

Estructura y Arquitectura de Computadores

GUÍA PRÁCTICA
para alumnos de primero y segundo
curso de Grado en Ingeniería
Informática de Gestión y Sistemas de
Información

**GUILLERMO BOSQUE PÉREZ
NEKANE AZKONA ESTEFANÍA**

Teknologia Elektronika Saila
Departamento de Tecnología Electrónica

ARGITALPEN ZERBITZUA
SERVICIO EDITORIAL

www.argitalpenak.ehu.es
ISBN: 978-84-9860-719-2

Universidad
del País Vasco
Euskal Herriko
Unibertsitatea



Estructura y Arquitectura de Computadores

Guía Práctica

Para Alumnos de Primero y Segundo Curso de Grado en:

Ingeniería Informática de Gestión y Sistemas de Información

Versión 2.0

Guillermo Bosque Perez

Profesor Agregado

Nekane Azkona Estefanía

Profesora Adjunta

Universidad del País Vasco - Euskal Herriko Unibertsitatea

Teknologia Elektronikoa Saila / Departamento de Tecnología Electrónica



Coordinación y Estilo: Guillermo Bosque.

La coordinación y estilo está referida a: Contenidos generales, capítulos, unicidad de criterios expositivos de Tablas y Figuras, bibliografía y control de versiones.

El coordinador se reserva el derecho a incluir nuevos capítulos que den mayor contenido a esta publicación, incluir nuevos autores que garanticen estos nuevos contenidos así como el mantenimiento de la línea editorial del libro.

Autores por capítulos:

Nekane Azkona: Capítulos 1, 2, y 6 (parcial).

Guillermo Bosque: Capítulos 1 (parcial), 3, 4, 5, 6, 7 y 8.

Estructura y Arquitectura de Computadores - Guía Práctica

Versión 0.0 2014

Versión 1.0 2016 Actualización y erratas

Versión 2.0 2018 Actualización (Cap. 4: Memorias de Estado Sólido, Cap. 5: Buses industriales y Cap. 6: Subrutinas), erratas. Incorporación de capítulos dedicados a CPU, Modos de Direcciónamiento, Interrupciones y Sincronización de Datos

©Euskal Herriko Unibertsitateko Argitalpen Zerbitzua
Servicio Editorial de la Universidad del País Vasco

ISBN: 978-84-9860-719-2

Impresión/Imprimatzea:

Servicio Editorial/Argitalpen Zerbitzua UPV/EHU

Cubierta:

Paradox - Oficina General de Inventos

guillermo.bosque@ehu.eus

gbosqueperez@gmail.com

nekane.azkona@ehu.eus

www.ehu.es/guillermo.bosque

Agradecimientos: A Enrique Bosque por su ayuda en el diseño de la portada y a Paradox “Oficina General de Inventos” (Guillermo y Javi) por las ideas y realización del diseño.

Índice general

Índice de figuras	11
Índice de tablas	17
PREFACIO	21
OBJETIVOS Y ESTRUCTURA DEL TRABAJO	23
GLOSARIO	27
TÉRMINOS y ACRÓNIMOS	27
BIBLIOGRAFÍA-Observaciones	31
Capítulo 1. UNIDAD CENTRAL DE PROCESOS	33
1.1. INTRODUCCIÓN	34
1.2. ARQUITECTURAS	34
1.3. ELEMENTOS CONSTITUTIVOS DE LA CPU	37
1.4. RUTA DE DATOS	39
1.5. UNIDAD DE CONTROL	57
1.6. COMPUTADOR EN CANALIZACIÓN	70
Capítulo 2. FORMATO DE INSTRUCCIÓN, MODOS DE DIRECCIONAMIENTO E INTERRUPCIONES	77
2.1. FORMATO DE INSTRUCCIÓN	77
2.2. MODOS DE DIRECCIONAMIENTO	79
2.3. CLASIFICACIÓN DE LOS TIPOS DE INSTRUCCIÓN	83
2.4. INTERRUPCIONES	83
Capítulo 3. LENGUAJE MÁQUINA Y ENSAMBLADOR	87
3.1. INTRODUCCIÓN	87
3.2. LENGUAJE ENSAMBLADOR (<i>Assembly Language</i>)	88
3.3. MACROINSTRUCCIONES (Macros)	94
3.4. SUBRUTINAS	99
3.5. ENSAMBLADOR (<i>Assembler</i>)	107
3.6. ENLAZADO Y CARGA	111
Capítulo 4. MEMORIAS	117
4.1. INTRODUCCIÓN	117

4.2. CARACTERÍSTICAS	118
4.3. JERARQUÍAS DE MEMORIA	127
4.4. MEMORIA PRINCIPAL SEMICONDUCTORA	127
4.5. MEMORIA CACHÉ	135
4.6. MEMORIA ASOCIATIVA	152
4.7. MEMORIA COMPARTIDA	155
4.8. MEMORIA VIRTUAL	156
4.9. OTRAS MEMORIAS	160
 Capítulo 5. BUSES	 171
5.1. INTRODUCCIÓN	171
5.2. CARACTERÍSTICAS DE LOS BUSES	172
5.3. EJEMPLOS DE BUSES DE PROPÓSITO GENERAL: ISA y PCI	179
5.4. EJEMPLOS DE BUSES INDUSTRIALES: VME, MULTIBUS II, NuBus y FUTUREBUS	184
5.5. EJEMPLOS DE BUSES SERIE: USB, FIREWIRE	193
 Capítulo 6. DISPOSITIVOS DE ALMACENAMIENTO SECUNDARIO	 201
6.1. INTRODUCCIÓN	201
6.2. DISCOS MAGNÉTICOS	202
6.3. CD-ROM	211
6.4. DVD	214
6.5. DISCOS MAGNETO-ÓPTICOS	215
6.6. DISCOS BLU-RAY	217
6.7. CINTAS MAGNÉTICAS	219
6.8. PENDRIVES	220
6.9. DISCOS DE ESTADO SÓLIDO	220
 Capítulo 7. DISPOSITIVOS DE E/S	 225
7.1. INTRODUCCIÓN	225
7.2. TERMINALES	225
7.3. RATONES	231
7.4. IMPRESORAS	233
7.5. MODEMS	238
7.6. CÓDIGO DE CARACTERES	241
 Capítulo 8. ARQUITECTURA DE COMPUTACIÓN EN PARALELO	 245
8.1. INTRODUCCIÓN	245
8.2. MODOS DE INTERCONEXIÓN	248
8.3. CLUSTERS	255
8.4. TOPOLOGÍAS DE RED	256
8.5. LEY DE AMDAHL	261

8.6. PERSPECTIVA HISTÓRICA – CLASIFICACIÓN DE LOS COMPUTADORES	262
Bibliografía	265
Índice alfabético	267

Índice de figuras

Capítulo 1	33
1. Arquitectura Harvard: Bloques y Buses	35
2. Arquitectura von Neumann: Bloques y Buses	36
3. Ejercicio von Neumann.	39
4. Una etapa del circuito lógico.	41
5. Bloques del circuito aritmético.	42
6. Bloques de la UAL: etapa aritmética y etapa lógica.	44
7. Conexión final para el circuito aritmético-lógico.	45
8. Diseño con multiplexores de un circuito desplazador ad hoc.	45
9. Bloques que componen la Ruta de Datos diseñada.	48
10. Circuito del bloque de registros.	51
11. Tiempos de ejecución de la Ruta de Datos sin canalización.	54
12. Ruta de Datos: ejecución en canalización.	55
13. Traducción de la instrucción por la Unidad de Control	59
14. Unidad de Control cableada: decodificador.	61
15. CPU con control cableado.	62
16. Secuenciamiento explícito	65
17. Secuenciamiento implícito	67
18. Control microprogramado.CPU	69
19. Esquema de la división en etapas de una CPU para la ejecución en canalización.	71
Capítulo 2	77
1. Interrupción hardware vectorizada.	86
Capítulo 3	87
1. Secuencia de tratamiento	88
2. Llamada y retorno de subrutina.	100
3. Subrutinas anidadas.	101
4. Gestión del bloque de activación. Pasos en el programa llamador.	106

5. Gestión del bloque de activación. Pasos en la subrutina.	106
6. Gestión del bloque de activación. Pasos en la subrutina al término de la misma.	106
7. Gestión del bloque de activación. Pasos en el programa llamador al término de la subrutina.	107
8. Esquema de Ensamblado – Enlazado - Carga	112
9. Llamadas a DLLs	115
Capítulo 4	117
1. Arquitectura de bloques de un Procesador (CPU)	119
2. Tiempos de Acceso y Ciclo de Memoria	121
3. Esquema de una ROM	122
4. Esquema de una RAM	123
5. Célula y Direcciónamiento en una DRAM	124
6. Organización 2D	125
7. Organización 2 1/2 D	126
8. Organización 3D	126
9. Jerarquía de Memoria de cinco Niveles	127
10. Esquema de señales de una memoria	132
11. Célula básica de Memoria	132
12. Estructuración en Bloques de memoria	133
13. Ejemplo de Memoria de 64Kxn bits	134
14. Ejemplo de Memoria de 64Kx8 bits	134
15. Ejemplo de Memoria de 64K(32K+32K)x8 bits	135
16. Esquema de transferencia de Datos entre CPU-MC-MP	137
17. Organigrama de acceso a caché	139
18. Arquitecturas de interconexión entre MP-MC-E/S	143
19. Caché de Correspondencia Directa	145
20. Ejemplo Caché de Correspondencia Directa	146
21. Caché de Correspondencia Asociativa	147
22. Ejemplo Caché de Correspondencia Asociativa	148
23. Caché de Correspondencia Asociativa por Conjuntos	149
24. Ejemplo Caché de Correspondencia Asociativa por Conjuntos	150
25. Esquema de Memoria Compartida	155
26. Esquema de Memoria de Doble Puerto	156
27. Tratamiento de Memoria Virtual	160
28. Organización S	163

29. Ejemplo Organización S	163
30. Organización C	164
Capítulo 5	171
1. Conexión de Dispositivos en un Bus	172
2. Bus Síncrono	174
3. Bus Asíncrono	174
4. Bus Centralizado: Encadenamiento Circular de un Nivel	175
5. Bus Centralizado: Encadenamiento Circular de Dos Niveles	176
6. Bus Descentralizado: Arbitraje	177
7. Transferencia de un Bloque	178
8. Controlador de Interrupciones	178
9. Placa Base con Bus PC/AT (ISA)	180
10. Interconexión de Bus PCI con ISA	181
11. Ciclo de Bus PCI	183
12. Arbitraje en un Bus PCI	183
13. Bus VME	184
14. VME: Tarjetas Simple y Doble Europa	185
15. VME: Operación de Lectura o Escritura	186
16. Multibus II (PSB): Operación de Lectura en transferencia simple	189
17. NuBus: Esquema de bloques	189
18. NuBus: operaciones de: a) Lectura y b) Escritura	191
19. Futurebus: Esquema de bloques	192
20. Futurebus: Operación de Escritura	193
21. Secuencia de Datos en el Bus USB	196
22. Firewire Bus: Transferencias síncronas	198
23. Firewire Bus: Transferencias asíncronas y síncronas	198
Capítulo 6	201
1. Cabezas de Lectura/Escritura	202
2. Aspecto físico de un Disco Duro	202
3. Información sobre el Disco	203
4. Esquema de Platos y Cabezas en un Disco Duro	205
5. Raid nivel 0	209
6. Raid nivel 1	209
7. Raid nivel 2	209
8. Raid nivel 3	210

9. Raid nivel 4	210
10. Raid nivel 5	210
11. Pista de un CDROM. "Pits" y "Lands"	212
12. Empaquetamiento de Datos en un CDROM	212
13. Estructura de un CDROM grabable	214
14. Estructura de un DVD	215
15. a) Disco no Grabado, b) Operación de Escritura	216
16. a) Operación de Lectura, b) Operación de Borrado	216
17. Cabezal y Pistas en una Cinta Magnética	219
18. SSD-SATA III: Esquema de bloques	222
19. SSD-PATA: Esquema de bloques	224
Capítulo 7	225
1. Tubo de Rayos Catódicos	226
2. Vibración de la Luz en dos Planos	227
3. Pantallas Planas	228
4. Terminal de Mapa de caracteres	229
5. Conexión entre Ordenadores	231
6. Cronograma de Señales en un Ratón	232
7. Muestras de impresión por Agujas	234
8. Esquema de una Impresora Laser	236
9. Densidad de puntos para escala de Grises	236
10. Modulación en Amplitud – Frecuencia - Fase	240
Capítulo 8	245
1. Esquemas de Arquitecturas de Procesamiento Paralelo	247
2. Multiprocesadores conectados por un solo Bus	249
3. Multiprocesadores con Protocolo Snooping	250
4. Sincronización de Memoria	252
5. Multiprocesadores conectados por una Red	252
6. Multiprocesadores con Protocolo de Directorios	254
7. Coste y Rendimiento según arquitecturas	254
8. Red en Anillo	257
9. Red en Cuadrícula	257
10. Red en Cubo con N=3	257
11. Red en Cuadrícula Toroide	258
12. Red Crossbar	259

13. Red Omega	259
14. Red Crossbar con Memoria	259
15. Conmutador	261
16. Array de 6x6 Procesador-Memoria ejecutando la misma instrucción	263

Índice de tablas

Capítulo 1	33
1. Operaciones para nuestra CPU.	40
2. Tabla de las operaciones lógicas.	40
3. Entradas A, B y Cin y salidas del sumador para las diferentes combinaciones de S3 y S2.	41
4. Modificación del segundo sumando de la etapa aritmética.	42
5. Simplificación mediante Karnaugh de la función F_i que modifica el segundo sumando.	43
6. Codificación de las operaciones aritméticas y lógicas.	44
7. Las doce operaciones propuestas y la combinación de bits resultante.	46
8. Palabra de Control.	49
9. Interpretación de los 7 campos de la palabra de control.	49
10. Operaciones de control de secuencia.	52
11. Operaciones de la Ruta de Datos, codificación y mnemónicos.	53
12. Ejemplo de ejecución en canalización de la Ruta de Datos.	56
13. Formato de instrucciones de la CPU diseñada.	58
14. Tabla de las etapas de la canalización.	72
15. Ejemplo de ejecución de una computadora en canalización.	72
16. Riesgo de datos. Ejemplo.	73
17. Solución software al riesgo de datos. Ejemplo.	74
18. Solución hardware al riesgo de datos. Ejemplo.	75
Capítulo 2	77
1. Formatos de instrucción de la CPU diseñada.	78
2. Formatos de instrucción genérico.	79
3. Modos de direccionamiento. Ejemplo.	81
Capítulo 3	87
1. Estructura de un programa en Lenguaje Ensamblador con un μ P8085	91
2. Ejemplo con un μ P8088, 8086, ..., 80846	91

3. Ejemplo con un μ P68000, .., 68040	92
4. EQU	93
5. SET	93
6. DB	93
7. DW	93
8. ORG	94
9. PUBLIC	94
10. EXTERN	94
11. Direcciones de Bits, Datos o Código	94
12. Macro	95
13. Inserción de una Macro	95
14. Parámetros en las Macros	97
15. Pseudoinstrucciones	97
16. Otras pseudoinstrucciones	98
17. Etiquetas en Macros	99
18. Llamada a subrutina	101
19. Cuerpo de una Subrutina	102
20. Referencias adelante	108
21. Identificación de pseudoinstrucciones	108
22. Identificación de símbolos de variables	108
23. Identificación de macros	108
24. Identificación de subrutinas	108
25. Creación de la Tabla de Símbolos	109
26. Estructura de un Módulo Objeto	110
27. Ejemplo con cuatro Módulos Objeto	112
28. Cont. Ejemplo con cuatro Módulos Objeto	113
29. Cont. Ejemplo con cuatro Módulos Objeto	114
 Capítulo 4	117
1. Tamaños de memorias	120
2. Tipos de memoria	130
3. Contenido y Dirección en una Memoria	131
4. Señales de Control de una memoria	132
5. Estructuración en Bloques de la MP	137
6. Líneas en la MC	138
7. Correspondencias en memoria caché	143

8. Partición de la Dirección en campos en Correspondencia Directa	144
9. Ejemplo de estructuración de Bloques de Memoria	145
10. Partición de la Dirección en campos en Correspondencia Asociativa	147
11. Partición de la Dirección en campos en Correspondencia Asociativa por Conjuntos	148
12. Ejemplo de estructuración de Bloques de Memoria	150
13. Tabla comparativa de cachés	152
14. Ejemplo de memoria CAM	153
15. Relación de señales en una Memoria de Doble Puerto	156
16. Partición de la Dirección en Campos	158
17. Relación entre Memoria Principal y Secundaria	159
18. Descriptor de Página de Memoria Virtual	159
19. Evolución histórica de las DRAM	169
Capítulo 6	201
1. Tiempo de una rotación	206
2. Evolución del Bus SCSI	207
3. Velocidad de transferencia de discos <i>Blu-ray</i>	218
4. Capacidades de los SSD - Ejemplos	220
5. Velocidades de acceso en un SSD-SATA III	222
6. Velocidades de acceso en un SSD-USB	223
7. Velocidades de acceso en un SSD-PCI	223
8. Velocidades de acceso en un SSD-PATA	223
Capítulo 7	225
1. Señales entre DTE y DCE	230
2. Formato del Paquete de Datos de un Ratón	232
3. (a) HEXADECIMAL, (b) DECIMAL	242
Capítulo 8	245
1. Dirección en el Bus	253

PREFACIO

La sabiduría sirve de freno a la juventud, de consuelo a los viejos, de riqueza a los pobres y de ornato a los ricos. (Diógenes Laercio: 180 - 240 d.C)

Esto solo sé: que no sé nada (Sócrates: 469 - 399 a.C)

Grados recientes, ya consolidados, como el de Ingeniería Informática de Gestión y Sistemas de Información, y dada la orientación profesional que va a obtener el/la egresado/a para su incorporación en el mundo de la industria, requieren una revisión del enfoque de determinadas materias relativas a, p.ej., cómo está estructurado un computador, cual es su arquitectura más íntima, qué elementos lo conforman, cómo interactúan, etc.; dando al alumnado una visión amplia sin buscar la exhaustividad (propia de otras titulaciones) pero manteniendo un rigor tanto técnico como científico según la materia tratada.

El/la estudiante, tanto en su evolución académica en esta titulación así como en su futura vida profesional (que deseamos pronta y exitosa) se va a encontrar y centrar en diversos lenguajes y plataformas de programación, donde va a hacer realidad las especificaciones de diversos tipos de proyectos, no teniendo que entrar en ciertos aspectos de la estructura de la plataforma de desarrollo utilizada ni tampoco en cómo están estructurados los datos en un nivel próximo a la máquina, pero sí debe sentirse cómodo entre todos los dispositivos que le van a rodear. Esto se consigue sentando las bases correctas en esta etapa de su formación.

Atendiendo a lo anteriormente expuesto, el/la estudiante debe “vivenciar” en su fuero interno (a veces más deseo que realidad) cómo se mueve la información dentro de un computador, qué caminos sigue, donde se transforma y se almacena así como donde se visualiza y sobre qué tipo de soportes.

Este es el espíritu que ha guiado la elaboración del presente trabajo y esperamos que su lectura resulte amena y anime al lector/a a seguir profundizando en las diversas materias tratadas.

OBJETIVOS Y ESTRUCTURA DEL TRABAJO

Alcanzarás buena reputación esforzándote en ser lo que quieras parecer. (Sócrates: 469 - 399 a.C)

RESUMEN. Teniendo en cuenta la visión aportada en el Prefacio, procederemos a presentar los objetivos y la estructura de la presente guía.

La asignaturas: Estructura de Computadores y Arquitectura de Computadores, se imparten en el primer y segundo curso, respectivamente, del Grado de Ingeniería Informática de Gestión y Sistemas de Información. El carácter de estas asignaturas es obligatoria por lo que deberán cursarlas todo el alumnado matriculado en este grado. Los objetivos a alcanzar por esta asignatura, teniendo en cuenta el perfil que se desea que el alumnado obtenga, una vez finalizada la carrera, se pueden resumir en los siguientes puntos:

1. Objetivo General: será dotar al alumno/a de los conocimientos amplios y generales del funcionamiento de un computador.
2. Objetivos específicos:
 - a) Conocer la estructura de un ordenador y su funcionamiento.
 - b) Conocer los diversos periféricos que conforman un ordenador y su programación.
 - c) Conocer los diversos componentes que conforman un ordenador.
 - d) Familiarizarse con los lenguajes de bajo nivel de programación.
 - e) Familiarizarse con los conceptos de Ensamblado y Enlazado de programas.
 - f) Conocer las arquitecturas de Procesamiento Paralelo.

Desde un punto de vista docente, el programa propuesto y su carga aproximada, para la parte teórica, es el siguiente:

- Capítulo 1. Unidad Central de Procesos (8 horas)

- Capítulo 2. Formatos de Instrucción, Modos de Direccionamiento y Tratamiento de Interrupciones (3 horas)
- Capítulo 3. Lenguajes Máquina y Ensamblador (5 horas)
- Capítulo 4. Memorias (12 horas)
- Capítulo 5. Buses (4)
- Capítulo 6. Dispositivos de Almacenamiento Secundario (4 horas)
- Capítulo 7. Dispositivos de Entrada/Salida (3 horas)
- Capítulo 8. Arquitectura de Computación en Paralelo (5 horas)

El contenido de los capítulos mencionados es el siguiente:

- Glosario: Nomenclatura utilizada a lo largo del trabajo y términos de uso extendido.
- Capítulo 1: Se presentan los bloques que componen la arquitectura de un computador, fundamentalmente la Unidad Central de Procesos (UCP), o procesador, que incorpora la Unidad de Control (UC) y la Unidad Aritmético-Lógica (UAL). Se desarrolla paso a paso una UAL sencilla y se analizan las posibles opciones para la UC. Se describe el funcionamiento de una computadora en canalización así como los riesgos derivados de su implementación.
- Capítulo 2: Se expone el formato de instrucción y su relación con la arquitectura del conjunto de instrucciones. Se presentan los posibles modos de direccionamiento. Se presentan las interrupciones.
- Capítulo 3: Acerca al tratamiento digital, por el procesador, de la información que constituye lo que se denomina “programa en bajo nivel”. Un programa en bajo nivel está escrito siguiendo un repertorio de “mnemónicos” que substituyen al “lenguaje máquina” constituido, este, por ristas de unos (1) y ceros (0). Cada mnemónico tiene asociada alguna acción por parte del procesador. Se describen las macroinstrucciones y las subrutinas, analizándose las posibles alternativas para la gestión de las subrutinas.
- Capítulo 4: Presenta una visión del almacenamiento de la información en un computador, centrándose en los dispositivos primarios como son la memoria principal, la memoria caché y la memoria virtual, así como las diversas relaciones entre memorias. Se revisan diversas tecnologías y organización.
- Capítulo 5: Muestra la interacción entre la UCP y los dispositivos que conforman un Computador, así como las interacciones entre dispositivos. Ésta interacción se realiza a través de caminos eléctricos denominados BUSES.
- Capítulo 6: Desarrolla una visión de los dispositivos de almacenamiento secundario y sus diversas tecnologías. En estos nos encontramos con los discos magnéticos, ópticos, de estado sólido, etc

- Capítulo 7: Se realiza una visión de diversos dispositivos de Entrada/Salida (E/S) como puedan ser una impresora, un monitor, etc.
- Capítulo 8: Introduce el concepto de computación en paralelo como partición de un trabajo en diversas tareas a realizar por diversos procesadores. De manera genérica un procesador por tarea, de tal manera que el tiempo de cómputo se verá notablemente reducido. Se cita la ley de Amdahl para mostrar los límites de la mejora que aporta la computación en paralelo.
- Bibliografía: Muestra la Bibliografía consultada a lo largo del presente trabajo.

El programa propuesto para la parte práctica (Laboratorio) se explica en cuaderno aparte de esta publicación: Ejercicios de Laboratorio de la Asignatura Estructura de Computadores y Arquitectura de Computadores.

GLOSARIO

RESUMEN. En este capítulo se detallan todos los términos y acrónimos empleados a lo largo del presente trabajo, así como aquellos de uso extendido, con el fin de facilitar la lectura de todo lo expuesto.

TÉRMINOS y ACRÓNIMOS

- A/D:** Analógico/Digital
- ACK:** *Acknowledge*
- ALU:** *Arithmetic Logic Unit*
- ARM:** *Advanced RISC Machine* (Procesador embebido por Altera en sus dispositivos FPGA)
- ASCII:** *American Standard Code for Information Interchange*
- ASIC:** *Application Specific Integrated Circuits*
- BEDO:** *Burst Extended Data Output RAM*
- BIOS:** *Basic Input Output System*
- BUS:** Agrupación de Señales Eléctricas
- CAD:** *Computer-Aided Design*
- CAE:** *Computer-Aided Engineering*
- CAM:** *Content Addressable Memory*
- CD:** *Compact Disk*
- CDROM:** *Compact Disk Read Only Memory*
- CPLD:** *Complex PLD*
- CPU:** *Central Process Unit*
- CRC:** *Código de Redundancia Cíclica*
- CRT:** *Cathode Ray Tube*
- CTS:** *Clear To Send*
- D/A:** *Digital/Analógico*
- DCD:** *Data Carrier Detect*
- DCE:** *Data Comunication Equipment*
- DDRRAM:** *Double Data Rate DRAM*
- DMA:** *Direct Memory Access*
- DRAM:** *Dynamic RAM*
- DSM:** *Distributed Shared Memory*
- DSP:** *Digital Signal Processor*
- DSR:** *Data Set Ready*

- DTE:** *Data Terminal Equipment*
DTR: *Data Terminal Ready*
DVD: *Digital Video Disk*
E/S: *Entradas/Salidas*
ECC: *Error Correcting Code*
EDA: *Electronic Design Automation*
EDO: *Extended Data Output RAM*
EEPROM: *Electrically Erasable Programmable Read Only Memory*
EIA: *Electronics Industries Association*
EIDE: *Extended IDE*
EISA: *Extended Industry Standard Architecture*
EOT: *End Of Transmisión*
EPROM: *Erasable Programmable Read Only Memory*
FFD: *Flip-Flop D*
FIFO: *First In First Out*
FPD: *Field Programmable Device*
FPGA: *Field Programmable Gate Array*
FPMRAM: *Fast Page Mode RAM*
FSK: *Frequency Shift Keying*
GAL: *Generic Array Logic*
GND: *Ground*
GMT: *Grant*
HP: *Hewlett Packard*
HW: *Hardware*
IBM: *International Business Machines*
IDE: *Integrated Device Electronics*
INT: *Interrupt*
INTA: *Interrupt Acknowledge*
IP: *Intelectual Property*
IRQ: *Interrupt Request*
ISA: *Industry Standard Architecture*
LCD: *Liquid Crystal Display*
LED: *Light Emision Diode*
LFU: *Least Frecuently Used*
LRU: *Least Recently Used*
LUT: *Look-up Table*
μC: *Micro-Controlador*
μP: *Micro-Procesador*
MESI: *(MESI protocol) Modified Exclusive Shared Invalid*
MIMD: *Multiple Instruction Multiple Data*
MISD: *Multiple Instruction Single Data*
MODEM: *Modulador/Demodulador*
MSI: *Medium Scale Integration*
MSYN: *Master SYNchronitation*

- NACK:** *No Acknowledge*
- NIOS:** *National Institute of Open Schooling* (Procesador instanciado en dispositivos de Altera)
- OTP:** *One Time Programmable*
- PAL:** *Programmable Array Logic*
- PBSRAM:** *Pipeline Burst Static RAM*
- PC:** *Personal Computer*
- PCI:** *Peripheral Component Interconnect*
- PIO:** *Peripheral Input Output*
- PLA:** *Programmable Logic Array*
- PLB:** *Processor Local Bus*
- PLD:** *Programmable Logic Device*
- PPC:** *Power PC* (Procesador de IBM)
- QDR:** *Quad Data Rate SRAM*
- PROM:** *Programmable Read Only Memory*
- RAID:** *Redundant Array Inexpensive Disks*
- RAM:** *Random Access Memory*
- RDRAM:** *Rambus DRAM*
- REQ:** *Request*
- RISC:** *Reduced Instruction Set Computer*
- RTS:** *Request To Send*
- RWM:** *Read Writable Memory*
- S/H:** *Sample and Hold*
- SAM:** *Sequential Acces Memory*
- SCSI:** *Small Computer System Interface*
- SDRAM:** *Synchronous DRAM*
- SGRAM:** *Synchronous Graphics RAM*
- SISD:** *Single Instruction Single Data*
- SIMD:** *Single Instruction Multiple Data*
- SLDRAM:** *Synchronous Link DRAM*
- SPLD:** *Simple PLD*
- SoC:** *System on Chip*
- SOH:** *Start Of Head*
- SoPC:** *System on a Programmable Chip*
- SRAM:** *Static RAM*
- SSYN:** *Slave SYNchronitation*
- SW:** *Software*
- TN:** *Twisted Nematic*
- UAL:** *Unidad Aritmético-Lógica*
- UCP:** *Unidad Central de Procesos*
- USART:** *Universal Syncronous Asyncronous Receiver Transmpter*
- USB:** *Universal Serial Bus*
- VHDL:** *Very High Speed Hardware Description Language* (Lenguaje de Descripción Hardware)

VLSI: *Very Large Scale Integration*

VME: *VERSA Module Eurocard* o *Versa Module Europa*

VRAM: *Video DRAM*

ZBT: *Zero Bus Turnaround SRAM*

BIBLIOGRAFÍA-Observaciones

Más libros, más libres. (Enrique Tierno Galván: 1918 - 1986)

La Bibliografía disponible para asignaturas que traten temas como Arquitectura de Computadores, Estructura de Computadores, Diseño de Computadores, etc., es muy extensa. Debido a ello, el objeto de este apartado no es ofrecer una recopilación enciclopédica de títulos, sino más bien al contrario, nombrar y comentar exclusivamente los textos que han sido seleccionados para la confección del programa en sus diversos contenidos. Como ya se comentará, unos títulos han sido más determinantes que otros en la elaboración de esta extensa guía, pero no por ello deben dejar de consultarse los restantes porque siempre nos aportarán una visión o complemento a lo expuesto y enriquecerán nuestro bagaje técnico.

La selección se ha realizado atendiendo a diversos criterios, a saber: la adecuación de los contenidos del texto al programa propuesto, la adecuación del nivel de complejidad presentado por los contenidos, la disponibilidad de los volúmenes (algunos solo en biblioteca), la calidad pedagógica de la exposición de la materia y la posibilidad que ofrecen algunos textos para “profundizar y ampliar conocimientos”.

En varias de estas referencias se ha valorado también la manera de tratar los temas por parte de los autores, el enfoque y lenguaje utilizado motivarán que el alumnado se acerque con menos temor que el que inspiran normalmente los libros especializados.

- A. S. Tanenbaum. “Organización de Computadoras: Un Enfoque Estructurado”. Ed. Prentice-Hall [1]. Libro básico para esta asignatura, trata todos los temas con rigor, profusión y amenidad.
- D. A. Paterson, J. L. Hennessy. “Estructura y Diseño de Computadores”. Vol. 2 y 3. Ed. Reverté, S.A. 2004 [2, 3, 4]. Estos dos volúmenes tratan los temas de la asignatura con rigor, son el complemento del anterior. El volumen 3 resulta especialmente relevante en el tema “Lenguaje Máquina y Ensamblador”. Tratamiento de

las memorias Caché muy didáctico. Los tres volúmenes no deberían faltar en nuestra biblioteca personal/profesional.

- J.L. Hennessy, D.A. Patterson. “Arquitectura de Computadores: Un Enfoque Cuantitativo”. Ed. McGraw-Hill. 2002 [5]. Libro de los mismos autores que en un solo volumen tratan los tres con menor alarde gráfico pero con gran rigor. Muy buen tratamiento de aspectos relacionados con la memoria Caché. Libro también básico.
- M. Morris Mano, C. R. Kime. “Fundamentos de Diseño Lógico y Computadoras”. Ed. Prentice-Hall. 3^a Edición. 2005 [6]. Énfasis en el diseño Digital de un Computador y el tratamiento de las instrucciones a nivel máquina. Excelente consulta para la memoria Virtual y Caché.
- I. Englander. “Arquitectura Computacional”. Ed. Cecsa. 2^a Ed. 2002 [7]. Complementa aspectos y enfoque. Lectura amena. Consulta general.
- Cisco Networking Academy Program. “HP Fundamentos de Tecnología de la Información: Hardware y Software para PC”, Ed. Pearson Educación , S.A. 2005 [8]. Ofrece una visión muy completa sobre la organización de un computador así como sobre los sistemas operativos basados en Windows. Consulta general.
- B.B. Brey. “Los Microprocesadores Intel”. Ed. Prentice-Hall. Visión del mundo “Intel” [9]. Imprescindible para el diseñador de computadores en base a CPU’s de Intel.
- M. A. de Miguel, T. Higuera. “Arquitectura de Ordenadores”. Ed. Ra-Ma [10]. Arquitectura basada en el entorno “Motorola”. Tratamiento de la memoria Virtual muy didáctica.
- L. Null & J. Lobur. “The essentials of computer organization and architecture”. Ed. Jones and Bartlett. 2^a edición. 2006 [11]. El contenido de este libro es más amplio que profundo: describe desde conceptos previos hasta conceptos avanzados, así como los aquí tratados; todo de un modo muy claro y con interesantes tips, ejemplos y ejercicios.
- X. Alcober. “Buses normalizados para tarjetas basadas en μ P”. Automática e Instrumentación. 1988 [12]. Visión histórica pormenorizada de los Buses paralelo, ampliamente descritos.
- P. de Miguel-Anasagasti. “Fundamentos de los Computadores”. Ed. Thomson-Paraninfo 2004 [13]. Libro generalista de asignaturas sobre computación.

Capítulo 1

UNIDAD CENTRAL DE PROCESOS

Un ordenador es para mí la herramienta más sorprendente que hayamos ideado. Es el equivalente a una bicicleta para nuestras mentes. (Steve Jobs: 1955 - 2011)

RESUMEN. Se dice que la **Unidad Central de Proceso** (o CPU por sus siglas en inglés) es el corazón, el cerebro, el núcleo de un ordenador. Ésta se divide en dos partes de características bien distintas: la **Ruta de Datos**, encargada de ejecutar operaciones aritméticas y lógicas en base a las señales de control recibidas; y la **Unidad de Control**, la parte “inteligente” encargada de descifrar cada una de las instrucciones de un programa, y generar las señales de selección que dirigirán el funcionamiento de la ruta de datos.

La Ruta de Datos comprende la Unidad Aritmético-Lógica (UAL) o Unidad de Funciones y el bloque de registros de la CPU. La UAL está formada por elementos combinacionales que realizan operaciones aritméticas y lógicas, en las que las señales de control definen la ejecución de la operación.

La Unidad de Control es la parte de la computadora encargada de traducir para la Ruta de Datos la instrucción almacenada en memoria. La Unidad de Control conoce el tipo de instrucciones de su arquitectura concreta (qué longitud tienen, en qué campos se dividen,...), y en base a ello genera la palabra de control necesaria. Ésta contiene todas las señales de control que determinan el funcionamiento de la Ruta de Datos, pero también las indicaciones que la propia Unidad de Control necesita para una correcta secuenciación.

Con el objetivo de ver de una manera amena y didáctica la importancia y la interrelación de los elementos que componen la CPU, en este capítulo diseñaremos nuestra propia Ruta de Datos y analizamos distintas opciones para la Unidad de Control. Para terminar se describe la ejecución en canalización o pipeline, sus características, ventajas y problemática asociada.

1.1. INTRODUCCIÓN

El computador actual es el resultado de las aportaciones de personas pertenecientes a diferentes ámbitos a lo largo de muchos años. Aunque durante la Segunda Guerra Mundial las necesidades militares como los cálculos de trayectorias de proyectiles o el cifrado/descifrado de mensajes hicieron que se desarrollaran máquinas tan importantes y conocidas como ENIAC, ENIGMA o COLOSSUS, no es hasta mediados del siglo XX que se considera la primera generación de computadores. Éstos, basados en válvulas de vacío, eran voluminosos, propensos a fallos y consumían mucho.

La sustitución de las válvulas de vacío por transistores dio lugar a la segunda generación de computadores durante la primera mitad de la década de los 60. Los circuitos integrados marcan la tercera generación de los computadores, durante la segunda mitad de la década de los sesenta, ya que, aunque la idea del circuito integrado es muy anterior, es con la aparición de la serie 360 de IBM que se extiende su utilización.

El siguiente paso en la miniaturización nos lleva al microprocesador, hito que marca la cuarta generación en la década de los setenta. De aquí en adelante se hace difícil llegar a consensos a la hora de delimitar sucesivas generaciones, dado que no hay hitos tan claros como los anteriores, que eran cambios físicos que suponían un gran avance en tiempo, consumo y tamaño. Centrémonos ahora en el análisis de las partes constituyentes de un computador.

1.2. ARQUITECTURAS

1.2.1. Arquitectura Harvard.

La arquitectura Harvard¹ se basa en la arquitectura de la computadora Mark I, computadora de propósito general diseñada en la universidad de Harvard y presentada en 1944. Funcionaba en base a relés (componente electromecánico), memorias de ferrita, etc. Es utilizada durante la Segunda Guerra Mundial. Las características principales de esta arquitectura son: memorias y buses² separados para instrucciones y datos; lo que permite un acceso simultáneo a datos e instrucciones. La Figura 1 muestra los bloques de los que se compone un computador de arquitectura Harvard.

¹ Su diseño se debe a Howard H. Aiken quien, en un rasgo de modestia, le dio el nombre de la universidad en la que trabajaba.

² El Capítulo 5 trata con más profusión el significado de los Buses como caminos de señales eléctricas

La parte izquierda muestra el bloque de memoria que contiene las Instrucciones y los buses que lo relacionan con la CPU, estos son: Bus de Instrucciones **IB[...]** (*Instruction Bus*), Bus de Direcciones **AIB[...]** (*Address Instruction Bus*) y Bus de Control **CIB[...]** (*Control Instruction Bus*). La parte derecha muestra el bloque de memoria que contiene los Datos, la conexión con el mundo exterior (Entradas/Salidas) y los buses que relacionan la CPU con ellos, estos son: Bus de Datos **DB[...]** (*Data Bus*), Bus de Direcciones **ADB[...]** (*Address Data Bus*) y Bus de Control **CDB[...]** (*Control Data Bus*).

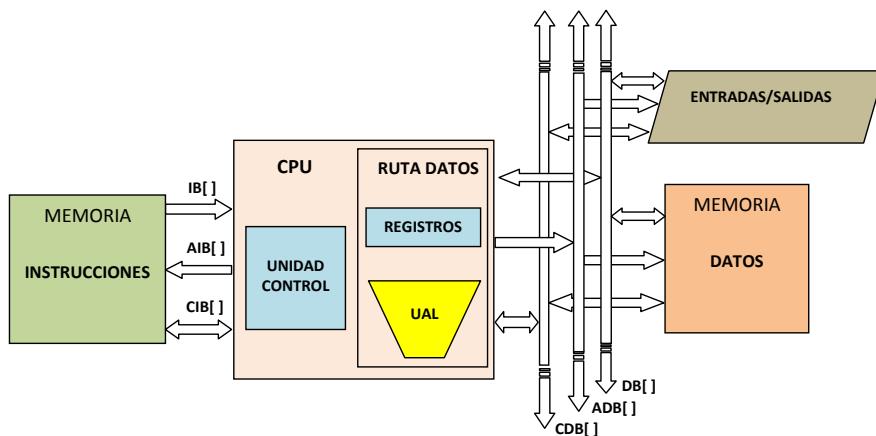


FIGURA 1. Arquitectura Harvard: Bloques y Buses

1.2.2. Arquitectura von Neumann.

La arquitectura von Neumann³ se basa en la descrita en 1945 por el matemático húngaro-estadounidense Jon von Neumann, del que toma su nombre, y es la que se ve en la Figura 2.

³A diferencia de Aiken, no se la conoce con el nombre de Princeton, universidad donde se desarrolló el trabajo, si no con el de su principal autor.

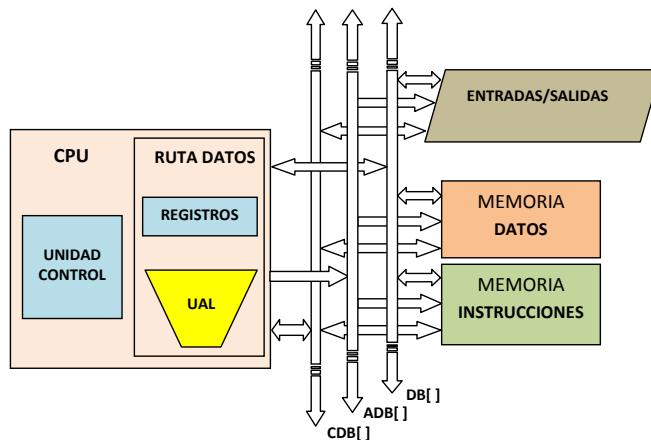


FIGURA 2. Arquitectura von Neumann: Bloques y Buses

Los computadores del momento requerían que tanto los datos como los programas se introdujesen manualmente. Cada nuevo programa requería que un operario recablease el computador. El reto de evitar esta necesidad dio lugar a computadores con programa almacenado. Es precisamente aquí, en el momento de decidir cómo se almacenarán tanto los datos como las instrucciones en el que la opinión de von Neumann dio lugar a la principal característica de esta arquitectura: una única memoria y un único bus para instrucciones y datos.

La Figura 2 muestra los bloques que componen un computador con arquitectura von Neumann. La parte derecha muestra el bloque de memoria que contiene los Datos e Instrucciones, la conexión con el mundo exterior (Entradas/Salidas) y los buses que relacionan la CPU con ellos, estos son: Bus de Datos **DB[...]** (*Data Bus*: mueve Datos e Instrucciones), Bus de Direcciones **ADB[...]** (*Address Data Bus*) y Bus de Control **CDB[...]** (*Control Data Bus*).

1.2.3. Harvard vs. von Neumann.

La arquitectura von Neumann, como ya se ha reflejado en 1.2.2, se basa en un único espacio de memoria y un único bus para instrucciones y datos; lo que imposibilita leer una instrucción mientras se ejecuta otra. Por otra parte, todas las instrucciones, incluso las de movimiento de datos, han de pasar por la UAL, provocando un cuello de botella. Las características principales de la arquitectura Harvard son las que la contraponen a la de von Neumann: memorias y buses separados para instrucciones y datos; lo que permite un acceso simultáneo a datos e instrucciones y, por lo tanto, mayor velocidad de procesamiento.

A pesar del cuello de botella que dificulta la programabilidad e impide el paralelismo, la arquitectura von Neumann es la más extendida aún hoy en día, debido por un lado, al amplio conjunto de programas desarrollados para esta arquitectura y por otro a la complejidad añadida de la arquitectura Harvard, que la hace rentable sólo en casos concretos, en los que el flujo de instrucciones y de datos es comparable. Hoy en día los procesadores más rápidos se basan en una arquitectura Harvard modificada.

1.3. ELEMENTOS CONSTITUTIVOS DE LA CPU

Para poder entender mejor la estructura de un computador, la importancia de cada uno de sus componentes, la tarea que éste realiza, sus limitaciones etc. vamos a desarrollar nuestra propia CPU.. Paso a paso iremos viendo las distintas necesidades y eligiendo entre las posibles soluciones. Comencemos por lo básico: ¿qué hace un computador y qué necesita para ello? Básicamente un ordenador ha de ejecutar instrucciones y devolver resultados. Estas instrucciones pueden formar parte de un programa almacenado en memoria o pueden ser introducidas por el usuario.

- **Ejecución de instrucciones:** El computador necesitará una unidad funcional, capaz de realizar operaciones aritméticas, lógicas y desplazamientos (la llamaremos **Unidad Aritmético-Lógica**, UAL⁴). Los datos para las operaciones deberán guardarse en registros, por lo que serán necesarios unos cuantos registros, un bloque de registros. Se denomina Ruta de Datos al conjunto de los elementos de la UAL más el bloque de registros. La **Ruta de Datos** estará por lo tanto formada por registros, multiplexores, un sumador/restador, un desplazador,... Pero estos son elementos combinacionales, que darán un resultado en función de sus valores de entrada y señales de selección. Dichas señales las generará la **Unidad de Control**, UC⁵, la encargada de “interpretar” cada una de las instrucciones. El conjunto formado por la UAL y la UC es la **Unidad Central de Proceso**, CPU⁶ por sus siglas en inglés.
- **Almacenamiento de las instrucciones:** Las instrucciones pueden ser parte de un programa almacenado en memoria, necesitamos por lo tanto una **memoria**. Suponiendo una ejecución secuencial, o *Batch*, dicho programa estará formado por una secuencia de instrucciones que han de ejecutarse una detrás de otra, salvo en los casos en los que se dé un salto o una bifurcación... De modo que se hace necesario un secuenciamiento en el acceso a memoria. La

⁴En inglés ALU (Arithmetic Logic Unit)

⁵En inglés CU (Control Unit)

⁶CPU (Central Process Unit)

Unidad de Control deberá pues incluir la lógica que permita saber en cada momento dónde está, en memoria, la siguiente instrucción.

- **Resultados de las operaciones:** En cuanto a los resultados de las operaciones, aunque a veces deberán almacenarse en memoria, es imprescindible tener también una forma de extraerlos, ya que no sirven de mucho si no podemos verlos. El computador necesita poder mostrar los resultados; lo hará a través de dispositivos de salida (pantalla, impresora,...); pero también ha veces ha de recoger datos introducidos por el usuario o eventos que determinarán el flujo del programa (en el caso de programación dirigida por eventos). Lo hará mediante dispositivos de entrada (teclado, micrófono,...). Es decir, el computador necesita **dispositivos de entrada/salida**.

La Figura 2 muestra los bloques que forman un computador.

La Unidad Central de Proceso (CPU por sus siglas en inglés) es el núcleo de un computador, en ella se concentran la capacidad tanto de interpretación como de ejecución de las instrucciones.

Los elementos necesarios para tan fundamentales tareas son:

- **La Ruta de Datos:** compuesta esencialmente por un sumador, un desplazador y puertas lógicas (UAL), realiza las operaciones aritméticas, lógicas y los desplazamientos indicados por las señales de control que gobiernan su funcionamiento. Los datos que utiliza, así como los resultados de sus operaciones, se almacenan (a excepción de las operaciones de transferencia con la memoria), en unos registros cercanos a la UAL, denominado **Bloque de Registros**. El conjunto UAL-Bloque de Registros forman la Ruta de Datos.
- **La Unidad de Control:** la UAL está compuesta por elementos combinacionales, incapaces de tomar decisiones. Necesitan por lo tanto que el conjunto de bits en los que se codificó una cierta instrucción se transforme a un conjunto aún mayor de '0's y '1's, formado por todas aquellas señales de control/selección que rigen el funcionamiento de la UAL, la denominada **Palabra de Control**. La encargada de traducir una instrucción en código máquina a la palabra de control necesaria para esa arquitectura concreta, será la Unidad de Control.

1.3.1. Ejercicios.

- Rellenad los espacios numerados:

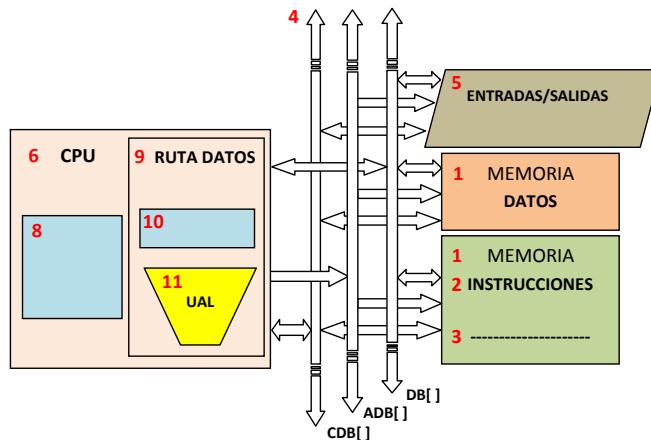


FIGURA 3. Ejercicio von Neumann.

- Enumerad las principales características de la arquitectura von Neumann.
- ¿Qué es la palabra de control? ¿Quién la utiliza y para qué?
- ¿Quién genera la palabra de control? ¿A partir de qué información?

1.4. RUTA DE DATOS

Tal como se adelantaba, la Unidad Aritmético-Lógica está formada por elementos combinacionales que realizan operaciones aritméticas y lógicas, en las que las señales de control definen la ejecución de la operación. Para comprender la importancia, la relación y el funcionamiento de los elementos que la componen, vamos a diseñar nuestra propia CPU.

1.4.1. Unidad aritmético-lógica.

Supongamos que queremos desarrollar una UAL que realice las operaciones recogidas en la Tabla 1. Para facilitar la comprensión vamos a desarrollar una UAL con unas pocas operaciones básicas. Diseñar una CPU comparable con las que tienen los ordenadores actuales supondría añadir mucha complejidad, dificultando la comprensión.

TABLA 1. Operaciones para nuestra CPU.

Operaciones	
Aritméticas	Decremento, incremento
	Suma, resta
Lógicas	NOT, AND, OR, XOR
Movimiento	$(RD) \leftarrow (RB)$ (1)
Desplazamiento	Hacia la izquierda (1 bit)
	Hacia la derecha (1 bit)
Borrado	$(RD) \leftarrow 0$ (2)
Control de secuencia	Salto/bifurcación incondicional
	Salto/bifurcación condicionada a cero

(1) El contenido del Reg. B pasa al Reg. D

(2) El contenido del Reg. D pasa a ser “0”

1.4.2. Circuito lógico.

Comenzamos por el diseño del circuito lógico, es decir, la parte encargada de realizar las microoperaciones lógicas. Antes de nada recordemos una cosa: el resultado de la operación $0100 + 0101$ será 1001 para una operación aritmética (SUMA), y 0101 para una operación lógica (OR); es decir, las operaciones lógicas son operaciones bit a bit. Por lo tanto, en este punto trabajaremos con una etapa genérica (la etapa $i - \text{ésima}$) del circuito lógico, sin olvidar que para cada operando de n bits, deberá haber n etapas como ésta. Mirando la Tabla 1 vemos que las operaciones lógicas deseadas son cuatro, necesitaremos por lo tanto dos bits para codificarlas. La Tabla 2 muestra las cuatro operaciones lógicas y la combinación de los bits S_1 y S_0 que las identifican.

TABLA 2. Tabla de las operaciones lógicas.

S_1	S_0	Salida	Operación
0	0	$A \cdot B$	AND
0	1	$A + B$	OR
1	0	$A \oplus B$	XOR
1	1	\bar{A}	NOT

Para cada etapa del circuito lógico utilizaremos por lo tanto cuatro puertas lógicas: una *AND*, una *OR*, una *XOR* y una *NOT*. Estas puertas tendrán a la entrada el i -ésimo bit de los datos A y B , y sus salidas irán a un multiplexor de 4 entradas. Los dos bits que identifican la operación serán las

señales de selección del multiplexor. En la Figura 4 se representa el circuito correspondiente a las cuatro operaciones lógicas descritas.

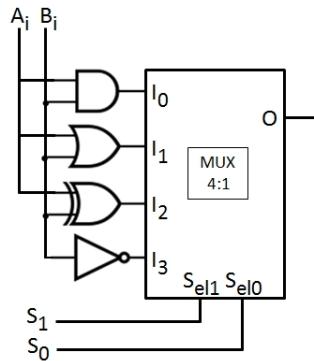


FIGURA 4. Una etapa del circuito lógico.

1.4.3. Circuito aritmético.

El circuito aritmético debe realizar sumas y restas, lo diseñaremos por lo tanto utilizando un sumador completo de n bits (en realidad serán varios sumadores conectados). Siendo A y B números de n bits, vamos a suponer que el operando A entra tal cual al sumador, y que es el operando B el que necesita, en función de la operación a realizar, ser modificado de alguna manera. Teniendo en cuenta que en la entrada A del sumador irá el dato A y que las restas se realizarán complementando a dos el minuendo (recordemos que el complemento a dos de B es $\bar{B} + 1$, y el numero decimal -1 en complemento a dos es una secuencia de n '1's), la cuestión es cómo deberán ser las entradas B y C_{IN} del sumador para realizar las operaciones $A - 1$, $A + 1$, $A + B$ y $A - B$. La respuesta y los bits para codificar las cuatro operaciones se recogen en la Tabla 3.

TABLA 3. Entradas A, B y Cin y salidas del sumador para las diferentes combinaciones de S_3 y S_2 .

S_3	S_2	A	B	C_{IN}	Salida
0	0	A	todo unos	0	$A - 1$
0	1	A	todo ceros	1	$A + 1$
1	0	A	B	0	$A + B$
1	1	A	\bar{B}	1	$A - B$

Por lo tanto el dato A entrará sin modificar al sumador, pero es necesario añadir una lógica que modifique el dato B en función de las señales S_3 y S_2 .

tal y como se recoge en la Tabla 3. Es decir, tenemos el circuito de la Figura 5.

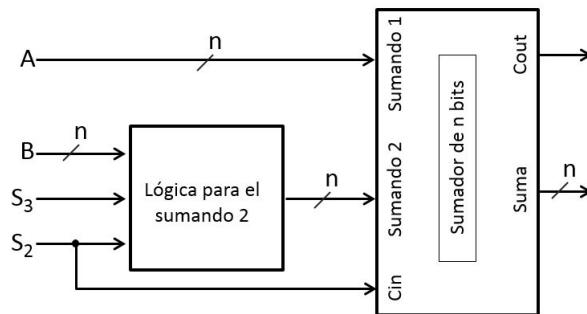


FIGURA 5. Bloques del circuito aritmético.

El siguiente paso es diseñar la lógica del bloque *Lógica para el sumando 2*.

Una opción para la obtención del segundo sumando es utilizar un multiplexor de 4 a 1. Sus dos entradas de selección pondrán en la salida una de las entradas, y en cada una de las entradas conectaremos unos lógicos (V_{CC}), ceros lógicos (GND), B o B negada (\bar{B}). Esta solución sería análoga a la utilizada en la etapa lógica, por lo que aquí veremos otra alternativa (queda en manos del lector implementar la solución con multiplexor). Vamos a diseñar el bloque *Lógica para el sumando 2* a partir de su función simplificada. Primeramente rellenemos la Tabla 4, tabla de la verdad del bloque que se quiere diseñar, para las entradas S_3 , S_2 y **un único bit del dato B**.

TABLA 4. Modificación del segundo sumando de la etapa aritmética.

S_3	S_2	B_i	$F_i(*)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(*) Bit i – ésmo del segundo sumando

Simplificando la función para el segundo sumando (el dato B modificado que entra al sumador) mediante Karnaugh (ver Tabla 5), nos queda que la lógica dentro de la caja que hemos denominado “Lógica para el sumando 2” ha de ser: $F_i = \bar{S}_3 \cdot \bar{S}_2 + S_3 \cdot (S_2 \oplus B_i)$, donde B_i representa el i -ésimo elemento del número B .

TABLA 5. Simplificación mediante Karnaugh de la función F_i que modifica el segundo sumando.

		S_3S_2			
		00	01	11	10
B_i	0	1	0	1	0
	1	1	0	0	1

$$\bar{S}_3 \cdot \bar{S}_2 \quad S_3 \cdot S_2 \cdot \bar{B}_i \quad S_3 \cdot \bar{S}_2 \cdot B_i$$

Llegados a este punto es el momento de unir la etapa del circuito lógico con el aritmético. Cada bit de los datos de entrada irá tanto a uno como a otro, y las salidas de ambos estarán siempre disponibles, de modo que necesitaremos otro bit de selección (S_4) para elegir en cada momento la salida de uno u otro. La Figura 6 muestra el esquema de bloques de la UAL diseñada.

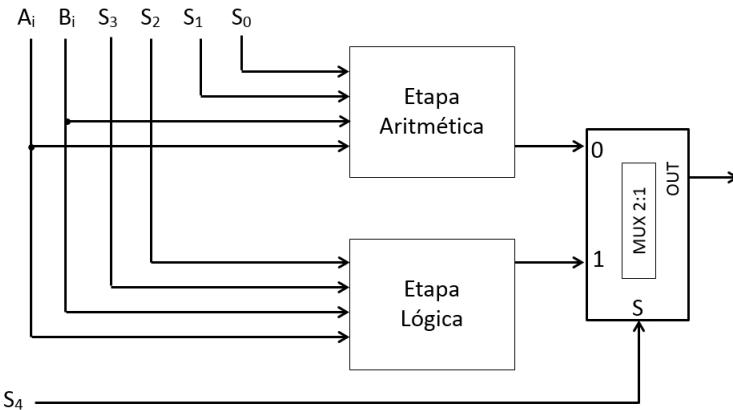


FIGURA 6. Bloques de la UAL: etapa aritmética y etapa lógica.

Así, combinando una etapa lógica y una aritmética tendríamos una etapa de la UAL (recordemos que para operandos de n bits, habrá n etapas lógicas). Sin embargo fijémonos en que hemos utilizado 5 variables (señales de selección $S[4..0]$) para 8 operaciones (4 aritmética y 4 lógicas), cuando para representar 8 combinaciones es suficiente con tres bits. Reordenando las conexiones para utilizar sólo 3 señales de selección, obtenemos la Tabla 6 y el circuito de la Figura 7. De esta manera las señales S_1 y S_0 se utilizan tanto en la etapa aritmética como en la lógica, dado que sus salidas se conectan a un multiplexor en el que mediante S_2 se seleccionará solo una u otra.

TABLA 6. Codificación de las operaciones aritméticas y lógicas.

S_2	S_1	S_0	Operación
0	0	0	$A - 1$
0	0	1	$A + 1$
0	1	0	$A + B$
0	1	1	$A - B$
1	0	0	$NOT(A)$
1	0	1	$(A)AND(B)$
1	1	0	$(A)OR(B)$
1	1	1	$(A)XOR(B)$

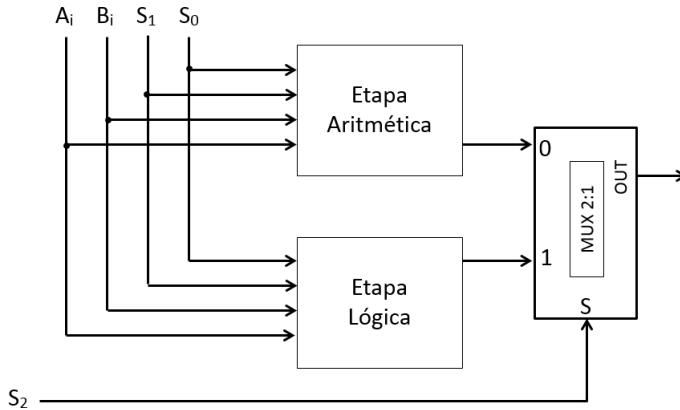


FIGURA 7. Conexión final para el circuito aritmético-lógico.

1.4.4. Desplazador.

Volviendo a la Tabla 1, vemos que ya tendríamos las operaciones aritméticas y lógicas, de modo que, además de la instrucciones de control de secuencia, nos faltarían las de transferencia entre registros (del registro B al D), desplazamiento de 1 bit a la derecha e izquierda, borrado de un registro (del registro D). Para los desplazamientos podríamos añadir un registro de desplazamiento universal, que permita hacer un desplazamiento de 0 bits o de un bit para uno u otro lado. Sin embargo aquí veremos otra solución que nos permite implementar nuestro desplazador bidireccional de un bit dando a la vez solución a la instrucción de borrado de registro.

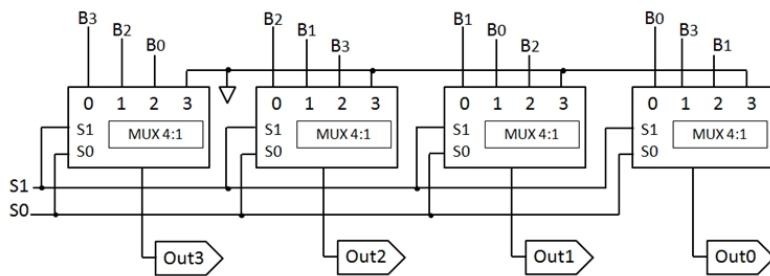


FIGURA 8. Diseño con multiplexores de un circuito desplazador ad hoc.

Para ello utilizaremos n multiplexores de 4 a 1. Realizaremos las conexiones de manera que según sean las entradas de selección la salida sea una de las siguientes: el dato sin modificar, desplazado un bit hacia uno u otro lado, o en ceros. La Figura 8 muestra la implementación para el caso

$n = 4$. Utilizando de nuevo S_1 y S_0 para controlar el funcionamiento de los multiplexores, la salida será: 00- el dato sin modificar, 01-desplazado un bit hacia la izquierda, 10- desplazado un bit hacia la derecha y 11- todo ceros.

A falta de las operaciones de control de secuencia, ya tenemos las 12 operaciones que realiza nuestra UAL: 4 aritméticas, 4 lógicas, 1 de transferencia entre registros, 2 de desplazamiento y una de borrado de registro. La Tabla 7 recoge las operaciones con su codificación binaria y su descripción.

TABLA 7. Las doce operaciones propuestas y la combinación de bits resultante.

$S_3S_2S_1S_0$	Operación	Descripción
0 0 0 0	DCR	A-1
0 0 0 1	INR	A+1
0 0 1 0	ADD	A+B
0 0 1 1	SUB	A-B
0 1 0 0	NOT	NOT (A)
0 1 0 1	AND	(A) AND (B)
0 1 1 0	OR	(A) OR (B)
0 1 1 1	COR	(A) XOR (B)
1 x 0 0	MOV	Copiar un registro a otro
1 x 0 1	sl B	Rotar una posición a la izqda.
1 x 1 0	sr B	Rotar una posición a la dcha.
1 x 1 1	CLR	Borrar un registro

Llamaremos $OS[3..0]$ (*Operation Select*) al conjunto de las cuatro señales de selección utilizadas. Vemos que aún quedan cuatro combinaciones, son las que utilizaremos más adelante para codificar las cuatro operaciones de control de secuencia.

1.4.5. Palabra de control.

Al final del punto anterior ya teníamos los cuatro bits que definen la operación a realizar, pero esto no es suficiente para dirigir el funcionamiento de la UAL, son muchas más las señales necesarias.

Para empezar, además de la operación, se le debe indicar cuáles son (dónde están) los operandos (A y B) con los que ha de realizarla. Puede ser que se encuentren en el bloque de registros, en cuyo caso se necesitan señales de control que seleccionen (direccíonen) los registros que contienen los operandos que han de cargarse en los dos buses de entrada a la UAL (los llamaremos A y B)..

Suponiendo un pequeño bloque de registros formado por 8 registros, necesitaremos las señales $AS[2..0]$ y $BS[2..0]$, de tres bits, que permiten seleccionar el registro cuyo contenido se cargará en el bus A y B respectivamente. En ciertos casos, interesa operar con un dato indicado por el usuario, por ejemplo cuando queremos inicializar un registro, o sumar o restar un valor concreto. En este caso, en el que utilizaremos instrucciones “de operando inmediato”, el valor vendrá dado como parte de la propia instrucción. Este valor deberá estar disponible, junto con el obtenido de un registro, a la entrada de un multiplexor (lo llamaremos mux B), de modo que otra señal (MB) pueda elegir uno de los dos valores como entrada a la UAL..

En cualquier caso la UAL generará un resultado que pasará al bus D, y que por lo general, deberá guardarse en un registro, por lo que, análogamente a AS y BS, será necesaria otra señal de 3 bits, $DS[2..0]$, para seleccionar el registro en el que se guardará el resultado de la operación. Por último la UAL activará en caso necesario el indicador (flag, bit) de estado Z, que indica que el resultado de la operación ha sido igual a cero (si el resultado ha sido cero $Z=1$, en caso contrario $Z=0$). Aunque en nuestro ejemplo no hay más que uno, por lo general existirán varios flags para indicar la llevada, el signo, la paridad, etc. del resultado. Estos indicadores permiten realizar operaciones de control de secuencia condicionadas. Pensemos en el ejemplo de un fragmento de código que ha de repetirse 10 veces, este código puede escribirse dentro de un bucle en el que decrementamos un contador, cada vez, tras la operación de decremento del contador, se comprueba el valor del flag Z; la siguiente instrucción será diferente si el resultado ha sido cero o no. Esto no sería posible sin los flags de estado y las instrucciones de salto y bifurcación condicionadas.

Pero sigamos con las señales necesarias para el correcto funcionamiento de nuestra Ruta de Datos. Si queremos que nuestra Ruta de Datos pueda realizar operaciones de transferencia de datos entre el registro y la memoria y viceversa, es necesario, por un lado, permitir la salida desde un bus que tenga el dato de un registro (bus A o B) hacia memoria; por otro lado, es necesario añadir otro multiplexor (lo llamaremos D) en cuyas entradas conectaremos el resultado de la operación o un dato extraído de memoria. De modo que, además de la señal de 4 bits $OS[3..0]$ que indica la operación (ver la Tabla 7), es necesaria otra señal de selección (**MB**).

Supongamos el caso de una instrucción que no realiza ninguna operación en la Unidad de Funciones, sólo mueve un dato de registro a memoria. En este caso el destino del resultado de la operación no está en el bloque de registros, sin embargo la señal $DS[2..0]$ siempre tendrá un valor (aunque sea 000), lo que sobreescribiría el valor guardado en un registro. Para evitar esta

escritura indeseada, se añade la señal **RW** (Register Write), que deberá estar activada para habilitar la escritura en el registro seleccionado por $DS[2..0]$.

De manera análoga, los buses A y B así como los bits de la señal OS siempre tendrán algún valor, también cuando la instrucción indique la lectura de un valor de memoria. En este caso, la señal **MD** valdrá 1 (ver Figura 9), de modo que el dato traído de memoria se cargará en el registro indicado por DS (siempre que RW sea 1), independientemente de la operación que se haya realizado en el UAL y de los operandos en ella implicados.

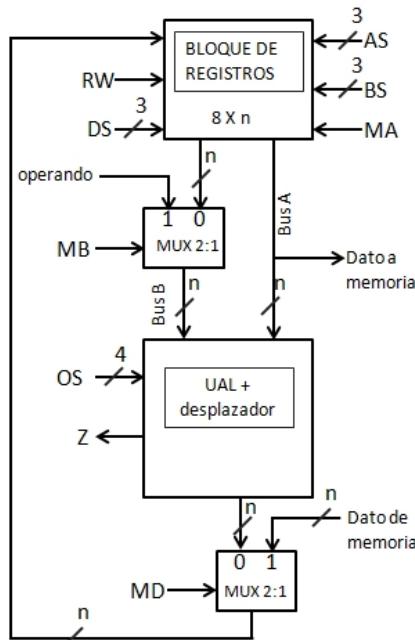


FIGURA 9. Bloques que componen la Ruta de Datos diseñada.

La Figura 9 representa el diagrama de bloques de la Ruta de Datos resultante. Haciendo recuento de las señales necesarias para el control de la Ruta de Datos obtenemos un total de 16 (en la Figura 9 hay 17, pero veremos que podemos prescindir de la señal MA). La **palabra de control** es el conjunto de las señales de selección que controlan el funcionamiento de la Ruta de Datos (ver Tabla 8). Es específica de una arquitectura, ya que depende de las instrucciones que se puedan ejecutar, de cómo se codifiquen éstas y de la manera de ordenar los bits. Está formada por campos de diferente longitud.

TABLA 8. Palabra de Control.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AS		BS		DS		M B	OS		M D	R W					

La Tabla 9 recoge la codificación, para la correcta interpretación, de la Palabra de Control.

TABLA 9. Interpretación de los 7 campos de la palabra de control.

SA, SB, SD	MB	OS		MD		RW			
R0	0 0 0	R	0	0 0 0 0	DCR	Resultado	0	No escribir	0
R1	0 0 1	Dato	1	0 0 0 1	INR	Dato	1	Escribir	1
R2	0 1 0			0 0 1 0	ADD				
R3	0 1 1			0 0 1 1	SUB				
R4	1 0 0			0 1 0 0	NOT				
R5	1 0 1			0 1 0 1	AND				
R6	1 1 0			0 1 1 0	OR				
R7	1 1 1			0 1 1 1	XOR				
				1 x 0 0	MOV				
				1 x 0 1	sl B				
				1 x 1 0	sr B				
				1 x 1 1	CLR				

1.4.6. Bloque de registros.

La implementación del bloque de registros es muy sencilla: tendremos 8 registros de n bits, cuyas salidas estarán disponibles en las entradas de dos multiplexores; las señales de selección de dichos multiplexores estarán unidas a las señales de 3 bits $AS[2..0]$ y $BS[2..0]$.

Realmente, cada uno de estos multiplexores estará compuesto por n multiplexores de 8 a 1, con señales de selección comunes. Sus salidas se cargarán en los buses de n bits A y B. En cuanto al bus D, su contenido ha de cargarse en el registro seleccionado por $DS[2..0]$, solo en caso de que así lo indique RW ($RW=1$). En este caso los n bits del bus D estarán disponibles en las entradas de los 8 multiplexores, aunque se cargarán como mucho en uno: los 3 bits de DS pasarán por un decodificador de 3 a 8; cada una de sus salidas pasará por una AND con la señal RW , de modo que cuando RW sea 0 todas serán 0, y cuando RW sea 1 solo una de ellas, la seleccionada por DS , estará activa.

En este bloque de registros hemos añadido un multiplexor para la carga en el bus A del contenido de uno de los registros o del resultado (que está siempre almacenado en otro registro, el acumulador ACC).

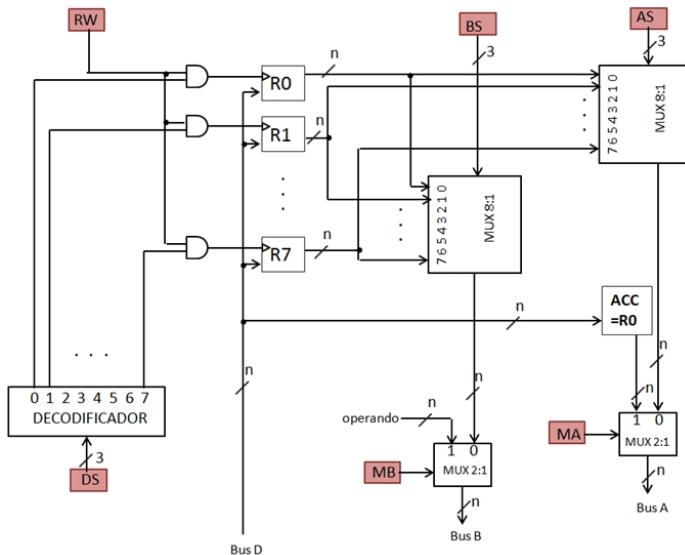


FIGURA 10. Circuito del bloque de registros.

1.4.7. Instrucciones de control de secuencia.

Ya hemos comentado la importancia de las instrucciones de control de secuencia. En ausencia de éstas el Contador de Programa (*PC* por sus siglas en inglés), se cargará siempre con la dirección de memoria de la siguiente instrucción (por simplicidad supondremos instrucciones que ocupan, cada una de ellas, una única posición de memoria). Es decir, el *PC* indica la dirección de una instrucción, y en cuanto esta pasa a ejecución, el *PC* actualiza su valor para indicar la dirección de la siguiente, en un caso normal $PC + 1$.

Las instrucciones de salto y bifurcación permiten alterar este orden de ejecución de las instrucciones cargando en el registro *PC* una dirección nueva. Aunque se habla de ellas indistintamente, la diferencia reside en que las instrucciones de salto redirigen a una instrucción sin relación con la actual, mientras que las bifurcaciones realizar un salto en un entorno del *PC* (se saltan un conjunto de instrucciones hacia adelante o hacia atrás).

Para nuestro diseño, proponemos la posibilidad de realizar tanto saltos como bifurcaciones; en ambos casos pueden ser incondicionales o estar condicionados por el valor de los flags de estado. Esos flags son modificados en cada instrucción, pero es necesario fijarse en el conjunto de instrucciones de cada computador, ya que no todas las instrucciones tienen efecto sobre todos los flags (por ejemplo, las instrucciones de movimiento

o transferencia de datos, no suelen modificar ningún flag de estado). En general diremos que el salto está condicionado por el resultado de la última operación ejecutada.

En nuestro caso, teniendo en cuenta que sólo hemos contemplado el bit de estado Z, nuestras instrucciones de salto y bifurcación condicionales no pueden depender sino de si el resultado de la operación fue o no igual a cero. En un caso más realista, los saltos podrán condicionarse a que el resultado haya sido positivo/negativo, de paridad par/ímpar, que haya provocado una llevada, etc. Definimos los bits LPC (*Load Program Counter*), JB (*Jump/Branch*), CZ (*Conditioned to Z*), que junto con el flag de estado Z determinarán la dirección de la siguiente instrucción a ejecutar. La Tabla 10 recoge las combinaciones de los bits LPC, JB, CZ y Z e indica qué se cargará en cada caso en el Contador de Programa.

Nota: queda como tarea personal el diseño de un circuito que haciendo uso de multiplexores, sumadores y puertas lógicas, cumpla lo descrito en la tabla.

TABLA 10. Operaciones de control de secuencia.

LPC	JB	CZ	Z	Carga en PC	Descripción
0	X	X	X	$PC \leftarrow PC + 1$	Ejecución secuencial (siguiente instrucción)
1	0	0	X	$PC \leftarrow \text{nuevadirección}$	Salto incondicional (JMP)
1	0	1	0	$PC \leftarrow PC + 1$	Salto condicionado a cero (no cumple condición) (JZ, Z=0)
1	0	1	1	$PC \leftarrow \text{nuevadirección}$	Salto condicionado a cero (cumple condición)(JZ, Z=1)
1	1	0	X	$PC \leftarrow PC + offset$	Bifurcación incondicional (BR)
1	1	1	0	$PC \leftarrow PC + 1$	Bifurcación condicionada a cero (no cumple condición) (BZ, Z=0)
1	1	1	1	$PC \leftarrow PC + offset$	Bifurcación condicionada a cero (cumple condición) (BZ, Z=1)

Llegados a este punto, nuestra Ruta de Datos está completa. Realiza operaciones aritméticas, lógicas, de transferencia entre registros y memoria, de desplazamiento de un bit a derecha e izquierda y de control de secuencia. La Tabla 11 muestra todas las operaciones, junto con la codificación resultante de la implementación elegida y el mnemónico que podríamos asignarle.

TABLA 11. Operaciones de la Ruta de Datos, codificación y mnemónicos.

DESCRIPCIÓN		CÓDIGO OPERACIÓN	ENSAMBLADOR
1	$(RD) \leftarrow (RA) - 1$	0 0 0 0 0	DCR RD, RA
2	$(RD) \leftarrow (RA) + 1$	0 0 0 0 1	INR RD, RA
3	$(RD) \leftarrow (RA) + (RB)$	0 0 0 1 0	ADD RD, RA, RB
4	$(RD) \leftarrow (RA) - (RB)$	0 0 0 1 1	SUB RD, RA, RB
5	$(RD) \leftarrow (\bar{RA})$	0 0 1 0 0	MVN RD, RA
6	$(RD) \leftarrow (RA) \text{ AND}(RB)$	0 0 1 0 1	AND RD, RA, RB
7	$(RD) \leftarrow (RA) \text{ OR}(RB)$	0 0 1 1 0	OR RD, RA, RB
8	$(RD) \leftarrow (RA) \text{ XOR}(RB)$	0 0 1 1 1	XOR RD, RA, RB
9	$(RD) \leftarrow (RB)$	0 1 0 0 0	MOV RD, RB
10	$(RD) \leftarrow \text{shift left } (RB)$	0 1 0 0 1	SL RD, RB
11	$(RD) \leftarrow \text{shift right } (RB)$	0 1 0 1 0	SR, RD, RB
12	$(RD) \leftarrow \text{CERO}$	0 1 0 1 1	CLR RD
13	$[M] \leftarrow (AC)$	0 1 1 0 0	STA
14	$(AC) \leftarrow [M]$	0 1 1 0 1	LDA
—		0 1 1 1 0	—
—		0 1 1 1 1	—
—		1 0 0 0 0	—
—		1 0 0 0 1	—
15	$(RD) \leftarrow (RA) + \text{OP}$	1 0 0 1 0	ADI RD, RA, OP
16	$(RD) \leftarrow (RA) - \text{OP}$	1 0 0 1 1	SUI RD, RA, OP
—		1 0 1 0 0	—
17	$(RD) \leftarrow (RA) \text{ AND OP}$	1 0 1 0 1	ANI RD, RA, OP
18	$(RD) \leftarrow (RA) \text{ OR OP}$	1 0 1 1 0	ORI RD, RA, OP
19	$(RD) \leftarrow (RA) \text{ XOR OP}$	1 0 1 1 1	XRI RD, RA, OP
20	$(RD) \leftarrow \text{OP}$	1 1 0 0 0	MVI RD, OP
21	$(RD) \leftarrow \text{shift left OP}$	1 1 0 0 1	SLI RD, OP
22	$(RD) \leftarrow \text{shift right OP}$	1 1 0 1 0	SRI RD, OP
—		1 1 0 1 1	—
23	$(PC) \leftarrow \text{address}$	1 1 1 0 0	JMP
24	$(PC) \leftarrow \text{address } [Z=1]$	1 1 1 0 1	JZ
	$(PC) \leftarrow PC + 1 [Z=0]$		
25	$(PC) \leftarrow PC + \text{offset}$	1 1 1 1 0	BR
26	$(PC) \leftarrow PC + \text{offset } [Z=1]$	1 1 1 1 1	BRZ
	$(PC) \leftarrow PC + 1 [Z=0]$		

1.4.8. Ruta de datos en canalización (pipeline).

Imaginemos un pequeño taller que fabrica zapatos artesanales. El artesano coge el cuero y lo corta, después lo cose a la suela, le inserta una plantilla, le pone unos cordones y lo lustra. Tarda cuatro minutos en cortar el cuero, cinco en coserlo, un minuto para ponerle los cordones, unos segundo para insertarle la plantilla y un minuto y medio para darle el betún y lustrarlo.

En total le lleva aproximadamente 12 minutos finalizar un zapato. Ante un pedido grande decide contratar dos ayudantes y repartirse las tareas para hacer los zapatos más rápido: el primero cortará el cuero, el segundo coserá la suela y le pondrá los cordones y el tercero le pondrá la plantilla y lo lustrará. Compran dos cajas de manera que el primero deposita el cuero una vez cortado en una caja, el segundo coge de ahí las piezas de cuero y deposita los zapatos (con suela y cordones) en otra caja, de donde se abastece el tercero.

Al principio las cajas están vacías y sólo el primero puede trabajar, 4 minutos después empieza el segundo y otros 6 minutos más tarde el tercero. Una vez que están a pleno rendimiento consiguen terminar un zapato cada 6 minutos (el tiempo del que más tarda). El tercero es el que antes termina su tarea, pues sólo le lleva dos minutos, y tiene que esperar otros cuatro minutos a que el segundo termine la suya, por lo que deciden que el último se haga también cargo de poner los cordones. Ahora el primero tarda 4 minutos, el segundo 5 y el tercero 3; por lo que a pleno rendimiento ¡terminan un zapato cada 5 minutos!

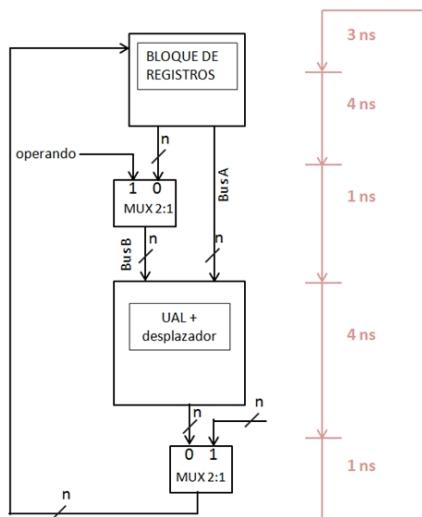


FIGURA 11. Tiempos de ejecución de la Ruta de Datos sin canalización.

Nuestra Ruta de Datos también ha de realizar diferentes tareas: buscar el operando, ejecutar una operación, y guardar el resultado. Suponiendo $4ns$ para leer el dato, otros $4ns$ para ejecutar la instrucción, $1ns$ para cada multiplexor (MUX B y MUX D) y finalmente $3ns$ para escribir el resultado en el bloque de registros, la ejecución de una instrucción requiere $13ns$. Si ha de ejecutarse una instrucción por cada ciclo de reloj, nuestro reloj deberá tener un periodo de al menos $13ns$; lo que nos daría 77 MIPS..

Si queremos sacarle mayor partido, es decir, que todas las partes estén funcionando al mismo tiempo, tendremos que dividir la operación completa en diferentes tareas, al igual que se hizo en el taller de zapatos. Ya sabemos que el tiempo mínimo requerido (y por lo tanto el periodo mínimo de reloj) vendrá determinado por la tarea que más tiempo requiera, y que si estos tiempos son muy dispares se desaprovecha el potencial de la canalización, ya que una etapa estará “esperando”; por lo que deberemos intentar hacer un reparto lo más equitativo posible.

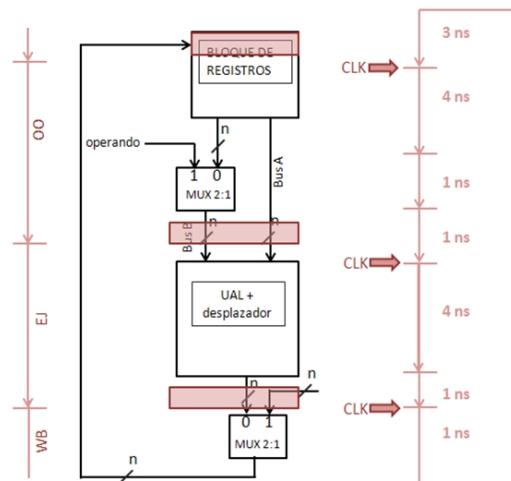


FIGURA 12. Ruta de Datos: ejecución en canalización.

Vamos a dividir la ejecución de nuestra instrucción en 3 etapas cuyas finalidades serán: obtener el operando (OO), ejecución de la operación (EJ) y escritura del resultado (WB). Mientras que la Figura 11 muestra un proceso sin canalización, la Figura 12 muestra las divisiones propuestas. Las cajetillas rojas representan registros, que harán las veces de las cajas del taller de zapatos, es decir, son el lugar en el que cada etapa dejará el resultado de su tarea a disposición de la siguiente etapa. Cada uno de estos registros introducirá un retardo adicional, que vamos a suponer de $1ns$. En el caso del bloque de registros se ha coloreado pero al tratarse de registros obviamente no es necesario añadir nada. Una vez que se haya llenado

la canalización (las “cajas” intermedias tienen datos) las tres etapas estarán trabajando simultáneamente. En el caso ideal en el que todas las etapas fueran idénticas, y una vez llena la canalización, la Ruta de Datos podría funcionar el triple de rápido. En el caso general esto no será posible, ya no todas las tareas requerirán exactamente el mismo tiempo. Analicemos nuestro ejemplo: la tarea OO requiere $4ns$ para la lectura del datos, $1ns$ para el MUX B y otro para el registro adicional; la tarea EJ requiere $4ns$ para la ejecución de la operación y otro para el registro adicional; por último, la tarea WB requiere $1ns$ para el MUX D y otros 3 para la escritura del resultado. Es decir el período mínimo del nuevo reloj, T_{min} , deberá ser $= \max \{4 + 1 + 1, 4 + 1, 1 + 3\} = \max \{6, 5, 4\} = 6ns$. Algo mayor que el ideal, que sería de $13/3 = 4,33ns$.

Tengamos en cuenta que la ejecución de una instrucción está compuesta ahora por 3 etapas, por lo que requiere $3 \times 6 = 18ns$. Ciertamente la ejecución de una única instrucción tardará más que antes, que sólo eran $13ns$. La ventaja reside en que al ejecutar una secuencia de instrucciones, varias etapas se ejecutan simultáneamente, por lo que cada $6ns$ se puede terminar una de ellas. Es cierto que al llenarse y al vaciarse la canalización el aprovechamiento no es máximo, pero esta penalización es menor cuanto mayor es la secuencia de instrucciones a ejecutar, tendiendo finalmente a esos $6ns$.

La Tabla 12 muestra la ejecución de 6 instrucciones. En los ciclos de 3 a 6 la canalización está llena, el rendimiento es óptimo y se termina una instrucción cada ciclo de reloj. Fíjemonos que aunque las instrucciones vienen representadas en horizontal, en los ciclos de 3 a 6, mirando en vertical, se están llevando a cabo simultáneamente las tres etapas. ¿Cuál es el tiempo medio de ejecución de una instrucción en el anterior ejemplo? Veamos: la ejecución de 6 instrucciones requiere 8 ciclos de reloj. Siendo cada uno de $6ns$, podemos decir que cada instrucción requiere de media $48/6=8ns$.

TABLA 12. Ejemplo de ejecución en canalización de la Ruta de Datos.

Operación	Ciclos de reloj							
	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
OP_1	OO	EJ	WB					
OP_1		OO	EJ	WB				
OP_2			OO	EJ	WB			
OP_3				OO	EJ	WB		
OP_4					OO	EJ	WB	
OP_5						OO	EJ	WB

Resumiendo: antes la ejecución de una instrucción requería $13ns$; hemos dividido la Ruta de Datos en 3 etapas por lo que idealmente podríamos ejecutar una instrucción cada $4,33ns$, pero debido a que las etapas no son idénticas por un lado y a la inserción de los necesarios registros por otro, el tiempo mínimo para ejecutar una instrucción resultaba ser de $6ns$. Incluyendo además la penalización debida al llenado y vaciado de la canalización ese tiempo puede ser mayor; $8ns$ para el caso de 6 instrucciones. Casi el doble de los $4,33ns$ del caso ideal, pero bastante mejor que los $13ns$ de la ejecución sin canalización.

1.5. UNIDAD DE CONTROL

1.5.1. Introducción.

La Unidad de Control es la parte de la computadora encargada de generar la palabra de control necesaria para la ejecución de una instrucción, a partir de la instrucción dada. Ésta estará diseñada para una arquitectura concreta, con un conjunto de instrucciones y un formato de instrucción determinado. Pero en lo relativo al diseño de la propia CPU hay, como a menudo, diferentes opciones, con sus ventajas y desventajas. Para empezar, podría interesarnos una computadora que sea capaz de ejecutar cada una de las instrucciones en un único ciclo de reloj. Para ello serían necesarias instrucciones cortas, todas de la misma longitud, lo cual nos llevaría hacia una arquitectura RISC (*Reduced Instruction Set Computer*). Este tipo de arquitecturas dan lugar a códigos más largos, pero se ha demostrado su eficiencia y rapidez. En el lado contrario estaría el paradigma CISC (*Complex Instruction Set Computer*) en el que están implementadas instrucciones que realizan tareas relativamente complejas, dando lugar a programas más compactos, a costa de un juego de instrucciones más irregular: instrucciones de diferente longitud, distinto número de campos de dirección... En cualquier caso, la decisión que se tome se verá reflejada en la CPU, especialmente en la Unidad de Control, como veremos más adelante. Aprovecharemos estos dos paradigmas de arquitecturas para analizar también las diferencias entre el control microprogramado y el cableado (diferentes opciones para la obtención de la palabra de control a partir de la instrucción).

1.5.2. Formato de instrucción.

Siguiendo con el diseño de nuestra CPU, es el momento de determinar el formato de las instrucciones. Para ello definamos primero algunos conceptos:

- **Instrucción:** conjunto de bits dividido en campos.

- **Campos:** cada uno se asocia a un elemento y define unas funciones.
- **Código de Operación (C.Op.):** campo de la instrucción formado por un conjunto de m bits, puede representar hasta 2^m operaciones diferentes.
- **Campos de dirección:** dirección de los operandos que participan en la instrucción.

La instrucción deberá tener siempre un campo de código que defina la operación a realizar, algún campo de dirección para indicar dónde están (o cuáles son, en el caso de operando inmediato) los operandos para dicha tarea. Sin olvidar las instrucciones de control de secuencia, que deberán tener datos que determinen los saltos o bifurcaciones a realizar. En el caso más sencillo, cada instrucción dará lugar a una microinstrucción (una palabra de control); pero en un caso genérico en el que la ejecución de cada instrucción requiera la ejecución de varias microinstrucciones, la Unidad de Control deberá generar, a partir del conjunto de bits de la instrucción, la secuencia de palabras de control necesarias.

TABLA 13. Formato de instrucciones de la CPU diseñada.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CÓDIGO DE OPERACIÓN					SD		SA		SB		XX				

Operación con registros

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CÓDIGO DE OPERACIÓN					SD		SA		OPERANDO						

Operando inmediato

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CÓDIGO DE OPERACIÓN					XX		XX		OFFSET						

Instrucción de bifurcación

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CÓDIGO DE OPERACIÓN					NUEVA DIRECCIÓN										

Instrucción de salto

Fijémonos en la Tabla 13, hay 4 formatos de instrucción: *operación con registros*, *operando inmediato*, *instrucción de bifurcación* e *instrucción de salto*. Todas ellas tienen un campo de *código de operación*, dicho campo define el tipo de operación, y tiene 5 bits para poder codificar hasta 32 operaciones. Los demás campos varían. Los dos primeros formatos son para

operaciones, por lo que ambos especifican la dirección del destino y la dirección del operando A. En cuanto al operando B, en un caso se indica el registro que lo contiene ($SB[2..0]$), mientra que en el otro se especifica directamente el operando (en un campo de 5 bits).

Los otros dos formatos de instrucción son para instrucciones de control de secuencia. En la *instrucción de bifurcación*, un campo de 5 bits llamado *offset*, indica el desplazamiento a realizar entorno a la posición de la instrucción actual (entorno al Program Counter). Teniendo en cuenta que el salto puede ser hacia delante o hacia atrás (dato *offset* en complemento a dos), la mayor distancia que podríamos alejarnos de la instrucción actual sería 15 posiciones hacia delante y 16 hacia atrás. En nuestro ejemplo, todas las instrucciones ocupan lo mismo, 2 bytes, por lo que haciendo uso de una memoria con una palabra de 16 bits, una posición significa una instrucción.

Por último, la *instrucción de salto*, en esta sólo hay un campo, además del *código de operación*: la *nueva dirección*. Este campo que indica la dirección de la instrucción a la que debemos saltar tiene 11 bits, con los que podemos direccionar una memoria de 2 k posiciones.

1.5.3. Control cableado.

En este apartado vamos a suponer instrucciones de ciclo único (en el caso de instrucciones complejas la ejecución de cada una de ellas puede necesitar más de un ciclo de reloj); es decir, con cada ciclo de reloj, se leerá de la posición de memoria indicada por el contador de programa una de estas instrucciones en código máquina. El contador de programa se actualizará y la unidad de control deberá generar, a partir de lo indicado por esos bits que conforman la instrucción, la palabra de control necesaria (ver Figura 13).

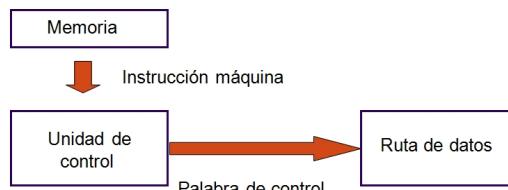


FIGURA 13. Traducción de la instrucción por la Unidad de Control

Hay dos formas de realizar esta “traducción”: utilizando lógica cableada o almacenada. En este punto haremos un diseño utilizando lógica cableada. Es decir, será una lógica combinacional que tendrá como variables de entrada los 16 bits de la instrucción (según el formato de instrucción visto), y como salidas cada uno de los bits que conforman la palabra de control

(los 19 bits de la Tabla 8). Realmente los campos relacionados con saltos u operando inmediato no son necesarios aquí, ya que pasarán directamente a donde corresponda (lógica para calcular el salto o entrada a la UAL); además los campos SD, SA y SB pasarán sin modificar, por lo que la parte más complicada está en la conversión del campo *código de operación*.

1.5.3.1. Control cableado: decodificador.

Este tipo de control se diseña siguiendo los métodos clásicos de diseño lógico, aunque hoy en día los programas de diseño asistido por ordenador (CAD) para VLSI resuelven de manera automática la mayoría de las dificultades de diseño, sigue siendo más laborioso y costoso que la lógica almacenada (que veremos más adelante).

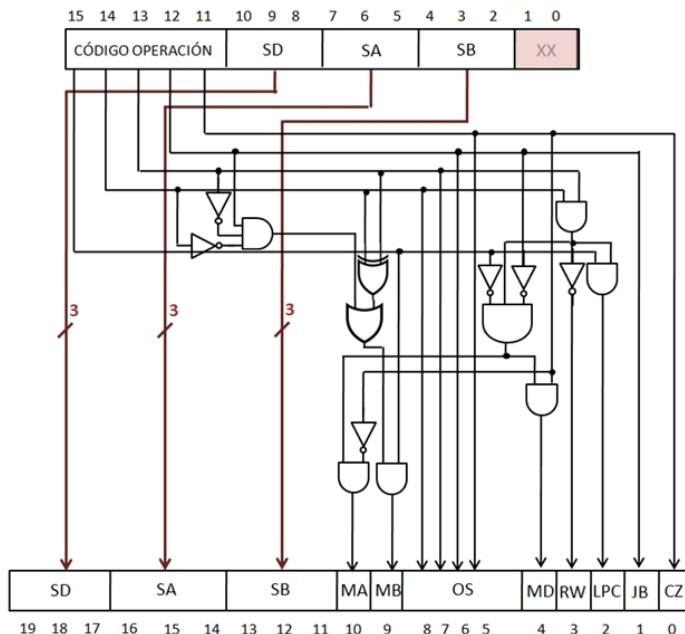


FIGURA 14. Unidad de Control cableada: decodificador.

Otro de sus inconvenientes se deriva del hecho de ser una lógica cableada, ya que una vez terminada la implementación, tenemos un circuito con conexiones físicas, lo que hace difícil introducir modificaciones. Pero no todo son desventajas, ya que en igualdad de condiciones la lógica cableada es más rápida que la almacenada.

1.5.3.2. Control cableado: CPU.

Uniendo los elementos diseñados: la Ruta de Datos (con su bloque de registros, unidad de funciones compuesta por una unidad lógica, una aritmética y un desplazador), el control de salto (para las instrucciones de control de secuencia) y la Unidad de Control (con el decodificador cableado de la Figura 14) a la memoria, obtenemos la Unidad Central de Proceso (CPU) de la Figura 15. Suponiendo una Unidad de Control de ciclo único (es decir, cada instrucción ha de completarse en un único ciclo de reloj),

las operaciones de transferencia de datos en las que se implique la memoria (mover un dato de memoria a registro o de registro a memoria), resultarían demasiado lentas con una única memoria a la que tuviéramos que acceder dos veces (una para leer la instrucción y otra para leer o escribir el dato); es por ello que en la Figura 15 hemos optado por memorias separadas para datos e instrucciones.

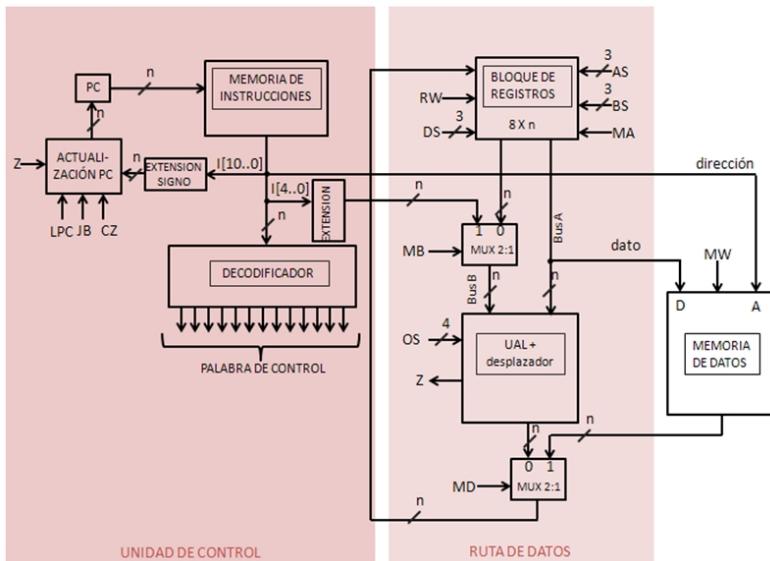


FIGURA 15. CPU con control cableado.

Fijándonos precisamente en estas memorias vemos que los últimos once bits de la instrucción (indicados como $I[10..0]$), van, por una parte a la memoria de datos, y por otra parte a la lógica para la actualización del PC . En la memoria de datos se utilizan como dirección, tenemos en este caso por lo tanto una memoria de $2k \times n$ ($2k$ posiciones de n bits). Sin embargo, hemos supuesto una memoria para instrucciones mayor, por lo que estos once bits no son suficientes para direccionarla. Es por ello que han de pasar por el bloque denominado “Extensión de signo” para poder sustituir el contenido del Contador de Programa (PC). Cuando la instrucción sea una instrucción de salto (véanse los formatos de instrucción, Figura 13) a estos últimos once bits de la instrucción se le han de añadir ceros por la izquierda hasta obtener los n bits necesarios para direccionar la memoria de instrucciones.

En el caso de tratarse de una instrucción de bifurcación sólo los últimos cinco ($I[4..0]$) definen el salto en un entorno del valor actual del PC . Estas instrucciones de bifurcación son las que utilizaremos para realizar bucles, por ejemplo, en los que una porción de código se repite un número determinado de veces. Para ello es necesario saltar un conjunto de instrucciones,

tanto hacia delante como hacia atrás. Como ya adelantábamos para ello utilizaremos la representación para números binarios con signo **complemento a dos**. Como hemos supuesto instrucciones que ocupan una única posición de memoria, esto supone poder saltar 15 o 16 instrucciones, lo cual parece un número razonable.

Ahora bien, al igual que en las operaciones de salto, también aquí debemos pasar de estos 5 bits a n . Imaginemos que tenemos el número decimal 7 representado en complemento a dos con cinco bits (00111), y que queremos pasarlo a diez bits (0000000111), sólo tenemos que añadir ceros por la izquierda. Si el número fuera el -7 (11001), y le añadiésemos ceros por la izquierda (0000011001) el número resultante en complemento a dos sería el 25. La forma correcta de pasar el número a 10 bits sería añadirle unos (1111111001). Siempre que la representación de números binarios con signo elegida tenga un '1' en el bit más significativo para los números negativos y un '0' para los positivos, el "alargar" esos cinco bits hasta el número de bits necesario supondrá copiar el bit más significativo las veces necesarias por delante de éste. A esta tarea de llenar con ceros o unos según proceda, se le denomina **extensión de signo**.

Fijémonos que los cinco últimos bits de la instrucción pasan por una lógica de extensión para, en caso de tratarse de una instrucción de operando inmediato, seleccionarse en el multiplexor B y poder funcionar como cualquier otro operando de n bits.

1.5.4. Control microprogramado.

El control microprogramado resulta conceptualmente sencillo: una memoria de control contiene las palabras de control correspondientes a cada una de las instrucciones; el código de operación de la instrucción se utiliza para dirigir la posición de memoria necesaria. Este tipo de control, a diferencia del cableado, es fácil de modificar en el sentido de que sólo supone reescribir una memoria; en cualquier caso el desarrollo del microcódigo no es baladí. El control microprogramado permite implementar computadoras que ejecuten diferentes juegos de instrucciones. Estos, además, pueden incluir instrucciones complejas. Es por ello que aprovecharemos para ver las diferencias, no sólo entre el control cableado y el microprogramado, sino también, entre el ciclo simple y el múltiple.

El diseño de un control microprogramado de ciclo múltiple ha de cumplir cuatro requerimientos básicos:

1. Memoria de control con capacidad para almacenar los microprogramas (al tratarse de instrucciones de ciclo múltiple, en el caso

general cada instrucción requerirá no una, sino un conjunto de microoperaciones, es decir un microprograma) correspondientes a todas las instrucciones

2. Procedimiento para asociar a cada instrucción máquina el microprograma que le corresponde
3. Mecanismo para leer de manera ordenada las instrucciones que conforman un microprograma (y saltar a un nuevo microprograma al terminar)
4. Mecanismo de microbifurcación condicionada que permita ejecutar instrucciones máquina de bifurcación condicionada

Si el código de operación tiene 5 bits, como es nuestro caso, la memoria de microcódigo deberá tener 32 posiciones para el caso de instrucciones de ciclo único. Pero en el caso de ciclo múltiple, cada instrucción puede requerir varios ciclos de reloj (y en consecuencia varias microoperaciones, varias palabras de control), la memoria que contiene el microcódigo ha de ser mayor (requerimiento básico 1). Supongamos que cada instrucción requiere 4 microoperaciones; en este caso necesitaremos una memoria de al menos 128 posiciones. Es necesario asociar no sólo una palabra de control con su instrucción (requerimiento básico 2), sino todas las palabras de control que forman el microprograma, y en el orden necesario (requerimiento básico 3).

Además, en el caso más general, habrá instrucciones muy complejas (mucho más que las simples instrucciones de transferencia, incremento, resta, etc. desarrolladas en nuestro ejemplo de la Ruta de Datos), que necesiten poder realizar operaciones de control de secuencia del propio microprograma (requerimiento básico 4).

Los dos últimos requerimientos hacen necesario un secuenciamiento; éste podrá ser explícito o implícito.

1.5.4.1. Secuenciamiento explícito.

En este caso el código de operación se utiliza para direccionar la primera microinstrucción. Siguiendo con la suposición de 4 microoperaciones por instrucción (memoria de 128 posiciones), los 5 bits del código de operación sólo podrán direccionar la primera cuarta parte de la memoria (posiciones de 0 a 32), con lo que las otras 3 microoperaciones de la instrucción estarán en las posiciones de 33 a 127. Inevitablemente será cada una de las microoperaciones la que especifique de manera explícita donde está la siguiente y si es o no la última del microprograma. La dirección de la siguiente microoperación necesita (en nuestro supuesto de memoria de 128 posiciones) 7 bits; por lo que cada microoperación tiene además de la

antigua palabra de control, 1 bit que indica si es o no la última del microprograma y 7 bits con la dirección de la siguiente.

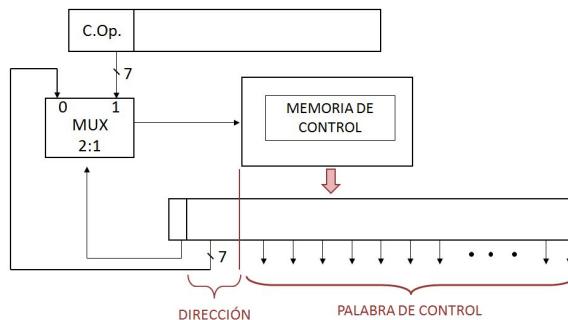


FIGURA 16. Secuenciamiento explícito

Resumiendo, en el secuenciamiento explícito:

- Cada microinstrucción tiene la dirección de la siguiente
- Las microinstrucciones pueden estar desordenadas
- Requiere gran parte de la memoria de control

La Figura 16 muestra el esquema de bloques del tratamiento del secuenciamiento explícito.

1.5.4.2. Secuenciamiento implícito.

La idea en este otro caso es que las microoperaciones estén ordenadas, de manera que una vez se lee la primera, las siguientes están en posiciones consecutivas de la memoria de microprograma (la dirección de la actual +1). Así no tenemos una parte significativa de la memoria ocupada debido al secuenciamiento. En este caso ya no es posible utilizar directamente el código de operación para direccionar la primera microoperación, ya que éste tiene solo 5 bits, y no permite direccionar cualquier posición de la memoria (recordemos que en nuestro supuesto el código de operación tiene 5 bits y se necesitan 7 para direccionar la memoria de 128 posiciones).

La solución pasa por utilizar el código de operación, no como dirección de la primera microoperación, sino como herramienta para obtenerla. Una memoria ROM de 32 posiciones guardará las direcciones de inicio de los microprogramas (direcciones que se formarán añadiendo dos ceros tras el código de operación, por lo que en nuestro ejemplo serán 0, 4, 8, 12, ... 116, 120 y 124). En lugar de una ROM podría ponerse algún otro dispositivo programable que realice la “traducción” del código de operación a la

dirección de memoria necesaria, ya que en general no todas las instrucciones requerirán el mismo número de microoperaciones. Resumiendo, en el secuenciamiento implícito:

- Las microinstrucciones de un microprograma han de estar almacenadas en orden en memoria
- Requiere un pequeño decodificador entre el código de operación y el multiplexor de las direcciones
- Un registro de microdirecciones y un incrementador

La Figura 17 muestra el esquema de bloques del tratamiento del secuenciamiento implícito.

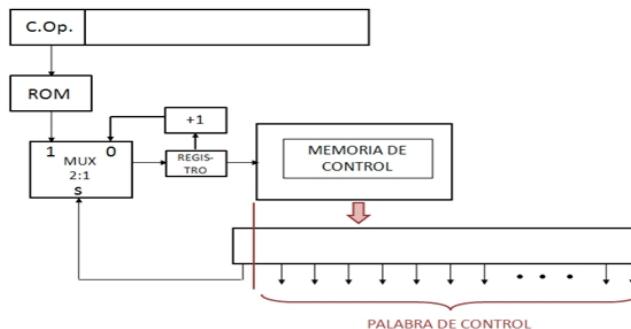


FIGURA 17. Secuenciamiento implícito

1.5.4.3. Control microprogramado: CPU.

Que el control sea cableado o almacenado en una memoria de microprograma no implica una diferencia sustancial en la organización total de la CPU. No se puede decir lo mismo del hecho de que se trate de un control de ciclo sencillo o de ciclo múltiple. El que la lectura de una instrucción de memoria, su decodificación y ejecución no terminen en un único ciclo de reloj, hace necesario añadir algunos registros que almacenen la información mientras sea necesaria. Para empezar, los campos de la propia instrucción deberán estar disponibles durante 4 ciclos de reloj (siguiendo con nuestro supuesto de que cada instrucción requiere 4 microoperaciones). Es decir, tendremos que almacenarla en un registro: el **registro de instrucción (IR)**. Este registro se cargará con una nueva instrucción cada vez que se termine la ejecución de la anterior, pero no antes; por lo que será necesaria una **señal de carga de instrucción (IL)** que habilite la escritura en el registro de instrucción.

Pero no solo los campos de la instrucción, también los operandos han de estar disponibles durante los 4 ciclos de reloj. Para ello se añaden en el bloque de registros, una serie de registros no accesibles por usuario. Supongamos que a los 8 registros que ya teníamos les añadimos otros 8. Necesitaremos los bits que seleccionan uno de estos registros tanto para pasar su

contenido al bus A (**ASS**), al bus B (**BSS**) o para cargarlo con lo que haya en el bus D (**DSS**).

Dado que ahora tenemos 16 registros en el bloque de registros, R0-R15 (aunque sólo 8 son direccionables por el usuario), la solución más sencilla será sustituir el decodificador de 3 a 8 por uno de 4 a 16 para seleccionar el registro el en que se carga el contenido del bus D (ver Figura 10). Análogamente, necesitaremos dos multiplexadores de 16 entradas conectando los registros con los buses A y B. Una señal de un bit (**US**) irá delante de los otros 3 indicando si estos direccionan un registro de usuario (**US=0**, R0 a R7) o uno de sistema (**US=1**, R8 a R15).

En el control cableado separábamos instrucciones y datos en dos memorias diferentes para poder así realizar instrucciones de transferencia de datos en las que la memoria estuviera implicada. En este caso, al tratarse de instrucciones de ciclo múltiple, podemos realizar estas instrucciones en varios pasos, por lo que podemos tener instrucciones y datos en una única memoria. En este caso, un multiplexor seleccionará entre una dirección de instrucción (**MM=0**) y una de dato (**MM=1**).

Por último tendríamos la memoria con el microcódigo, las señales necesarias para el secuenciamiento (diferentes en función de que sea ésta explícita o implícita), así como las señales y la lógica para implementar las microbifurcaciones; todo contenido en el bloque denominado *memoria de control y secuenciación*.

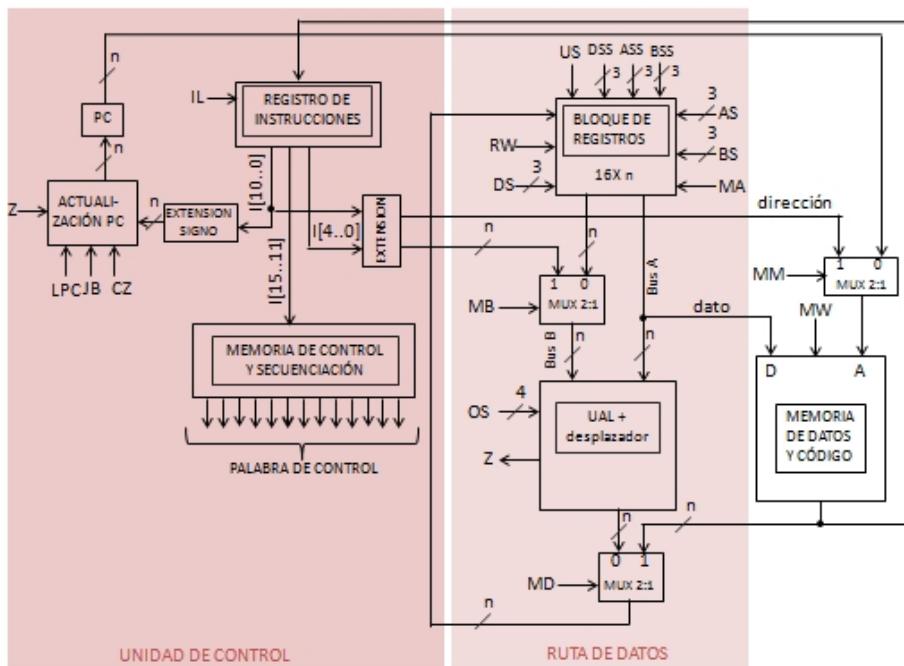


FIGURA 18. Control microprogramado.CPU

1.6. COMPUTADOR EN CANALIZACIÓN

La ejecución de una instrucción supone varias etapas en las que intervienen diversas partes. La ejecución en canalización permite un mejor aprovechamiento de estas partes, ya que cada una realiza su tarea sin esperar a las demás. Esta idea se presentó tras ver la Ruta de Datos, en el punto 1.4.8. En este punto la veremos aplicada a toda la CPU.

1.6.1. Ciclo de ejecución de un computador.

Podemos dividir el ciclo de ejecución básico de un computador en los siguientes pasos:

1. Leer la instrucción de memoria señalada por el Contador de Programa a un registro de la Unidad de Control
2. Decodificar la instrucción
3. Buscar los operandos (obtener la dirección efectiva)
4. Recuperar los operandos
5. Ejecutar la operación
6. Almacenar resultado

Volver al paso 1 a por la siguiente instrucción.

1.6.2. Ejecución en canalización .

Ejecución sin canalización:

- La ejecución de una instrucción requiere un periodo de reloj mayor o igual al tiempo de retardo de todos los elementos que intervienen en la ejecución de la misma.
- Solo una de las partes funciona en cada momento.

Ejecución en canalización:

- La ejecución de una instrucción se divide en etapas (han de añadirse registros para mantener los valores entre etapas) .
- Se sustituye el reloj por uno cuyo periodo sea mayor o igual al tiempo de retardo de la etapa de mayor duración) .
- Una vez llena la canalización, todas las etapas funcionan simultáneamente.

Supongamos que viendo los retardos de los diferentes elementos involucrados, dividimos la ejecución del computador en las cuatro etapas representadas en la Figura 19, intentando buscar divisiones con tiempos de ejecución similares. Tal y como se adelantaba, será necesario añadir registros entre

algunas de estas divisiones, para que los datos y las señales que necesitará la siguiente etapa queden almacenados. En la Figura 19 estos registros se han representado como cajetines rojos.

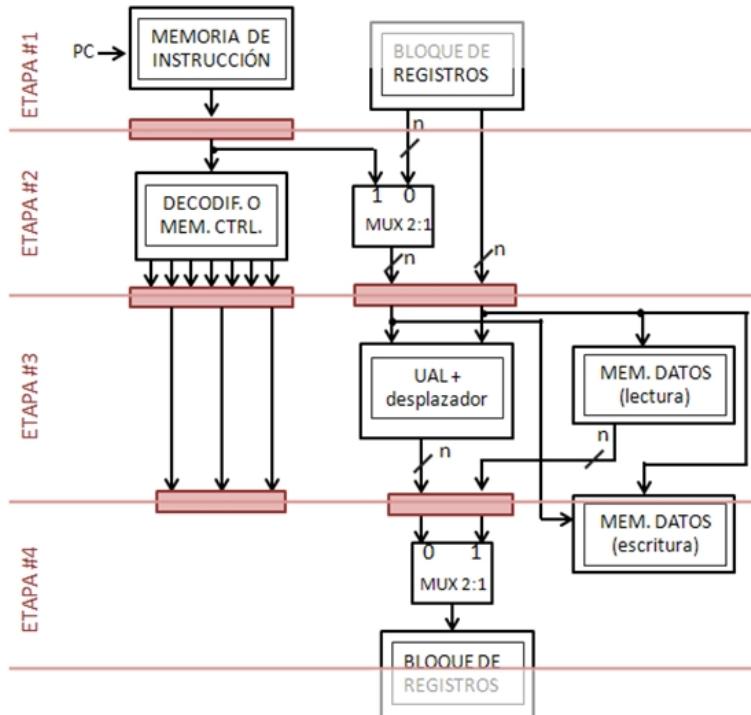


FIGURA 19. Esquema de la división en etapas de una CPU para la ejecución en canalización.

La Tabla 14 muestra las cuatro etapas en las que se ha dividido la ejecución de una instrucción, la tarea que se lleva a cabo y las variables que intervienen en cada una, así como las señales que se necesitan en etapas sucesivas, por lo que quedan almacenadas en los registros (en color rojo).

Supongamos que antes una instrucción tardaba 17ns y que ahora cada instrucción se divide en 4 etapas, de las cuales, la que más tarda (incluyendo el retardo adicional por el registro añadido en caso necesario), tarda 5ns . En este caso la ejecución de una sola instrucción tardaría 20ns más que sin canalización. Sin embargo cuanto mayor sea la secuencia de instrucciones que se ejecuten de manera consecutiva, más se acercará el tiempo medio empleado en cada instrucción a esos 5ns .

TABLA 14. Tabla de las etapas de la canalización.

Etapa	Intervienen	Función
1	Contador de programa (<i>PC</i>) Memoria de instrucción	Obtener la instrucción de la memoria de instrucción y actualizar el <i>PC</i>
		Plataforma de canalización que juega el papel de registro de canalización
2	Decodificador Bloque de registros	Decodificar IR → señales de control SA , SB , MA y MB para obtención de datos. El resto más adelante.
		En este registro se guardan las señales de control que usaremos más adelante
3	Unidad de ejecución Lectura/Escritura en memoria de datos	Operación de UAL, de desplazamiento o de memoria → OS y MW La lectura de memoria de datos se realiza en esta etapa → dato direccionado → Salida de datos
		Resultados de la etapa 3, señales de control de la etapa 4 La escritura en memoria de datos es parte de esta plataforma
4	Escritura en el bloque de registros	SD , MD y RW se usan en la etapa de escritura

En el ejemplo de la Tabla 15 vemos que la ejecución de una única instrucción requiere 4 ciclos de reloj (20ns si estos son de 5ns). En el caso ideal, cuando la canalización está llena (ciclos 4, 5 y 6) se ejecuta, de media, una instrucción cada ciclo de reloj. En general la mejora es menor debido al tiempo de llenado y vaciado de la canalización. En este ejemplo, 6 instrucciones requieren 9 ciclos, 7,5ns de media, más que los 5ns óptimos, pero bastante menos que los 17ns de la ejecución sin canalización. La penalización debida al llenado y vaciado de la canalización es menos evidente según aumenta el número de instrucciones consecutivas, tiendiendo al caso óptimo de los 5ns.

TABLA 15. Ejemplo de ejecución de una computadora en canalización.

Instrucción	Ciclos de reloj								
	<i>T</i> ₁	<i>T</i> ₂	<i>T</i> ₃	<i>T</i> ₄	<i>T</i> ₅	<i>T</i> ₆	<i>T</i> ₇	<i>T</i> ₈	<i>T</i> ₉
Nº1	#1	#2	#3	#4					
Nº2		#1	#2	#3	#4				
Nº3			#1	#2	#3	#4			
Nº4				#1	#2	#3	#4		
Nº5					#1	#2	#3	#4	
Nº6						#1	#2	#3	#4

1.6.3. Riesgo de datos debido a la ejecución en canalización.

Dado que una instrucción recopila los operandos necesarios antes de que la anterior instrucción haya terminado, haya reescrito su resultado, la ejecución en canalización puede dar lugar al denominado riesgo de datos. De esta forma puede darse que una instrucción requiera un operando resultado de una instrucción anterior, y que debido a la ejecución en canalización, lea el registro que debería contenerlo, antes de que éste se haya actualizado. Veámoslo con un ejemplo: supongamos la división en cuatro etapas de la Figura 19, y una secuencia de instrucciones que opera con los datos contenidos en los registros R1 a R6. Por simplicidad vamos a suponer que el contenido inicial de estos registros es $R1 = 1, R2 = 2, \dots, R6 = 6$. La Tabla 16 muestra la secuencia de instrucciones, los valores esperados y los obtenidos tras la ejecución de las mismas.

TABLA 16. Riesgo de datos. Ejemplo.

MOV R1, R3	#1	#2	#3	#4		
ADD R2, R1, R6	#1	#2	#3	#4		
ADD R5, R1, R2	#1	#2	#3	#4		
<hr/>						
	R1	R2	R3	R4	R5	R6
Valor inicial	1	2	3	4	5	6
Resultado esperado	3	9	3	4	12	6
Resultado obtenido	3	7	3	4	5	3

Tras la primera instrucción R1 debería ser 3; tras la segunda $R2 = 9$ y finalmente $R5 = 12$. Sin embargo, cuando la segunda instrucción recupera sus operando (etapa 2) la primera aún no los ha reescrito (lo hará en la etapa 4) por lo que R1 aún vale 1. Cuando la tercera instrucción recupera sus operandos, R1 ya vale 3, pero R2 sigue valiendo 2. En este ejemplo, tras la ejecución del programa, $R5 = 5$ en lugar de 12. En general el resultado es imprevisible.

1.6.3.1. Solución software al riesgo de datos.

1. Añadir retardos (sw)

La solución software pasa por introducir donde sea necesario instrucciones nulas, de modo que sin modificar el programa, retrasen la ejecución de aquellas instrucciones que requieren los resultados de las anteriores, dando tiempo a que estos se reescriban en el bloque de registros. En el ejemplo se han utilizado instrucciones NOP (No OPeration), que consumen un ciclo de reloj sin

provocar ningún cambio. La Tabla 17 muestra la introducción de instrucciones NOP.

TABLA 17. Solución software al riesgo de datos. Ejemplo.

MOV R1, R3	#1	#2	#3	#4	
NOP	#1	#2	#3	#4	
ADD R2, R1, R6		#1	#2	#3	#4
NOP		#1	#2	#3	#4
ADD R5, R1, R2		#1	#2	#3	#4

Esta solución resulta un tanto complicada ya que el programador ha de tener un conocimiento exhaustivo de la arquitectura así como de la canalización, para poder detectar los puntos en los que puede darse riesgo de datos, y así solucionarlo introduciendo operaciones nulas. Además, es obvio que implica una penalización en el tiempo.

2. Reordenación del código

Otra opción, en lugar de alargar el código añadiendo instrucciones nulas, es reordenarlo, viendo qué instrucciones se pueden mover sin modificar el resultado del programa y colocándolas estratégicamente entre aquellas instrucciones en las que se da el riesgo de datos. Esta tarea es laboriosa y además podría suceder que se introduzcan nuevos riesgos de datos o incluso errores en el programa por reordenar las instrucciones.

1.6.3.2. Solución hardware al riesgo de datos.

1. Añadir retardos (hw)

Con el objetivo de liberar al programador de la tarea descrita en el punto anterior, se añade una lógica capaz de detectar el riesgo de datos, es decir, detecta aquellas instrucciones que requieren un operando (el contenido de un registro), que siendo resultado de una instrucción anterior, se escribirá en una etapa posterior. En caso de detectar esta circunstancia, detiene durante un ciclo de reloj la ejecución en la canalización, evitando así el riesgo de datos. En este caso se dice que se ha introducido una burbuja en la canalización: se mantiene el valor del contador de programa para que no se lea la siguiente instrucción y en la etapa de ejecución equivale a una operación NOP. Es decir, la adición de un hardware que detecte el riesgo de datos libera al programador de introducir operaciones nulas para garantizar el correcto funcionamiento del programa, pero

en cuanto al retardo introducido, no supone ninguna ventaja frente a la solución software. La Tabla 18 muestra la introducción de burbuja en ejecución.

TABLA 18. Solución hardware al riesgo de datos. Ejemplo.

MOV R1, R3	#1	#2	#3	#4	
ADD R2, R1, R6	#1	0	#2	#3	#4
ADD R5, R1, R2	0	#1	#2	#3	#4

2. *Forwarding*

El *forwarding* consiste en acercar la salida de la UAL a aquel segmento que la necesita. El dato que necesita la instrucción actual, es el resultado de la anterior, y aunque aún no se haya reescrito, está disponible en la salida de la unidad de funciones (o de la UAL). Añadiendo la circuitería necesaria (un multiplexor y cableado) se puede acercar este dato al segmento deseado y con la lógica que detecta el riesgo de datos, podríamos seleccionar entre un operando nuevo o el resultado de la anterior instrucción.

Volviendo a la Figura 10 del punto 1.4.6, vemos que un multiplexor (mux A) permite seleccionar el dato que se quiere cargar en el bus A: si la señal MA vale 0, se carga la salida del registro seleccionado por la señal AS; sin embargo, si MA=1 el valor que pasará al bus A es el que hay en el bus D (recordemos que el bus D es el que tiene la solución, la salida de la Unidad de Funciones). Este sería un ejemplo de *forwarding*, donde la señal MA la deberá generar la lógica que detecta el riesgo de datos. (Recordemos que en la Sección 1.4.5 indicábamos cómo la señal MA no era necesaria en la palabra de control, dado que esta lógica que da una solución hardware al riesgo de datos es la que ha de generarla).

Para un análisis más profundo de la canalización (implementación, riesgos, posibles soluciones, tiempo de ejecución etc.) se recomienda la lectura de los apuntes de clase de la asignatura *Computer Organization and Design* de la Universidad de Florida [14].

Capítulo 2

FORMATO DE INSTRUCCIÓN, MODOS DE DIRECCIONAMIENTO E INTERRUPCIONES

En este mundo traidor
nada es verdad ni mentira
todo es según el color
del cristal con que se mira
(Ramón de Campoamor: 1817 - 1901)

RESUMEN. En este Capítulo definiremos el formato de instrucción y veremos la necesidad de que una CPU trabaje con instrucciones de diferentes formatos. El formato de una instrucción lo constituyen distintos campos. Existen instrucciones sin campo de dirección, otras que tienen uno, dos o hasta tres. Hay un campo, denominado modo, formado por una combinación de bits que permite interpretar de una u otra manera los bits del campo dirección; lo que nos lleva a los diferentes modos de direccionamiento. Veremos 7 modos de interpretar el campo dirección.

Las Interrupciones rompen el normal funcionamiento del procesador; se ejecuta un microcódigo en la CPU (ajeno al usuario) que asocia a cada interrupción una dirección de salto a una posición de memoria, donde se ejecutará un código, escrito por el usuario, que atiende a la naturaleza de la Interrupción. Éste microcódigo resuelve el retorno al programa una vez atendida (servida) la Interrupción.

2.1. FORMATO DE INSTRUCCIÓN

Un computador tiene (generalmente) un juego de instrucciones y algunos formatos de instrucción. Comencemos definiendo algunos conceptos:

- **Formato de instrucción:** caja rectangular que representa los bits del código binario de la instrucción, y se separa en campos. Por ejemplo: campo de código de operación (código de instrucción), campo de dirección o campo de modo.
- **Lenguaje máquina:** lenguaje binario para escribir las instrucciones y almacenarlas en memoria.

- **Lenguaje ensamblador** (lenguaje de bajo nivel): lenguaje simbólico que sustituye los códigos binarios de instrucciones y direcciones por nombres simbólicos (mnemónicos).
- **Arquitectura del conjunto de instrucciones** (ISA¹): formado por el nombre simbólico y el código binario de todas las instrucciones contempladas.
- **Arquitectura**: la forman la Arquitectura del conjunto de instrucciones, la organización y el hardware.
- **Dirección efectiva**: dirección de memoria en la que se encuentra el operando.

En el Capítulo 1, se diseñó una CPU con un total de 26 instrucciones diferentes (Tabla 11). Entre ellas había algunas que realizaban operaciones con operandos que estaban en algún registro del bloque de registros, o con algún operando explícitamente indicado (operando inmediato). Había también instrucciones que permitían el control del flujo del programa realizando saltos y bifurcaciones. Es decir, en nuestro diseño existen 4 tipos de formato de instrucción.

TABLA 1. Formatos de instrucción de la CPU diseñada.

CÓDIGO DE OPERACIÓN	SD	SA	SB	XX
Operación con registros				
CÓDIGO DE OPERACIÓN	SD	SA	OPERANDO	
Operando inmediato				
CÓDIGO DE OPERACIÓN	XX	XX	OFFSET	
Instrucción de bifurcación				
CÓDIGO DE OPERACIÓN	NUEVA DIRECCIÓN			
Instrucción de salto				

La CPU diseñada es muy básica y aunque en cierto modo puede decirse que utiliza diferentes modos de direccionamiento, estos se distinguen en base al código de operación, no en base a un campo modo. Es decir, se distingue si los últimos bits son un operando o un offset gracias al código de operación. En general, una CPU discernirá entre varios modos de direccionamiento en base al valor del campo modo (la combinación de bits en dicho campo permite interpretar el campo de dirección).

¹Instruction Set Architecture

La Tabla 2 muestra un formato de instrucción genérico: los bits del campo código de operación identifican la instrucción y el campo modo permite interpretar los bits del campo de dirección. En una arquitectura real tendremos instrucciones sin campo de dirección (por ejemplo las instrucciones para el manejo de pila: PUSH, POP), instrucciones con uno o dos campos de dirección (el segundo operando y/o el destino están implícitamente indicados; por ejemplo: $ADDR1\ Acc \leftarrow Acc + R1$ un operando está en el acumulador; el resultado también se guarda en el acumulador) o instrucciones en las que la dirección de ambos operandos así como el destino están especificados.

TABLA 2. Formatos de instrucción genérico.

CÓDIGO DE OPERACIÓN	MODO	DIRECCIÓN Y OPERANDO
Formato de instrucción		

En el ejemplo $ADDR1$ ambos operandos están en registros; no sólo eso, también el resultado se guarda en un registro. Pero esto no siempre será así: a veces el dato estará en memoria, e interpretando el campo dirección, con ayuda del campo modo, es como se obtiene la dirección efectiva.

2.2. MODOS DE DIRECCIONAMIENTO

Los punteros son valores que “apuntan” a la dirección de memoria en la que se encuentra el dato. Su uso mejora el rendimiento de operaciones repetitivas; es por ello que son varios los lenguajes de programación que permiten su uso. Esta forma de hacer referencia al operando constituye uno de los diferentes modos de direccionamiento. El uso de modos de direccionamiento ofrece flexibilidad sin aumentar el número de bits de la instrucción, permitiendo al programador con conocimientos suficientes generar código más compacto. De todas formas su uso también puede reducir el rendimiento; además, a menudo no se utilizan de forma efectiva por parte del compilador.

A continuación se describen siete modos de direccionamiento y se ve un ejemplo de su utilización

Directo El campo dirección de la instrucción (tanto para una transferencia como transformación del dato) contiene la dirección de memoria donde está el dato (dirección efectiva)

Inmediato El dato se especifica en la propia instrucción, se puede hablar de campo operando en lugar de campo de dirección.

Útil para inicializar registros a una constante.

- Indirecto** El campo dirección de la instrucción contiene la dirección de memoria en la cual se encuentra la dirección efectiva.
- Relativo** La dirección efectiva se forma sumando al campo dirección de la instrucción el contenido de un registro específico de la CPU, generalmente el Contador de Programa (PC).
- Práctico cuando sabemos a qué distancia de la actual instrucción se encuentra el dato deseado.*
- Indexado** Dirección efectiva se obtiene sumando al campo dirección de la instrucción (dirección base), el contenido de un registro (distancia entre esa dirección inicial y el dato). Puede tratarse de un registro especial, en este caso no se hace referencia (implícito). También puede ser uno cualquiera del bloque de registros (o una pareja); en este caso debe especificarse de manera explícita.
- Sirve, por ejemplo, para recorrer un array (incrementando el registro).*
- Registro** El contenido del registro es el dato (no hay dirección de memoria, no hay dirección efectiva).
- Registro-indirecto** El registro (o pareja de registros) contiene la dirección efectiva. Es decir, el registro contiene la dirección de memoria donde está el dato, por lo que este funciona como un apuntador a memoria.

Veamos un ejemplo. Supongamos que el código de operación indica “carga en el acumulador”; para cada uno de los modos de direccionamiento vistos, la interpretación del campo dirección lleva a una dirección efectiva y un dato tal y como se representa en la Tabla 3.

TABLA 3. Modos de direccionamiento. Ejemplo.

DIR.	CONTENIDO DE MEMORIA		
		
00FFH	CÓD. OPERACIÓN	MODO	
0100H	DIRECCIÓN: 0200H		
0101H	010AH		
		
0200H	0300H		
0201H	0F00H		
		
0250H	0400H		
0251H	0701H		
		
0300H	0120H		
0301H	100EH		
		
0450H	0890H		
		
	PC	0101H	
	R	0250H	
	MODO	DIR. EFECT.	DATO
	DIRECTO	0200H	0300H
	INMEDIATO	0100H	0200H
	INDIRECTO	0300H	0120H
	RELATIVO (a PC)	0301H	100EH
	INDEXADO (R)	0450H	0890H
	REGISTRO	—	0250H
	REGISTRO INDIRECTO	0250H	0400H

A menudo, tras ver los diferentes modos de direccionamiento, la pregunta que nos surge es ¿y esto cuándo se usa? Intentaremos ilustrar su utilidad poniéndonos en situación. Veamos los siguiente supuestos:

- **Supuesto #1:** necesito recorrer un bucle seis veces, para ello quiero inicializar a 6 un registro que utilizaré como contador.

Lo que conozco en este caso es el dato. Indicaré el operando (no una dirección). El modo de direccionamiento es inmediato.

$$(R1) \leftarrow 06$$

- **Supuesto #2:** quiero guardar en el acumulador el contenido de la posición de memoria 9100H.

Conozco la dirección del dato, por lo que ejecutaré una instrucción especificando la dirección efectiva. El modo de direccionamiento es directo.

$$(Acc) \leftarrow M[9100H]$$

- **Supuesto #3:** quiero copiar en el acumulador el valor del registro R1.

Sé dónde está el dato, pero no está en memoria, sino en un registro. El modo de direccionamiento en este caso es el modo registro.

$$(Acc) \leftarrow (R1)$$

- **Supuesto #4:** quiero sumar los elementos de un array. Dicho array se encuentra en la posición de memoria 9000H.

*En este caso la elección depende del número de elementos del array. Si solo fueran dos elementos, por ejemplo, podríamos utilizar un par de accesos a memoria indicando las direcciones 9000H y 9001H; en cuyo caso utilizaríamos (dos veces) direccionamiento directo. Sin embargo, si el número de elementos es mayor (supongamos 50), esto no es viable. En este caso nos interesa crear un pequeño bucle y un contador que permitan que en cada vuelta se sumen la suma parcial y el elemento que corresponda. Para ello inicializaremos un registro (supongamos el registro RP) con la dirección del primer elemento (dirección del array). Este registro es por tanto un puntero, y haremos que su valor se actualice (incremente) cada vez que se ejecute el bucle; de modo que el registro tiene en todo momento la dirección del operando. En este caso el modo de direccionamiento que se utiliza es por tanto **registro indirecto**.*

$$(Acc) \leftarrow (Acc) + M[(RP)]$$

- **Supuesto #5:** una subrutina toma un número de medidas, número especificado por el usuario cada vez que se invoca la subrutina. Dicha subrutina toma las medidas necesarias creando un array que almacena en memoria. La dirección del primer array (es decir, la dirección del primer elemento de la primera medida) se copia en la posición 9001H; la dirección del segundo array (dirección del primer elemento de la segunda medida) en 9002H,... Otra subrutina realiza una tarea con los datos de dichos arrays, pero necesita conocer la dirección del array. ¿Cómo pasarle la dirección del tercer array de medidas?

No sé cuántas son las medidas, ni dónde está el tercer array, lo que sé en este caso es donde está guardada la dirección de dicho array: en la posición 9003H. De modo que en este caso utilizaremos direccionamiento indirecto.

$$(Acc) \leftarrow M[9003H]$$

- **Supuesto #6:** un fragmento de código contiene instrucciones y datos. En un cierto punto hay una instrucción que realiza la suma utilizando como operandos dos datos especificados 3 y 4 posiciones antes ¿cómo se accede a dichos operandos?

*No sé dónde se cargará este código, pero sé que cuando llegue a la instrucción en cuestión, el Contador de Programa (PC) se actualizará e indicará la siguiente dirección, por lo que los operandos estarán 4 y 5 posiciones antes. En este caso el tipo de direccionamiento es **relativo** (a PC)*

$$(Acc) \leftarrow M[PC-5] + M[PC-4]$$

- **Supuesto #7:** quiero cargar en el acumulador el octavo elemento de un array, cuya dirección está en un registro (RP).

*Aunque desconozco la dirección del array sé que ésta está en el registro RP y conozco la posición relativa del elemento dentro del array. El direccionamiento en este caso es **indexado**.*

$$(Acc) \leftarrow M[(RP)+8]$$

2.3. CLASIFICACIÓN DE LOS TIPOS DE INSTRUCCIÓN

En función de la tarea que realizan, se clasifican las instrucciones en tres grupos:

1. Instrucciones de transferencia de datos
2. Instrucciones de procesamiento de datos
 - a) Aritméticas
 - b) Lógicas
 - c) De desplazamiento
3. Instrucciones de control de secuencia

En el Capítulo 3, se verán mnemónicos de instrucciones de los tres grupos.

2.4. INTERRUPCIONES

Imaginemos que estamos en casa viendo una película, cuando suena el teléfono. Pararíamos la película para ver quién ha llamado; tras atender la llamada, volveríamos a la tarea de ver la película en el punto en el que la habíamos dejado. Esto mismo pero extrapolado al mundo de la informática sería una **interrupción**. Cuando se da una interrupción, se rompe la ejecución normal del programa para ejecutar la Rutina de Servicio a la Interrupción (RSI²), volviendo tras ésta, a la ejecución normal del programa, al punto en el que estaba cuando sucedió la interrupción. En un principio vemos que tiene similitudes con la subrutina, sin embargo tiene también algunas diferencias cruciales:

- No la provoca una instrucción, sino una señal (externa/interna) no previsible.
- La dirección del programa de atención se facilita mediante hardware (no hay campo de dirección).

²En inglés la denominación es ISR: *Interrupt Service Request*, ya que es una petición de servicio por la interrupción

- Ante una interrupción hay que guardar la información del bloque de registros (o parte al menos), no sólo el PC. De manera genérica, se guarda el PC y los flags de estado (SR, Status Register)

2.4.1. Tipos de interrupciones.

Aunque sólo las interrupciones hardware cumplen las características que acabamos de enumerar, también se engloban en el grupo de las interrupciones otras que no lo son propiamente. Así podríamos diferenciar tres tipos de interrupciones:

- **Interrupciones hardware:** se dan cuando un recurso hardware requiere atención. Es asíncrona por lo que puede darse en cualquier momento, en general se dará durante la ejecución de una instrucción, por lo que primero se termina la instrucción máquina en curso, y luego se atiende. Puede deberse a diversas causas: temporizador, fallo de alimentación, dispositivos de E/S que tienen datos, etc.
- **Interrupciones software:** no son propiamente interrupciones ya que están introducidas por el programador. Se trata de instrucciones (INT) y por lo tanto se sabe cuándo van a suceder y además se dan de manera sincronizada.
- **Traps:** se trata de **Trampas** generadas por la CPU ante una condición de error como: la utilización errónea o ilegal de un conjunto de datos, la división por cero, direccionamiento no permitido, etc. De modo que, aunque no sea de manera intencionada, están introducidas por el programador, y están por tanto sincronizadas.

Sólo las interrupciones hardware son propiamente interrupciones, ya que no podemos prever cuándo sucederán y se darán de manera asíncrona. Hemos visto que pueden ser diversas las causas de una interrupción de este tipo; en ese caso ¿cómo sabe la CPU quién a solicitado una rutina de atención (quién ha interrumpido)? Hay tres opciones:

- **Multinivel:** cada dispositivo tiene una entrada asociada. Resulta sencillo de gestionar pero muy caro.
- **Línea única más polling:** una línea única indica a la CPU que algún dispositivo requiere atención, por lo que la CPU pasa a realizar un sondeo (pregunta uno a uno).
- **Vectorizadas:** en este caso, además de activar la señal de solicitud de interrupción, el dispositivo que interrumpe pone en el bus de datos un vector que lo identifica (o pone directamente la dirección de la rutina de atención requerida, como en la Figura 1). La CPU utiliza el vector indicado para buscar en una tabla la dirección de la rutina de servicio correspondiente.

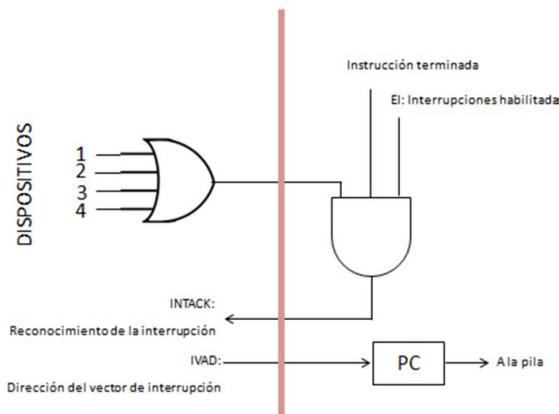


FIGURA 1. Interrupción hardware vectorizada.

2.4.2. Pasos en la gestión de la interrupción.

A continuación se describen los pasos que se han de dar en el momento en el que se produce una interrupción:

1. $SP \leftarrow SP-1$ se actualiza el SP (se hace sitio en la pila)
2. $M[SP] \leftarrow PC$ se guarda la dirección de retorno en la pila (PC: dirección de la siguiente instrucción a ejecutar una vez finalizado el servicio)
3. $SP \leftarrow SP-1$ se actualiza el SP (se hace sitio en la pila)
4. $M[SP] \leftarrow PSR$ se guarda la palabra de estado (valor del acumulador y de los flags de estado)
5. $EI \leftarrow 0$ se deshabilitan las interrupciones (en principio las interrupciones se deshabilitan de modo que mientras se está atendiendo una interrupción no pueda darse otra, pero hay interrupciones que debido a su importancia no son enmascarables)
6. $INTACK \leftarrow 1$ se acusa la recepción de la solicitud de interrupción (se acepta la interrupción)
7. $PC \leftarrow IVAD$ se carga en el Contador de Programa la dirección de la RSI (Rutina de Servicio a la Interrupción)

NB: téngase en cuenta que se ha realizado una descripción genérica pero no exhaustiva debido a que no se está teniendo en cuenta la arquitectura del procesador en cuanto, p.ej., el ancho del Bus de Direcciones. Según el ancho de este Bus, el acceso a la Pila puede requerir varios accesos: acceso para la parte alta de la dirección y acceso para la parte baja de la dirección. Idem dirección de Servicio.

Capítulo 3

LENGUaje MÁQUINA Y ENSAMBLADOR

La lengua es lo mejor y lo peor que poseen los hombres. (Anárcasis: siglo VI a.C)

RESUMEN. Los procesadores leen o escriben, de o en los buses, información en formato binario, es decir, ristras de "1"s o "0"s, empaquetados, p. ej., en bytes o words. Estos empaquetamientos constituyen el lenguaje máquina cuando representan las instrucciones que gobiernan al procesador. Estas instrucciones se representan bajo un formato comprensible acudiendo a un código mnemónico. Este código se conoce como lenguaje ensamblador.

Se tratarán conceptos como Macroinstrucción y Subrutina. La Macroinstrucción es una agrupación de instrucciones bajo un nombre, que es el invocado cada vez que se requiere ejecutar esa agrupación de instrucciones. La Subrutina también es una agrupación de instrucciones bajo un nombre pero su ejecución requiere de menor código. Tanto las Macroinstrucciones como las Subrutinas, pueden requerir de paso de parámetros para su ejecución.

Finalmente se verá cómo se trata un programa escrito en lenguaje ensamblador para generar el programa que debe ejecutar un procesador.

3.1. INTRODUCCIÓN

Un procesador entiende solamente códigos escritos como ristras de "Unos" y "Ceros", trabaja en el nivel más bajo, se conoce como nivel "Máquina". Un programador no puede plantearse el conocer los códigos máquina que componen todo el repertorio de instrucciones de un procesador. Por ello se recurre a crear un lenguaje "Mnemónico" que nos acerque al significado final de la instrucción. Este primer lenguaje mnemónico se denomina "Lenguaje Ensamblador (*assembly language*)". Representa "Símbólicamente" la codificación binaria de la máquina (*machine language*)

Se trata del primer nivel de acercamiento al lenguaje máquina y al programa escrito en este código se denomina "Programa Fuente". Se trata de un lenguaje de "bajo nivel" a diferencia de p.ej. "C" que es un lenguaje de

"alto nivel", pero que en ambos casos, los programas escritos en estos códigos, se denominan "Fuente". Este programa no es directamente entendible (interpretable) por el procesador.

Cuando se escribe un programa en lenguaje ensamblador, este tiene una significación directa con respecto al código máquina pero esto necesita de una conversión previa a lenguaje máquina. La conversión la realiza un programa denominado "Ensamblador (*assembler*)". Es decir, este programa realiza una lectura del mnemónico y genera su significado máquina en forma de "unos" y "ceros" (expresión binaria, octal o hexadecimal).

El programa generado por este traductor, se denomina "Programa Objeto" o "Programa Ejecutable".



FIGURA 1. Secuencia de tratamiento

Es importante resaltar la diferencia entre "intérprete" y "traductor". Un programa intérprete ejecuta las instrucciones según las va leyendo, es decir, va interpretando el código leído y va generando los comandos adecuados para el procesador. Esto en un principio sucedía con los primeros lenguajes Basic. Actualmente este nivel también se da con Java.

Un programa traductor, realiza una o varias lecturas del programa hasta que genera una información coherente de cara al procesador, en este momento genera el programa objeto que puede ser cargado en la memoria principal del computador y ser ejecutada directamente por el procesador. Un programa escrito en lenguaje Ensamblador, Fortran, Pascal, Algol, Basic (últimas generaciones), C, C++, Ada, etc., será traducido y generado su ejecutable.

El traductor de un lenguaje mnemónico, como el lenguaje ensamblador, se denomina "Ensamblador". El traductor de un lenguaje de alto nivel se denomina "Compilador". En ocasiones los Compiladores pueden generar un programa objeto mnemónico.

3.2. LENGUAJE ENSAMBLADOR (*Assembly Language*)

Como se ha comentado, el lenguaje ensamblador es la expresión mnemónica de un código máquina escrito en binario, octal o hexadecimal. Su

significación es directa, por lo tanto podríamos escribir en p.ej. hexadecimal directamente, pero esto exige un nivel de abstracción muy fuerte y nos haría perder la visión de lo que escribimos, además de ser muy dudoso que podamos aprender de memoria todos o casi todos los códigos.

Parece claro que es más fácil recordar que sumar es ADD que 24H, o que la instrucción de salto es JMP que su código 80H.

Otro aspecto importante es que si el programa debe realizar el salto a una dirección, resulta muy complicado el conocer cuál es la dirección en términos absolutos y su expresión en hexadecimal. Por ello es mejor recurrir a una expresión simbólica de la dirección, denominada “Etiqueta”, mediante la cual podamos reconocer los puntos de programa donde realiza el salto.

Es el Ensamblador quien realiza la interpretación de los códigos mnemónicos y los convierte a código máquina, además reconoce las etiquetas y las asigna un valor binario correspondiente a la dirección física de la memoria donde se realizará el salto.

Un aspecto negativo del lenguaje ensamblador es su baja portabilidad entre máquinas, un procesador admite solamente su propio ensamblador. Si pretendemos llevar (migrar) un programa realizado para el ensamblador de una máquina hacia otra, nos encontraremos que no va a funcionar.

Esta portabilidad normalmente se da entre procesadores de la misma familia pero en sentido ascendente, es decir, el programa fuente (en lenguaje ensamblador) realizado para el procesador más bajo de la familia, lín sup funcionará en los procesadores más altos (mayores prestaciones) pero no a la inversa. Esto es así porque cada nueva generación de procesadores incorpora nuevas instrucciones orientadas a mejorar el rendimiento con las nuevas prestaciones.

El problema de la portabilidad está mejor resuelto con los lenguajes de alto nivel como C (estándar) o Java, ya que el compilador (o traductor en caso de Java) va a realizar la adaptación al código máquina específico (o a la *Java Virtual Machine* caso de Java).

3.2.1. Uso del Lenguaje.

Un punto adicional y que suele tener sus controversias es el de si un programa escrito en lenguaje ensamblador es más eficiente que uno escrito en, p. ej., lenguaje C. Normalmente se aduce que en lenguaje ensamblador podemos acudir a bits de status del procesador y que esto no es posible en alto nivel. Además, se dice que al realizar la compilación de un programa en alto nivel, el compilador genera más instrucciones que las necesarias (las que se supone que escribiría un experto programador). Esto era así en

algunos compiladores, hoy en día los compiladores tienen la opción de optimización del código, de tal manera que podrían ser más eficaces que si lo realizara un programador experto, además y volviendo a la primera crítica, el repertorio de instrucciones permiten acceder a registros internos de la máquina. Resulta evidente que si se crearon los lenguajes de alto nivel fue para poder facilitar la expresión de conceptos y desarrollos de programa de cara al programador y que terceros programadores pudieran interpretar mejor lo que implementaba el programa. Por lo tanto, si es más fácil programar en alto nivel y resulta que hay compiladores optimizadores de código ¿por qué debemos seguir utilizando el lenguaje ensamblador?

Pues bien, la respuesta está en volver a interpretar el primer párrafo de este apartado. Hemos dicho que podemos tener acceso a registros internos y que el compilador optimiza, pues bien, si tenemos dudas o es claro que esto no se corresponde con nuestra realidad y necesitamos una optimización eficaz, será necesario que esa parte de código crítica, se realice en el lenguaje ensamblador de la máquina. Hay ciertos accesos a registros de la CPU que deben seguir una secuencia y, por lo tanto, deben ser tratados en lenguaje ensamblador para que no se introduzca ninguna instrucción no deseada que pueda introducir un retardo no permitido.

Como criterios de revisión de un programa escrito en alto nivel, puede apuntarse el someter al programa a un analizador de tareas/tiempo (*profiling*) durante un periodo largo y revisar qué tareas del programa se ejecutan más veces y si son las que consumen más tiempo de CPU que otras. En este caso podría plantearse el reescribir esa parte de código en ensamblador, evaluando previamente el tiempo necesario para llevarlo a cabo. Si este tiempo no fuera asumible por costos o tiempo, no deberá realizarse ningún cambio.

Por concluir este tipo de disquisiciones, comentar que los mayores apologistas del uso del ensamblador como recurso de optimización, dentro de un programa de alto nivel, se dieron a finales de los 60 y principios de los 70. Realizaron estudios de comparación y optimización llegándose a conclusiones lapidarias en favor del uso del ensamblador. Téngase en cuenta que se está trabajando con Fortran y comienzos de Algol y Pascal. El desarrollo de nuevos compiladores redujo dramáticamente estas diferencias.

Por último decir que, desde un punto de vista académico, resulta IMPRESCINDIBLE el iniciarse en el uso del lenguaje ensamblador por tener mejor vivenciada la arquitectura interna de un procesador.

3.2.2. Formato de las Instrucciones.

Se ha comentado que los ensambladores de los diferentes procesadores no son compatibles, pero esto no implica que no tengan un parecido muy grande. Pueden diferir los mnemónicos en alguna letra, aparecer nuevos mnemónicos, disponer de operandos ampliados, etc., pero seguirán teniendo un gran parecido. Fundamentalmente la estructura que presenta un programa en lenguaje ensamblador es la siguiente:

TABLA 1. Estructura de un programa en Lenguaje Ensamblador con un μ P8085

Etiqueta	Cod. Operación	Operandos	Comentarios
ACCIÓN:	MOV	A, #55H	;Mueve valor inmediato
	ADI	#AAH	;Suma val. Inm. a reg. A
	JZ	ACCION	;Instr. de Salto si Z=1
	ADI	#01H	;Suma val. Inm. a reg. A
	JZ	ACCION	;Instr. de Salto si Z=1
VARIABLE	DW	#AA55H	;Reserva espacio a este valor

Este ejemplo está expresado en instrucciones del microprocesador 8085 de Intel.

TABLA 2. Ejemplo con un μ P8088, 8086, ..., 80846

Etiqueta	Cod. Operación	Operandos	Comentarios
ACCIÓN:	MOV	AL,55H	;Mueve valor inmediato a AL
	ADD	AL,AAH	;Suma val. Inm. a reg. AL
	JZ	ACCION	;Instr. de Salto si Z=1
	ADD	AL,1	;Suma val. Inm. a reg. AL
	JZ	ACCION	;Instr. de Salto si Z=1
VARIABLE	DW	AA55H	;Reserva espacio a este valor

Este ejemplo está expresado en instrucciones del microprocesador 8088, 8086, 80286, 80386, 80486, Pentium, Pentium Pro y II de Intel.

TABLA 3. Ejemplo con un µP68000, ..., 68040

Etiqueta	Cod. Operación	Operandos	Comentarios
ACCIÓN	MOVE	#55H,D1	;Mueve valor inmediato a D1
	ADDI.B	#AAH,D1	;Suma val. Inm. a reg. D1
	BEQ	ACCION	;Instr. de Salto si Z=1
	ADDI.B	#1,D1	;Suma val. Inm. a reg. D1
	BEQ	ACCION	;Instr. de Salto si Z=1
VARIABLE	DC.W	\$AA55H	;Reserva espacio a este valor

Este ejemplo está expresado en instrucciones del microprocesador 68000, 68010, 68020, 68030 y 68040 de Motorola.

Desde un punto de vista genérico, se observa que los programas se enumeran, o se estructuran, en cuatro columnas o campos:

- **Columna nº 1:** Dedicada a la definición de las “Etiquetas”. Puede requerir de los “:” o no, dependiendo del procesador. Para un compilador es más fácil distinguir una etiqueta con los “:” en caso de estar ocupando una sola línea (no confusión con posible mnemónico ocupando una sola línea).

También esta columna se utiliza para denominar una variable a la que se va a reservar espacio.
- **Columna nº 2:** Dedicada a los “Códigos de Operación”. Difieren entre procesadores, p.ej. para “mover” contenido de Memoria a Registro y viceversa, Intel usa MOV, Motorola MOVE y Sparc LD para lo primero y ST para lo segundo.

También dedicada a “Reservar” espacio, p.ej. DW reserva una palabra (16 bits).
- **Columna nº 3:** Dedicada a los “Operandos” necesarios del código de operación. En el caso de haber realizado reserva de espacio, esta columna se utiliza para dar un valor inicial a la variable. Según los distintos procesadores, hay diferencias entre Destino Origen u Origen Destino.
- **Columna nº 4:** Dedicada a los “Comentarios”. Aquí se expresará con la mayor claridad posible, las acciones que se van a realizar.

3.2.3. Pseudoinstrucciones.

Estamos acostumbrados a ver instrucciones orientadas al procesador, es decir, qué acciones va a realizar; pero también pueden existir instrucciones orientadas al Ensamblador, estas le van a pedir al Ensamblador que realice determinadas acciones, como p.ej., asignar espacio en la memoria. Estas instrucciones se denominan “Pseudoinstrucciones” o también “Directivas” del Ensamblador. Veamos como son algunas de las pseudoinstrucciones que podemos encontrar en los Ensambladores:

- EQU: asigna un nombre simbólico a un valor, a un registro o a una expresión. No se puede redefinir.

TABLA 4. EQU

AÑO	EQU	2003	;En vez de usar 2003, se utilizará AÑO
ENERO	EQU	1	
FEBRERO	EQU	ENERO+1	;Admite expresiones aritméticas

- SET: asigna un nombre simbólico a un valor, a un registro o a una expresión. Se puede redefinir.

TABLA 5. SET

LIMITE	SET	120	;Asigna un valor numérico
LIMITE	SET	80	;Asigna un valor numérico
VALOR	SET	R2	;Asigna el valor del registro R2

- END: terminar el programa Ensamblador.
- DB: reserva uno o más bytes de memoria inicializados. Símbolo, Cadena de Caracteres o Expresión.

TABLA 6. DB

VALOR1	DB	#01H	;Reserva un byte
VALOR2	DB	'ESPERA'	;Reserva 6 bytes

- DW: reserva una o más palabras de memoria inicializadas. Símbolo, Cadena de Caracteres o Expresión.

TABLA 7. DW

VALOR2	DW	#1234H
--------	----	--------

- ORG: asigna dirección, en memoria, de comienzo de un bloque de programa.

TABLA 8. ORG

```
ORG    100H
ORG    START
```

- PUBLIC: indica que esta variable puede ser utilizada en todos los archivos de programas que componen el programa total.

TABLA 9. PUBLIC

```
VARI1  PUBLIC
```

- EXTERN: variable definida en otro módulo de programa y que se está utilizando en este módulo. Avisa al Ensamblador para que no de alarma.

TABLA 10. EXTERN

```
VARI2  EXTERN
```

- BIT, CODE, DATA, IDATA, XDATA: Direcciones de Bits, Datos o Código

TABLA 11. Direcciones de Bits, Datos o Código

SIMBOLO1	BIT	20H
SIMBOLO2	CODE	1E00H ;Código 0000H .. FFFFH
SIMBOLO3	DATA	126D ;0..127, 128..255 (SFR)
SIMBOLO4	IDATA	8AH ;0 .. 255
SIMBOLO5	XDATA	1E00H ;Datos 0000H .. FFFFH

3.3. MACROINSTRUCCIONES (Macros)

Durante el proceso de escritura de un programa, es muy normal el tener que escribir el mismo código varias veces, esto puede resultar tedioso siempre y cuando no sea un código de pocas líneas. Ante esta situación, pueden plantearse varias estrategias:

- Llamada a un Procedimiento o Subrutina: Presenta un pequeño inconveniente, cada llamada necesita de dos instrucciones adicionales, la llamada y el retorno. Problema ante tiempos de ejecución críticos ya que ralentiza la ejecución del programa¹. Este proceso será tratado en la Sección 3.4.
- Uso de Macroinstrucciones: Se asigna un “Nombre” que se define como “Macro” y es donde va el cuerpo de instrucciones que se repiten. En el resto de lugares del programa se escribe el nombre de la macroinstrucción que va a suplir al código de programa repetido.

Las “Macros” se definen, normalmente de la siguiente manera (válido para 8085, 8051, 8088, 8086, 80286, …, Pentium II):

TABLA 12. Macro

MIMACRO	MACRO	:Cabecera de definición
	Instrucción1	:Cuerpo de la macro
	Instrucción2	
	
	InstrucciónN	
	ENDM	:Pseudoinstrucción de finalización

Por lo tanto, el programa se escribirá de la siguiente manera:

TABLA 13. Inserción de una Macro

InstrucciónA
InstrucciónB
MIMACRO
InstrucciónC
InstrucciónD
.....
MIMACRO
InstrucciónJ
.....
END

Veamos cómo trata el Ensamblador este tipo

¹De todas formas, no necesariamente el uso de Subrutinas penaliza la ejecución del programa, teniendo en cuenta que la velocidad de los procesadores actuales es muy alta. Además, puede ser más necesario el optimizar memoria que el ejecutar dos instrucciones (salto/retorno) por llamada a Subrutina.

1. El Ensamblador realiza una lectura del texto y cuando encuentra la definición de una Macro, guarda en una tabla el cuerpo de la definición, para usarla posteriormente.
2. Cuando encuentra el nombre de la macro como código de operación, es decir, cuando se usa, substituirá el nombre por el cuerpo en el código a ejecutar. Este proceso se denomina “Expansión” de la Macro.

Es importante realizar las siguientes precisiones:

- La expansión de la Macro se efectúa durante el proceso de Ensamblado y no durante el proceso de Ejecución del programa.
- Antes de “usar” la Macro, debe estar definida.

La diferencia fundamental con el uso de Procedimientos o Subrutinas, es que en estas, el cuerpo de instrucciones invocado, solamente se escribe una vez en el código y cuando es invocado este procedimiento, el programa realiza un salto a la posición donde se ejecuta el procedimiento y al finalizar vuelve el programa a la instrucción siguiente a la que invocaba al procedimiento. Requiere guardar la dirección de retorno del procedimiento. Como generalidad, nos podemos encontrar que las instrucciones requeridas en una Macro, pueden necesitar de parámetros con diferente valor según el punto de ejecución. Esto se resuelve dotando a la definición de la Macro de un conjunto de parámetros que van a permitir el paso de valores diferentes. En la definición se introducen los Parámetros Formales y en la llamada los Parámetros Reales.

Cada Ensamblador dispone de un número máximo de parámetros. Van separados por comas.

TABLA 14. Parámetros en las Macros

MIMACRO	MACRO	A,B, ..,N	;Cabecera y parámetros “Formales”
	Instrucción1		;Cuerpo de la macro
		
	InstrucciónN		
	ENDM		;Pseudoinstrucción de finalización
		
	MIMACRO	A1,B1, ..,N1	;Llamada y Parámetros “Reales”
	InstrucciónD		
		
	MIMACRO	A2,B2, ..,N2	;Llamada y Parámetros “Reales”
	InstrucciónJ		
		
	END		

Dentro de una Macro, podemos encontrarnos “pseudoinstrucciones” como por ejemplo REPT cuyo objetivo es repetir una o varias instrucciones “N” veces (ver Tabla 15).

TABLA 15. Pseudoinstrucciones

MIMACRO	MACRO		;Cabecera de definición
	Instrucción1		;Cuerpo de la macro
	REPT	N	
	Instrucción2		
	ENDM		;Finalización de Repetición
	InstrucciónN		
	ENDM		;Pseudoinstrucción de finalización

Otra pseudoinstrucción importante es EXITM, cuando el Ensamblador encuentra esta pseudoinstrucción, termina la expansión de la Macro. Se utiliza en ensamblado condicional.

TABLA 16. Otras pseudoinstrucciones

MIMACRO	MACRO	A	;Cabeza de definición
	IF NUL	A	;Condición sobre parámetro A
	EXITM		
	ENDIF		
	REPT	N	
	NOP		;Repite "A" veces
	ENDM		;Finalización de Repetición
	ENDM		;Pseudoinstrucción de finalización

En una Macro se admiten “etiquetas” para poder realizar lazos. Estas etiquetas deben ser declaradas como “Locales” para que no existan problemas de duplicidad en nombres y no puedan ser accedidas por zonas de programa externas a la Macro.

TABLA 17. Etiquetas en Macros

MIMACRO	MACRO	ITER	;Cabecera de definición y parámetro
	LOCAL	LAZO	;Declaración etiqueta Local
	MOV	R7,#ITER	
LAZO:	DCR	R7	
	JZ	LAZO	
	ENDM		;Pseudoinstrucción de finalización

En una Macroinstrucción, pueden definirse más Macros, es decir, pueden anidarse más Macroinstrucciones, definidas las etiquetas de las Macros interiores como Locales.

Las Macroinstrucciones pueden ser recursivas, es decir, llamarse a si mismas N veces. El número de llamadas es diferente por cada Ensamblador. Una cifra orientativa es 9.

3.4. SUBRUTINAS

3.4.1. Introducción.

Una Subrutina puede definirse de la siguiente manera:

- Porción de código que realiza una operación en base a unos valores dados como parámetros y que puede ser invocado desde cualquier parte del código, incluso desde sí misma (contexto de lenguaje ensamblador)

Para aquellos que hayan tenido contacto con los Lenguajes de Alto Nivel, las Subrutinas son, en lenguaje de bajo nivel, lo que las funciones o procedimientos en el de alto nivel.

El uso de Subrutinas aporta diversas ventajas:

1. División del problema en tareas más fáciles de escribir y depurar
2. Evita código redundante
3. Encapsulamiento del código. *Las tareas se comunican a través del paso de parámetros y resultados, por lo que un cambio en una de las tareas no implica cambios en el resto del programa. Además permite su reutilización en más de un programa (bibliotecas)*

3.4.2. Generalidad.

A partir de lo que acabamos de comentar deducimos que es necesario dotar a la Subrutina de cierta generalidad, de modo que una misma tarea se realice con diferentes datos (parámetros). Además, ha de poder ser invocada en diferentes momentos (ha de poder ser llamada desde diferentes partes del programa principal); por lo que se hace necesaria una forma de pasar operandos y resultados.

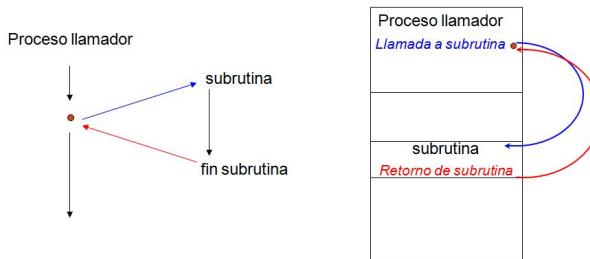


FIGURA 2. Llamada y retorno de subrutina.

El proceso llamador, llama a la Subrutina, pasándole unos parámetros para que ésta realice los cálculos pertinentes y le devuelva un resultado; tras lo cual seguirá su ejecución. Por lo tanto, es necesario guardar la dirección de retorno (dirección desde donde se invocó la Subrutina), para poder volver a ese punto una vez finalice la ejecución de la Subrutina. La Figura 2 muestra este proceso.

Atendiendo a definición de Subrutina, vemos que ésta debe poder ser llamada desde cualquier parte del código, incluso desde una Subrutina. Supongamos que durante la ejecución de un cierto código es invocada la Subrutina A. Se pasa a ejecutar el código correspondiente a dicha Subrutina, pero dentro de esta se llama a la Subrutina B (ver Figura 3).

La ejecución de la Subrutina B ha de terminar para poder volver al punto en el que fue llamada y así seguir con la ejecución de la Subrutina A. Cuando ésta también termine, se vuelve al punto del proceso llamador desde el que se invocó. A esta posibilidad que se les da a las Subrutinas de llamarse unas dentro de otras se le denomina anidamiento de Subrutinas. Fijémonos que siempre ha de terminar primero la última que se llamó. La Figura 3 muestra las llamadas y ejecuciones de las Subrutinas A y B..

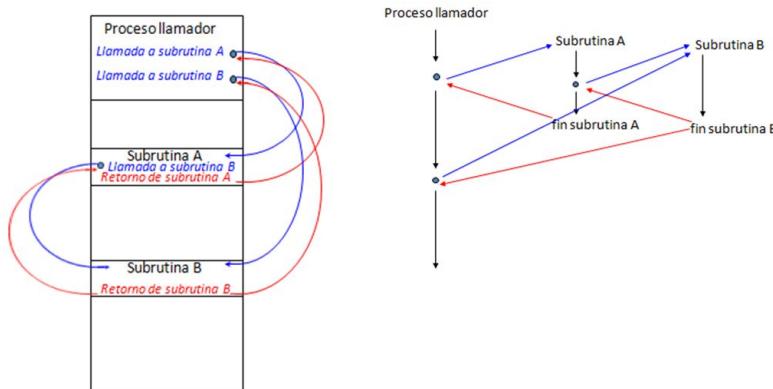


FIGURA 3. Subrutinas anidadas.

Veamos, a continuación, un ejemplo de escritura de una Subrutina y cómo es llamada desde cualquier parte del programa. Entre las instrucciones para el control de secuencia de un programa encontramos las correspondientes a la llamada y retorno de Subrutina. Atendiendo a los mnemónicos recomendados por la norma IEEE 694, éstas son: **CALL** etiqueta para la llamada, y **RET** para el retorno.

La llamada a una Subrutina se muestra en la Tabla 18.

TABLA 18. Llamada a subrutina

InstrucciónA	
InstrucciónB	CALL MISUBRUTINA
InstrucciónC	
InstrucciónD	
.....	
InstrucciónJ	CALL MISUBRUTINA
.....	
InstrucciónM	CALL TUSUBRUTINA
.....	
END	

El cuerpo de una Subrutina puede observarse en la Tabla 19.

TABLA 19. Cuerpo de una Subrutina

MISUBRUTINA	Instrucción1	;Cuerpo de la subrutina
	Instrucción2	
	
	InstrucciónN	
RET		;Pseudoinstrucción de finalización

3.4.3. Gestión de la Subrutina.

3.4.3.1. Dirección de retorno.

En la llamada se utiliza una etiqueta, como se ha visto en la Tabla 18, en lugar de la dirección absoluta, ya que por lo general ésta será desconocida. El Ensamblador se encarga de traducir esta etiqueta al pasar del lenguaje ensamblador a lenguaje máquina, y sustituirla por la dirección correspondiente (vease la Sección 3.5).

Fijémonos que en el momento de la llamada, ha de guardarse el valor que en ese momento tiene el *Contador de Programa* (PC), ya que este registro tiene la dirección de la siguiente instrucción a ejecutar. Esta dirección, que llamaremos dirección de retorno, es la que nos permitirá retomar la ejecución en el punto que le corresponde, una vez terminada la Subrutina. Despues de salvar el valor del PC, se carga en éste la dirección de la Subrutina en sí; dirección que, como decíamos, viene indirectamente especificada a través de la etiqueta.

Con la nueva dirección (la de la Subrutina) cargada en el PC, se lleva a cabo la ejecución de las instrucciones que la componen, hasta llegar a la última: RET. Es el momento de volver al punto en el que se llamó, pero ¿cuál es éste? Ya hemos tenido en cuenta que la dirección de retorno ha de guardarse antes de pasar a ejecutar la Subrutina, pero ¿dónde está este valor? Hemos pasado por encima este detalle que ahora veremos, resulta ser importante. Analizamos las diferentes opciones:

- ¿En memoria? No parece buena idea guardar la dirección de retorno en memoria, ya que RET no toma parámetros, por lo que no puede indicar dónde está guardado.
- ¿En un registro? Podríamos definir un registro que se utilice para esto, siempre el mismo, de modo que no necesite direccionarse; pero el uso de un único registro impediría el anidamiento de Subrutinas, por lo que también lo descartamos.

- ¿En la pila? Aunque la pila es parte de la memoria, tiene sus particularidades: funciona como una estructura LIFO (*Last Input First Output*) y la posición del último elemento (*Top Of Stack*, TOS) está siempre apuntada por el Puntero de Pila (*Stack Pointer*, SP), que se actualiza automáticamente. La pila empieza al final de la memoria y crece hacia posiciones inferiores; de modo que el SP decremente su valor cada vez que se introduce un elemento en la pila, y lo incrementa cuando sale. Por lo tanto, la pila parece el lugar ideal para guardar la dirección de retorno, ya que el funcionamiento como LIFO es el adecuado para el anidamiento de Subrutinas y su dirección está siempre disponible, sin necesidad de que se especifique.

Esta última opción parece la más adecuada, pero hay que prestar atención a un detalle: la pila puede utilizarse durante la ejecución de una Subrutina. Es responsabilidad del programador que al término de la Subrutina la pila quede como estaba al principio, para garantizar el correcto funcionamiento del anidamiento.

3.4.3.2. *Paso de parámetros.*

En la definición de la Subrutina ha de especificarse qué parámetros se van a pasar y en qué orden, de qué tipo son, si se van a pasar por valor (el parámetro) o por referencia (la dirección del mismo) y dónde se dejarán. De manera análoga a lo visto para la dirección de retorno, también podríamos plantearnos el paso de parámetros a través de memoria, de registros o de la pila.

- Paso de parámetros a través de registro

Funcionamiento: el programa llamador y el llamado asumen que los parámetros se almacenan en ciertos registros específicos. Antes de la instrucción de llamada, el programa llamador deposita los valores pertinentes en estos registros y tras la llamada, la Subrutina comienza a procesarlos directamente.

La desventaja está en el número limitado de registros de propósito general, que hace este método muy eficiente con Subrutinas con pocos parámetros y un único resultado (ya que el procesador no requiere almacenar datos en memoria), pero inviable en el caso general.

- Paso de parámetros a través de memoria

Se define una zona de memoria conocida tanto para el programa llamador como para el llamado y en ella se depositan los parámetros y resultados para su intercambio.

Esta opción es apta para un número arbitrario de parámetros, ya que a diferencia del paso a través de registro, no hay limitaciones de espacio. Sin embargo, se requieren múltiples espacios de parámetros para posibilitar las Subrutinas anidadas y estas zonas han de estar previamente definidas y reservadas para este fin.

□ Paso de parámetros a través de la pila

Parámetros y resultados son valores temporales que sólo tienen vigencia en un periodo concreto; por lo que sería más eficiente reservar el espacio de manera dinámica, tal y como sucede en la pila. Además, el orden de creación y destrucción de estos parámetros en llamadas anidadas es el adecuado para los mecanismos de gestión de la pila.

Siendo conocido el orden de almacenamiento en la pila de parámetros, espacios para resultados, etc. el problema es cómo acceder a los diferentes puntos de esta zona de parámetros. La cuestión no es baladí, ya que aunque en principio pensaríamos en el direccionamiento indexado (sec:ModosDir) haciendo uso del registro SP (SP + desplazamiento), no debemos olvidar que la pila puede sufrir modificaciones durante la ejecución de la Subrutina, por lo que no está garantizado que el valor SP tenga un valor constante durante el proceso.

Es decir, parece que la pila es el lugar idóneo para el intercambio de parámetros y resultados; parece también que el direccionamiento indexado podría ser adecuado, pero se hace patente la necesidad de definir una estructura y una forma de referenciarla; lo que nos lleva al bloque de activación y su puntero.

3.4.3.3. Bloque de activación .

En el momento de hacer la llamada a la Subrutina, se copian en la pila la dirección de retorno, los parámetros, los valores de los registros que se utilizarán durante la ejecución de la Subrutina y se dejan también posiciones libres necesarias para gestionar la ejecución de la misma (para variables locales, parámetros temporales o resultados parciales). Para poder acceder a la zona de parámetros y resultados con desplazamientos constantes, las primeras instrucciones copian el valor de SP en otro registro (previa copia de su valor en la pila).

Se denomina **bloque de activación** a la porción de memoria de la pila que contiene el espacio para los resultados, los parámetros, la dirección de retorno y la copia del valor original del registro en el que se guardará el SP; y **puntero al bloque de activación** a este registro que se utilizará para

acceder a los distintos puntos del bloque de activación mediante direccionamiento indexado. Al terminar la ejecución de la Subrutina, la estructura de datos creada en la pila se deshace en orden inverso.

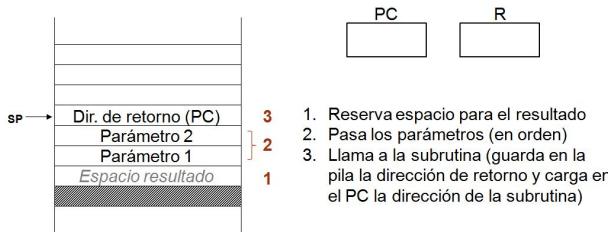


FIGURA 4. Gestión del bloque de activación. Pasos en el programa llamador.

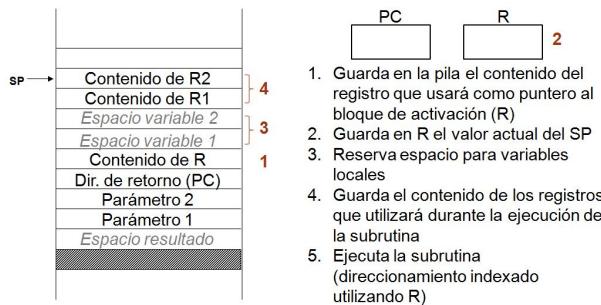


FIGURA 5. Gestión del bloque de activación. Pasos en la subrutina.

En las figuras 4 y 5 se describen los pasos que se dan tanto por parte del programa llamador como de la Subrutina, antes de la ejecución de la misma. Una vez obtenidos los resultados, estos han de copiarse en el lugar indicado para, a continuación, proceder a deshacer la estructura creada. Los pasos implicados se indican en las figuras 6 y 7

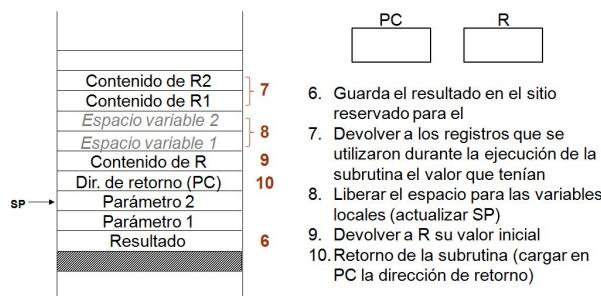


FIGURA 6. Gestión del bloque de activación. Pasos en la subrutina al término de la misma.

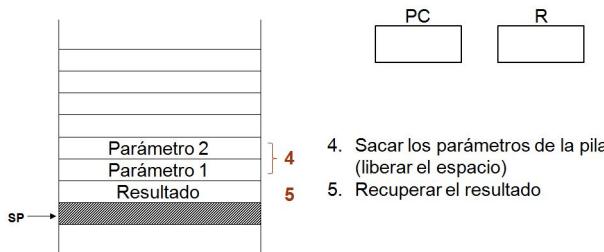


FIGURA 7. Gestión del bloque de activación. Pasos en el programa llamador al término de la subrutina.

3.5. ENSAMBLADOR (*Assembler*)

Desde un punto de vista general, el proceso de ensamblado se encarga de realizar las siguientes tareas:

- Traducir todas las instrucciones en lenguaje ensamblador al código máquina del procesador objetivo.
- Resolver todos los saltos a “etiquetas”, es decir, asignar las direcciones “simbólicas” de salto.
- Expandir las “macros” que aparecen en el programa.
- Interpretar las Directivas como p.ej. DB, asignando espacio.

La realización de estos procesos se lleva a cabo en lo que se conoce como “Ensamblado de dos Pasadas”. La idea fundamental que transmite este proceso es que, mediante dos pasadas (dos lecturas completas del programa fuente), resuelve todos los puntos enumerados anteriormente.

3.5.1. Primera Pasada.

En este proceso se tratan los siguientes puntos:

- Se identifican las “etiquetas” que constituyen los saltos, para resolver el problema de **“las referencias adelante”**.

TABLA 20. Referencias adelante

```

MOV  A,1EAFH
DCR  A
JZ   FIN
..
FIN: MOV  B,#55H
      END

```

- Se identifican los símbolos que constituyen las pseudoinstrucciones.

TABLA 21. Identificación de pseudoinstrucciones

AÑO EQU 2003

- Se identifican los símbolos que constituyen las “variables” que van a estar almacenadas en memoria de datos.

TABLA 22. Identificación de símbolos de variables

STA VALOR1

- Se identifican las Macroinstrucciones.

TABLA 23. Identificación de macros

```

MIMACRO  MACRO
         MOV      R7,#AÑO
         MOV      A,R7
         ADI      #01H
         MOV      R7,A
         ENDM

```

- Se identifican las Subrutinas

TABLA 24. Identificación de subrutinas

```

MISUBRUTINA  MOV  R7,#AÑO
              MOV  A,R7
              ADI  #01H
              MOV  R7,A
              RET

```

Para realizar una identificación completa de los procesos mencionados, se crea la “Tabla de Símbolos”, donde se incorporan todos los símbolos encontrados junto con un valor numérico, que identifica su posición en el programa fuente; este valor numérico se conoce como “Contador de Posición de Instrucciones” (ILC: *Instruction Location Counter*). Esta variable comienza con valor “0” y se incrementa en el valor necesitado por cada instrucción leída; p. ej., si encuentra:

TABLA 25. Creación de la Tabla de Símbolos

```
LAZO1: SUI #01H ;Resta 1 al reg. A
        JNZ LAZO1 ;Salta si Z=0
```

Incrementa en “2” por ser este el espacio necesitado por la instrucción SUI. Incrementa en “3” por ser este el espacio necesitado por la instrucción JNZ, esto es:

Código de Operación (SUI) + Operando (01)

Código de Operación (JMP) + Dirección L + Dirección H

Nota: Puede suceder que haya símbolos que no se encuentran en este fichero de programa, estos símbolos se denominan “Referencias no resueltas”, esto normalmente es debido a que están contemplados en otro fichero de programa, como se mencionará más adelante. Se resuelven con el Enlazador (*Linker*).

En la primera pasada, se crean las siguientes tablas: Tabla de Símbolos, Tabla de Pseudoinstrucciones y Tabla de Códigos de Operación.

- **Tabla de Símbolos:** Donde están ubicados los símbolos (etiquetas) y su posición de línea en el módulo.
- **Tabla de Pseudoinstrucciones:** Contiene los símbolos y su valor asignado. Indicará si los símbolos son Locales o Globales (accesibilidad). Se indica el tamaño si es necesario.
- **Tabla de Código:** Donde se incluye el mnemónico, operandos, código máquina (Hex) y la longitud de la instrucción.

3.5.2. Segunda Pasada.

El objetivo de la segunda pasada es el generar el Programa Objeto. En esta operación, el ensamblador debe generar información adicional que será útil al programa denominado “Enlazador”. Este necesitará información

para poder unir varios programas objeto y generar un único Programa Ejecutable. Se leen las líneas de código, generando el código máquina y se asigna el valor de nº de línea de los símbolos referenciados, esto último se toma de la tabla de símbolos. Estos símbolos tendrán una “dirección relativa” al módulo, teniéndose que reubicar en el “enlazado”.

En esta segunda pasada, es donde se notifican los errores, estos pueden ser los siguientes:

1. Se utilizó un símbolo pero no se definió.
2. Se definió un símbolo más de una vez.
3. El mnemónico ubicado en el campo de código de operación no se corresponde con el repertorio de instrucciones.
4. Faltan operandos a un código de operación.
5. Un número Octal vale 8 o más.
6. Un número Hexadecimal difiere de 0..9, A..F.
7. Uso incorrecto de registros .
8. Falta la instrucción END.

3.5.3. Estructura de un Módulo Objeto.

Resumiendo lo visto hasta este momento, podemos decir que la estructura que crea el compilador, después de la segunda pasada, presenta el aspecto reflejado en la Tabla 26.

TABLA 26. Estructura de un Módulo Objeto

Fin de Módulo
Diccionario
de
Reubicación
Instrucciones Máquina
y
Constantes
(La única que se cargará en Memoria)
Tabla de Referencias Externas
Tablas de Puntos de Ingreso
Identificación

La Tabla 26 contiene los siguientes campos:

- Identificación:** Contiene el Nombre del módulo y datos auxiliares (fecha ensamblado, longitud de las partes del módulo, etc..).

- **Puntos de Ingreso:** Los símbolos declarados como PUBLIC (pseudoinstrucción) y su dirección además puede ser el nombre del módulo si este es una Subrutina o procedimiento y su dirección.
- **Referencias Externas:** Símbolos utilizados en este módulo pero que estarán declarados en otro (si no “error”). Declarados como EXTERN (pseudoinstrucción). Esta tabla junto con la anterior pueden ser la misma.
- **Instrucciones Máquina y Constantes:** Contiene el código Objeto del programa.
- **Diccionario de Reubicación:** Suma constantes de reubicación a las instrucciones que disponen de direcciones de memoria.
- **Fin de Módulo:** Puede contener un Código de verificación de errores y la dirección absoluta de comienzo de ejecución del módulo.

3.6. ENLAZADO Y CARGA

Es muy normal que un programa no se escriba en un solo fichero sino que se escriba en varios, según la funcionalidad requerida. Estos ficheros se denominan “Módulos” o también “Tareas”.

Cada módulo va a realizar una serie de funciones específicas y diferenciadas del resto de módulos. El conjunto de módulos constituye la funcionalidad global del programa, por lo tanto, para que sean operativos, es necesario que un programa, denominado “Enlazador” (Linker) realice la unión de todos los módulos para ser cargados en memoria como un programa único.

También se requerirá el uso del enlazador, en el caso de que se invoque a una función de librería, trayéndola de esta e incorporándola al programa objeto a ser cargado en la memoria.

Cuando se realice un cambio en un programa fuente, solo será necesario ensamblar este módulo ya que el enlazador trabajará con los ficheros objeto no modificados y el nuevo fichero objeto.

La Figura 8 muestra el proceso seguido hasta la obtención de un programa ejecutable.

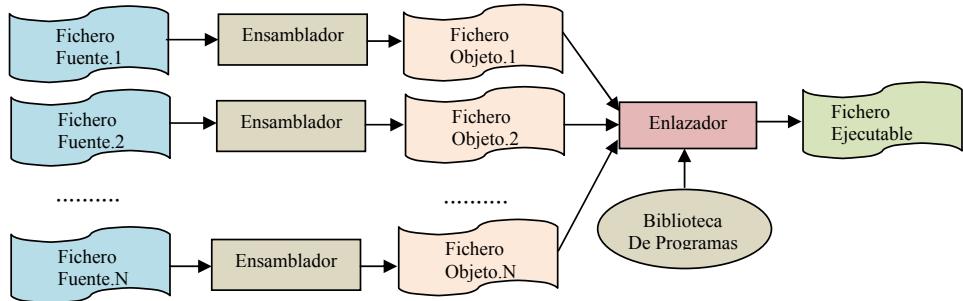


FIGURA 8. Esquema de Ensamblado – Enlazado - Carga

Resumiendo, la actividad desarrollada por el Enlazador, se centra en los siguientes puntos:

- Búsqueda en la Biblioteca de programas (Librería) para añadir al programa final la función de librería referenciada.
- Adjudica las direcciones de memoria que van a ocupar todos los módulos de programa y “reubica” sus instrucciones ajustando las referencias absolutas.
- Resuelve las referencias entre ficheros. Adjudica la dirección de las referencias.

Por situar el problema, supongamos que partimos de cuatro módulos realizados en lenguaje ensamblador:

TABLA 27. Ejemplo con cuatro Módulos Objeto

Módulo 1	Módulo 2	Módulo 3	Módulo 4
ORG 0000H	ETI2: LDA DIR1	ETI3: LDA DIR2	ETI4: LDA DIR3
JMP ETI0	ADI #02	ADI #02	ADI #0BH
ORG 100H	CMP #AAH	CMP #1CH	CMP #7BH
ETI0: MOV A,#55H	JNZ ETI3	JNZ ETI3	END
ETI1: ADI #01	JMP ETI4	JMP ETI4	
JNZ ETI1	END	END	
JMP ETI2			
END			

TABLA 28. Cont. Ejemplo con cuatro Módulos Objeto

Módulo 1		Módulo 2		Módulo 3		Módulo 4	
12	ETI2H	12	ETI4H	12	ETI4H	6	7BH
11	ETI2L	11	ETI4L	11	ETI4L	5	CMP
10	JMP	10	JMP	10	JMP	4	0BH
9	ETI1H	9	ETI3H	9	ETI3H	3	ADI
8	ETI1L	8	ETI3L	8	ETI3L	2	DIR3H
7	JNZ	7	JNZ	7	JNZ	1	DIR3L
6	1	6	AAH	6	1CH	*0	LDA
*5	ADI	5	CMP	5	CMP		
4	55H	4	2	4	2		
*3	MOV	3	ADI	3	ADI		
2	DIR0H	2	DIR1H	2	DIR2H		
1	DIR0L	1	DIR1L	1	DIR2L		
0	JMP	*0	LDA	*0	LDA		

El compilador crea cuatro módulos con comienzo, cada uno, en la posición “0000”, tal y como puede observarse en la Tabla 28.

Los Enlazadores requieren normalmente dos pasadas. La actividad desarrollada en cada pasada se resume en lo siguiente:

- Primera pasada:
 1. Se leen todos los módulos Objeto
 2. Se construye una tabla de nombres y longitudes de módulos
 3. Se construye una tabla de símbolos global
- Segunda pasada:
 1. Se leen los módulos Objeto
 2. Se reubican
 3. Se enlazan uno a uno, dejando, normalmente, un espacio de direcciones lineal

El ejemplo que se muestra a continuación, ver Tabla 29, realiza el tratamiento de cuatro módulos objeto que son enlazados.

Una vez que se han dispuesto de manera contigua los módulos, el enlazador resuelve las referencias, dando direcciones absolutas.

Los valores de DIR1, DIR2 y DIR3, los asigna el enlazador del espacio de memoria de datos.

El resultado se muestra en la segunda columna, que es la que sería cargada a memoria por el “Cargador” (Loader) de programas.

TABLA 29. Cont. Ejemplo con cuatro Módulos Objeto

119	123H - 00H		142	7BH			142	7BH			
118	123L - 7BH		141	CMP		119	00H		141	CMP	
117	JNZ		140	0BH		118	7BH		140	0BH	
116	AAH		139	ADI		117	JNZ		139	ADI	
115	CMP		138	DIR3H		116	AAH		138	DIR3H	
114	2		137	DIR3L		115	CMP		137	DIR3L	
113	ADI		*136	LDA		114	2		*136	LDA	
112	DIR1H		135	136H - 00H		113	ADI		135	00H	
111	DIR1L		134	136L - 88H		112	DIR1H		134	88H	
*110	LDA		133	JMP		111	DIR1L		133	JMP	
109	110H - 00H		132	123H - 00H		*110	LDA		132	00H	
108	110L - 6EH		131	123L - 7BH		109	00H		131	7BH	
107	JMP		130	JNZ		108	6EH		130	JNZ	
106	102H - 00H		129	1CH		107	JMP		129	1CH	
105	102L - 66H		128	CMP		106	00H		128	CMP	
104	JNZ		127	2		105	66H		127	2	
103	1		126	ADI		104	JNZ		126	ADI	
*102	ADI		125	DIR2H		103	1		125	DIR2H	
101	55H		124*	DIR2L		*102	ADI		124*	DIR2L	
*100	MOV		123	LDA		101	55H		123	LDA	
...		122	136H - 00H		*100	MOV		122	00H	
2	100H -- 00H		121	136L - 88H			121	88H	
1	100L -- 64H		120	JMP		2	00H		120	JMP	
0	JMP					1	64H		0	JMP	

3.6.1. Enlazado Dinámico.

Todo lo visto anteriormente se trata de un enlazado que se realiza antes de cargarse el programa a memoria y queda fija la estructura hasta que sea cambiada por actualización voluntaria por parte del programador. Podemos verlo como un enlazado estático.

Se abre una posibilidad de realizar otra estrategia que se denomina “Enlazado Dinámico”. Consiste en enlazar un módulo cuando este es invocado por otro módulo que ya está residente en Memoria Principal y se está ejecutando. De esta manera, si disponemos de un programa muy extenso y partido en muchos módulos, realizaremos el enlazado de los módulos básicos para el arranque de la aplicación, dejando el resto a la dinámica del propio programa. La Figura 9 muestra este proceso.

Una de las ventajas que trae este tipo de estrategia es que sacamos más rendimiento de la memoria virtual y solamente enlazaremos módulos cuando estos sean invocados. Puede haber módulos que rara vez deban de ser utilizados, como por ejemplo ciertas llamadas de error.

Recurriendo a un Sistema Operativo muy extendido en el mundo de la Ofimática, como Windows, diremos que emplea un enlazado dinámico con un formato de archivo especial denominado DLL (*Dynamic Link Library*) Biblioteca de Enlace Dinámico. Su extensión es .dll, .drv (*drivers*) o .fon (tipos de letras).

Los archivos .dll pueden contener: Procedimientos o Datos de Librería. La configuración más usual de una .dll consiste en una librería que contiene varios procedimientos que pueden ser cargados en memoria a la vez. Varios programas ejecutándose en memoria pueden requerir de esos procedimientos indistintamente.

Cada vez que se invoca a una DLL (un proceso se liga), se realiza un proceso de adjudicación de memoria y cuando se deja de utilizar (un proceso se desliga) se libera memoria.

Una DLL no puede ejecutarse sola, a diferencia de un programa binario ejecutable. No puede iniciarse ni ejecutarse sola ya que no dispone de programa principal.

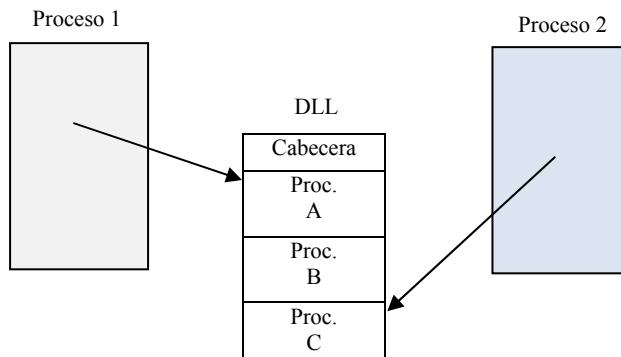


FIGURA 9. Llamadas a DLLs

Los programas se ligan a las dll de las dos maneras siguientes:

- Enlazado Implícito: Un archivo especial denominado Biblioteca de Importación, permite que el programa de usuario, pueda acceder a la DLL. El programa de usuario se enlaza estéticamente con la biblioteca de importación.

Un programa de usuario cargado en memoria que sigue el procedimiento implícito, realiza una inspección de las DLL que son necesarias y si están en memoria; las que no estén son cargadas inmediatamente. Los procedimientos que están en las DLL, se preparan para ser direccionados en el espacio de memoria virtual. Ya pueden ser llamadas como si hubieran sido ligadas estáticamente.

- Enlazado Explícito: No requiere de bibliotecas de importación, no requiriéndose que se carguen las DLL al mismo tiempo que el programa de usuario. El programa de usuario invoca a la DLL para ligarse dinámicamente y posteriormente realiza llamadas adicionales para obtener las direcciones de los procedimientos necesarios. Cuando ha finalizado el último proceso, el programa realiza una llamada para desligarse de la DLL, liberándose memoria.

Capítulo 4

MEMORIAS

El silencio es la primera piedra del templo de la sabiduría. (Pitágoras: 569 - 475 a.C)

RESUMEN. En computación se presenta la necesidad de almacenar la información necesaria tanto para realizar un proceso como para la información generada por dicho proceso. El almacenamiento se va a realizar sobre unos dispositivos denominados memorias. Las memorias van a presentar diversos formatos, tamaños y tecnologías. en este capítulo se realiza una visión exhaustiva de estos dispositivos.

4.1. INTRODUCCIÓN

Lo primero que debemos definir es el concepto de “memoria”: “dispositivo capaz de almacenar información”. No se precisa el tipo de dispositivo (soporte), si es mecánico, de estado sólido, magnético u óptico, etc. Tampoco importa el tipo de información, qué es, cómo está codificada, etc...

Desde el punto de vista de un Computador, vamos a distinguir tres apartados

1. La memoria va a contener el Programa (conjunto de instrucciones) y los Datos de trabajo
2. Se trata de un elemento sencillo pero con una gran diversidad de tipos, tecnologías, prestaciones, etc.
3. Se da una jerarquía entre los “diversos” dispositivos que componen la memoria, localizándose unos en el interior del computador y otros en su exterior.

Por otro lado, un dispositivo de memoria, debe presentar una determinada estructura para poder reconocerlo como tal; deberá disponer de los siguientes elementos:

- Medio o Soporte: donde se almacenarán los distintos estados que codifiquen la información (óptico, magnético, eléctrico, etc.).
- Transductor: convierte una magnitud física en otra, por ejemplo, una presión a corriente (sensor) o un voltaje a movimiento (actuador). (Mecanismos de lectura óptica y su paso a eléctrico, etc...).
- Mecanismo de Direcccionamiento: que permita procedimientos de Lectura/Escritura de información en el lugar e instante deseado.

4.2. CARACTERÍSTICAS

4.2.1. Localización.

Dependiendo de donde se ubiquen físicamente las memorias, distinguiremos tres tipos.

- Memoria Interna del Procesador (CPU)¹: Memoria de alta velocidad, presenta un tamaño pequeño en comparación con las siguientes. Memoria de trabajo para almacenamiento temporal de Instrucciones y Datos. Podrán ser Registros o la Memoria Caché.
- Memoria Interna del Computador: Se trata de la Memoria Principal o Primaria, en ella residen los Programas que van a ser ejecutados o bien los Datos obtenidos. Es relativamente grande pero más pequeña y más rápida que la Memoria Externa.
- Memoria Externa al Computador: Memoria Secundaria o Auxiliar, es la más lenta pero es la que más información puede almacenar (programas y grandes ficheros); está limitada a la velocidad del Bus y a aspectos electro-mecánicos. Podemos citar las unidades Zip, CDs, DVDs, etc..

Esta característica de localización la podemos ver en la Figura 1:

¹CPU: Central Process Unit / UCP: Unidad Central de Procesos

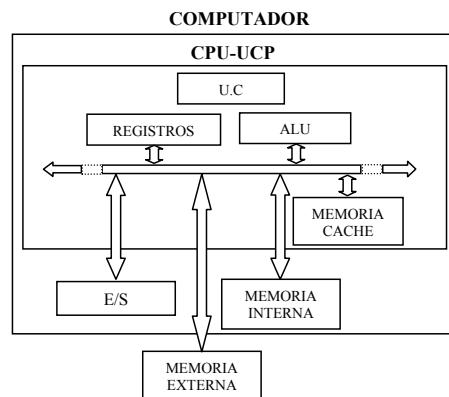


FIGURA 1. Arquitectura de bloques de un Procesador (CPU)

TABLA 1. Tamaños de memorias

1 bit	-		1 Mb	1024 Kb	2^{20} bits
1 nibble	4 bits		1 Gb	1024 Mb	2^{30} bits
1 byte - octeto	8 bits		1 Tb	1024 Gb	2^{40} bits
1 Kb	1024 bits	2^{10} bits			
1 word	16 bits	32 bits			
1 long word	32 bits	64 bits			

4.2.2. Capacidad.

Es la cantidad de información que puede almacenar el dispositivo, se cuantifica en múltiplos de “bit” (0 o 1), la práctica habitual es medirla en múltiplos de “byte” (octetos). La nomenclatura habitual es la siguiente:

- 1B significa “un byte”.
- 1Kb significa 1K bit (1024 bits, potencia más cercana de 2 a 1000).
- 1KB es una memoria de extensión 1K (posiciones) de 8-bit de tamaño cada posición.

Resumiendo, en la Tabla 1 vamos a ver las unidades y múltiplos más habituales.

4.2.3. Método de Acceso.

Es el modo de acceder a las celdas de memoria.

- Acceso Secuencial (SAM: *Sequential Acces Memory*): Para acceder a un Dato, hay que recorrer todos los datos anteriores. Es un método lento (memoria lenta) pero útil en grandes sistemas de almacenamiento, como p.ej. cintas.
- Acceso Aleatorio (RAM: *Random Acces Memory*): Se accede a un Dato de manera directa a través de su dirección. El tiempo de acceso es siempre el mismo.
- Acceso Asociativo (CAM: *Content Addressable Memory*): Se accede a un Dato por su contenido, se busca en toda la memoria simultáneamente. Una vez encontrado el Dato, se da la dirección donde se ubica.

4.2.4. Velocidad.

Para medir el rendimiento de una memoria, se recurre a tres parámetros:

- Tiempo de Acceso (T_A): Según el tipo de memoria, se darán dos tipos de T_A :
 - ▷ Si es un acceso aleatorio RAM: Tiempo que transcurre desde el instante en el que se presenta una dirección a la memoria hasta que el dato es memorizado o disponible para su lectura.
 - ▷ Si es un acceso CAM o SAM: Tiempo empleado en actuar el mecanismo de L/E en la posición deseada, es decir, tiempo empleado en localizar la dirección.
 - Tiempo de Ciclo de Memoria (T_C): Tiempo transcurrido desde la orden de operación de L/E hasta que se puede dar la siguiente orden de L/E. Por debajo de este tiempo la memoria no responde. Interesa que este tiempo sea lo menor posible.
- Gráficamente el T_A y el T_C se pueden representar, en el tiempo, según el siguiente gráfico. T_A debe ser menor que T_C .

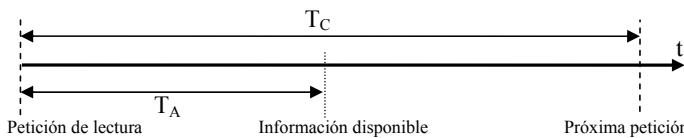


FIGURA 2. Tiempos de Acceso y Ciclo de Memoria

- Velocidad de Transferencia (V_T): Es la velocidad a la que se pueden transferir datos a, o desde, una unidad de memoria. Según el tipo de memoria, existen dos casos:

- ▷ Caso aleatorio (RAM) : $V_T = 1/T_C$
- ▷ Caso no aleatorio (CAM-SAM) : $T_N = T_A + (N/V_T)$

T_N : Tiempo medio de L/E de N bits. Tiempo de disponibilidad de datos

T_A : Tiempo de acceso al dato, es variable en acceso secuencial.
Fijo si CAM

N : Número de bits a transferir

V_T : Velocidad de transferencia (bits/s) una vez encontrado el dato

Los dos primeros parámetros definen la Latencia de una memoria.

4.2.5. Dispositivo Físico.

Los sistemas de memoria empleados en los computadores utilizan diferentes dispositivos físicos. En un principio se empleaba la memoria de ferrita que era de “lectura destructiva”, ya que tenía que volver a escribir lo que se leía. Esto las hacía lentas; si no hubiera tenido que realizarse la sobre escritura, las habría convertido en memorias muy rápidas. Actualmente los tipos más usados son:

- Memoria principal: dispositivos semiconductores
- Memoria secundaria: debido a la cantidad de información se recurre a:
 - ▷ Memorias magnéticas: Cintas, discos, etc.
 - ▷ Memorias ópticas: CDROM
 - ▷ Memorias magneto-ópticas

4.2.6. Aspectos Físicos.

Las principales características físicas a tener en cuenta para trabajar con determinados tipos de memoria son:

- **Alterabilidad:** Esta propiedad hace referencia a la posibilidad de alterar el contenido de una memoria, las hay de solo lectura o de lectura/escritura. Memorias ROM (*Read Only Memory*) y RWM (*Read Writable Memory*). La Figura 3 muestra una estructura de memoria ROM; los nodos marcados con “*” son los susceptibles de ser modificados. La Figura 4 presenta una memoria de Lectura/Escritura.

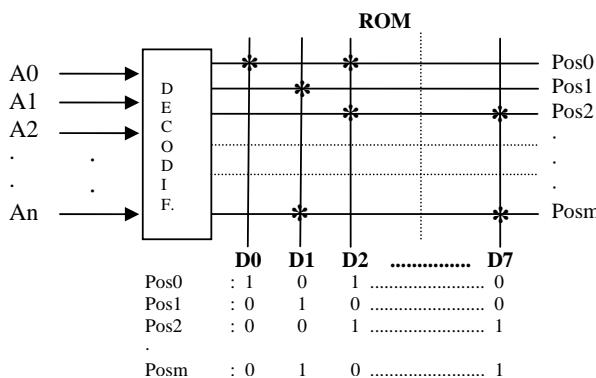


FIGURA 3. Esquema de una ROM

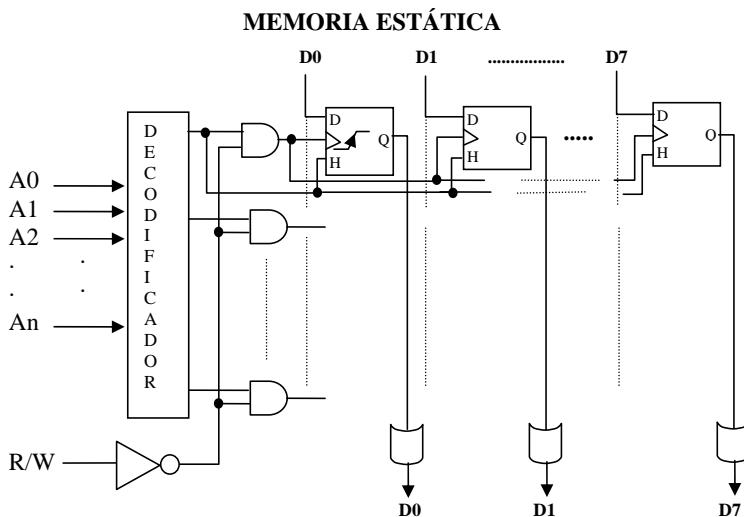


FIGURA 4. Esquema de una RAM

- **Permanencia de la Información:** Relacionado con la duración de la información almacenada en memoria
 - ▷ Lectura destructiva: Memorias de lectura destructiva DRO (*Destuctive Read Out*) y memorias de lectura no destructiva NDRO (*Non Destuctive Read Out*).
 - ▷ Volatilidad: Esta característica hace referencia a la posible destrucción de la información almacenada en un cierto dispositivo de memoria cuando se produce un corte en el suministro eléctrico. Memorias Volátiles y no Volátiles.
 - ▷ Almacenamiento Estático/Dinámico: Una memoria es estática SRAM (*Static Random Access Memory*) si la información que contiene no varía con el tiempo (La Figura 4 muestra una memoria estática). Una memoria es dinámica DRAM (*Dinamic Random Access Memory*) si la información almacenada se va perdiendo conforme transcurre el tiempo; para que no se pierda el contenido se debe “refrescar” la información (la Figura 5 muestra la celda de un bit de una memoria dinámica).

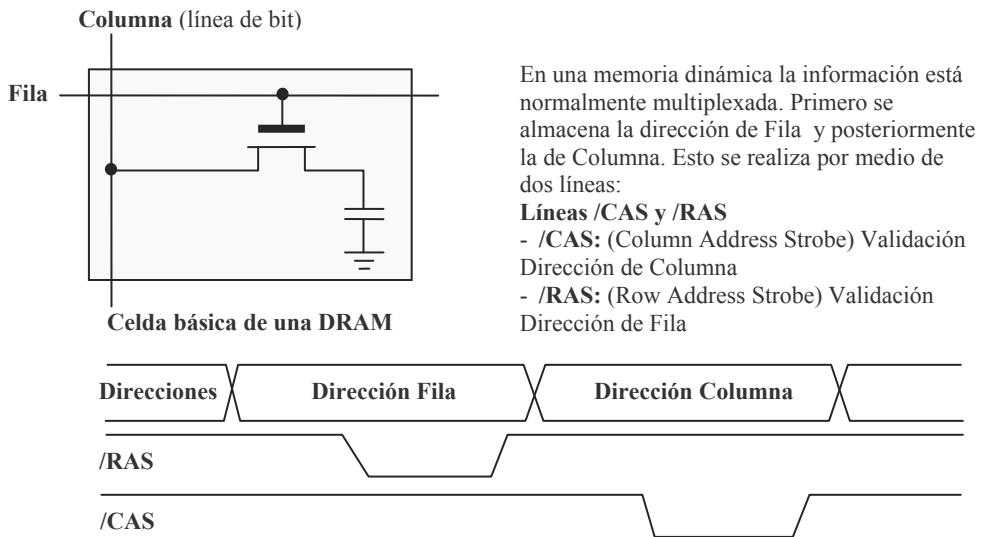


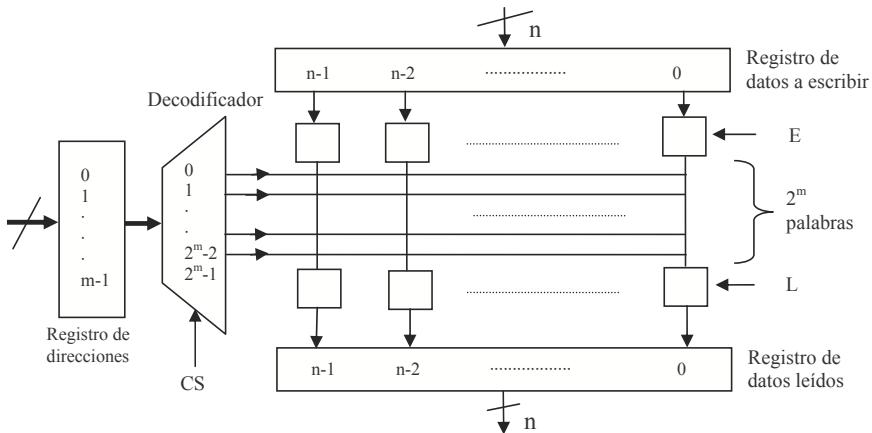
FIGURA 5. Célula y Direccionamiento en una DRAM

4.2.7. Organización.

Hace referencia a la disposición física de los bits para formar palabras. La organización depende del tipo de memoria que se trate. Para una memoria semiconductora distinguimos tres tipos de organización:

- Organización 2D.
- Organización 2 1/2D.
- Organización 3D.

1. **Organización 2D:** RAM de 2^m palabras de ancho “ n ” bits cada una, la matriz de celdas está formada por 2^m filas (n^o de posiciones de la memoria interna que van de la 0 a la 2^{m-1} en el bus de direcciones) y n columnas (n^o de bits en el registro de la memoria o ancho de palabra). Se trata de una organización lineal que se emplea en memorias de poca capacidad y gran rapidez de acceso (ver Fig.6).



Si tuviéramos, p.ej. 1K x 8 de memoria tendríamos 1024 posiciones que son 2^{10} , por lo tanto las direcciones van de la 0 .. 1023, o bien, de la 0000H .. 03FFH. Si se aumenta el ancho de palabra a mas de 8 bits entonces, la superficie de silicio de la memoria, se vería ampliada

FIGURA 6. Organización 2D

2. **Organización 2 1/2D:** Utiliza dos decodificadores con $m/2$ entradas y $2m/2$ salidas. El bus de direcciones de la organización 2D se divide en dos buses encaminados a dos decodificadores que van de $0 .. [(m/2) - 1]$ y de $[m/2] .. m - 1$ y donde coincidan las líneas de salidas de los decodificadores estará el bit que busco, p.ej. bus de direcciones de 16 pasa a dos de 8-bit que implican 28 (256) posiciones. Esta organización se emplea en memorias de un solo bit como unidad de transferencia pues solo coincide un bit entre los dos decodificadores y precisa menor tamaño para la memoria que la de la organización 2D por lo que requiere menor n° de puertas lógicas (ver Fig.7).

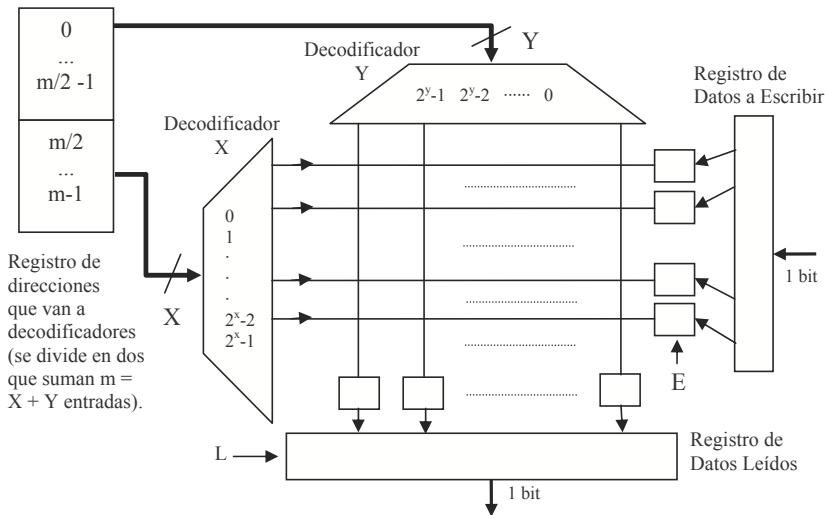


FIGURA 7. Organización 2 1/2 D

3. Organización 3D: Es similar a la organización 2 1/2D pero la palabra de n bits se almacena en n planos y dentro de cada plano se selecciona la posición “ x ” y la posición “ y ”, por lo que solo se habilitarán los bits que estén en el mismo plano; p. ej., si tenemos 16 bits tendremos 16 planos. Tiene un diseño más complicado y es una organización para memorias de acceso lento. Son chips de mayor densidad de silicio y, por lo tanto, más anchos para así poder ampliar el ancho de palabra más fácilmente (ver Fig.8).

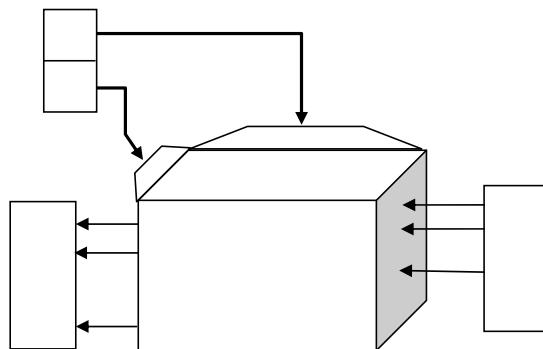


FIGURA 8. Organización 3D

4.3. JERARQUÍAS DE MEMORIA

La manera usual de almacenamiento de información, comenzando por poca información y llegando a una cantidad masiva de información, es una jerarquía planteada en la Figura 9:

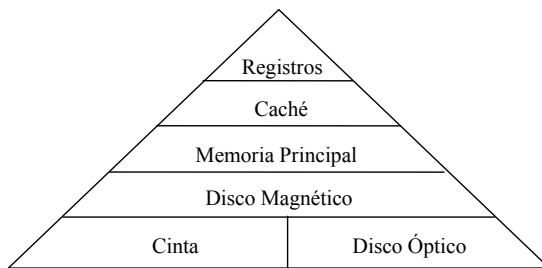


FIGURA 9. Jerarquía de Memoria de cinco Niveles

En esta jerarquía se establecen tres parámetros fundamentales:

- **Velocidad:** Tiempo de acceso a la memoria. Si bajamos en la jerarquía, este tiempo aumenta. Los registros de la CPU se acceden en unidades de ns , la caché en decenas de ns ., etc..
- **Capacidad:** Almacenamiento de información. Si bajamos en la jerarquía, este parámetro aumenta; pasamos de p. ej. 128 bytes en registros de CPU a Megabytes en caches, a decenas o miles de Megabytes en la principal, hasta Gigabytes y Terabytes en el resto de memorias. Los dispositivos que forman el 5º nivel son externos al computador
- **Coste:** El coste por bit desciende al bajar en la jerarquía

4.4. MEMORIA PRINCIPAL SEMICONDUCTORA

Veamos a continuación los diferentes tipos de memorias y sus correspondientes diseños.

4.4.1. Tipos de memorias.

Descripción según sus características de Lectura y/o Escritura.

1. Memorias de solo Lectura

- a) Escritura Destructiva (solo Lectura)

- 1) **ROM (Read Only Memory)**: Memorias grabadas durante el proceso de fabricación del circuito (silicio). Esta memoria puede formar parte de microcontrolador o ser independiente. En ella residen programas de uso muy frecuente (partes de sistema) o bien programas muy contrastados que no van a sufrir modificaciones en el tiempo. Barata en grandes series.
 - 2) **PROM (Programmable Read Only memory)**: Memoria grabada normalmente por el cliente (desarrollador), puede ser grabada por el fabricante. Presenta las mismas consideraciones que la anterior. Una única grabación. Más cara que la ROM.
- b) Escritura no Destructiva (Lectura y Escritura)*
- 1) **EPROM (Erasable Programmable Read Only Memory)**: Memoria grabada eléctricamente y borrable por luz ultravioleta. Presenta un nº no muy elevado de borrados. Tiempo de borrado en torno a los 15'-20'. En ella residen los programas. Muy extendido su uso.

2. Memorias de Lectura/Escritura

a) Lectura principalmente

- 1) **EEPROM (Electrically Erasable Programmable Read Only Memory)**: Se trata de una Memoria grabable eléctricamente y borrable eléctricamente. Presenta una gran ventaja y es que puede ser grabada por el propio procesador de placa, así como almacenar información del tipo consignas variables en el proceso. Es más lenta en Escritura que en Lectura. La grabación puede ser a nivel de byte o bloques de bytes. Más cara que la EPROM.
- 2) **FLASH** (Nombre debido a su velocidad de grabación eléctrica): De características similares a la EEPROM, presentan mayor capacidad de almacenamiento que las EEPROM; debido a esta característica substituyen, en ocasiones, a los discos duros. Éstas memorias han sufrido una evolución histórica; las primeras se denominaron FLASH NOR por similitud de sus celdas a las puertas NOR, siendo direccionables aleatoriamente. Unos años después de la aparición de las FLASH NOR, aparecieron en el mercado las de tipo NAND, estas presentan mayor densidad de almacenamiento pero se les elimina la circuitería de acceso aleatorio y la escritura/lectura es a nivel de bloques.

b) Lectura/Escritura (RAM)

1) **DRAM** (*Dinamic Random Access Memory*): Memorias que necesitan “refrescar” su información debido a que esta se almacena en condensadores (de estado sólido). (*Row Address Strobe-Column A S*). Es un proceso repetitivo que solamente se altera con un ciclo de Lectura/Escritura. Pierden la información al desaparecer la energía. De entre los diversos tipos destacamos dos:

- **SDRAM**: *Synchronous DRAM*.
 - ▷ Alta velocidad. - “Pipeline” interno completo.
 - ▷ Interface síncrono (sincronizado con el clock del μ P).
- **DDR**: *Double Data Rate DRAM*
 - ▷ Memoria estándar en comunicaciones.
 - ▷ Salida de datos con flanco de subida y de bajada del reloj. Transfiere dos datos por pulso.
 - ▷ Dirección y señal de control con flanco de subida.

2) **SRAM** (*Static Random Access Memory*): Memorias que almacenan la información en biestables. No necesitan refresco. Pierden la información al desaparecer la energía. Son más rápidas que las DRAM. Son más caras que las DRAM. Destacamos dos tipos:

- **QDR**: *Quad Data Rate SRAM*.
 - ▷ Read y Write separados trabajando como DDR
 - ▷ “4” datos de salida por pulso. - Aplicaciones de alto ancho de banda (memoria principal para Look Up Tables, Listas ligadas, controlador Buffer de Memoria).
- **ZBT**: *Zero Bus Turnaround SRAM*
 - ▷ Elimina tiempos muertos de Bus durante los ciclos de Read/Write y Write/Read (100 % de utilización).
 - ▷ Otros nombres: *No Bus Latenci* (NBL), *No Turnaround RAM* (NTRAM).

En la Tabla 2 se refleja un resumen de los diferentes tipos de memorias.

TABLA 2. Tipos de memoria

Tipo	Clase	Borrado	Escritura	Volatilidad
RAM	Lectura/Escritura	Eléctrico por Bytes	Eléctricamente	Volátil
ROM	Solo Lectura	No	Máscara	No Volátil
PROM				
EPROM	Lectura/Escritura	Luz Ultravioleta	Eléctricamente	No Volátil
FLASH	Fundamentalmente	Eléctricamente por Bloques		
EEPROM		Lectura		
		Eléctricamente por Bytes y Bloques		

4.4.2. Diseño.

En el diseño de la memoria, hay que tener en cuenta dos aspectos muy importantes:

- Dispositivo de memoria
- Mapa de memoria

Está organizado internamente como una matriz de celdas de memoria de $n \times m$, donde n es el nº de palabras que puede almacenar el dispositivo y m es el nº de bits por palabra.

Ejemplo: Memoria de $4K \times 8 \rightarrow 4K$ palabras con ancho de 8-bit por palabra. La memoria tiene un total de:

- $4K = 4 \times 1024 = 4096$ palabras. $n = 4096$ direcciones o posiciones de memoria.
- 8 bits = m bits de ancho de palabra.

Gráficamente el dispositivo de memoria podría representarse de la siguiente manera.

TABLA 3. Contenido y Dirección en una Memoria

0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	← Contenido de la palabra 0 de la memoria
1									
...	
4094									
4095									

Un dispositivo físico de memoria presenta las siguientes señales en sus patillas:

- n patillas para el Bus de Direcciones, direccionándose $2n$ palabras.
- m patillas para el Bus de datos, indicando esto el ancho de palabra de m bits.
- (Read/Write) indica si la operación es de Lectura o Escritura. Existen otras patillas de escritura WE (Write Enable) y lectura OE (Output Enable).
- /CS (Chip Select) o /CE (Chip Enable). Selecciona el chip de memoria al que se accede.
- Vcc. Positivo de la alimentación del chip
- Vss. Negativo (0V. Lógico) de la alimentación.

La pastilla presentará una configuración como la siguiente:

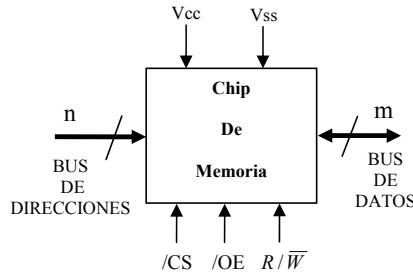


FIGURA 10. Esquema de señales de una memoria

Para el correcto funcionamiento de la memoria, que se rige por una tabla de verdad, es necesario incorporar una circuitería auxiliar que realice la decodificación, multiplexión, bufferización, etc. En la Tabla 4 y en la Figura 11 se muestra el comportamiento de la memoria:

TABLA 4. Señales de Control de una memoria

Entradas			Operación
/CE (R/W)	/WE	/OE	(Salida del Dispositivo)
H	X	X	Z
L	H	L	Lectura
L	L	X	Escritura
L	H	H	Power Down / Z

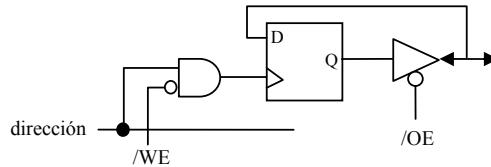


FIGURA 11. Célula básica de Memoria

Veamos un ejemplo de estructuración de una memoria:

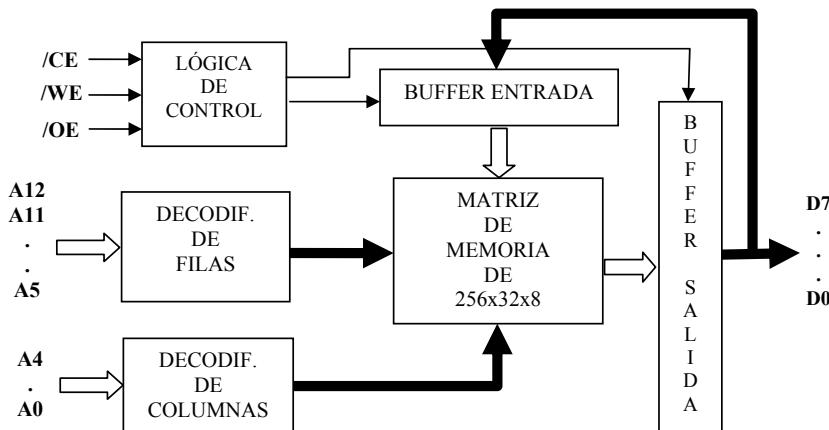


FIGURA 12. Estructuración en Bloques de memoria

4.4.2.1. *Mapa de Memoria.*

Es el espacio que puede direccionar un procesador según la memoria física instalada. Este mapa indica al procesador donde comienza y donde acaba la memoria, así como el tipo de memoria empleado. Normalmente no se ocupa toda la extensión de memoria, sino que se ubican los dispositivos en fragmentos de la extensión total de memoria. Para realizar el mapa de memoria, necesito conocer aspectos como los siguientes:

1. Conocer el nº de líneas de Direcciones del Procesador, si son m , tendré un espacio total de $2m$ localidades $[0 .. 2m - 1]$.
2. Conocer el nº de líneas del Bus de Datos, si son n podré escribir hasta $2n$ datos diferentes de n bits cada dato.
3. Conocer el nº de líneas necesario para realizar las funciones de L/E. Normalmente solo es necesaria una línea, pues la señal de control es, por lo que con “1” realizará la Lectura y con “0” la Escritura. Esto dependerá de cada Procesador, deberá estudiarse cada caso. Veamos unos casos concretos para clarificar estos conceptos:
 - a) Supongamos un computador que presenta una memoria de $64K \times 8$, esto nos dice que tiene $64K$ palabras de un ancho de 8-bit. El nº de líneas de dirección se calcula de la siguiente forma.
 - $64K = 64 \times 2^{10} = 2^6 \times 2^{10} = 2^{16} = 65,536$ posiciones de memoria accedidas por “16” líneas (bits) de dirección.
 - Estas $2n = 2^{16}$ posiciones de memoria, codificadas en hexadecimal, se expresarán de la siguiente forma: $0000H .. FFFFH$.

- Los Datos son $2m = 2^8 = 256$ datos diferentes accedidos por 8-bit. Expresado en hexadecimal, será 00H .. FFH.

La implementación física del mapa de memoria, se realiza con uno o varios chips de memoria. Los fabricantes de dispositivos de memoria disponen de una gran variedad de formatos. Se nos presentan dispositivos como los siguientes:

$nK \times 1$, $nK \times 4$, $nK \times 8$, $nK \times 16$, $nK \times 32$, $nM \times 1$, $nM \times 4$, $nM \times 8$, $nM \times 16$, $nM \times 32$, etc.; donde “ n ” es un múltiplo de 2, p.ej., si disponemos de una memoria de $4K \times 8$, accederemos a 4096 posiciones de memoria de 8-bit de ancho de información.

- a) Si quisiéramos diseñar una memoria de n bits y dispusiéramos de dispositivos de 1 bit de tamaño 64K, dispondremos en paralelo n dispositivos para su realización.

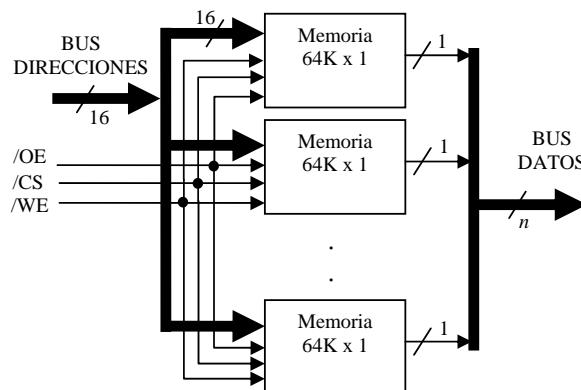


FIGURA 13. Ejemplo de Memoria de 64Kxn bits

- Si quisiéramos diseñar una memoria de 8 bits y dispusiéramos de dispositivos de 4 bit de tamaño 64K, dispondremos en paralelo 2 dispositivos para su realización.

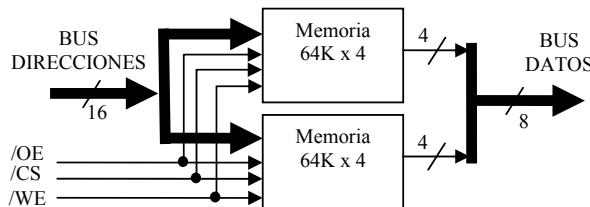


FIGURA 14. Ejemplo de Memoria de 64Kx8 bits

- Si quisiéramos diseñar una memoria de $64K \times 8$ bits y dispusiéramos de dispositivos de 8 bits de tamaño 32K, dispondremos 2

dispositivos para su realización. En este caso podremos utilizar A15 como señal de /CS.

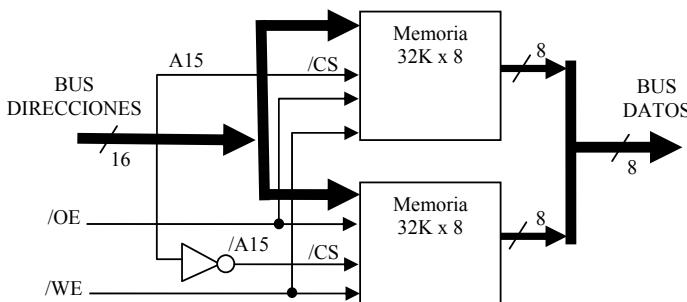


FIGURA 15. Ejemplo de Memoria de $64K(32K+32K)x8$ bits

4.4.2.2. *Ejemplos.*

Se presentan una serie de sencillos ejemplos a resolver por el alumno:

1. Queremos diseñar una memoria principal de 128 Kbytes
 - a) ¿Cuántos dispositivos de memoria de $32K \times 8$ necesitamos?
 - b) ¿Cuántos dispositivos de memoria de $64K \times 4$ necesitamos?
2. Dado un computador con datos de 16 bits y que permite direccionar 1 Megapalabra, teniendo instaladas 128 Kpalabras con chips de $64K \times 1$.
 - a) Obtener el nº de bits del Bus de Direcciones
 - b) Averiguar el nº de bits necesarios para direccionar el chip de memoria utilizado
 - c) Calcular el nº de chips necesarios
 - d) Obtener el nº de bits del Bus de direcciones que permitan seleccionar los chips utilizados

4.5. MEMORIA CACHÉ

4.5.1. Principio de Localidad.

Antes de comenzar con la explicación de lo que es una memoria caché (MC), es necesario introducir el concepto de “Localidad” para comprender porqué existe la memoria caché. El principio de Localidad establece que “las direcciones que genera un programa durante su ejecución, no son uniformes, sino que tienden a estar concentradas en pequeñas regiones del espacio de direcciones”.

Los programas tienden a reutilizar los datos e instrucciones que han utilizado recientemente. En la mayoría de los programas, un pequeño porcentaje del código, es el responsable de un gran tiempo de ejecución. Es corriente que el 10 % del programa ocupe el 90 % del tiempo de ejecución. Este comportamiento es debido al flujo secuencial y a las estructuras de control de flujo (bucles), así como a la agrupación en bloques, tanto de las instrucciones como de los datos dentro de los programas.

En la localidad de las referencias que genera un proceso, coexisten tres componentes, que pueden ser excluyentes:

- Tiempo**
- Espacio**
- Orden**

1. **Localidad Temporal:** Tendencia del proceso a hacer referencia a elementos a los que se ha accedido recientemente. Un elemento podrá ser referenciado pronto.
2. **Localidad Espacial:** Tendencia del proceso a efectuar referencias en la vecindad de la última referencia que ha realizado. Referenciando a un elemento es probable que se refiera a elementos próximos a él.
3. **Localidad Secuencial:** Tendencia del proceso a hacer referencia al siguiente elemento en secuencia al último referenciado.

4.5.2. Concepto de Memoria Caché.

Recurriendo al Principio de Localidad podemos pensar que si disponemos de una memoria muy rápida, donde podemos almacenar bloques secuenciales de código y datos, conseguiremos aumentar el rendimiento del computador. Esta memoria va a ser la caché² y va a estar situada entre el procesador y la memoria principal (MP). Se dispone en el mismo dispositivo del procesador por lo que va a tener la misma tecnología o lo que es lo mismo “velocidad”.

Debido al alto coste que supone el disponer en el silicio este tipo de almacenamiento, su tamaño no es grande pero se ha llegado a compromisos satisfactorios.

²Fue Wilkes en 1965 el que introdujo el concepto de memoria cache y poco tiempo después, en 1968 el modelo 360/85, de IBM, la incluía en su arquitectura. El primer microprocesador con la memoria cache en el interior del chip fue el Motorola 68020. Posteriormente aparecieron computadores con dos niveles de cache uno interno y otro externo como por ejemplo el SUN 3/250 en 1986 y caches separadas para datos e instrucciones (4D/240 de Silicon en 1988).

El funcionamiento de la memoria caché se basa en la transferencia de bloques de la memoria principal y la memoria caché, así como la transferencia de palabras entre la memoria caché y la CPU.

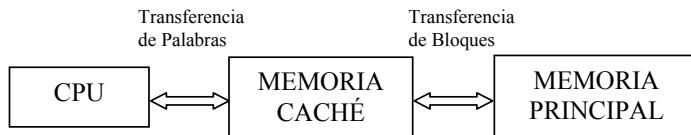
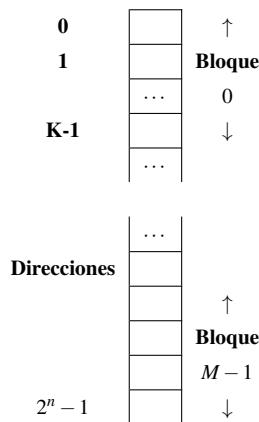


FIGURA 16. Esquema de transferencia de Datos entre CPU-MC-MP

La memoria Principal de 2^n palabras, siendo n el nº de bits del bus de direcciones, está organizada en M bloques de longitud fija (K palabras/bloque), capacidad de una línea de caché, donde $M = 2^n/K$ representa el nº de bloques totales. En la Tabla 5 se muestra la configuración de la memoria:

TABLA 5. Estructuración en Bloques de la MP



Una Línea de caché contiene un Bloque de memoria, o lo que es lo mismo, K palabras consecutivas.

La memoria caché está dividida en C líneas o particiones de K palabras siendo $C << M$. En la Tabla 6 se muestra la configuración de la memoria.

En cada línea de caché cabe un bloque de la memoria principal. La “Etiqueta” o “Tag” representa la dirección del bloque en la MP.

TABLA 6. Líneas en la MC

	Bit	Etiquetas	Bloque
Validez			
0			
1			
Líneas			
		
		
$C - 1$			

$\leftarrow K \text{ palabras} \rightarrow$

- **Tasa de acierto:** Cuando la CPU necesita una palabra de memoria y la encuentra en la memoria caché, se dice que se produce un acierto (“hit”); si la palabra no está en la memoria caché se contabiliza un fallo (“miss”). La relación entre el nº de aciertos y el nº total de referencias a memoria (aciertos + fallos) es la tasa de acierto. En sistemas bien diseñados se suelen conseguir tasas de aciertos de 0,9. La tasa de acierto se calcula mediante la siguiente expresión:

$$\text{tasa_de_aciertos} = \frac{\text{nº de aciertos}}{\text{nº de aciertos} + \text{nº de fallos}}$$

4.5.2.1. Organigrama de Funcionamiento.

Veamos a continuación como se expresa, mediante el organigrama de la Figura 17, el mecanismo de búsqueda y sustitución en una memoria caché.

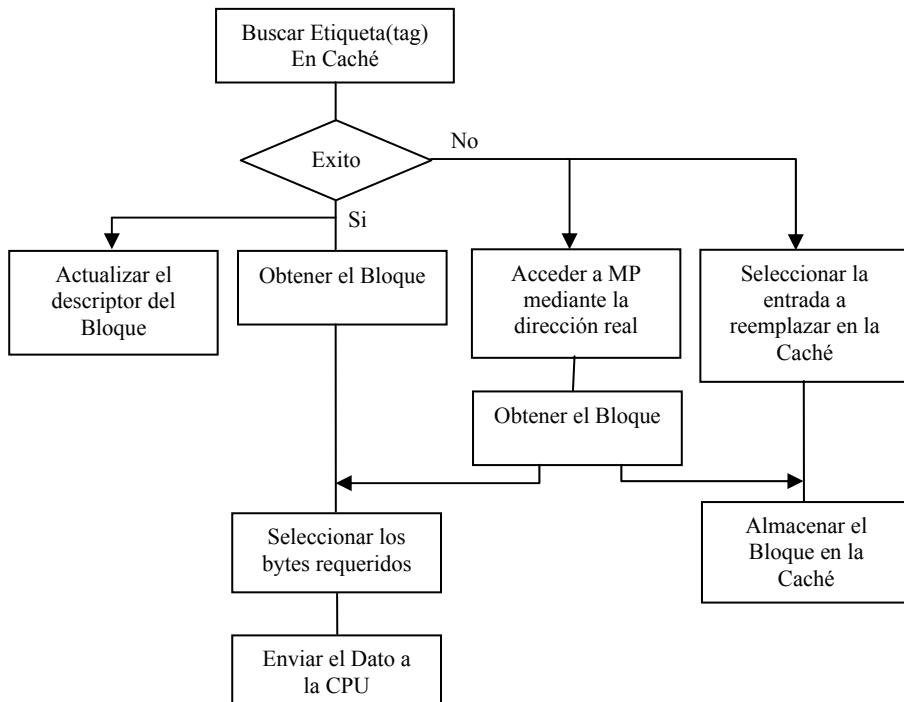


FIGURA 17. Organigrama de acceso a caché

4.5.3. Parámetros de Diseño.

La memoria caché surge como consecuencia del aumento de la velocidad de la CPU, es decir, los primeros procesadores presentaban una velocidad de acceso a memoria igual o menor a la de la memoria, por lo tanto, la CPU disponía de las Instrucciones/Datos sin introducir ciclos de espera. Al aumentar la velocidad de los procesadores debido a las nuevas tecnologías de integración era necesario el introducir ciclos de espera ya que las memorias no siguieron creciendo en velocidad al mismo ritmo que los procesadores con un resultado de bajo rendimiento, o bien, el costo de las memorias más rápidas podía hacer inasumible las nuevas arquitecturas; por lo tanto, se hizo necesario el plantear una nueva estrategia en el acceso a memoria, introduciendo el concepto de memoria caché. A la hora de diseñar la memoria caché debemos de tener en cuenta los siguientes parámetros:

4.5.3.1. *Tamaño de la Memoria Caché.*

Este parámetro plantea un cierto compromiso:

- Debería ser lo suficientemente pequeña como para que el coste medio por bit de información almacenada en la memoria interna del computador estuviese próximo al de la memoria principal.
- Tendría que ser lo suficientemente grande como para que el tiempo de acceso medio total (MP y MC) fuese lo más próximo posible al de la memoria caché.

De acuerdo con estos estudios empíricos, se sugiere que el tamaño de una caché esté situado entre 1KB y 1MB.

4.5.3.2. *Tamaño del Bloque.*

Partimos de un tamaño de caché fijo. Cuando se va aumentando el tamaño del bloque, a partir de valores muy pequeños, la tasa de aciertos inicialmente aumentará; pero a partir de un cierto tamaño del bloque la tasa de acierto comenzará a disminuir.

- Cuanto mayor sea el tamaño de los bloques, menos bloques se instalarán en la memoria y más veces se ejecutará el algoritmo de substitución de bloques (mas fallos). Se favorece la localidad “espacial” pero va en detrimento de la “temporal” al disminuir el nº de bloques.
- Cuando crece el tamaño de un bloque, cada nueva palabra añadida a ese bloque estará a mayor distancia de la palabra requerida por la CPU, y por tanto es menos probable que se necesite a corto plazo.

La línea de caché que es la que contiene el bloque, consta generalmente de entre 4 y 64 bytes (excepcionalmente 128) consecutivos (tamaño del bloque).

4.5.3.3. *Número de Cachés.*

Este parámetro plantea dos tipos de caché distintos .

- Caché interna. Nivel 1 (primario): Físicamente está ubicada en el mismo chip que el procesador (CPU) y aumenta la velocidad de procesamiento. Los accesos a esta caché se efectúan más rápido. La capacidad de esta caché es bastante pequeña. Oscila entre 16KB y 64KB.
- Caché externa. Nivel 2 (secundario): Físicamente está ubicada fuera del chip del procesador (CPU) por lo que será más lenta que la

caché de Nivel 1 pero seguirá siendo más rápida que la memoria principal. Al estar fuera (situada entre la memoria principal y el procesador) la capacidad podrá ser mayor que la caché Nivel 1. Su uso se está extendiendo de forma que cada dispositivo se fabrica con su propia caché pero son dos dispositivos diferenciados. Oscila entre 512KB y 1MB. Ésta caché se dispone físicamente junto a la CPU para que los caminos eléctricos sean lo mas cortos posible; de esta forma se minimizan los retardos y los problemas de Compatibilidad Electromagnética (CEM).

Generalmente las cachés son inclusivas, es decir:

$$CN_1 \subset CN_2 \subset CN_3 \subset MP$$

4.5.3.4. Contenido de la Caché.

Lo ideal es que la memoria caché contenga instrucciones y datos, empleándose actualmente en diseños segmentados.

- Caché unificada: Datos e Instrucciones
- Caché dividida: Caché de Datos y caché de Instrucciones.

Una caché que contiene tanto datos como Instrucciones presenta las siguientes ventajas:

- Presenta una tasa de aciertos mayor ya que automáticamente equilibra la carga (instrucciones y datos), es decir, si un patrón de ejecución implica muchas más captaciones de Instrucciones que de Datos, la caché tenderá a llenarse con instrucciones y viceversa.
- Solo se necesita implementar y diseñar una caché, por lo que el coste será más reducido.

Hoy en día, la tendencia es hacia dos cachés. Se trabaja en ejecución paralela de instrucciones y los diseños “*pipelining*” en los que el uso de dos cachés da mejores prestaciones. (Ref. Harvard). Duplica el ancho de banda del sistema de memoria. Ventaja: elimina la competición entre el procesador de instrucciones y la unidad de ejecución.

4.5.3.5. Estrategia de Escritura de Datos – Política de Actualización.

A modo de introducción, comentaremos que las escrituras en memoria son menos frecuentes que las lecturas de memoria. Téngase en cuenta que hay un flujo de lectura de instrucciones muy importante, así como, el flujo de datos es mayoritariamente de lectura. La estadística en máquinas RISC

es la siguiente: el 7% de accesos a memoria son de escritura y las escrituras en la caché es del orden del 20% de los accesos totales [13]. Por lo tanto, aunque las escrituras a MP sean menos eficientes que las lecturas, su incidencia en el rendimiento global, no es importante.

Antes de que pueda ser reemplazado un bloque que está en la caché, es necesario saber si se ha modificado o no, es decir si el bloque es un bloque limpio o modificado (sucio). Los datos se modifican en la caché al ejecutar el programa. Si el bloque no fue modificado no hay problema pero si no, existe un problema en la escritura; esto nos lleva a dos tipos de escritura:

- **Escritura Inmediata o Directa (Write Through):** Todas las operaciones de escritura se realizan tanto en la MC como en la MP, lo que asegura que los contenidos sean siempre válidos; tiene la ventaja de no realizar limpieza al reemplazar la línea de la caché. El principal inconveniente es que genera mucho tráfico con la memoria principal, pudiendo causar un colapso (cuello de botella).

Este tipo de escritura también se conoce como “escritura a través”.

- **Post-Escritura (Write Back):** Las escrituras se realizan solo en la memoria caché. Asociada a cada línea de la caché existe un bit de “modificación”. Cuando se escribe en la caché, el bit de modificación, se pone a 1. En el caso de reemplazar una línea, se mira el bit de modificación y si está a 1, se escribirá la línea en la memoria principal, mientras que si está a 0 no. Se escribe el bloque en la MP cuando este es expulsado de la MC.

Este tipo de escritura se conoce también como “escritura diferida o reescritura”.

Inconveniente: se obliga a que los módulos de E/S accedan a la memoria principal a través de la memoria caché. Esto complica la circuitería y genera un cuello de botella. Incrementa la tasa de fallos de caché.

Ventaja: se utiliza menos ancho de banda de memoria principal haciendo idóneo su uso en multiprocesadores. Consistencia de la información.

Por último, este parámetro conlleva un problema de coherencia de datos si no se realiza la reescritura a la memoria principal³.

Las arquitecturas más comunes de conexión entre la MP, la MC y las E/S son las que se muestran en la Figura 18.

³NOTA: La idea genérica es mantener el máximo tiempo posible las líneas más utilizadas.

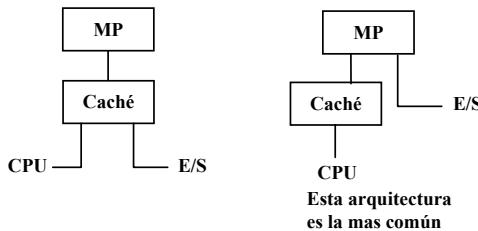


FIGURA 18. Arquitecturas de interconexión entre MP-MC-E/S

4.5.3.6. Función de Correspondencia – Organización de la Caché.

Debido a que existen menos líneas que bloques, se necesita un algoritmo que haga corresponder bloques de memoria principal a líneas de memoria caché. Existen tres tipos de correspondencia por lo que existirán tres tipos de función de correspondencia distinta:

- **Correspondencia Directa:** Ejemplo para 8 líneas. El bloque 12 de memoria principal solo podrá almacenarse en la línea 4 (= 12 módulo 8). Es la Dirección del Bloque MODULO N° de bloques de la Caché.
- **Correspondencia Totalmente Asociativa:** Se puede almacenar en cualquier línea.
- **Correspondencia Asociativa por Conjuntos:** Si se escoge asociatividad 2 (2 bloques por conjunto). Se puede almacenar en cualquier línea del conjunto 0 (= 12 módulo (8/2)).

Este ejemplo de cada tipo de correspondencia se ve mejor en la Tabla 7:

TABLA 7. Correspondencias en memoria caché

Número de línea	Directa	Asociativa	Asociativa por conjuntos	
0		■	■	Conjunto 0
1		■	■	
2		■		Conjunto 1
3		■		
4	■	■		Conjunto 2
5		■		
6		■		Conjunto 3
7		■		

A continuación vamos a describir las tres técnicas enumeradas. Veremos la estructura general en cada caso y un ejemplo concreto para cada caso.

En los tres casos, para los ejemplos, trabajaremos con un sistema que presentará las siguientes características:

- El tamaño de la memoria caché es de 4 KBy = 4096 Bytes.
- Los datos se transfieren entre memoria principal y la memoria caché en bloques de 16Bytes (las “K” palabras del gráfico). Esto nos indica que la caché está organizada en 256 líneas (4096/16) (la “C” del gráfico).
- La memoria principal consta de 64 KB, por lo que el bus de direcciones es de 16-bit ($64K = 2^{16}$). Esto nos indica que la memoria principal está constituida por 4096 bloques (la “M” del gráfico).

1. Correspondencia Directa

Consiste en hacer corresponder cada bloque de memoria principal a solo una línea de memoria caché. La función de correspondencia se expresa mediante la siguiente expresión:

$$i = j \bmod m,$$

donde:

- i = nº de línea de caché.
- j = nº de bloque de la memoria principal (dirección del bloque).
- m = nº de líneas de la memoria caché (nº de bloques de la caché).

Cada dirección de la memoria principal puede verse dividida en tres campos (vease la Tabla 8):

TABLA 8. Partición de la Dirección en campos en Correspondencia Directa

← s bits →	← w bits →	
Etiqueta	Línea	Palabra
↑	↖	

Identifica qué Bloque es de todos los que pueden ocupar la misma línea ($j \bmod m$)

Línea de la caché ocupada por el Bloque

Donde:

- w bits = identifica cada palabra dentro de un bloque.
- s bits = identifica el nº de bloque.

El uso de una parte de la dirección como número de línea proporciona una asignación única de cada bloque de memoria principal en la caché tal y como se muestra en la Figura 19. Observese que “ r ” selecciona la “Línea” de la Caché y de ella extrae la “Etiqueta” (Bloque que ocupa la línea de entre todos los posibles) y la compara con el campo “Etiqueta” de la dirección de memoria.

Ventaja: La técnica de correspondencia directa es simple y poco costosa de implementar (requiere poco hardware).

Inconveniente: Hay una posición concreta de caché para cada bloque dado. Puede existir un trasiego alternativo de bloques que ocupan el mismo lugar.

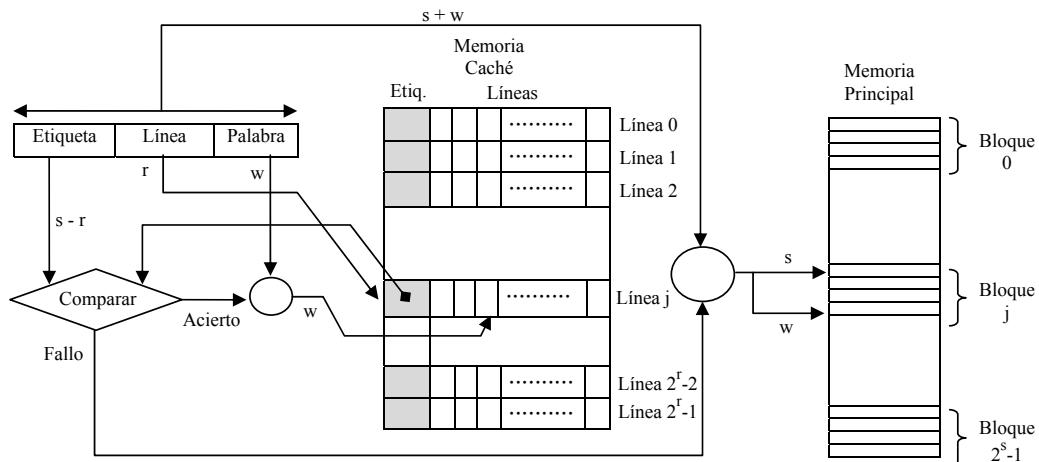


FIGURA 19. Caché de Correspondencia Directa

La Tabla 9 muestra un ejemplo de estructuración de la memoria en bloques.

TABLA 9. Ejemplo de estructuración de Bloques de Memoria

Etiqueta (Bloque)	Línea (Bloque Genérico)	Palabra (Palabra)
0	00	0
	01	
	
	FF	F
1	00	0
	
	FF	F
F	00	0
	
	FF	F

La Figura 20 muestra un ejemplo de correspondencia directa teniendo en cuenta la estructura de bloques propuesta en el ejemplo.

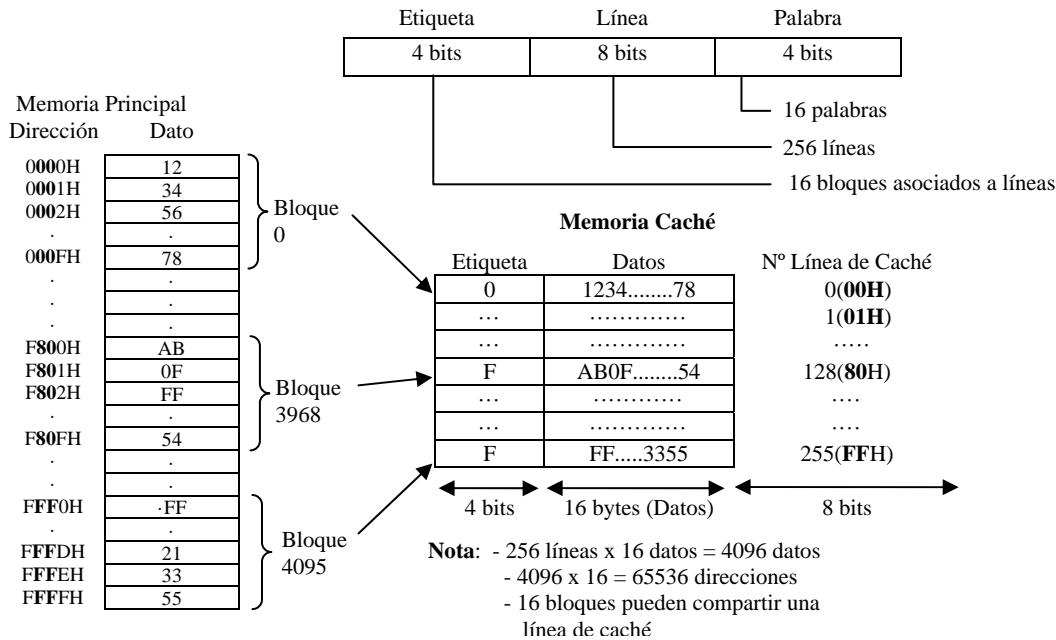
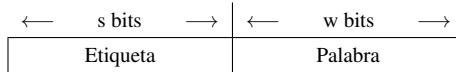


FIGURA 20. Ejemplo Caché de Correspondencia Directa

2. Correspondencia Asociativa

Permite que se cargue un bloque de memoria principal en cualquier línea de la memoria caché. La lógica de control de la memoria caché interpreta una dirección de memoria como una etiqueta y un campo de palabra.

TABLA 10. Partición de la Dirección en campos en Correspondencia Asociativa



Donde:

- Palabra: identifica cada palabra dentro de un bloque de MP.
- Etiqueta: identifica únicamente un bloque de MP.

Para determinar si un bloque está en la memoria caché, se debe examinar simultáneamente todas las etiquetas de las líneas de memoria caché, si la etiqueta está, entonces es un “acuerdo” y se indica que la palabra escogida es la que corresponde en la caché, sino es un fallo y la dirección es la que nos dice donde está en la memoria principal.

Ventaja: Mayor tasa de aciertos.

Inconveniente: Necesidad de una circuitería bastante compleja. No es tan rápida como la de acceso Directo.

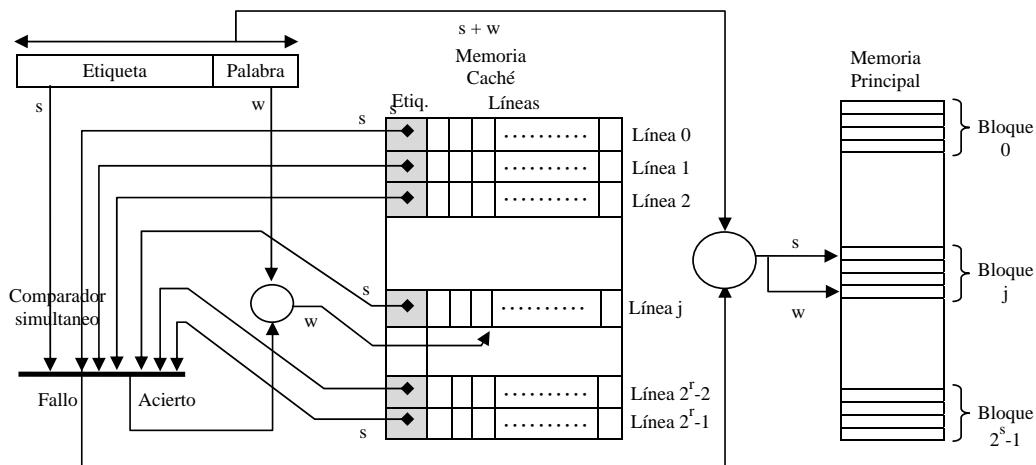


FIGURA 21. Caché de Correspondencia Asociativa

La Figura 22 muestra un ejemplo de correspondencia asociativa.

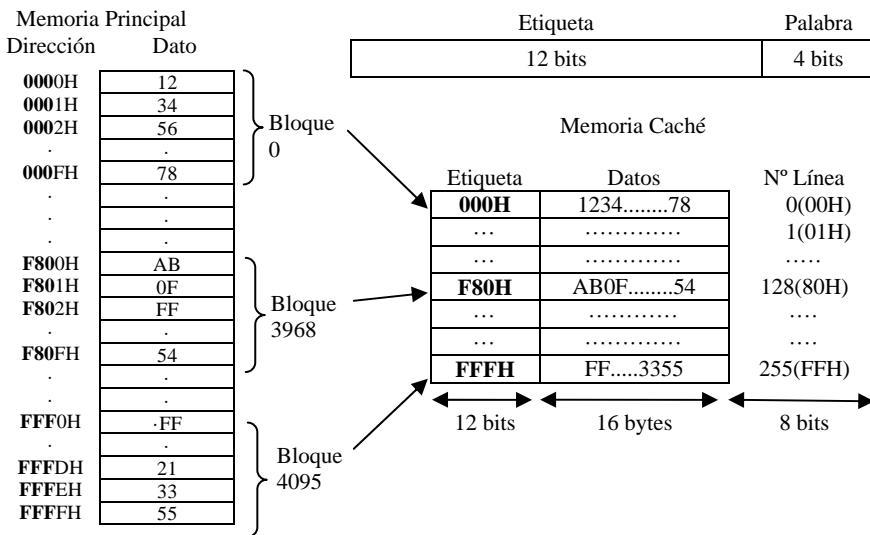


FIGURA 22. Ejemplo Caché de Correspondencia Asociativa

3. Correspondencia Asociativa por Conjuntos

Esta técnica es un compromiso que trata de aunar las ventajas de las dos técnicas vistas anteriormente. Memoria caché dividida en T conjuntos de L líneas. Las relaciones que se tienen son:

$$C = T \times L \text{ (nº de líneas de la MC) y } i = j \text{ módulo } T$$

Donde:

- i = número de conjunto de memoria caché (MC)
- j = número de bloque de memoria principal (MP)

El bloque B_j puede asociarse a cualquiera de las líneas del conjunto i . La lógica de control de la memoria caché interpreta una dirección de MP con tres campos.

TABLA 11. Partición de la Dirección en campos en Correspondencia Asociativa por Conjuntos

\leftarrow s-d bits \rightarrow	\leftarrow d bits \rightarrow	\leftarrow w bits \rightarrow
Etiqueta	Línea	Palabra
\leftarrow s bits \rightarrow	bits	

Donde:

- w bits de menor peso: palabra dentro de un bloque.
- s bits: identifica un bloque de MP.
- d bits: especifica uno de los conjuntos de la MC. Lleva directamente a la zona de MC donde está el conjunto.

- ▷ $s-d$ bits: etiqueta asociada a las líneas del conjunto “ d ” bits. Lleva al bloque del conjunto.

Nota: Experimentalmente se ha trabajado con asociatividad entre 2 y 16. En la práctica entre 2 y 4 vías. No más de 4.

Para saber si una dirección está o no en la memoria caché, lo primero se aplica correspondencia directa y luego correspondencia asociativa.

La Figura 23 muestra la estructura de la correspondencia asociativa por conjuntos.

Ventaja: Más rápida y menos costosa que la Asociativa. No presenta el problema de la correspondencia Directa.

Inconveniente: Necesidad de una circuitería bastante compleja.

Nota: Si $T = 1$ es Asociativa, un único conjunto. Si $L = 1$ es Directa, un único bloque.

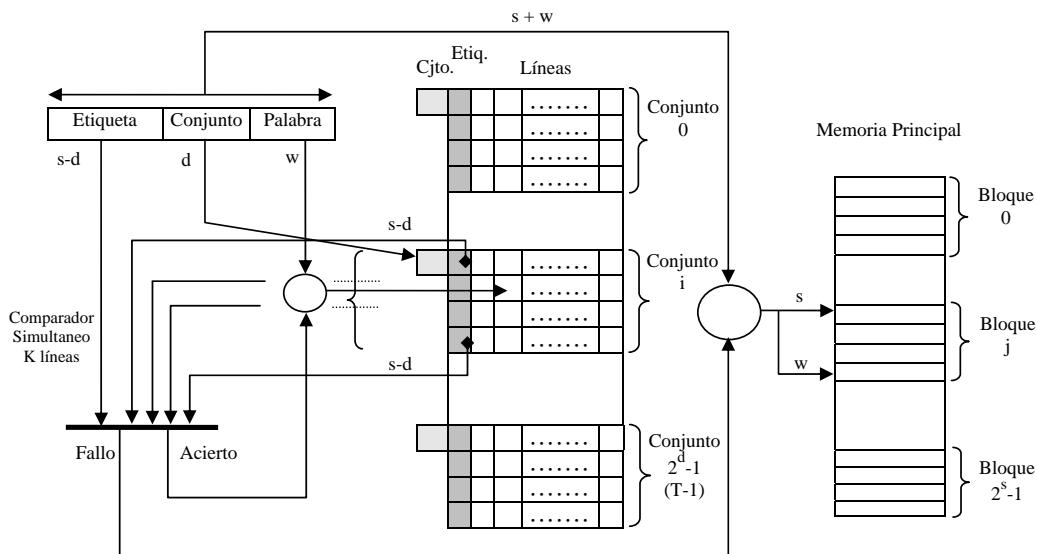


FIGURA 23. Caché de Correspondencia Asociativa por Conjuntos

La Tabla 12 muestra un ejemplo de estructuración de la memoria en bloques y la Figura 24 muestra un ejemplo de correspondencia asociativa por conjuntos teniendo en cuenta la estructura de bloques propuesta en el ejemplo.

TABLA 12. Ejemplo de estructuración de Bloques de Memoria

Etiqueta	Conjunto	Palabra
000000	000000	0

.....	111111	.

111111	000000	F
	
111111	111111	

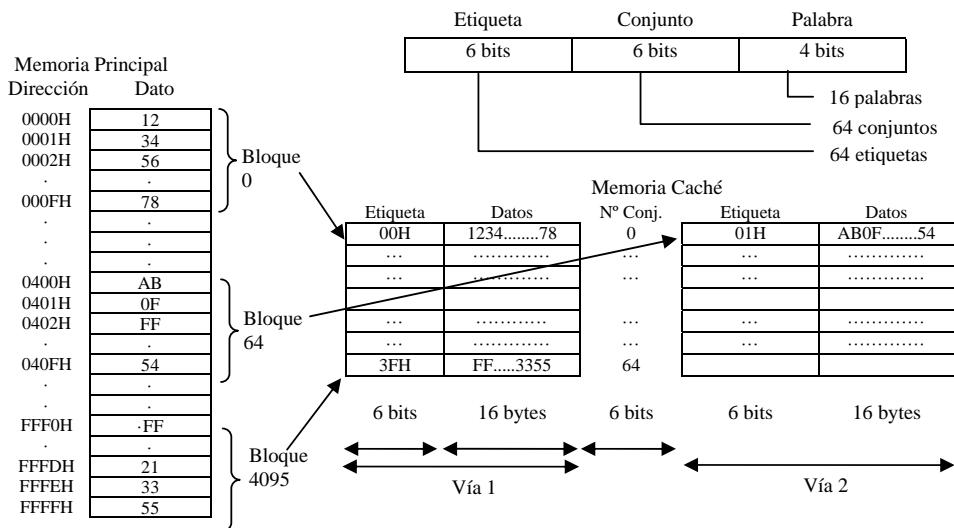


FIGURA 24. Ejemplo Caché de Correspondencia Asociativa por Conjuntos

4.5.3.7. Fuentes de Fallos en las Memorias Caché.

Una vez introducidas las funciones de correspondencia, fijémonos en cuales son las fuentes de fallos en la MC[5], es decir, en los fallos que afectan a la Tasa de aciertos. Los fallos se atribuyen a una de las tres situaciones siguientes:

- *Forzosos*: Cuando se accede por primera vez a un bloque y que, por lo tanto, no está en la MC, éste bloque debe ser traído a la MC. Este fallo también se denomina *fallo de arranque* o *fallo de primera referencia*.
 - *Capacidad*: Cuando la MC no puede contener todos los bloques necesarios durante la ejecución de programa, se presentarán fallos

de capacidad debido a los bloques que se han eliminado y posteriormente deben ser recuperados.

- **Conflict:** Cuando la estrategia de ubicación de bloques es de correspondencia directa o asociativa por conjuntos, los fallos de conflicto van a estar presentes ya que se puede descartar un bloque y posteriormente recuperarlo si a un conjunto le corresponde un nº excesivo de bloques. Este fallo también se denomina *fallo de colisión*.

Desde un punto de vista conceptual, la correspondencia asociativa (asociatividad completa) evita los fallos por conflicto, pero es una solución cara en hardware, ralentizando el tiempo de acceso; influyendo en el rendimiento global.

En cuanto a los fallos por capacidad, esta puede aumentarse introduciendo dispositivos mayores pero esto conlleva una penalización económica. Si la MC es muy inferior al tamaño requerido por el programa, serán necesarios muchos accesos, influyendo estos en el rendimiento; pueden presentarse situaciones en las cuales la velocidad de la máquina está próxima a la velocidad de acceso a la MP si no más lenta por sobrecarga de fallos.

Hacer bloques mayores reduce el número de fallos forzados, pero puede incrementar los fallos por conflicto. Téngase en cuenta que, por definición de fallo forzoso, si ya se ha cargado en MC un bloque (ahora mayor), la búsqueda de instrucciones/datos que antes (bloques más pequeños) era necesario ir a la MP, ahora pueden estar contenidos en un bloque ya residente en la MC.

4.5.3.8. *Algoritmos de Substitución.*

Cuando un nuevo bloque se transfiere a la memoria caché, debe substituir a uno de los existentes si la línea estuviera ocupada. En el caso de correspondencia directa la línea no tiene sentido en estos algoritmos. Entre los diferentes algoritmos que se han propuesto destacan los siguientes:

- **LRU (Least Recently Used):** Se substituye el bloque que recientemente haya sido menos utilizado. Cuando un bloque es cambiado se le califica como el más reciente. Requiere el disponer de la “edad” de los bloques en la MC. Este algoritmo es el que aporta la mejor tasa de aciertos.
- **FIFO (First In First Out):** Se substituye el bloque que entró el primero, independientemente de su uso.
- **LFU (Least Frequently Used):** Se substituye el bloque menos frecuentemente utilizado. Es necesario registrar el número de accesos

a los bloques. El bloque que tenga el contador más pequeño, será substituido.

- **Aleatorio:** Se seleccionan aleatoriamente los bloques susceptibles de ser modificados. Produce una tasa de aciertos algo menor a los otros métodos.

4.5.3.9. *Ejemplos de Memoria Caché.*

Fijándonos en los procesadores de Intel, estos han ido incorporando la memoria caché para aumentar su rendimiento. En la Tabla 13 vemos una serie de procesadores de Intel con las características de sus memorias caché:

TABLA 13. Tabla comparativa de cachés

Procesador	Nivel 1	Nivel 2
Inferior al 80386	NO	
80386	NO	16K, 32K, 64K
80486	8K Datos/Instrucciones	64K, 128K, 256K
Pentium	8K Datos y 8K Instrucciones	Pentium_Pro: 256K, 512K, 1M
Procesador	Descripción	
80486	Caché Interna	Tamaño de línea de 16 bytes Organización Asociación por Conjuntos de 4 vías
	Caché Externa	Tamaño de línea de 32, 64 o 128 bytes Org. Asoc. por Conjuntos de 2 vías
Pentium	Caché Interna	Tamaño de línea de 32 bytes Org. Asoc. por Conjuntos de 2 vías
	Caché Externa	Tamaño de línea de 32, 64 o 128 bytes Org. Asoc. por Conjuntos de 2 vías

4.6. MEMORIA ASOCIATIVA

4.6.1. Concepto.

Una memoria asociativa se caracteriza por el hecho de que el acceso a una posición de memoria, se realiza especificando su contenido o parte de él y no por su dirección. A las memorias asociativas también se les denomina direccionables por contenido: CAM (*Content Addressable Memory*).

4.6.2. Estructura de una CAM.

Una memoria asociativa consiste en un conjunto de registros y una matriz de celdas de memoria, con su lógica asociada, organizada en “ n ” palabras con “ m ” bits/palabra. El conjunto de registros está formado por un **registro argumento** (A) de “ m ” bits, un **registro máscara** (K) de “ m ” bits y un **registro marca** (M) de “ n ” bits.

Cada palabra de memoria se compara simultáneamente con el contenido del **registro argumento**, y se pone a “1” el bit de **registro de marca** asociado a aquellas palabras cuyo contenido coincide con el del registro argumento. Al final del proceso, aquellos bits del registro de marca que están a “1” indican la coincidencia de las correspondientes palabras de la memoria asociativa y del registro argumento. La comparación simultánea se realiza bit a bit. El bit A_j ($j=1,2, \dots, m$) del registro argumento se compara con todos los bits de la columna j si $K_j = 1$. Si existe coincidencia entre todos los bits $M_j = 1$. En caso contrario $M_j = 0$.

Resumen:

TABLA 14. Ejemplo de memoria CAM

0	San Sebastián	0	← Registro Argumento (A)
0	1	0	← Registro Máscara (K)
			↑ Registro de Marca (M)
			↓
Aitor	Vitoria	945 654321	0
Gilen	Bilbao	944 987654	0 ↓
Amaia	San Sebastián	943 765432	1 ←
Illan	Pamplona	948 554433	0
Naia	Vitoria	945 654432	0 ↓
Unai	San Sebastián	943 332211	1 ←

- **Registro Argumento:** Lo que quiero averiguar si está en el contenido de la memoria.
- **Registro de Máscara:** Podré quedarme con el que quiero averiguar que esté en la memoria. Determina filtrado del argumento a buscar.
- **Registro de Marca:** Tiene tantos bits como palabras tenga la memoria y pone un bit a 1 donde encuentre el argumento buscado y a 0 donde NO esté. Sirve para saber si en esa palabra (fila de la matriz) está el argumento que busco.

Por regla general, en la mayoría de las aplicaciones la memoria asociativa almacena una tabla que no tiene, para una máscara dada, dos filas iguales. Las memorias asociativas se utilizan sobre todo con memorias caché de

tal forma que la identificación de la etiqueta de cada línea se realice de forma simultánea. La TAG RAM es un claro ejemplo de memoria asociativa utilizada como parte de memoria caché en los sistemas con Pentium de Intel.

Los tiempos de acceso a una CAM están entre $4ns$ y $20ns$.

4.7. MEMORIA COMPARTIDA

4.7.1. Concepto.

La memoria compartida surge por la necesidad de que dos o más procesadores compartan información, para ello deberán acceder a la misma unidad de memoria, dejando o recogiendo la información adecuada.

El árbitro es el elemento encargado de permitir el acceso a la unidad de memoria, en un instante dado, a cada uno de los elementos que solicitan ese recurso, es decir que arbitra que elemento en cada momento puede compartir el uso de la memoria. El árbitro se diseña de forma que asigne un tiempo de servicio, en promedio, análogo a todas las unidades que solicitan el recurso. La Figura 25 muestra el esquema de bloques de una memoria compartida.

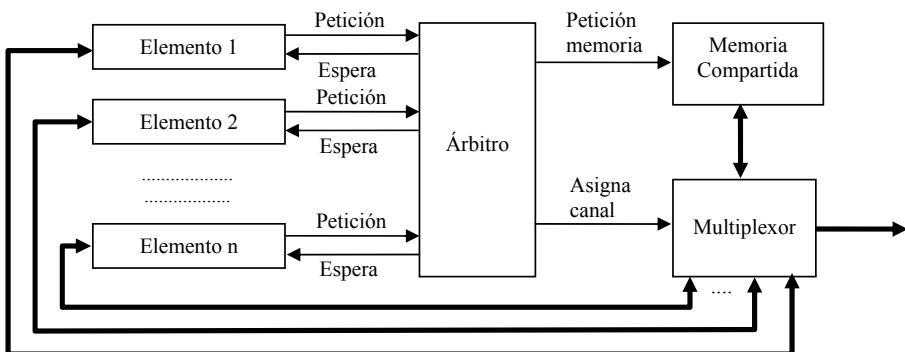


FIGURA 25. Esquema de Memoria Compartida

Existen diferentes estrategias:

- Asignación de la menor prioridad al elemento servido.
- Rotación de prioridades. En un estado cualquiera, el próximo estado se calcula rotando el orden de prioridades actual hasta que al elemento que se acaba de dar servicio tiene la menor prioridad.

4.7.2. Memorias de Doble Puerta.

Surgen como consecuencia de la aparición de la memoria compartida. Las memorias de doble puerto son memorias compartidas que permiten trabajar con dos dispositivos a la vez (dos buses de datos y direcciones). Se basan en duplicar los buses de direcciones, los decodificadores, la selección de chip, la L/E y bits de semáforo para establecer prioridad de las CPUs que acceden a la misma dirección de memoria, p. ej., con VRAM puede acceder

la tarjeta gráfica y el monitor a la vez. La configuración de las memorias de doble puerto se muestra en la Figura 26:

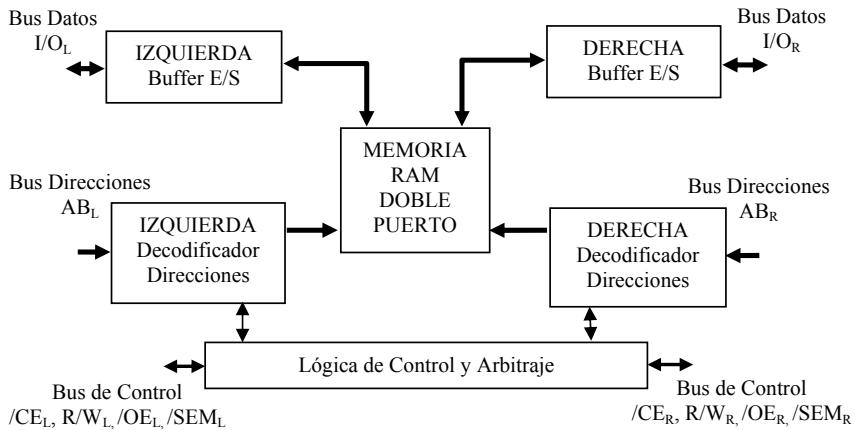


FIGURA 26. Esquema de Memoria de Doble Puerto

La memoria de doble puerto tiene prácticamente duplicados todos los componentes (puerto izquierdo L) y (puerto derecho R). La Tabla 15 recoge las señales que intervienen en la gestión de este tipo de memorias.

TABLA 15. Relación de señales en una Memoria de Doble Puerto

Puerto Izquierdo	Puerto Derecho	Descripción
I/O_L	I/O_R	Bus de Datos
AB_L	AB_R	Bus de direcciones
$/CE_L$	$/CE_R$	Selección Chip
R/W_L	R/W_R	Lectura/Escritura
$/OE_L$	$/OE_R$	Habilita Lectura
$/SEM_L$	$/SEM_R$	Habilita Semáforo

4.8. MEMORIA VIRTUAL

4.8.1. Concepto.

El término “Virtual” hace referencia a sistemas en los que el usuario dispone de un espacio de direcciones virtual más grande que el número de posiciones de memoria real con que cuenta. En los primeros diseños de computadores, el bus de direcciones de la CPU se conectaba directamente a la memoria principal y no tenía más que una única interpretación. Si un

programa era muy grande y no cabía en la memoria principal, era tarea del programador ajustar los distintos módulos de programa.

El programador dividía los programas en módulos o bloques, llamados recubrimientos o superposiciones (*overlays*).

Los recubrimientos se cargaban o descargaban de memoria durante la ejecución del programa. El programa del usuario se encargaba de llevar a memoria los bloques que fueran necesarios, reemplazando los no necesarios. Esta técnica creaba una gran carga de trabajo tanto al programador como al sistema, que debía saber que partes de su programa necesitaban estar residentes en memoria en un momento dado.

En 1961 se propuso un método para efectuar de manera automática el proceso de los recubrimientos sin que el programador se diera cuenta de ello (grupo de Manchester⁴). La idea consistía en separar los conceptos de espacio de direcciones y posiciones de memoria. Esto se conoce hoy en día como Memoria Virtual.

Supongamos un procesador (MC68000) en el que se ha implementado una memoria de 64 KB, si disponemos de un programa que exceda de la posición 65536, causará un fallo de ejecución. Sin embargo el procesador presenta un bus de direcciones de 24-bit, por lo tanto, puede direccionar $2^{24} = 16$ GB de memoria. Es decir, podemos direccionar más palabras que las que realmente se han implementado. Todo el conjunto de direcciones que pueden ser referenciadas por un programa se denomina espacio de direcciones virtual. Este espacio deberá estar soportado por una memoria secundaria.

Un sistema de memoria virtual gestiona automáticamente los dos niveles de la jerarquía de memoria principal-memoria secundaria.

4.8.2. Diseño.

4.8.2.1. Paginación.

La principal técnica utilizada en el uso de la memoria virtual consiste en dividir, tanto el espacio de direcciones virtual como el espacio de direcciones real en bloques de igual tamaño. Un bloque, en un sistema paginado, se denomina página y un fallo de memoria virtual se denomina falta de página. El tamaño de página es potencia de 2 y oscila entre 512 y 4096 palabras.

⁴Esta memoria la utilizó por primera vez el computador ATLAS, diseñado en la Universidad de Manchester, aunque los primeros computadores comerciales que la utilizaron fueron los IBM/360. En 1974 la familia IBM/370 introdujo el mecanismo Translation Lookaside Buffer (TLB) para la traducción de direcciones.

Es necesario implementar una función para relacionar el espacio de direcciones de los programas y las posiciones de memoria real (memoria implementada en el diseño).

4.8.2.2. *Implementación.*

La CPU produce una dirección virtual que debe ser traducida por una combinación del Hw y el Sw en una dirección real para acceder a memoria principal. Debemos entender que la “página” tiene una ubicación fija en la memoria secundaria pero una ubicación variable en la memoria principal (física), esta ubicación se denomina “marco”, es decir, una página en la MP ocupará un determinado “marco” asignado por el sistema gestor de memoria.

Cada dirección virtual consta de dos partes:

- Nº de página: Parte superior de la dirección (bits de mayor peso) y determina el nº de páginas direccionables.
- Desplazamiento: Parte inferior de la dirección (bits de menor peso). Determina el tamaño de la página. Localización de la palabra en la página .

Y cada dirección real consta de dos partes:

- Nº de Marco: Parte superior de la dirección real y determina el nº de marcos direccionables de memoria real.
- Desplazamiento: Ídem anterior. Localización de la palabra en la página.

TABLA 16. Partición de la Dirección en Campos

Dirección Virtual	
PAGINA	DESPLAZAMIENTO
Dirección Real	
MARCO	DESPLAZAMIENTO

En la traducción de una dirección virtual a una dirección real, se traduce el nº de página en un nº de marco y se concatena el desplazamiento. Veamos un ejemplo, descrito en la Tabla 17:

TABLA 17. Relación entre Memoria Principal y Secundaria

MEMORIA PRINCIPAL	MEMORIA SECUNDARIA
000 A	0000
001 B	0001
010 C	0010
011 D	0011
100 E	0100 A
101 F	0101 B
110 G	0110 C
111 H	0111 D
1000	1000
1001	1001
1010	1010
1011	1011
1100 E	1100 E
1101 F	1101 F
1110 G	1110 G
1111 H	1111 H

Correspondencia de Direcciones**en un sistema Paginado**

- a.** Si dispongo de 64K de memoria y escojo 2 bits de página (4 páginas), tendré que 16 bits-2 bits = 14 bits, por lo tanto 4 páginas de 16K ($2^{14} = 16K$)
- b.** Si dispongo de 64K de memoria y escojo 4 bits de página (16 páginas), tendré que 16 bits-4 bits = 12 bits, por lo tanto 16 páginas de 4K ($2^{12} = 4K$)

4.8.2.3. Gestión.

Si una página puede residir en cualquier marco, necesitamos un mecanismo para encontrarla. El mecanismo se denomina “Tabla de Páginas”, por cada página contiene un “descriptor” de página. Consta de los siguientes campos (Tabla 18):

TABLA 18. Descriptor de Página de Memoria Virtual

P	M						MARCO
Bit de Presencia							
Bit de Modificado							
Número de Marco							

La tabla de páginas, que debe residir en memoria, está indexada por nº de página. La búsqueda de un descriptor en la tabla de páginas se realiza mediante el campo nº de página de la dirección virtual, que actúa como índice de la tabla.

Cuando el bit de presencia “P” está a “1”, la página está cargada en MP, y la tabla de páginas contiene la dirección del marco asignado a esta página. Si $P = 0$, no está la página en MP. Este nº de marco, concatenado con el

desplazamiento de la dirección virtual, define la dirección real correspondiente.

Veamos esto en el esquema siguiente (Figura 27):

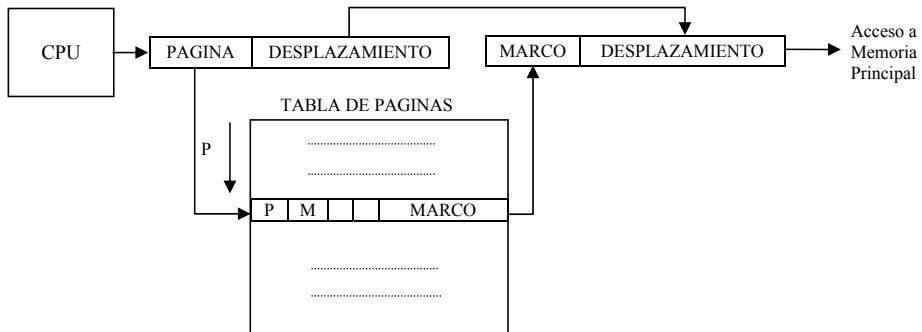


FIGURA 27. Tratamiento de Memoria Virtual

La no existencia de página en MP, provoca la “Excepción” de Falta de Página, transfiriéndose la página, que ha provocado la excepción, de la Memoria secundaria a la MP. Se expulsará una Página de la MP mediante la Política de Substitución.

Si la página que se decide expulsar tiene el bit $M = 1$, indica que esta página se ha modificado durante su estancia en la MP, por lo tanto habrá que transferirla a la MS copiando dicha página, antes de traer la nueva página.

Para el tratamiento de Páginas la CPU se complementa con un dispositivo denominado MMU (*Memory Management Unit*), este dispositivo puede estar dentro o fuera de la CPU. La incorporación de MMUs libera a la MP de la Tabla de Páginas, esta se ubica en los registros de la MMU.

Una solución intermedia consiste en que casi toda la Tabla de Páginas esté en MP y un conjunto relativamente pequeño de Descriptores esté en la MMU. Estos serán los más recientemente utilizados (referenciados). Estos registros se denominan Buffers de Traducciones Anticipadas (TLB: *Translation Lookahead Buffer*). “Principio de Localidad de las Referencias”.

4.9. OTRAS MEMORIAS

4.9.1. Introducción.

El rendimiento de un computador se ve afectado por la velocidad del bus, nos movemos entre 66 MHz, 100 MHz, 133 MHz, etc.. La velocidad

final de un procesador se obtiene a partir de un multiplicador interno. Para no reducir el rendimiento de un computador, las memorias deberían ser igual de rápidas que el procesador, pero las tecnologías disponibles no lo permiten; por lo tanto, cada vez que el procesador accede a la memoria tiene que esperar (tw: time wait). El uso de las memorias caché aumentan el rendimiento del sistema, pero debido a su alto coste y al espacio que ocupan físicamente no es viable añadir gran cantidad de memoria caché.

A medida que ha evolucionado la tecnología, las memorias cada vez se han fabricado más rápidas para reducir los tiempos de espera del procesador. Así pues, otra alternativa para aumentar el rendimiento del sistema es utilizar memorias principales más rápidas.

4.9.2. Memorias Entrelazadas.

Se denomina “entrelazado” al diseño de memoria en módulos. Se dan dos tipos de entrelazado:

- **Orden Inferior** (entrelazado simple): Direcciones consecutivas en módulos consecutivos. Selección del módulo con los bits de menor peso.
- **Orden superior** (entrelazado complejo): Cada módulo contiene direcciones consecutivas. Selección del módulo con los bits de mayor peso. En este tipo de memorias, los módulos se organizan de tres formas que estudiaremos más adelante:
 1. Organización con acceso S
 2. Organización con acceso C
 3. Organización con acceso C/S

4.9.2.1. Organización S.

Este tipo de configuración se denomina de acceso S porque a todos los módulos se accede “simultáneamente” tal y como se refleja en la Figura 28. Es una de las configuraciones más sencillas, presentando la siguiente funcionalidad:

- Permite acceder de manera simultánea a los módulos de memoria.
- Utiliza entrelazado de orden inferior y aplica los “ $n - m$ ” bits superiores de la dirección (mayor peso) a todos los módulos $M = 2^m$ módulos de memoria simultáneamente en un acceso.
- En un único acceso se obtienen las M palabras consecutivas de información procedentes de los M módulos de memoria.

- Los m bits inferiores de la dirección (menor peso) se utilizan para seleccionar la información de un módulo particular.

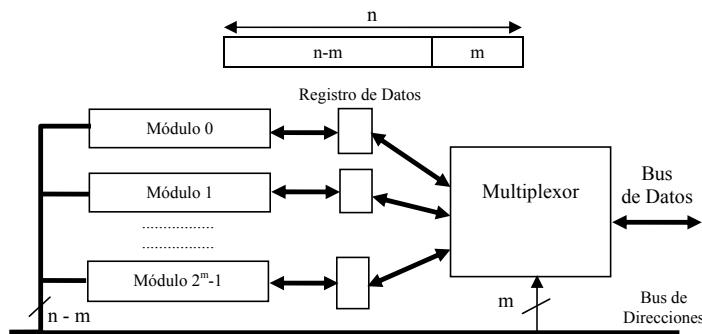


FIGURA 28. Organización S

Veamos con más atención su funcionamiento:

- La dirección de memoria se divide en dos por tiempos de acceso y tiempo de recuperación; solo nos quedamos con una palabra del módulo de memoria, lo que implica que por cada módulo de memoria tendremos un registro de datos.
 - ▷ Los módulos de memoria se organizan partiendo de las matrices de bits de las memorias; con los " $n-m$ " bits más significativos tengo la selección de cada uno de los módulos de memoria, de ellos leo a la vez así como de los registros, pero a la salida solo tengo una palabra por el multiplexor que se ha puesto antes del bus de datos.

El ejemplo de la Figura 29 clarifica su funcionamiento:

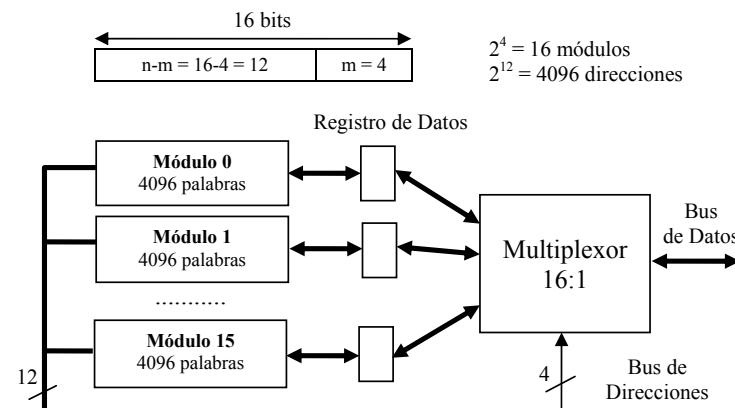


FIGURA 29. Ejemplo Organización S

Ventajas: Este tipo de organización es ideal para acceder a un vector de datos o para la búsqueda de instrucciones y operandos secuenciales. También es útil en la transferencia de bloques en sistemas con memoria caché.

Inconveniente: Para el acceso a posiciones de memorias no consecutivas, se utilizan todos los módulos.

4.9.2.2. Organización C.

Este tipo de configuración se denomina de acceso C porque a todos los módulos se accede de manera “concurrente” tal y como se refleja en la Figura 30. Presenta la siguiente organización:

- Permite acceder a posiciones distintas de los módulos concurrentemente
- Los “ m ” bits de orden inferior se utilizan para seleccionar el módulo y los “ $n - m$ ” bits restantes direccionar el elemento deseado dentro del módulo.
- El controlador de memoria se utiliza para mantener una petición que refierece un módulo ocupado. Existen configuraciones en las que el controlador dispone de una única cola FIFO o bien dispone de una cola FIFO por cada módulo de memoria.

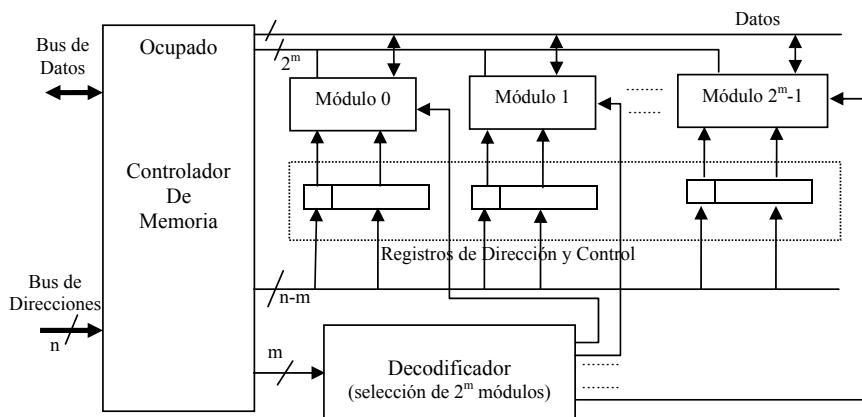


FIGURA 30. Organización C

Ventaja: Para acceder a una posición de memoria ya no se ocupan todos los módulos sino que ocupa únicamente el módulo que contiene esa dirección y el resto están libres. Para seleccionar el módulo correspondiente se emplea el decodificador.

4.9.2.3. *Organización C/S.*

Es una configuración mixta entre Concurrente y Simultánea.

- Combina los esquemas con acceso S y acceso C.
- Los módulos se organizan en forma de matriz bidimensional. Si el acceso S es un entrelazado de M vías y el acceso C un entrelazado de L vías, hasta L accesos diferentes a bloques de M palabras consecutivas por estar en progreso simultáneamente.

4.9.3. Memorias Dinámicas.

4.9.3.1. *Fast Page Mode RAM (FPM RAM).*

Es la primera memoria dinámica más popular. Este tipo de memoria incorpora un sistema de paginado porque considera probable que el próximo dato referenciado esté en la misma columna, ganando tiempo en caso afirmativo. Incorpora un contador de columna automático. Las transferencias de datos desde la memoria se realiza en paquetes de 4 datos denominados ráfagas (*burst*). El uso de ráfagas elimina los tiempos de establecimiento de dirección y de precarga de líneas de fila (también “página”) y columna posteriores al primer acceso.

Los tiempos de acceso están entre 120 ns. y 50 ns.

El rendimiento de un tipo de memoria se expresa con cuatro números separados por guiones que se corresponden con los ciclos de reloj que la memoria necesita para responder. La memoria ideal sería 1-1-1-1 que significa que en cada ciclo de reloj se transfiere un dato. Para una caché del tipo L2 (SRAM) es 2-1-1-2.

La memoria FPM RAM presenta un esquema, en el caso más favorable, 5-3-3-3, es decir, cuatro estados de espera en el primer dato y dos en los sucesivos, haciendo un total de 10 por ráfaga.

Los anchos de banda están entre 16 MHz. Y 66 MHz

4.9.3.2. *Extended Data Output RAM (EDO RAM).*

Es una modificación de las FPM RAM. Consigue una mejora entre el 10 % y el 15 % en velocidad con respecto a la FPM. Mientras se accede a los datos se prepara la siguiente dirección. La EDO RAM ha sido la memoria más popular debido a que su velocidad de acceso se incrementa y a que los fabricantes han tenido que hacer muy pocos cambios respecto de la FPM RAM.. La EDO RAM puede trabajar, en el caso más favorable, con un esquema 5-2-2-2, aumenta el ancho de banda de la memoria, lo que puede

suponer una mejora de hasta un 40 % en el rendimiento global. Aún así, el ancho de banda alcanzado con memorias de 60 ns, es de 40 MHz (este valor suele estar entre 33 MHz. Y 75 MHz.), todavía lejos de los 66 MHz del bus de un PC.

Algunos PCs permiten trabajar con memorias de 50 ns consiguiendo un ancho de banda de 50 MHz. Se consiguen tasas de transferencia de 264 MB/s.

La aparición, en los computadores, de un bus de 100 MHz deja a la memoria EDO sin sentido en su utilización ya que, como se ha visto, su ancho de banda llega a 50 MHz y esto reduce mucho el rendimiento del sistema.

4.9.3.3. Burst Extended Data Output RAM (BEDO RAM).

Esta memoria permite trabajar con un esquema 5-1-1-1 con 50 ns de tiempo de acceso consiguiendo alcanzar los 66 MHz. Estas memorias no han tenido mucho éxito debido a la aparición del bus de 100 MHz y las memorias SDRAM.. Las transferencias de datos son por ráfagas de 4 datos tanto en lectura como escritura. Mientras envía datos al procesador, está leyendo la siguiente dirección.

4.9.3.4. Syncronous DRAM (SDRAM).

Consigue una mejora del 25 % en velocidad con respecto a la EDO. La memoria DRAM síncrona o SDRAM a diferencia de las DRAM típicas, que son asíncronas, intercambia datos con el procesador de forma sincronizada con una señal de reloj externa, que opera a la velocidad del bus sin imponer estados de espera. El reloj maneja una Máquina de Estados Finita (FSM).. El núcleo DRAM es mucho más rápido que el de la memoria convencional, llegando a alcanzar velocidades de hasta cuatro veces más. Una arquitectura de doble banco permite entrelazados, de forma que mientras un banco prepara los datos otro los proporciona. Así mismo, la longitud de la ráfaga es programable, permitiendo un diseño flexible en función del sistema que deba soportarlo.

El esquema de trabajo de la SDRAM es de 5-1-1-1. Con memorias de 15 ns se pueden alcanzar los 100 MHz. La firma IDT dispone de memorias con esquema 2-1-1-1 a 50 MHz. Los tiempos de acceso también han sufrido mejoras estando entre 12 ns. y 6 ns.

Aplicaciones de las SDRAM se dan en los Buses PC66 a 66 MHz., PC100 a 100 MHz. y PC133 a 133 MHz. La tasa de transferencia llega a 528 MB/s.

Un ejemplo de SDRAM es la PC-100 RAM que cumple determinadas restricciones establecidas por Intel para el correcto funcionamiento con el bus de 100 MHz implementado en su chipset 440BX.

En el año 2000 las SDRAM supieron a las memorias de tecnologías anteriores. Suele ser utilizada en cachés grandes.

4.9.3.5. *Double Data Rate SDRAM (DDR RAM).*

Ya han sido enunciadas en el apartado de memorias RAM. Son un desarrollo posterior de las SDRAM. Presenta arquitecturas de 8 bytes y 16 bytes. Transfieren información en el flanco de subida y de bajada del reloj, duplicándose la cantidad de información transferida. De esta forma con 200 MHz. de ancho de banda, tendremos en el bus del PC una tasa de hasta 3.2 Gb/s. para arquitecturas de 8 bytes y de 6.4 Gb/s para arquitecturas de 16 bytes. Las DDR han ido evolucionando hacia la DDR-2 y DDR-3. Siendo la DDR-2 estandarizada en el 2005 y la DDR-3 se ha estandarizado en el 2009 bajo normas JEDEC. La diferencia entre ellas está en velocidades mayores, menores tensiones de alimentación y ciertas diferencias en sus interfaces. La DDR-4 se espera su lanzamiento en el 2012.

Funciona a velocidades de 100 MHz, 133 MHz, 166 MHz y 266 MHz y frecuencias superiores. La nomenclatura es PC1600 (se trata de PC100 para DDRAM), PC2100 (PC133 con DDRAM), siendo 1600, 2100 la tasa de transferencia en Mbps.

La DDR-2 trabajan con 4 bits por ciclo de reloj, 2 de ida y 2 de vuelta. Las velocidades del reloj están entre 100 MHz y 300 MHz (efectivos: 400 MHz y 1200 MHz), lo que permite transferencias entre 3200 MB/s y 9600 MB/s.

La DDR-3 puede ser dos veces más rápida que la anterior. Las velocidades de reloj están comprendidas entre 133 MHz y 250 MHz, lo que permite transferencias entre 8530 MB/s y 16000 MB/s.

De la DDR-4 se esperan tasas de transferencia comprendidas entre 2133 MT/s y 4266 MT/s (Mega-Transferencias por segundo).

Las DDR, como factor positivo, presentan una arquitectura abierta, por lo que no paga derechos a fabricantes. Como factor negativo, en sus comienzos, Intel no apoyaba este tipo de arquitecturas aunque posteriormente se ha sumado a su evolución.

4.9.3.6. *Rambus DRAM (RDRAM).*

También se conoce como *Direct Rambus*. Son memorias construidas con bus de 16 bits y la frecuencia de reloj de 400 MHz. Trabaja, como las DDR, con flancos de subida y de bajada, alcanzando un ancho de banda de 1,6 GBs. Es el complemento de las tarjetas gráficas AGP, evitando cuellos de botella. Los circuitos se empaquetan en tarjetas denominadas

RIMMs (*Rambus Inline Memory Module*) no compatibles con los SIMMs (SDRAM).

Memoria desarrollada en cooperación con la empresa Rambus Company. Paga derechos a Intel.

4.9.3.7. Synchronous Link DRAM (SLDRAM).

Posee las ventajas de las SDRAM y la DDR RAM. Internamente es similar a una DDR. Incorpora un protocolo orientado a paquetes para el control/direcccionamiento y un sistema de calibrado de tiempos para mantener la compatibilidad con las antiguas memorias. Planteada para buses de 64 bits a velocidades de 200 MHz, 400 MHz efectivos, Alcanzando velocidades de 3.2 GB/s. Pensada para grandes servidores.

Esta memoria ha sido desarrollada por un consorcio de alrededor de 20 empresas, no debiéndose pagar licencias por su fabricación. Es la alternativa a las RDRAM y se plantea como estándar de los PCs de altas prestaciones.

4.9.3.8. Video DRAM (VRAM).

Se trata de la versión video de una FPM. Versión doble puerta de una DRAM. Uso para gráficos. Actualmente obsoleta, substituida por las SDRAM y por las SGRAM. - Puerta 1: Como una DRAM - Puerta 2: Solo salida Video.

4.9.3.9. Synchronous Graphics RAM (SGRAM).

Versión video de una SDRAM para adaptar gráficos. Añade un bit de máscara y escritura de bloque.

4.9.3.10. Pipeline Burst Static RAM (PBSRAM).

Opera en modo ráfaga, típicamente 3-1-1-1. Requiere un ciclo extra de reloj. Típico 4 ns. de acceso. Amplio rango de relojes.

4.9.3.11. Tabla Resumen.

A continuación veamos la evolución histórica de las memorias dinámicas, reflejada en la Tabla 19.

TABLA 19. Evolución histórica de las DRAM

Año	Tipo de memoria	Tacceso o Velocidad
1987	FPM	50 ns.
1995	EDO	50 ns.
1997	PC66 SDRAM	66 MHz.
1998	PC100 SDRAM	100 MHz.
1999	RDRAM	800 MHz.
1999-2000	PC133 SDRAM	133 MHz.
2000	DDR SDRAM	266 MHz.
2001	DDR SDRAM	333 MHz.
2002	DDR SDRAM	434 MHz.
2003	DDR-2 SDRAM	600 MHz.
2009	DDR-3 SDRAM	1000 MHz
2012	DDR-4 SDRAM	4.2 GHz

Capítulo 5

BUSES

Sócrates, al mirar las mercancías en el mercado, exclamaba: ¡Cuantas cosas hay que no necesito! (Diógenes Laercio: 180 - 240 d.C)

RESUMEN. La información, tanto generada como consumida por el computador, va a recorrer determinados caminos eléctricos denominados buses que van a presentar diferentes características tanto eléctricas como de estructura de datos.

5.1. INTRODUCCIÓN

En este capítulo se va a tratar el movimiento de la información en el computador. Esta información generada por la CPU del procesador, almacenada y/o generada en un dispositivo, necesita ser transportada por una serie de caminos denominados BUSES.

Los Buses son caminos eléctricos que unen dispositivos. Podemos distinguir tres tipos de Buses.

- Buses internos del Procesador: Unión de CPU con ALU y registros.
- Buses de comunicación CPU con Memoria.
- Buses destinados a E/S.

Los dos últimos Buses disponen de tres tipos de información:

- Bus de Datos.
- Bus de Direcciones.
- Bus de Control.

Normalmente estos Buses discurren a través de lo que se denomina una “Placa Madre”, en ella se insertan las distintas tarjetas electrónicas controladoras de dispositivos y se comunican con el Procesador por medio del

Bus de la Placa Madre. Podemos insertar tarjetas controladoras de Discos, tarjetas controladoras de Monitores, etc..

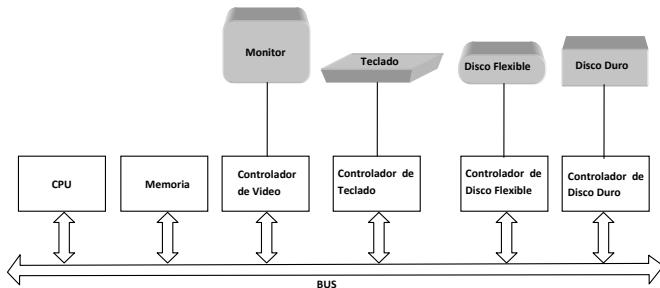


FIGURA 1. Conexión de Dispositivos en un Bus

Según criterios constructivos, la placa Procesadora junto con la memoria puede presentar los siguientes modos de montaje:

- Tarjeta insertada en la Placa Madre: Soluciones industriales (Incluso PCs industriales).
- Componentes montados sobre la Placa Madre: Mundo del PC.

Los aspectos más importantes, en cuanto al diseño de un Bus, son los siguientes:

- Ancho del Bus.
- Temporización.
- Arbitraje.
- Operaciones.

5.2. CARACTERÍSTICAS DE LOS BUSES

5.2.1. Ancho de Bus.

Por ancho de Bus entendemos el nº de señales eléctricas que transportan algún tipo de información (pistas de cobre). Ya se ha comentado que va a estar formado por Direcciones, Datos y Control. Con respecto al Bus de Direcciones, según distintas soluciones, nos movemos entre 16 bits y 32 bits. En máquinas con necesidad de mayor potencia de direccionamiento se llega a 64; téngase en cuenta que el direccionamiento es 2^n .

En cuanto a los Buses de Datos nos movemos entre 8 y 32 bits. Los Buses de control no suelen presentar problemas en cuanto a su tamaño.

Cuanto más denso sea un Bus más cara será la placa. Mayor densidad de pistas y más finas, además será necesario disponer de más capas para poder realizar una conexión correcta entre placas.

En el mundo del PC, una modificación del Bus puede provocar la incompatibilidad de las tarjetas existentes en el mercado con la nueva solución. Por ello los crecimientos de Bus o incorporación de nuevos Buses, debe dejar inalterada la forma de conexión con lo ya existente.

Esta modificación puede ser debida a un aumento de la velocidad o bien a la incorporación de nuevas señales con nuevos cometidos.

5.2.2. Temporización del Bus.

El intercambio de información entre diversos dispositivos, a través del Bus, puede estar regido por un Reloj (Bus Síncrono) o bien no depender de un reloj (Bus Asíncrono). En el caso síncrono, un reloj maestro genera una onda cuadrada que marca las transiciones. La frecuencia está comprendida entre 5 y 100 MHz. Un Bus ISA de PC trabaja a 8.33 MHz y un PCI trabaja a 33 MHz o a 66 MHz.

En el caso asíncrono, los ciclos de bus pueden tener un ancho cualquiera.

La mayoría de los Buses implementados son síncronos.

5.2.2.1. Buses Síncronos.

Un Bus síncrono puede estar movido por las siguientes señales:

- Petición de Memoria: /MREQ (indica acceso a memoria, no a E/S).
- Señal de Lectura: /RD (si “1” es una Escritura - WR).
- Señal de Espera: /WAIT (ciclo de espera generado por el dispositivo si este es lento).
- Buses de Datos y Direcciones.

La secuenciación de las señales se refleja en la Figura 2.

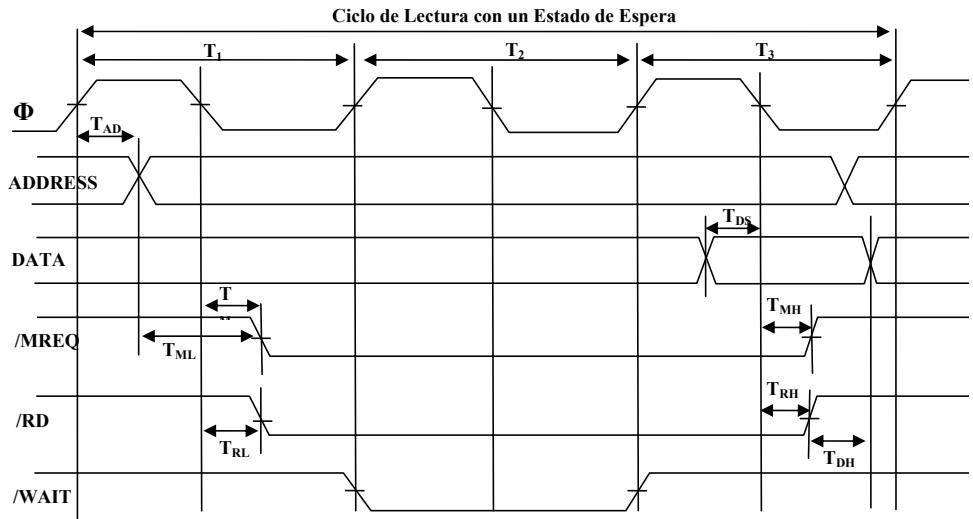


FIGURA 2. Bus Síncrono

5.2.2.2. Buses Asíncronos.

La introducción de este tipo de Buses, libera a los dispositivos de la dependencia del reloj. Los dispositivos irán a la velocidad que puedan ir sin necesidad de supeditarse al ancho del periodo del reloj. Las señales que se van a necesitar, además de las vistas anteriormente, son las siguientes:

- /MSYN : Master SYNchronitation.
- /SSYN : Slave SYNchronitation.

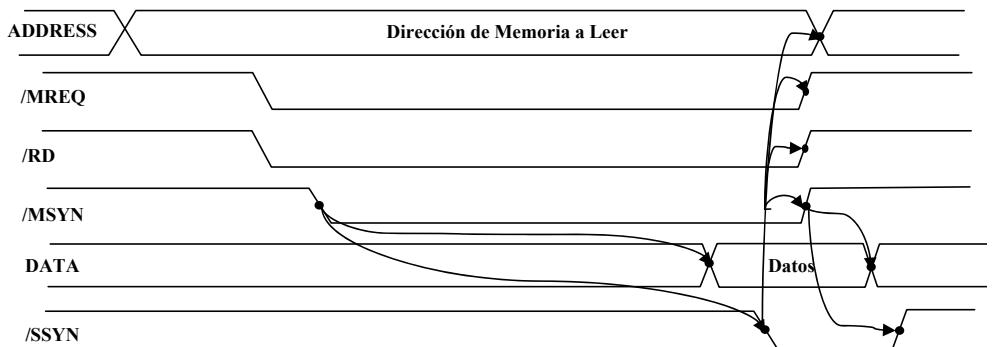


FIGURA 3. Bus Asíncrono

El dispositivo que controla el Bus (Master) habilita una señal de sincronización /MSYN, al ser entendida esta por el dispositivo Esclavo, este realiza

su actividad y cuando termina, este genera la señal /SSYN, es en este momento cuando el Master reconoce que están disponibles los datos y los lee, deshabilitando posteriormente las señales de /MREQ, /RD y /MSYN. El Esclavo, una vez que detecta la desactivación de /MSYN desactiva /SSYN. Resumiendo:

- Activación de /MSYN.
 - Activación de /SSYN al reconocer /MSYN.
 - Desactivación de /MSYN al reconocer a /SSYN.
 - Desactivación de /SSYN al reconocer la desactivación de /MSYN.

Este tipo de dialogo o protocolo hardware se denomina “*Handshake*” (apretón de manos). Algun autor lo denomina “saludo completo”.

5.2.3. Arbitraje de Bus.

Cuando dos o más dispositivos quieren acceder al Bus al mismo tiempo, se presenta un conflicto de colisión. No se permite que todos accedan en el mismo instante de tiempo. El conflicto se resuelve introduciendo un Arbitro de Bus, este mecanismo determinará quien de los dispositivos se lleva el control. Los mecanismos de arbitraje pueden presentar dos estructuras:

- Centralizado.
 - Descentralizado.

1. **Centralizado:** Este árbitro puede estar integrado en la CPU o bien ser un dispositivo diferenciado. Este árbitro determina quien es el siguiente en ser servido.

El Bus presenta una única línea de petición que consiste en un OR de todas las peticiones; no tiene manera de conocer quién es el peticionario. Cualquiera de los participantes puede pedir el Bus.

La línea de “Concesión de Bus” sale del árbitro y se encadena entre todos los participantes hasta llegar al último. La prioridad más alta le corresponde al dispositivo más próximo al árbitro y la más baja al más lejano (último). Este mecanismo se conoce como encadenamiento circular.

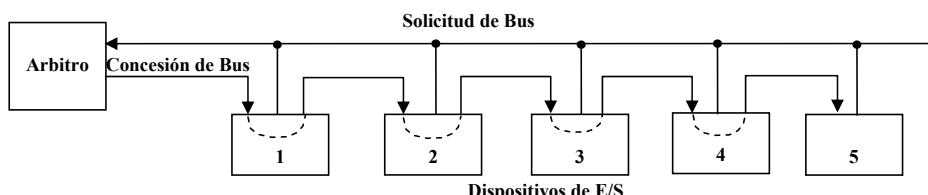


FIGURA 4 Bus Centralizado: Encadenamiento Circular de un Nivel

De lo anterior se desprende que si siempre pide el dispositivo más próximo al árbitro, nunca los menos prioritarios podrán acceder al Bus. Esta situación se resuelve disponiendo de más de una línea de petición, agrupando en ella (OR) los dispositivos de prioridad similar. De esta manera puede llegar al árbitro solicitudes diferenciadas, pudiendo dar servicio a estos dispositivos.

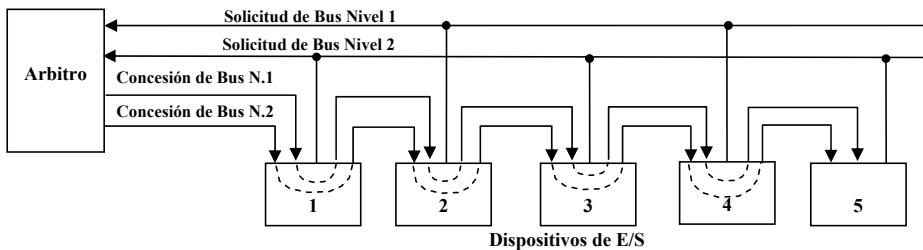


FIGURA 5. Bus Centralizado: Encadenamiento Circular de Dos Niveles

Mientras un dispositivo tiene la concesión del Bus (está realizando alguna transacción) no podrá darse servicio a otros dispositivos más prioritarios. Estos detectarán que está activa la señal de “Concesión de Bus” y no activarán su señal de “Petición”. Cuando se desactive aquella, el árbitro detectará la nueva petición y volverá a activar la concesión.

Se presenta un problema si la CPU comparte el Bus de la memoria con el de E/S, si el dispositivo de E/S es de carácter prioritario, la CPU retrasará su actividad con la memoria. Para evitar este tipo de conflictos, se recurre a disponer la memoria en un Bus distinto al dedicado a E/S.

2. **Descentralizado:** Un tipo de arbitraje descentralizado puede ser el disponer de un nº elevado de líneas de solicitud en un orden de prioridad. A cada dispositivo le llegan todas las líneas de solicitud pero solo se le asigna una. Todos los dispositivos ven el estado del resto de las líneas. Esto les permite conocer si son ellos los servidos o si serán los próximos en ser servidos.

Esta estructuración requiere más líneas de control pero evita el uso de árbitro, además limita el nº de dispositivos participantes.

Otro tipo de arbitraje descentralizado, es el que utiliza tres líneas. Una línea es un OR de las peticiones de los dispositivos, otra se corresponde con la función BUSY (ocupado) que es activada por el dispositivo que tiene concedido el Bus, la tercera y última es la señal de prioridad encadenada, de tal manera que si un dispositivo de mayor prioridad tiene concedido el Bus, esta señal pasará a los siguientes dispositivos desactivada, indicando que no pueden pedir Bus.

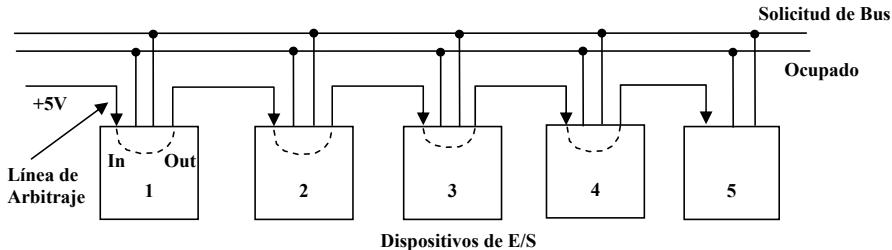


FIGURA 6. Bus Descentralizado: Arbitraje

En este tipo de arbitraje si un dispositivo de menor prioridad tiene concedido el Bus, habrá activado la señal de BUSY y si un dispositivo de mayor prioridad quiere activar su petición, no podrá realizar la petición porque verá esta señal activa, deberá esperar a su desactivación.

5.2.4. Operaciones de Bus.

Lo que se ha visto hasta el momento, es el acceso al Bus para lectura o escritura de un Dato principalmente de Memoria. Se dan otro tipo de transacciones como las que vamos a ver a continuación:

- **Transferencia por Bloques:** Pensemos en el caso de una memoria caché, cuando hay que traer de la MP a la MC una palabra, es necesario el traer la Línea (bloque) que contiene la palabra requerida, esta línea estará formada p. ej. por 16 palabras consecutivas de n bits cada palabra. Si se organiza una transferencia por bloque, será más eficiente que traer palabra a palabra accediendo a MP de manera consecutiva.

Mediante una señal de solicitud de transferencia de bloque, un primer dato indicando al dispositivo esclavo el nº de palabras a mover, podrá desencadenarse el acceso indicando solo la dirección de la primera palabra. La Figura 7 muestra el cronograma de una transferencia por bloques.

- **Acceso por Semaforización:** Se da el caso de disponer de una arquitectura multiprocesador, que comparten información de una misma área de memoria, entonces el acceso puede realizarse “escribiendo y leyendo” unos bits de control situados en la memoria RAM, de tal manera que cuando un procesador quiera acceder al área de memoria, lea primero el bit “semáforo” y si está “verde”,

lo ponga “rojo” para que no sea utilizado por otro de los procesadores. Si está en “rojo”, deberá esperar a que cambie.

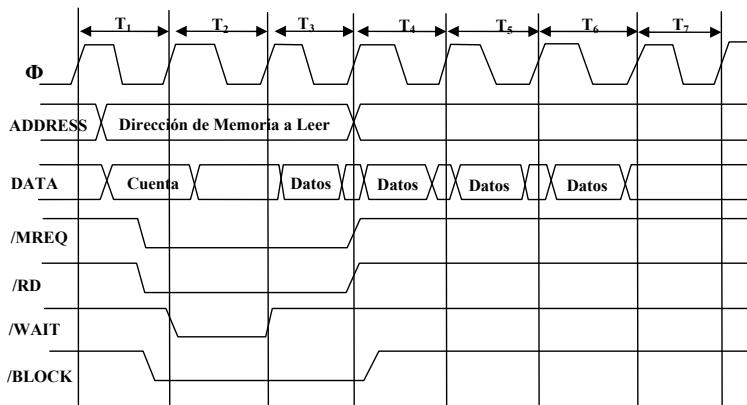


FIGURA 7. Transferencia de un Bloque

- **Servicio de Interrupciones:** El procesador central puede desencadenar una acción en un dispositivo periférico y en vez de escrutar constantemente a este dispositivo, puede continuar su proceso principal y entrar en un estado de “espera” que es resuelto por la llegada de una “interrupción” generada por el dispositivo periférico. La señal generada se denomina “Petición de Interrupción” (IRQx o IRx: *Interrupt Request*).

Cuando el procesador va a reconocer qué periférico ha provocado la interrupción (señal INTA: INTerrupt Acknowledge), se dirige a un dispositivo especial, denominado “Controlador de Interrupciones” que le devuelve en el Bus un código del dispositivo que ha provocado la interrupción. Este código es utilizado por el procesador como un “índice” que va a apuntar (Tabla de Apuntadores) a una dirección de la memoria donde se va a tratar la interrupción. Este índice recibe el nombre de “Vector de Interrupciones”.

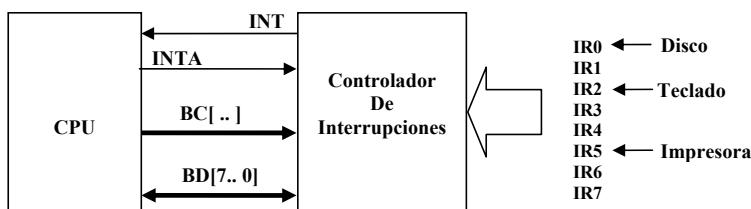


FIGURA 8. Controlador de Interrupciones

Para finalizar, comentar que los dispositivos controladores de interrupciones, pueden conectarse en cascada, de tal manera que pueden manejarse 8x8 interrupciones.

5.3. EJEMPLOS DE BUSES DE PROPÓSITO GENERAL: ISA y PCI

5.3.1. ISA.

Es en 1981 cuando IBM anuncia un ordenador personal bajo la sigla PC. El Bus inicial de los IBM PC (PC/XT) está basado en el Procesador 8088 de Intel. Este Bus comprende 62 líneas de señales, con la siguiente distribución (se muestran las más significativas):

- 20 para la Dirección de memoria (A0-A19).
- 8 para los Datos (D0-D7).
- 5 para Lectura Memoria (MEMR), Escritura Memoria (MEMW), Lectura E/S (IOR), Escritura E/S (IOW) y registro de la dirección (ALE).
- Señales de solicitud de Interrupciones (IRQ3-IRQ7).
- Señales para trabajo en DMA (4 canales DMA): petición (DRQ1-DRQ3), reconocimiento (DACK0-DACK3), habilitación de direccionamiento (AEN).

Estas señales discurren por la Placa Madre y van pasando por conectores insertados en ella, es en estos conectores, separados 2 cm., donde se van a insertar las tarjetas controladoras de diversos dispositivos.

Cuando se introdujo el Procesador 8086, este disponía de un Bus de Datos de 16 bits, necesitándose multiplexar el Bus de Datos al disponer solo de 8 patas para el Bus.

Al introducirse el Procesador 80286, se produjo el siguiente problema; este procesador presenta un ancho de Bus de datos de 16 bits, por lo tanto si se hubiese diseñado un nuevo Bus (nuevas tarjeta madre), las tarjetas diseñadas hasta el momento, habrían sido incompatibles con el nuevo Bus, esto hubiera provocado un cierto caos en la industria, tanto a nivel de usuario como de fabricante de tarjetas. Por ello, se implementó una extensión del Bus (PC/AT), creando una nueva fila de pistas y unos nuevos conectores, dejando los ya existentes sin tocar. Esto permitía utilizar las tarjetas desarrolladas hasta el momento y las de nuevo desarrollo.

Este segundo conector consta de 36 contactos (36 señales), ocupados por más líneas de Dirección (4 más, pasando de 20 a 24) (LA20-LA23), Datos

(8) (SD8-SD15), Control de Bus de Datos (8 o 16bits) (MEM CS16), canales de DMA (DRQ0, DRQ5-DRQ7), líneas de Interrupción (5) (IRQ10-IRQ12 y IRQ14-IRQ15) y líneas de alimentación.

En un momento determinado, (aparición del PS2) IBM decide dejar de utilizar éste Bus y comienza a incorporar uno nuevo, el Microchanel. Es en este momento cuando el consorcio de fabricantes de PC compatibles y tarjetas para PC, decide normalizar el Bus existente, con ciertas adaptaciones, denominándolo ISA (*Industry Standard Architecture*). Básicamente es un Bus PC/AT a 8,33 MHz. Esta decisión mantuvo la compatibilidad con todo el mercado existente.

Posteriormente el Bus ISA se extendió a 32 bits con funciones adicionales, denominándose EISA (*Extended ISA*). Hoy en día este bus ha quedado obsoleto y, fundamentalmente, ha sido substituido por el Bus PCI.

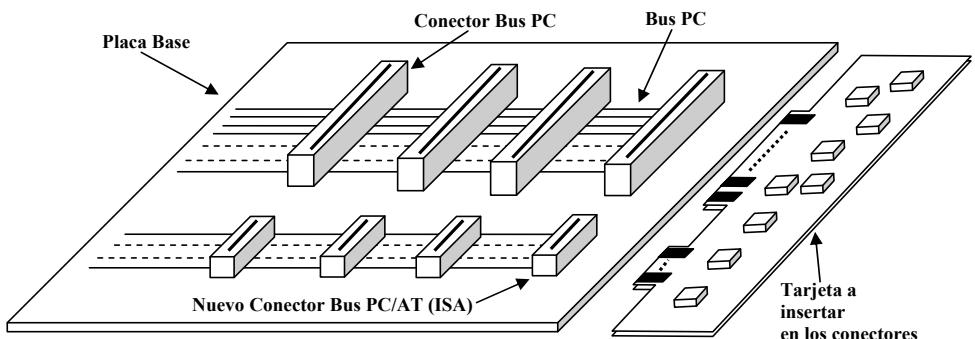


FIGURA 9. Placa Base con Bus PC/AT (ISA)

5.3.2. PCI.

Debido al incremento de velocidad de los procesadores y a los mayores requerimientos de todos los equipos periféricos, había que aumentar el ancho de banda del Bus actual. Se podía detectar necesidades como las siguientes:

- 67.5 MB/s para pantallas en continuo movimiento.
- 135 MB/s para vídeo.
- Etc.

El Bus ISA puede transferir datos a 16.7 MB/s y el EISA a 33.3 MB/s, de estos datos se concluye que era necesario el desarrollar otro Bus que pudiera cumplir con estos requerimientos. Es Intel quien en 1990, diseña el Bus PCI (*Peripheral Component Interconnect*). Es un Bus propietario

pero de dominio público. Existe un consorcio para el control y desarrollo de equipos basados en el Bus PCI.

EL Bus PCI ha pasado por las siguientes versiones: PCI 1, PCI 2.0, PCI 2.1 y PCI 2.2. Actualmente opera a 66 MHz con un ancho de banda total de 528 MB/s.

Para continuar manteniendo la compatibilidad con “casi todo” lo existente en el mercado, Intel ideó una solución consistente en disponer de tres o más Buses por computador. La CPU se comunica con un Bus especial para Memoria, crea un puente a PCI y este crea un puente a ISA y a controladores IDE (vease la Figura 10).

Actualmente se da una circunstancia especial, debida a la evolución tecnológica de la industria del hardware, cuando se crea el Bus PCI, todos los circuitos operan a 5 Vcc. pero la evolución de la microelectrónica, incorpora tecnologías de 3,3 Vcc, por lo tanto tarjetas PCI que trabajan a esta tensión no deben conectarse a los 5 Vcc, y viceversa, por ello se introducen unas pestañas que impiden conectar tarjetas de 3,3 Vcc en 5 Vcc. También existen tarjetas universales que trabajan con ambas tensiones (convertidor 5/3.3).

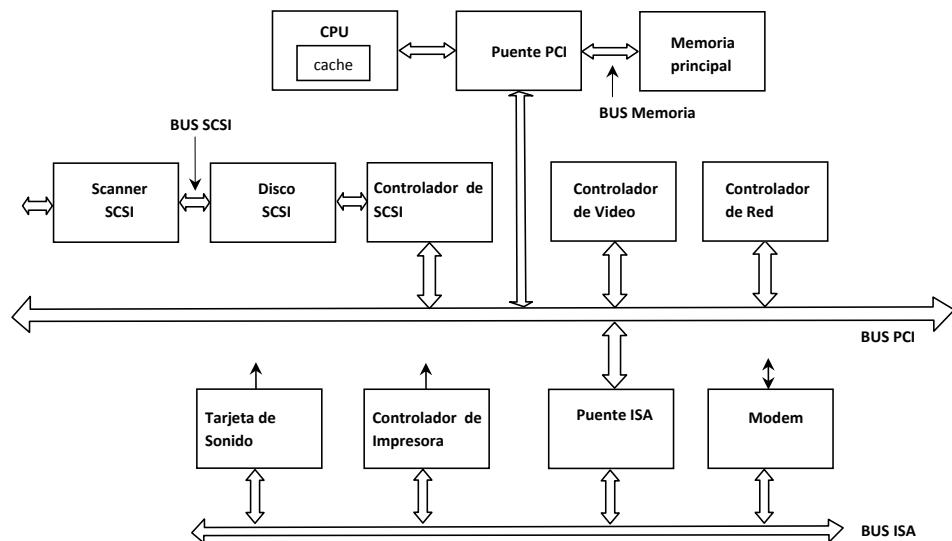


FIGURA 10. Interconexión de Bus PCI con ISA

Existen versiones de 32 bits (conector de 120 terminales) y de 64 bits (conector de 120 terminales y conector adicional de 64 terminales).

Las tarjetas PCI funcionan a 33 MHz, 66 MHz y frecuencias superiores. Es un Bus síncrono. Todas las transacciones se realizan entre un Maestro y un Esclavo. Las líneas de Direcciones (64) y de Datos (64), están multiplexadas.

Tratamiento de la Información: El tratamiento de la información es como sigue:

- Las transacciones se inician con el flanco de bajada del reloj, a diferencia del bus ISA que comienza con el flanco de subida.
- Durante una Lectura, en el ciclo nº 1 (T1), el Master coloca una Dirección en el Bus, en el flanco de bajada del reloj. Coloca la señal C/BE# para indicar Lectura de Palabra, Bloque, etc.; termina habilitando FRAME# para desencadenar la transacción.
- En el ciclo nº 2 (T2), se invierte el Bus para que pueda utilizarlo el Esclavo. El Master deja el Bus en Alta Impedancia. El Master modifica C/BE# para indicar que bytes de la palabra direccionada quiere leer.
- En el ciclo nº 3 (T3), el esclavo envía los datos solicitados. El Esclavo activa DEVSEL#, coloca los Datos y activa TRDY# para avisar al Master.
- Si el Esclavo es más lento que el Master, activa DEVSEL# pero introduce ciclos de espera y activa TRDY# cuando ya está listo.
- El ciclo nº 4 (T4), no tiene efecto externo, se trata de un ciclo muerto.
- Si la transacción es de escritura, se requieren también tres ciclos de reloj, aunque no necesita inversión durante el ciclo nº 2.

La Figura 11 muestra la evolución de los ciclos.

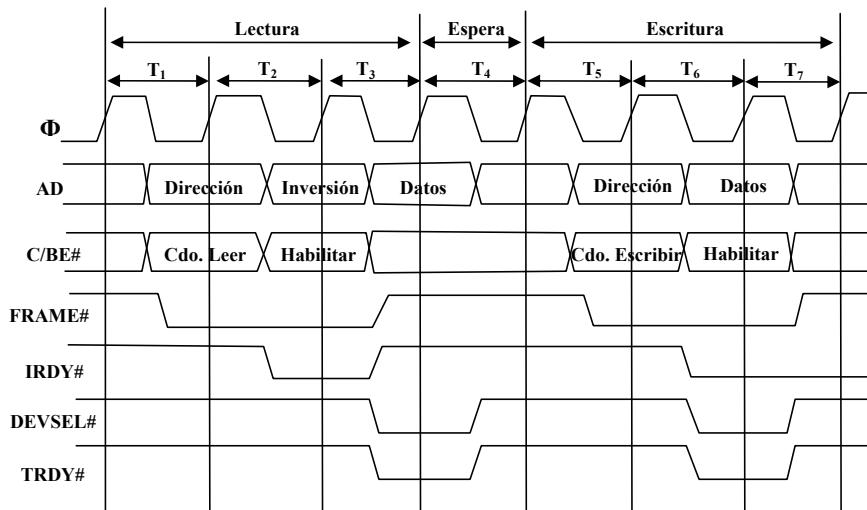


FIGURA 11. Ciclo de Bus PCI

Arbitraje: El problema del arbitraje se resuelve de la siguiente manera:

Un dispositivo realiza una petición REQ# (REQUEST) y el Arbitro la concede GNT# (GRANT). En este momento el dispositivo puede utilizar el Bus en el siguiente ciclo. El arbitraje se resuelve de las formas ya comentadas, circular, por prioridad y otros sistemas (vease la Figura 12).

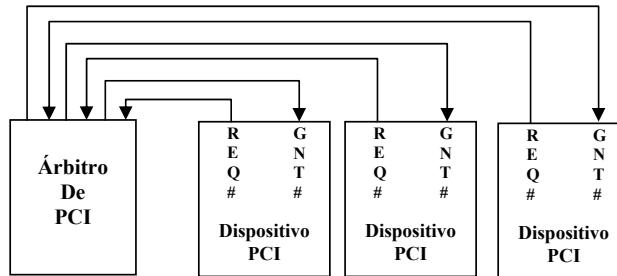


FIGURA 12. Arbitraje en un Bus PCI

El paso de un dispositivo a otro introduce siempre un “ciclo muerto”, este no se producirá solo si el que pide el Bus es el mismo dispositivo que lo tiene en su poder y quiere continuar transacciones, no existiendo ningún otro dispositivo que lo esté pidiendo en ese instante.

Si un dispositivo lleva utilizando mucho tiempo el Bus, el árbitro puede quitárselo, deshabilitando la señal GNT#, de esta forma el dispositivo se da cuenta y termina para el siguiente ciclo.

Conexión y Operación (Plug and Play): Los dispositivos PCI disponen de un área de configuración de 256 bytes. Esta área puede ser leída por otros dispositivos, activando la señal IDSEL. Este espacio de memoria contiene información sobre las propiedades del dispositivo. La capacidad de Plug and Play de algunos sistemas operativos, visitan esta zona de configuración para reconocer al dispositivo.

5.4. EJEMPLOS DE BUSES INDUSTRIALES: VME, MULTIBUS II, NuBus y FUTUREBUS

Nos centremos en los buses de mayor penetración histórica como VME, Multibus II, NuBus y Futurebus orientados a 32 bits [1][12], dejando para otra lectura los buses: STEBus, STD y G64/G96. Téngase en cuenta que dada la evolución de los sistemas, muchos de estos buses han dejado de estar presentes en el entorno industrial, así como en las estaciones de trabajo.

5.4.1. VME.

Bus asíncrono impulsado por Motorola, en 1981, involucrando a varias compañías como: Philips/Signetics, Mostek y Thomson CSF. Posteriormente se crea, en 1984, la asociación VITA (*VMEbus International Trade Association*) con más de 300 fabricantes. Se trata de una redefinición de un Bus existente, el Versabus, a las Normas Europeas, estas fundamentalmente diferían en los tamaños y en el formato de las conexiones. Versabus pasa a denominarse VME (*Versa Module Eurocard*) siendo su principal objetivo el dar cabida a los recientes μ P de 32 bits.

Su uso se da fundamentalmente en el entorno industrial. Se normaliza bajo las siguientes Normas: IEC 821 VMEbus y ANSI/IEEE 1014-1987. La tasa de transferencia está entre 24 MB/s y 57 MB/s máximo; siendo el dato más común de 40 MB/s máximo.

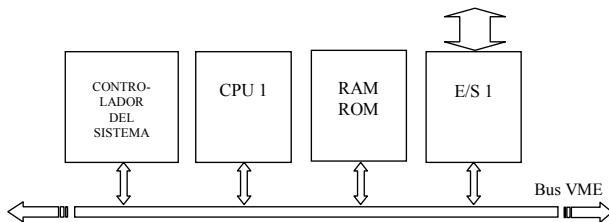


FIGURA 13. Bus VME

El Bus nace por el impulso dado por la familia de μ P s 680xx de Motorola, esta familia pasa por los siguientes μ P s: 68000, 68008, 68010, 68012,

68020, 68030 y 68040. Las tarjetas normalizadas en Europa presentan dos tamaños:

- Simple Europa: 100mm × 160mm
- Doble Europa: 233,68mm × 160mm

En ambas tarjetas el sistema de conexión se realiza por medio de conectores DIN (extensivamente utilizados en industria). La Norma define dos conectores denominados P1 y P2, ambos de 3x32 pines. La tarjeta Simple Europa (rack 3U: 5.25") lleva el P1 y la Doble Europa (rack 6U: 10.5") P1 y P2 según se muestra en la Figura 14.

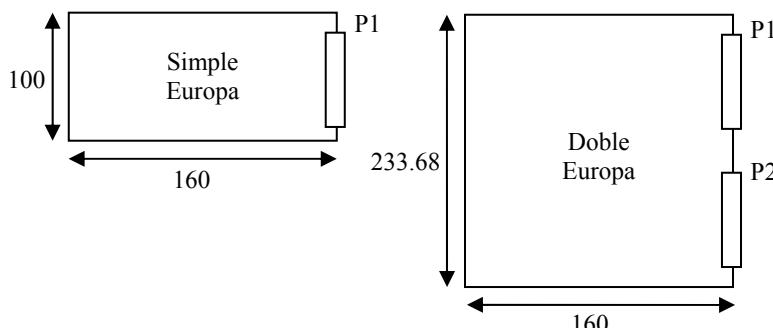


FIGURA 14. VME: Tarjetas Simple y Doble Europa

El conector P1 permite trabajar con la siguiente arquitectura:

- Bus de Datos de 16 bits: D0-D15
- Bus de Direcciones de 24 bits: A01-A23. A0 no aparece ya que se direccionan solo direcciones pares. 16 M de direccionamiento.
- Arbitraje de Bus: 4 líneas (trabajo en modo DMA) BREQ (petición de bus), BUSY (bus ocupado por el maestro), BGIN/BOUT (entrada/salida de un módulo: daisy chain).
- Interrupciones: 8 líneas; IRQ1-IRQ7 (petición) y IACK (reconocimiento).
- Resto de señales de Control: AM0-AM5 (modificador de dirección), AS (*Strobe* de la dirección) y LWORD (dato de 32 bits).
- Arbitraje: BR0-BR3 (petición de bus), BG0OUT-BG3OUT (concesión de bus a la siguiente tarjeta), BG0IN-BG3IN (bus concedido), BBSY (bus ocupado) y BCLR (petición de abandono de bus por otra petición prioritaria).
- Alimentaciones.

La extensión del conector P2 permite la siguiente ampliación:

- Bus de Datos: incorpora los 16 bits de datos restantes, obteniendo-se 32 bits de datos. D16-D31
- Bus de Direcciones: incorpora los 8 bits de direcciones restantes, obteniendose 32 bits de direcciones; permitiendo un espacio de di-recciones de 4 G. A24-A31.
- Resto de líneas de libre disposición.

El Bus VME contiene otros tres buses auxiliares: VSB, VMS y VXS.

- VSB (*VME Subsystem Bus*): Bus asíncrono de 32 bits con Datos/direcciones multiplexados. Las señales se dividen en dos gru-pos: Bus de Transferencia de Datos (DTB) y Bus de Arbitraje. VME admite varios subbuses VSB. Se orienta, principalmente, a la ampliación de memoria local, canales de E/S y canales DMA.
- VMS: Bus serie síncrono con velocidades entre 2-9 Mb/s con técni-ca de paso de testigo (*token passing*). Orientado a la comunicación entre placas.
- VXS: Bus serie de interconexión a la red de la empresa. Inicial-mente bajo protocolo Ethernet aunque se estudian otras tecnolo-gías, p.ej., Infiniband. La velocidad es del orden de GB/s.

La Figura 15 muestra los tiempos teóricos de una Lectura o Escritura sobre el Bus VME. Las barras delante de la señal, significan señal activa por nivel bajo.

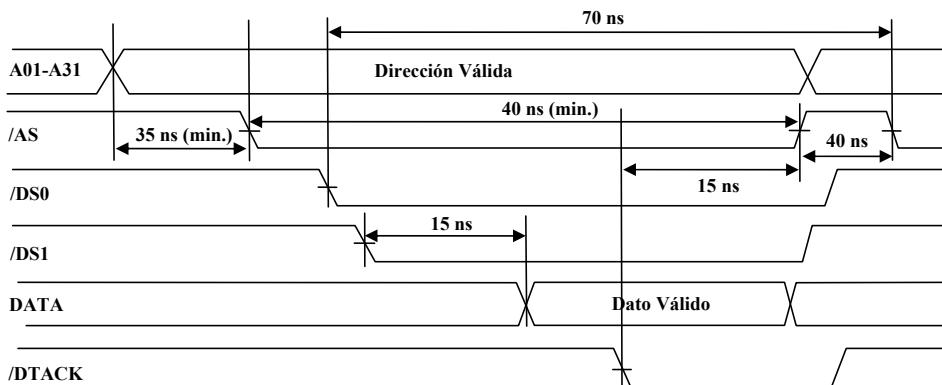


FIGURA 15. VME: Operación de Lectura o Escritura

Debido a la necesidad surgida por la comprobación de instrumentación de manera automatizada, en 1987 se introduce el bus VXI-1 (**VME eXtensions for Instrumentation System Especification**) revisión 1. La revisión 3 del

protocolo, orientada a VME64 64-bit de transferencias, es normalizado como IEEE Std. 1155-1992. Protocolo de amplio uso en sectores como: Telecomunicaciones, Militar, Aeroespacial y aplicaciones de adquisición de datos. VME64 presenta una tasa de transferencia máxima de 80 MB/s y el VME64x de 160 MB/s.

5.4.2. MULTIBUS II.

Previo a este bus, surge el Multibus I en torno a 1977, diseñado por Intel y orientado a 8/16 bits; básicamente es un Bus multiproceso, paralelo asíncrono con 24 líneas de Dirección, 16 de Datos, 12 de Control, 9 de Interrupciones y 6 de intercambio de Bus (control de Bus) y un ancho de banda de 10 MB/s. Intel junto a otras 18 empresas comienza un proceso de normalización de éste bus. Es en 1985 cuando se observa la madurez del mercado de tarjetas orientadas a Multibus II; su normalización comienza bajo el código IEEE P1296.

Multibus II se trata de un Bus de Buses: PSB, LBX, SSB, SBX y Multichannel; siendo el Bus PSB el principal.

- **PSB (Parallel System Bus):** Protocolo síncrono, Direcciones y Datos multiplexados, soportando una transferencia de 40 MB/s.
- **LBX II (Local Execution Bus):** Bus paralelo síncrono con transferencias hasta 48 MB/s. Su función principal es ampliar la memoria local, además, puede trabajar de forma similar al PSB, descongestionando a éste. El Bus de Datos puede trabajar con 8, 16, 24 o 32 bits y el Bus de Direcciones contiene 26 bits. Permite conectar hasta 6 tarjetas no separadas más de 5" (12,7 cm).
- **SSB (Serial System Bus):** Bus de bajo coste permitiendo la conexión de 32 dispositivos máximo. La conexión se realiza por medio de dos líneas de datos en colector abierto. El acceso es aleatorio con detección de colisión CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) como el protocolo Ethernet.
- **SBX (I/O Expansion Bus):** Compatible con Multibus I y, por lo tanto, con los productos desarrollados para Multibus I. Bus paralelo que conecta pequeñas tarjetas de expansión al bus PSB.
- **Multichannel:** Compatible con Multibus I y, por lo tanto, con los productos desarrollados para Multibus I. Bus paralelo asíncrono que conecta Memoria y E/S. Su velocidad máxima de transferencia es de 8 MB/s.

Las transferencias en el bus pueden ser de tres maneras: simple, secuencial y difundida (*broadcast*). Una transferencia simple consta de un ciclo

y se efectúa entre dos módulos. En la transferencia secuencial están involucrados dos o más datos y en la transferencia por difusión el módulo peticionario direcciona a varios módulos al mismo tiempo y la operación es exclusivamente de escritura.

Centrándonos en el Bus PSB, las principales señales son las siguientes:

- Direcciones y Datos: 32 (AD0-AD31) multiplexadas. Se añaden 4 señales (PAR0-PAR3) destinadas al control de la paridad de cada byte (PAR0: AD0-AD7, .., PAR3: AD24-AD31).
- Arbitraje: ARB0-ARB5 (arbitran 3 fases diferenciadas), BREQ (bus Request)
- Control de transferencias: SC0-SC9 con diferentes funcionalidades; SC8-SC9 controlan la paridad de SC4-SC7 y SC0-SC3 respectivamente.
- Excepción: BUSERR (por error de paridad) y TIMEOUT (generada por retardo excesivo en un dispositivo).

La Figura 16 muestra una operación de Lectura en una transferencia simple.

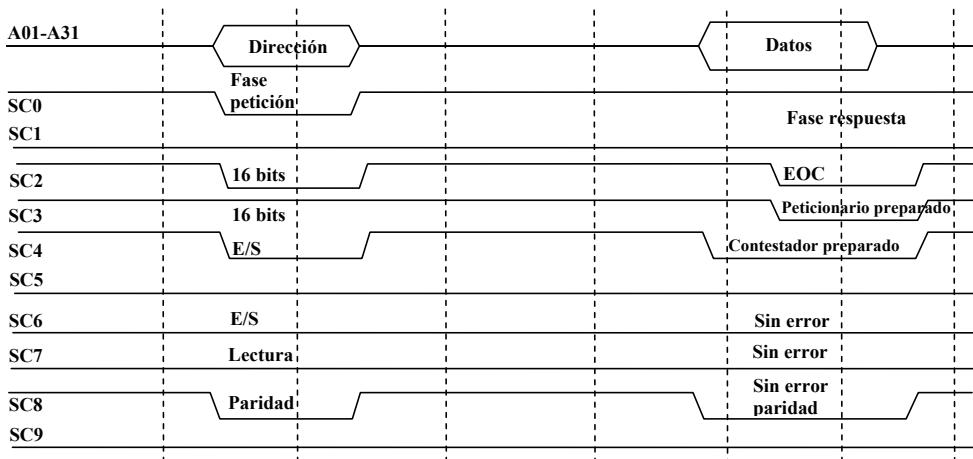


FIGURA 16. Multibus II (PSB): Operación de Lectura en transferencia simple

5.4.3. NuBus.

Es en 1981 cuando, en el MIT (*Massachusetts Institut of Technology*), se presenta el NuBus, licenciándose a Western Digital (AT&T). En 1982 Western Digital lo traspasa a Texas Instrument. Incorporado por Apple en el Macintosh II. National Instrument desarrolla tarjetas NuBus; también se incorporan GW Instrument y Strawberry. NuBus presenta unas elevadas prestaciones además de simpleza. El objetivo inicial es el dedicarlo a estaciones de trabajo aunque también evoluciona hacia el control industrial.

Es un Bus síncrono orientado a 32 bits de Datos/Direcciones multiplexados sobre las mismas líneas. Con una previsión de hasta 37,5 MHz de ancho de banda con transferencia en bloques. El periodo del reloj es de 100ns asimétrico (75/25). El espacio de memoria es único para Memoria, I/O e Interrupciones. Admite tarjetas Maestras (multimaestro) y Esclavas y el tamaño de las tarjetas es doble Europa (366 x 280mm) (rack 9U). Las tarjetas disponen de 3 conectores (P1, P2 y P3) de 96 contactos cada uno. P1 concentra las señales y P2-P3 las alimentaciones, dejando el resto de patas libres para el usuario. La Figura 17 muestra un esquema de bloques de este bus.

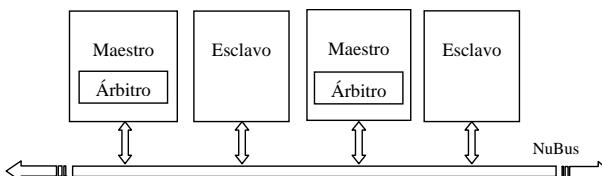


FIGURA 17. NuBus: Esquema de bloques

Si bien, en un principio, es un Bus orientado a 32 bits, TI lo adapta a 8 y 16 bits, además de 32 bits, con el fin de ampliar su funcionalidad a μ P s de menores prestaciones pero de amplia presencia en el mercado.

Las señales más significativas son las siguientes:

- Direcciones y Datos: 32 (AD0-AD31).
- Control de transferencias: START, ACK, TM0-TM1 (transferencias de L/E), SP (paridad), SPV (paridad válida).
- Generales: ID0-ID3 (identificación de tarjeta en el bus).
- Arbitraje: ARB0-ARB3 (control de bus), RQST (bus Request).

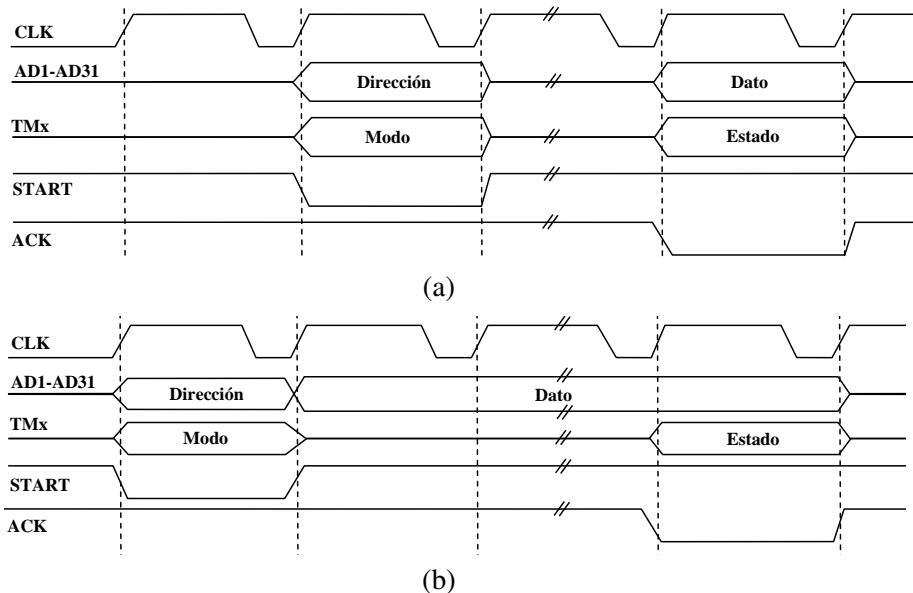
La Figura 18 muestra las operaciones de Lectura/Escritura sobre NuBus:

Macintosh II modifica ligeramente el bus introduciéndole una interrupción, no hay paridad y no hay transferencia de bloques.

Atendiendo a lo comentado sobre su simpleza, reseñar que, en cuanto a transferencias, presenta 3 líneas frente a 6 de Futurebus o 10 de Multibus II. Además, es de verificación simple e independiente del microprocesador que lo controle.

Industrialmente está por detrás de VME y Multibus II. En el bus solo hay dos operaciones básicas: Lectura/Escritura. Las transferencias de datos pueden ser simples (un dato) o múltiples (bloque de datos). La interrupción se trata como una operación de escritura (del módulo que genera la interrupción) sobre un área de memoria específica.

La norma que describe el bus NuBus es IEEE P1196 y está redactada por: TI, AT&T, Apple y MIT.



5.4.4. FUTUREBUS.

El desarrollo de este bus está motivado por encontrar un estandar que compatibilice la mayor parte de sistemas; es en 1979 cuando IEEE crea un grupo de trabajo para crear el bus del “futuro”; este trabajo obtiene sus frutos en 1987, creándose la norma IEEE P896¹ y en 1989 se deja como IEEE 896; a esta revisión le siguen la 896.1 y 896.2.

Es un bus asíncrono de Datos/Direcciones multiplexados y orientado a 32 bits, la velocidad está marcada por el módulo más lento, por lo tanto, si la tecnología evoluciona, mejoran las prestaciones. La velocidad máxima teórica es superior a 270 MB/s. Téngase en cuenta que Multibus II, VME o NuBUs presentan una velocidad máxima en torno a 50 MB/s. Una característica importante es que permite la extracción e inserción de tarjetas “en caliente”, es decir, no es necesario apagar el sistema para realizar estas operaciones, si bien, requiere una secuencia de acciones por parte del usuario (IEEE 896.2).

Permite múltiples tarjetas Maestras y Esclavas; la posesión del bus se realiza por medio de un arbitraje descentralizado; ninguna tarjeta controla al resto. Además, considera que las tarjetas que disponen de microprocesador, pueden contener memoria caché y puede existir una memoria conectada al

¹P: provisional

Futurebus y compartida por todos los participantes del Bus. Un esquema se presenta en la Figura 19.

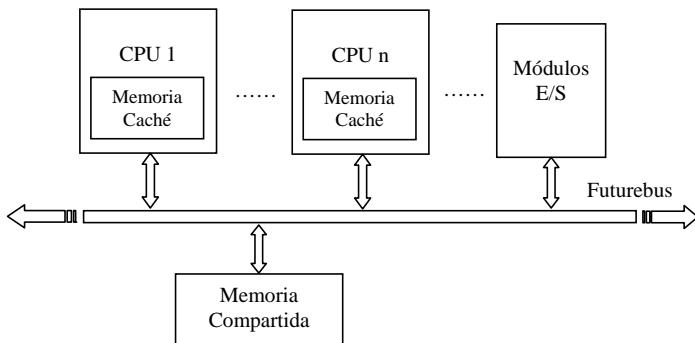


FIGURA 19. Futurebus: Esquema de bloques

AL ser un Bus multiproceso, se ha cuidado que exista una coherencia entre las memorias caché de cada μ P, así como con la memoria compartida. Hay una serie de normas (opcionales) que permiten mantener esta coherencia² (IEEE 896.2). Cuando un μ P ha actualizado datos en su caché, estos son enviados al bus para actualización del resto de μ P s (*write-through caching*), los μ P s pueden actualizar los datos o bien marcarlos como no válidos. Cada línea de caché dispone de 3 bits para indicar su estado. El protocolo MOESI (*Modified, Owned, Exclusive, Shared e Invalid*) proporciona esta coherencia.

- Modified*: Ese dato es el único válido en el sistema
- Owned*: La caché es responsable de la integridad del dato aunque otras cachés puedan disponer de ese dato.
- Exclusive*: La caché tiene la única copia del sistema pero no es propietaria del dato. Si otra caché lee el dato de la memoria compartida, deja de ser exclusivo.
- Shared*: Otras cachés tienen una copia de ese dato.
- Invalid*: El dato no debe ser leido de la caché por no ser válido (no está actualizado).

Las señales son las siguientes:

- Direcciones y Datos: 32 (AD0-AD31)
- Control de transferencias: TG (información sobre transferencia de Datos/Direcciones), TP (paridad de TG), CM0-CM5 (comando de

²Sobre estos aspectos el Capítulo 8 muestra diversas estrategias de mantenimiento de la coherencia en cachés.

transferencias en fases de direccionamiento y datos), BPW-BPX-BPY-BPZ (paridad de bus: $W \rightarrow AD24-AD31$, $X \rightarrow AD16-AD23$, $Y \rightarrow AD8-AD15$, $Z \rightarrow AD0-AD7$), CP (paridad de las señales de comando), ST0-ST3 (status respuesta de esclavos en fases de direccionamiento y datos)

- Sincronismo: AS-AK (sincronización de la dirección (Maestro → Esclavo) - reconocimiento (Esclavo → Maestro)), AI (reconocimiento adicional), DS-DK (sincronización de datos (Maestro → Esclavo) - reconocimiento (Esclavo → Maestro)), DI (reconocimiento adicional)
 - Arbitraje: AP-AQ-AR (secuencia de *handshake*), AC (control en adquisición)
 - Generales: GA0-GA4 (identificación de tarjeta en el bus)
 - Bus serie: SB0-SB1 (bus auxiliar serie)

La Figura 20 muestra una operación de Escritura.

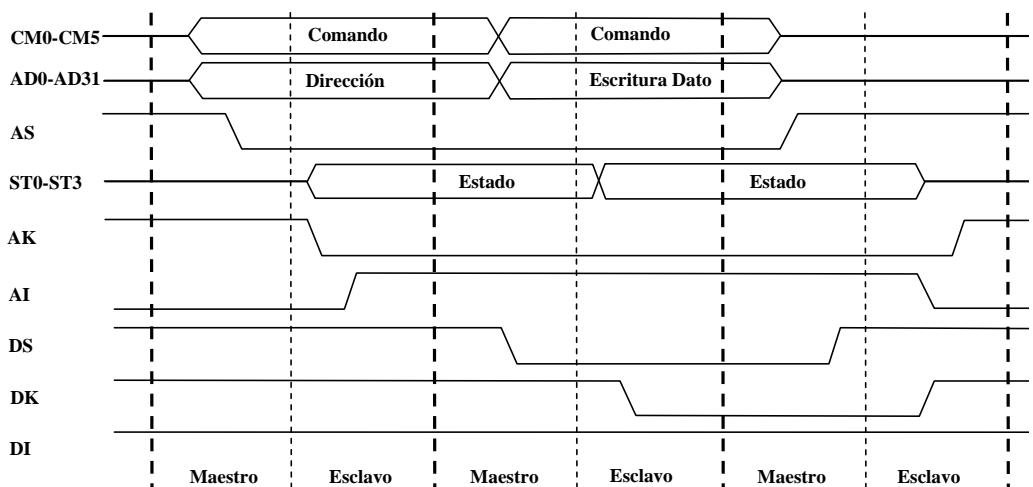


FIGURA 20. Futurebus: Operación de Escritura

5.5. EJEMPLOS DE BUSES SERIE: USB, FIREWIRE

5.5.1. USB.

Con el fin de facilitar la integración de nuevos dispositivos al en torno del computador, nueve compañías, líderes mundiales del sector informático, crean un consorcio para diseñar un nuevo Bus para un nuevo modo de conexión al computador. Crean el Bus USB: Bus Serie Universal. Las ideas iniciales del planteamiento, son las siguientes:

1. El usuario no tendrá que actuar sobre microinterruptores en el dispositivo.
2. El usuario no tendrá que abrir el computador para incorporar nuevos dispositivos.
3. Solo existirá un único tipo de cable de interconexión.
4. Los dispositivos de E/S serán alimentados desde el cable de Bus.
5. Podrán conectarse hasta 127 dispositivos por computador.
6. Podrán conectarse dispositivos que trabajen en tiempo real (teléfono, etc.).
7. Los dispositivos podrán conectarse estando en funcionamiento el computador (conexión en caliente).
8. No se necesitará reiniciar el computador cada vez que se instale un dispositivo.
9. El Bus y sus dispositivos de E/S deberán ser de bajo costo.

Veamos ordenadamente las versiones de éste protocolo:

1. **USB 1.0 (1.1)** se presenta en septiembre de 1998 (Compaq, Intel, Microsoft y NEC); la velocidad máxima es de 12 Mbits/s, denominada “*full-speed*” que permite un ancho de banda total del Bus USB de 1.5 MB/s (12/8), tasa suficiente para la mayoría de dispositivos periféricos. Los dispositivos más comúnmente utilizados son: teclados, ratones, scaners, etc. Dispone de otra velocidad de transmisión de 1.5 Mbits/s, denominada “*low-speed*”. El controlador detecta cual es la velocidad máxima de trabajo del periférico conectado y se adapta. La transmisión es en modo *half-duplex*. El conector presenta 4 pines: 2 líneas de alimentación y 2 líneas de comunicación.
2. **USB 2.0** se introduce en abril del 2000 (se incorporan Hewlett-Packard, Lucent y Philips), alcanzando una velocidad máxima de 480 Mbits/s lo que permite un ancho de banda de 60 MB/s. Este modo se denomina “*high-speed*”. La transmisión es en modo *half-duplex*. La compatibilidad está garantizada con USB 1.1. Al aumentar la velocidad, se incorporan dispositivos con mayores requerimientos como cámaras fotográficas, pendrives e impresoras o scaners más exigentes.
3. **USB 3.0 (2008)** eleva a 4,8 Gbps (600 MB/s) la capacidad de transferencia. Se mantiene el cableado interno de cobre para asegurar la compatibilidad con las tecnologías USB 1.0 y 2.0. Añade cinco líneas, dos de ellas se usan para el envío de información y otras dos para la recepción, de forma que se permite el tráfico bidireccional, en ambos sentidos al mismo tiempo (transmisión *full-duplex*). El aumento del número de líneas permite incrementar la velocidad de

transmisión desde los 480 Mbps hasta los 4,8 Gbps. También se conoce como “*super-speed*”. La compatibilidad está garantizada con USB 2.0. El conector amplía el número de conexiones a 9, ya que, como se ha comentado anteriormente, se ha introducido un segundo canal de comunicaciones para soportar la bidireccionalidad.

4. **USB OTG (OnThe-Go):** es una variante de la versión 2.0 demandada por los fabricantes de periféricos para poder conectar los periféricos entre ellos sin necesidad de depender de un PC, p.ej., cámara fotográfica con impresora. Es introducida en diciembre del 2001. Conlleva un cambio en los conectores, siendo estos más pequeños.

De un cable principal se van extrayendo las ramificaciones para los dispositivos. La conexión de un nuevo dispositivo es detectado por el controlador del Bus, interrumpiendo al S.O y comenzando un ciclo de reconocimiento del tipo de dispositivo y su ancho de banda permitido. Si el ancho de banda es permitido por el S.O, teniendo en cuenta los dispositivos existentes en el Bus, le asigna una dirección única (1-127). La dirección “0” está asignada a los dispositivos que no han sido reconocidos todavía.

El Bus consta de cuatro hilos, dos de alimentación (5Vcc, 0V) y dos de Datos. El tráfico de información va de Controlador a Dispositivo o viceversa, nunca entre dispositivos. Cada milisegundo (1.00 ± 0.05 ms) se transmite una trama de sincronización de dispositivos.

- Las transacciones se denominan “tramas”.
- Las tramas están formadas por “paquetes”.
- Las tramas comienzan por SOF: *Start Of Frame*.

El bus USB reconoce cuatro clases de tramas:

- Control:** Configuran dispositivos, emisión de órdenes y revisión del estado.
- Isócronas:** Dispositivos de tiempo real, p.ej. altavoces. Se envían datos o reciben en tiempos precisos. No se retransmite en caso de error.
- Volumen:** Transferencias de datos de gran volumen, como impresoras. No requiere tiempo real.
- Interrupción:** USB no reconoce interrupciones, por lo tanto es necesario desencadenar escrutaciones a tiempos fijos para recoger informaciones pendientes.

Los paquetes se dividen en cuatro tipos:

- **Testigo:** Viajan del controlador al dispositivo. Son de control. Estos son SOF, IN (pide datos), OUT (indica que se envían datos) y SETUP (configuración).
- **Datos:** Paquetes DATA que transmiten hasta 64 bytes de información en ambos sentidos.
- **Saludo:** ACK (paquete anterior recibido correctamente), NAK (error en paquete – CRC), STALL (ocupado) y NYET.
- **Especiales:** PRE, ERR, SPLIT, PING, RESERVED.

La Figura 21 muestra el movimiento de información en un Bus USB.

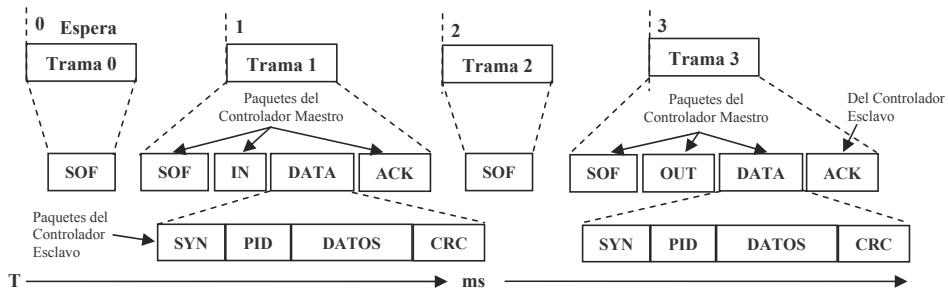


FIGURA 21. Secuencia de Datos en el Bus USB

SYN: Sincronización. 8 bits

PID: Packet Identifier Field. 8 bits; secuencia numérica que identifica un paquete.

CRC: Código de Redundancia Cíclica; código que determina si es correcto o no el paquete recibido.

5.5.2. FIREWIRE.

Bus serie creado por Apple Computer en 1986. Escogen el nombre debido a su velocidad. También se conoce como i.LINK en Japón (Sony añade prestaciones a firewire denominando al bus i.LINK). La primera especificación se completa en 1987. Pensado para conexión multimedia de bajo coste, gran ancho de banda y tiempo real.

Característica Plug and Play en caliente y conexión Peer-to-Peer (todos los participantes son Maestros).

Conexión entre ordenadores, periféricos y dispositivos de electrónica de consumo (cámaras de vídeo, cámaras digitales, impresoras, etc...). Una red

puede soportar 64 nodos (0 - 63). Conectando múltiples redes (Bridges), en una súper red, puede llegar a 1000 nodos (teóricamente).

En 1995 la organización IEEE adopta como norma las especificaciones del bus FireWire como IEEE 1394. Actualmente existe la organización 1394 TA compuesta por aproximadamente 170 compañías de todo el mundo (www.1394ta.org).

Evoluciona a 1394a (2000) y 1394b (2002-2008) "High-Performance Serial Bus" o Firewire 2.

- 1394 y 1394a: Rango de velocidades desde 100 - 200 - 400 Mbits/s sobre 4.5 m. Conector 4 pines
- 1394b: Rango de velocidades desde 800Mb/s - 1.2 - 1.6 - 3.2 Gb/s sobre 100 m. Con CAT-5 (par trenzado) conector 6 pines y fibra óptica plástica.

El flujo de datos entre nodos es segmentado en canales síncronos y asíncronos.

- Canal asíncrono:** El modo de transferencia asíncrono envía paquetes a intervalos de tiempo variables; esto requiere que el receptor devuelva una señal de reconocimiento del paquete recibido (*Ack*), de no ser así, éste sería vuelto a enviar. Está dedicado a Comandos, Funciones de Control y Ficheros.
- Canal síncrono:** El modo de transferencia síncrono envía paquetes a intervalos regulares; esto requiere que se envíe un mensaje de sincronización cada $125\mu s$. De esta manera no se necesita un acuse de recibo, permitiendo un ancho de banda fijo. Al no requerir de acuse de recibo, implica que los errores de transmisión no se consideran relevantes, por lo tanto es adecuado para la transmisión de Audio y Vídeo. Este modo de transferencia garantiza una tasa de transferencia de hasta el 80 % del ancho de banda.

La incorporación de buses paralelo de 32 y 64 bits ha permitido el aumento de velocidad en los buses serie, por ello la versión "b" admite llegar a los Gigabits.

Estos flujos de Datos son válidos para las tres versiones (1394-1394a-1394b). La versión 1394b incorpora el denominado Beta-Mode³.

El bus requiere de un arbitraje que cumple con los requisitos siguientes:

³Entre otras modificaciones, está la codificación 8B10B; esta codificación consiste en transformar cada cadena de 8 bits en una cadena de 10 bits antes de ser transmitida, teniendo en cuenta que no puede haber más de cinco ceros o cinco unos seguidos.

- El tiempo de bus se divide en ciclos de 125ms de periodo.
- El nodo raíz es el maestro de ciclo.
- Un ciclo se inicia con un paquete de inicio de ciclo que se difunde a todo el bus.
- Inmediatamente se inician las transacciones síncronas (tiempo dividido en canales síncronos). Un dispositivo síncrono debe estar autorizado por el manejador de recursos síncronos.
- Un dispositivo síncrono puede tener asignado uno o más canales síncronos.
- Los dispositivos que desean utilizar los recursos síncronos acuden al árbitro del bus → notifican a su nodo padre que quieren el bus.
- Los dispositivos más cercanos al nodo raíz conseguirán ganar el bus.
- Un dispositivo que haya ganado un canal síncrono no competirá hasta el próximo ciclo por los canales síncronos.
- Tras las transferencias síncronas se inician las transferencias asíncronas con un mecanismo similar (requieren de respuesta *Ack*).

La Figura 22 muestra un cronograma de transferencias síncronas. Si tenemos en cuenta que los dispositivos están orientados, principalmente a audio/vídeo, las tramas deberán alternar transferencias asíncronas y síncronas, p.ej., se desea que una cámara comience a grabar o reproducir; estas acciones requieren la emisión de la orden de grabar o reproducir y posteriormente transmitir las imágenes. Las órdenes serán de tipo asíncrono y la transmisión de imágenes de tipo síncrono. La Figura 23 muestra este tipo de transferencias.

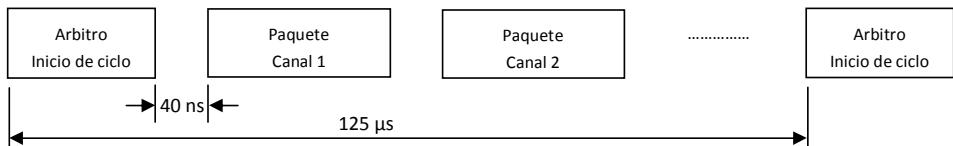


FIGURA 22. Firewire Bus: Transferencias síncronas

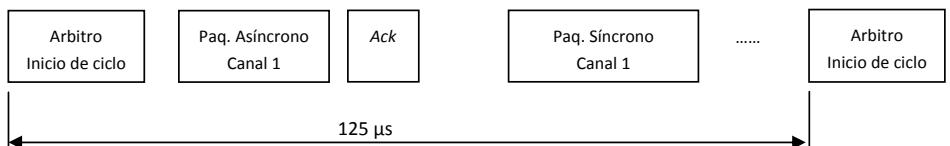


FIGURA 23. Firewire Bus: Transferencias asíncronas y síncronas

Como comentario final relativo a la piratería del mercado, para evitar las copias ilegales, se ha desarrollado un protocolo de protección denominado

Digital Transport for Content Protection (5C/DTCP) (creado por las cinco industrias más importantes). Presenta tres opciones:

- copiar una vez
- copiar libremente
- no copiar nunca

Capítulo 6

DISPOSITIVOS DE ALMACENAMIENTO SECUNDARIO

Cuando un hombre tiene voluntad y entusiasmo, los dioses son sus aliados. (Esquilo de Eleusis: 525 - 456 a.C)

RESUMEN. Los dispositivos de almacenamiento secundario se van a ocupar de guardar la información generada por un procesador y que no puede ser almacenada en la memoria principal por diferentes razones como: falta de espacio, seguridad de almacenamiento (no volatilidad), portabilidad, etc. Se analizarán diversos soportes y su tecnología.

6.1. INTRODUCCIÓN

Surgen dos necesidades, una consiste en tener almacenados todos los archivos de programas generados por el equipo de desarrollo y otra en poder disponer del conjunto de particiones que forman el programa para ser cargado a Memoria Principal cuando así sea requerido por el sistema gestor.

Aun cuando los avances de la tecnología permitan disponer de dispositivos con más capacidad de almacenamiento que sus antecesores, nunca podremos llegar (en determinadas aplicaciones) a poder almacenar la totalidad del programa en la memoria del computador. Es por ello que se hace necesario el disponer de dispositivos secundarios que permitan almacenar información de manera “masiva”.

6.2. DISCOS MAGNÉTICOS

6.2.1. Disco Duro (HD Hard Disk¹).

Consiste en uno o más platos de aluminio con un recubrimiento de características magnéticas. Originalmente estos platos medían 50 cm., hoy en día estamos en 3 cm. Con el avance de la ciencia en materia de electromagnetismo, esta dimensión irá disminuyendo. Una cabeza consistente en un entrehierro (u otro material magnético) bobinado, se sitúa sobre la superficie de la película magnética, no llega a tocarla por existir una fina lámina de aire que actúa como colchón (0.5 micras). La cabeza está situada en el extremo de un brazo mecánico que entra o sale de la superficie de manera radial.

La cabeza magnética polariza los dominios magnéticos de la superficie durante el proceso de escritura. Durante el proceso de lectura la cabeza presenta el proceso de inducción magnética a causa de los diferentes dominios magnéticos.

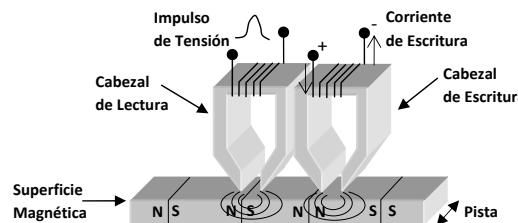


FIGURA 1. Cabezas de Lectura/Escritura

Como cada disco presenta dos caras magnéticas, se disponen dos brazos con sendas cabezas, aumentando la capacidad de almacenamiento el doble. La información está “serializada”.

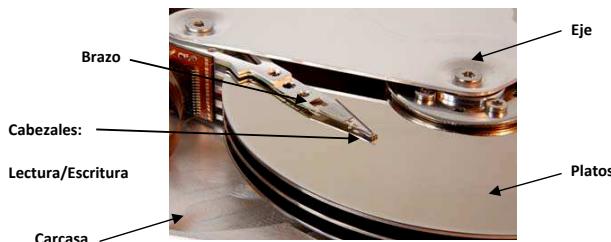


FIGURA 2. Aspecto físico de un Disco Duro

¹Tambien HDD: Hard Disk Drive

La estructuración de un disco es la siguiente:

- Una rotación completa del disco, crea una circunferencia imaginaria, se denomina “pista”. Se trata de una línea magnetizada.
- Una pista está dividida en “sectores” de longitud fija; normalmente son de un tamaño de 512 bytes (4096 bits).
- Cada sector está precedido de un “preámbulo” de bits que permiten sincronizar la cabeza antes de realizar las operaciones de lectura o escritura.
- Al final de cada sector hay un campo “corrección de errores” (ECC: *Error Correcting Code*). Estos códigos pueden ser Haming o Reed-Solomon.
- Entre sectores hay una pequeña separación. No contiene información.

La Figura 3 muestra la estructura comentada.

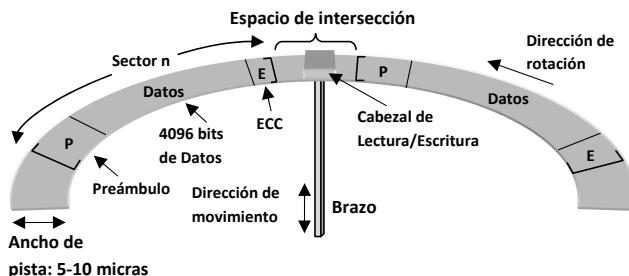


FIGURA 3. Información sobre el Disco

La práctica totalidad de discos están construidos agrupando más de un plato sobre el mismo eje, cada superficie tiene su propio brazo y cabeza magnética. Todos los brazos se mueven solidariamente.

El conjunto de todas las pistas, en una posición radial dada, se denomina “cilindro”.

La dirección de un sector es: nº de cabeza – nº de cilindro – nº de sector

El mecanismo de escritura/lectura de un sector, presenta dos aspectos:

- Movimiento del brazo a una posición radial determinada, esto se denomina “búsqueda” ($SEEK \Rightarrow T_{seek}$). Entre 5 y 15 ms. entre pistas (medición al azar). Hay mediciones en el orden de 1 ms.
- Giro de la pista hasta que el sector deseado queda debajo de la cabeza, esto se denomina “latencia rotacional”. Entre 4 y 8 ms la rotación.

La Figura 4 muestra un ejemplo de la disposición de los platos y cabezas magnéticas.

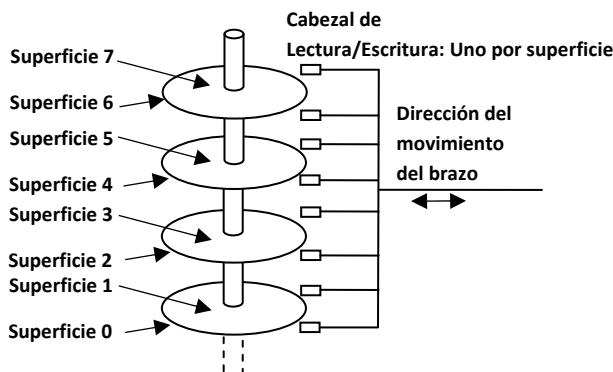


FIGURA 4. Esquema de Platos y Cabezas en un Disco Duro

Cuando vamos a almacenar un archivo, dependiendo de su tamaño, este podrá caber en unidades de almacenamiento consecutivo pero si hay unidades ocupadas anteriormente, tendrá que fragmentarse el archivo y ubicar los fragmentos en diferentes zonas o “clusters”. Esta es una labor que la realiza el controlador. Un disco, con el tiempo, presenta problemas de fragmentación, lo que lleva a aumentar el tiempo de búsqueda de la información. Se hace necesario el uso de “desfragmentadores” que vuelvan a ubicar archivos en áreas contiguas.

Como un disco inicialmente es una superficie de material magnético, es preciso dar un formato a esta superficie, de esta manera se divide el disco en sectores y pistas y se descubren zonas con errores superficiales. Toda esta información se guarda en un archivo denominado FAT (*File Allocation Table*) que es la Tabla de Asignación de Archivos, ubicado en el sector cero. El Sistema Operativo almacena la estructura del directorio del disco, el nombre de los ficheros, su tamaño, la fecha y hora de su última modificación, los atributos de los ficheros y los “clusters” que se han utilizado para almacenar los archivos.

Cada unidad de disco contiene un dispositivo denominado “controlador de disco”. El procesador del computador se comunica con el controlador.

La velocidad de giro está entre los siguientes valores: 3600, 5400, 7200 y 10800 r.p.m. Afecta a todos los tiempos apuntados anteriormente: T_{seek} , latencia rotacional y la velocidad de transferencia de datos una vez localizados.

La tasa de transferencia típica está entre 5 y 20 MB/s.

La Tabla 1 da una idea acerca del tiempo invertido en dar una vuelta completa, es decir, en encontrar el mismo punto en la misma pista.

TABLA 1. Tiempo de una rotación

r.p.m	rotación: ms
3600	16,6
5400	11,1
7200	8,3
10800	5,5

6.2.2. Discos flexibles (FD *Floppy Disk*).

Su aparición es determinante² en cuanto la distribución del software. La necesidad de “portar” el software lleva a la industria a crear este tipo de soporte. IBM los introduce y en sus comienzos son “flexibles” puesto que el disco magnético está realizado sobre un soporte de poliéster, además la protección es una funda de cartón. Las primeras unidades tenían un tamaño de 8” y de simple densidad, podían almacenar 128 KB.. La evolución de las tecnologías nos lleva a tamaños de 3.5” (y menores) con capacidades de 1.44 MB. La fundas están realizadas en plástico rígido.

La velocidad de giro es de 360 r.p.m. Consta de 80 cilindros y 18 sectores. La Tasa de Transferencia es de 54 KB/s.

La diferencia fundamental con los discos duros, consiste en que las cabezas magnéticas están en contacto con la superficie, esto conlleva un desgaste de cabezas y de la superficie magnética. Para reducir este impacto, se retraen las cabezas cuando no hay lectura/escritura, deteniendo también la rotación del disco. Esto provoca retrasos considerables cuando se emite la orden de lectura/escritura ya que primero hay que poner en rotación al disco y posteriormente realizar la búsqueda.

$T_{promedio\ búsqueda}(T_{seek})$ es aproximadamente de 95 ms y la Latencia rotacional de 83 ms.

Dado el estado actual de la tecnología de estado sólido y su miniaturización, estos dispositivos han caído en desuso en favor de los dispositivos denominados “pendrive” (ver 6.8) de capacidades muy superiores y tamaño muy inferior.

6.2.3. Discos IDE.

Este tipo de disco duro surge a partir de los ordenadores personales (PC). En los PC XT, se introdujo un disco Seagate de 10MB controlado por un dispositivo Xebec dispuesto sobre una tarjeta insertable en una de las

²Hoy en día, podríamos decir que “fue determinante” dada su desaparición de los escenarios actuales.

ranuras del Bus del PC, permitiéndose controlar 2 unidades de disco. La evolución consistió en disponer de un controlador ubicado en la electrónica del disco (solidario a él), denominándose a estas unidades IDE (*Integrated Device Electronics*). La máxima capacidad estaba dada por:

- 16 cabezas
- 63 sectores
- 1024 cilindros

Esto permite 1.032.192 sectores lo que limita a 528 MB la capacidad máxima. Debido a las anteriores limitaciones impuestas por el sistema BIOS (*Basic Input Output System*) de los PCs, pronto se pasó a realizar una extensión de este tipo de unidades denominándose EIDE (*Extended IDE*), permitiéndose referenciar hasta $2^{24} - 1$ sectores, ampliándose la capacidad hasta 8 GB y pudiéndose controlar hasta 4 unidades de disco.

6.2.4. Discos SCSI.

Este tipo de discos presentan la misma estructura que los anteriores en cuanto a nº de platos, pistas, cilindros, etc., la diferencia consiste en el interfaz de transferencia de datos. Presentan una tasa de transferencia mucho más alta que los IDE.. SCSI significa *Small Computer System Interface*, el nombre y estándar definitivo se establece en 1986 por el comité ANSI, efectuándose una revisión en 1994 denominándose SCSI-2, actualmente estamos en el SCSI-3.

La Tabla 2 nos muestra las potencialidades de las diferentes actualizaciones:

TABLA 2. Evolución del Bus SCSI

Nombre		Nº Bits Datos	Transf. MB/s
SCSI-1	(conector de 25 pines)	8	5
SCSI-2	(conector de 50 pines)	8	5
SCSI-2	Rápido	8	10
SCSI-2	Rápido y Ancho	16	20
SCSI-2		16	40
SCSI-2	Ultra Rápido y Ancho	16	80

El estándar SCSI ha extendido su uso a otro tipo de periféricos, estos pueden ser, p. ej., CD-ROM, Impresoras, Scaners, etc.

El nº de dispositivos que pueden conectarse en un Bus SCSI es de 8, se referencian de 0 a 7 en la cadena. En el SCSI ancho se llega a 15 periféricos.

Los dispositivos se conectan siguiendo una cadena, el 2º al 1º, el 3º al 2º, etc., los identificadores no tienen porqué ser consecutivos. Disponen, por lo tanto, de un conector de entrada y uno de salida (siguiente dispositivo). El último dispositivo debe llevar un elemento terminador, se trata de una red de resistencias de 50Ω , que evita las reflexiones eléctricas en los cables.

Este tipo de conexión en discos ha sido normalizado por Macintosh, HP y Sun, actualmente hay más compañías que lo han ido incorporando.

El interfaz SCSI de 8 bits de Datos (mas común), requiere un cable de 50 hilos, de estos, 8 son bits de Datos, 1 bit de Paridad, 9 bits de Control el resto para alimentación y expansiones futuras.

6.2.5. Discos RAID.

Más que un tipo de disco consiste en una organización distinta de los discos. Este tipo de organización surge por la confluencia de dos necesidades:

1. Aumentar la tasa de transferencia de datos entre el procesador y el disco. Los procesadores aumentaban su rendimiento pero los discos sufrían un cierto estancamiento.
2. Disminuir la tasa de errores y mejorar su recuperación. El fallo de un disco podía provocar la pérdida de datos definitiva.

Es por esto por lo que se idea una nueva estructura de organización de los discos, denominándose RAID (*Redundant Array Inexpensive Disks*); si la idea inicial también era el proveer discos económicos, pronto la industria cambió la “I” por Independientes, de tal manera que en una caja se incluyen una serie de discos independientes físicamente pero que van a presentar una organización entre ellos, de tal manera que el usuario o el computador lo ve como un solo disco. Su aplicación se da en grandes servidores o sistemas de control críticos. Esta estructuración puede presentar las siguientes configuraciones:

- Se divide la información en bloques de tal manera que se graban en discos independientes y después son leídos en paralelo.
- Se duplica la información, grabándose en paralelo.
- Se generan bits de control (paridad) para recuperación de errores, según diferentes organizaciones.

Combinando estos modos de operación se establecen seis niveles de discos RAID, que se denominan RAID nivel 0, ..., RAID nivel 5. No establece una jerarquía, simplemente es una organización.

- RAID nivel 0: Disco dividido en tiras (*strips*) de K sectores cada una. De 0 a $K - 1$ en la primera tira (1er. disco), de K a $2K - 1$ en la segunda (2º disco), etc.. La distribución de datos entre varias unidades se denomina *striping* (multiplexaje de datos). Si hay un comando de lectura de un bloque que ocupa varias tiras consecutivas (varios discos), el controlador del RAID genera procesos de lectura simultánea a los discos, entregando la información en bloque sin retrasos. Muy eficiente cuando se le piden grandes bloques de información.

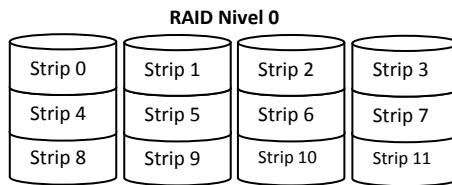


FIGURA 5. Raid nivel 0

- RAID nivel 1: Trabaja con tiras. RAID puro. Duplica la información. Muy alta tolerancia a fallos. El rendimiento en lectura es el doble, puesto que puede acceder a los dos conjuntos de discos. Ante fallo de disco, solamente hay que substituir el disco dañado.

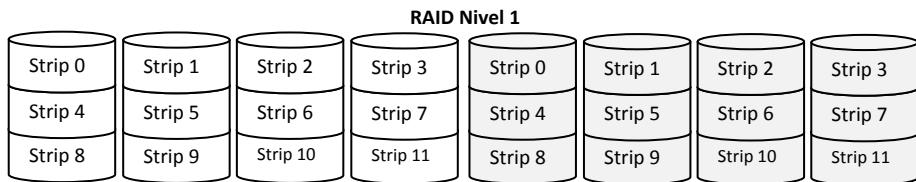


FIGURA 6. Raid nivel 1

- RAID nivel 2: Trabaja con palabras y con bytes. Se divide la información en *nibbles* y se introducen 3 bits de paridad código Hamming (bits 1, 2 y 4). Las unidades de disco están sincronizadas.

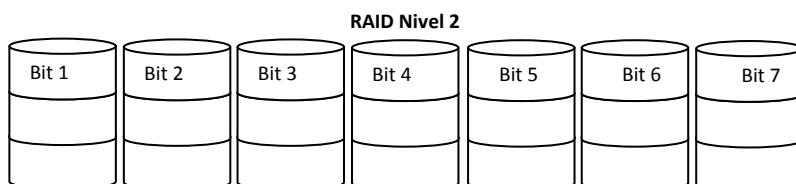


FIGURA 7. Raid nivel 2

- RAID nivel 3: Trabaja con palabras. Simplifica el nivel 2, introduciendo un solo bit de paridad. Las unidades de disco están sincronizadas.

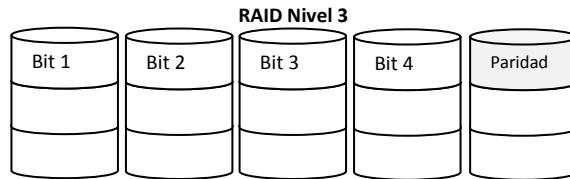


FIGURA 8. Raid nivel 3

- RAID nivel 4: Trabaja con tiras. Parecido a nivel 0 pero introduciendo paridad por tira. Se calcula el “OR exclusivo” de los bytes que componen todas las tiras. Protegido contra pérdidas pero de rendimiento bajo ante actualizaciones. Gran carga sobre el disco de paridades. Las unidades de disco no están sincronizadas.

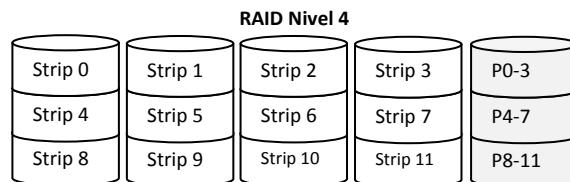


FIGURA 9. Raid nivel 4

- RAID nivel 5: Trabaja con tiras. Distribuye los bits de paridad entre todas las unidades. Resuelve la carga de los bits de paridad. Si una unidad falla, su reconstrucción es un proceso complejo. Las unidades de disco no están sincronizadas.

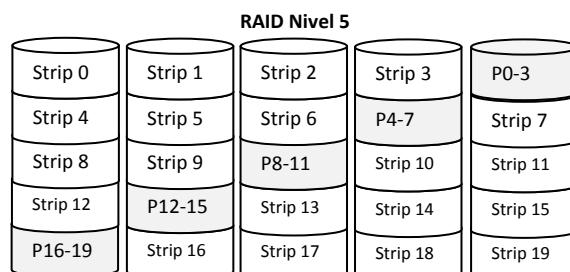


FIGURA 10. Raid nivel 5

6.3. CD-ROM

Compact Disk Read Only Memory. Es el primer medio de almacenamiento masivo no magnético, está basado en tecnología óptica láser. Su densidad de grabación es mucho más alta que la magnética. Irrumpen en el mercado en 1980. Los inventores son Philips y Sony. Su diámetro es de 120 mm. Y su espesor de 1.2 mm.

La escritura se realiza de la siguiente forma, un láser infrarrojo de alta potencia quema orificios de 0.8 micras de diámetro en un disco maestro recubierto de vidrio. Con este disco maestro se genera un molde, presentando salientes donde han estado los orificios creados por el láser. Posteriormente se inyecta resina de policarbonato fundida para formar el CD, creándose el mismo patrón de orificios que el máster de vidrio, finalmente se deposita una fina capa de aluminio reflectante sobre el policarbonato, seguido de una capa de resina protectora.

Las depresiones en el substrato se denominan “fosos” (*pits*) y el área entre fosos “planicies” (*lands*). La escritura se realiza en espiral desde el agujero central hacia el exterior hasta 3.2 mm del borde. La espiral describe 22,188 revoluciones alrededor del disco (aprox. 600 por mm), su longitud total llegaría a 5.6 Km.

La reproducción se realiza moviendo la cabeza láser del interior hacia el exterior, en el interior la velocidad de rotación es de 530 rpm y en el exterior de 200 rpm, esto hace que la velocidad lineal sea constante y de valor 120 cm/s.

Un láser infrarrojo de baja potencia hace incidir su luz sobre los fosos y planicies, los fosos tienen una profundidad de $\frac{1}{4}$ de la longitud de onda de la luz láser, por lo que al ser reflejada, esta luz estará desfasada en $\frac{1}{2}$ de longitud de onda respecto a la luz reflejada circundante por lo que se produce una interferencia destructiva y llega menos luz al receptor, es así como se diferencia un foso de un planicies. Transición es “1”, no transición “0”. La Figura 11 muestra la pista espiral así como los “*pits*” y “*lands*”.

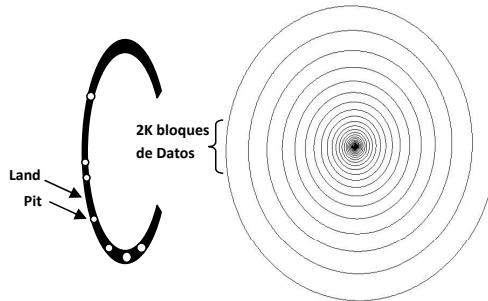


FIGURA 11. Pista de un CDROM. “Pits” y “Lands”

La codificación es como sigue:

- Cada byte se codifica sobre 14 bits (código de corrección de error) denominándose “símbolo”.
- La agrupación de 42 símbolos forma un “cuadro” (588 bits).
- La agrupación de 98 cuadros forma un “sector” de cd-rom.
- Cada sector comienza con un “preámbulo” de 16 bytes. 12 bytes indican comienzo de sector, los 3 bytes siguientes el nº de sector, el último byte se denomina “modo” (se definen dos modos en el Libro Amarillo). El modo 1 es el que se ha comentado.

Esto nos lleva a 7203 bytes para una información útil de 2048 bytes. Eficiencia del 28 %. La corrección de errores se realiza siguiendo el siguiente esquema:

- Los errores de un solo bit se corregen en el nivel inferior.
- Los errores de ráfaga corta se corregen a nivel de cuadro.
- Resto de errores residuales se corregen a nivel de sector.

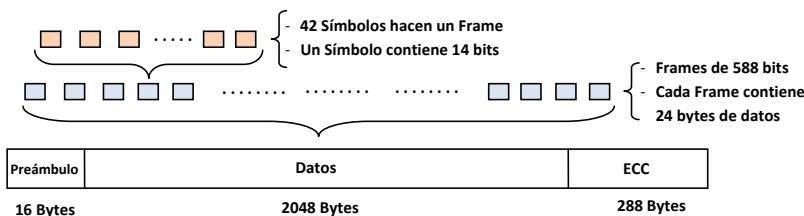


FIGURA 12. Empaquetamiento de Datos en un CDROM

6.3.1. Grabables.

Los CD-R son parecidos a los CD-ROM, son discos de policarbonato blanco de 120 mm de diámetro. Se utiliza oro en lugar del aluminio para crear la capa reflectante. Este tipo de CD no tiene fosos por lo que hay que simularlos, para que exista una diferencia de reflectividad entre foso y planicie. Esto se realiza añadiendo una capa de colorante (cianina o ftalocianina) entre el policarbonato y el oro. En el estado inicial la capa colorante es transparente y permite el paso del haz de luz y su reflexión en la capa de oro.

Para realizar la escritura, se aumenta la potencia del láser entre 8 y 16 mW. Cuando el haz incide sobre un punto con colorante, este se calienta y rompe uno de sus enlaces químicos. Este cambio en la estructura molecular crea un punto oscuro.

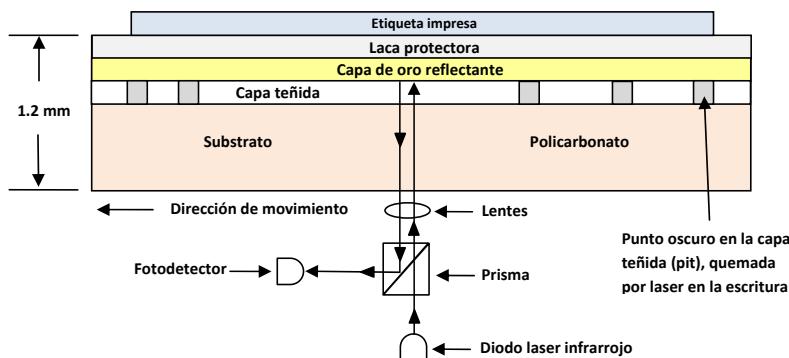


FIGURA 13. Estructura de un CDROM grabable

Durante la reproducción (potencia de 0.5 mW), el fotodetector percibe una diferencia entre los puntos oscuros (quemados) y las áreas transparentes (no quemadas). Interpretando esto como una transición entre foso y planicie.

Una evolución de este sistema es el permitir grabar por sesiones, no teniendo que grabar el CD completamente la primera vez que queramos realizar una grabación.

6.3.2. Regrables.

Los CD-RW cambian el colorante de los CD-R por una aleación de plata-indio-antimonio-telurio. Esta aleación presenta dos estados estables: cristalino y amorfo con diferente reflectividad. Se emplean tres potencias distintas en el haz del láser:

- Potencia alta: Funde la aleación, pasando de estado cristalino (alta reflectividad) a estado amorfo (baja reflectividad). Representa “foso”.
- Potencia media: Funde la aleación recristalizando a su estado natural. Representa “planicie”.
- Potencia baja: Estado de lectura, no altera la estructura cristalina del material.

6.4. DVD

Se trata del Vídeo Disco Digital (*Digital Vídeo Disk*). Actualmente se denomina Digital Versátil Disk. Existen dos tamaños reconocidos, uno de 8 cm y otro de 12 cm. La diferencia con los CD-ROM es la siguiente:

- Fosos de 0.4 micras.
- Espiral más cerrada (0.74 micras entre pistas en vez de 1.6 micras).
- Láser rojo de 0.65 micras en vez de 0.78 (puede presentar problemas de compatibilidad con los lectores CD-ROM antiguos).

Esto multiplica la capacidad por siete, pasando a 4.7 GB.

Se han definido cuatro formatos:

- Un solo lado, una sola capa (4.7 GB).
- Un solo lado, capa dual (8.5 GB).
- Dos lados, una sola capa (9.4 GB).
- Dos lados, capa dual (17 GB) (8.5 horas de grabación como película).

La tecnología “dual” dispone de una capa semirreflectante y una capa reflectante, según la distancia focal del láser, trabaja con una o con otra. La Figura 14 muestra la estructura de un DVD.

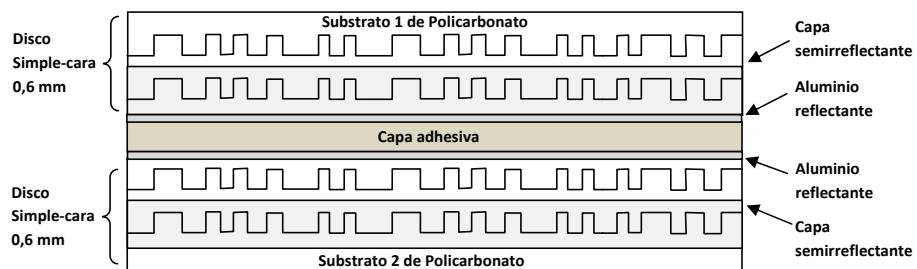


FIGURA 14. Estructura de un DVD

6.5. DISCOS MAGNETO-ÓPTICOS

Este tipo de discos presenta un método diferente en Lectura/Escritura con relación a los Discos Duros. En escritura, un haz de luz láser (potencia alta) calienta la superficie del disco (temperatura Curie aproximadamente 200°C) que está recubierta de un material cristalino, liberando cristales que pueden desplazarse por medio del campo magnético. Una cabeza de L/E parecida a la cabeza de los discos magnéticos, escribe los datos, cambia la dirección (polarización) de los cristales. La magnetización cambia el comportamiento óptico (efecto Kerr). Los puntos representan ceros o unos según su polaridad.

La ventaja de este procedimiento radica en que la zona magnetizada se reduce únicamente a la parte calentada por el láser. De esta manera se pueden

obtener discos de más de 1 GB por cada cara (1.2 – 9.1 GB). Los lectores suelen incorporar una única cabeza y hay que dar la vuelta al disco. La Figura 15(a) muestra la situación inicial en un disco no grabado y la Figura 15(b) muestra la operación de escritura.

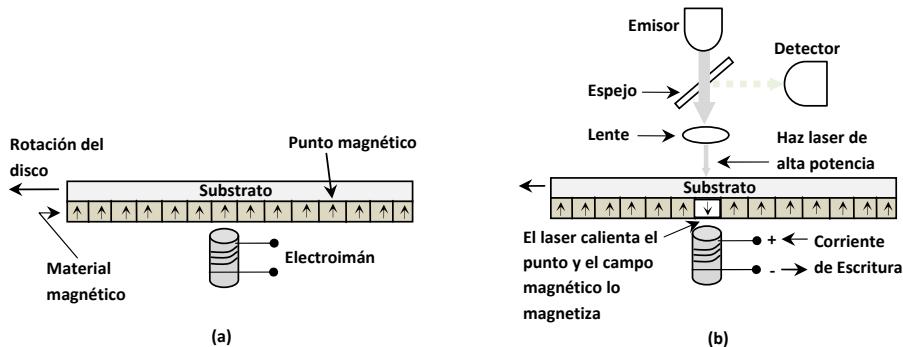


FIGURA 15. a) Disco no Grabado, b) Operación de Escritura

El proceso de lectura requiere proyectar un haz de luz láser de baja potencia, donde la diferente orientación de los cristales creará dos diferentes intensidades de reflejo de luz. La operación de borrado requiere la proyección del haz láser de alta potencia y suministrar al electroimán una corriente que invierta el campo magnético. La Figura 16(a) muestra la situación la operación de lectura y la Figura 16(b) muestra la operación de borrado.

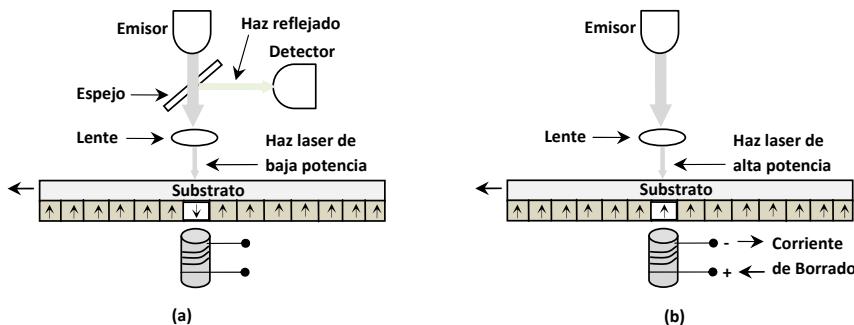


FIGURA 16. a) Operación de Lectura, b) Operación de Borrado

Útiles tanto para copias de seguridad como para intercambio de datos.

6.6. DISCOS BLU-RAY

En el año 2005 se produce el lanzamiento de un nuevo soporte denominado “*Blu-ray*”. El término *Blu-ray* proviene de “*Blue*: azul” y “*Ray*: rayo”, es decir, “rayo azul” atendiendo a la tecnología utilizada en el diodo laser de emisión en el espectro azul. Se trata de un formato de disco óptico de nueva generación de 12 cm de diámetro (compatible en tamaño, por tanto, con los CD y los DVD) para vídeo de alta definición y almacenamiento de datos de alta densidad. Siendo superior en 5 veces al DVD. Su capacidad de almacenamiento llega a 25 GB por capa (doble capa: 50GB), aunque Sony y Panasonic han desarrollado un nuevo índice de evaluación (i-MLSE) que permitiría ampliar un 33% la cantidad de datos almacenados, desde 25 a 33,4 GB por capa.

El formato de video soportado por los discos *Blu-ray* es de 1920x1080 (1080p), siendo el de los DVD de 720x480 (NTSC) o 720x576 (PAL).

Su desarrollo, en el 2002, parte de un consorcio de compañías tecnológicas denominado *Blu-Ray Disc Association* (BDA), liderado por dos empresas del ámbito electrónico-informático como son Sony y Philips (otras empresas del mismo ámbito son: Apple, Dell (se une en el 2004), Hitachi³, HP, JVC, LG, Mitsubishi, Panasonic, Pioneer⁴, Samsung, Sharp, TDK⁵ y Thomson). Además, dentro de este consorcio se encuentran empresas del mundo cinematográfico como:

1. Estudios en exclusiva

- Sony Pictures Entertainment (Columbia Pictures y TriStar Pictures, entre otros).
- Buena Vista (Walt Disney Pictures, Touchstone Pictures, Yasser Entertainment, Hollywood Pictures y Miramax, entre otros).
- 20th Century Fox (incluye el catálogo de Metro-Goldwyn-Mayer y United Artists).
- Lions Gate Films.
- Warner Bros. Pictures.
- New Line Cinema.

2. Estudios colaboradores

- StudioCanal.
- Paramount Pictures (sólo para las películas dirigidas por un determinado director).

³Desarrolla un disco de 100 GB compatible con los lectores actuales que requiere una actualización del *firmware*

⁴Desarrolla un disco de 500 GB de 20 capas no compatible con los lectores actuales

⁵Desarrolla un disco de 100 GB de 4 capas no compatible con los lectores actuales

- Filmax.
- Mar Studio.

La funcionalidad de los discos *Blu-ray* se basa en un haz de luz láser de color azul con una longitud de onda de 405 nanómetros, a diferencia del láser rojo utilizado en lectores de DVD, que tiene una longitud de onda de 650 nanómetros (0,65 micras). Esto, junto con otros avances tecnológicos, permite almacenar sustancialmente más información que el DVD en un disco de las mismas dimensiones y aspecto externo.

El tamaño del punto mínimo en el que un láser puede ser enfocado, está limitado por la difracción, y depende de la longitud de onda del haz de luz y de la apertura numérica de la lente utilizada para enfocarlo. En el caso del láser azul-violeta utilizado en los discos *Blu-ray*, la longitud de onda es menor con respecto a tecnologías anteriores, aumentando por lo tanto la apertura numérica (0,85, comparado con 0,6 para DVD). Con ello, y gracias a un sistema de lentes duales y a una cubierta protectora más delgada, el rayo láser puede enfocar de forma mucho más precisa en la superficie del disco. Dicho de otra forma, los puntos de información legibles en el disco son mucho más pequeños y, por tanto, el mismo espacio puede contener mucha más información. Esto ha permitido incorporar un sistema mejorado de codificación de datos que aumenta el empaquetamiento de la información.

Ventajas adicionales del disco *Blu-ray* frente al DVD son:

1. El disco *Blu-ray* tiene una capa de sólo 0,1 mm lo que evita los problemas de difracción presentes en el DVD.
2. Gracias a la cercanía de la lente y la rápida convergencia del láser la distorsión es inferior, evitándose posibles errores de lectura.

La velocidad de transferencia se muestra en la Tabla 3.

TABLA 3. Velocidad de transferencia de discos *Blu-ray*

Velocidad de la unidad	Velocidad de trasferencia	Tiempo teórico de escritura (minutos)	
		MB/s	Una capa
1×	4,5	90	180
2×	9	45	90
4×	18	22,5	45
6×	27	15	30
8×	36	11,25	22,5
12×	54	7,5	15

6.7. CINTAS MAGNÉTICAS

Son dispositivos de Lectura/Escritura de acceso secuencial. Especialmente útil cuando la información a almacenar es considerable. Especialmente recomendadas en salvaguarda de Sistemas completos. Dentro del abanico de cintas podemos destacar las tres siguientes: QIC, DAT y DLT.

La información está codificada digitalmente. La estructura del cabezal/cinta se refleja en la Figura 17. La velocidad y anchura corresponden al tipo de cinta QIC.

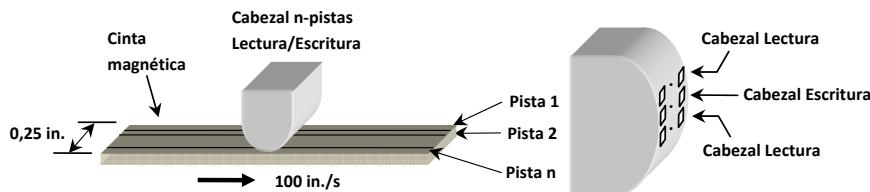


FIGURA 17. Cabezal y Pistas en una Cinta Magnética

Las tecnologías actuales permiten disponer de hasta 36 cabezas de L/E en un mismo cabezal. El nº de pistas puede llegar a 144 según tecnologías.

Las cintas QIC (*Quarter-inch Cartdrige*) de 5,25" y 3,5", son parecidas a un casete de audio, variando el nº de pistas entre 36 y 72, pudiendo almacenarse desde 40 Mbytes, 1,2 Gbytes, 4 Gbytes hasta 25 Gbytes. En la Figura 17 se observa que el cabezal consta de una cabeza de escritura y dos cabezas de lectura a cada lado de la de escritura. Introducida por 3M en 1972.

Una de las cintas de menor tamaño es la denominada DAT (*Digital Audio Tape*), fue introducida para el almacenamiento de audio pero el comité ISO la adaptó al estándar DDT (*Digital Data Tape*) pero se sigue conociendo por DAT. Su capacidad de almacenamiento está entre los valores siguientes: 2,6 GB - 4 GB. - 8 GB. - 20 GB. y 36 GB (o 72 si comprimido). Se esperan evoluciones. Esta es una cinta muy extendida e introducida por Sony en 1987.

Las cintas DLT (*Digital Linear Tape*) por su tecnología constructiva aporta la mayor capacidad de almacenamiento, según tecnologías estamos entre 35 Gbytes y 500 Gbytes (recientemente 1200 Gbytes). Fueron inventadas por Digital Equipment Corporation.

6.8. PENDRIVES

Fundamentalmente se trata de agrupaciones de memorias FLASH (vease Sección 4.4 apartado 2a2) controladas por un dispositivo que se comunica con el computador a través del bus USB (más adelante se verá el protocolo USB en la Subsubsección 5.5.1). Por lo tanto, se trata de una unidad Flash USB.

Los inicios presentaban capacidades de almacenamiento comprendidas entre 8 MBytes y 256 Mbytes. Actualmente las capacidades oscilan, sin ser exhaustivos ya que la evolución es muy rápida, entre 1 Gbyte y 256 Gbytes dentro de una oferta comercial razonable.

También hay diferencias entre los tiempos de acceso y la versión de protocolo USB incorporado en el controlador del pendrive. Los últimos modelos son compatibles con el formato USB 3.0 y 3.1.

La información puede permanecer del orden de 20 años y pueden ser reescritas, estas unidades, del orden de varios millones de veces. Sin olvidar que todo está sujeto a la evolución tecnológica de las memorias Flash.

6.9. DISCOS DE ESTADO SÓLIDO

Los Discos de Estado Sólido (SSD⁶) están llamados a substituir a los Discos Magnéticos (HDD) cuando es requerida una velocidad de acceso superior (p.ej., un sistema industrial que requiere un arranque rápido) y/o el sistema donde se va a ubicar el SSD, puede estar sujeto a vibraciones. Su tecnología es la ya mencionada FLASH (vease Sección 4.4 apartado 2a2). Cuentan con la funcionalidad de Código de Corrección de Errores ECC (Error Correction Code) que puede detectar y corregir errores, p.ej., 4 bits en bloques de 512 bytes; si no es recuperable, se marca el bloque como defectuoso.

Actualmente hay discos de diferentes capacidades, reflejándose estas en la Tabla 4.

TABLA 4. Capacidades de los SSD - Ejemplos

Rango	Capacidad
MB	128 - 256 - 512
GB	1 - 2 - 4 - 8 - 16 - 32 - 64 - 128 - 256 - 512
TB	1 - 2

⁶Solid State Drive

Hay otras capacidades como: 60 - 120 - 192 - 250 - 480 - 500 GB.

El costo es superior a los HDD pero la decisión está, fundamentalmente, en qué velocidad de respuesta se necesita en el sistema de computación y en el entorno donde va a trabajar el sistema.

La conexión se realiza bajo las siguientes interfaces: SATA II, SATA III, mSATA (mini-SATA), USB, PCI y PATA (*Parallel ATA*).

El estandar más utilizado es SATA III, que permite una tasa de transferencia de 6 Gbits/s (SATA II de 3 Gbits/s).

Los discos SSD de tipo externo, presentan una interface USB. Las versiones del protocolo USB son 3.0 y 3.1.

- **SSD-SATA III:** La Tabla 5 muestra diversos tiempos de acceso de los discos internos SSD con interface SATA III. Se ha recurrido a datos de diversos fabricantes, agrupándose según capacidad. Se muestran capacidades más significativas.

TABLA 5. Velocidades de acceso en un SSD-SATA III

Capacidad	L/E Secuencial (1) en MB/s
64 GB	450/80 - 534/450 - 520/80 - 460/160
128 GB	34/450 - 550/170 - 534/471 - 560/160 - 550/470 - 460/160
256 GB	560/460 - 534/450 - 560/320 - 534/482 - 550/520
512 GB	550/520 - 560/460
1 TB	530/510 - 560/460 - 550/520 - 560/520

(1) Si los accesos son de tipo aleatorio, presentan una diversidad elevada y un notable aumento del tiempo de acceso; un ejemplo tomado al azar puede ser el siguiente: 92K/83/K accesos por segundo.

Obsérvese la diversidad de velocidades de transferencia en L/E, relacionadas con la tecnología del fabricante. Un aspecto interesante es que los dispositivos con mayor capacidad de almacenamiento, presentan mejores tasas de transferencia.

La Figura 18 muestra un esquema de bloques⁷ de este tipo de memorias.

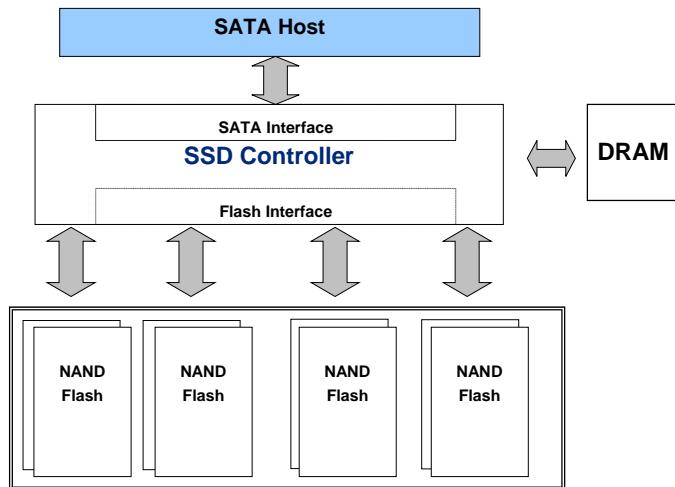


FIGURA 18. SSD-SATA III: Esquema de bloques

⁷Compañía **Trascend**

- **SSD-USB:** La Tabla 6 muestra cuatro ejemplos de SSD-USB⁸:

TABLA 6. Velocidades de acceso en un SSD-USB

Capacidad	USB	L/E Secuencial (1) en MB/s
128 GB	3.0	480/480
256 GB	3.1	450/450
512 GB	3.0	450/245
2 TB	3.1	450/450

(1) Idem accesos aleatorios SATA III

- **SSD-PCI:** La Tabla 7 muestra dos ejemplos de SSD-PCI (PCIe 3,0 × 4)⁹:

TABLA 7. Velocidades de acceso en un SSD-PCI

Capacidad	L/E Secuencial (1) en MB/s
450 GB	1200/600
2 TB	3500/2100

(1) Idem accesos aleatorios SATA III

El coste de la unidad de 2 TB es muy elevado. Hay capacidades menores como: 512 GB y 1 TB.

- **SSD-PATA:** La Tabla 8 muestra varios ejemplos de SSD-PATA:

TABLA 8. Velocidades de acceso en un SSD-PATA

Capacidad	L/E Secuencial (1) en MB/s
8 - 256 MB	90/90
128 MB - 1 GB	20/10
1 - 4 GB	40/20
8 GB	40/28

(1) Idem accesos aleatorios SATA III

El interface PATA (*Parallel ATA*) es la primitiva norma ATA (*Advanced Technology Attachment*) para conexión de discos en PCs, a la que se añade la “P” para indicar que es una conexión en paralelo. Esta conexión también es conocida como IDE (ver 6.2.3).

Sin ser exhaustivos, el interface PATA admite transferencias de datos según, principalmente, tres modos: PIO¹⁰ (modos: 0-4)

⁸Vease la Subsección 5.5.1 donde se trata el Bus USB

⁹Vease la Subsección 5.3.2 donde se trata el Bus PCI.

¹⁰*Programmed Input/Output*

avanzado, Multi-Word DMA (modos: 0-2) y Ultra DMA (modos: 0-4). Todos los modos van de mayor a menor tiempo entre accesos.

La Figura 19 muestra el esquema de bloques de una tarjeta SSD-PATA¹¹.

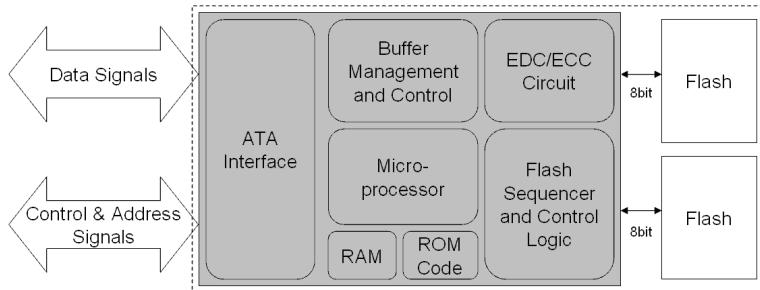


FIGURA 19. SSD-PATA: Esquema de bloques

El aspecto físico exterior, de un disco SSD-SATA, es idéntico a un HD ya que uno de sus fines inmediatos, es substituir a los HDD y físicamente deben ubicarse en el mismo lugar que el HDD. El consumo energético, estando activo, es bajo: entre 2 y 5 W (según capacidades). En estado de reposo del orden de 120 mW (mSATA: 60 mW).

Dentro de los fabricantes de discos SSD, podemos citar a los siguientes: Samsung, Transcend, Crucial, Toshiba, Kingston, SK Hynix, Freecom, Lexar, InnoDisk, etc.

Si nos atenemos a los fabricantes de discos SSD-USB (externos) son, fundamentalmente: Samsung, Freecom y Lexar.

¹¹Fabricante **innodisk**

Capítulo 7

DISPOSITIVOS DE E/S

Sócrates, al mirar las mercancías en el mercado, exclamaba: ¡Cuantas cosas hay que no necesito! (Diógenes Laercio: 180 - 240 d.C)

RESUMEN. La información puede ser introducida por un usuario, a través de un medio determinado, o bien debe ser visualizada sobre algún medio inteligible. Estos medios constituyen los dispositivos de entrada/salida.

7.1. INTRODUCCIÓN

En este capítulo se va a tratar de la interacción del usuario del computador con la información generada por el computador. El computador va a generar una información para el usuario que será inteligible solo si se dispone del periférico adecuado.

El usuario podrá introducir cierta información requerida por el computador a través de determinados dispositivos. Se trata de la relación Hombre-Máquina (Man-Machine Interface).

Esta información generada por la CPU del procesador o bien generada por el usuario para ser consumida por la CPU, necesita ser transportada por una serie de caminos denominados BUSES.

7.2. TERMINALES

Se denomina terminal al equipo que permite la relación de un operador con el Computador Central. Se trata del Interface entre el Hombre y la Máquina. Un terminal de ordenador consta de dos partes, teclado y monitor. En muchas ocasiones son dos elementos solidarios. La conexión se realiza vía serie bien directamente bien línea telefónica.

En el mundo del PC son dos elementos separados e independientes.

7.2.1. Monitores de Tubo de Rayos Catódicos (CRT).

Se basan en el principio de la emisión de electrones por calentamiento de un filamento, estos electrones forman un haz que atraviesa el espacio existente entre cuatro placas paralelas, dispuestas dos en orientación vertical y dos en orientación horizontal. Si a estas placas se aplica un potencial eléctrico, el haz de electrones verá modificada su trayectoria según la polaridad de las placas. Tengamos en cuenta que la carga del electrón es negativa, por lo tanto será repelida por la placa de polaridad negativa y atraída por la positiva.

El haz de electrones incide sobre una pantalla de fósforo que presenta la propiedad de emitir luz ante la colisión de los electrones; a esta propiedad se denomina fosforescencia. En la pantalla hay una rejilla con potencial (+) o (-), si es (+) atrae al electrón y si (-) lo repele.

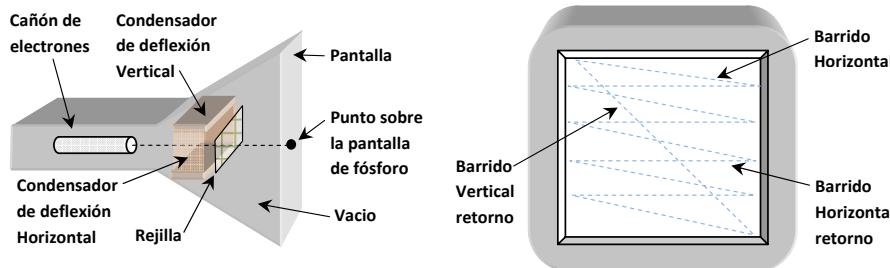


FIGURA 1. Tubo de Rayos Catódicos

Si se trata de un monitor a color, existen tres haces uno para el rojo, otro para el verde y otro para el azul, de esta manera se forma el color por adición de los tres colores complementarios que impactan sobre fósforo rojo-verde-azul, se denominan RGB (*Red – Green – Blue*).

El haz realiza un barrido horizontal y un barrido vertical.

7.2.2. Pantallas Planas LCD (TFT).

Este tipo de pantallas están basadas en la tecnología LCD (*Liquid Crystal Display*) Visualizador de Cristal Líquido. El origen de esta tecnología hay que situarlo en la necesidad planteada por la industria de disponer de visualizadores de bajo consumo en equipos de muy bajo consumo (tecnologías CMOS) y/o alimentados por baterías. En la década de los sesenta se implanta esta tecnología en calculadoras y relojes digitales de pulsera, aunque tardará casi dos décadas en ser una solución industrial universal.

Los visualizadores más comúnmente empleados estaban basados en tecnología LED (Light Emision Diode) Diodo Emisor de Luz, Tubos de Vacío., presentando ambos un consumo alto para lo requerido.

De la evolución de estos pequeños visualizadores, Una fila de 16 caracteres o dos filas de 16 o 20 caracteres, etc., surgen las pantallas que trabajan con puntos “píxel” al estilo de los monitores CRT.

El cristal líquido consiste en moléculas orgánicas viscosas que fluyen como un líquido pero que presentan una estructura espacial como un cristal. Esto es descubierto por el botánico Rheinitzer en 1888.

Cuando todas las moléculas están alineadas en la misma dirección, las propiedades ópticas del cristal dependen de la dirección y de la polarización de la luz incidente. Mediante un campo eléctrico, es posible modificar la orientación de las moléculas y por tanto las propiedades ópticas del cristal. Colocando una luz en la parte posterior, se puede controlar la intensidad de luz transmitida a través del cristal. Para completar el efecto, es necesario disponer en ambas caras de filtros polarizadores; uno de los filtros polariza horizontalmente y el otro verticalmente.

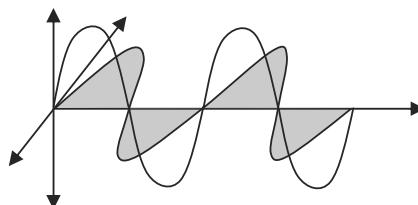


FIGURA 2. Vibración de la Luz en dos Planos

El cristal líquido se confina entre dos placas de vidrio transparente, en estas placas se han insertado matrices de hilos conductores transparentes, aplicando diferentes voltajes a estos hilos, se crearán campos eléctricos que crearán diferentes zonas de alineación. Colocando una fuente de luz en la parte posterior (o luz natural) veremos diferentes imágenes sobre la parte anterior.

Una tecnología muy aplicada es la TN (*Twisted Nematic*) Nemática Torcida. Consiste en la Luz, El Polarizador Horizontal, la Placa de cristal trasera con surcos Horizontales, el Cristal Líquido, la Placa de cristal delantera con surcos verticales y el Polarizador Vertical delantero.

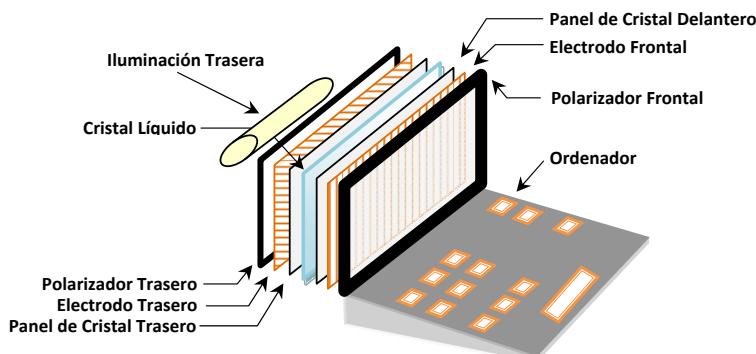


FIGURA 3. Pantallas Planas

Si incidiera luz en la parte posterior, debido a la posición de los polarizadores, la pantalla permanecería oscura; pero como el cristal líquido se va a alinear de surco horizontal a surco vertical, sufre una rotación de 90°, formando una guía de luz, permitiendo el paso de la luz y mostrando la pantalla uniformemente iluminada. Es decir, en ausencia de campos eléctricos, la pantalla se verá iluminada. Si se aplican campos eléctricos a diferentes zonas de la pantalla, se rotará el camino de la onda de luz, apareciendo oscuridad en ese área.

La matriz de hilos conductores mencionada anteriormente, es la que va a dar la resolución de la pantalla en “pixels”. Si la matriz es de 640x480, tendremos el equivalente a un CRT de la misma resolución.

Aplicando un voltaje a uno de los hilos verticales y a otro de los horizontales, se habrá actuado sobre un píxel, oscureciéndolo. Si se realiza el barrido por los hilos horizontales y verticales, con una cadencia de 60 veces por segundo, el ojo tendrá la sensación de imagen estática del mismo modo a como se realiza en un CRT.

Las pantallas de color, utilizan el mismo principio pero disponen de tres filtros, rojo, verde y azul, para separar la luz blanca en cada píxel. Estos tres colores se denominan primarios por poder generar cualquier color con ellos por superposición lineal.

7.2.3. Terminal de Mapa de Carácteres.

Hay tres tipos de terminales más comúnmente utilizados:

- Mapa de Carácteres
- Mapa de Bits
- Terminales RS232C

Un computador genera una información para ser consumida en una pantalla de dos maneras diferentes: Mapa de Carácteres y Mapa de Bits.

Terminal de Mapa de Carácteres: La CPU envía caracteres (bytes) a la memoria de video. A cada carácter se asocia un byte de atributo, este atributo va a indicar al controlador de pantalla el color, intensidad, parpadeo, etc..

Como ejemplo, una pantalla de 25×80 caracteres, requiere 4000 bytes de memoria de video, 2000 para los caracteres y 2000 para los atributos. Es normal que las tarjetas controladoras de video disponga de memoria suficiente como para almacenar varias pantallas.

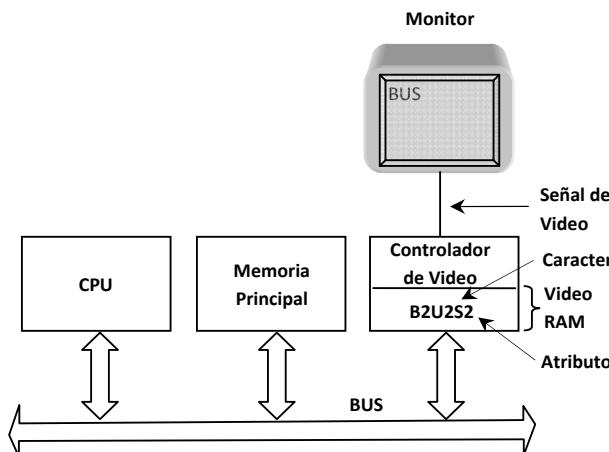


FIGURA 4. Terminal de Mapa de caracteres

Terminal de Mapa de Bits: La CPU ve a la pantalla como un mapa de puntos “pixels”, dividiendo la pantalla en 640×480 puntos u 800×600 o 1024×768 , etc. A cada bit se le asocia un atributo formado por 3 bytes (R-V-A) para poder definir bien el color. Por lo tanto, una pantalla de 1024×768 requiere una memoria de video de 2.3 MB por pantalla a visualizar. Como se puede realizar un Mapa de la memoria, si trabajamos en áreas bien especificadas (ventanas), no será necesario cargar en la memoria de video toda la pantalla sino solamente el área necesaria.

Si se utiliza la pantalla para mostrar video, teniendo en cuenta que el ojo requiere 25 cuadros/s., pensando en la pantalla anterior, se requerirá una tasa de transferencia de 57.6 MB/s. Esto con un Bus EISA, no se consigue, hay que utilizar Bus PCI.

Terminal RS232C: Este tipo de conexión se creó con el fin de permitir una estandarización en las conexiones de los terminales con cualquier

computador. La asociación Americana EIA (*Electronics Industries Association*) Asociación de Industrias Electrónicas creó este tipo de interface.

La norma definía el tipo de conector que es un Sub-D de 25 puntos. También define los niveles eléctricos de las señales transmitidas, el “1” lógico está entre -3 .. -24 Vcc, y el “0” lógico entre +3 .. +24 Vcc. Las tensiones más normalizadas son 12Vcc, aunque dispositivos actuales trabajan con 5Vcc o 9Vcc.

La norma define dos tipos de equipos conectados, son los siguientes:

- **DTE** (*Data Terminal Equipment*): Equipo Terminal de Datos, va a ser el Terminal con el que estamos trabajando, intercambiando datos con el Computador o también es considerado el Computador. Las conexiones del conector presentan una determinada significación.
- **DCE** (*Data Communication Equipment*): Equipo de Comunicación de Datos, este equipo es el que va a unir el Terminal y el Computador a través de una línea telefónica de funcionamiento analógico.

Las conexiones del conector presentan el significado opuesto al del DTE, y está previsto de esta manera para poder conectar un cable “paralelo” sin necesidad de realizar cruzamientos de cables, p.ej.: si en DTE 2 es TxD, en DCE 2 es RxD (TxD del DCE), si en DTE 3 es RxD, en DCE 3 es TxD (RxD del DCE).

Otras señales son: RTS(4), CTS(5), DCD(8), DTR(20), DSR(6), 0V(7), GND(1), etc.

TABLA 1. Señales entre DTE y DCE

DTE	DCE
RTS →	
	← CTS
	← DCD
TxD →	
RxD ←	

El DCE va a ser denominado MODEM, acrónimo de Modulador-Demodulador. Las señales digitales son transformadas a señales analógicas por medio del Modulador y las señales analógicas son pasadas a digitales por medio del Demodulador. Las técnicas se explicarán más adelante.

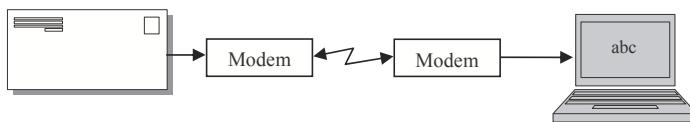


FIGURA 5. Conexión entre Ordenadores

Si el Terminal y el Computador están lo suficientemente cerca, no será necesario el MODEM, podrán conectarse directamente, teniendo en cuenta que habrá que cruzar los hilos de TxD y RxD (2 con 3 y 3 con 2).

Los datos son generados por la CPU y serializados por un dispositivo denominado UART o USART (*Universal Synchronous Asynchronous Receiver Transmitter*). Cada Byte es convertido a una ristra de bits, se trata de un conversor Paralelo/Serie - Serie/Paralelo, añadiéndose bits con el significado siguiente:

- **Bit de Start:** Comienzo de carácter, pasa de 1 a 0.
- **Bit de Paridad:** Paridad Par o Impar, Puede no haber Paridad (no hay bit).
- **Bit de Stop:** Indica fin de carácter.

7.3. RATONES

Hasta la aparición de los ratones, el usuario introducía comandos, vía teclado, para realizar acciones sobre el programa. Estos comandos eran poco amigables para un usuario no experto. Al introducir aspectos más gráficos de cara al usuario, se introdujo este dispositivo que permitía moverse por la pantalla (aplicación) de manera mucho más grata, realizando acciones sobre líneas de comandos de manera más intuitiva.

La funcionalidad del ratón consiste en transmitir al computador, coordenadas (x,y) al desplazarse sobre una superficie plana. Estas coordenadas se transmiten por medio de un código serializado junto con una señal de reloj de validación de datos, el reloj lo transmite el ratón. En este código serializado también se incorpora el estado de los pulsadores que incorpora el dispositivo. El protocolo de comunicaciones es el PS/2 que admite comunicación “bidireccional” y “síncrona”. Cada byte se transmite con: “bit de Start”-“8 bits de Datos”-“bit de Paridad”-“bit de Stop”.

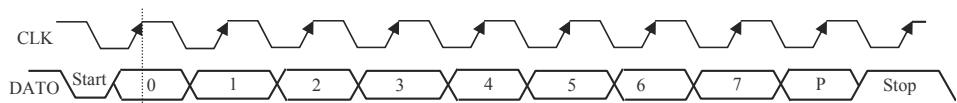


FIGURA 6. Cronograma de Señales en un Ratón

Cada coordenada puede ir codificada en uno o dos bytes y el estado de los pulsadores en otro byte. Esta información es presentada en pantalla en forma de flecha y la acción de los pulsadores, tendrá diferentes resultados según si es zona sensible (menú) o no (uso de botón derecho o central).

TABLA 2. Formato del Paquete de Datos de un Ratón

Formato de los tres bytes del mensaje								
	7	6	5	4	3	2	1	0
Byte 1	Overflow (Δy)	Overflow (Δx)	Signo (Δy)	Signo (Δx)	1	Activ.botón medio	Activ.botón derecho	Activ.botón izquierdo
Byte 2	Δx							
Byte 3	Δy							

- a. Overflow(Δx), Overflow(Δy): “1” si el valor de Δx o Δy excede del rango [-256,255] (Δx y Δy tomarán el valor correspondiente al límite), “0” si no hay overflow.
- b. Signo(Δx), Signo(Δy): “1” si el valor de Δx o Δy es negativo, “0” si es positivo.
- c. Activación de botones: “1” si está presionado, “0” no presionado.

Se han desarrollado tres tecnologías:

- **Mecánicos:** Ruedas de caucho que mueven dos ejes, horizontal y vertical, estos a su vez mueven una resistencia variable. La diferencia de tensión indica el movimiento.
- **Ópticos:** Un Led emite luz y esta es reflejada sobre una superficie de cuadrícula muy fina, detectándose las transiciones en un fotodetector.
- **Optomecánicos:** Una bola de caucho mueve dos ejes, horizontal y vertical, que a su vez mueven dos ruedas con orificios en su periferia, un diodo Led emite luz y esta es detectada por un fotodetector cuando pasa por la ventana de la rueda, la transición de luz a oscuridad y viceversa, va a codificarse como un dato proporcional al movimiento.

7.4. IMPRESORAS

Periférico sobre el que se va a volcar determinada información. Esta información podrá provenir del nivel del Sistema Operativo o bien del nivel del programa de usuario. Las impresoras pueden ser Monocromáticas o de Color.

7.4.1. Impresoras Monocromáticas:

Dentro de este apartado, se presentan distintas tecnologías:

- **Matriciales:** El mecanismo de impresión consiste en una cabeza móvil que consta de entre 7 y 24 agujas que forman el núcleo de un electroimán. Actúan por excitación del electroimán. Con las cabezas de 7 agujas, los caracteres se componen con una matriz de 5x7 puntos. Las impresoras se suelen presentar en dos formatos 80 y 120 caracteres por línea; existen otros formatos más pequeños pero no tienen usos ofimáticos, normalmente en transacciones comerciales con el gran público. Su velocidad se mide en caracteres por segundo.

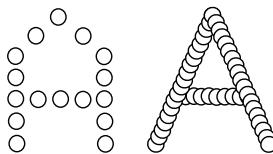


FIGURA 7. Muestras de impresión por Agujas

La impresión se consigue por existir una cinta entintada entre las agujas y el papel, produciéndose la transferencia de tinta por impacto de la aguja. El aumento de la calidad de la impresión suele consistir en realizar dos pasadas, además de superponer puntos entre los puntos impresos en la primera pasada. Esto provoca una operación más lenta de este periférico.

Ventajas: Son muy económicas en su mantenimiento. Permiten realizar copias por el uso de papel de carbón. Abanico amplio de precios (para gran público y empresarial).

Inconvenientes: Son lentas y su uso está orientado a la impresión de caracteres. Los gráficos son de baja calidad. Ruidosas.

En impresoras de pequeño formato, las impresoras matriciales, han sido substituidas por las de tecnología térmica.

▷ **Inyección de Tinta:** Consiste en una cabeza móvil provista de un cartucho de tinta negra, la impresión se produce por expulsión de gotas, a través de boquillas o conductos, de tinta sobre el papel.

La técnica consiste introducir una gota de tinta en una cámara de expansión, calentar la gota de tinta, llevándola al punto de ebullición, en este momento explota (burbuja generada) y sale expulsada por la boquilla. Posteriormente, el enfriamiento provoca un vacío que absorbe otra gota de tinta y así continúa el proceso.

Las densidades de impresión más comunes pueden ser 300 – 720 – 1440 dpi (*dots per inch*). La velocidad está supeditada al ciclo “ebullición/enfriamiento”. La calidad va a depender de la cantidad de tinta expulsada por el inyector, llegándose a p.ej. 5 picolitros (5 x 10-12 litros).

Ventajas: Impresión de calidad. Gráficos de calidad. Silenciosas. Abanico amplio de precios (para gran público y empresarial).

Inconvenientes: Caras en su mantenimiento, los cartuchos son caros. Lentas.

- ▷ **Láser:** Básicamente consiste en transferir una línea completa de puntos (ancho de la página) al papel. Básicamente es la misma tecnología que las de las fotocopiadoras y facsímiles.

La descripción completa del proceso es la siguiente:

- ◆ Un cilindro de precisión es cargado electroestáticamente con una tensión entorno a los 1000Vcc.
- ◆ Un haz de luz láser incide sobre el rodillo, recorriéndolo horizontalmente. Si se modula el haz de luz, de tal manera que sobre unas zonas incide más luz que en otras, ahí donde incide luz, pierde carga eléctrica. Esta operación crea una imagen eléctrica de la línea original.
- ◆ El haz de luz es guiado por un espejo octogonal giratorio.
- ◆ Posteriormente al girar el cilindro y pasar por la zona donde existe un polvo negro ionizable (toner), se adhiere a las áreas cargadas. Esta operación crea una imagen real de la línea original.
- ◆ Posteriormente el cilindro pasa por el papel, presionándolo, y le transfiere el polvo negro.
- ◆ El papel pasa por unos rodillos calientes que fijan (funden) el polvo al papel de manera permanente.
- ◆ Una vez que el rodillo abandona el papel, pasa por un descargador de corriente electroestática.

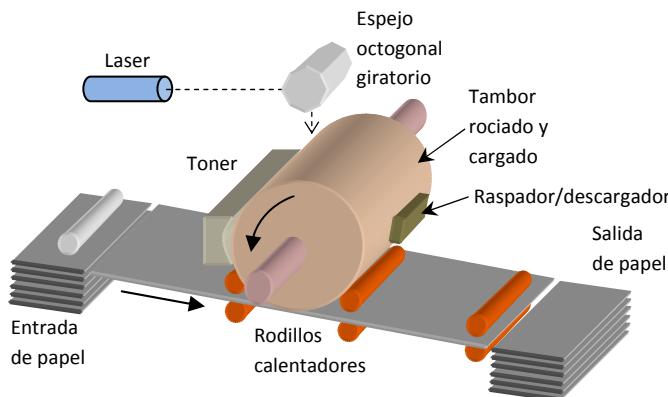


FIGURA 8. Esquema de una Impresora Laser

Las impresoras láser incorporan CPU y memoria (del orden de MB) para poder capturar una imagen completa (mapa de bits) de cada página y distintos tipos de letras enviados por el computador. Incorporan, además, distintos tipos de letras.

Aceptan comandos especiales para el manejo de las páginas, como por ejemplo los lenguajes PCL de HP y PostScript de Adobe.

Si se quiere imprimir imágenes, y estas presentan matices de grises, hay que recurrir a una argucia técnica similar a la empleada en los carteles comerciales. Esta se basa en el empleo de los medios tonos. El matiz de gris se consigue por cambiar la densidad de puntos negros en una matriz, cuantos menos puntos negros más clara la percibirá el ojo humano, cuantos más puntos negros más oscura se percibirá.

De esta manera, la imagen se trabaja por zonas, las zonas se trabajan por matrices de 6x6, insertando celdas oscuras desde al centro hacia la periferia, esto reduce la densidad de la impresora ya que si tenemos una densidad de 600 dpi, al utilizar 6 celdas por zona, la densidad útil será 100 dpi.

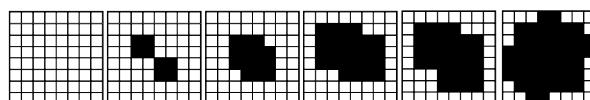


FIGURA 9. Densidad de puntos para escala de Grises

Ventajas: Impresión de alta calidad. Gráficos de alta calidad. Silenciosas. Relativamente rápidas. Alta fiabilidad. Precios adecuados para el entorno empresarial.

Inconvenientes: Caras en su mantenimiento, los cartuchos son caros. Caras para el gran público.

7.4.2. Impresoras de Color:

En este tipo de impresoras, el color se crea de forma “sustractiva”, al contrario que en los CRT (aditivo: RGB). Los colores básicos son el Cian, Magenta y Amarillo (CMY). Además se incorpora un cuarto color, el Negro (K: blacK) porque con la aportación total de CMY (absorción total de los colores), no se consigue un verdadero negro. Esta impresión se denomina CYMK.. Dentro de este apartado, se presentan distintas tecnologías:

- **Inyección de Tinta (CYMB):** El funcionamiento es idéntico a las monocromáticas, solo que incorporan cuatro cartuchos.

Ventajas Generales: Impresión de calidad. Gráficos de calidad. Calidad fotográfica aceptable. Silenciosas. Abanico amplio de precios (para gran público y empresarial).

Inconvenientes Generales: Caras en su mantenimiento, los cartuchos son caros. Lentas.

Hay dos tipos de tintas con las que se obtiene una impresión de calidad:

- ▷ **Tinta de Colorantes:** Tintas basadas en colorantes, consiste en tintes disueltos en un portador.

Ventajas: Impresión de calidad. Colores brillantes. Silenciosas. Abanico amplio de precios (para gran público y empresarial).

Inconvenientes: Se altera el color por exposición a la luz ultravioleta (sol). Caras en su mantenimiento, los cartuchos son caros. Lentas.

- ▷ **Pigmentos:** Tintas basadas en pigmentos, consiste en partículas de pigmentos, suspendidas en un portador; este se evapora en el papel, dejando como residuo el pigmento.

Ventajas: Impresión de calidad. No se altera el color por exposición a la luz ultravioleta (sol). Buena calidad fotográfica. Silenciosas. Abanico amplio de precios (para gran público y empresarial).

Inconvenientes: Colores menos brillantes que la anterior. Los pigmentos tienden a taponar las boquillas. Requieren un papel especial para fotografía (retenedor de pigmentos). Caras en su mantenimiento, los cartuchos son caros. Lentas.

□ **Tinta Sólida:** Formada por cuatro bloques de cera de tinta, que es fundida para ser introducida en un deposito de tinta caliente. El tiempo de preparación de la impresora está en torno a los 10 minutos. La tinta se rocía sobre el papel, solidificándose y fusionándose con el papel al pasar entre dos rodillos que la comprimen.

□ **Láser:** Funcionamiento idéntico a la monocromática, solo que incorpora las tres deposiciones necesarias para el C, M y Y. Además dispone de la K. Necesita cuatro toner distintos.

Requieren de una gran cantidad de memoria para poder almacenar una página, esto las hace más costosas, por lo demás su calidad es alta.

□ **Cera:** Disponen de una cinta ancha de cera que contiene los cuatro colores. La cinta de cera está segmentada en bandas. La anchura de la banda es la de la hoja.

Hay miles de elementos calefactores que funden la cera y la depositan sobre el papel a su paso. Estas deposiciones conforman los píxel.

Este tipo de impresoras están siendo substituidas por las anteriores.

□ **Sublimación:** Por sublimación se entiende el paso de “fase sólida a fase gas” sin pasar por fase líquida. Un portador de los cuatro colores CMYK pasa por una cabeza térmica que calienta los colorantes y estos pasan a fase gas siendo absorbidos por el papel.

Cada elemento calefactor puede producir 256 temperaturas distintas, cuanto más alta es la temperatura, más colorante se deposita y más intenso será el color. A diferencia de las anteriores, no es necesaria la técnica de los medios tonos por ser casi posible obtener colores continuos por cada píxel.

Esta es la tecnología utilizada para las impresoras fotográficas de calidad. El papel es caro.

7.5. MODEMS

Como se ha comentado anteriormente, un MODEM es un dispositivo que sirve para poner en comunicación dos equipos separados físicamente por una gran distancia. La conexión se apoya en la línea telefónica.

Un computador, a través de su conexión RS232C, transmite y recibe datos de manera digital, bien por cambio de tensión de p. ej -12Vcc a +12Vcc (“1” a “0”) o bien 0Vcc a 5Vcc (“0” a “1”). Si se transmitiesen estas señales directamente por la línea telefónica, estas sufrirían una gran distorsión, debido a la resistencia, capacidad y autoinducción de las líneas.

Si se transmite una onda senoidal, esta sufre una muy baja distorsión si nos movemos entre 1000 Hz y 3000 Hz. (aproximadamente). Esta señal va a recibir el nombre de “Portadora”. Una onda senoidal dada una frecuencia en el rango de las anteriores, no transmite información alguna. En cambio si alteramos esta onda en frecuencia, amplitud o fase, sí se podrá obtener información de estos cambios.

Este proceso se conoce como “Modulación”. Veamos como se efectúa este proceso:

- **Modulación en Amplitud:** Se emplean dos niveles de voltaje, el nivel alto va a indicar un “1” y el nivel bajo un “0”.
- **Modulación en Frecuencia:** El nivel de voltaje es constante durante toda la transmisión. Para diferenciar “0” de “1”, hay transición de frecuencia baja a frecuencia alta. Esta técnica se conoce por Modulación por Desplazamiento de Frecuencia FSK (*Frequency Shift Keying*).
- **Modulación en Fase:** La amplitud y la frecuencia no cambian, cambia la fase de la portadora, esta se invierte 180° cuando se pasa de “1” a “0” o viceversa. Fijándonos en esta técnica, si se consigue realizar cambios de fase en ángulos diferentes a 180°, podrían transmitirse más bits de información por unidad de tiempo indivisible. Si se desplazara la fase de la portadora de manera abrupta, en 45, 135 (45+90), 225 (135+90) o 315 (225+90), se podrían transmitir dos bits por cada transición, es decir, 45° “00”, 135 “01”, 225 “10” y 315 “11”. Esto se conoce como “codificación por fase díbit”. Hay técnicas que aumentan el nº de bits por periodo.

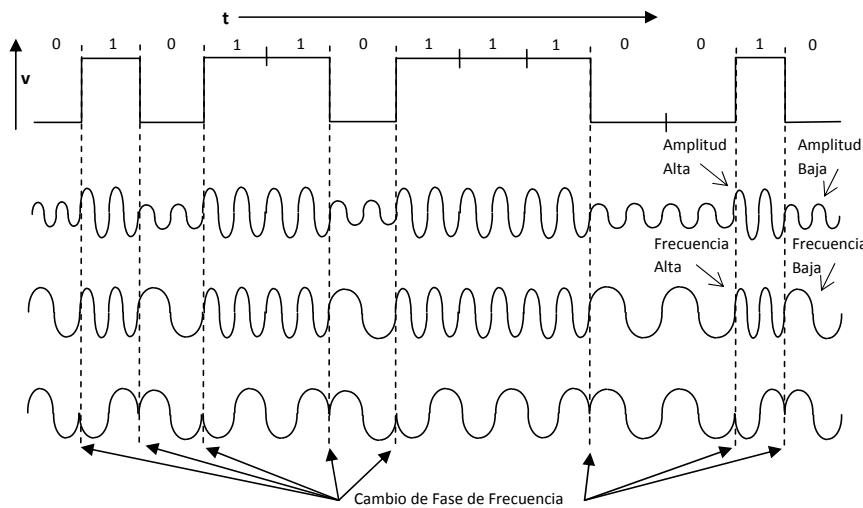


FIGURA 10. Modulación en Amplitud – Frecuencia - Fase

La velocidad de comunicación se mide en “baudios” (*baud rate*), e indica el nº de “símbolos” que se envía en la unidad de tiempo, estos símbolos pueden representar un único bit (caso en el que coinciden bits por seg. con los baudios) o varios bits donde ya es diferente la tasa de bits por segundo de los baudios.

El equipo que realiza estas modulaciones es el MODEM, que a su vez, detecta en su recepción esta modulación y “Demodula” la señal, obteniendo la información digital. La información enviada, normalmente se codifica en 8 bits por carácter más un bit de comienzo “Start” y un bit de paro “Stop”, además puede haber un bit de paridad “*Parity*” (par o impar). En sistemas más antiguos (lentos), el nº de bits de Stop podía ser de 2 o 1 ½.

Las velocidades de modems, sin modulación de fase dividida, más comúnmente manejadas son: 150 – 300 – 600 – 1200 – 2400 – 4800 – 9600 – 19200 – 38400 bits/s. Los modems que manejan velocidades superiores, p. ej., 57600 bps, emplean modulaciones combinadas.

Los equipos, según su capacidad de transmisión y/o recepción simultánea, se dividen en tres categorías:

- **Full Duplex:** Transmiten y reciben simultáneamente.
- **Half Duplex:** Transmiten o Reciben, pero no simultáneamente.
- **Simplex:** Solo transmiten o solo reciben.

7.6. CÓDIGO DE CARACTERES

Por código de caracteres se entiende la correspondencia que existe entre el símbolo que queremos representar y el nº entero que podemos asignar en la máquina. Si nuestro computador va a trabajar con nuestros propios computadores o periféricos, podremos utilizar un código de caracteres inventado por nosotros, pero normalmente necesitamos conectarlos y trabajar con equipos diferentes. Por ello se hace necesario que exista una normalización en cuanto a la representación de los caracteres comúnmente utilizados.

Para resolver esta normalización, se crean los códigos ASCII y UNICODE.

7.6.1. ASCII.

El significado del acrónimo es, Código Estándar Americano para Intercambio de Información (*American Standard Code for Information Interchange*). El código ASCII inicial, representa 128 caracteres sobre 7 bits. Se representa en Octal, Hexadecimal y en Decimal. Podemos dividir los caracteres en dos grupos, según su significado:

- **Caracteres de Representación Gráfica:** Son todos aquellos símbolos que representan letras mayúsculas o minúsculas, números, símbolos de operación algebraica, etc., son los caracteres impresibles; p. ej., la “A” se representa por 41H, “+” se representa por 2BH, etc.
- **Caracteres de Control:** Son aquellos símbolos que indican a los computadores el comienzo de una acción, el resultado de ella, etc.; p. ej., SOH (*Start Of Head*) significa “Comienzo de Cabecera”, EOT (*End Of Transmision*) significa “Fin de Transmisión”, ACK, NACK, etc.

Posteriormente se amplió la representación de 7 bits a 8, denominándose ASCII extendido, pudiendo representarse 256 símbolos. Ya se permiten caracteres de determinadas lenguas, algunos códigos gráficos para poder realizar marcos de ventanas sobre la pantalla o impresora, etc. Esta extensión se denominó Latín-1, norma IS 646. Posteriormente y sobre este mismo marco de 256 códigos, se crean normalizaciones para diferentes idiomas, con el inconveniente de tener diferente significado el código según el idioma.

A continuación se representa el código ASCII en forma Hexadecimal y Decimal. La Tabla 3 (a) representa el código hexadecimal del carácter representado y la Tabla 3 (b) representa el código decimal del carácter representado.

TABLA 3. (a) HEXADECIMAL, (b) DECIMAL

00 NUL	01 SOH	02 STX	03 ETX	04 EOT	05 ENQ	06 ACK	07 BEL
08 BS	09 HT	0A NL	0B VT	0C NP	0D3 CR	0E SO	0F SI
10 DLE	11 DC1	12 DC2	13 DC3	14 DC4	15 NAK	16 SYN	17 ETB
18 CAN	19 EM	1A SUB	1B ESC	1C FS	1D GS	1E RS	1F US
20 SP	21 !	22 "	23 #	24 \$	25 %	26 &	27 '
28 (29)	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5A Z	5B [5C \	5D]	5E ^	5F _
60 ‘	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7A z	7B {	7C	7D }	7E ~	7F DEL

(a)

0 NUL	1 SOH	2 STX	3 ETX	4 EOT	5 ENQ	6 ACK	7 BEL
8 BS	9 HT	10 NL	11 VT	12 NP	13 CR	14 SO	15 SI
16 DLE	17 DC1	18 DC2	19 DC3	20 DC4	21 NAK	22 SYN	23 ETB
24 CAN	25 EM	26 SUB	27 ESC	28 FS	29 GS	30 RS	31 US
32 SP	33 !	34 "	35 #	36 \$	37 %	38 &	39 '
40 (41)	42 *	43 +	44 ,	45 -	46 .	47 /
48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7
56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G
72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W
88 X	89 Y	90 Z	91 [92 \	93]	94 ^	95 _
96 ‘	97 a	98 b	99 c	100 d	101 e	102 f	103 g
104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w
120 x	121 y	122 z	123 {	124	125 }	126 ~	127 DEL

(b)

7.6.2. UNICODE.

La realización de la extensión del código ASCII, no era suficiente para incorporar todos los idiomas que se sumaban al mundo de la informática. Es por ello que se forma un consorcio de compañías informáticas y crean el estándar IS 10646 que se corresponde con el código UNICODE. Este estándar ha ido pasando por distintas versiones estando en estos momentos en la 4.0.1. La representación es más ambiciosa y se realiza en 16 bits, lo que permite incorporar 65,536 códigos, donde los 256 primeros se corresponden con el ASCII extendido (Latín-1) facilitando la conversión entre códigos.

UNICODE asigna un único código a cada símbolo existente, de esta manera se evita el que el computador o el usuario, tenga que cambiar la página de código según la situación. Esta asignación se denomina Punto de Código. Además, las letras con signos “diacríticos”, como p. ej. en Alemán “ü”, diferencia la letra del signo y deja al software que realice la composición de caracteres como este o nuevos que deban ser incorporados.

La estructuración de este código es como sigue:

- Cada alfabeto internacional, dispone de una zona consecutiva, p. ej. Latin (336), griego (144), etc. . Cada alfabeto tiene asignados más puntos que los necesarios. Caso de mayúsculas y minúsculas, etc.
- Zona de código a signos diacríticos (112).
- Zona de puntuación (112).
- Zona de subíndices y superíndices (48).
- Zona de símbolos de divisas (48).
- Zona de símbolos matemáticos (256).
- Zona de formas geométricas (96).
- Zona de ornamentos tipográficos (192).

Además de estas zonas hay otras dedicadas a los símbolos del Chino, Japonés y Coreano:

- Zona de 1024 símbolos fonéticos (Katakana y bopomofo).
- Zona de 20,992 ideogramas “Han” unificados (Chino y Japonés).
- Zona de 11,156 sílabas Hangul Coreanas.

Existe otra zona de uso general:

- Zona de 6400 códigos de uso local.

Nota: Bopomofo ocupa de 3100H – 312FH, son 37 símbolos derivados de caracteres Chinos. Símbolos que representan sonidos del Mandarín en unión con caracteres Chinos.

Capítulo 8

ARQUITECTURA DE COMPUTACIÓN EN PARALELO

Esforcémonos de modo que cada uno de nosotros pueda considerarse a sí mismo artífice de la victoria.
(Jenofonte: 430 -355 a.C)

RESUMEN. Cuando los requerimientos de computación son muy elevados y con un solo procesador no es posible dar respuesta en un tiempo considerado como razonable, se hace necesario reestructurar el programa, descomponiéndolo en N bloques ejecutables independientes para que puedan ser ejecutados en N procesadores independientes, recopilando todas las partes sobre un procesador. Los diferentes procesadores/computadores van a presentar estructuras soportadas por memorias o por una red.

8.1. INTRODUCCIÓN

La cuestión fundamental que se plantea es como conseguir mayor capacidad de cómputo dentro de una inversión razonable. Unos resolverán la cuestión con la adquisición de un computador más rápido que el anterior pero para otros esto nunca será la solución porque el nivel de cálculo necesario siempre superará a la mejor de las expectativas de evolución del mercado en aspectos de velocidad. Por ello es necesario invertir en esfuerzos para, por una parte mejorar el diseño de los programas, optimizando su ejecución o aislando tareas independientes o “cuasi” independientes y por otra conseguir crear una arquitectura de procesadores trabajando en “paralelo”, de tal manera que cada procesador realice la tarea específica y comparta resultados con el resto de procesadores; esto en el caso de tratarse de tareas “cuasi” independientes. Podemos tener computación en paralelo de tareas que son absolutamente independientes.

Veamos un ejemplo sencillo de Procesamiento Paralelo:

$$\begin{pmatrix} a_1 & b_1 \\ c_1 & d_1 \end{pmatrix} + \begin{pmatrix} a_2 & b_2 \\ c_2 & d_2 \end{pmatrix} = \begin{pmatrix} a_1 + a_2 & b_1 + b_2 \\ c_1 + c_2 & d_1 + d_2 \end{pmatrix}$$

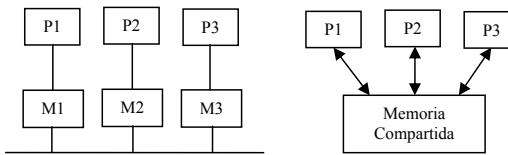


FIGURA 1. Esquemas de Arquitecturas de Procesamiento Paralelo

Si disponemos de 4 procesadores, cada procesador realizará la suma de la fila-columna correspondiente, entregando el resultado a uno de ellos. Una situación real es más compleja, hay un mayor nº de Datos, uno de los procesadores recogerá los Datos de entrada repartiéndolos al resto de procesadores según el algoritmo de distribución y el resultado será recibido por uno de los procesadores.

La computación en paralelo presenta un inconveniente añadido que es que el programador debe conocer, en alguna medida, la arquitectura de su multicomputador. Aspecto este que con un único procesador, es el compilador sobre quien recae la labor.

Para la computación en paralelo se necesitan Procesadores y Memorias. Cada Procesador dispondrá de una Memoria de Datos propia o bien dispondrá de una Memoria de Datos compartida por los Procesadores. Pueden existir más soluciones dependiendo de arquitecturas.

Por lo tanto, el diseño de una arquitectura de multiprocesamiento, requiere plantearse las siguientes cuestiones:

- ¿Cómo se comparten los Datos?
- ¿Cómo se coordinan los Datos?
- ¿Cuantos Procesadores tiene?

Una solución a la compartición de memoria se basa en disponer memorias tipo caché para poder trabajar a alta velocidad, disponiéndose de 2 a 4 niveles de caché.

Los procesadores con memoria compartida presentan tres variantes:

- Tiempo de acceso a memoria igual para todos los procesadores y todas las palabras. Estas máquinas se denominan UMA (acceso uniforme a memoria), dentro de esta categoría están los SMP (multiprocesadores simétricos).
- Algunos accesos a memoria son más rápidos que otros dependiendo de que procesador haga la referencia y a que palabra. Se denominan NUMA (acceso no uniforme a memoria). El acceso a memoria local es más rápido que a la remota.

- Cuando solo hay acceso a memoria caché. Estas máquinas se denominan COMA (*Caché Only Memory Access*)

Las máquinas NUMA presentan un mejor rendimiento y son más escalables (mejor crecimiento). La tendencia comercial es hacia esta solución.

El modelo alternativo al de memoria compartida es el de “paso de mensajes”. En este caso se realizan intercambios de mensajes entre los procesadores. Esto es necesario cuando la memoria es “privada” para cada procesador. Para ello deben implementarse rutinas o directivas que realicen la función de “enviar” y “recibir” datos. Esto lo podemos ver en computadores conectados a través de una red local como ejemplo extremo de paso de mensajes entre procesadores.

La coordinación de los Datos requiere de mecanismos que garanticen la validez de ellos en todo momento, es decir si un procesador modifica un dato y estamos trabajando con memoria compartida, se debe asegurar que el procesador que realiza el cambio es el único que accede a memoria y una vez realizado el cambio este es comunicado a los demás procesadores. Esta técnica se denomina de “Bloqueo” (*Lock*). El mecanismo se controla por semaforización con acceso indivisible (uso de instrucción “*swap*”).

Cuando se trata de memorias propias, el cambio es notificado con SEND o RECEIVE. Un procesador sabe cuando envía y cuando recibe por lo tanto el acceso a memoria propia se realiza de manera voluntaria y con total control.

Hemos visto los dos tipos de comunicación entre procesadores, ahora podemos contemplar el tipo de organización en la interconexión que puede existir entre los procesadores.

8.2. MODOS DE INTERCONEXIÓN

Recogiendo las ideas anteriores, podemos distinguir dos tipos de configuraciones de interconexión:

- Multiprocesadores conectados por un solo Bus
- Multiprocesadores conectados por una Red

8.2.1. Multiprocesadores conectados por un solo Bus.

Los microprocesadores de alto rendimiento y bajo coste renovaron el interés por los multiprocesadores en los años 80. Existen varias razones para poder optar a este tipo de conexión por medio de un Bus:

- Los microprocesadores son más pequeños que un procesador multichip (conf. Bitslice), por lo tanto pueden colocarse más procesadores en el Bus.
- Las cachés pueden reducir el tráfico del Bus
- Existen mecanismos para mantener la coherencia de caché y memoria para multiprocesamiento. Esto simplifica la programación.

La Figura 2 nos da una idea de la arquitectura.

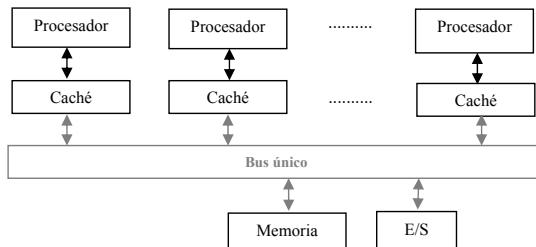


FIGURA 2. Multiprocesadores conectados por un solo Bus

El tráfico por procesador y el ancho de banda del bus determinan el número de procesadores útiles en este sistema multiprocesador. Una configuración comercial como la anterior maneja entre 2 y 36 procesadores.

El modo en como un procesador trata el movimiento de datos de o hacia memoria es con instrucciones LOAD y STORE (o equivalentes), no es necesario nada más.

Las cachés replican los datos en sus memorias tanto para reducir la latencia de acceso a los datos como para reducir el tráfico de datos con la memoria en el Bus. Esto da lugar a los “Protocolos de Coherencia de Caché”.

Protocolo “*Snooping*” (husmear o espiar): Mediante este protocolo todos los controladores de las cachés vigilan el tráfico en el Bus de tal manera que determinan si tienen una copia del bloque compartido. El esquema de la Figura 2 puede expandirse de la siguiente manera:

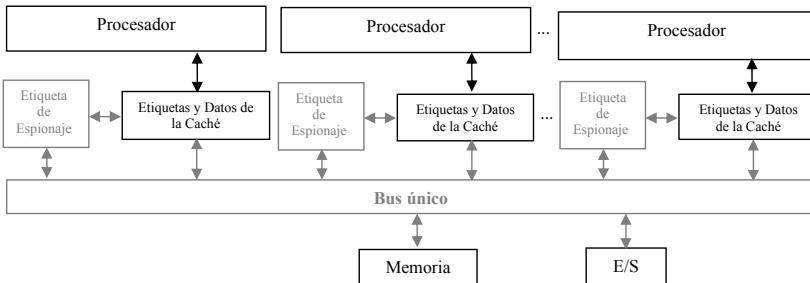


FIGURA 3. Multiprocesadores con Protocolo Snooping

La coherencia implica Lectura y Escritura. Copias múltiples en caso de lectura no presenta ningún problema, pero la escritura de una palabra en un procesador necesita un acceso exclusivo a la palabra. El resto de procesadores debe tener acceso a esa palabra en el momento de ser actualizada. Por lo tanto una escritura en una palabra requiere la actualización de todas las palabras compartidas en todas las cachés o bien la invalidación de todas las palabras después de la escritura. El protocolo debe conocer qué cachés comparten el dato. En las etiquetas de “espionaje” están replicadas las etiquetas de direcciones de la caché (no la caché al completo).

Los bits de estado ya presentes en la caché se expanden para acomodar el protocolo de “espionaje”.

Se dan dos tipos de protocolos de “espionaje” según como se traten las escrituras:

- **Invalidación por escritura:** El procesador que realiza la escritura, invalida previamente las palabras de las copias de las otras cachés y después actualiza los datos locales. El procesador que escribe envía una señal de invalidación al bus y todas las cachés comprueban si tienen una copia, si la tienen invalidan el bloque compartido que la contiene.
- **Actualización por escritura:** El procesador que escribe, envía los nuevos datos a todos los procesadores, de esta manera las copias se actualizan con el nuevo valor. Esto se conoce como “Escritura Difundida” (*Write Broadcast*).

La escritura por invalidación es similar a la “escritura diferida” (recuérdese la memoria caché). Esta escritura usa el bus una única vez ya que se envía la invalidación una única vez. Reduce el tráfico del Bus por lo tanto reduce el ancho del Bus. Permite disponer de más procesadores para un mismo ancho

de Bus. Prácticamente todas las máquinas comerciales utilizan el protocolo de escritura por invalidación.

La escritura por actualización es similar a la “escritura a través” (recuérdese la memoria caché) porque todas las escrituras son enviadas por el bus para actualizar las copias de los datos compartidos.

Un ejemplo de protocolo de Invalidación por escritura, es el MESI (Pentium Pro y Power PC), basado en un protocolo anterior denominado “Protocolo de Escritura Única”. Consta de cuatro estados que son los que dan el nombre al acrónimo.

1. Modified (Modificada): La entrada es válida, la memoria no es válida (no existen copias).
2. Exclusive (Exclusiva): Ninguna caché contiene la línea (memoria actualizada).
3. Shared (Compartida): Varias cachés podrían contener la línea (memoria actualizada).
4. Invalid (No Válida): La entrada de caché no contiene datos válidos.

Retomando el tema central de este apartado, conexión a través de un solo Bus, se puede realizar la consideración final siguiente, y es que el diseño de sistemas multiprocesador con un solo bus presenta limitaciones debido a que las tres características deseables “Gran Ancho de Banda”, “Latencia Pequeña” y “Gran Longitud”, son incompatibles. Existe una limitación en el ancho de banda de los módulos de memoria conectados al bus. Esto hace que hoy en día el número de procesadores conectados vaya disminuyendo (los 36 mencionados anteriormente va decreciendo). La ampliación se consigue acudiendo a la conexión por red, como se verá más adelante.

Esta estructura de datos compartidos presenta una “localidad” espacial y temporal menor que el resto de tipos de datos. Los fallos de caché deciden la conducta de la caché aunque el acceso a datos esté entre un 10 % y un 40 %. El tamaño del bloque también es un factor a tener en cuenta.

8.2.1.1. Sincronización.

Como se ha comentado en la introducción, debe existir un método de coordinación para poder acceder a la memoria compartida por parte de un único procesador, debe garantizarse un proceso de exclusión mutua. El método garantizará un “bloqueo” de la memoria por medio de la conveniente “semaforización”. Se leerá/escribirá sobre la “variable de bloqueo”.

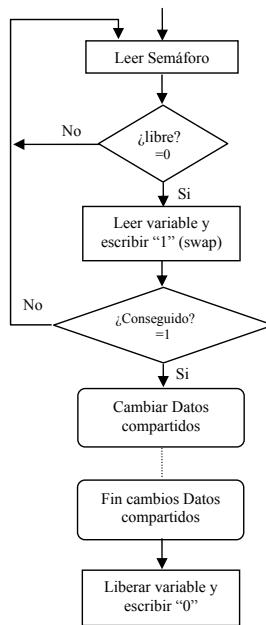
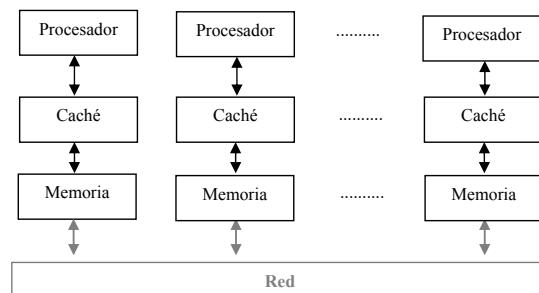


FIGURA 4. Sincronización de Memoria

8.2.2. Multiprocesadores conectados por una Red.

Mediante esta configuración se consigue aumentar la longitud y el ancho de banda. Hemos añadido más buses para el movimiento de datos a memoria, cada memoria está conectada a un procesador de tal manera que el bus se utiliza para accesos a memoria por cada procesador pero la red solo es utilizada por los procesadores, no se requiere acceso a memoria. Las memorias son privadas.



Podemos ver esta arquitectura como una Memoria Compartida “Distribuida” (DSM), es decir, una memoria dividida en páginas de direcciones virtuales donde cada página es asignada a un procesador o de múltiples memorias “privadas”, donde la comunicación se realiza por medio de directivas del Sistema Operativo SEND (enviar) y RECEIVE (recibir), en contraposición a las instrucciones LOAD y STORE vistas en arquitecturas de un solo Bus. Realmente es mejor definir esta memoria como “Memoria Compartida Distribuida” (DSM), ya que cuando un procesador necesite un dato y no lo tenga realizará una petición por medio de la red.

Si la memoria es distribuida, cuando un procesador haga referencia a una zona de memoria ubicada en el espacio de otro procesador, el Sistema Operativo ejecutará una “trap” (trampa) y buscará en el procesador adecuado la memoria requerida (SEND y RECEIVE). Si no es distribuida, las directivas “enviar-recibir” serán realizadas por el programa con llamada al S.O.

Los multiprocesadores conectados en Red, deben presentar también una coherencia de Caché, en este caso uno de los protocolos más importantes es relacionado con el manejo de un directorio donde están relatadas las líneas de caché utilizadas.

Protocolo de Directorios: Se mantiene una Base de Datos que indica donde está cada línea de caché y cual es su estado. Cuando se referencia a una línea de caché, se consulta este directorio para averiguar donde está y si está limpia o sucia (modificada). El hardware debe ser de alta velocidad ya que este directorio se consulta con cada instrucción. La Figura 99 muestra esta arquitectura.

Cuando un Procesador emite una dirección, el MMU la traduce a una dirección física, dividiendo la información entre campos, un primer campo indicará en qué Nodo está el dato requerido, un segundo campo indicará que línea (bloque) de caché es la requerida y un tercer campo indicará la distancia (palabra requerida).

TABLA 1. Dirección en el Bus

Nodo	Bloque	Distancia
------	--------	-----------

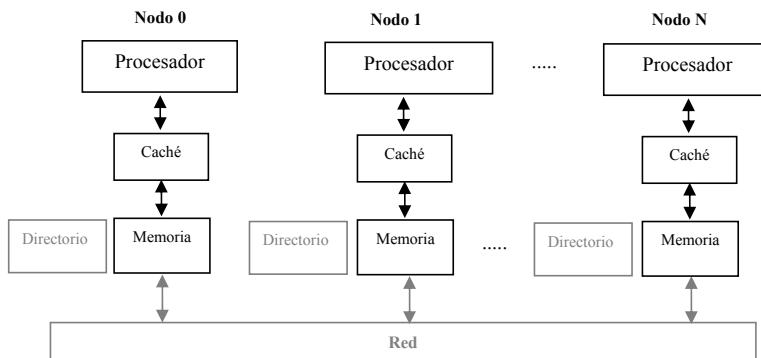


FIGURA 6. Multiprocesadores con Protocolo de Directorios

La memoria se reparte entre todos los procesadores, de tal manera que cada procesador dispone de un área reservada exclusivamente para el.

El controlador del directorio envía comandos explícitos a los nodos (procesadores) que tienen copia de los datos. Como se aprecia, no está constantemente revisado el tráfico de la Red para ver si hay peticiones que afectan al nodo.

8.2.3. Comparativa de Arquitecturas.

La Figura 100 muestra unas gráficas comparativas entre los dos tipos de arquitecturas. (datos obtenidos en el año 1997):

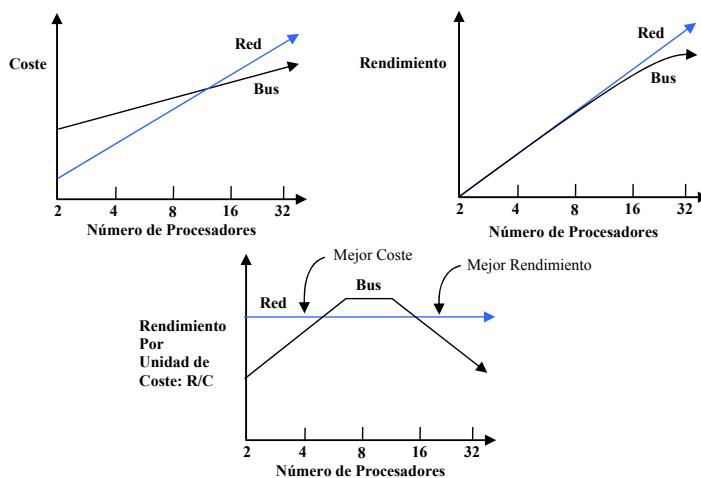


FIGURA 7. Coste y Rendimiento según arquitecturas

El diseñador decidirá la zona óptima de trabajo del Bus.

8.3. CLUSTERS

Se dan otro tipo de aplicaciones para grandes computadores, como Bases de Datos, Servidores, Procesamiento por Lotes, etc., que pueden ser ejecutados en máquinas “menos acopladas” que las máquinas con arquitecturas como las anteriores con coherencia de caché. Estas máquinas necesitan una disponibilidad permanente, lo que implica cierta tolerancia a fallos y reparabilidad. Si nos basamos en las redes de área local de alta velocidad, los commutadores de alta velocidad y los ordenadores de sobremesa, la arquitectura para un procesamiento a gran escala se construirá en torno a agrupaciones de los elementos mencionados, creándose “*Clusters*” (agrupaciones) de computación. Es decir, los “*clusters*” son agrupaciones de computadores (normalmente PCs) libremente acopladas. Están conectados por medio de redes locales de características no propietarias. Un usuario verá al *cluster* como una única máquina.

Desventaja: Una desventaja de los clusters es que el coste de administrar N máquinas es similar al coste de administrar N máquinas independientes, mientras que el coste de administración de un sistema multiprocesador con un espacio de direcciones compartido por N procesadores es similar a administrar una sola máquina.

Ventaja: Teniendo en cuenta que un cluster se construye con computadores (no con procesadores), cuando se deba reemplazar una máquina, esto será una tarea mucho más fácil que en una arquitectura NUMA, no será necesario el tener que parar todo el sistema. Permite una alta disponibilidad y una expansión rápida.

Una evolución actual de los clusters es el realizar arquitecturas mixtas, es decir, cluster formados por computadores conectados en red y los computadores con varios procesadores (arquitectura multiprocesador) UMA.

Podemos ver una arquitectura denominada híbrida y como ejemplos podemos tomar un cluster de 32 procesadores que puede ser construido con 8 procesadores UMA con 4 procesadores cada UMA, o bien, 4 procesadores UMA con 8 procesadores cada UMA. A esta agrupación también se la conoce como “cluster de memoria compartida”.

Por último comentar que los clusters se clasifican en “centralizados” y “descentralizados”; en los primeros, se encuentran en la misma sala todas sus unidades mientras que en los segundos se encuentran sus unidades distribuidas en el edificio o edificios.

8.4. TOPOLOGÍAS DE RED

En este apartado se van a describir las diferentes topologías que afectan tanto a Procesadores como a Computadores (*Clusters*), es decir, los modos de interconexión aplican a ambos tipos de configuración ya sea Multiprocesador conectado por Red como Multicomputador conectado por Red.

8.4.1. Introducción.

Dado que no se considera operativo el que todos los Procesadores (Computadores) estén conectados con todos a la vez, se introducen redes o retículas de interconexión donde cada nodo tiene conectado un Procesador (Computador) por medio de un dispositivo que envía o recibe mensajes de o hacia la red. Este dispositivo recibe el nombre de conmutador. Las Redes se representan como “grafos”, los arcos (o aristas) del grafo representan los enlaces de la red de comunicación y que unen los nodos, en los nodos se disponen los “conmutadores”. Todos los enlaces vamos a considerarlos “bidireccionales”, es decir, la información puede viajar en ambos sentidos. Además todas las redes están basadas en el uso de “conmutadores” que son los que unen los nodos “procesador-memoria” con los otros enlaces (a conmutadores) y son los encargados de “encaminar” los datos.

Hay que introducir una medida que nos permita realizar comparaciones de velocidad entre diferentes clusters, esta es la de “ancho de banda total de la red”, esta se define como el ancho de banda de cada enlace multiplicado por el número de enlaces.

Esta medida es la óptima pero no se adecua al peor de los casos, para ello se define el “ancho de banda de la biseción”. El cálculo es como sigue, se divide la máquina en dos partes, a cada lado se deja el mismo número de nodos y se suma el ancho de banda de los enlaces que atraviesan esta mitad imaginaria. La idea primordial es escoger la división imaginaria que de peor medida. Los programas paralelos están limitados por el enlace más débil.

Otra medida introducida es “diámetro de una red”, es el nº de arcos o aristas que hay entre los nodos más alejados, cuanto menor sea el diámetro menor será el retraso de los mensajes o lo que es lo mismo, mejor será su rendimiento.

8.4.2. Topologías.

Veamos a continuación varias topologías de Red:

- **Red en Anillo:** Arquitectura 1-D. Los mensajes van de O – E o de E – O. Hay nodos que no están conectados entre si por lo tanto sus mensajes deben pasar por otros nodos intermedios hasta llegar a su destino.

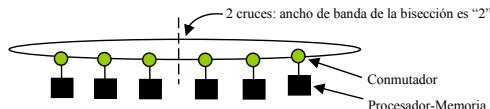


FIGURA 8. Red en Anillo

A diferencia de un Bus, el anillo permite varias transferencias en la misma unidad de tiempo. El ancho de banda total de la red, si hay P procesadores, es P veces el ancho de banda de cada enlace. El ancho de banda de la bisección es dos veces el ancho de banda del enlace.

- **Red en Cuadrícula:** Arquitectura 2-D. Los mensajes van de O – E o de E – O o de N – S o S - N. Si hay N procesadores por lado, requiere de N^2 nodos.

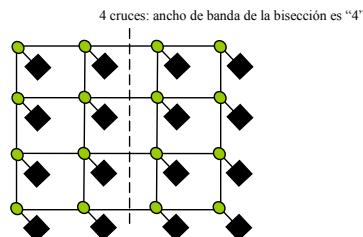


FIGURA 9. Red en Cuadrícula

- **Red en Cubo:** Arquitectura N-D. Es una arquitectura n -cubo con $2N$ nodos, que requiere N enlaces por commutador más uno para el procesador. Dispone de N enlaces vecinos.

La Figura 10 ilustra el caso de $N=3$.

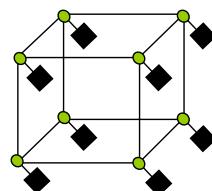


FIGURA 10. Red en Cubo con $N=3$

Con el fin de aumentar el rendimiento y fiabilidad de las redes en cuadrícula o cubo, suelen añadirse arcos que unen nodos extremos, de tal manera que disminuye el camino de enrutamiento. Esto lo podemos ver en la Figura 11:

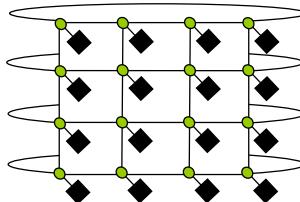


FIGURA 11. Red en Cuadrícula Toroide

Para aumentar la eficiencia de un cluster pueden realizarse topologías totalmente conectadas “Interconexión Total”, es decir, todos los nodos disponen de un enlace entre si. El número de enlaces requerido es $k(k - 1)/2$, siendo k el n° de nodos. El valor obtenido puede resultar inmanejable cuando crece el n° de nodos.

$$\begin{aligned} \text{Ancho de banda total:} & P \times (P - 1)/2 \quad \text{siendo } P \text{ el N° de Procesadores} \\ \text{Ancho de banda de la bisección:} & (P/2)^2 \end{aligned}$$

8.4.3. Otras Topologías - Redes Multietapa.

Las topologías de redes totalmente conectadas que acabamos de ver, presentan un gran rendimiento pero a costa de un incremento considerable en su coste. Una alternativa a colocar un Procesador (Computador) en cada nodo, consiste en disponer en los nodos de solamente el conmutador o bien en algunos nodos. Los conmutadores son más pequeños que el nodo procesador-memoria-conmutador y puede aumentarse la densidad de empaquetamiento de estas soluciones. La consecuencia directa es que se acortan las distancias y se incrementa el rendimiento. Estas redes se denominan “redes multietapa”. Se dan dos arquitecturas:

- **Crossbar o Totalmente Conectada:** Todos los procesadores están conectados con los otros por medio de una malla (*crossbar*) que une sus caminos con conmutadores. El n° de conmutadores necesarios es N^2 , siendo N el n° de procesadores. Se consiguen mejores empaquetamientos del conjunto procesador-conmutador.

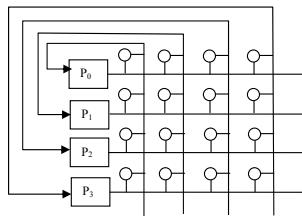


FIGURA 12. Red Crossbar

- **Red Omega:** Utiliza menos circuitería que la crossbar, $(N/2)\log 2N$.

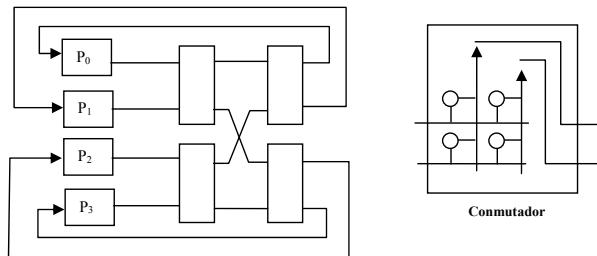


FIGURA 13. Red Omega

Tanto la red Crossbar como la Omega pueden presentar conexiones con memorias en vez de con los procesadores. Como ejemplo podemos ver la primera conectada a memorias:

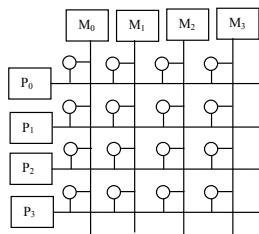


FIGURA 14. Red Crossbar con Memoria

8.4.4. Comunicadores.

Como ya se ha comentado, los comunicadores son los dispositivos encargados de encaminar los paquetes de información entre nodos. Además pueden realizar un almacenamiento del mensaje con el fin de descargar al procesador envante y mantener un protocolo de reenvíos en caso de errores de comunicación. El almacenamiento es FIFO. Cuando el comunicador recibe un mensaje, revisa el encabezamiento del mensaje con el fin de decidir

que ruta va a tomar el mensaje ya que el conmutador “conoce” la conexión de todos los nodos (procesador conectado a nodo). Además si el canal del conmutador está ocupado por otro mensaje, lo mantendrá en la FIFO hasta que se libere o bien decidirá si lo envía por otra ruta alternativa.

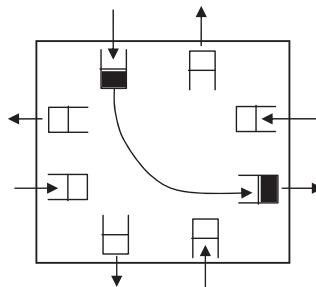


FIGURA 15. Comutador

Según como sea la arquitectura del comutador y su algoritmo de movimiento de mensajes internamente, un mensaje podría pasar a la FIFO del canal de salida adecuado y esperar a su emisión, dejando libre el canal de entrada para posible encolamiento de otro mensaje entrante.

8.5. LEY DE AMDAHL

En la introducción de este tema se comentó que la incorporación de técnicas de multiprocesamiento hacía indispensable el efectuar optimizaciones en la programación, para poder alcanzar mejores rendimientos.

En todos los programas se da una parte de código en forma “secuencial” y otra parte de código en forma “paralelizable”. La parte “secuencial” está normalmente asociada a tareas como “Iniciación” – “Lectura de Datos” – “Agrupación de Resultados”. La ley de Amdahl, aplicada al multiprocesamiento, dice lo siguiente: *“Para conseguir tasas altas de procesamiento paralelo, hay que conseguir tasas de procesamiento secuencial de parecida magnitud”*.

La expresión matemática de esta ley es de la siguiente forma:

(8.5.1)

$$T_{ejecución \text{ } después \text{ } de \text{ } la \text{ } mejora} = \frac{T_{ejecución \text{ } afectado \text{ } por \text{ } la \text{ } mejora}}{Cantidad \text{ } de \text{ } mejora} + T_{ejecución \text{ } no \text{ } afectado}$$

Otra manera de expresar esta ley es mediante el razonamiento siguiente:

Si disponemos de un programa que se ejecuta en un tiempo T , donde una fracción “ f ” está dedicada al tratamiento secuencial, tendremos que “ $1 - f$ ” es la fracción del tiempo dedicado al programa paralelizable. Si ahora introducimos “ n ” procesadores paralelos, tendremos que la fracción

paralelizable queda reducida a “ $(1 - f)/n$ ”. Por lo tanto el nuevo tiempo de ejecución será el siguiente:

$$(8.5.2) \quad f \cdot T + (i - f) \cdot \frac{T}{n}$$

La aceleración obtenida se expresa de la siguiente forma:

$$(8.5.3) \quad \text{Aceleración} = \frac{T_{\text{inicial}}}{T_{\text{nuevo}}} = \frac{T}{f \cdot T + (i - f) \cdot \frac{T}{n}} = \frac{n}{1 + (n - 1) \cdot f}$$

Vemos, por tanto, que la aceleración va a depender del tiempo de tratamiento secuencial.

8.6. PERSPECTIVA HISTÓRICA – CLASIFICACIÓN DE LOS COMPUTADORES

Como cierre de tema dedicado al multiprocesamiento, realizaremos una revisión histórica de cómo se han clasificado los computadores según su arquitectura.

Según distintas fuentes consultadas, es en 1966 o 1972 cuando Flynn, observando el nº de instrucciones paralelas y los flujos de datos, etiqueta a las computadoras del siguiente modo:

1. Flujo único de Instrucciones, único flujo de Datos : SISD
2. Flujo único de Instrucciones, múltiple flujo de Datos : SIMD
3. Flujo múltiple de Instrucciones, único flujo de Datos : MISD
4. Flujo múltiple de Instrucciones, múltiple flujo de Datos : MIMD

Existen máquinas híbridas entre estas categorías pero se utiliza el modelo de Flynn por ser sencillo y fácil de entender.

- **SISD:** Sistema monoprocesador clásico de flujo secuencial (arquitectura von Newman) sobre el que no se comentará nada en este capítulo
- **SIMD:** Este sistema opera con “Vectores” de Datos. Como solo dispone de un único flujo de Instrucciones, solo ejecuta una instrucción por unidad de tiempo pero esta instrucción opera sobre un conjunto de Datos a la vez.

Como ejemplo supongamos que una instrucción suma 64 números + 64 números. Envía 64 flujos de Datos a 64 ALUs para realizar 64 sumas en un solo ciclo.

Las instrucciones son mezcla de SISD y SIMD, es decir, instrucciones secuenciales e instrucciones parallelizables. El máximo rendimiento se consigue con bucles “*for*” y el peor con “*case*” o “*switch*”.

Una variante de SIMD lo constituyen los “Procesadores Vectoriales”. Trabajan con vectores de números y disponen de “Unidades funcionales” segmentadas que operan con unos pocos elementos del vector. Un SIMD opera con todos los elementos a la vez.

Hemos visto una arquitectura de un procesador con múltiples ALUs, en esta misma clasificación se da la arquitectura de “Arrays” de procesadores que trabajan con la misma instrucción y sobre vectores de datos. Una Unidad de Control es la encargada de alimentar a los procesadores con la instrucción.

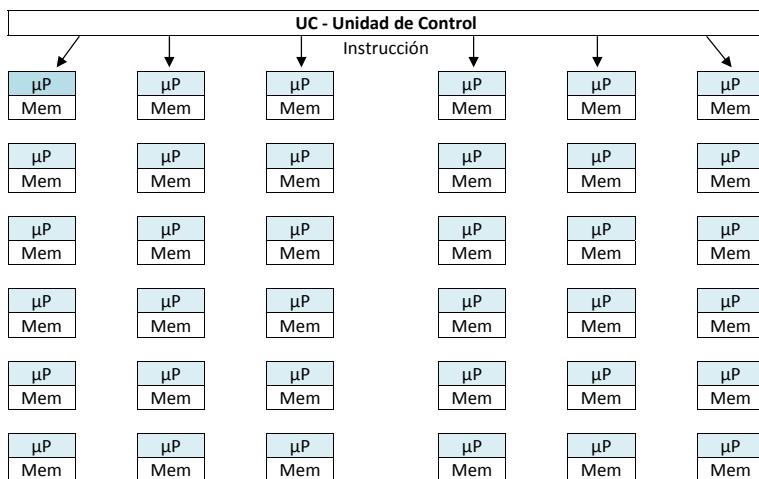


FIGURA 16. Array de 6x6 Procesador-Memoria ejecutando la misma instrucción

- **MISD:** Difícil imaginar un “Multiflujo de Instrucciones” con un solo Dato. No hay aplicaciones conocidas. Algún autor incluye a los Procesadores Sistólicos en esta categoría. No presenta interés.
- **MIMD:** Aquí se engloban las máquinas multiprocesadoras actuales. Hoy en día, los programadores tienden a escribir un único programa que será ejecutado en todos los procesadores. Se evita escribir programas distintos para los diferentes procesadores.

Uno de los primeros proyectos mejor conocidos se realizó en los años 70 en la Carnegie-Mellon University. Constaba de 16 PDP-11 (Digital Equipment Corporation) conectados por un “crossbar” a 16 unidades de memoria; este sistema se denominó C.m.m.p. Una evolución de esta máquina, la Cm*, incorporó multiprocesadores con “clusters” con memoria distribuida y tiempo de acceso no uniforme. No disponía de cachés.

La evolución posterior se marcó según los siguientes hitos:

- ▷ 1984: Primer multiprocesador (Synapse N+1) conectado por Bus y con Cachés con protocolo de “espionaje”.
- ▷ 1985: “Cosmic Cube”, reúne las dos direcciones de trabajo, “multicomputadores con paso de mensaje” y “multiprocesadores escalables con memoria compartida”, estructurando en “mallas” e “hipercubos”. Esta arquitectura redujo el coste de las interconexiones.
- ▷ 1986: Protocolos alternativos de coherencia de Cachés.

Bibliografía

- [1] A. S. Tanenbaum, Organización de Computadoras: Un Enfoque Estructurado, Prentice-Hall, 2000. 31, 184
- [2] D. A. Paterson, J. L. Hennessy, Estructura y Diseño de Computadores, Vol. 1, Morgan Kaufmann Publishers, Inc., 2000. 31
- [3] D. A. Paterson, J. L. Hennessy, Estructura y Diseño de Computadores, Vol. 2, Morgan Kaufmann Publishers, Inc., 2000. 31
- [4] D. A. Paterson, J. L. Hennessy, Estructura y Diseño de Computadores, Vol. 3, Morgan Kaufmann Publishers, Inc., 2000. 31
- [5] J. Hennessy, D. Patterson, Arquitectura de Computadores: Un Enfoque Cuantitativo, 2002. 32, 150
- [6] M. Morris-Mano, C. R. Kime, Fundamentos de Diseño Lógico y Computadoras, Prentice-Hall, 2005. 32
- [7] I. Englander, Arquitectura Computacional, Cecsa, 2002. 32
- [8] C. N. A. Program, HP Fundamentos de Tecnología de la Información: Hardware y Software para PC, Cisco Systems, 2005. 32
- [9] B. Brey, Los Microprocesadores Intel, Prentice-Hall, 2000. 32
- [10] M. de Miguel, T. Higuera, Arquitectura de Ordenadores, Ra-Ma, 2000. 32
- [11] L. Null, J. Lobur, al., The essentials of computer organization and architecture, Jones & Bartlett, 2014. 32
- [12] X. Alcober, Buses normalizados para tarjetas basadas en uP, Automática e Instrumentación, 1988. 32, 184
- [13] P. de Miguel-Anasagasti, Fundamentos de los Computadores, 9th Edition, 2004. 32, 142
- [14] Computer organization and design, Universidad de Florida.
URL <https://www.cise.ufl.edu/~mssz/CompOrg/CDA-pipe.html> 75

Índice alfabético

A

Assembler, 105
Assembly Language, 86

B

Buses, 169
Ancho de Bus, 170
Arbitraje de Bus, 173
Asíncronos, 172
Características, 170
FIREWIRE, 191, 194
Futurebus, 182
G64/G96, 182
Introducción, 169
ISA, 177
Multibus I-II, 182
NuBus, 182
Operaciones de Bus, 175
PCI, 177, 178
Síncronos, 171
STD, 182
STEBus, 182
Temporización del Bus, 171
USB, 191
VME, 182

C

Cintas Magnéticas, 217
Código de Caracteres, 239
ASCII, 239
UNICODE, 241
Computación paralela, Clusters, 253
Computación paralela, Comutadores, 257
Computación paralela, Interconexión, 246
Multiprocesadores conectados por Red, 250

Multiprocesadores conectados por un solo Bus, 246

Computación paralela, Ley de Amdahl, 259

Computación paralela, Perspectiva Histórica y Clasificación, 260

Computación paralela, Topologías de Red, 254

Redes Multietapa, 256
Topologías, 254

Computadora en canalización, 68

Ciclo de ejecución, 68

Ejecución en canalización, 68
Riesgo de datos, 71

Solución hardware, 72
Solución software, 71

CPU, 32

Arquitectura Harvard, 32

Arquitectura von Newman, 33
vonNeumann vs. Harvard, 34

D

Discos: Blu-ray, 215

Discos: CD-ROM, 209

Grabables, 210

Regrabables, 212

Discos: DVD, 212

Discos: Magnéticos, 200

Disco duro, 200

Discos flexibles, 204

IDE, 204

RAID, 206

SCSI, 205

Discos: Magneto-Ópticos, 213

Dispositivos de E/S

Modems, 236

Ratones, 229

Dispositivos de E/S: Impresoras, 231

Color, 235

- Monocromáticas, 232
- Dispositivos de E/S: Terminales, 223
 - CRT, 224
 - LCD, 224
 - Terminal Carácteres, 226
- E**
 - Ejercicios CPU, 36
 - Ensamblador, 105
 - Enlazado Dinámico, 112
 - Enlazado y Carga, 109
 - Estructura de un Módulo Objeto, 108
 - Primera pasada, 105
 - Segunda pasada, 107
- F**
 - Formato de instrucción, 75
- G**
 - Glosario, Términos y Acrónimos, 25
- I**
 - Interrupciones, 81
- L**
 - Lenguaje Ensamblador, 86
 - Formato de las instrucciones, 89
 - Macros, 92
 - Pseudoinstrucciones, 91
- M**
 - Memorias, Asociativa, 150
 - Concepto, 150
 - Estructura de una CAM, 150
 - Memorias, Caché, 133
 - Concepto de memoria caché, 134
 - Funciones de correspondencia
 - Correspondencia asociativa, 145
 - Correspondencia asociativa por conjuntos, 146
 - Correspondencia directa, 142
 - Parámetros de diseño, 137
 - Algoritmos de sustitución, 149
 - Contenido de la caché, 139
 - Ejemplos de memorias caché, 150
 - Estrategia de escritura de datos -
 - Actualización, 139
 - Fuentes de fallos en la memoria caché, 148
 - Funciones de correspondencia, 141
 - Número de cachés, 138
 - Tamaño de la memoria caché, 138
 - Tamaño del bloque, 138
 - Principio de Localidad, 133
 - Memorias, Características, 116
 - Acceso, 118
 - Capacidad, 118
 - Físico, 120
 - Físicos, 120
 - Localización, 116
 - Organización, 122
 - Velocidad, 119
 - Memorias, Compartida, 153
 - Concepto, 153
 - Memorias de doble puerta, 153
 - Memorias, Jerarquías, 125
 - Memorias, Otras, 158
 - Dinámicas, 163
 - BEDO RAM, 164
 - DDR RAM, 165
 - EDO RAM, 163
 - FPM RAM, 163
 - PBSRAM, 166
 - RDRAM, 165
 - SDRAM, 164
 - SGRAM, 166
 - SLDRAM, 166
 - Tabla resumen, 166
 - VRAM, 166
 - Entrelazadas, 159
 - Introducción, 158
 - Memorias, Semiconductora, 125
 - Diseño, 129
 - Tipos de memorias, 125
 - Memorias, Virtual, 154
 - Concepto, 154
 - Diseño, 155
 - Gestión, 157
 - Modos de direccionamiento, 77
 - P**
 - Pendrives, 218
 - R**
 - Ruta de Datos, 37
 - Circuito aritmético, 39
 - Circuito lógico, 38
 - Ruta de Datos en canalización, 52
 - Unidad aritmético-lógica, 37
 - Ruta de datos
 - Bloque de registros, 47
 - Desplazador, 43
 - Instrucciones de Control de Secuencia, 49
 - Palabra de control, 44

S

Subrutinas, 97

 Gestión de subrutinas, 100

 Bloque de activación, 102

U

Unidad aritmético-lógica, 37

Unidad de Control, 55

 Control cableado, 57

 CPU, 59

 Decodificador, 59

Control microprogramado

 CPU, 65

Control microprogramado de ciclo

 múltiple, 61

 Secuenciamiento explícito, 62

 Secuenciamiento implícito, 63

Formato de Instrucción, 55