnetizado. Bajo la influencia de un campo **magnético** externo fuerte, **todas** los campos internos se orientan en la misma dirección y sentido, presentando el **material** un estado de **magnetización**. Una vez que desaparece el campo externo el **material** magnético conserva su estado, ya que este proceso tiene un cierto grado de **histéresis**.

La grabación magnética consiste en un proceso que magnetiza posiciones del material con campos diferentes. Para generar el campo magnético externo se emplea un cabezal formado por un núcleo de **ferrita**, al que se le **arrolla** una **bobina**. Si por la bobina pasa una corriente constante de 10-20 mA se forma un campo magnético constante en el núcleo, que orienta los campos del material magnético. Un cambio en el sentido de paso de la corriente en la bobina hace que el sentido del campo magnético se invierta, creando una zona de material magnetizada **inversamente**. El material **magnético** se divide en posiciones llamadas celdas de bit, donde todas las partículas están magnetizadas en el mismo sentido. Dependiendo del **código** de grabación utilizado, se usan una o varias celdas de bit para cada uno de los bits de **información** que se desean almacenar. Para poder determinar los límites de estas celdas de bit, **es** necesaria la existencia de mecanismos de **sincronización**, ya que pueden existir varias celdas consecutivas con la misma magnetización.

Cuando las celdas se magnetizan con polos Norte y Sur sobre la superficie magnética, el proceso de denomina grabación horizontal. También es posible encontrar métodos de grabación vertical o perpendicular, que crean celdas a través de todo el medio con el polo Norte en una superficie y el polo Sur en otra. Con este método se consiguen celdas 30 veces más estrechas. Sin embargo la grabación horizontal es más simple y se puede realizar por ambas caras. La grabación vertical se utiliza en los discos magneto-ópticos.

Cuanto más pequeñas sean las celdas mayor será la densidad de información obtenida, sin embargo, si se hacen demasiado pequeñas se anularán unas con otras perdiendo la magnetización. El tamaño del área en el que se expande el campo magnético creado por el cabezal determina cuál es el tamaño de la celda de bit. Este factor depende de la distancia entre el cabezal y la **superficie** magnética, así como de la tecnología utilizada en el cabezal. Los cabezales más comunes son los de núcleo de femta, y los MR, formados por materiales **magnetoresistivos**. Los cabezales MR se basan en materiales que presentan variaciones de su resistencia eléctrica en presencia de un campo magnético externo. Pueden detectar pequeñas áreas magnetizadas, ya que poseen una gran sensibilidad, y permiten aumentar considerablemente la densidad de grabación. Este tipo de cabezal es el más utilizado actualmente.

El proceso de lectura utiliza el mismo cabezal, pero en este caso sin aplicar ninguna comente a la bobina que rodea el núcleo. Al mover el soporte magnético por debajo del cabezal, se producen variaciones en el flujo magnético que atraviesa el núcleo de **ferrita**, debidas a las diferentes **magnetizaciones** de las celdas de bit. Estas variaciones generan una comente inducida en la bobina que se utiliza para determinar la información almacenada. Si se mide la tensión en los extremos de la bobina se observan pulsos de voltaje ($100\mu V - 100mV$) cada vez que se produce un cambio en el flujo magnético que pasa por el núcleo. Este cambio en el flujo se da al pasar de una zona con un sentido de magnetización a otra zona con sentido distinto. En la figura 11.1 se muestra el esquema de lectura **magnética**.

A medida que las celdas se van haciendo más pequeñas, el proceso de **lectura** presenta el problema de que los pulsos de tensión producidos por cambios de flujo consecutivos interfieren entre si, siendo por tanto difícil determinar cuándo **empieza** y cuándo termina una celda. Si se basa la detección de las celdas en la detección de los pulsos de tensión, pueden ocurrir que la deformación en la

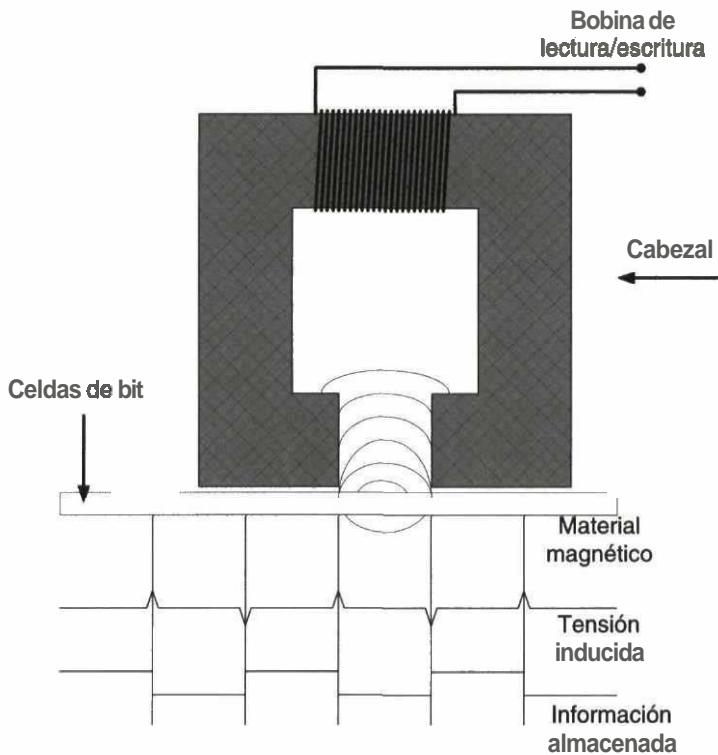


Figura 11.1: Lectura magnética

señal producida por la interferencia con los pulsos producidos por transiciones cercanas, no sean detectados por un sensor de nivel. Para evitar esta degradación e incrementar la densidad de grabación, se utiliza la técnica **PRML (Partial Response Maximum Likelihood)**. Esta técnica en lugar de detectar pulsos individuales para determinar los cambios de flujo, emplea algoritmos de procesado digital para manipular la señal analógica y averiguar la secuencia más parecida de bits que representa. El método se divide en dos pasos:

- **PR (partial response):** La señal analógica procedente del cabezal es **ecualizada** mediante un filtro programable, y a continuación se **muestrea** mediante un conversor analógico digital.
- **ML (maximum likelihood):** Convierte la señal digital a datos, para ello utiliza un detector de **Viterbi** que comprueba todas las posibles combinaciones de datos que pueden encajar, para buscar entre todas **ellas** la que presenta un menor error.

este método de **lectura** permite **incrementar** más de un 30% la densidad de **grabación**, y junto con los cabezales MR **forma** la tecnología utilizada en los discos duros actuales.

11.3 CÓDIGOS DE GRABACIÓN MAGNÉTICA

La información **digital** se almacena codificada en una secuencia de celdas de bit magnetizadas en ambas direcciones. La codificación **más** simple utiliza una celda para representar cada bit. En la práctica se utilizan **códigos** de grabación más sofisticados que **traducen** los datos **digitales** (secuencias de 1's y 0's) a una secuencia de celdas **magnetizadas** en dirección Norte o en dirección **Sur**, formando los datos magnéticos.

Para realizar las lecturas se **muestrea** la **superficie** magnética a una velocidad **predeterminada**, detectando de este modo la orientación de las celdas. Aunque la **longitud** de las celdas y la velocidad con la que se mueve la superficie **magnética** suele ser bastante precisa en todos los dispositivos, **pequeñas** variaciones de estas variables hacen que no se **pueda** asegurar **que** las celdas se vayan a leer correctamente. Si el circuito que **muestrea** las celdas va demasiado **rápido** o la **superficie magnética** va demasiado lenta, algunas celdas **magnéticas** serán **muestreadas** dos veces. Para resolver este problema, además de utilizar un dispositivo **temporizador** preciso, también se **extrae** información de reloj de la superficie magnética para poder resincronizar el dispositivo anterior. Así, cada vez que se detecta un cambio de flujo **magnético**, se **resincroniza** el reloj. También hay que asegurar que se produzcan cambios de flujo regularmente en las celdas, independientemente de la secuencia de **bits** que se quiera almacenar. Por lo tanto no **será** posible representar directamente los dos estados de la **magnetización** como 1's y 0's, sino que se tendrá que aplicar alguna solución alternativa, ya que una secuencia de muchos 1's o muchos 0's seguidos no produce cambios de flujo. Posibles soluciones a este problema son:

- Usar una codificación que garantice los cambios de flujo regularmente. A este tipo de códigos se les denomina **códigos autoreloj**.
- **Utilizar** una pista especial de celdas que van en paralelo con las pistas de datos que contengan una secuencia de cambios de flujo utilizada para resincronizar los circuitos de **temporización**. Para **leer** esta pista se **utiliza**

un cabezal adicional. Este mdtodo presenta los siguientes inconvenientes:

1. Se produce un incremento del coste al utilizar un cabezal **más**.
2. Esta nueva pista ocupa espacio que **podría** ser utilizado por otra pista de datos.
3. La **superficie** se puede deformar, con lo que existirá un pequeño retraso en la información de reloj con respecto a los datos de una pista interna que puede llegar a inducir **algún** error.

Normalmente se utiliza el primer mdtodo para almacenar la información. Algunos métodos de **codificación** requieren una celda para representar cada bit, mientras que otros requieren más. Algunos de elios **también** se utilizan en la transmisión de datos, mientras que otros más complejos **sólo** se utilizan para la grabación magnética.

Códigos no autoreloj

- No retorno a **cero** (**NRZ**): Es el mdtodo más simple. Un 1 se representa por una celda magnetizada en sentido Norte y un 0 por una celda magnetizada en sentido Sur.
- No retorno a **cero** invertido (**NRZI**): Un 1 **se** representa invirtiendo la dirección del flujo. Un 0 se representa no cambiando la dirección del mismo.

Códigos autoreloj

- **Codificación de fase (PE)**: Para cada bit se necesita producir dos cambios de flujo, por lo que con las mismas celdas de bit, representa la mitad de datos que en los dos ejemplos anteriores. Se almacena un 1 mediante un flujo con sentido Sur seguido de un flujo con sentido Norte y un 0 se representa de forma **contraria**.
- **Modulación en Frecuencia (FM)**: Siempre existe un cambio de flujo al comienzo de cada bit. Para almacenar un 1 **se** utiliza otra celda adicional para producir un cambio de flujo. Los 0's se graban a una frecuencia y los 1's al doble de esa frecuencia.

- **Modulación de frecuencia modificada (MFM):** Es la misma que el código FM excepto en que el cambio de flujo al principio de cada codificación de bit se produce únicamente si el bit actual y el anterior son Os. Con ello se garantiza que **sólo** se necesita un cambio de flujo para cada bit.
- **Carrera de distancia Limitada (*Run Length Limited* o *RLL*):** Está constituida por un conjunto de códigos que definen el número de celdas máximo y mínimo (*run*), que debe haber entre dos cambios de flujo. Como estos cambios de flujo se utilizan para sincronizar el reloj de lectura, cuanto mayor sea este número más tiempo **transcurrirá** entre dos **sincronizaciones** sucesivas, y por lo tanto los dispositivos deberán ser más precisos. La distancia máxima indica el número máximo de celdas iguales que **permiten** que los cambios de flujo sean lo suficientemente frecuentes como para que el código sea autoreloj. La distancia mínima permite una codificación de alta densidad, ya que las celdas **pueden** ser más pequeñas al haber siempre como mínimo varias celdas iguales seguidas.

Dentro de este tipo se encuentra la grabación por grupos codificados (GCR), tal como se muestra en la tabla 11.1, que **agrupa** los bits a grabar en pequeñas secuencias y le asigna un código a cada una de ellas asegurando que se produce al menos un cambio de flujo. El nuevo código se graba usando un código eficiente como el NZRI. Este código tiene una distancia **mínima de 1 (101 ó 010)** y una distancia máxima de 8 (011111110), es por tanto un *RLL*_{1,8}. Además, como **sólo** se utilizan 16 de los 32 posibles códigos de 5 bits, se puede utilizar la **información** redundante para detectar errores.

El código **RLL** más utilizado es el *RLL*_{2,7}, que agrupa los bits a codificar en secuencias de 2, 3 ó 4 bits y les asigna un código que duplica el número de **bits**. La secuencia se describe como una secuencia de códigos-S (espacios) y de códigos-R (inversiones). Este código garantiza que siempre habrá al menos 2 códigos-S entre dos códigos-R y que no hay más de 7 códigos-S entre dos códigos-R. Como se necesitan tres códigos **R/S** como mínimo para producir un cambio de flujo, en cada celda magnética se podrán representar tres códigos. **Así**, aunque se doble el número de códigos para representar un bit, a la hora de almacenarlos es posible agruparlos en secuencias de 3 códigos, resultando una densidad de 1.5 bits por celda magnética. El **código RLL**_{2,7} se muestra en la tabla 11.2.

GCR	
0000	11001
0001	11011
0010	10010
0011	10011
0100	11101
0101	10101
0110	10110
0111	10111
1000	11010
1001	01001
1010	01010
1011	01011
1100	11110
1101	01101
1110	01110
1111	01111

Tabla 11.1: Código GCR

RLL	
10	SRSS
11	RSSS
000	SSSRSS
010	RSSRSS
011	SSRSSS
0010	SSRSSSRSS
0011	SSSSRSSS

Tabla 11.2: Código RLL_{2,7}

La figura 11.2 compara la densidad de grabación de los códigos descritos anteriormente. Cuantos más bits puedan ser representados con un único cambio de flujo, mejor será el código. Hay que tener en cuenta que la celda magnética tiene un tamaño mínimo que limita la densidad. Así, los códigos PE y FM permiten una codificación de 0.5 bits por celda, los NRZ, NRZI y MFM almacenan 1 bit y el código $RLL_{2,7}$, que es el que mayor densidad de grabación permite, alcanzar 1.5 bits por celda magnética.

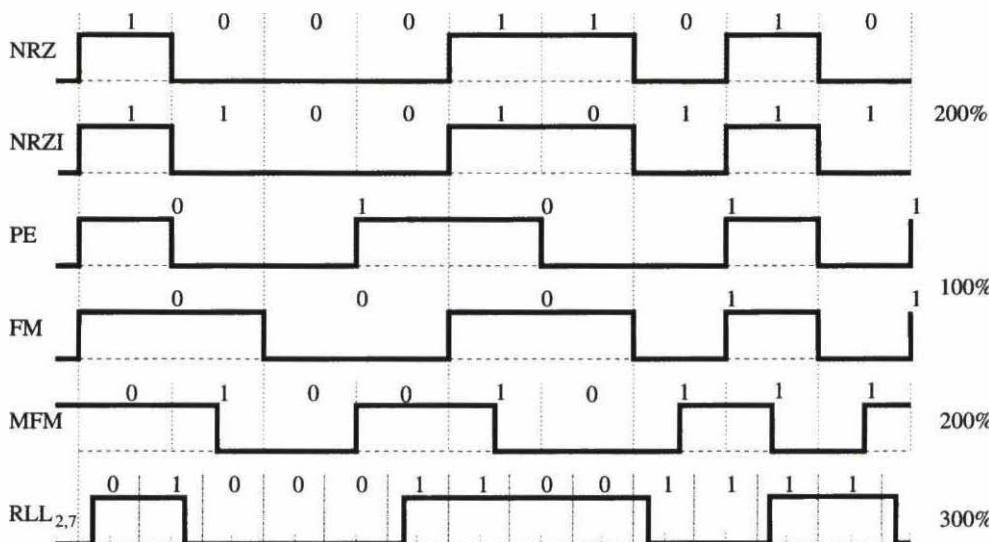


Figura 11.2: Códigos de grabación magnética

11.4 DISCOS MAGNÉTICOS

Los discos magnéticos son los **dispositivos más** comúnmente utilizados en el **almacenamiento** secundario. Se caracterizan por poder tener un acceso aleatorio a la **información**, tiempos de acceso pequeños, una gran capacidad de almacenamiento y un bajo coste.

Un disco **está** formado por uno o varios platos que giran **simultáneamente** movidos por el mismo motor, teniendo **cada** plato sus propios **cabezales** de lectura/escritura, tal como se muestra en la figura 11.3. La información se almacena **distribuida** en pistas **concéntricas**, grabadas sobre el **material magnético** en una o en las dos caras de **cada** plato que hace de **sustrato**. Para acceder a los datos

almacenados, existe un brazo de acceso sobre el que se montan los cabezales. El posicionamiento de los cabezales se realiza gracias a un **codificador** y un engranaje, que convierte el movimiento de rotación del motor paso a paso en movimiento lineal del brazo. Para acceder a los datos, el movimiento lineal del brazo y el de **rotación** del disco, sitúan el cabezal sobre los datos **almacenados** en cualquier pista. En el peor de los casos, para acceder a un determinado dato, el cabezal necesitará esperar una vuelta entera del disco y mover **linealmente** por completo el brazo de acceso. En los discos flexibles el tiempo de una **rotación** es como **mínimo** de 170 **ms** y el movimiento lineal del **brazo** necesita unos 200 **ms**, como estos dos movimientos se **realizan** en paralelo el tiempo de espera es aproximadamente de 200 **ms**.

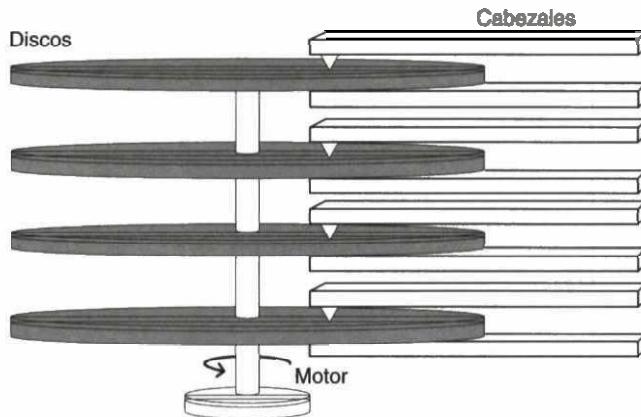


Figura 11.3: Estructura de un disco

Cada pista se divide en sectores formando una estructura de **tarta**, **tal** como se muestra en la figura 11.4. Cuando el disco **está** formado por varios platos, al conjunto de pistas que tienen el mismo número se le denomina cilindro. No toda la superficie del disco se **utiliza** para **almacenar** datos, **sólo** se usa la parte **más** exterior, ya que debido a la velocidad angular constante del disco, los patrones **magnéticos** de la superficie del interior quedarían muy comprimidos. A su vez, cada sector contiene un conjunto de bloques de tamaño fijo donde se almacena la información. Cada bloque dentro del disco se distingue por su posición especificada mediante un número de bloque, sector, cilindro y cabezal.

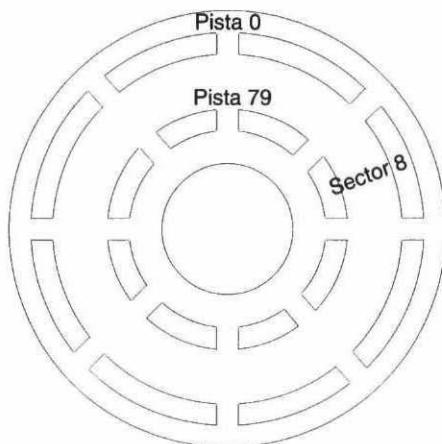


Figura 11.4: División del disco

11.4.1 Discos flexibles

Este tipo de disco está formado por un **único** plato de plástico flexible recubierto de material magnético. El cabezal está en contacto con la superficie del disco, lo que permite que los campos magnéticos necesarios para **introducir** una comente en el cabezal sean más pequeños. De esta **manera** es posible obtener celdas magnéticas de un tamaño **menor** y una mayor **densidad de grabación**. Sin embargo, el desgaste de la superficie del disco es mayor, lo que acorta su vida. Para no degradar en exceso la **superficie**, el disco no **está continuamente girando**, sino que **sólo** lo hace cuando se **accede** a los datos. El disco se protege de daños mediante una funda plástica rígida que lo envuelve. El **plástico** dispone de cuatro zonas agujereadas:

- Una **rectangular** en la parte baja para **acceder** al **material** magnético. En algunos casos, esta **zona** se protege con una chapa metálica, que al **introducirse** en el lector se retira y deja el material **magnético** al descubierto, como en el caso del disco de 3.5".
- Una zona central circular que permite al motor girar el disco.
- Una zona pequeña **en** la parte superior izquierda que indica la densidad.
- La **última** zona es un pequeño agujero enmascarable en la parte superior derecha, que indica si se puede **escribir sobre** el disco.

Nombre	360k 5.25"	1.2M 5.25"	720k 3.5"	1.44M 3.5"	2.88M 3.5"
TPI	48	96	135	135	135
BPI	5,876	9,869	8,717	17,434	34,868
Densidad	DD doble	HD alta	DD doble	HD alta	ED extra - alta

Tabla 11.3: Clasificación de los discos por densidades

Los discos flexibles se pueden agrupar según tres características: tamaño, número de caras y densidad. Atendiendo a su tamaño los discos se agrupan en **8, 5.25** y 3.5 pulgadas, aunque los dos primeros **grupos** hoy en día están en desuso. En la figura 11.5 se muestran los discos de 5.25" y 3.5". Atendiendo al número de caras, se distingue entre discos utilizables por ambas caras o solamente por una. La densidad del disco consiste en la cantidad de datos que pueden **almacenarse** en una determinada cantidad de espacio, y depende de dos factores: la densidad de pistas que puede tener el disco, que mide la cantidad de pistas que pueden ser empaquetadas en una pulgada de radio del plato, y la cantidad de bits en cada pulgada de pista. Hay que destacar que este **último** factor no es constante a lo largo de todas las pistas del disco, puesto que las pistas más **internas** tienen una longitud menor y su densidad de bits por unidad de longitud debe ser mayor. El producto de estos dos factores nos da la cantidad de bits que pueden almacenarse por unidad de **área**. Este factor se suele aplicar para especificar la capacidad de los discos duros. Por otro lado, los discos flexibles se suelen clasificar por dos factores separados, como son la densidad de pistas, medida en pistas por pulgada (**TPI**), y la densidad de bits, medida en bits por pulgada (**BPI**). En la tabla 11.3 se muestra una clasificación de discos de 5.25" y 3.5" según su densidad: doble, alta o extra-alta.

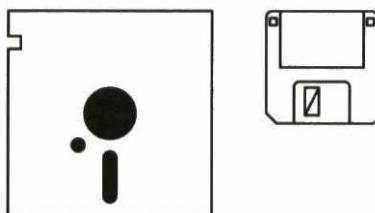


Figura 11.5: Discos flexibles de 5.25 y 3.5 pulgadas

Los discos flexibles de 3.5" y 5.25" de alta densidad poseen 80 pistas. El **número** de sectores varía con el modelo, pero alcanza como máximo los 18 por pista en el caso de 3.5" alta densidad. La anchura de las pistas también vacía

entre los **0.330 mm** del **5.25"** baja densidad y los **0.115 mm** del **3.5"** alta densidad. El código de grabación utilizado es FM o MFM, aunque la modulación FM esta obsoleta debido a que **ofrece** una densidad de grabación igual a la mitad de la **permitida** por la codificación MFM. Por cuestiones de compatibilidad los controladores de disco flexible actuales suelen soportar las dos codificaciones.

11.4.2 Discos duros

Los discos duros **están** formados por **varios** platos que **están** continuamente girando y poseen una capacidad y velocidad superiores a la del disco flexible. **Además**, para alcanzar altas densidades de grabación, el **equipamiento** mecánico y el control del disco es más preciso.

Disco Winchester

Es el disco duro por excelencia. Los platos **están formados** por un material metálico, **normalmente** aluminio, ya que su ligereza reduce la inercia del disco. El metal se **recubre** con óxido de níquel o cobalto que permite altos niveles de densidad de grabación. Suelen tener un **tamaño** de **3.5"**, para que el tiempo de posicionado del cabezal sobre la pista sea pequeño, al ser el radio del disco pequeño.

En estos discos el cabezal no está en contacto con la superficie del disco, sino que vuela por encima del mismo a una distancia que varía dependiendo del tipo de cabezal y de su forma.

Al girar el disco se produce una comete de **aire** entre su superficie y el cabezal que a modo de colchón lo mantiene elevado a una distancia que varía dependiendo de la **aerodinámica** del cabezal. **Otro** factor que influye es la presión del **aire**, y por lo tanto a mayor altura sobre el nivel del mar la presión disminuye y el cabezal se acercara más. La distancia típica para los cabezales de **ferrita** es de $1\mu m$. Cuando el disco para, el cabezal **aterriza** sobre el disco. Para evitar que se dañen los datos se elige alguna pista no **utilizada** como zona de parada del cabezal. Este proceso se denomina *parking* y la pista especial elegida se denomina la zona de aterrizaje.

Para proteger el disco de los peligros que representan las impurezas y el polvo, se **sella** totalmente y la entrada de **aire** se **regula** mediante un filtro.

La velocidad a la que giran los discos es superior a **7200 rpm**, con un circuito de control que la regula para reducir sus variaciones a menos del **0.1%**. Esto supone una velocidad **10** veces superior a la de los discos flexibles.

Por último, hay que destacar que el número de platos varía de 2 a 8, y la capacidad puede **alcanzar** varias decenas de **GigaBytes**.

11.4.3 Organización de los discos magnéticos

El disco se divide en pistas, y cada una de éstas en un número fijo de sectores donde se conforman las celdas **magnéticas**. Al utilizar esta **configuración**, aparece el problema de que las pistas interiores del disco son de longitud menor, pero como todas contienen los mismos sectores, la densidad de celdas es mucho mayor en las pistas internas. Ésta es la causa de que la parte interna no se utilice, ya que las celdas **serían** demasiado pequeñas. Normalmente un disco flexible contiene cerca de 100 pistas con 8, 18 o 30 sectores y un disco duro alrededor de las 1000 pistas con unos 35 o 63 sectores. El espacio entre pistas es varias veces mayor que la anchura de la misma para facilitar su localización.

Para evitar el problema de los sectores internos, se utiliza una técnica que divide el disco en varias zonas (*zoning*), teniendo cada zona un número de sectores constante, tal como se muestra en la figura 11.6. En la zona interna, como hay menos sectores, las celdas magnéticas tendrán el mismo **tamaño** que las de la zona **externa**. La zona externa tendrá más sectores, y como la velocidad angular es la misma, éstos se leerán mucho más rápido. Es interesante poner en esa zona los bloques más utilizados para aumentar el rendimiento.

El tiempo que se tarda en **transferir** un bloque de datos desde el disco al controlador de disco se divide en el tiempo de transferencia de los datos y el tiempo de acceso, que es el tiempo que tarda el cabezal en posicionarse justo encima del bloque que se desea leer. El tiempo de acceso se divide a su vez en el tiempo de búsqueda, tiempo que tarda el cabezal en posicionarse sobre la pista adecuada y el tiempo de latencia, tiempo que espera el cabezal a que el sector requerido se posicione debajo de **él**.

Los mejores tiempos de acceso se lograrán al acceder sucesivamente a bloques localizados en sectores consecutivos dentro de la misma pista, puesto que se anula el tiempo de acceso. Sin embargo, algunos controladores no tienen tiempo suficiente para leer dos sectores consecutivos, ya que mientras procesan los datos del primero el segundo ya ha pasado bajo el cabezal. En este caso la **latencia** es la peor posible, puesto que se debe esperar a que el disco **dé** la vuelta completa. La solución a este problema es numerar los sectores de forma intercalada a lo largo de una pista (*interleaving*). Esta estructura se muestra en la **figura 11.6**. Esta técnica ha quedado hoy en desuso debido a la alta velocidad que presentan las controladoras actuales.

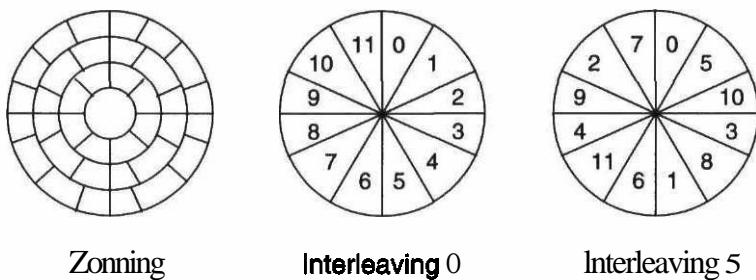


Figura 11.6: Formatos con interleaving y zoning

Antes de utilizar un disco es necesario formatearlo, tanto a bajo nivel como a alto nivel. El formato de bajo nivel es el proceso de creación de las pistas sectores y cilindros. Para definirlos sobre el disco se graban marcas de dirección, de datos, de sincronización y de información que identifica el controlador. Estas marcas servirán como plantilla para el almacenamiento posterior de datos. Cada pista contendrá información e identificación de los datos que están guardados (información sobre la pista y el sector que ocupan), así como información para sincronizar los circuitos de lectura (marcas y huecos). El formateo de alto nivel crea la organización **lógica** de los datos sobre el disco de acuerdo con los requisitos del sistema operativo. Hay que destacar que los discos flexibles no suelen estar formateados a bajo nivel al adquirirlos y es necesario darles formato antes de ser usados con alguna aplicación del sistema operativo. Por otro lado los discos duros sí suelen comercializarse formateados a bajo nivel, siendo necesario solamente el formato **lógico**.

Un ejemplo de formato a bajo nivel para discos flexibles es el **estándar** de IBM 3740 que permite estructuras de 125,256,512,1024 ó 2048 **bytes** por sec-

ter, teniendo **también** una cantidad variable de sectores por pista. Este formato se usa para una codificación de simple densidad.

En la figura 11.7 **se** muestra un ejemplo para una pista con **26** sectores y **128 bytes/sector**. Los números de la parte superior representan la longitud del campo en **bytes**.

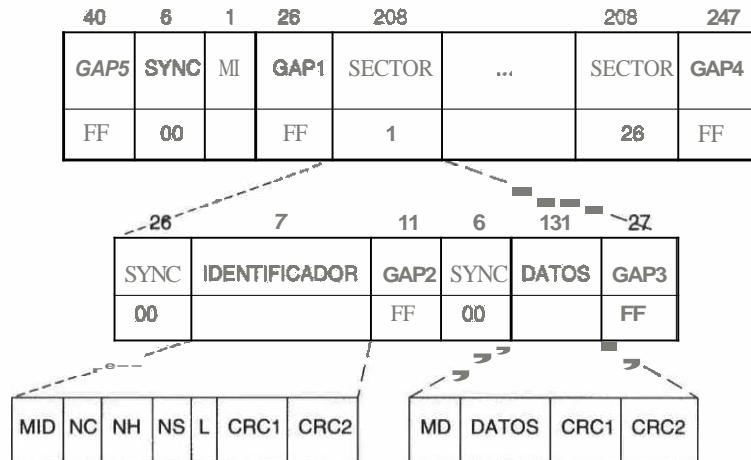


Figura 11.7: Formato IBM 3740

Cada pista comienza cuando se detecta la marca índice. Existen unos campos, **llamados** huecos, (*gaps*), que se usan para separar los campos de **información** de la pista y que compensan las tolerancias **mecánicas** y eléctricas que puedan existir en el posicionamiento de la cabeza y las **pequeñas** variaciones en la velocidad de rotación del disco. **También** permiten la actualización de un campo de **información** sin afectar a los campos adyacentes, ya que el inicio de cada **campo puede** variar ligeramente. En cada hueco se tiene un número **variable** de bytes puestos a uno, seguidos siempre por **6** bytes con ceros, que sirven para **sincronizar** el circuito separador de datos.

En cada sector hay:

- Un campo de **identificación**, que contiene una marca identificador, el **número** de cilindro, el **número** de cabeza, el número de sector, la longitud del sector y 2 bytes para guardar el CRC (*Cyclic Redundant Check*). El CRC es un **código** para detectar **errores** en los datos **almacenados** que se

obtiene como resultado de dividir los datos por un **polinomio** de un valor determinado $G(x)$, como por ejemplo $X^{16} + X^{12} + X^5 + 1$.

- Un campo de datos que contiene la marca de datos, 128 octetos de datos de **usuario** y 2 bytes para guardar el **CRC**.

En el disco existen cuatro tipos de octetos especiales llamados marcas. Las distintas marcas tienen patrones de bit únicos para distinguirlas de los datos:

- **MI** (marca índice - definido en hexadecimal como **FC**): Al principio de cada pista.
- **MID** (marca **identificador** - definido en hexadecimal como **FE**): En el primer octeto del campo identificador del sector. Indica si el sector es físicamente correcto o **está corrupto**.
- **MD** (marca de datos): En el primer octeto del campo de datos. Puede ser de **dos tipos**, **MD** (definido en hexadecimal como **FB**) y **MDB** (definido en hexadecimal como **F8**) indicando si los datos que vienen a continuación son válidos o si por el contrario han sido borrados.

Una vez se **formatea** un disco a bajo nivel, sólo es posible alterar el campo de datos y su **CRC** asociado. En cada sector se incluyen 2 bits de información en forma de **CRC**. Al leer los datos, se vuelve a calcular el **CRC** y se comprueba si es el mismo.

Una lectura típica de disco comprende varios pasos:

1. El controlador de disco busca la marca de dirección de identificación para encontrar la pista y el sector adecuado.
2. Se comprueba el **CRC** para comprobar que la información de identificación del sector es correcta.
3. Se busca la marca de datos para ver el número de bytes por bloque y se comprueba su **CRC**.
4. Se leen los datos.

Un **fallo** en cualquiera de los CRC iniciará una **rutina de comprobación de errores** que **reintentará** nuevas lecturas sobre el disco. Si se **vuelve** a fallar, la cabecera del sector se marca como errónea y el sector ya no **podrá** volver a ser utilizado.

11.5 CINTAS MAGNÉTICAS

La cintas magnéticas se caracterizan porque son dispositivos de acceso secuencial. Para encontrar un bloque en particular es necesario **rebobinar** o avanzar la cinta hasta su posición adecuada, teniendo un elevado tiempo de acceso, lo que limita su aplicabilidad. La principal ventaja es el bajo coste por bit de **información** almacenado. Decenas de **megabytes** de datos se pueden almacenar en una sola cinta, por lo que se utilizan para realizar copias de seguridad o intercambiar software entre sistemas. Las formas de acceso pueden ser:

1. Acceso *Stop-Start*, donde se leen o escriben los bloques individualmente. Para conseguir una tasa de transferencia adecuada, la cinta se debe mover **rápidamente** sobre los **cabezales**. Las limitaciones de inercia de las **rápidas** aceleraciones y **deceleraciones** se solucionan manteniendo un **buffer** de cinta entre los carretes y los **cabezales**.
2. Acceso continuo (*streaming*): Varios bloques se leen sucesivamente. En ellos no existen problemas de inercia, lo único que hay que hacer es **rebobinar** la cinta hasta la posición correcta cuando ésta se para.

Las cintas magnéticas funcionan de manera similar a las cintas de audio, **pero** deben operar mucho más rápido. Se necesitan tres características fundamentales:

1. Se requieren mecanismos especiales para tensar las cintas y prevenir que ésta se **atasque** o se rompa.
2. El mecanismo de arrastre debe ser muy preciso para poder situarse en el punto adecuado, puesto que los datos **están** muy juntos entre sí.
3. El mecanismo de arrastre debe conseguir poner rápidamente a la cinta a la velocidad máxima desde un estado de reposo.

El cabezal de escritura es de tipo **electromagnético** y el cabezal lector se monta sobre el cabezal de escritura (de hecho el mismo cabezal se puede usar para realizar las dos operaciones). Los datos **se** almacenan **en** pistas paralelas sobre la longitud de la cinta, **con** densidades que varían desde los 800 a los 6250 bits por pulgada.

A **continuación** se describen los **formatos** de cinta más populares:

Cintas con carretes

La cinta tiene media pulgada de ancho y un grosor de **200 μm** . El espesor de la cubierta **magnética** es de **100 μm** . Hay dos tipos básicos de cintas con carretes dependiendo de si se utilizan **mediante** acceso *Stop-Start* o continuo. En el primer caso, para obtener una velocidad de acceso aceptable, la cinta debe moverse, **acelerarse** y desacelerarse **rápidamente** sobre el cabezal. Debido a la inercia que presentan los carretes se mantienen **buffers** de cinta no tensada entre el carrete y el cabezal contenido en una columna de vacío. Cuando se produce una parada de la cinta el freno hace que la posición de la cinta sobre el cabezal no varíe debido a la inercia, y los **buffers** de cinta permiten un ligero desplazamiento de los carretes. Los **sensores ópticos** aplicados sobre los tubos de vacío **aseguran** que la longitud del **buffer** permanece constante. El esquema de este tipo de cinta se muestra en la figura 11.8.

En el segundo caso el problema de la inercia no es tan grave debido a que se accede a un **gran número** de bloques antes de que **sea** necesario parar la cinta. **Este** tipo de **cintas** son mucho más simples, ya que no es necesaria la columna de vacío ni los rodillos. **Los** datos se almacenan en una serie de **pistas** que transcurren paralelas a la cinta. Los primeros modelos utilizaban 9 pistas, pero ahora es más común tener 18 e incluso más. Con 9 pistas **en** el caso de lecturas **en paralelo** se utilizan 8 para datos y 1 para paridad, teniendo por tanto la necesidad de disponer de 9 cabezales. El problema de la lectura paralela es que la cinta debe tener el **ángulo** adecuado en todos los cabezales, para no producir una **desincronización** entre las **lecturas** de distintos cabezales. En la lectura **serie** todos los datos se guardan de **forma** secuencial en cada una de las pistas. En este caso la **grabación se realiza** en una **dirección** para las pistas pares y en la otra dirección para las impares, evitando tener que **rebobinar** la **cinta** para acceder a una pista consecutiva.

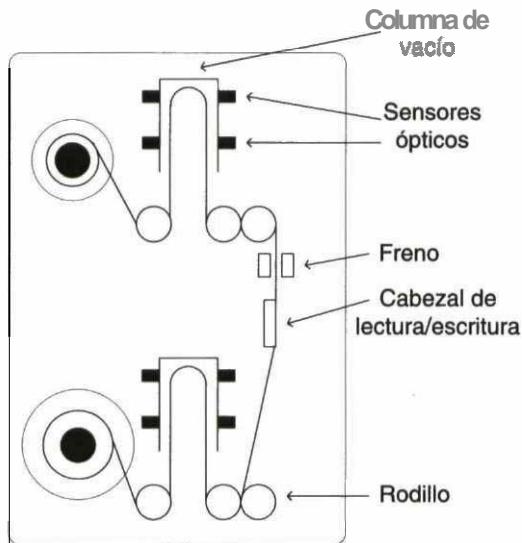


Figura 11.8: Cinta con carretes

Los códigos de grabación **más** usuales son el **NRZI** (se utiliza con paridad impar para asegurar que hay al menos una inversión en cada bit), el **NRZI Modificado** y el de grabación por codificación de fase (**PF**).

El formato de un bloque se muestra en la figura 11.9:

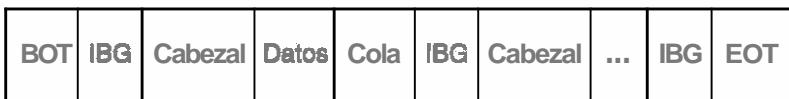


Figura 11.9: Formato de bloque

1. **BOT** y **EOT** son los campos de comienzo de cinta y de **fin** de cinta
2. En el cabezal existe información para identificar el bloque y el tipo del mismo
3. En la cola se guarda el CRC de los datos
4. **IBG (Interblock Gap)** mantiene la cinta en blanco para dar tiempo a calcular el CRC anterior, para no solapar bloques en el proceso de escritura y para poder detectar los bloques en el proceso de **rebobinado** o de avance.

Cartucho de cinta

Las cintas de ordenador fueron reemplazadas por los cartuchos, los cuales incluyen la cinta y los dos carretes, con lo que su uso es mucho más cómodo. Los *drivers* contienen cabezales separados de lectura y de escritura para poder leer inmediatamente lo que se está escribiendo y así poder detectar cuanto antes un posible error. Al cartucho de datos se le conoce como el cartucho de **1/4** de pulgada, y existen tamaños **estándar** de 5.25 y de 3.5 pulgadas pero que operan de la misma forma. La longitud de la cinta oscila entre los 200 y los **1000** pies, y tiene una anchura de **1/4** de pulgada. Para mover la cinta existe un cinturón de goma que está en contacto con la cinta y al mover el mecanismo accionador, el cinturón de goma mueve los carretes y desplaza la cinta. Este mecanismo es muy simple y produce una velocidad lineal constante independientemente de la cantidad de cinta que se encuentre en cada carrete. Para este tipo de cintas existen muchos estándares de códigos de grabación y de densidades de pista. El esquema se muestra en la figura 11.10.

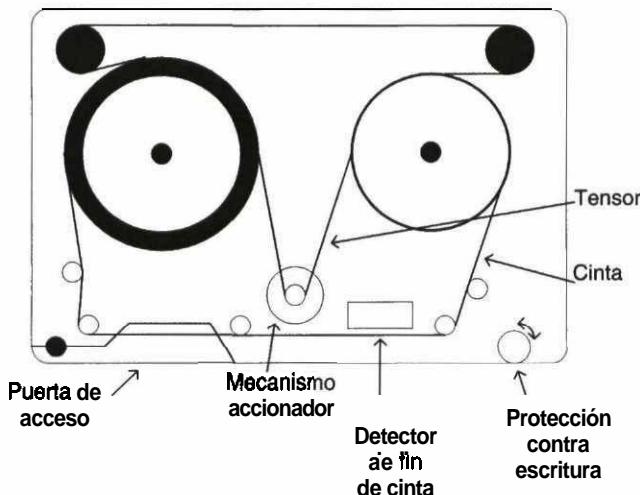


Figura 11.10: Cartucho de cinta

Cinta de vídeo de 8 mm

Con los métodos de grabación anteriores, para generar velocidades de acceso grandes es necesario pasar la cinta por delante del cabezal a gran velocidad. En el caso de las cintas de vídeo y las DAT, se considera un nuevo **método**

helicoidal de **lectura/escritura**, en el que la cinta se mueve lentamente y se **arrolla** en diagonal al cabezal que gira a una velocidad de unas **2000 rpm**. De esta forma, el cabezal se pone en contacto con una pequeña banda diagonal de la cinta de unos 5° produciendo **pequeñas pistas diagonales**. El cabezal giratorio al mover la cinta hace necesaria la **utilización** de al menos dos cabezales de **lectura/escritura** para que siempre haya alguno de ellos en contacto con la cinta. La capacidad oscila entre los **2** y los **5 Gb**. El cabezal se muestra con detalle en la figura 11.11.

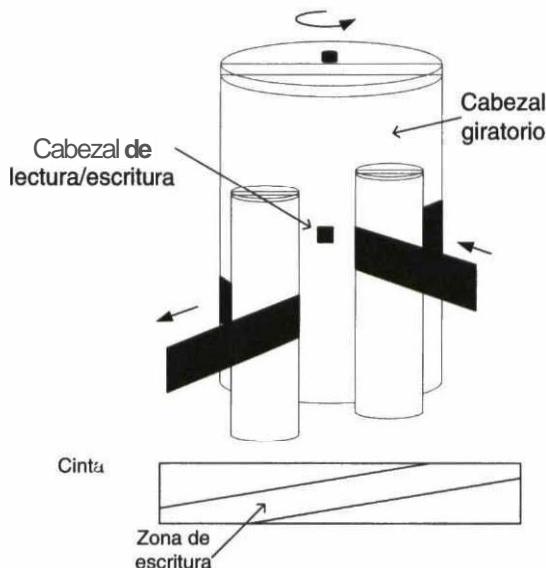


Figura 11.11: Cabezal de lectura

Cintas de audio digital

Más conocidas por cintas DAT (*Digital Audio Tape*). Aparecieron como soporte de audio con calidad digital. El formato es similar al anterior **pero** con un cartucho más **pequeño**, que se conoce como cinta de **4 mm**, aunque realmente es de **3'81 mm** de ancho. También utiliza un método de grabación helicoidal parecido al anterior, pero con un **ángulo** de contacto entre cinta y carrete menor. En la figura 11.12 se muestra el **método** de lectura de este tipo de cintas. El formato de grabación utilizado es el DDS (*Digital Data Storage*) definido por Sony y HP en 1988, con el cual se obtienen las siguientes capacidades:

1. DDS-1 2GB
2. DDS-2 4GB
3. DDS-3 12GB

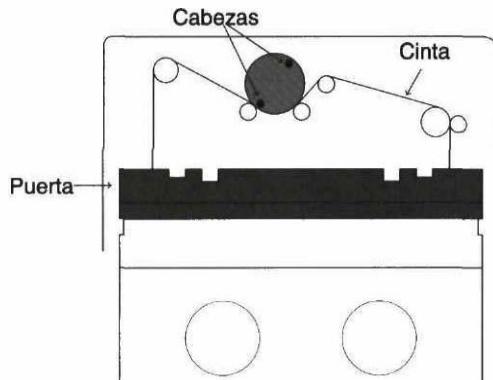


Figura 11.12: Cinta DAT

11.6 DISCO ÓPTICO

El disco óptico es un soporte de almacenamiento de datos que en su origen era de sólo lectura. Este formato apareció como una aplicación de los CD de audio al almacenamiento de datos. Como en el caso del CD de audio existe una sola pista en espiral sobre una de las caras, aunque el formato de grabación es distinto. Los discos ópticos utilizan como medio de codificación de los datos la reflexión de un haz de luz emitida por un diodo láser infrarrojo sobre una superficie óptica, como se muestra en la figura 11.13. La cabeza lectora en este caso está compuesta por un espejo y una lente, que **focaliza** el haz del láser sobre un determinado punto del disco. Durante el proceso de lectura el disco gira y la cabeza lectora se mueve radialmente, para acceder a las diferentes partes del disco.

Los datos digitales se escriben sobre el disco mediante la aplicación de pozos sobre el material. Cuando el láser se focaliza, mediante el espejo y la lente de la cabeza lectora, sobre un punto del disco, se refleja una cierta cantidad de luz. Esta cantidad de luz reflejada depende de si el punto donde incidió el

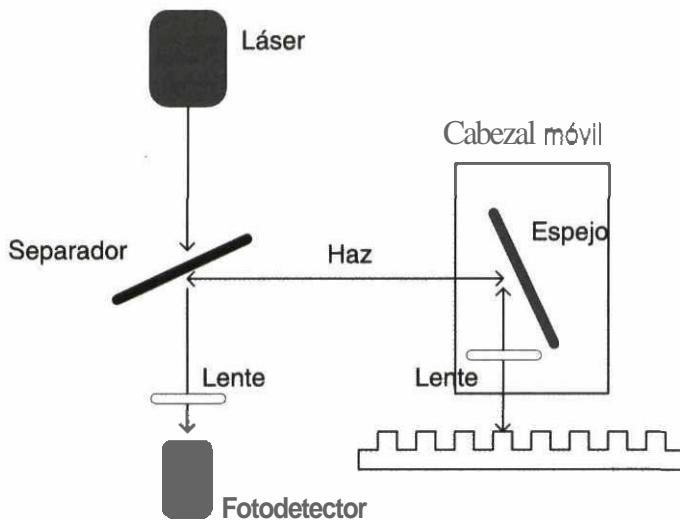


Figura 11.13: Esquema de funcionamiento del disco óptico

haz presenta un pozo o no, ya que los pozos reflejan el haz de luz de forma dispersa. A continuación, mediante una serie de espejos y lentes, se **focaliza** la luz reflejada hacia un **fotodetector**, que la transforma en una **señal** eléctrica. Como las **reflexiones** de luz **provenientes** de pozos presentan un gran dispersión del haz, la intensidad de luz reflejada recogida en el **fotodetector** **será** menor y la diferencia de **señal eléctrica** **permite** la **codificación** de 1's y 0's. Existen **dos** tipos principalmente de dispositivos **ópticos**: el CD-ROM y el DVD.

11.6.1 CD-ROM (*Compact Disc Read Only Memory*)

Fue el **primer** disco óptico comercial aplicado al almacenamiento de datos. En los primeros ejemplares de este tipo de dispositivos, y a diferencia de los discos magnéticos, el movimiento de giro del disco es variable. Con esta característica se pretende conseguir una velocidad lineal de lectura constante en todos los puntos, ya sean interiores o exteriores. Esta propiedad de los **CD-ROM** es una herencia del CD de audio, en los cuales se necesita conseguir una lectura de datos constante por unidad de tiempo sin que influya la posición de los datos en el disco. La velocidad del motor de giro se controla mediante un **microcontrolador**, dependiendo de la posición que ocupa la cabeza lectora, **sin-**

cronizándose la velocidad del motor con los datos recibidos. Así, los primeros CD-ROM funcionan a la misma velocidad que los CD de audio, con velocidades de rotación que varían entre 210 a 539 revoluciones por minuto (rpm), dependiendo de la localización del cabezal. Con estas velocidades de giro se consigue una tasa de transferencia de 150 **KB/s**. Posteriormente empezaron a aparecer lectores que multiplican la velocidad de rotación aumentando la tasa de transferencia. El problema de este tipo de lectores comenzó al superar velocidades multiplicadas por 12, ya que en el diseño de estos lectores mucho más rápidos, se trataba de mantener la velocidad lineal constante. Con velocidades de rotación variables en el rango de 210 rpm a 539 no se presentan graves problemas, pero el diseño se complica cuando es necesario variar la velocidad de rotación entre 5,040 a 12,936 rpm. Esta aceleración y desaceleración de la velocidad de rotación es uno de los factores que más degradan el acceso aleatorio a estos dispositivos, ya que no puede realizarse de forma **brusca**. Con el aumento gradual de la velocidad de los lectores, se ha tenido que volver al diseño de velocidades de transferencia variables, en los que la variación de la velocidad de rotación no es lo suficiente como para mantener una velocidad lineal constante. Actualmente, la velocidad de transferencia anunciada en los lectores indica la máxima velocidad que pueden alcanzar en las zonas externas del disco.

Mientras que en el nivel más bajo todos los CD graban la información de la misma manera, hay muchos tipos de datos distintos que pueden ser almacenados en un CD. Como ejemplo se tienen los CD de audio donde se almacenan bits y bytes, como en un CD-ROM, pero la forma de situar esta información en el disco es distinta. Esta manera de organizar los 1's y 0's en el disco es el formato. Estos **formatos** son equivalentes a las estructuras lógicas y sistemas de ficheros usados en los discos duros. Hay varios tipos distintos de **formatos** que se describen a continuación:

- Audio **digital** (**CD-DA (CD Digital Audio)**). Almacenan música con formato digital en **estéreo**, muestreada a 44.1 **Khz**. Cada muestra tiene 16 **bits**, por lo tanto cada segundo **estéreo** ocupa **(44,100*2*2) bytes**. Los datos se almacenan en bloques, llamados también sectores, que contienen 2,352 bytes de datos e información adicional sobre número de bytes usados para detección de errores. Son necesarios 75 bloques por segundo de sonido en un CD **estándar** de 74 minutos y la capacidad total del disco es de unos 747 **Mbytes**.

- CD de datos (CD-ROM, ISO 9660). El **estándar** original fue **desarrollado** por Philips y Sony en 1983, a **partir** del CD de audio. Pero esta **primera** especificación no **fue** lo suficientemente concreta como para evitar múltiples versiones, por lo que fue revisada y consensuada por un mayor **número** de fabricantes en 1985 bajo el nombre High Sierra *Format*, y finalizada poco **después** bajo el **estándar** ISO 9660. Bajo este **estándar** hay dos modos definidos:
 - Modo 1: Éste es el formato de datos más extendido y usado en los CD de datos. Están organizados de forma similar al CD de audio, con la **salvedad** de que de los 2,352 **bytes** de datos en cada bloque, 2,048 son para datos reales y el resto para códigos de detección y corrección de errores. Esto es debido a que la información **digital** debe protegerse mucho mejor contra posibles errores. Hay 75 bloques por segundo con 74 minutos de duración total, lo que da una capacidad aproximadamente de unos 650 **Mbytes**. Al comienzo del CD hay una tabla o índice de contenidos que muestra los datos almacenados en el disco, y ofrece información de dónde localizarlos.
 - Modo 2: Utilizan el mismo formato que en el Modo 1 pero se omite la **detección** y corrección de errores. Este disco se utiliza para almacenar datos que no son muy susceptibles a **errores**, como son los **gráficos** y el video. Además, diferentes clases de datos pueden mezclarse en el mismo disco. Ésta es la **razón** de que a menudo se le conozca como CD-ROM con arquitectura extendida (CD-ROM XA).
- CD-Interactivo (CD-I). En 1986, Philips y Sony de nuevo, crearon el **CD-interactivo**. Este nuevo concepto de CD-ROM, no muy utilizado, pretendía convertirse en un **estándar** multimedia en el que se pudieran crear discos que incluyeran texto, gráficos, audio, vídeo y programas. Para **ello**, se basaba en el **estándar** CD-ROM XA, pero con un formato algo diferente.
- Video CD (VCD). Utiliza el **estándar** de compresión MPEG-1 para almacenar vídeo sobre el mismo formato utilizado por los CD de audio, almacenando unos 74 minutos de vídeo. El problema es su poca calidad debido a que MPEG-1 no proporciona **unas** buenas prestaciones.
- Foto CD. Desarrollado a principios de 1990 por Kodak y Philips, es una extensión del CD-ROM XA, para almacenar imágenes fotográficas.

Como ya se ha mencionado, los CD-ROM son dispositivos de sólo lectura. Pero a partir de ellos, se han **desarrollado** nuevos soportes compatibles, llamados CD-R y CD-RW, que sí permiten la grabación. El CD-R (**CD-Recordable**) es un CD grabable una única vez. En su fabricación se utiliza un disco reflectante muy delgado, normalmente de oro o plata, cubierto de una capa de tinte inicialmente transparente. Para realizar los pozos sobre esta capa se utiliza un láser que deforma el tinte, haciéndolo más oscuro y menos reflectivo. El proceso de grabación mediante láser resulta muy lento y poco efectivo para realizar múltiples copias de un mismo CD. La lectura es similar a la de un CD-ROM.

El CD-RW (**CD-ReWritable**) es un CD regrabable unas mil veces. Este dispositivo utiliza la tecnología de cambio de fase para variar sus propiedades reflectante~. El disco está cubierto por un material cristalino que refleja la luz, pero mediante un láser de cierta intensidad la estructura del material cambia a estado amorfo no reflectivo. Un láser de intensidad media devuelve la estructura cristalina y para la lectura se usa un haz de baja intensidad.

11.6.2 DVD (*Digital Versatile Disc*)

Este dispositivo de más reciente aparición en el mercado intenta servir de soporte universal para el almacenamiento de información informática y datos multimedia, reemplazando al CD de **audio**, los soportes de vídeo juegos, cintas de vídeo, disco láser de vídeo y CD-ROM.

El proceso de lectura es básicamente el mismo que en el caso de los **CD-ROM**, tiene las mismas dimensiones y es también de sólo lectura. Las mayores diferencias son una capacidad de almacenamiento y una velocidad de transferencia muy superiores. También presenta **formatos** de grabación distintos y la utilización de **las** dos caras para almacenar información. El incremento de la capacidad se basa en la reducción del tamaño de los pozos y el espaciado entre pistas. Para conseguir una mayor resolución en la detección se usa un láser con una longitud de onda menor (**6351650 nm**), en comparación con el láser infrarrojo de (**780 nm**) utilizado en el CD-ROM. Además, existe la posibilidad de que en una misma cara se sitúen dos capas de información separadas por un material semi-reflectivo. El lector puede acceder a cada una de ellas focalizando a diferentes profundidades.

Atendiendo al formato es posible encontrar **tres tipos de dispositivos DVD**: DVD-video, **DVD-audio** y **DVD-ROM**. El primer y el segundo tipo fueron desarrollados buscando un **sostén de video** y audio de alta calidad **respectivamente**, con especificaciones provenientes de la industria **cinematográfica y discográfica**. Sin embargo, el tercer tipo fue impulsado por la industria **informática** buscando un formato más rápido y de mayor capacidad que el CD-ROM. El DVD-ROM en sus **especificaciones engloba** al **DVD-video** y al **DVD-audio**, siendo un lector DVD-ROM capaz de leer todos los formatos, mientras que no sucede lo mismo de forma contraria. Dentro de la familia DVD-ROM, es posible encontrar, al igual que sucede **con** el CD-ROM, versiones grabables, una sola vez **DVD-R** o múltiples veces **DVD-RAM**. Las características básicas de cada uno de estos **formatos** son:

- El **DVD-video** es capaz de almacenar unos **133 minutos** de **vídeo comprimido** en **MPEG-2** por cada cara, además de **5 canales** de sonido (**Digital Dolby Surround**).
- El **DVD-audio** tiene un formato de **música** en **PCM (Pulse Code Modulation)** con **96 KHz** de frecuencia de **muestreo** y 24 bits de **resolución** superando el actual CD de audio con **44.1 kHz** y **16 bits**.
- El **DVD-ROM** presenta las siguientes capacidades:
 - **DVD-5: 4.7 GB Una cara/Una capa.**
 - **DVD-9: 8.5 GB Dos caras/Dos capas.**
 - **DVD-10: 9.4 GB Dos caras/Una capa.**
 - **DVD-18: 17 GB Dos caras/Dos capas.**

El almacenamiento óptico presenta una serie de ventajas y desventajas frente a las grabaciones **magnéticas**. Dentro de las ventajas podemos **destacar** la gran robustez, pues se trata de dispositivos que no se ven alterados por campos magnéticos y la superficie de lectura está protegida por materiales plásticos transparentes que evitan su **deterioro**. No presenta la **degradación** de los discos flexibles producida por el contacto con el cabezal, al ser un proceso de lectura **óptico**. No se deteriora con la presencia de impurezas, ya que el **cabezal** no está tan **próximo** a la **superficie** como en el caso de los discos duros. Aunque estas impurezas pueden provocar errores de **lectura**. Entre las desventajas es posible **encontrar tambien una** capacidad de almacenamiento y una velocidad de acceso **inferior** a la de los **discos duros magnéticos**.

11.7 DISCO MAGNETO-ÓPTICO

Las unidades de disco magneto-ópticas leen y escriben la información sobre el disco mediante una combinación de principios ópticos y magnéticos. Utilizan láser para leer, mientras que utilizan campos magnéticos combinados con la acción del láser para escribir la información. La combinación de ambos principios para escribir se realiza aplicando una cabeza magnética en una cara del disco y simultáneamente un láser sobre la cara **opuesta**. El material **posee** sobre la **superficie** dominios magnéticos que se alinean verticalmente. El campo magnético de los dominios del material decrece en intensidad con un aumento de la **temperatura**. Al alcanzar la temperatura del material un cierto nivel, éste pierde todas sus propiedades magnéticas. Esta temperatura es cercana a los 300°C y se denomina temperatura de Curie. La utilización de un láser permite alcanzar esta temperatura sobre una **pequeña** área del material. Este dominio **desmagnetizado** durante el proceso de enfriamiento se ve afectado por un campo magnético externo, que induce una nueva polarización del campo magnético del dominio. De esta forma el láser controla el tamaño y la zona del material, mientras que el campo externo controla la polaridad inducida. Una vez almacenada la información, ésta no se verá afectada por campos externos a temperatura ambiente. Un esquema del proceso de escritura se muestra en la figura 11.14.

En la lectura se usa un láser de menor potencia que hace que el material no alcance temperaturas elevadas y no varíe la magnetización. El haz del láser incide sobre el material y es reflejado. Esta luz reflejada tiene un ángulo de polarización, modificado dependiendo de la dirección de la magnetización del dominio. Esta propiedad se denomina efecto **Keer**, y diferencia los dos posibles ángulos de polarización que aparecen al reflejar la luz, en los dos tipos de sentido de magnetización.

Estos discos presentan la ventaja de que pueden ser regrabados varios millones de veces, a diferencia de los discos ópticos tradicionales, y alcanzan tiempos de acceso inferiores a **25ms**, con capacidades de hasta **5Gb** con discos de tamaño **5.25"** y **640Mb** para la **serie** de **3.5"**.

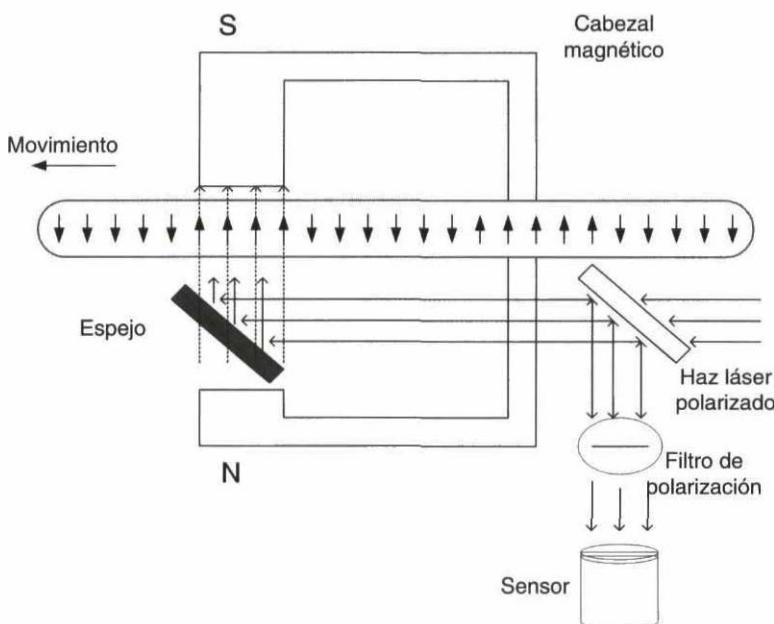


Figura 11.14: Escritura de un dispositivo magneto-óptico

11.8 CONCLUSIONES

Los dispositivos de **almacenamiento** secundarios permiten **almacenar** programas y datos de forma no **volátil**. Los dispositivos **más** comunes **actualmente**, utilizan métodos ópticos o magnéticos para almacenar la información. Los **métodos** magnéticos **más** utilizados son los discos, tanto flexibles como duros, y las cintas. Estos dispositivos se basan en detección de la orientación del campo magnético registrada sobre pequeñas celdas de bit del material. Los **dispositivos** ópticos como los **CD-ROM** y **DVD** utilizan la **reflexión** de un haz de luz sobre el disco para detectar la **información**. Los dispositivos más utilizados **actualmente** como almacenamiento secundario son los discos duros, ya que presentan una capacidad y velocidad de acceso mayores. Los dispositivos ópticos, por otro lado, presentan la ventaja de que son un método **más** robusto al no ser afectados por **campos** magnéticos extremos, y su tiempo de vida es mayor, por lo que suelen ser ampliamente utilizados para **realizar** copias de seguridad y como formato para la **distribución** de software.

11.9 CUESTIONES

11.1 *¿Qué problema presenta la excesiva reducción del tamaño de las celdas de bit a la hora de leer la información magnética?*

11.2 *¿Qué ventajas tienen los códigos autoreloj?*

11.3 *Codificar mediante los diversos códigos vistos en el capítulo la secuencia de 1's y 0's: 101110010100.*

11.4 *¿Para qué sirve el formato a bajo nivel y a alto nivel de los discos magnéticos?*

11.5 *¿Qué utilidad presenta el método de zonning en los discos duros?*

11.6 *Enumerar las ventajas y desventajas de usar cintas magnéticas como método de almacenamiento secundario.*

11.7 *¿Por qué la velocidad de rotación en los CD-ROM es variable?*

11.8 *Enumerar las ventajas y desventajas de los discos ópticos frente a los magnéticos.*

11.9 *¿Para qué utilizan los discos magneto-dípticos el láser en la grabación de datos?*

11.10 BIBLIOGRAFÍA

- **Computer Peripherals.** Barry M. Cook, Neil H. White. 3^a ed. Edward Arnold. ISBN 0-340-60658-4. 1995.
- **The Indispensable PC Hardware Book.** Hans-Peter Messmer. 3^a ed. Addison Wesley Longman, Inc. 1997. ISBN 0-201-40399-4.

CAPÍTULO 12

ARQUITECTURAS AVANZADAS

12.1 INTRODUCCIÓN

En los capítulos anteriores se ha descrito la estructura de los computadores y periféricos más utilizados. Las tecnologías implicadas en el diseño de estos sistemas van mejorando muy rápidamente y **hacen obsoletos** los computadores y **periféricos** en plazos muy cortos. Sin embargo, la estructura básica del computador personal sigue el modelo de máquina Von Neumann de búsqueda y ejecución secuencial de instrucciones.

A partir del modelo básico Von Neumann se pueden realizar modificaciones que, siguiendo básicamente el modelo de computación de una máquina secuencial, mejoren el rendimiento significativamente. La mejora más significativa consiste en dividir la ejecución de instrucciones en etapas más sencillas que se ejecutan de forma encadenada. De esta manera se construyen los cauces **segmentados**, aproximación similar a una cadena de montaje, donde la ejecución de una instrucción está dividida en etapas.

Es sencillo darse cuenta de que el modelo de búsqueda y ejecución secuencial de instrucciones Von Neumann puede ser poco eficiente, a pesar de la utilización de los cauces **segmentados**. Hay tareas con un inherente grado de paralelismo que podrían realizarse **más** rápidamente. De hecho las **CPUs** de los computadores personales modernos incorporan algún grado de paralelismo aunque sigan globalmente el esquema de la máquina Von Neumann original.

La **forma** más clara de implementar paralelismo consiste en que varios elementos de proceso (**también** llamados **EPs**) efectúen simultáneamente las tareas que inicialmente realizaba un solo procesador. Inicialmente cabe pensar que se acelerará la ejecución de las instrucciones, ya que si con un procesador se realizan X cálculos por unidad de tiempo, con 2 procesadores o **EPs** se realizarán $2X$ cálculos. Este razonamiento es claramente falso, ya que las operaciones en paralelo con los datos no son siempre independientes. Hay veces en que la ejecución secuencial es necesaria al existir una dependencia de datos. De la misma manera puede ocurrir que en función del resultado de ciertas operaciones se tenga que ejecutar una parte del **código** u otra, con lo que no es posible predecir qué partes del programa se van a ejecutar. Adicionalmente el resultado de las diversas operaciones debe comunicarse entre los diversos **EPs**, lo cual va a consumir un tiempo que dependerá de la distribución de los **EPs** en una red de **interconexión**, además de los retrasos introducidos por la propia red. La mejora en el rendimiento no es transparente al usuario, ya que éste debe conocer las características de la red para paralelizar el código de manera eficiente.

Por último cabe destacar **cómo** la aproximación de simplemente **replicar** los elementos de proceso no es la más adecuada. Más **EPs** significa **más** coste y el rendimiento no tiene por qué aumentar de manera proporcional a la inversión realizada.

En este capítulo se van a presentar computadores que, utilizando de alguna manera **aproximaciones paralelas**, son más **eficientes que** la simple **máquina Von Neumann**. En primer lugar se presentará la segmentación como una aproximación que, **siguiendo la misma filosofía** de **búsqueda** y ejecución **secuencial** que la máquina **Von Neumann**, aumenta el numero de instrucciones ejecutadas por segundo. A continuación se introducirá una clasificación de los computadores que utilizan las denominadas **Arquitecturas Avanzadas**. Posteriormente **se realizará** una descripción de las **características** principales de estas arquitecturas y se introducirá el concepto de aceleración o speedup, concepto útil para evaluar el rendimiento en computadores avanzados.

12.2 SEGMENTACIÓN

La segmentación consiste en dividir la ejecución de una tarea en etapas. De esta manera, una tarea se completa cuando se han realizado todas las etapas en las que se **subdivide** de manera ordenada.

Cuando se segmenta una tarea (es decir, se divide una tarea en etapas) se realiza un **cauce segmentado**. Es importante que cada etapa tenga **capacidad** de almacenamiento de los **resultados** parciales producidos en ella. Con estos requisitos el cauce segmentado puede **simultáneamente** realizar varias tareas, ya **que** cada etapa podrá funcionar en paralelo almacenando sus resultados parciales. **Tías** la finalización de cada etapa se transmitirá el resultado a **la** etapa siguiente para así poder completar la tarea.

Existen varios tipos de segmentación. Cuando la **ejecución de instrucciones** se divide en diversas etapas se denomina **segmentación de instrucciones** o **cauce segmentado de instrucciones**. De manera análoga hay operaciones aritméticas que son divisibles en etapas, almacenando el resultado de las operaciones parciales. Cuando se segmentan las unidades funcionales de la unidad aritmética, se realiza una **segmentación aritmética** o **cauce segmentado de datos**.

La ejecución de cada instrucción se puede dividir en una **serie** de etapas especializadas, de manera que la suma de la ejecución de todas las etapas da como resultado la ejecución de la instrucción. La instrucción va atravesando el cauce de instrucciones y en cada etapa se van haciendo las operaciones necesarias. Claramente no todas las instrucciones utilizan **todas** las etapas, pero en cualquier caso todas las instrucciones atraviesan el cauce. Si una **instrucción** no necesita realizar las operaciones implementadas en una etapa, simplemente atraviesa esa etapa **sin** realizarse ninguna operación.

En una máquina Von **Neumann** tradicional, hasta que no se termina de forma completa la ejecución de una instrucción no se inicia la ejecución de la **siguiente**. Un ejemplo de **procesador** con diversas etapas se muestra en la figura 12.1. En la figura 12.1 (A) se observa la situación de la instrucción *I2* e a **T=7** y en la figura 12.1 (B) se muestra la evolución temporal de las instrucciones *I1* y *I2* por las diversas etapas del procesador secuencial. En este caso sólo se está ejecutando

una **instrucción** cada vez en el procesador, y hasta que no se acaba de ejecutar de forma completa una **instrucción**, no se ejecuta la siguiente.

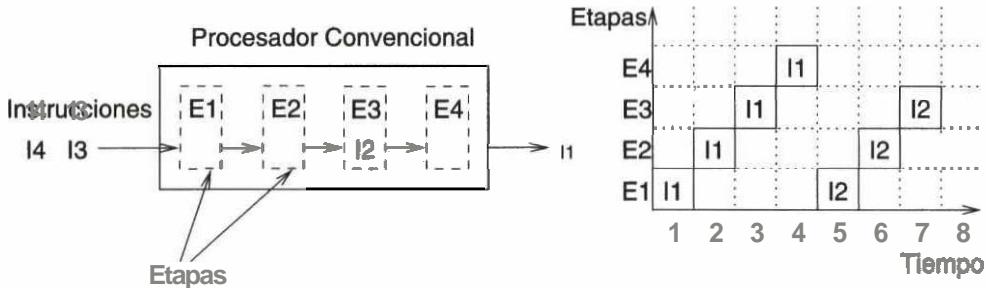


Figura 12.1: (A) Esquema de un procesador secuencial (B) Evolución temporal

En el caso de que las etapas estén bien **definidas** y se pueda **almacenar** información entre ellas, se pueden empezar otras instrucciones sin haber terminado de ejecutar una **instrucción**. De hecho, en **cada** etapa se podría estar ejecutando una instrucción en el caso ideal, llegándose a la máxima **paralelización** posible. Esta arquitectura es la que se conoce como cauce **segmentado** de instrucciones, y se utiliza de forma masiva en cualquier procesador moderno. Un ejemplo de esta arquitectura se puede **observar** en la figura 12.2 (A), y la evolución temporal a la que daría paso en la figura 12.2 (B).

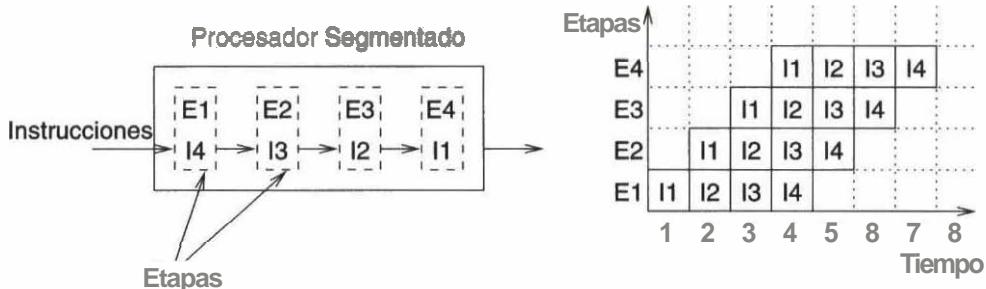


Figura 12.2: (A) Cauce segmentado de instrucciones (B) Evolución temporal

De hecho puede darse el caso de que coexistan **varios** cauces especializados en el mismo procesador (por ejemplo, operaciones enteras o en coma flotante). Una vez que se decodifica la **instrucción** en función del tipo de instrucción se envía al cauce adecuado.

La división en etapas de la unidad de ejecución de instrucciones puede acelerar de forma global la ejecución de instrucciones. Los problemas que plantea la segmentación de instrucciones son fácilmente identificables. Puede ocurrir que en un programa una instrucción en una etapa necesite datos todavía no disponibles, ya que la instrucción anterior no los ha generado todavía (no han salido del cauce). Esto ocurriría, por ejemplo, en la figura 12.2 si la instrucción I3 necesita datos modificados por la instrucción I2. Este problema no existe con la aproximación secuencial de la figura 12.1. Para evitar este problema en los cauces segmentados deben existir mecanismos que avancen los datos entre etapas y detecten este tipo de inconsistencias, paralizando el cauce si es necesario. Este tipo de problemas en el diseño de cauces segmentados se conocen como **riesgos** y se pueden agrupar en 3 tipos.

1. **Riesgo por dependencia de datos:** Es el tipo de dependencia de datos descrito anteriormente. Aparece cuando los datos necesarios para ejecutar una instrucción dependen de una instrucción anterior. De esta manera, ambas instrucciones podrían estar en el cauce de manera solapada y la segunda instrucción tomar el valor de los datos antes de que fueran modificados por la primera instrucción. Es necesario por tanto detener el cauce hasta que los datos correctos estén disponibles, adelantándolos **internamente** entre etapas.
2. **Riesgo estructural:** Surge cuando es necesaria una misma unidad funcional del cauce para ejecutar dos instrucciones distintas. Hasta que no se libere la unidad, acabando de ejecutar una etapa de la instrucción, no es posible que la ocupe otra instrucción. El cauce se detiene hasta que desaparece este riesgo. Un ejemplo sería una CPU con una sola unidad de división en coma flotante, operación costosa que requiere más ciclos de reloj que otras etapas. Las instrucciones atravesarán el cauce segmentado y en el caso de que se tenga que realizar una división, ésta será enviada a la unidad especializada de división en coma flotante. Si las instrucciones que van a continuación no dependen del resultado de la división éstas se podrán ejecutar en paralelo en otras unidades del cauce segmentado, finalizando incluso antes que la división. En el caso de que las instrucciones posteriores sean también divisiones en punto flotante, el cauce también se paralizará, no porque haya dependencia de datos, sino porque hay una sola unidad de división en punto flotante y **ésta** está ocupada.

3. **Riesgo de control:** Estos riesgos aparecen **cuando** se va a ejecutar una instrucción que modifica el contador de programa (por ejemplo, en las instrucciones de salto). Puede ocurrir que no se sepa **la dirección del salto** hasta que no se ejecuten completamente instrucciones contenidas en el cauce. De esta manera, cuando entra en el cauce una instrucción de salto, no se permite que entren más instrucciones o se ejecutan condicionalmente hasta que se comprueba si el salto ha sido correcto. Existen **CPU**s que disponen de unidades de predicción de saltos que aumentan el rendimiento evitando las detenciones en un alto porcentaje de casos.

Es posible **también** clasificar las arquitecturas segmentadas en función de si pueden realizar **sólo** una o más de una **operación** distinta. En el caso de **que** el cauce segmentado sólo realice una **tarea** cada vez se denomina cauce **estático**. Si por el contrario se puede realizar más de una tarea simultáneamente se denomina cauce **dinámico**. Este segundo tipo de cauce segmentado cambia su funcionalidad cuando se puede seleccionar el **camino** de los datos dentro del cauce. La operación realizada sobre los datos será distinta, dependiendo de las etapas que hayan atravesado.

Un ejemplo de este tipo de cauces podría ser un cauce de 3 etapas **A, B y C**. Si una dato atraviesa las etapas **A, B y C** se realiza una operación sobre ellos. Si por el contrario los datos pasan de la etapa **A** a la **C** directamente, la transformación final será distinta al no haber pasado por la etapa **B**. En este caso los datos a los que se aplica la transformación **A-B-C** tardan más en salir del cauce que los datos transformados por las etapas **A-C**.

En el caso de los cauces dinámicos se va a disponer de datos que no van a **recorrer** de forma completa y secuencial el cauce segmentado. Es por tanto importante evitar que se produzca la **confluencia** en una misma etapa de datos que no han atravesado las mismas etapas, y por tanto no van a la misma velocidad, caso denominado **colisión**. El diseño de la unidad de control de los cauces segmentados debe garantizar que se evitan las colisiones entre diferentes datos.

La eficiencia de un cauce segmentado es transparente al usuario, el cual no debe **utilizar** ninguna **técnica** de programación especial **del** procesador. El **compilador** se encargará de generar un código eficiente que produzca la menor cantidad posible de detenciones en el cauce segmentado debido a los riesgos.

12.3 CLASIFICACIÓN DE LAS ARQUITECTURAS AVANZADAS

La segmentación hace aumentar el rendimiento de los computadores, aunque en el fondo se sigue la misma estrategia de **búsqueda/ejecución** secuencial. Los computadores modernos de grandes prestaciones utilizan diversas técnicas de procesamiento paralelo, lo cual ha dado lugar a una clasificación de las estrategias seguidas en estos computadores. La **primera** clasificación ampliamente aceptada fue la clasificación de Flynn, que ha sido utilizada durante más de 30 años. Sin embargo, el rápido desarrollo de las tecnologías ha hecho borrosas las fronteras entre los grupos definidos por Flynn. De la misma manera alguno de los **grupos** de la clasificación de Flynn no está claramente definido y varios autores **discrepan** sobre la catalogación de un grupo de máquinas en una clase u otra.

Estas dificultades para encajar algunas de las máquinas existentes en la clasificación de Flynn ha propiciado la aparición de otras clasificaciones. Un ejemplo es la presentada en Zargham en su libro *Computer Architecture. Single and Parallel Systems*.

12.3.1 Clasificación de Flynn

Las ideas presentadas en los capítulos anteriores reflejan el modelo más simple de arquitectura de un computador. En todo computador se puede distinguir entre el flujo de instrucciones, que es la secuencia de instrucciones que va ejecutando un procesador, y el flujo de datos a procesar.

En 1966 M. J. Flynn definió una clasificación de las arquitecturas de los computadores en función de cómo se procesaba el flujo de instrucciones y el flujo de datos:

- **Arquitectura SISD (Single Instruction Single Data):** Son las arquitecturas en las que hay un solo flujo de instrucciones que se van ejecutando de manera **secuencial**, y van procesando un único flujo de datos tal como se muestra en la figura 12.3 a). La máquina Von Neumann puede considerarse perteneciente a este grupo.

- Arquitectura **SIMD** (*Single Instruction Multiple Data*): En este caso el **flujo** de instrucciones es único, existiendo varios flujos de datos. Un **ejemplo** de esta arquitectura puede ser el de las **máquinas matriciales** en **las que hay una sola unidad de control Y múltiples unidades de datos**, **tal como se muestra en la figura 12.3 b)**. Todas **las** unidades de proceso ejecutan la misma instrucción, siendo el flujo de datos múltiple.
- Arquitectura **MISD** (*Multiple Instruction Single Data*): En esta arquitectura se realizan múltiples instrucciones simultáneamente sobre el mismo flujo de datos, **tal como se muestra en la figura 12.3 c)**. Varios autores **consideran** como impracticable una máquina de este tipo. Sin embargo, otros autores incluyen **en este grupo** las máquinas con encauzamiento de datos, como son las **vectoriales** y las **sistólicas**, máquinas que se describirán en la clasificación de Zargham.
- Arquitectura **MIMD** (*Multiple Instruction Multiple Data*): Es el caso de un sistema con un flujo de **instrucciones** y datos múltiple, tal como se muestra en la figura 12.3 d). Ejemplo **de** esta arquitectura son los **multi-procesadores**, donde cada **procesador** tiene una fuente de instrucciones y una fuente de datos. La comunicación entre estos elementos de proceso y el flujo de datos **dependerá** de la topología de interconexión entre los mismos.

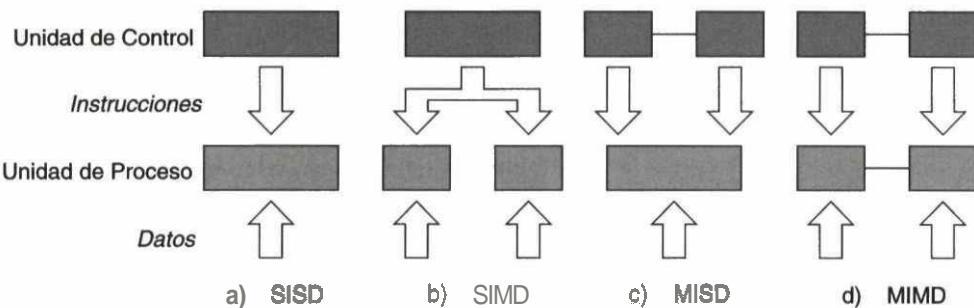


Figura 12.3: Clasificación de Flynn

12.3.2 Clasificación de Zargham

La clasificación de Flynn ha demostrado ser útil durante más de 30 años. A pesar de esto, los **rápidos** avances en tecnología de computadores han dado lugar

a arquitecturas que no pueden ser clasificadas de una manera clara en ninguno de los grupos de Flynn. Zargham propone una clasificación de las arquitecturas paralelas más fina, heredada de los grupos de Flynn. Algunos de estos grupos de máquinas se describirán con más detalle en las siguientes secciones.

1. Arquitecturas MIMD

- **Multiprocesadores:** Un multiprocesador es una máquina paralela formada por varios **microprocesadores** que comparten memoria y otros recursos. La figura 12.4 a) muestra un esquema de la **estructura** de un multiprocesador, que se **describe** con más detalle en la sección 12.4.
- **Multicomputadores:** La estructura es similar a la de los **multiprocesadores**, salvo que en este caso la memoria es local a cada procesador, tal como se muestra en la figura 12.4 b). Se describe este tipo de máquinas con más detalle en la sección 12.4.
- **Multi-multiprocesadores:** Esta arquitectura consiste en un **multicomputador** en el que cada nodo es un multiprocesador.

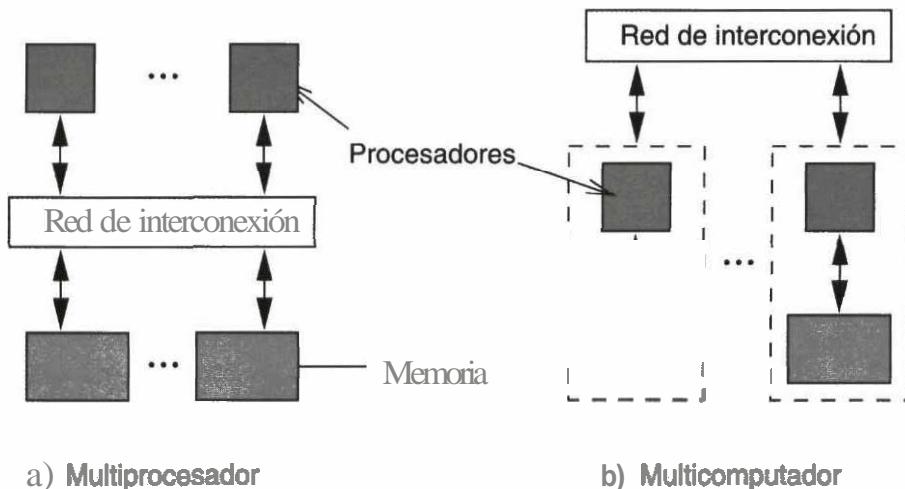


Figura 12.4: Esquema de un Multiprocesador y un Multicomputador

- **Máquinas de flujo de datos:** En este tipo de máquinas una **instrucción se ejecuta** cuando los datos necesarios para la **instrucción** están listos. La disponibilidad de los datos se consigue canalizandolos resultados de la ejecución de instrucciones anteriores hacia las instrucciones que esperan su ejecución. Las **instrucciones** de una máquina

de flujo de datos son **autocontenidas**, ya que ellas **mismas** contienen las variables y operandos en vez de **direccionar** una memoria global. Este comportamiento permite un alto grado de concurrencia, ya que la ejecución de una **instrucción** no afecta a otras que **estén** listas para ser ejecutadas. La figura 12.5 muestra la estructura **genérica** de una máquina de flujo de datos. **Las** instrucciones y los datos se almacenan de **forma** conjunta en memoria. Cuando una **instrucción** **está** lista para ser ejecutada se envía a un EP. Cuando esta **instrucción** se ha ejecutado los resultados se distribuyen hacia las instrucciones que los necesitan para que puedan ser ejecutadas.

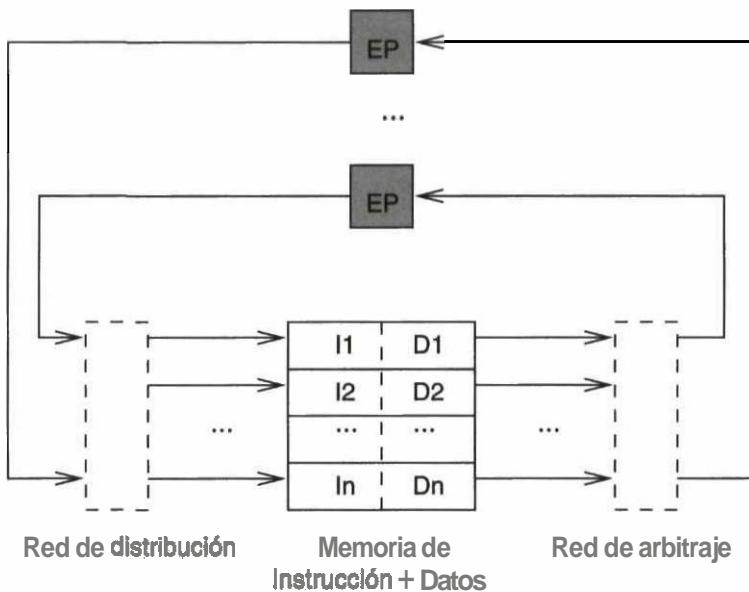


Figura 12.5: Esquema de una máquina de flujo de datos

2. Arquitecturas SIMD:

- **Procesadores matriciales:** La figura 12.6 representa la estructura **genérica** de un **procesador matricial**. Estos procesadores están formados por un **procesador escalar** (procesado; **secuencial**) y un conjunto de **procesadores idénticos** dotados con memoria local. Cuando la unidad de control **decodifica** la **instrucción**, la envía al **procesador escalar** o bien a todos los **procesadores simultáneamente**, dependiendo del tipo de instrucción. Si la instrucción es **matricial** se procesan múltiples datos simultáneamente.

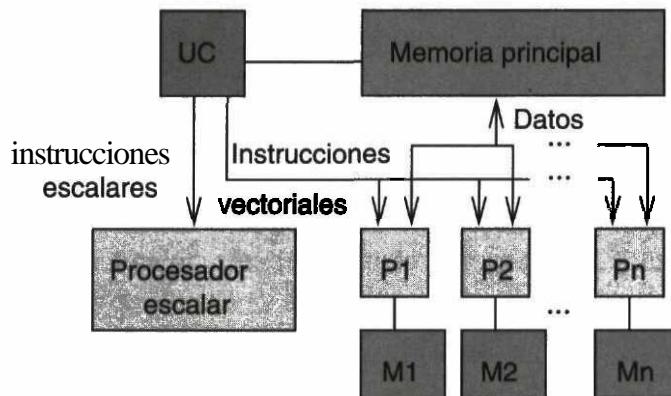


Figura 12.6: Esquema de un procesador matricial

3. Arquitecturas MISD

- **Procesadores vectoriales:** Un procesador vectorial es una máquina optimizada para la ejecución de instrucciones que involucren la utilización de **vectores**. Normalmente consta de una unidad escalar **segmentada** más una unidad vectorial segmentada que **optimiza** el tiempo de ejecución en el caso de operaciones sistemáticas con **vectores**. La unidad de control decodifica las instrucciones y las envía al cauce escalar o vectorial en **función** del tipo de **instrucción**. La figura 12.7 muestra la estructura genérica de un procesador vectorial.

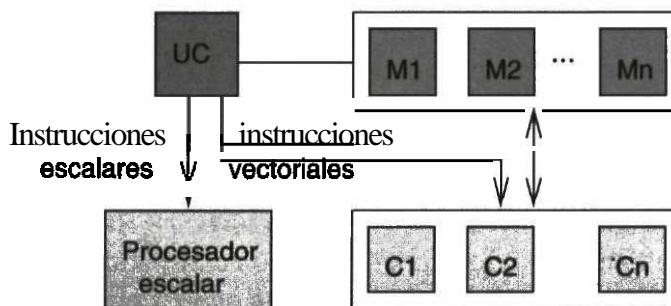


Figura 12.7: Esquema de un procesador vectorial

- **Matrices sistólicas:** En este tipo de arquitectura se tiene un número de EPs dispuestos en un cauce segmentado de datos (por ejemplo, lineal o bidimensional, tal como se muestra en la figura 12.8). Cada EP dispone de una **pequeña** capacidad de almacenamiento y habitualmente **sólo** se comunica con sus vecinos, realizando todos los

EPs la misma operación simultáneamente. Sólo los EPs de los bordes de la matriz pueden realizar funciones distintas de E/S. Los datos fluyen de forma síncrona a través del cauce, obteniéndose los resultados finales cuando los datos han atravesado completamente el cauce. Estas arquitecturas son extremadamente regulares y adecuadas para su implementación en circuitos de muy alta escala de integración. Son especialmente útiles en problemas muy especializados que involucren vectores, matrices y cálculos de manera recursiva. Su principal desventaja es su falta de flexibilidad.

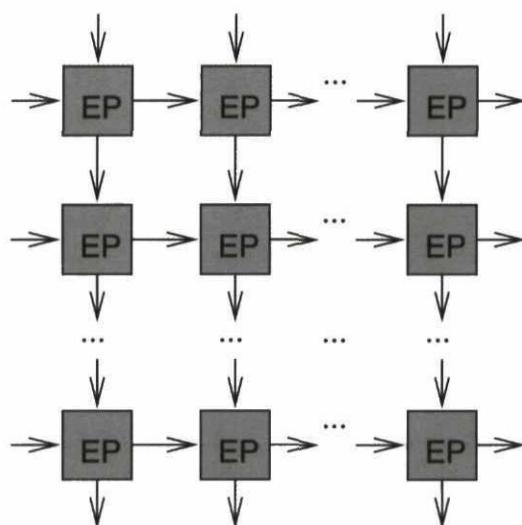


Figura 12.8: Matriz sistólica bidimensional

4. Arquitecturas híbridas

- Máquinas **MIMD-SIMD** y **MIMD-MISD**: El paralelismo tiene dos vertientes básicas: paralelismo de datos y paralelismo de instrucciones. Es posible realizar máquinas que en primer lugar utilicen el paralelismo de instrucciones típico de un máquina MIMD como un multiprocesador. A continuación el resultado puede ser la entrada de una matriz SIMD o de una arquitectura segmentada MISD que utilicen el paralelismo de datos. En cualquier caso éstas son máquinas catalogadas como **híbridas** al combinar diversos tipos de procesamiento paralelo.

5. Arquitecturas especiales

- Redes **neuronales**: Este tipo de arquitecturas, **útiles** para problemas especiales, se compone de un gran número de **EPs** conectados formando una red. El nombre proviene de la similitud con las redes neuronales biológicas. Todas las entradas de una neurona o EP se multiplican por un coeficiente y se suman, para determinar de esta manera el umbral de activación de las salidas de la neurona. El valor de los coeficientes puede variar de manera **adaptativa** y fijarse con un proceso de aprendizaje de la **red**. En este caso el elemento más importante no es el EP o **neurona**, sino la forma de la red de interconexión. Una aplicación típica de las redes **neuronales** es el reconocimiento de patrones. La red fija sus coeficientes identificando una serie de patrones ya definidos. Al final del **proceso** de aprendizaje la red deberá clasificar patrones para los que no ha sido previamente preparada. La figura 12.9 muestra un esquema de una red **neuronal**.

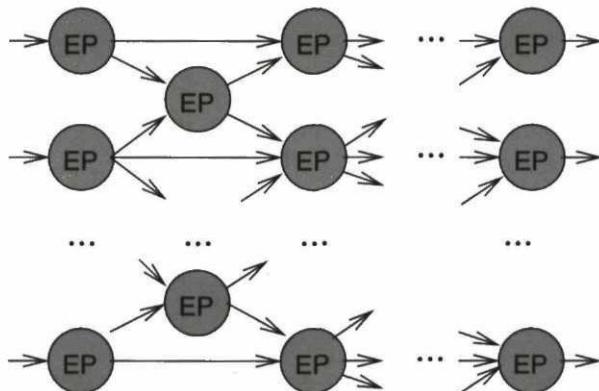


Figura 12.9: Diagrama de una red neuronal

12.4 MULTIPROCESADORES Y MULTICOMPUTADORES

En las arquitecturas definidas por Flynn como **MIMD** cada **procesador** ejecuta su propio flujo de **instrucciones** sobre su propio flujo de datos. La arquitectura **multiprocesador** consiste en dotar al computador de varios procesadores, compartiendo **éstos** la memoria y recursos del sistema. El programador debe **dividir**

las tareas de forma eficiente a lo **largo** de la red de procesadores para aprovechar al máximo el paralelismo. Análogamente en los **multicomputadores** la tarea global se ha dividido entre varios computadores conectados a **través** de una red de interconexión. Las ventajas obtenidas del paralelismo de la división del tiempo de proceso entre varios **procesadores** o computadores pueden perderse en la comunicación a través de la red de interconexión. Es **por** tanto necesario que la red de interconexión entre los elementos que **realizan los** cálculos no **degrade** las prestaciones globales del multiprocesador o **multicomputador**.

12.4.1 Redes de interconexión

Una red de interconexión entre diversos **nodos** consta de enlaces (físicamente cables o canales) y conmutadores que conectan **unas** salidas con unas entradas. Los **nodos** de la red pueden ser procesadores, módulos de memoria, unidades de **Entrada/Salida, etc.** Se pueden clasificar las **topologías** de red en dos grupos: estáticas y dinámicas. Las redes **estáticas** mantienen unas conexiones fijas entre **nodos**, mientras que las dinámicas pueden **reconfigurar** la conexión entre los **nodos**. Un aspecto interesante en las redes de interconexión es el número de conexiones de cada nodo de proceso, lo cual **determina** la capacidad de comunicación de un nodo con sus vecinos. Las redes con una gran complejidad y potencia de cálculo serán las que tengan una **gran** conectividad entre sus **nodos**. Estas redes por el contrario presentarán como desventaja la mayor **dificultad** para encaminar o **rutar** los datos a través de la **red** y el mayor coste de fabricación.

Redes **estáticas**

Las redes estáticas se pueden clasificar a su vez en función del número de enlaces **bidireccionales** que parten de un nodo, obteniéndose los siguientes grupos:

1. Cuando los **nodos** tienen un solo enlace se tiene un **estructura** de bus compartido, tal como se ha descrito en el capítulo 6. Es la red de **interconexión** más económica y en la que se pueden añadir y suprimir elementos con más facilidad. La desventaja fundamental de este tipo de red es que el propio bus es un cuello de botella en las comunicaciones. Adicionalmente, se necesita un arbitraje de bus para determinar entre qué **nodos** se realizan las transacciones. Este tipo de redes se muestra en la figura 12.10 a).

2. Las redes lineales y los **anillos** son ejemplos de redes de interconexión con dos enlaces por nodo, tal como se muestra en la figura 12.10 b) y c). Las redes lineales son **estructuras** simples que tienen como principal desventaja que la **comunicación** entre el primer y último elemento puede ser excesivamente lenta. Esta comunicación puede ser **más rápida** con un **anillo bidireccional**.

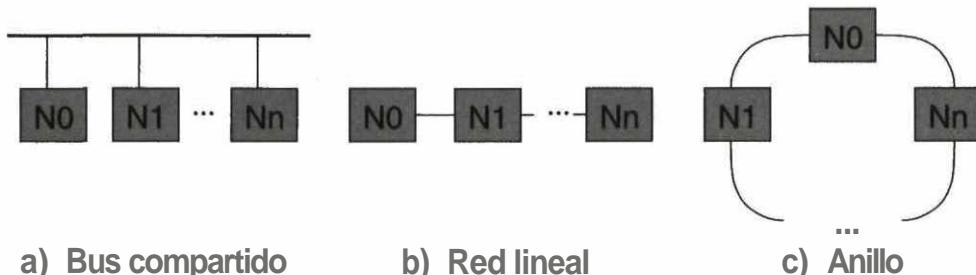


Figura 12.10: Topologías de red estáticas de 1 y 2 enlaces por nodo

3. Un ejemplo de red de interconexión con 3 enlaces por nodo es el **árbol binario** que se muestra en la figura 12.11 a). Desde cada nodo se puede acceder a **3 nodos**, salvo el nodo superior, denominado **raíz**, desde el cual **sólo** se puede acceder a **2 nodos**. Los dos **nodos** inferiores se denominan **hijos**, y los **nodos** superiores **padres**. Este tipo de redes de interconexión tiene como ventajas que son fácilmente expandibles y que los datos pueden ser encaminados con facilidad de un nodo origen a su destino, evitando transmisiones **innecesarias**.
4. Una red de **interconexión** con 4 enlaces por nodo es la **malla bidimensional** de $i \times j$ **nodos** que se muestra en la figura 12.11 b). En esta topología es **también** sencillo transmitir datos entre **nodos** evitando **transmisiones innecesarias**. Las **mallas** bidimensionales son especialmente adecuadas para problemas que requieren cálculos en una rejilla de puntos, como puede ser la resolución de ecuaciones diferenciales. Un caso especial de **mallas** bidimensionales son aquellas en las que los extremos también están conectados con el extremo opuesto. De esta manera se **construyen** las conocidas como **redes de Illiac**, redes donde todos los **nodos** tienen 4 enlaces y por tanto no hay extremos.

5. Otro tipo de redes son las genéricas redes ***n*-cubos** también conocidas como hipercubos de orden *n*. En un **hipercubo** de orden *n* cada nodo tiene *n* enlaces que lo comunican con *n* **nodos** vecinos. Si se utilizan *n* **bits** para **identificar** a cada nodo, entonces cada nodo se comunicará con **aquellos nodos** cuyo identificador se diferencie en un solo bit. De esta manera es posible un encaminado eficiente de la información manteniendo redes de gran potencia y complejidad. La figura 12.11 c) muestra la topología de un **hipercubo** de orden 3.

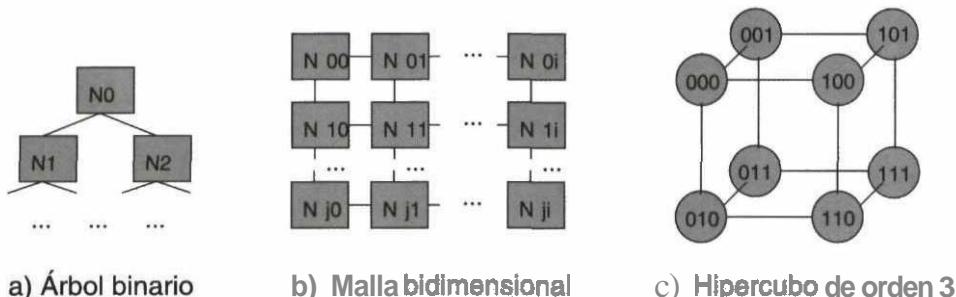


Figura 12.11: Topologías de red estáticas de 3 y más enlaces por nodo

Redes dinámicas

En este tipo de redes, además de los enlaces y los **nodos**, aparecen los **comunicadores de interconexión** como elementos que definen la topología de la red. Los comunicadores de interconexión son unos circuitos que, en función de unas señales de control, conectan un conjunto de salidas con un conjunto de entradas. De esta manera los enlaces entre **nodos** son **reconfigurables** simplemente cambiando las señales de control del comunicador de interconexión.

Dentro de la redes de **interconexión dinámicas** se puede realizar una primera **clasificación** en tres **grupos**:

1. Red de **barras cruzadas**: En este tipo de redes los comunicadores de **interconexión** pueden conectar cualquier entrada con cualquier salida, de esta manera todos los **nodos** pueden conectarse entre si. La figura 12.12 a) muestra el tipo de comunicadores de interconexión utilizados en estas redes. Se puede realizar un esquema de este tipo de redes como un número

n de enlaces verticales **cruzados** con un número n de enlaces horizontales. Con esta configuración se obtiene un número de n^2 puntos de cruce, estableciéndola conexión entre dos **nodos** eligiendo el punto de cruce adecuado como conectado. Este tipo de redes es ideal cuando n es pequeño, siendo excesivamente complejas cuando n es grande.

2. **Red Monoetapa:** También conectan cualquier entrada con cualquier salida, pero para ello es necesario que los algoritmos de **rutado** dirijan el flujo de datos atravesando la red más de una vez. **Ésta** es la **forma** en que se pueden establecer las conexiones adecuadas en los conmutadores de conexión. La figura 12.12 b) muestra este **tipo** de redes, más económicas que las de barras cruzadas pero con menos prestaciones.
3. **Red Multietapa:** **Las** redes multietapa disponen de varias etapas de conmutadores que pueden ser **reconfigurados** dinámicamente, estando conectadas por redes de interconexión fijas. Hay muchos patrones distintos de interconexión entre etapas y las prestaciones de estas redes aumentan con la complejidad de los conmutadores. La figura 12.12 c) muestra la arquitectura **genérica** de este tipo de redes.

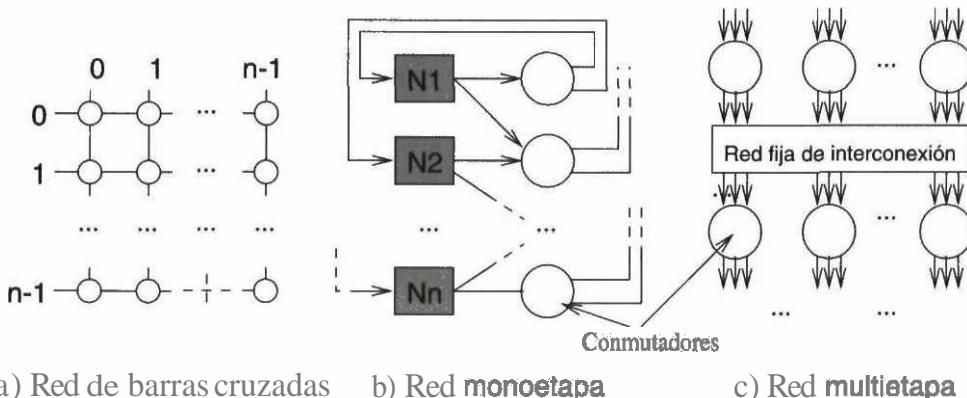


Figura 12.12: Topologías de red dinámicas

12.4.2 Multiprocesadores

Una forma de **abordar** la realización de tareas de forma paralela es mediante la utilización de los multiprocesadores. Una de las opciones posibles consiste

en que varios procesadores ejecuten diversas partes de la tarea global en paralelo, compartiendo una memoria global. Si todos los procesadores trabajan en paralelo de forma simultánea, la ejecución de la **tarea** se **acelerará** lo máximo posible. Claramente esto no es siempre posible debido a las dependencias de datos y la **secuencialidad** inherente de muchas tareas. **Habitualmente** el programador describirá el programa en un lenguaje que permita directivas de compilación paralelas. El **compilador** se encargará de distribuir la carga de tareas a cada **procesador** y optimizar la comunicación entre ellos. Si todos los **procesadores** son iguales se dice que el sistema es homogéneo. En caso de que haya varios tipos de **procesadores** se dice que el sistema es heterogéneo.

En función de la **interrelación** que exista entre **los** procesadores compartiendo recursos, o nivel de **acoplamiento**, se pueden definir dos clases de sistemas multiprocesador.

1. Acoplamiento débil: Se comparten pocos recursos además de la memoria del sistema. Cada procesador tiene una memoria local accesible por el resto de **procesadores**, tal como muestra la figura 12.13 a). El tiempo de acceso para la memoria local será mucho menor que para el resto de memoria distribuida por el resto de procesadores.
2. Acoplamiento **fuerte**: La memoria compartida no es local a los procesadores y se encuentra accesible a través de la red de **interconexión**. Todos los **procesadores** del sistema pueden utilizar sus recursos y todos los procesadores acceden de la misma manera. En la figura 12.13 b) se muestra la arquitectura de un sistema **multiprocesador fuertemente** acoplado.

En la práctica el nivel de prestaciones deseado complica las exigencias para la comunicación entre los **procesadores**. Cuando se tienen muchos **procesadores** rápidos compitiendo por acceder a la memoria común a través del bus, los **conflictos** pueden degradar seriamente el sistema. Es por esto **por** lo que se les suele dotar de una memoria local rápida o **cache**, apareciendo entonces problemas de coherencia de datos entre las diversas **copias** de datos en las caches de los **procesadores**. Si un procesador modifica un dato en su propia **cache** las copias del mismo dato existentes en el resto de caches ya no serán **válidas**. Para solucionar estos conflictos el sistema debe estar dotado con un protocolo de coherencia de **caches** que garantice la integridad de los datos. **Algunas** veces la resolución de este problema se aborda por software, mediante compiladores que eviten la

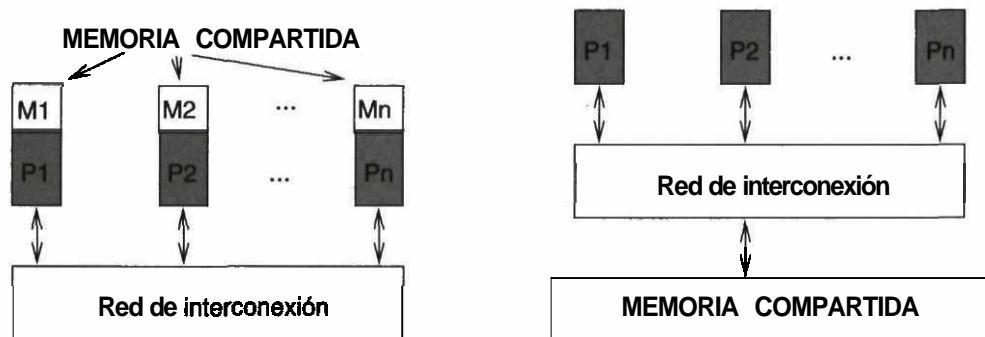


Figura 12.13: *Multiprocesador a) débilmente acoplado y b) fuertemente acoplado*

incoherencia de datos **compartidos** entre caches. Si embargo la mayoría de las veces hay mecanismos hardware que garantizan la coherencia de caches que dotan de total **transparencia** al software. Los diversos **protocolos** de coherencia de caches se basan en dos metodologías **básicas**:

- 1. Protocolos de sondeo.** Se basan en que cada procesador espía las transacciones que realiza el resto de procesadores entre memoria y las caches locales. De esta manera cada procesador observa las transacciones de los demás que podrían introducir incoherencia con su propia cache e **invalidar** sus copias si es necesario. Este tipo de protocolos es adecuado en los sistemas multiprocesador basados en bus con un solo nivel de jerarquía.
- 2. Protocolos basados en directorio.** En el caso de que la red de interconexión sea más compleja que un solo bus o mantenga una jerarquía de **buses**, es complicado que todos los procesadores se espíen mutuamente cuando se realizan transacciones de memoria. En este caso la mejor solución es crear un directorio común donde se guarda la validez de las líneas de cache de todos los procesadores.

A partir de los algoritmos típicos de coherencia de cache en **monoprocesadores** (**write-through** o **write-back**) se heredan algoritmos similares para **multiprocesadores**, tanto en protocolos basados en sondeo como en protocolos basados en directorio. En cualquier caso existen muchas aproximaciones distintas cuyo rendimiento es más óptimo dependiendo del número de procesadores y del tipo de la memoria cache de **éstos**.

12.4.3 Multicomputadores

Los **multicomputadores** son redes de computadores en los que cada elemento de proceso tiene su propio espacio de memoria que no resulta accesible por otros procesadores. El motivo principal para el desarrollo de **multicomputadores** es el de superar las limitaciones en la escalabilidad de los **multiprocesadores**. Estas **limitaciones** se deben a que en sistemas **multiprocesador** basados en bus el rendimiento se puede degradar rápidamente si se aumenta el número de **procesadores** debido a que aumenta la competencia por usar el bus y éste puede no satisfacer las nuevas necesidades de ancho de banda. Para sistemas **multiprocesadores** con un número grande de procesadores sería necesaria una **topología** de red más compleja, lo cual **dificultará** la **escalabilidad** del sistema.

El hecho de que en un multicomputador cada procesador disponga de sus propios recursos y memoria hace que adquiera la entidad de computador, de ahí el nombre de **multicomputador**. De nuevo si los **procesadores** de los computadores son iguales el sistema se dice que es homogéneo, y si no lo son se dice que es heterogéneo.

Un elemento clave en la definición de un multicomputadores la red de interconexión entre los computadores o la topología de red. Cada computador tendrá un **número** de enlaces, formando una red de interconexión, como las descritas en la sección 12.4.1. Muchas redes distintas han sido utilizadas en **multicomputadores**, dependiendo la eficiencia de la red del **número** de computadores y de la tarea desempeñada (necesidades de comunicación entre procesos). Al no **compartirse** el espacio de memoria la comunicación entre procesos se arbitra mediante técnicas de paso de mensajes.

125 EL RENDIMIENTO

Las palabras rendimiento y **mejora** han sido utilizadas hasta ahora de una manera bastante intuitiva en el presente texto, permitiendo hacer comparaciones de sistemas paralelos entre sí o entre estos sistemas y la máquina Von **Neumann**. Para estos conceptos existen unas **definiciones** matemáticas que permiten comparar computadores entre sí sin **ambigüedades**. Una de las medidas consideradas más populares, como los **MIPS** definidos en la sección 2.10.1, no son en **realidad**.

dad **fiables** de las prestaciones de un computador. En esta sección se introducen **algunas** ideas que van a ser útiles al estudiante para **realizar** comparaciones entre computadores.

12.5.1 Sistemas con un solo procesador

En procesadores **secuenciales** se define el rendimiento a partir del tiempo de **ejecución** de un programa en segundos. De esta forma el ordenador que realiza un **programa** en poco tiempo tiene un alto rendimiento y a la inversa. El rendimiento R_x para la tarea X que tiene un tiempo de **ejecución** T_x **vendrá** definido por:

$$R_x = \frac{1}{T_x} \quad (12.1)$$

El rendimiento en la **ejecución** de un programa en un cierto computador se **medirá** en s^{-1} .

Otro **parámetro** interesante para medir la velocidad de un computador **secuencial** puede ser la velocidad del reloj del procesador expresada como la duración del ciclo en segundos (CLK). De esta manera el tiempo de CPU para un programa (T_{CPU}) podría expresarse en función de la **duración** del ciclo CLK y del número de ciclos de un programa N:

$$T_{CPU} = CLK \cdot N \quad (12.2)$$

Análogamente es posible expresar la misma **cantidad** en **función** de la frecuencia de reloj ν .

$$T_{CPU} = \frac{N}{\nu} \quad (12.3)$$

De la **ecuación** 12.3 se concluye que la manera de disminuir el tiempo de ejecución en una máquina Von **Neumann** es, o bien reduciendo el número de

ciclos de reloj del programa, o bien aumentando la **frecuencia** de reloj (cabe hacer notar que se **están** obviando aspectos tales como que si se aumenta la frecuencia de reloj la **memoria también** debe ser más rápida, o en su defecto incluir más ciclos de espera en los accesos a memoria. con lo que N aumentará). El número total de ciclos de reloj del **programa** será la suma del número de ciclos de cada instrucción tantas veces como aparezca dicha **instrucción**. Por tanto, cuantos menos ciclos tarde en ejecutarse una instrucción, el tiempo de **CPU** será menor.

Cada instrucción utiliza en su ejecución un número de ciclos en un **procesador** que no tiene por qué ser el **mismo** que la misma instrucción en **otro procesador**. **Más aún**, el juego de instrucciones cambia de un procesador a otro, y una instrucción máquina en un procesador puede no tener **equivalente** en **otro** y ser equivalente a un conjunto de varias instrucciones. El problema es mayor cuando se utiliza esta cantidad para hacer comparaciones, ya que los **MIPS varían** entre programas en un mismo **computador**, dependen del **repertorio** de instrucciones (lo cual hace difícil la comparación entre computadores con repertorios diferentes), e incluso pueden **variar** inversamente al **rendimiento**. De esta manera se concluye que no es adecuado **evaluar** el rendimiento de **procesadores** secuenciales realizando **una** comparación de **MIPS**.

Se puede **definir** el promedio de ciclos de reloj por instrucción o CPI, como el número medio de ciclos que tarda en ejecutarse una instrucción en un procesador al ejecutarse un programa. Utilizando esta ecuación y la ecuación 12.2 se puede expresar el tiempo de CPU para ejecutar una tarea como

$$T_{CPU} = I \cdot CPI \cdot CLK \quad (12.4)$$

donde **I** es el número de instrucciones del programa. **Análogamente**,

$$T_{CPU} = \frac{I \cdot CPI}{\nu} \quad (12.5)$$

Esta ecuación da la relación **entre** los tres factores clave para obtener el **rendimiento** de un **procesador secuencial**.

12.5.2 Máquinas paralelas

Para evaluar la mejora de la **implementación** de una tarea en una máquina paralela, frente a la ejecución de la misma tarea en una máquina secuencial, se define la magnitud conocida como aceleración o *speedup*.

$$\text{acelecidn} = \frac{\text{tiempo de ejecución en la máquina secuencial}}{\text{tiempo de ejecución en la máquina paralela}} \quad (12.6)$$

Una tarea tarda en ejecutarse en un **máquina** secuencial un tiempo T_s y un tiempo T_p en una máquina paralela. En el caso ideal de que la tarea se reparta perfectamente entre los N elementos de proceso iguales (**procesadores** o computadores) se tendrá que $T_p = \frac{T_s}{N}$, con lo cual la aceleración **será** N . Éste es el caso ideal conocido como aceleración perfecta que en casos reales se ve degradada por factores como la **serialización** de la **tarea**, la dependencia de datos, la comunicación entre elementos de proceso y cuellos de botella de **Entrada/Salida**.

En una tarea **genérica** **habrá** partes **paralelizables**, y por tanto susceptibles a una mejora importante en su ejecución en una máquina paralela, y partes sometidas a una inevitable ejecución secuencial. Si s es el tiempo de ejecución en una máquina **secuencial del código** secuencial, y p es el **tiempo** de ejecución del **código** restante **paralelizable** en una máquina secuencial, Ahmdal definió en 1967 que

$$\text{aceleración} = \frac{s + p}{s + \frac{p}{N}} \quad (12.7)$$

donde N es el número de **procesadores**. Claramente sólo en el caso en que $s=0$ la aceleración puede ser **perfecta** e igual a N . Esta expresión de la aceleración se suele expresar **normalizada** con $s + p = 1$, con lo que la aceleración tendrá un valor **máximo** cuando $N \rightarrow \infty$ de $1/s$, conclusión conocida como cuello de botella secuencial.

$$\text{aceleración} = \frac{1}{s + \frac{p}{N}} \quad (12.8)$$

La ecuación anterior muestra la evolución de la aceleración en función del número de elementos de proceso para una carga fija. **Guftafson mostró** en 1988 que para ciertos problemas el tiempo de ejecución paralelo p crece junto con el número de **procesadores**. De esta manera la medida de la aceleración expresada en la ecuación 12.8 suele denominarse **aceleración de ancho fijo** y la ecuación 12.9 o ley de Guftafson **aceleración de ancho escalado**. En este caso s y p representan el tiempo serie y paralelo utilizado por N elementos de proceso, con $N > 1$.

$$\text{aceleración de ancho escalado} = \frac{s + p \cdot N}{s + p} \quad (12.9)$$

Como se asume que $s + p = 1$, se obtiene que

$$\text{aceleración de ancho escalado} = N - s \cdot (N - 1) \quad (12.10)$$

En **cualquier** caso todas estas medidas de la aceleración de un computador se refieren al tiempo de ejecución de una tarea concreta en función del número de elementos de proceso. En **ningún** caso se tiene una magnitud absoluta que permita asegurar qué máquina tendrá siempre más prestaciones que otra.

Diversos computadores pueden estar **más** orientados a tareas concretas de cálculo en punto flotante, cálculo vectorial o aplicaciones de cálculo intensivo diferencial. En cualquier caso la adecuación para un **tipo** concreto de tareas no **garantiza** que para otro tipo de programas las prestaciones sean similares.

Se han escogido una serie de programas de **prueba** o **Benchmarks** que evalúan las prestaciones de los computadores. Así, dependiendo de las aplicaciones que el usuario vaya a ejecutar **mayoritariamente**, el programa de prueba será alguna aplicación típica lo más parecida posible a la aplicación final de esta máquina.

Estos programas de prueba pueden ser útiles para evaluar de forma comparativa máquinas que van a destinarse a la misma aplicación; cálculo **científico**, **ingeniería**, bases de datos, etc. La asociación sin **ánimo** de lucro SPEC (**Standard Performance Evaluation Corporation**) dispone de una amplio catálogo de programas y aplicaciones que pueden ser útiles para evaluar: grandes **aplica-**

ciones industriales, arquitecturas **secuenciales** y paralelas de memoria **compartida/distribuida**, estaciones de trabajo y computadores paralelos de altas prestaciones.

En cualquier caso la elección del computador con las mejores prestaciones para una aplicación **concreta** depende del coste permitido. Hoy en día la tendencia hacia la utilización de multiprocesadores en aplicaciones se explica porque son arquitecturas con una **escalabilidad** razonable con una pequeña inversión económica, lo cual permite que la potencia de la máquina sea proporcional al dinero disponible. Posterior ampliaciones son posibles **sólo** con añadir más procesadores en función de la inversión que se quiera realizar.

12.6 CUESTIONES

12.1 *El diseño de un cauce segmentado impone la división de una tarea en etapas, almacenando en estas **etapas intermedias** el resultado parcial de cada etapa. Este **esquema** parece que va a **ralentizar** la **ejecución** de una sola tarea, ya que **además** del tiempo de ejecución en cada etapa es necesario cumplir unos **tiempos mínimos** para realizar el **almacenamiento** entre etapas. Indicar de qué manera la segmentación **mejora** el **tiempo** de ejecución de instrucciones.*

12.2 *Diseñar un multiplicador segmentado de números binarios positivos a partir de la **matriz** de multiplicación **combinacional** basada en el **método del lápiz y papel** que se muestra en la figura 3.15.*

12.3 *Describir en qué casos aparecen los riesgos en los cauces **segmentados**. ¿De qué manera pueden evitarse?*

12.4 *¿En qué se basa la taxonomía de Flynn? Describir las principales dificultades de esta clasificación.*

12.5 *¿Qué ventajas y desventajas presentan las **matrices sistólicas**? ¿Cuándo es adecuado su uso y en qué casos no es aconsejable?*

12.6 Enumerar los **diferentes** tipos de redes estáticas indicando **sus** ventajas y desventajas **así como** en qué casos es adecuado el w o de cada tipo.

12.7 ¿Por qué aparecen problemas de coherencia de **cache** en los **multiprocesadores**? ¿Existe el **mismo** problema en los **multicomputadores**? Enumerar las **metodologías** básicas para realizar protocolos de coherencia de caches, indicando sus ventajas y desventajas.

12.8 Realizar un **análisis** de las principales características de los **multiprocesadores frente** a los multicomputadores.

12.9 Indicar las razones que desaconsejan la **utilización** de un **parámetro** tan popular como los **MIPS** para evaluar el rendimiento **entre** computadores.

12.10 La ley de **Ahmdal** expresa la **aceleración de ejecución** de una tarea en una máquina paralela en **función** del tiempo de ejecución de la misma tarea en un máquina **secuencial**, del tiempo de **ejecución** en la máquina paralela y del **número** de procesadores. Analizar la **expresión** suministrada por Ahmdal indicando en qué casos se **obtendría** la mejor y peor **aceleración** posible. ¿Por qué fue necesario introducir la ley de **Gustafson** en 1988? Analiza esta nueva expresión.

12.7 BIBLIOGRAFÍA

- Computer Architecture. Single and Parallel Systems. *Mehdi R. Zargham*. Editorial Prentice Hall. USA, 1996. ISBN 0-13-529497-5.
- Computer Architecture. A Quantitative Approach. *John L. Hennessy y David A. Patterson*. Morgan Kaufmann publishers INC. USA, 1996. ISBN 1-55860-327-7.
- Computer Architecture. Pipelined and Parallel Processor Design. *Michael J. Flynn*. Editorial Jones and Bartlett. UK, 1995. ISBN 0-86720-204-1.

- **Computer Architecture. Design and Performance.** *Barry Wilkinson*. Editorial Prentice Hall. USA, 1996. ISBN 0-13-518200-X.
- **High Performance Computer Architecture.** *Harold S. Stone*. Editorial Addison Wesley. USA, 1993. ISBN 0-201-52688-3.
- **Advanced Computer Architecture.** *Richard Y. Kain*. Editorial Prentice Hall. USA, 1996. ISBN 0-13-353673-4.

ÍNDICE ALFABÉTICO

A

Acceso

directo a memoria, 176, 179,
181

Stop-Start, 322,323
transparente, 179

Aceleración

de ancho escalado, 360
de ancho fijo, 360

Algoritmo

aleatorio, 142
de Booth, *véase* Multiplicación
entera con signo
de reemplazo, 135, 142

Almacenamiento

masivo, 130, 133, 305
secundario, 131,305,313

ALU (Arithmetic Logic Unit), *véase*
Unidad aritmético lógica

Ancho

de banda, 186

Anillos, 351

Arbitraje

daisy-chain, 198

centralizado-paralelo, 198

del bus, 196

distribuido por autoselección,
199

distribuido por detección de co-
lisión, 199

Árbol

binario, 351
de Wallace, 116

Arquitectura

Harvard, 85.88
MIMD, 344,345
MIMD-MISD, 348
MIMD-SIMD, 348
MISD, 344,347
SIMD, 344,346
SISD, 343
Von Neumann, 4, 32

Arquitecturas

avanzadas, 337
híbridas, 348

B

Baudio, 217

Benchmark, 360

BiCMOS, 22

Biestable SR, 242

Bit

de sucio, 154

de **válido**, 149

de comienzo, 217

de condición, véase Registro de estado

de **fin**, 217

de paridad, 217

de referencia, 154

Bloque de memoria, 136

BPI, 316

BTL, 187

Bus, 6,185

de **alimentación**, 14,189

de control, 14,189

de datos, 14, 189

de direcciones, 14,189

de expansión, 191

del **procesador**, 191

del sistema, 191

local, 191

PCI, 203

serie universal, 220

VME, 209

C

Cabezal

de **núcleo de ferrita**, 306

MR, 307

Cache, véase Memoria **cache**

copy-back, 136

write-back, 136,154

write-through, 135

asociativa *por* conjuntos, 140

totalmente asociativa, 136

Campo

de condición, 49

de dirección, 47

de código de operación, 46

Canal de E/S, 179

Cartucho de cinta, 325

Cauce **segmentado**, 339

CCD (Charge Coupled Device), 255

CD-DA (CD Digital Audio), 329

CD-I (CD interactivo), 330

CD-R (CD-Recordable), 331

CD-ROM, 328

CD-RW (CD-ReWritable), 331

Celda de bit, 307

CGA (Color Graphics Adapter), 298

Ciclo

de escritura, 66,212

de lectura, 55, 192.194

Cinta

de **vídeo** de 8 mm, 325

cintas

en **carretes**, 323

DAT, 326

magnéticas, 322

Circuitos integrados, 21

de memoria, 24

CISC, 86

Clasificación

de **Flynn**, 343

de **Zargham**, 344

CMOS, véase Tecnología CMOS

Código

autoreloj, 309,310

break, 247

CRC, 320

de grabación magnética, 309

M, 310

- make***, 246
máquina, 16
MFM, 311
no **autoreloj**, 310
NRZ, 310
NRZI, 310
PE, 310
RLL, 311
scan, 246
Coherencia de caches, 354
Coma flotante, 122
Computador elemental, 50
Concentrador, 222
Conector DB25, 226
Conexión
 daisy-chain, 188, 198, 210, 230
 en estrella, 222
 multipunto, 217
 null-modem, 220
 punto a punto, 216
Controlador
 de **cache**, 135
 de DMA, 176
 de pantalla, 291
 de pantalla **alfanumérica**, 291
 de pantalla gráfica, 297
 de teclado, 245
 USB, 224
Coprocesador, 167
CPI, 84.358
CPU (Central Process Unit), véase **Unidad central de proceso**
CRC, véase **Código CRC**
CRT (Cathode Ray Tube), véase **Tubo de rayos catódicos**
- D**
- DACRAM**, 298
- Decodificación** de la instrucción, 60
Degaussing, véase **Desmagnetización**
Densidad de grabación, 307, 309, 313
Desmagnetización, 288
Desplazamiento
 aritmético, 102
 circular, 102
 lógico, 101
Digitalizador de imágenes, 253
Diodo emisor de luz, 284
Dirección virtual, 149
Direccionamiento
 lineal, 292
 por filas y por columnas, 292
DMA (Direct Memory Access), véase **Acceso Directo a Memoria**
Directorio de la cache, 141
Disco
 óptico, 327
 duro, 317
 flexible, 315
 magnético, 318
 magneto-óptico, 333
 Winchester, 317
Displays, 284
División, 119
 con restauración, 120
 de números enteros sin signo, 119
 en coma flotante, 125
 sin restauración, 121
DRAM, véase **Memoria DRAM**
Driver, 187
DVD (Digital Versatile Disc), 331

E**E/S**

aislada, 172
mapeada en memoria, 171

E/S (Entrada/Salida), 165**EDVAC, 4**

EEPROM, véase Memoria EPROM

PROM

Efecto Keer, 333

EGA(Enhanced Graphics Adapter), 298

Enlace exterior, 215

Enmascaramiento, 175

Ensamblador, 16, 34

EPROM, véase Memoria EPROM

Equipo

terminal de datos, 218

transmisor de datos, 218

Error

de protección, 171

de paridad, 218

de **rotura, 218**

de **sobrepasamiento, 218**

de trama, 218

Escáner, véase Digitalizador de imágenes

Escrutinio, 173

Espacio de **direcciónamiento, 129**
virtual, 148,154,157

Estándar IEEE 1284,228

Excepciones, 72

F**Fallo de página, 139,142**

Fase de

arbitraje, 196,232

estado, 233

mensaje, 233

orden, 232

reselección, 232

Fetch, véase Lectura de la instrucción

Filtro RC, 242

Flags de condición, véase Registro de estado

Formato de grabación, 326

Frames, véase Marcos de página

Frecuencia

de reloj, 5

barrido horizontal, 289

carácter, 295

cuadro, 295

fila de caracteres, 295

línea, 295

Función de mapa, 135

G

GDI (Graphical Device Interface), 272

Grabación magnética, 306

Gráficos

raster, 288

vectoriales, 288

H

HGA (Hercules Graphic Card), 298

Hipercubo, 352

I

Impresora, 261

contone, 275

de burbuja, 274

de caracteres, 264

de impacto, 263

de **inyección de tinta, 272**

de líneas, 263
de leds, 268
de margarita, 261
de matriz de puntos, 264
de sublimación de tinta, 277
de tinta sólida, 277
de transferencia directa, 266
electrostática, 264
láser, 267
 en color, 269
térmica
 de cera, 278
 de color, 278
windows, 270

Inconsistencia de memoria, 143

Iniciador, 230

Instrucción máquina, **15, 32**

Intel **iX86**, 85

Interfaz, 216

- Centronics, 226
- EPP, 228
- paralelo, 226
- SCSI, 228
- serie, 217

Zinterleaving, 147, 319

Interrupción, 73

Interrupciones, 199

- Daisy-Chain**, 200
- mediante exploración **secuencial**, 200
- mediante líneas individuales, 199

Interruptor, 240

J

Jerarquía

- de **buses**, 190
- de memoria, 129

L

Lápiz óptico, 250

LCD (Liquid Cristal Display), véase Pantalla de cristal líquido

Lectura

- de la instrucción, 60
- de los operandos, 60

LED Light Emitting Diode, véase Diodo emisor de luz

Lenguaje de

- alto nivel, 16
- descripción de página, 270

Ley de

- Ahmdal, 359
- Guftafson, 360

Línea de

- cache, 136

Longitud de palabra, 51

LRU (Last Recently Used), 142

M

Maestro de bus, 190

Main memory, véase Memoria principal

Malla

- bidimensional, 351
- de apertura, 288
- de control, 287

Mapeado

- asociativo por conjuntos, 140
- directo, 137
- totalmente asociativo, 141

Máquina de flujo de datos, 364

Marcos de página, 148

Máscara

- binaria de condición, *véase Campo de condición*

de sombras, 288
Matrices sistólicas, 347
Matriz de multiplicación combinacional, 112
MDA (Monocrome Display Adapter), 298
Memoria
 cache, 135
 de control, 77
 de video, 292
DRAM, 25
 EDO DRAM, 293
 EEPROM, 27
EPROM, 27
 física, 148
FLASH, 27
 interpolada, 147
 jerárquica, véase Jerarquía de memoria
 lógica, 148
 principal, 129
PROM, 27
ROM, 26
SDRAM, 293
 segmentada, 154
SRAM, 25
SRAM síncrona, 25
 virtual, 148
 VRAM, 293
Microinstrucción, 77
Microprograma, 78
Microprogramación
 horizontal, 79
 vertical, 79
MIPS, 84,356
MMU (Memory Management Unit),
 véase Unidad de gestión de memoria

Módem, 218
Modo de direccionamiento
 directo, 39
 indexado, 43
 indirecto, 43
 inmediato, 38
 mediante registro, 40
 relativo a registro, 40
Multi-multiprocesador, 345
Multicomputador, 345,356
Multiplicación, 111
 de alta velocidad, 115
 en coma flotante, 125
 entera con signo, 117
 entera sin signo, 112
 entera sin signo **combinacional**, 112
 entera sin signo **secuencial**, 113
Multiprocesador, 345,353

N

Nivel físico del bus, 187
Null modem, 220
Número
 de bloque, 138
 de **página**, 149, 156
 de **página virtual**, 149
 de segmento, 155

O

OCR (Reconocedor Óptico de Carácteres), 253
Operación
 aritmética, 96,100,106
 de cambio de signo, 104
 de desplazamiento, 98,101
 de extensión de signo, 104
 en coma **flotante**, 124

- lógica, 96, 100, 103
Operando, 8, 96
Overlay, 131, 150
- P**
- Página**, 132, 148
PAL, véase señal de video PAL
Pantallas
- alfanuméricas, 291
 - de cristal líquido, 300
 - de plasma, 301
 - de rayos catódicos, 285
 - de transistores de película fina, 301
 - gráficas, 297
 - sensibles al tacto, 251
- Papel
- continuo, 265
 - dieléctrico, 266
- Paralelismo, 338, 348
- PC (Program Counter)**, véase Registro contador & programa
- PCI**, véase Bus PCI
- PCL (Printer Control Language)**, 271
- PCM (Pulse Code Modulation)**, 332
- Periféricos**, 2, 6, 15
- Petición
- de bus, 189, 198
 - de interrupción, 189, 199
- Pila, 36
- Píxel**, 286
- Planos de deflexión, 286
- Plotter**, 276
- Plumillas, 276
- PMS (Processor Memory Switch)**, 3
- Política de acceso, 130
- Polling**, véase Escrutinio
- Postscript**, 271'
- Principio** de localidad
- espacial, 133
 - temporal, 133
- PRML (Partial Response Maximum Likelihood)**, 308
- Procesador de E/S, 215
- Procesadores
- matriciales, 346
 - segmentados, véase Segmentación
 - vectoriales, 347
- Profundidad** de bit, 256
- Programa
- de canal, 179
 - reentrante, 151
- PROM**, véase Memoria PROM
- Protocolos
- basados en directorio, 355
 - de sondeo, 355
- Prueba** de estado, 173
- Puerto
- de salida, 243
- Pulsador, 240
- R**
- Rack**, 187
- Rango dinámico, 256
- Ratón, 247
- óptico, 249
 - electromecánico, 248
 - optomecánico, 248
- Rayo láser, 327
- Rebotes, 241
- Red
- de barras cruzadas, 352
 - de interconexión, 350

- dinámica, 352
estáticas, 350
Illiac, 351
lineales, 351
monoetapa, 353
multietapa, 353
neuronal, 349
- Reflexiones, 202
- Refresco de la pantalla, 286
- Registro
 acumulador, **9, 52**
 base, 42
 contador de programa, **8, 52**
 de control, 166
 de datos, 166
 de enmascaramiento, 175
 de estado, **9, 36, 52, 95, 98**, 166
 de **instrucción**, **8, 51**
 de **interfaz** con el bus, 53
 de **propósito general**, **9, 52**
 de tabla de **páginas**, 152
 interno, **8, 50**
 puntero a pila, 70
 universal, 101
- Rejilla de control, 286
- Relocalización**, 151
- Rendering**, 271
- Rendimiento, 84, 356
- Repertorio de instrucciones, 32
- Resolución, 255
- Resta, 106
 en coma flotante, 124
 entera, 109
- Riesgo
 de control, 342
 estructural, 341
 por dependencia de datos, 341
- RISC**, 86
- Robo de ciclo, 178
- Rollover**, 246
- ROM, véase Memoria ROM
- RS232**, véase **Interfaz serie**
- RTL (Register Transfer Level)**, 3
- Ruido, 187
- Rutina de tratamiento, 72
- S**
- Scanner**, véase **Digitalizador de imágenes**
- Señal**
 de barrido horizontal, 287
 de barrido vertical, 287
 de petición, 217
 de reconocimiento, 194, 217
 de video PAL, 289
- Sector, 314
- Segmentación, 339
 paginada, 156
- Segmento, 154
- Sensor**
 de **ultrasonidos**, 251
 de **infrarrojos**, 251
- Sincronismo
 horizontal, 287
 vertical, 287
- Sincronización**, 172
 por **interrupción**, 173
 por prueba de estado, 173
- Sistema
 bajo demanda, 273
 operativo, 16
- Slots**, 187
- Solenoide, 265
- SRAM**, véase Memoria SRAM
- SRAM síncrona**, véase Memoria SRAM síncrona

- S**ubrutina, 69
Subsistema de **entrada/salida**, 165
Suma, 106
 BCD, 110
 en coma flotante, 124
 entera, 106
Sumador
 con acarreo adelantado, 107
 elemental **binario**, 106
 restador, 109
SVGA (Super Video Graphics Adapter), 298
- T**
Tóner, 264
Tabla
 de **páginas**, 149
 de segmentos, 155
Tableta digitalizadora, 252
Tag, 141
Tambor fotosensible, 267
Tambores, 261
Tarjeta de video, 298
Tasa
 de aciertos, 134
 de fallos, 134
Tecla, 240
Teclados, 243
Tecnología
 CMOS, 22
 de **circuitos integrados**, 3
 de computadores, 20
m, 22
Teletipo, 261
Temperatura de Curie, 333
Terminales de vídeo, 290
Tiempo
 de acceso, 130
de acierto, 134
de ciclo, 130
de **fallo**, 134
de transferencia, 144
TLB (Translation Lookaside Buffer), 152
TPI, 316
Trackballs, 247
Traducción de las direcciones, 149
Transductores ópticos, 283
Transferencia
 asíncrona, 193
 en bloque, 195
 full-duplex, 219
 half-duplex, 219
 programada, 173
 read-modify-write, 196
 síncrona, 192
Transmisión
 balanceada, 224
Trap, 72
Trazador, 276
TTL, véase **Tecnología TTL**
Tubo de rayos **catódicos**, 283,285
- U**
UC (Central Unit), véase **Unidad de control**
Unidad
 aritmético lógica, 95
 central de proceso, 5,6
 de control, 7, 50, 54
 de control **cableada**, 76
 de control **microprogramada**, 77
 de **gestión** de memoria, 149
Unidad de gestión de memoria, 84, 149

USB *Universal Serial Bus*, véase
Bus serie universal

VDT (*Video Display Terminals*),
290

V

VCD(Vídeo CD), 330

VGA(*Video Graphics Adapter*), 298
VLSI (*Very Large Scale of Integration*), 3

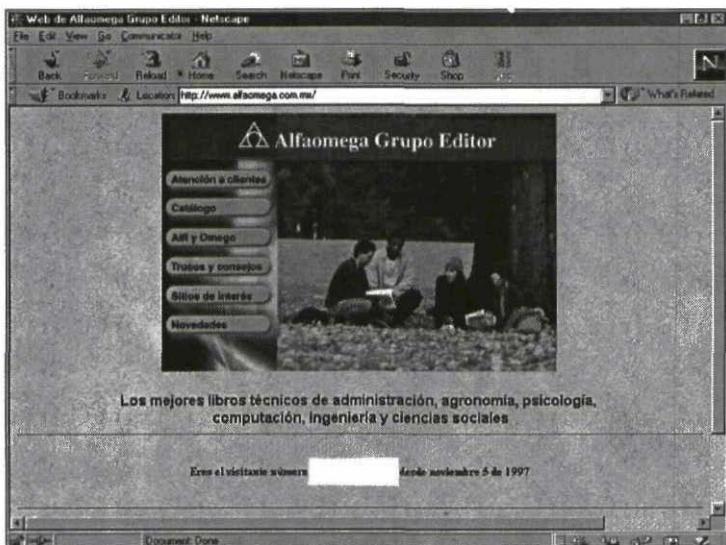
VME, véase Bus *VME*

CE25/E1/01

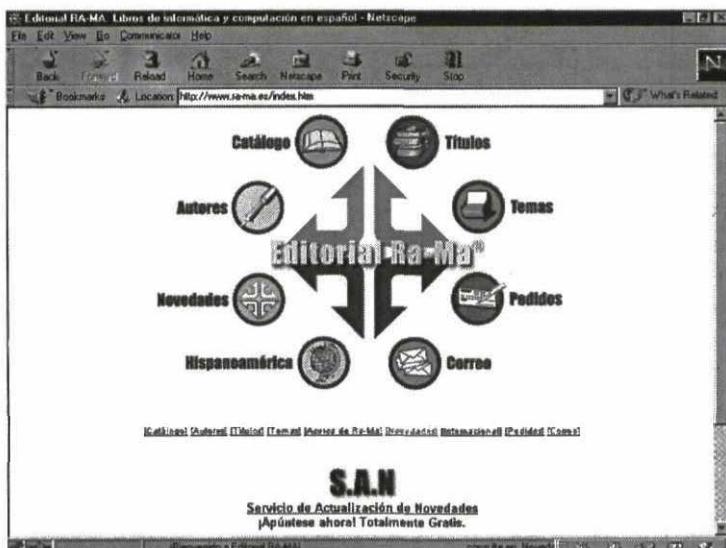
Esta edición se terminó de imprimir en julio de 2001. Publicada por ALFAOMEGA GRUPO EDITOR, S.A. de C.V. Apartado Postal 73-267, 03311, México, D.F. La impresión y encuadernación se realizaron en QUEBECOR WORLD BOGOTÁ, S.A.. Calle 15 No. 39A/34, Santafé de Bogotá, D.C.- Colombia.

Visítenos en Internet:

<http://www.alfaomega.com.mx>



<http://www.ra-ma.es>



Los mejores libros de computación