UNIVERSIDAD DE GRANADA

Tecnología y Organización de Computadores

Temario de la asignatura

Ismael Sallami Moreno

Recursos Ingeniería Informática y Ade

Licencia

Tecnología y Organización de Computadores

Ismael Sallami Moreno

https://elblogdeismael.github.io

Índice general

Capítulo 1

Introducción

Tema 1 del programa. Los conceptos básicos, la estructura funcional de un computador, cómo se representan los números y en qué niveles se describe una máquina.

1.1 Conceptos básicos

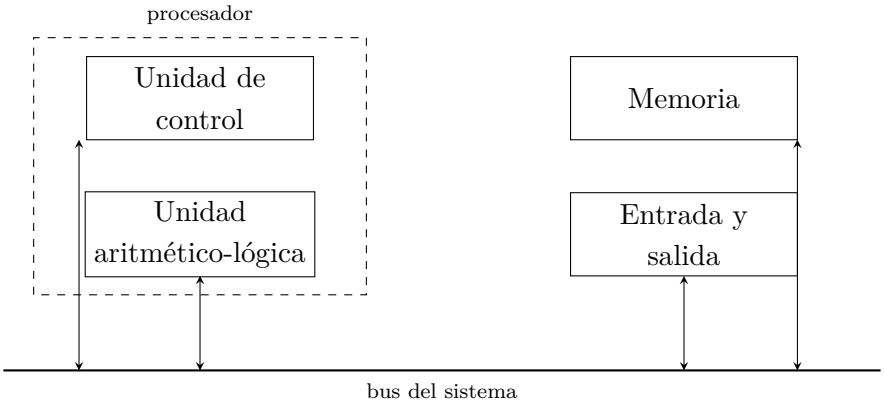
Término	Qué es
Dato	representación de un hecho o una magnitud
Información	dato interpretado en un contexto
Algoritmo	secuencia finita de pasos que resuelve un problema
Programa	algoritmo escrito en un lenguaje que la máquina ejecuta
Computador	máquina que ejecuta programas sobre datos

La definición de computador esconde la idea que lo hace universal, y es la de von Neumann: **el programa se guarda en la misma memoria que los datos**. Antes de eso, cambiar de tarea significaba recablear la máquina.

Las consecuencias de esa decisión llegan hasta hoy:

- Un programa puede leer y modificar otro programa, y de ahí compiladores, cargadores y sistemas operativos.
- Instrucciones y datos compiten por el mismo bus, que es el cuello de botella clásico de la arquitectura.
- Una instrucción es una cadena de bits sin nada que la distinga de un dato: lo que la convierte en instrucción es que el procesador la lea en la fase de captación.

1.2 Estructura funcional



Unidad	Función
Unidad de control	interpreta las instrucciones y gobierna al resto
Unidad aritmético-lógica	hace las operaciones sobre los datos
Memoria	guarda instrucciones y datos
Entrada y salida	comunica con el exterior
Buses	transportan datos, direcciones y señales de control

Los tres buses cumplen papeles distintos, y su anchura tiene efectos medibles:

Bus	Qué lleva	Su anchura determina
Datos	la información	cuántos bits se transfieren a la vez
Direcciones	la posición	cuánta memoria se puede direccionar
Control	las señales de mando	qué operaciones existen

Con n líneas de dirección se direccionan 2^n posiciones. De ahí que 32 bits limiten a 4 GB, que es la razón concreta por la que se pasó a 64.

1.3 Representación de datos numéricos

1.3.1 Sistemas de numeración

En base b, un número se escribe como suma de potencias:

$$N = \sum_{i=-m}^{n-1} d_i \, b^i, \qquad 0 \le d_i < b$$

Base	Dígitos	Por qué se usa
2	0, 1	los dos estados físicos de un circuito
8	0 a 7	tres bits por dígito
10	0 a 9	la humana

Base	Dígitos	Por qué se usa
16	0 a 9, A a F	cuatro bits por dígito, compacta

Que 8 y 16 sean potencias de 2 es lo que hace la conversión inmediata: se agrupan los bits de tres en tres o de cuatro en cuatro. Con base 10 hace falta dividir repetidamente.

Ejemplo 1.1 . $1011\,0110_2$ agrupado de cuatro en cuatro es $\texttt{B6}_{16}$, y de tres en tres, partiendo por la derecha, $10\,110\,110_2 = 266_8$. La conversión a decimal, en cambio, exige sumar $128+32+16+4+2 = 182$.

1.3.2 Enteros con signo

Cuatro convenios, y solo uno sobrevive:

Convenio	Cómo	Problema
Signo y magnitud	un bit de signo	dos ceros; sumar exige comparar magnitudes
Complemento a 1	se invierten los bits	dos ceros; hay que sumar el acarreo final
Complemento a 2	complemento a 1 más uno	ninguno de los anteriores
Exceso a 2^{n-1}	se suma una constante	solo se usa en el exponente del coma flotante

El complemento a 2 gana porque la resta desaparece. Restar es sumar el opuesto, y el mismo circuito sumador sirve para las dos operaciones. Además solo hay un cero.

Con n bits, el rango es $[-2^{n-1},\ 2^{n-1}-1]$:

n	Rango
8	-128 a 127
16	$-32\,768$ a $32\,767$
32	$\pm 2{,}1 \times 10^9$ aproximadamente
64	$\pm 9{,}2 \times 10^{18}$ aproximadamente

El rango es **asimétrico**: hay un negativo más que positivos, porque el cero ocupa un sitio del lado positivo. De ahí que el opuesto del mínimo no se pueda representar, y que negar -128 en 8 bits devuelva -128.

Nota 1.1 . El desbordamiento en complemento a 2 se detecta con una regla sencilla: ocurre cuando se suman dos números del mismo signo y el resultado sale con el signo contrario. Sumar números de signos distintos nunca desborda.

1.3.3 Coma flotante

Para cubrir un rango enorme con un número fijo de bits se sacrifica precisión:

$$N = (-1)^s \times 1{,}M \times 2^{E-sesgo}$$

Formato	Signo	Exponente	Mantisa	Cifras decimales
Simple (32 bits)	1	8	23	unas 7
Doble (64 bits)	1	11	52	unas 16

El **1 implícito** de la parte entera no se guarda: como todo número normalizado empieza por 1, se da por supuesto y se gana un bit de precisión.

Tres consecuencias que hay que conocer antes de usar coma flotante para algo serio:

- **La mayoría de los decimales no son exactos.** 0,1 en binario es periódico, así que $0{,}1 + 0{,}2 \neq 0{,}3$ exactamente. Comparar dos flotantes con igualdad es un error; se compara la diferencia contra una tolerancia.
- **La suma no es asociativa.** Sumar una lista de menor a mayor y de mayor a menor da resultados distintos, porque al sumar un número pequeño a uno grande el pequeño se pierde al alinear exponentes.
- **La densidad no es uniforme.** Hay tantos representables entre 1 y 2 como entre 1024 y 2048, así que la precisión absoluta empeora al crecer el número.

1.3.4 Otras representaciones

Representación	Para qué
BCD	cada dígito decimal en cuatro bits; cálculo financiero exacto
ASCII	caracteres en 7 u 8 bits
Unicode y UTF-8	todos los sistemas de escritura, con longitud variable
Códigos detectores	paridad, para detectar un error de transmisión
Códigos correctores	Hamming, para detectar y corregir

El **BCD** aparece justo por el primer problema del coma flotante: en contabilidad no se admite que sumar céntimos dé un error de redondeo, así que se representa en decimal aunque cueste más.

1.4 Niveles conceptuales de descripción

Un computador se describe en varios niveles, cada uno con su vocabulario. Ninguno es el verdadero: cada uno responde preguntas distintas.

Nivel	Elementos	Pregunta que responde
Aplicación	programas	qué hace la máquina para el usuario
Lenguaje de alto nivel	sentencias, tipos, funciones	cómo se expresa el algoritmo
Ensamblador y máquina	instrucciones, registros	qué sabe hacer el procesador

Nivel	Elementos	Pregunta que responde
Transferencia entre registros	registros, buses, señales de control	cómo se ejecuta una instrucción
Lógico	puertas, biestables	cómo se construye cada operación
Circuito	transistores, resistencias	de qué está hecho
Físico	materiales, geometría	cómo se fabrica

Esta asignatura recorre los tres del medio: el nivel lógico en los temas 3 y 4, y el de transferencia entre registros en el tema 5, para llegar al computador sencillo completo.

Cada nivel oculta el de abajo, y esa es la razón de que se pueda programar sin saber electrónica. Pero la abstracción tiene fugas: un programa que recorre una matriz por columnas es más lento que por filas, y eso no se explica en ningún nivel por encima del de memoria.

1.5 Sistemas analógicos y digitales

	Analógico	Digital
Valores	continuos	discretos
Ruido	se acumula etapa tras etapa	se elimina si está bajo el margen
Precisión	la del componente	la del número de bits
Almacenamiento	se degrada con las copias	copia exacta indefinida
Diseño	específico de cada problema	general y programable

La segunda fila es la razón de todo lo demás. En un sistema digital, una señal que se ha ensuciado se regenera al pasar por la puerta siguiente, porque la puerta la reinterpreta como 0 o 1 y emite un valor limpio. En uno analógico, cada etapa suma su propio ruido y el error crece sin remedio.

El precio es la **cuantificación**: al convertir una magnitud continua en un número se pierde lo que hay entre dos niveles. Con n bits hay 2^n niveles, y el error relativo baja a la mitad por cada bit añadido.

$$\mathrm{SNR} \approx 6{,}02\,n + 1{,}76 \text{ dB}$$

Por eso el audio de 16 bits da unos 98 dB, suficiente para el oído, y el profesional usa 24.

1.6 Ejercicios

Ejercicio 1.6.1. Representar -45 en complemento a 2 con 8 bits, y comprobar el resultado sumándole 45.

Solución 1.6.1. $45 = 0010\,1101_2$. Invirtiendo, 1101 0010; sumando uno, 1101 0011, que es -45. Al sumarle 0010 1101 da 1 0000 0000: el acarreo de salida se descarta y queda 0000 0000, el cero. Que ese acarreo se descarte sin más es justo lo que hace cómodo el convenio.

Ejercicio 1.6.2. ¿Por qué $0{,}1 + 0{,}2$ no da exactamente 0,3 en coma flotante?

Solución 1.6.2. Porque 0,1 y 0,2 no tienen representación binaria finita: en base 2 son fracciones periódicas y se almacenan redondeadas. La suma de los dos valores redondeados no coincide con el valor redondeado de 0,3. Comparar flotantes con igualdad es incorrecto; se compara $|a - b| < \varepsilon$ con una tolerancia acorde a la magnitud.

Ejercicio 1.6.3. Un bus de direcciones tiene 20 líneas y el de datos 16. ¿Cuánta memoria direcciona y en qué unidades?

Solución 1.6.3. $2^{20} = 1\,048\,576$ posiciones. Si cada posición almacena una palabra de 16 bits, son 2 MB; si almacena un byte, 1 MB. La anchura del bus de datos no cambia cuántas posiciones hay, solo cuánto se transfiere en cada acceso, y por eso hay que decir siempre qué unidad se direcciona.

Los conceptos de este tema están desarrollados en Prieto y Prieto, 2010, Stallings, 2022 y Díaz Ruiz et al., 2009.

Capítulo 2

Unidades funcionales de un computador

Tema 2 del programa. El procesador, la memoria, los periféricos, cómo se conectan y qué parámetros miden lo que rinde el conjunto.

2.1 El procesador

Ejecuta instrucciones. Sus partes:

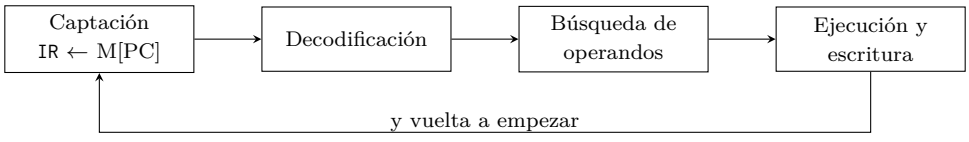
Parte	Función
Unidad de control	decodifica la instrucción y genera las señales de mando
Unidad aritmético-lógica	opera sobre los datos
Banco de registros	almacenamiento rápido de trabajo
Registros de estado	banderas del resultado y modo de ejecución

2.1.1 Registros

Registro	Contiene
Contador de programa (PC)	dirección de la instrucción siguiente
Registro de instrucción (IR)	la instrucción en curso
Registro de dirección de memoria (MAR)	dirección del acceso en curso
Registro de datos de memoria (MDR)	dato leído o por escribir
Acumulador y registros generales	operandos y resultados
Registro de estado	acarreo, cero, signo, desbordamiento

MAR y MDR son la interfaz con la memoria y aparecerán en cada micro-operación del tema 5: **todo acceso a memoria pasa por esos dos registros**, y por eso una lectura son siempre dos pasos, poner la dirección y recoger el dato.

2.1.2 El ciclo de instrucción



Al final de la captación el contador de programa ya apunta a la instrucción siguiente. Ese incremento anticipado es lo que hace que un salto se implemente simplemente escribiendo un valor nuevo en el PC.

2.1.3 El juego de instrucciones

Grupo	Ejemplos
Transferencia	cargar, almacenar, mover
Aritméticas	sumar, restar, multiplicar
Lógicas	AND, OR, NOT, desplazamientos
Control	saltos incondicionales y condicionales, llamadas
Entrada y salida	leer y escribir puertos

Una instrucción tiene **código de operación** y **operandos**, y la forma de nombrar los operandos se llama modo de direccionamiento:

Modo	Dónde está el operando
Inmediato	en la propia instrucción
Directo	en la dirección que indica la instrucción
Indirecto	en la dirección que hay en esa dirección
De registro	en un registro
Indexado	en base más desplazamiento
Relativo al PC	a una distancia de la instrucción actual

El **indexado** es el que permite recorrer un vector: la base fija el comienzo y el índice avanza. Y el **relativo al PC** es el que hace que un programa funcione esté cargado donde esté, porque los saltos no citan direcciones absolutas.

2.1.4 CISC y RISC

	CISC	RISC
Número de instrucciones	muchas	pocas
Longitud	variable	fija
Acceso a memoria	desde muchas instrucciones	solo carga y almacenamiento
Ciclos por instrucción	variable	uno, con segmentación
Complejidad	en el hardware	en el compilador

La distinción histórica se ha difuminado: los procesadores actuales tienen juego CISC por compatibilidad y por dentro traducen a micro-operaciones de tipo RISC. Pero la idea de fondo sigue viva, y es que **un juego regular es más fácil de segmentar**.

2.2 La memoria

Un conjunto de posiciones numeradas. Los parámetros que la caracterizan:

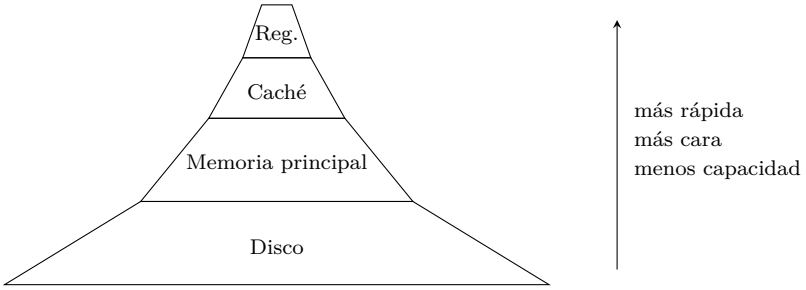
Parámetro	Qué mide
Capacidad	cuántos bits guarda
Tiempo de acceso	desde la petición hasta el dato
Tiempo de ciclo	mínimo entre dos accesos consecutivos
Ancho de banda	bits por segundo
Coste por bit	lo que cuesta la capacidad
Volatilidad	si pierde el contenido al apagar

2.2.1 Tipos

Tipo	Escritura	Volátil	Uso
SRAM	sí	sí	caché; rápida y cara
DRAM	sí	sí, y necesita refresco	memoria principal
ROM	no	no	arranque
Flash	por bloques	no	almacenamiento

La **DRAM guarda cada bit en un condensador que se descarga**, así que hay que reescribirlo periódicamente. Ese refresco consume ciclos y es la razón de que sea más lenta que la SRAM, que usa biestables y no necesita refresco pero gasta seis transistores por bit en vez de uno.

2.2.2 La jerarquía



Nivel	Acceso	Capacidad típica
Registros	menos de 1 ns	cientos de bytes
Caché L1	1 a 2 ns	decenas de kilobytes
Caché L2 y L3	5 a 20 ns	de cientos de kilobytes a decenas de megabytes
Memoria principal	50 a 100 ns	gigabytes
Disco	microsegundos o milisegundos	terabytes

La jerarquía funciona por el **principio de localidad**: temporal, porque lo usado recientemente se vuelve a usar; y espacial, porque se accede a posiciones cercanas. Los dos son observaciones empíricas sobre cómo son los programas reales, no teoremas, y por eso un programa que los viole —un recorrido aleatorio de un vector enorme— no aprovecha la caché.

El tiempo medio de acceso con dos niveles:

$$t_{medio} = t_{caché} + (1 - h)\, t_{penalización}$$

con h la tasa de aciertos. La fórmula explica por qué la tasa importa tanto: con $t_{caché} = 2$ ns y penalización de 100 ns, pasar de $h = 0{,}95$ a $h = 0{,}99$ baja el tiempo medio de 7 ns a 3 ns, más del doble de velocidad por cuatro puntos de acierto.

2.3 Periféricos y entrada/salida

Clase	Ejemplos
Entrada	teclado, ratón, sensores, escáner
Salida	pantalla, impresora, altavoces
Almacenamiento	disco, SSD, memorias extraíbles
Comunicación	tarjeta de red, módem

Un **controlador** adapta el periférico al bus, y hace tres cosas: convierte el formato, sincroniza velocidades muy distintas y detecta errores.

Las tres técnicas de gobierno, ya vistas en su lógica:

Técnica	Quién copia los datos	Coste
E/S programada	el procesador, consultando	lo ocupa entero
Por interrupciones	el procesador, avisado	una interrupción por transferencia
Acceso directo a memoria	el controlador de DMA	una interrupción por bloque

2.4 Estructuras de interconexión

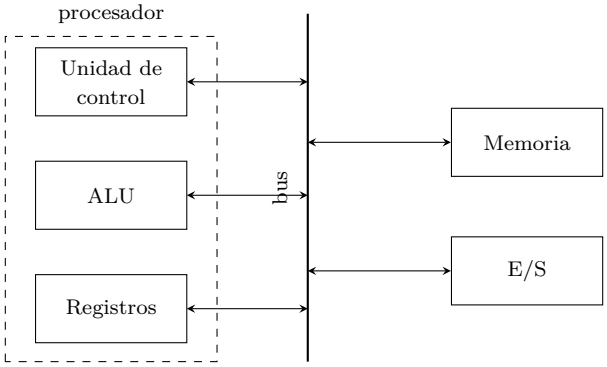
Estructura	Cómo	Ventaja	Problema
Bus único	todos comparten las mismas líneas	simple y barata	se convierte en cuello de botella
Buses jerárquicos	uno rápido cerca del procesador y otros lentos	separa velocidades	necesita puentes
Punto a punto	enlaces dedicados	máximo ancho de banda	más líneas y más coste
Conmutada	una matriz conecta pares	muchas transferencias simultáneas	cara

En un bus compartido hace falta **arbitraje**, porque solo un maestro puede transmitir a la vez. Puede ser centralizado, con un árbitro que concede el bus, o distribuido, con las unidades poniéndose de acuerdo.

La evolución real ha ido del bus único a los enlaces punto a punto, y la causa es física: a frecuencias altas, un bus paralelo compartido sufre desajustes entre líneas que impiden subir la velocidad. Por eso los buses modernos son serie y diferenciales, lo contrario de lo que la intuición sugiere.

2.5 Un computador sencillo a nivel de bloques

Con lo anterior ya se puede dibujar una máquina completa, que es la que el tema 5 detalla:



Su funcionamiento es el ciclo de instrucción del principio, y sus señales de control son lo que el tema 5 genera.

2.6 Prestaciones

Cómo se mide lo que rinde una máquina, y por qué casi todas las medidas fáciles engañan.

Parámetro	Definición
Frecuencia de reloj f	ciclos por segundo
Periodo T	$1/f$
CPI	ciclos por instrucción, en media
MIPS	millones de instrucciones por segundo
Tiempo de ejecución	lo único que importa de verdad

$$T_{ejecución} = \frac{N_{instrucciones} \times \text{CPI}}{f}$$

La ecuación tiene tres factores y los tres se pueden tocar:

Factor	Depende de
Número de instrucciones	el algoritmo, el compilador y el juego de instrucciones
CPI	la organización del procesador
Frecuencia	la tecnología de fabricación

Nota 2.1 . **Comparar máquinas por MIPS es engañoso**, porque una instrucción de un juego no hace el mismo trabajo que una de otro. Un procesador con instrucciones muy simples ejecuta más por segundo y puede tardar más en el mismo programa. La única comparación válida es el tiempo de ejecución sobre el mismo programa.

2.6.1 La ley de Amdahl

Si se mejora una parte que ocupa una fracción p del tiempo, con un factor k:

$$S = \frac{1}{(1-p) + \dfrac{p}{k}}$$

p	k	Ganancia total
0,5	2	1,33
0,5	∞	2
0,9	10	5,26
0,9	∞	10

La lectura importa: **la parte que no se mejora pone un techo**. Con el 10 % del tiempo fuera de la mejora, la ganancia máxima es 10 aunque el resto se haga infinitamente rápido. Es el argumento que gobierna cualquier decisión de optimización, y el que dice que hay que medir antes de optimizar.

2.7 Ejercicios

Ejercicio 2.7.1. Un procesador a 2 GHz ejecuta un programa de 10^9 instrucciones con CPI medio 1,5. ¿Cuánto tarda?

Solución 2.7.1. $T = (10^9 \times 1{,}5)/(2 \times 10^9) = 0{,}75$ segundos. Su tasa es de 1333 MIPS, cifra que no sirve para compararlo con un procesador de otro juego de instrucciones: para eso hay que ejecutar el mismo programa en los dos y medir segundos.

Ejercicio 2.7.2. Una caché tiene tiempo de acceso 2 ns y la memoria principal 80 ns. ¿Qué tasa de aciertos hace falta para que el tiempo medio no pase de 5 ns?

Solución 2.7.2. $2 + (1-h) \cdot 80 \leq 5$, de donde $(1-h) \leq 0{,}0375$ y $h \geq 0{,}9625$, es decir un 96,25 %. Es una exigencia alta y sin embargo habitual: las cachés reales superan el 95 % gracias al principio de localidad, y por eso funcionan.

Ejercicio 2.7.3. Un programa dedica el 25 % del tiempo a una rutina. Se optimiza para que sea cuatro veces más rápida. ¿Cuánto gana el programa entero?

Solución 2.7.3. $S = 1/(0{,}75 + 0{,}25/4) = 1/0{,}8125 = 1{,}23$: un 23 % de mejora. Aunque la rutina se hiciese instantánea, el techo sería $1/0{,}75 = 1{,}33$. Optimizar donde no está el tiempo produce ganancias pequeñas por muy espectacular que sea la mejora local.

La organización de las unidades funcionales y el análisis de prestaciones están en Stallings, 2022 y Hamacher et al., 2003, y su tratamiento con problemas en Prieto y Prieto, 2010 y Díaz Ruiz et al., 2009.

Capítulo 3

Estudio de sistemas combinacionales

Tema 3 del programa. Qué es un sistema combinacional, cómo se analiza, cómo se diseña y cuáles son los bloques estándar que se repiten en todos los circuitos.

3.1 Concepto

Definición 3.1 (Sistema combinacional). Circuito cuyas salidas dependen únicamente de la combinación de valores presente en sus entradas, sin memoria de lo ocurrido antes.

La diferencia con un sistema secuencial es esa: aquí la misma entrada produce siempre la misma salida. Un sumador es combinacional; un contador no, porque lo que saca depende de cuántos pulsos ha recibido.

3.2 Álgebra de conmutación

Las tres operaciones y sus símbolos:

Operación	Notación	Resultado
Producto lógico	$a \cdot b$	1 si los dos son 1
Suma lógica	$a + b$	1 si alguno es 1
Complemento	$\bar{a}$	invierte

Las propiedades que se usan al simplificar:

Propiedad	Expresión
Identidad	$a + 0 = a$, $a \cdot 1 = a$
Elemento nulo	$a + 1 = 1$, $a \cdot 0 = 0$
Idempotencia	$a + a = a$, $a \cdot a = a$
Complemento	$a + \bar{a} = 1$, $a \cdot \bar{a} = 0$
Conmutativa	$a + b = b + a$
Asociativa	$(a + b) + c = a + (b + c)$
Distributiva	$a(b + c) = ab + ac$, y también
	$a + bc = (a + b)(a + c)$
Absorción	$a + ab = a$, $a(a + b) = a$
De Morgan	$\overline{a + b} = \bar{a}\,\bar{b}$, $\overline{ab} = \bar{a} + \bar{b}$

Propiedad	Expresión

La segunda forma de la distributiva no tiene equivalente en el álgebra ordinaria, y es la que más simplificaciones permite. Y **De Morgan** es la ley práctica del tema: deja convertir cualquier circuito a puertas de un solo tipo, que es lo que se fabrica.

3.3 Formas canónicas

Una función de conmutación se describe con su tabla de verdad, y de ahí salen dos expresiones estándar:

Forma	Cómo se construye	Se llama
Suma de productos	un mintérmino por cada fila con salida 1	primera forma canónica
Producto de sumas	un maxtérmino por cada fila con salida 0	segunda forma canónica

Ejemplo 3.1 . Sea $f(a, b, c)$ que vale 1 en las filas 1, 3, 5 y 6.

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

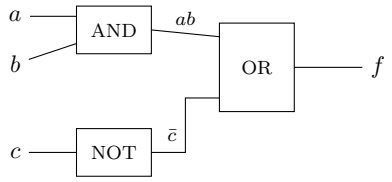
Su suma de productos es $f = \bar{a}\bar{b}c + \bar{a}bc + a\bar{b}c + ab\bar{c}$, que se abrevia $f = \sum m(1, 3, 5, 6)$.

La forma canónica siempre existe y casi nunca es la mejor: en el ejemplo, los tres primeros términos se combinan en uno solo, c, porque cubren todas las combinaciones de a y b. Simplificar es justo eso.

3.4 Análisis de sistemas combinacionales

Analizar es partir del circuito y obtener qué hace. El procedimiento:

1. Etiquetar las salidas de cada puerta con su expresión.
2. Propagar hasta la salida final.
3. Simplificar la expresión.
4. Construir la tabla de verdad, para describir el comportamiento sin ambigüedad.



Para este circuito, $f = ab + \bar{c}$. La tabla de verdad se rellena evaluando las ocho combinaciones, y con ella el análisis está cerrado.

3.5 Simplificación

3.5.1 Mapas de Karnaugh

Una tabla de verdad reordenada de forma que **casillas contiguas difieran en un solo bit**. Esa disposición convierte la simplificación algebraica en un problema visual: agrupar unos adyacentes.

$a\backslash bc$	00	01	11	10
0	0	1	1	0
1	0	1	1	0

agrupación de 4: $f = c$

Las reglas de agrupación:

Regla	Por qué
Los grupos son de 2^k casillas	solo así desaparecen variables limpiamente
Cuanto mayor el grupo, menos variables quedan	un grupo de 2^k elimina k variables
Los grupos pueden solaparse	cubrir un uno dos veces no cambia la función
El mapa se cierra por los bordes	la primera columna es adyacente a la última
Hay que cubrir todos los unos	si no, la función cambia

La cuarta regla es la que más agrupaciones se pierde por olvido. El mapa es un toro: la casilla de arriba a la izquierda y la de abajo a la derecha son adyacentes en un mapa de cuatro variables.

3.5.2 Indiferencias

Cuando una combinación de entrada no puede darse, su salida es indiferente y se marca con X. Al agrupar, cada X se toma como 1 si conviene y se ignora si no. Es gratis y suele simplificar mucho.

Nota 3.1 . Aprovechar una indiferencia asigna un valor concreto a esa entrada, y si la combinación acaba ocurriendo el circuito hace algo. Marcar como indiferente lo que solo es «poco probable» es un error clásico de diseño.

Para más de cuatro o cinco variables el mapa deja de ser manejable y se usa el método de **Quine-McCluskey**, que hace lo mismo de forma tabular y sistemática, o directamente una herramienta de síntesis.

3.6 Diseño de sistemas combinacionales

El procedimiento completo:

1. **Especificar** el problema con palabras, sin ambigüedad.
2. Definir entradas y salidas, y **codificarlas** en binario.
3. Construir la **tabla de verdad**.
4. Obtener la expresión y **simplificarla**.
5. **Implementar** con las puertas disponibles.
6. **Verificar** con la tabla de verdad original.

Los dos pasos que se saltan son el 2 y el 6, y son los que más cuestan. La codificación decide la complejidad del circuito: con la codificación adecuada un problema sale en tres puertas y con otra en quince.

Ejemplo 3.2 . Un circuito recibe tres bits y saca 1 si hay mayoría de unos. La tabla tiene un 1 en las filas con dos o tres unos, y la función simplificada es

$$f = ab + ac + bc$$

Tres puertas AND y una OR. Es el circuito de voto por mayoría, base de los sistemas tolerantes a fallos por triplicación.

3.6.1 Implementación con un solo tipo de puerta

NAND y NOR son **funcionalmente completas**: con cualquiera de las dos se construye todo. Es lo que se aprovecha en la fabricación, donde interesa un solo tipo de celda.

Con NAND	Cómo
NOT	las dos entradas unidas
AND	NAND seguida de NOT
OR	De Morgan: $a + b = \overline{\bar{a}\,\bar{b}}$

La conversión de una suma de productos a solo NAND es mecánica: se sustituyen todas las AND y la OR final por NAND, y el resultado es equivalente por De Morgan.

3.6.2 Riesgos

Un circuito correcto en la tabla de verdad puede producir un pulso espurio al cambiar la entrada, porque los caminos tienen retardos distintos. Se llama **riesgo** o azar.

Tipo	Cuándo	Solución
Estático	la salida debería quedarse igual y da un pulso	añadir un término redundante
Dinámico	la salida cambia y oscila antes de estabilizarse	rediseñar la red

El término redundante es un grupo del mapa de Karnaugh que solapa dos grupos adyacentes. Es **lógicamente innecesario y eléctricamente imprescindible**, y es la razón de que un circuito minimizado no sea siempre el mejor circuito.

En sistemas síncronos el problema desaparece si se espera a que todo se estabilice antes del flanco de reloj, y esa es una de las razones de que casi todo se diseñe síncrono.

3.7 Componentes combinacionales estándar

3.7.1 Codificador y decodificador

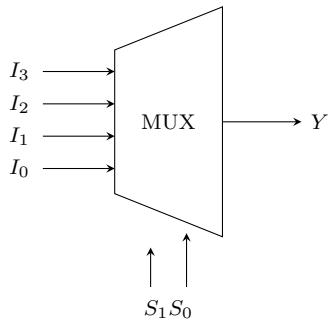
Bloque	Entradas	Salidas	Qué hace
Decodificador	n	2^n	activa la salida cuyo número indican las entradas
Codificador	2^n	n	da el número de la entrada activa

El decodificador es lo que selecciona un chip de memoria a partir de los bits altos de la dirección, y lo que activa una línea de la matriz. El codificador **con prioridad** resuelve el caso de varias entradas activas devolviendo la de mayor prioridad, y es lo que hay en un controlador de interrupciones.

Además, un decodificador implementa cualquier función: cada salida es un mintérmino, así que basta una OR de las salidas que valgan 1.

3.7.2 Multiplexor y demultiplexor

Bloque	Qué hace
Multiplexor	selecciona una de 2^n entradas según n líneas de selección
Demultiplexor	envía una entrada a una de 2^n salidas



El multiplexor es el bloque más versátil del tema. Sirve para tres cosas distintas:

- **Seleccionar** una fuente de datos entre varias, que es su uso obvio.
- **Implementar cualquier función** de n variables con un multiplexor de n selecciones, conectando cada entrada a 0 o a 1 según la tabla de verdad.
- **Convertir de paralelo a serie**, barriendo las selecciones.

El segundo uso es el que sorprende: un multiplexor de 8 a 1 implementa cualquier función de tres variables sin una sola puerta.

3.7.3 Comparador

Compara dos números de n bits y da tres salidas: mayor, igual y menor. La igualdad se construye con XNOR bit a bit y una AND; el orden, comparando desde el bit más significativo hacia abajo y quedándose con la primera diferencia.

3.7.4 Circuitos aritméticos

Bloque	Entradas	Salidas
Semisumador	a, b	suma $= a \oplus b$, acarreo $= ab$
Sumador completo	a, b, c_{in}	suma $= a \oplus b \oplus c_{in}$, acarreo
Sumador de n bits	dos números	suma y acarreo final

Encadenando sumadores completos sale el **sumador con propagación de acarreo**, cuyo retardo crece linealmente con n porque cada etapa espera el acarreo de la anterior. Con 64 bits eso es demasiado, y la solución es el **sumador con anticipación de acarreo**, que calcula los acarreos en paralelo a partir de dos señales por bit:

$$g_i = a_i b_i \quad \text{(genera)}, \qquad p_i = a_i \oplus b_i \quad \text{(propaga)}$$

$$c_{i+1} = g_i + p_i c_i$$

Desarrollando la recurrencia, cada acarreo sale en dos niveles de puertas independientemente de n: se cambia área por velocidad, que es el compromiso típico del diseño digital.

Restar no necesita circuito nuevo: en complemento a 2 se invierten los bits del segundo operando y se pone el acarreo de entrada a 1. Un sumador con una fila de XOR gobernada por una señal hace las dos operaciones, y ese es el núcleo de la ALU.

3.7.5 La unidad aritmético-lógica

Combina el sumador-restador con las operaciones lógicas y un multiplexor que selecciona el resultado según la operación pedida. Sus salidas de estado —cero, signo, acarreo y desbordamiento— son las banderas que el tema 5 usa para los saltos condicionales.

3.8 Ejercicios

Ejercicio 3.8.1. Simplificar $f(a,b,c) = \sum m(0,1,2,3,5,7)$ con un mapa de Karnaugh.

Solución 3.8.1. Los mintérminos 0 a 3 son todos los que tienen $a = 0$, así que forman un grupo de cuatro que da $\bar{a}$. Los mintérminos 1, 3, 5 y 7 son los que tienen $c = 1$, otro grupo de cuatro que da c. Entre los dos cubren todos los unos, así que $f = \bar{a} + c$: de seis términos de tres variables a una sola puerta OR.

Ejercicio 3.8.2. ¿Por qué un circuito minimizado puede presentar riesgos y uno con términos redundantes no?

Solución 3.8.2. Porque al pasar de un grupo del mapa a otro adyacente hay un instante en que el primero ya se ha desactivado y el segundo aún no se ha activado, por la diferencia de retardos, y la salida da un pulso a 0 que no debería existir. Un término redundante que solape los dos grupos se mantiene activo durante la transición y tapa el hueco. Es lógicamente superfluo y eléctricamente necesario.

Ejercicio 3.8.3. Implementar $f(a,b,c) = \bar{a}b + ac$ con un multiplexor de 4 a 1 usando a y b como selecciones.

Solución 3.8.3. Se evalúa f para cada combinación de a y b dejando c como variable: con $ab = 00$, $f = 0$; con $ab = 01$, $f = 1$; con $ab = 10$, $f = c$; con $ab = 11$, $f = c$. Así que las entradas del multiplexor son 0, 1, c y c, sin ninguna puerta adicional.

El álgebra de conmutación y el diseño combinacional están desarrollados en Floyd, 2016, Mano y Kime, 2005 y Roth, 2004, y los bloques estándar en Lloris et al., 2003 y Gajski, 2004.

Capítulo 4

Estudio de sistemas secuenciales

Tema 4 del programa. Los circuitos con memoria: biestables, registros, contadores, y cómo se analiza un sistema secuencial.

4.1 Concepto

Definición 4.1 (Sistema secuencial). Circuito cuyas salidas dependen de las entradas actuales y del estado interno, que resume la historia de entradas anteriores.

	Combinacional	Secuencial
Salida depende de	entradas actuales	entradas y estado
Memoria	no	sí
Realimentación	no	sí
Ejemplo	sumador, multiplexor	contador, registro

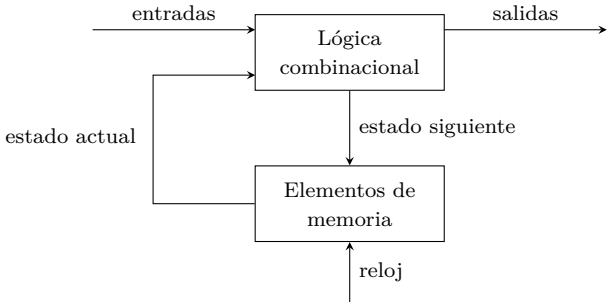
La realimentación es lo que crea la memoria: la salida vuelve a la entrada y el circuito se sostiene a sí mismo.

4.1.1 Síncronos y asíncronos

	Síncrono	Asíncrono
Cuándo cambia el estado	solo con el reloj	en cuanto cambia una entrada
Análisis	manejable	complicado por las carreras
Velocidad	la fija el reloj	potencialmente mayor
Uso	casi todo	casos concretos

Casi todo se diseña síncrono, y la razón es práctica: con un reloj basta con que la lógica combinacional se estabilice antes del flanco siguiente, y los pulsos espurios del tema anterior dejan de importar. En un circuito asíncrono, cualquier diferencia de retardo entre dos caminos puede llevar a un estado equivocado.

4.2 Modelos



Modelo	La salida depende de	Consecuencia
Moore	solo del estado	la salida cambia solo con el reloj: es estable
Mealy	del estado y de las entradas	reacciona antes, pero puede tener pulsos espurios

Mealy suele necesitar menos estados para el mismo comportamiento; Moore da salidas más limpias. La elección se hace por eso, no por gusto.

4.3 Elementos básicos secuenciales

4.3.1 El biestable RS

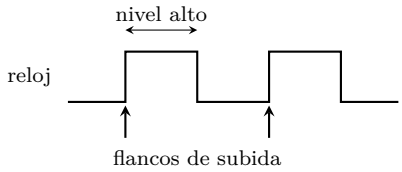
Dos puertas NOR realimentadas. Es el elemento de memoria más simple:

R	S	Q siguiente
0	0	se mantiene
0	1	1
1	0	0
1	1	**prohibida**

La combinación prohibida lo es porque fuerza las dos salidas al mismo valor, y al volver las entradas a cero el estado final depende de cuál cambie primero. Es una carrera, y el resultado es impredecible. Todo el diseño posterior de biestables consiste en eliminar esa combinación.

4.3.2 Disparo por nivel y por flanco

Tipo	Cuándo atiende a las entradas
Cerrojo (*latch*)	mientras la señal de habilitación está activa
Biestable por flanco	solo en el instante del flanco



El disparo por flanco es lo que hace posible el diseño síncrono. Con cerrojos, el estado nuevo puede propagarse por varias etapas dentro del mismo nivel activo y producir una carrera; con flanco, todos los elementos capturan a la vez y en un instante puntual.

4.3.3 Tipos de biestable

Tipo	Entradas	Comportamiento
D	D	$Q^+ = D$
JK	J, K	mantiene, pone a 1, pone a 0, o **conmuta** con $J = K = 1$
T	T	conmuta si $T = 1$
RS síncrono	R, S	como el RS, con reloj

Las ecuaciones características:

$$Q^+ = D, \qquad Q^+ = J\bar{Q} + \bar{K}Q, \qquad Q^+ = T \oplus Q$$

El **D** es el que se usa para registros, porque copia la entrada sin más. El **JK** elimina la combinación prohibida del RS convirtiéndola en conmutación, y por eso es el más versátil para contadores. El **T** es un JK con las dos entradas unidas.

4.3.4 Parámetros temporales

Parámetro	Qué exige
Tiempo de establecimiento (*setup*)	la entrada estable **antes** del flanco
Tiempo de mantenimiento (*hold*)	la entrada estable **después** del flanco
Retardo de propagación	del flanco a la salida válida

De ahí sale la restricción que fija la frecuencia máxima de cualquier circuito síncrono:

$$T_{reloj} \geq t_{propagación} + t_{lógica} + t_{setup}$$

El camino combinacional más lento entre dos biestables se llama **camino crítico**, y es lo único que limita el reloj. Optimizar cualquier otro camino no sube la frecuencia ni un hercio.

Nota 4.1 . Violar el tiempo de establecimiento no produce un valor equivocado sino algo peor: el biestable puede quedar en **metaestabilidad**, con la salida a un nivel intermedio durante un tiempo indeterminado. Es el problema de cruzar señales entre dos dominios de reloj, y se mitiga con dos biestables en cascada, no se elimina.

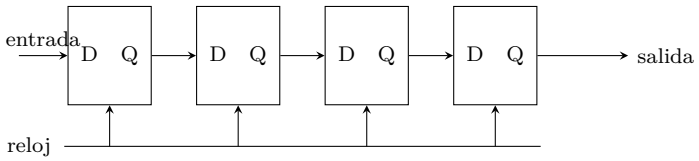
4.4 Componentes secuenciales estándar

4.4.1 Registros

Un conjunto de biestables D con el mismo reloj. Guarda una palabra.

Variante	Qué añade
Registro con carga	solo carga cuando la señal de habilitación lo permite
Registro de desplazamiento	cada biestable alimenta al siguiente
Con carga paralela y salida serie	conversión de paralelo a serie
Bidireccional	desplaza a izquierda o derecha

El **registro de desplazamiento** hace tres cosas distintas: convierte entre serie y paralelo, que es la base de cualquier comunicación; multiplica o divide por dos, ya que desplazar un bit equivale a eso; y genera secuencias pseudoaleatorias si se realimenta con XOR.



4.4.2 Contadores

Recorren una secuencia de estados a cada pulso.

Clasificación	Tipos
Por el reloj	asíncronos (en cascada) o síncronos (reloj común)
Por el sentido	ascendentes, descendentes o reversibles
Por el módulo	binarios de 2^n, o de módulo arbitrario

El contador asíncrono es más simple y peor. Cada biestable dispara al siguiente, así que los retardos se acumulan y durante un instante la cuenta pasa por valores que no existen en la secuencia. Con cuatro etapas de 10 ns, el valor no es fiable hasta 40 ns después del flanco. El síncrono cambia todos a la vez y no tiene ese problema, a costa de más lógica combinacional.

Un contador de módulo arbitrario se construye detectando el valor final y forzando la puesta a cero. En síncrono se hace con la carga síncrona, y no con la puesta a cero asíncrona: esta última produce un pulso muy corto en el estado transitorio.

4.4.3 Memoria y otros

Una memoria RAM estática es una matriz de celdas biestables con un decodificador que selecciona la fila y multiplexores que eligen la columna. Es la unión directa de los bloques de los dos temas.

4.5 Análisis de sistemas secuenciales

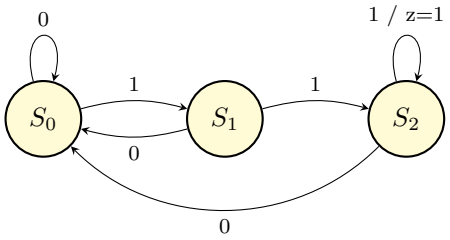
Analizar es partir del circuito y obtener qué hace. El procedimiento:

1. Identificar los biestables y sus entradas.
2. Escribir las **ecuaciones de excitación**: la entrada de cada biestable en función del estado y de las entradas.
3. Aplicar la ecuación característica para obtener el **estado siguiente**.
4. Construir la **tabla de estados**.
5. Dibujar el **diagrama de estados**.
6. Describir el comportamiento con palabras.

Ejemplo 4.1 . Un circuito con dos biestables D y una entrada x tiene $D_1 = Q_0$ y $D_0 = x \oplus Q_1$. La tabla de estados sale de sustituir:

Q_1Q_0	$x = 0$	$x = 1$
00	00	01
01	10	11
10	01	00
11	11	10

El diagrama de estados de un detector de secuencia, que es el ejemplo canónico:



Detecta tres unos seguidos. Cada estado recuerda cuántos unos consecutivos van, y esa es la idea general: **el estado codifica lo que hay que recordar del pasado**, y nada más. Un estado por cada cosa distinta que hay que recordar, ni uno más.

4.5.1 Diseño

El camino inverso, para completar:

1. Especificar el comportamiento y dibujar el diagrama de estados.
2. **Minimizar estados**: dos estados equivalentes si dan la misma salida y van al mismo sitio.
3. **Asignar códigos binarios** a los estados.
4. Elegir el tipo de biestable y construir la tabla de excitación.
5. Simplificar las funciones de excitación y de salida con mapas de Karnaugh.
6. Implementar y verificar.

El paso 3 tiene más peso del que parece. Con una asignación adecuada, la lógica de excitación sale mucho más simple, y una regla práctica es dar códigos adyacentes a estados que se suceden. El **código Gray** —donde dos valores consecutivos difieren en un bit— es la elección habitual, y además evita estados transitorios falsos.

4.6 Ejercicios

Ejercicio 4.6.1. ¿Por qué la combinación $R = S = 1$ está prohibida en un biestable RS?

Solución 4.6.1. Porque fuerza las dos salidas al mismo valor, rompiendo la relación $\bar{Q}$ que las define, y sobre todo porque al volver las dos entradas a cero el estado final depende de cuál cambie antes. Es una carrera: la salida queda a 0 o a 1 según retardos que el diseño no controla.

Ejercicio 4.6.2. Un contador asíncrono de 4 bits usa biestables con 12 ns de retardo. ¿Cuánto tarda en estabilizarse una cuenta y qué se observa mientras tanto?

Solución 4.6.2. Hasta $4 \times 12 = 48$ ns, porque cada etapa dispara a la siguiente. Durante ese intervalo la salida pasa por valores intermedios que no pertenecen a la secuencia: al pasar de 0111 a 1000 se ven 0110, 0100 y 0000. Si esa salida alimenta un decodificador, aparecen pulsos espurios en salidas que no deberían activarse nunca.

Ejercicio 4.6.3. Un circuito síncrono tiene $t_{propagación} = 3$ ns, $t_{setup} = 2$ ns y un camino combinacional máximo de 7 ns. ¿Cuál es su frecuencia máxima?

Solución 4.6.3. $T \geq 3 + 7 + 2 = 12$ ns, así que $f_{máx} = 1/12$ ns ≈ 83 MHz. Para subirla hay que acortar el camino crítico, por ejemplo partiéndolo en dos etapas con un registro entre medias: eso es exactamente la segmentación.

Los biestables y los componentes secuenciales están desarrollados en Floyd, 2016 y Mano y Kime, 2005, y el análisis y diseño de máquinas de estados en Roth, 2004 y Gajski, 2004.

Capítulo 5

Sistemas en el nivel de transferencia entre registros

Tema 5 del programa. El nivel que une el diseño lógico con la arquitectura: cómo se describe un computador en términos de registros, transferencias y señales de control, y cómo se construye la máquina sencilla CS1.

5.1 Introducción y definiciones

El nivel RT describe el sistema como **registros** que guardan información y **transferencias** entre ellos, gobernadas por señales de control. Es el nivel donde una instrucción máquina se descompone en pasos elementales.

Elemento	Qué es
Registro	almacena una palabra
Micro-operación	transferencia elemental entre registros, en un ciclo
Señal de control	activa una micro-operación
Camino de datos	los registros y las unidades operativas, con sus conexiones
Unidad de control	genera las señales de control en el orden adecuado

La notación:

$$R_1 \leftarrow R_2 \qquad \text{transferir el contenido de } R_2 \text{ a } R_1$$

$$R_3 \leftarrow R_1 + R_2 \qquad \text{sumar y guardar}$$

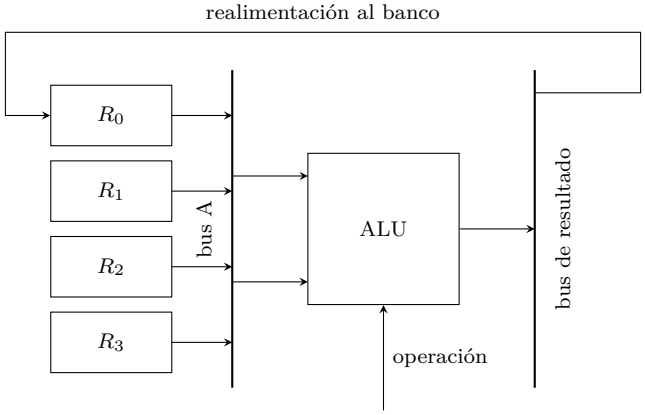
$$K_1\colon\ R_1 \leftarrow R_2 \qquad \text{condicionada a la señal } K_1$$

$$R_1 \leftarrow R_2,\ R_3 \leftarrow R_4 \qquad \text{simultáneas, en el mismo ciclo}$$

La cuarta forma importa: **las micro-operaciones separadas por coma ocurren a la vez**, no una tras otra, porque todos los registros capturan en el mismo flanco. Eso permite intercambiar dos registros en un solo ciclo, algo imposible con asignaciones secuenciales.

5.2 Unidad de procesamiento

El camino de datos contiene los registros, la unidad aritmético-lógica y los caminos que los conectan.



Con una estructura de un bus, una transferencia con operación necesita varios ciclos porque los dos operandos no pueden circular a la vez. Con dos buses de lectura y uno de escritura, la operación completa cabe en un ciclo. **La anchura de la estructura de buses decide cuántos ciclos cuesta cada instrucción**, y es la primera decisión de diseño del camino de datos.

5.2.1 Ejemplos de operaciones

Micro-operación	Qué hace
$R_1 \leftarrow R_2$	transferencia
$R_1 \leftarrow R_1 + R_2$	suma acumulada
$R_1 \leftarrow R_1 + 1$	incremento
$R_1 \leftarrow \overline{R_2}$	complemento
$R_1 \leftarrow R_1 \wedge R_2$	AND bit a bit
$R_1 \leftarrow \mathrm{desp}_i(R_1)$	desplazamiento a la izquierda
$MAR \leftarrow PC$	preparar un acceso a memoria
$MDR \leftarrow M[MAR]$	lectura de memoria
$M[MAR] \leftarrow MDR$	escritura en memoria

Las tres últimas son las que aparecen en todas las instrucciones: **cualquier acceso a memoria son al menos dos ciclos**, uno para poner la dirección y otro para mover el dato.

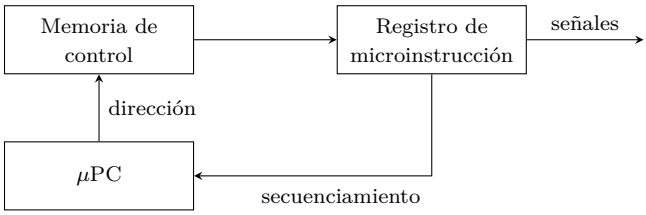
5.3 Unidad de control

Genera las señales que activan las micro-operaciones, en el orden que exige cada instrucción. Dos formas de construirla:

Tipo	Cómo	Ventaja	Problema
Cableada	máquina de estados con lógica combinacional	rápida	modificarla exige rediseñar

Tipo	Cómo	Ventaja	Problema
Microprogramada	una memoria con una palabra por paso	flexible y ordenada	más lenta

La cableada es la máquina de estados del tema 4, con un estado por paso del ciclo de instrucción. La microprogramada guarda en memoria una **microinstrucción** por paso, con un bit por señal de control, y un contador que las recorre.



La decisión entre las dos es histórica y explica CISC y RISC: los juegos de instrucciones complejos se microprogramaron porque cablearlos era inviable, y los RISC volvieron a la lógica cableada porque su juego regular sí lo permite y es más rápido.

5.3.1 Ejemplos de generación de señales

Para la instrucción de carga desde memoria, la secuencia de micro-operaciones y las señales que cada paso activa:

Ciclo	Micro-operación	Señales activas
1	$MAR \leftarrow PC$	salida de PC al bus, carga de MAR
2	$MDR \leftarrow M[MAR]$, $PC \leftarrow PC + 1$	lectura de memoria, incremento de PC
3	$IR \leftarrow MDR$	salida de MDR, carga de IR
4	$MAR \leftarrow IR_{dir}$	salida del campo de dirección, carga de MAR
5	$MDR \leftarrow M[MAR]$	lectura de memoria
6	$AC \leftarrow MDR$	salida de MDR, carga del acumulador

Los tres primeros ciclos son **iguales para todas las instrucciones**: son la fase de captación. Los tres últimos son la ejecución, y dependen del código de operación. Esa separación es la que estructura toda la unidad de control.

Un **salto condicional** añade un elemento más: la señal depende de una bandera del registro de estado.

$$Z\text{: } PC \leftarrow IR_{dir}$$

Es decir, la transferencia solo ocurre si la bandera de cero está activa. Con eso, la unidad de control ya sabe tomar decisiones y la máquina es programable en el sentido completo.

5.4 El computador sencillo CS1

Reuniendo todo lo anterior sale una máquina completa. Sus elementos:

Elemento	Función
Memoria	instrucciones y datos, con MAR y MDR como interfaz
Acumulador (AC)	operando implícito de las operaciones
Contador de programa (PC)	dirección de la instrucción siguiente
Registro de instrucción (IR)	instrucción en curso
ALU	suma, resta y operaciones lógicas
Registro de estado	banderas de cero, signo, acarreo y desbordamiento
Unidad de control	genera las señales

Un juego de instrucciones mínimo pero suficiente:

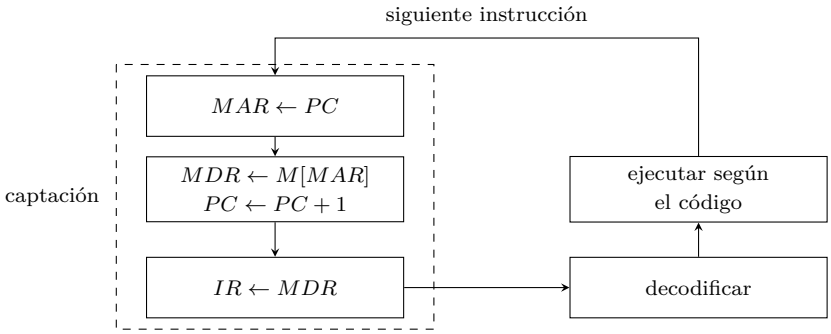
Instrucción	Efecto
`LOAD dir`	$AC \leftarrow M[dir]$
`STORE dir`	$M[dir] \leftarrow AC$
`ADD dir`	$AC \leftarrow AC + M[dir]$
`SUB dir`	$AC \leftarrow AC - M[dir]$
`AND dir`	$AC \leftarrow AC \wedge M[dir]$
`JMP dir`	$PC \leftarrow dir$
`JZ dir`	si Z, $PC \leftarrow dir$
`HALT`	detiene la máquina

Con esas ocho instrucciones se programa cualquier cosa computable, y ese es el punto del ejemplo: **la potencia de un computador no viene de tener muchas instrucciones**.

Ejemplo 5.1 . Sumar los elementos de un vector de n posiciones a partir de la dirección `V`, con el acumulador como única variable, exige modificar la dirección de la instrucción `ADD` en cada vuelta. En una máquina sin registro índice eso se hace **modificando el propio programa**: se carga la instrucción como si fuera un dato, se le suma uno y se vuelve a almacenar.

Es la consecuencia directa de guardar programa y datos en la misma memoria, y explica por qué los modos de direccionamiento indexado aparecieron pronto: escribir programas que se automodifican es correcto y es una pesadilla de depurar.

5.4.1 El ciclo completo, paso a paso



Ese bucle es lo que hace una máquina, y es literalmente todo lo que hace: captar, decodificar, ejecutar y repetir. Cada paso son las micro-operaciones de arriba, cada micro-operación son señales de control, cada señal gobierna registros y multiplexores del tema 4, y cada registro está hecho de biestables construidos con las puertas del tema 3.

Ese recorrido completo, de la puerta lógica al computador, es el objetivo de la asignatura.

5.5 Ejercicios

Ejercicio 5.5.1. ¿Por qué $R_1 \leftarrow R_2,\ R_2 \leftarrow R_1$ intercambia los dos registros y no deja los dos con el mismo valor?

Solución 5.5.1. Porque las dos transferencias se ejecutan a la vez: los dos registros leen la salida actual del otro y capturan en el mismo flanco de reloj. Lo que se lee es el valor anterior al flanco, no el nuevo. En un lenguaje de programación, en cambio, la primera asignación destruye R_1 antes de la segunda y hace falta una variable auxiliar.

Ejercicio 5.5.2. Una máquina tiene un solo bus interno. ¿Cuántos ciclos necesita $R_3 \leftarrow R_1 + R_2$?

Solución 5.5.2. Tres: uno para llevar R_1 a un registro temporal de la ALU, otro para llevar R_2 al otro operando y ejecutar la suma, y otro para llevar el resultado a R_3. Con dos buses de lectura y uno de escritura basta un ciclo, porque los dos operandos viajan a la vez y el resultado tiene su propio camino de vuelta.

Ejercicio 5.5.3. ¿Qué ventaja tiene una unidad de control microprogramada frente a una cableada, y qué se paga por ella?

Solución 5.5.3. Que añadir o modificar una instrucción es reescribir la memoria de control, sin tocar el circuito, lo que la hace mucho más manejable con juegos de instrucciones grandes. El precio es velocidad: cada paso exige leer una microinstrucción de la memoria de control, y eso es más lento que la salida directa de una red combinacional. Por eso los procesadores RISC volvieron a la lógica cableada.

El nivel de transferencia entre registros y el diseño de la unidad de control están desarrollados en Mano y Kime, 2005, Gajski, 2004 y Stallings, 2022, y su tratamiento con problemas resueltos en Prieto y Prieto, 2010 y Díaz Ruiz et al., 2009.

Capítulo 6

Temario práctico

Los cinco seminarios y las siete prácticas de laboratorio del programa.

6.1 Seminarios

6.1.1 S1. Sistemas de numeración usuales en Informática

Conversión entre bases, aritmética binaria y representación de enteros con signo.

Conversión	Método
Decimal a binario, parte entera	divisiones sucesivas por 2, restos al revés
Decimal a binario, parte fraccionaria	multiplicaciones sucesivas por 2, partes enteras en orden
Binario a octal o hexadecimal	agrupar de tres o de cuatro bits desde la coma
Binario a decimal	suma de potencias

La parte fraccionaria es donde aparece el resultado que hay que ver una vez para no olvidarlo: 0,1 **decimal es periódico en binario**, $0{,}0\overline{0011}$. No hay número de bits que lo represente exactamente, y de ahí vienen todos los errores de redondeo del coma flotante.

Se practica también la aritmética en complemento a 2 y la detección de desbordamiento, con la regla del tema 1: solo desborda al sumar dos números del mismo signo y obtener el contrario.

6.1.2 S2. Representación de información multimedia

Información	Cómo se digitaliza	Parámetros
Texto	un código por carácter	ASCII, Unicode, UTF-8
Sonido	muestreo y cuantificación	frecuencia de muestreo, bits por muestra
Imagen	matriz de píxeles	resolución, profundidad de color
Vídeo	secuencia de imágenes	lo anterior más los fotogramas por segundo

Los dos cálculos que se piden:

$$\text{tamaño del audio} = f_{muestreo} \times \text{bits} \times \text{canales} \times \text{segundos}$$

$$\text{tamaño de la imagen} = \text{ancho} \times \text{alto} \times \text{bits por píxel}$$

Un minuto de audio de calidad de CD son $44\,100 \times 16 \times 2 \times 60 = 10{,}6$ MB, y una foto de 12 megapíxeles sin comprimir, 36 MB. Esas cifras son las que justifican la compresión, y de paso la distinción entre **compresión sin pérdida** —reversible, para texto y datos— y **con pérdida** —descarta lo que el oído o el ojo no perciben, para audio, imagen y vídeo—.

Sobre UTF-8 conviene retener que es de longitud variable: un carácter ASCII ocupa un byte y uno acentuado dos. Por eso el número de caracteres de una cadena no es su número de bytes, y contar mal ahí rompe programas de forma sutil.

6.1.3 S3. Álgebra de conmutación. Funciones de conmutación

Los postulados y teoremas del tema 3, aplicados a demostrar identidades y a manipular expresiones. Lo que se ejercita:

- Demostrar una identidad por tabla de verdad y por manipulación algebraica.
- Aplicar De Morgan a expresiones con varios niveles de negación.
- Obtener las dos formas canónicas a partir de la tabla de verdad.
- Pasar de suma de productos a producto de sumas y al revés.

La comprobación por tabla de verdad es la red de seguridad: una demostración algebraica con un error da una expresión que parece razonable, y la tabla lo delata en ocho filas.

6.1.4 S4. Minimización de funciones de conmutación

Mapas de Karnaugh de tres, cuatro y cinco variables, con indiferencias.

Paso	Qué hacer
1	volcar la tabla de verdad al mapa, con el orden Gray de las etiquetas
2	localizar los **implicantes primos**: grupos que no se pueden agrandar
3	identificar los **esenciales**: los que cubren algún uno que nadie más cubre
4	completar la cobertura con los mínimos implicantes restantes
5	escribir la expresión

Los dos errores que se repiten: **olvidar que el mapa se cierra por los bordes**, y no tomar el grupo más grande posible. El segundo no da una función incorrecta, solo una implementación más cara, y por eso pasa desapercibido.

Se practica también la minimización **multisalida**, donde varias funciones comparten términos y el objetivo no es minimizar cada una por separado sino el circuito conjunto.

6.1.5 S5. Manejo de un simulador y de un entrenador lógico

El puente entre el papel y el laboratorio.

Herramienta	Para qué
Simulador lógico	montar el circuito, aplicar entradas y ver salidas
Cronograma	observar retardos y pulsos espurios
Entrenador lógico	montaje físico con circuitos integrados reales

Lo que el simulador enseña y el papel no: **el cronograma con retardos**. Un circuito correcto en la tabla de verdad muestra pulsos espurios al cambiar la entrada, y ahí se ve el riesgo estático del tema 3 en vez de leerlo.

En el entrenador aparecen además los problemas físicos: entradas al aire que flotan y dan valores aleatorios, alimentación mal conectada, y salidas de dos puertas unidas entre sí, que es un cortocircuito y no una OR.

6.2 Prácticas de laboratorio

6.2.1 P1. Análisis y diseño de circuitos combinacionales con puertas lógicas

Montar circuitos a partir de una expresión, y obtener la expresión a partir de un montaje.

Actividad	Qué comprobar
Verificar De Morgan con puertas	las dos formas dan la misma tabla
Implementar una función con AND, OR y NOT	la tabla coincide con la especificada
Implementarla solo con NAND	el resultado es idéntico
Comparar la versión canónica con la minimizada	menos puertas, mismo comportamiento

La comparación final es la que da sentido al seminario S4: se mide cuántos circuitos integrados hacen falta antes y después de minimizar.

6.2.2 P2. Diseño de circuitos aritméticos

Montaje	Qué se observa
Semisumador con XOR y AND	suma y acarreo
Sumador completo	tres entradas, dos salidas
Sumador de 4 bits en cascada	el acarreo se propaga
Sumador-restador con XOR de control	una señal cambia la operación

Lo que la práctica demuestra: **la resta no necesita circuito propio**. Con las XOR gobernadas por la señal de modo y el acarreo de entrada a 1, el mismo sumador resta en complemento a 2.

Y se mide el **retardo de propagación del acarreo**: con el sumador de 4 bits, el resultado no es válido hasta que el acarreo ha atravesado las cuatro etapas. Es la motivación concreta de la anticipación de acarreo.

6.2.3 P3. Unidad aritmético-lógica

Construir una ALU con selección de operación y comprobar sus banderas.

Comprobación	Qué confirma
Suma con desbordamiento	la bandera se activa con operandos del mismo signo
Resta que da cero	la bandera de cero
Acarreo de salida	distinto del desbordamiento
Operaciones lógicas	AND, OR, XOR, complemento

La distinción entre **acarreo y desbordamiento** es lo que la práctica fija: son dos banderas distintas y se usan en contextos distintos. El acarreo importa en aritmética sin signo y el desbordamiento en complemento a 2, y confundirlos produce comparaciones incorrectas.

6.2.4 P4. Codificadores, decodificadores, multiplexores y demultiplexores

Montaje	Qué se comprueba
Decodificador 3 a 8	una sola salida activa por combinación
Implementar una función con el decodificador y una OR	cada salida es un mintérmino
Multiplexor 8 a 1	selecciona la entrada indicada
Implementar una función de 3 variables con el multiplexor	sin ninguna puerta
Demultiplexor	distribuye una entrada a la salida elegida
Codificador con prioridad	con varias entradas activas gana la mayor

Las dos filas de implementación son las importantes. Ver que **un multiplexor implementa cualquier función de tres variables sin una sola puerta** cambia la forma de abordar los diseños.

6.2.5 P5. Biestables y registros básicos

Montaje	Qué se observa
Biestable RS con NOR	la combinación prohibida y su resultado impredecible
Biestable D disparado por flanco	captura solo en el flanco
Comparación con un cerrojo	el cerrojo es transparente mientras el nivel está activo
Registro de 4 bits	carga paralela
Registro de desplazamiento	conversión entre serie y paralelo

La comparación entre cerrojo y biestable por flanco se hace mejor con el cronograma: mientras la habilitación está activa, la salida del cerrojo sigue a la entrada; la del biestable solo cambia en un instante.

Se comprueban además los **rebotes de los pulsadores**: cerrar un contacto mecánico produce varias transiciones en unos milisegundos, así que un contador conectado directamente a un pulsador

cuenta tres o cuatro por pulsación. Se corrige con un circuito antirrebote, que es el disparador de Schmitt o un biestable RS.

6.2.6 P6. Implementación y funcionamiento de sistemas secuenciales

Montaje	Qué se comprueba
Contador asíncrono de 4 bits	los valores intermedios espurios
Contador síncrono de 4 bits	todos los bits cambian a la vez
Contador de módulo arbitrario	detección del final y puesta a cero
Detector de secuencia	el estado recuerda lo necesario

La primera fila es la lección de la práctica: **el contador asíncrono da valores que no pertenecen a la cuenta**. Con un decodificador a la salida se ven pulsos en líneas que no deberían activarse, y esa observación justifica por sí sola el diseño síncrono.

6.2.7 P7. Descripción a nivel RT de un computador sencillo

La práctica que cierra la asignatura: describir y simular la máquina CS1 del tema 5.

Actividad	Qué produce
Identificar los registros y los caminos	el camino de datos
Escribir las micro-operaciones de cada instrucción	la secuencia de ciclos
Determinar las señales activas en cada ciclo	la tabla de control
Simular la ejecución de un programa corto	la traza, ciclo a ciclo

La comprobación final es seguir la traza de un programa sencillo —cargar, sumar, almacenar y saltar— viendo en cada ciclo qué registros cambian y qué señales están activas.

Ahí es donde encaja todo: la instrucción que el programa escribió es una palabra en memoria, la unidad de control la descompone en micro-operaciones, cada micro-operación activa señales, y esas señales gobiernan registros y multiplexores construidos con las puertas de las prácticas anteriores.

6.3 Sobre las memorias de prácticas

Lo que se entrega:

1. Especificación del problema y tabla de verdad o de estados.
2. Proceso de simplificación, con los mapas.
3. Esquema del circuito, con los circuitos integrados usados.
4. Resultados de la simulación y del montaje, **incluidos los cronogramas**.
5. Diferencias entre lo esperado y lo observado, con su explicación.

El punto 5 es el que se evalúa de verdad, y las causas posibles son casi siempre las mismas: retardos de propagación, riesgos estáticos, rebotes de los pulsadores, entradas sin conectar que flotan, o una simplificación mal hecha. Lo que se pide es identificar cuál explica lo observado, no enumerarlas.

Los guiones y sus problemas siguen Prieto y Prieto, 2010, Díaz Ruiz et al., 2009 y Floyd, 2016, y el material de simulación, Lloris et al., 2003 y Deschamps et al., 2017.

Bibliografía

Deschamps, J.-P., Valderrama, E., & Terés, L. (2017). *Digital Systems: From Logic Gates to Processors*. Springer. https://doi.org/10.1007/978-3-319-41198-9

Díaz Ruiz, S., Romero Ternero, M. C., & Molina Cantero, A. J. (2009). *Estructura y Tecnología de Computadores. Teoría y problemas*. McGraw-Hill.

Floyd, T. L. (2016). *Fundamentos de Sistemas Digitales* (11.ª ed.). Prentice-Hall.

Gajski, D. D. (2004). *Principios de diseño digital*. Prentice Hall.

Hamacher, C., Vranesic, Z., & Zaky, S. (2003). *Organización de computadores* (5.ª ed.). McGraw-Hill.

Lloris, A., Prieto, A., & Parrilla, L. (2003). *Sistemas Digitales*. McGraw-Hill.

Mano, M. M., & Kime, C. R. (2005). *Fundamentos de diseño lógico y de computadores* (3.ª ed.). Pearson Education.

Prieto, A., & Prieto, B. (2010). *Conceptos de Informática. Problemas*. McGraw-Hill.

Roth, C. H. (2004). *Fundamentos de Diseño Lógico* (5.ª ed.). International Thomson.

Stallings, W. (2022). *Computer Organization and Architecture: Designing for Performance* (11.ª ed.). Pearson Higher Education.