Universidad Nacional de Ingeniería
Facultad de Ciencias

Introducción a la Ciencia de la Computación

Organizacion del Computador

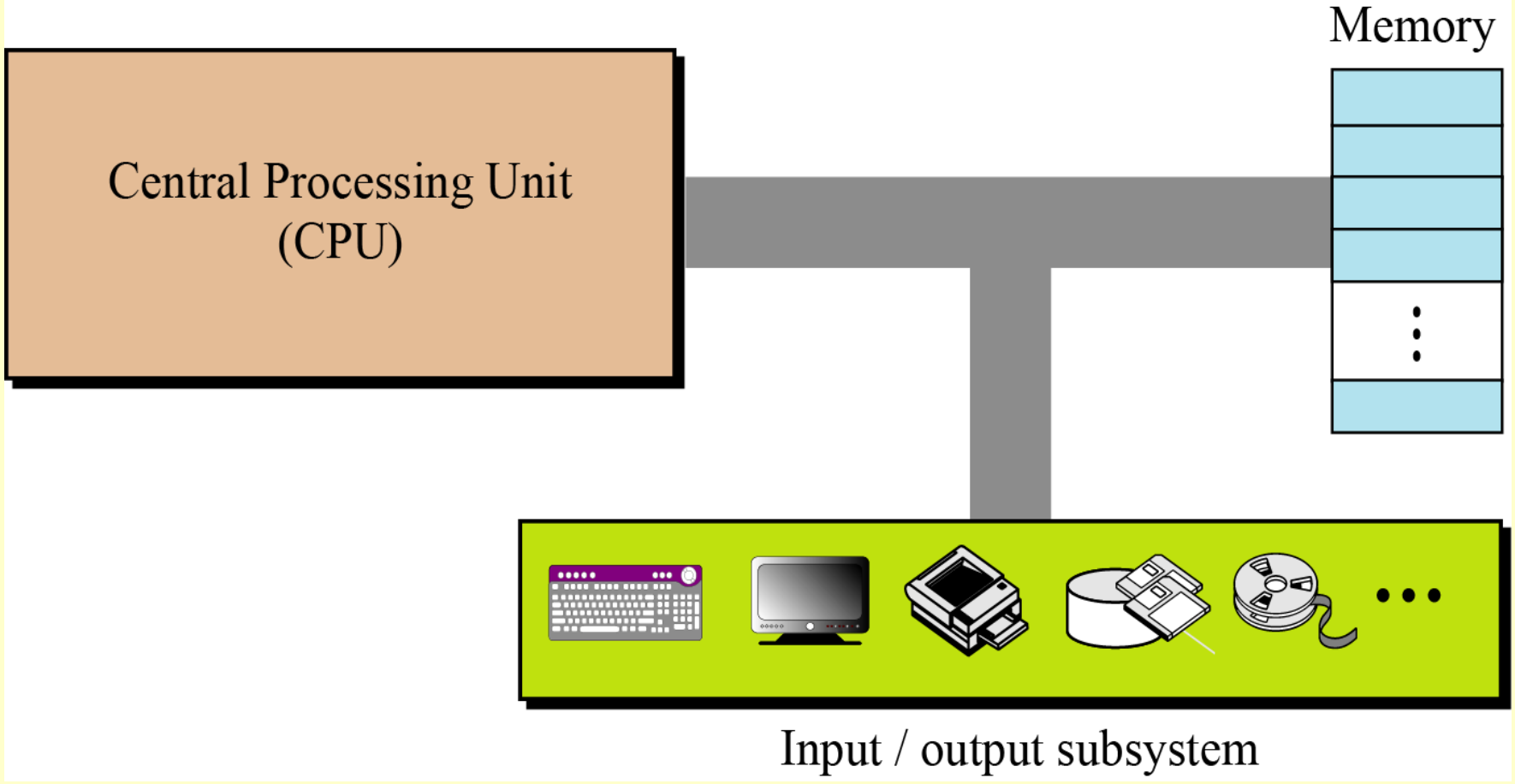
Prof: J. Solano
2011-I

Objetivos

Despues de estudiar este cap. el estudiante sera capaz de:

- **Listar los tres subsistemas de un computador.**
- **Describir el rol de la unidad de procesamiento central (CPU).**
- **Describir las fases fetch-decode-execute de un ciclo.**
- **Describir la memoria principal y su espacio de direccionamiento.**
- **Describir el subsistema de entrada/salida.**
- **Entender la interconeccion de subsistemas.**
- **Describir diferentes metodos de direccionamiento de entrada/salida.**
- **Distinguir los dos grandes tendencias en el diseño de computadoras.**
- **Entender cómo el rendimiento (throughput) del ordenador se puede mejorar mediante la canalización (pipelining) y el procesamiento en paralelo.**

Podemos dividir las partes que componen un ordenador en tres grandes categorías o subsistemas: la unidad de procesamiento central (CPU), la memoria principal y el subsistema de entrada / salida.



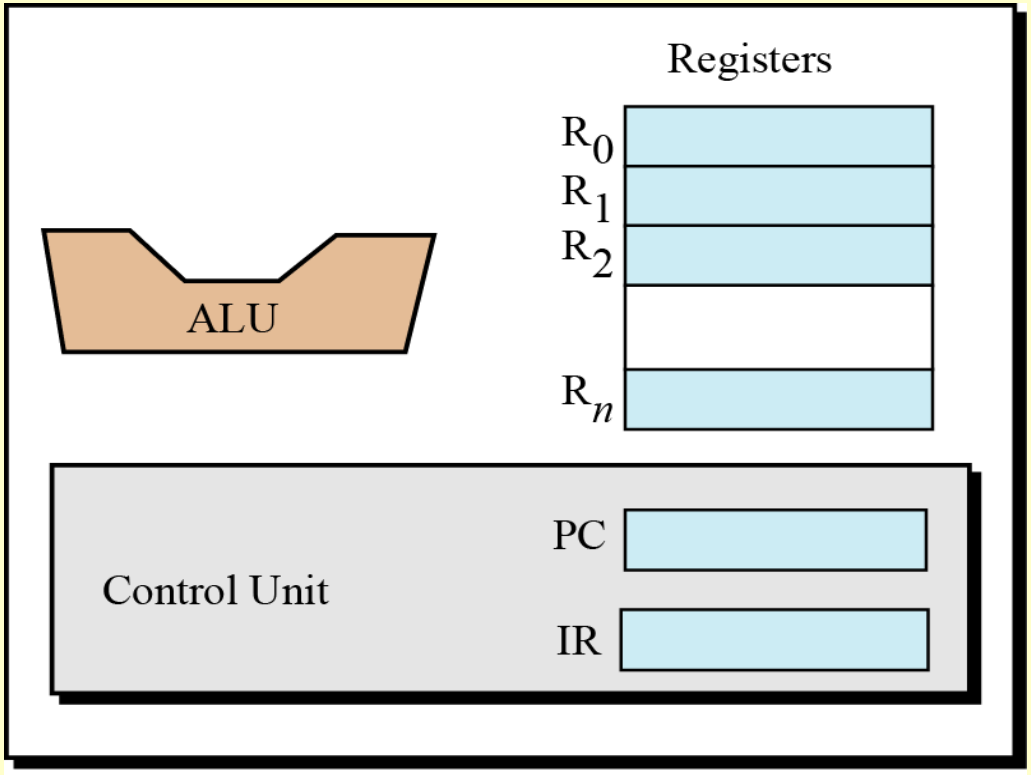
Material informatico / computer hardware (subsistemas)

UNIDAD DE PROCESAMIENTO CENTRAL

La **unidad de procesamiento central (CPU)** realiza operaciones en los datos. En la mayoría de las arquitecturas consta de tres partes: una **unidad aritmética lógica (ALU)**, una **unidad de control** y un conjunto de **registros**, ubicaciones de almacenamiento rápido.

La unidad aritmética lógica (ALU)

La **unidad aritmética lógica (ALU)**, realiza operaciones **lógicas**, de **desplazamiento** y **aritméticas** en los datos.



Registros

Los registros son lugares de almacenamiento rápido independiente que guardan datos en forma temporal. Registros múltiples son necesarios para facilitar el funcionamiento del CPU. Algunos de estos registros se muestran en la figura anterior.

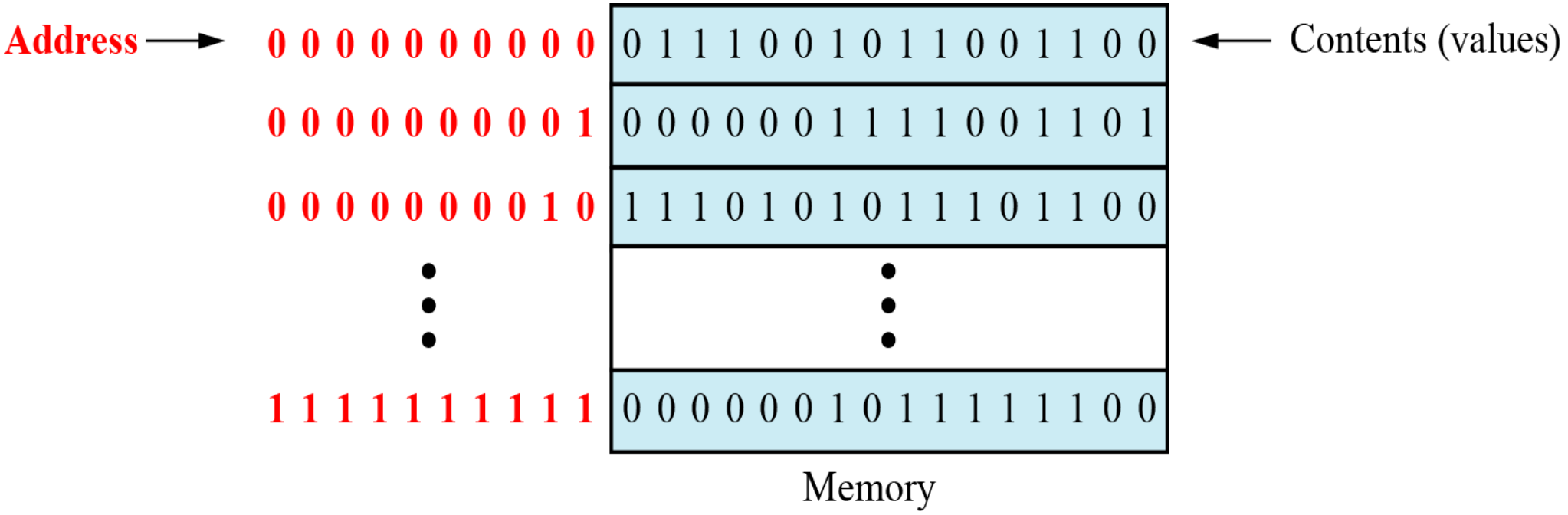
- **Registros de datos**
- **Registro de Instruccion (IR)**
- **Contador de Programa (PC)**

La Unidad de Control

La tercera parte de cualquier CPU es la unidad de control. La unidad de control controla el funcionamiento de cada subsistema. El control se logra a través de señales enviadas desde la unidad de control a otros subsistemas.

MEMORIA PRINCIPAL

La **memoria principal** es el segundo subsistema importante en un equipo. Se trata de una colección de lugares de almacenamiento, cada uno con un identificador único, llamado **dirección**. Los datos se transfieren hacia y desde la memoria en grupos de bits llamados **palabras**. Una palabra puede ser un grupo de 8 bits, 16 bits, 32 bits o 64 bits (y creciendo). Si la palabra es de 8 bits, se conoce como un **byte**. El término "byte" es tan común en Ciencia de la Computación que a veces una palabra de 16 bits se conoce como una palabra de 2-bytes, o una palabra de 32 bits se conoce como una palabra de 4-bytes.



Memoria principal

Espacio de direcciones

Para acceder a una palabra en la memoria se requiere un identificador. Aunque los programadores utilizan un nombre para identificar una palabra (o una colección de palabras), a nivel de hardware se identifica cada palabra por una dirección. El número total de localidades únicas identificables en la memoria se llama el espacio de direcciones. Por ejemplo, una memoria de 64 kilobytes y un tamaño de palabra de 1 byte tiene un espacio de direcciones que va desde 0 a 65,535.

Unit	*Exact Number of Bytes*	*Approximation*
kilobyte	2^{10} (1024) bytes	10^3 bytes
megabyte	2^{20} (1,048,576) bytes	10^6 bytes
gigabyte	2^{30} (1,073,741,824) bytes	10^9 bytes
terabyte	2^{40} bytes	10^{12} bytes

Direcciones de memoria son definidas usando enteros binarios sin signo

Ejemplo 1

Un computador tiene 32 MB (megabytes) de memoria. Cuantos bits son necesarios para direccionar un byte en memoria

Ejemplo 2

Un computador tiene 128 MB de memoria. Cada palabra en este computador es de 8 bytes. Cuantos bits son necesarios para direccionar una palabra en memoria

Ejemplo 1

Un computador tiene 32 MB (megabytes) de memoria. Cuantos bits son necesarios para direccionar un byte en memoria

Solución

El espacio de direccion de memoria es 32 MB, o 2^{25} ($2^5 \times 2^{20}$). Esto significa que necesitamos $\log_2 2^{25}$, or **25 bits**, para direccionar cada byte.

Ejemplo 2

Un computador tiene 128 MB de memoria. Cada palabra en este computador es de 8 bytes. Cuantos bits son necesarios para direccionar una palabra en memoria

Solución

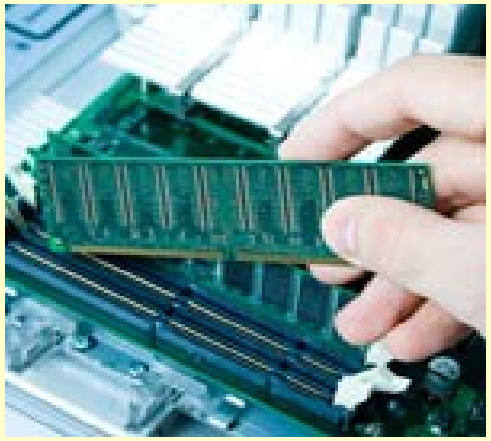
El espacio de direccion de memoria es 128 MB, o 2^{27}. Sin embargo cada palabra es de ocho (2^3) bytes, que significa que tiene 2^{24} palabras. Esto significa que necesitamos $\log_2 2^{24}$, or **24 bits**, para direccionar cada palabra.

Tipos de Memoria

Existen principalmente dos tipos de memoria: **RAM** y **ROM**.

Random access memory (RAM)

- Static RAM (SRAM)
- Dynamic RAM (DRAM)

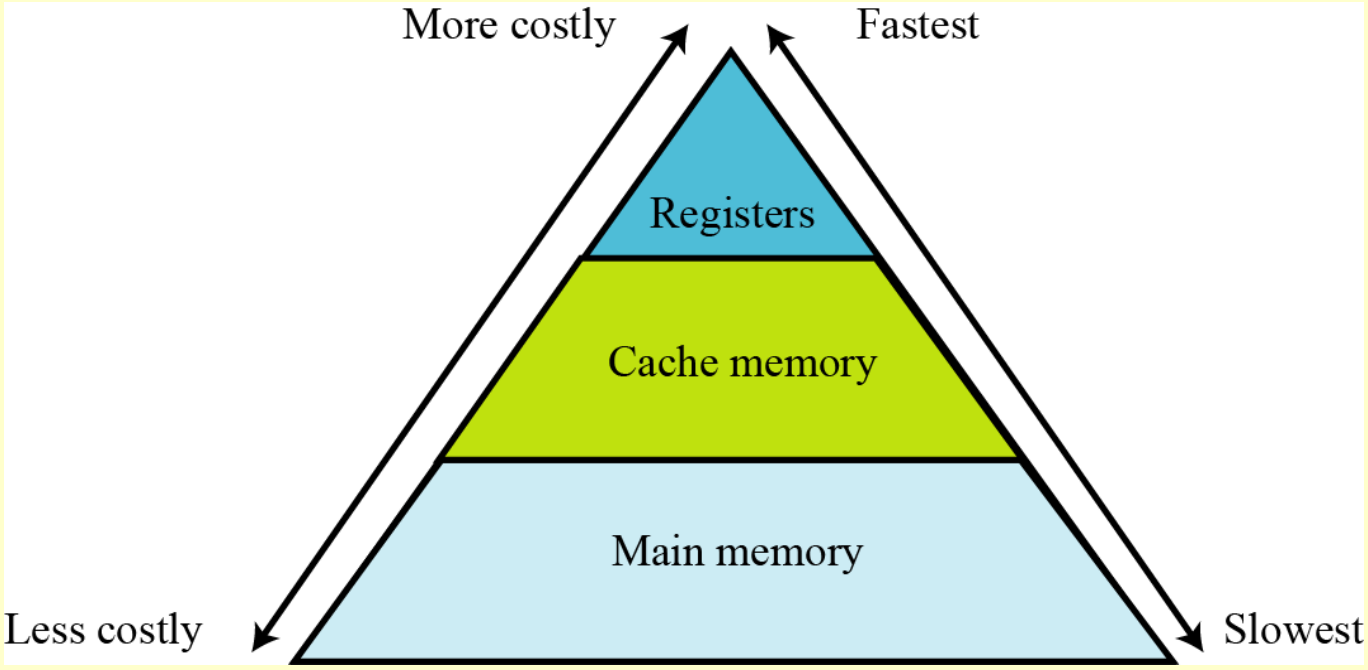


Read-only memory (ROM)

- Programmable read-only memory (PROM).
- Erasable programmable read-only memory (EPROM).
- Electrically erasable programmable read-only memory (EEPROM).

Jerarquia de la memoria

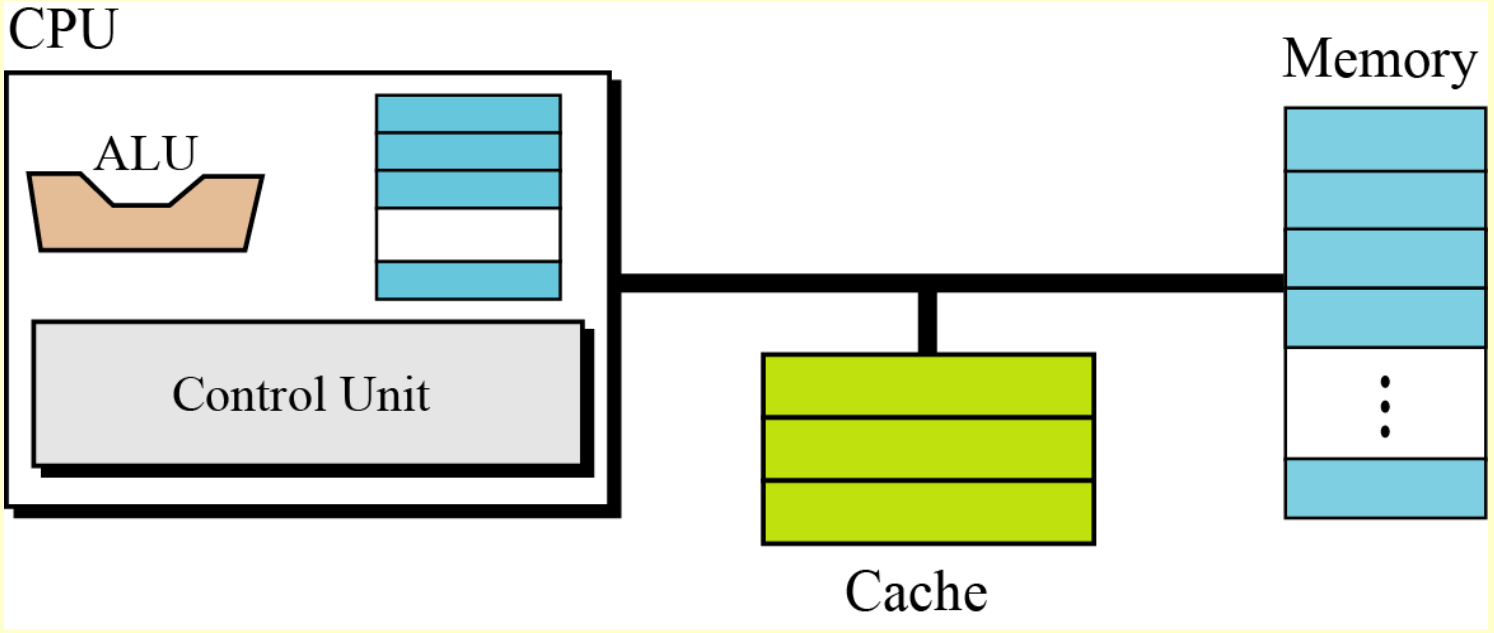
Los usuarios de computadoras necesitan una gran cantidad de memoria, especialmente memoria rápida y barata. Esta demanda no siempre es posible satisfacer – memoria muy rápida no suele ser barata. El compromiso tiene que ser hecho. La solución consiste en los niveles **jerárquicos** de la memoria.



Jerarquia de la memoria

Memoria cache

La memoria caché es más rápida que la memoria principal, pero más lenta que la CPU y sus registros. La memoria caché, que normalmente es de tamaño pequeño, se coloca entre la CPU y la memoria principal.



Memoria cache

SUBSISTEMA ENTRADA/SALIDA

El tercer subsistema importante en un computador es la colección de dispositivos referidos como el subsistema de entrada/salida (E/S). Este subsistema permite al computador comunicarse con el mundo exterior y almacenar programas y datos incluso cuando el aparato está apagado. Dispositivos de entrada/salida pueden ser dividido en dos grandes categorías: dispositivos de **no-almacenamiento** y de **almacenamiento**.

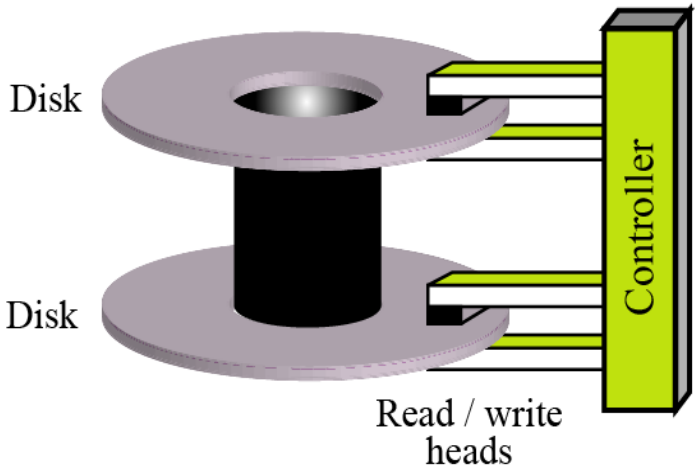
Dispositivos de no-almacenamiento

Dispositivos de no-almacenamiento permiten a la memoria/CPU comunicarse con elmundo exterior, pero ellos no pueden almacenar informacion.

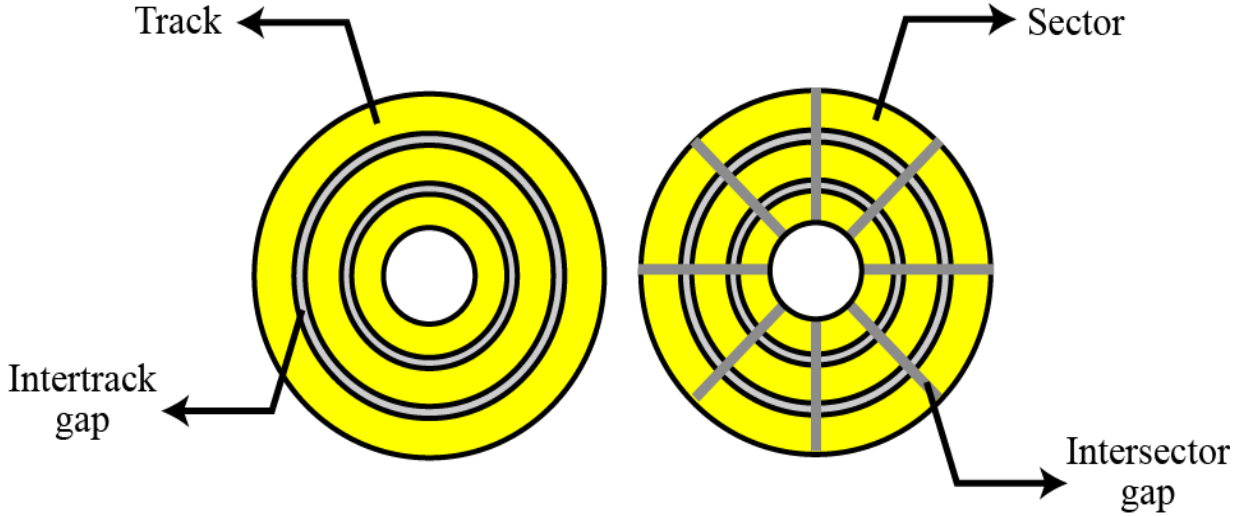
- **Teclado y monitor**
- **Impresora**

Dispositivos de almacenamiento

Dispositivos de almacenamiento, aunque se clasifican como dispositivos I/O, puede almacenar grandes cantidades de información a ser recuperada en un momento posterior. Son más baratos que la memoria principal, y sus contenidos no son volátiles -- es decir, no se borra cuando el computador está apagado. A veces se refiere a dispositivos auxiliares de almacenamiento. Podemos clasificarlos ya sea **magnéticos** u **ópticos**.

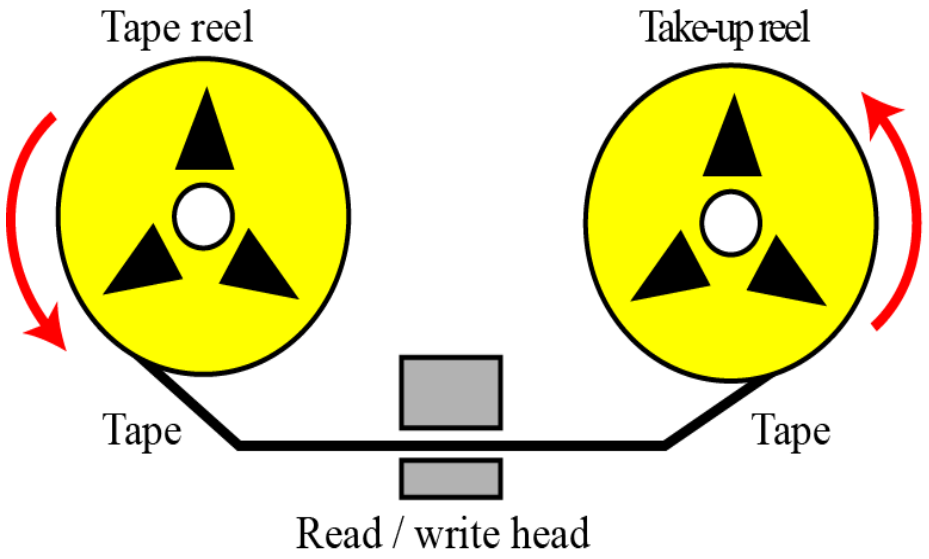


a. Disk drive

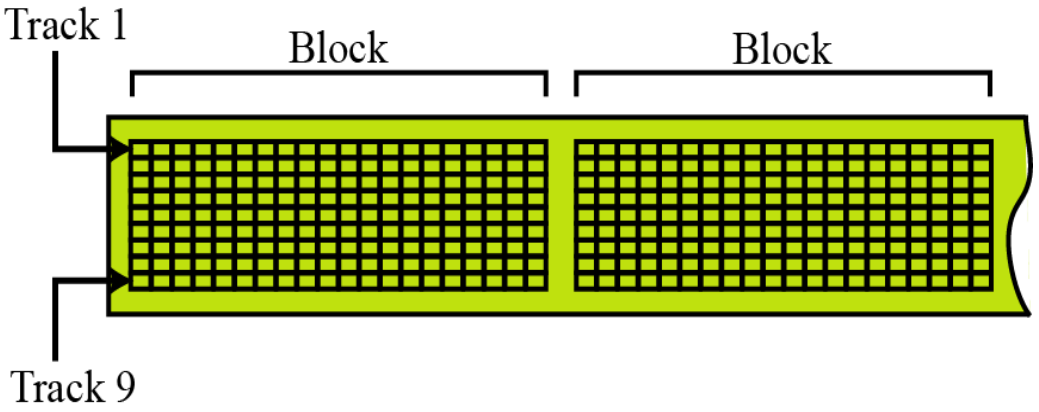


b. Tracks and Sectors

Un disco magnetico



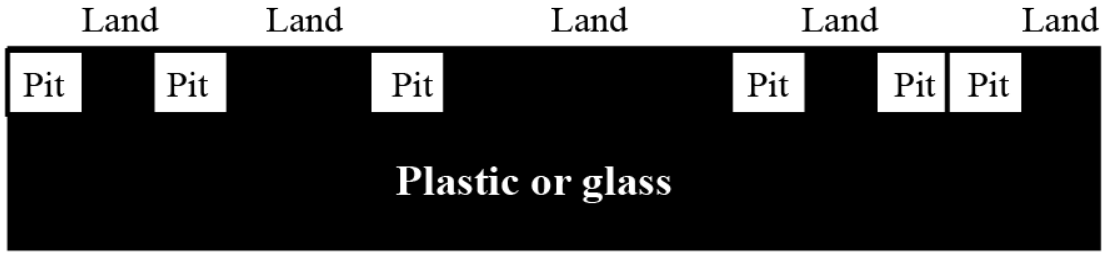
a. Tape drive



b. Surface orgnization

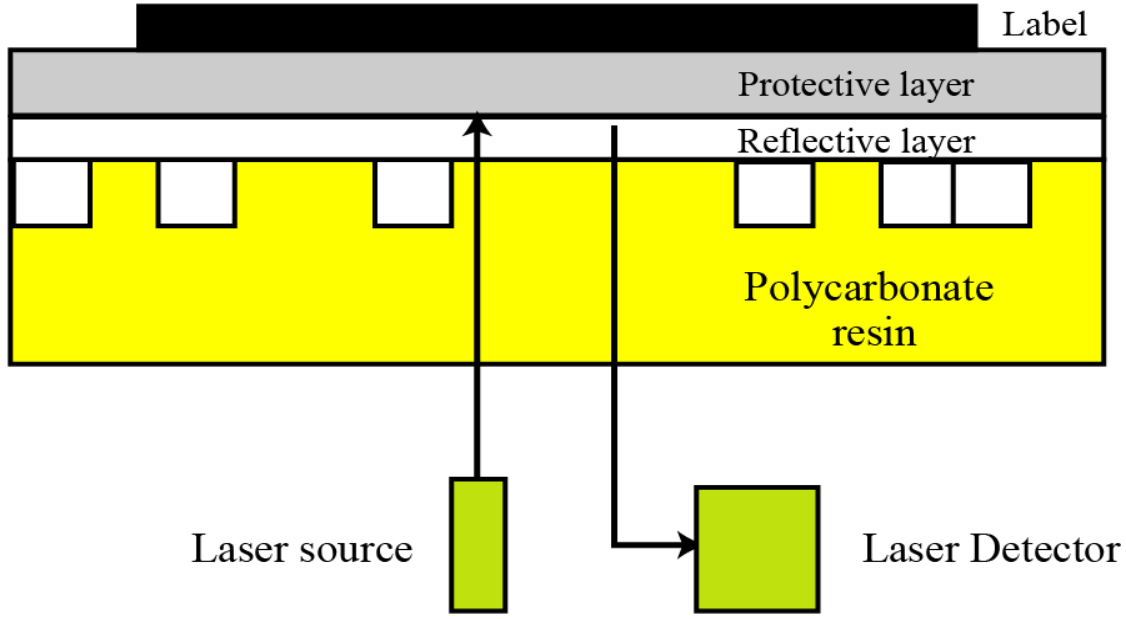
Una cinta magnetica

a. Master disc



b. Mold

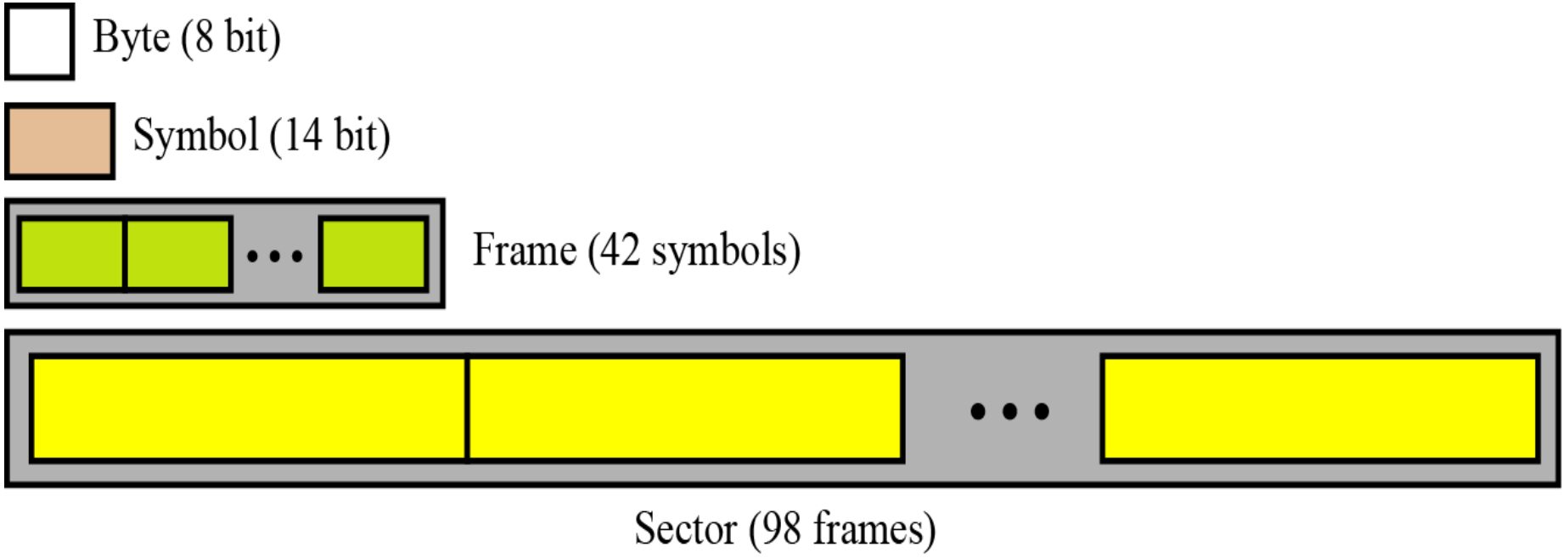
c. CD-ROM



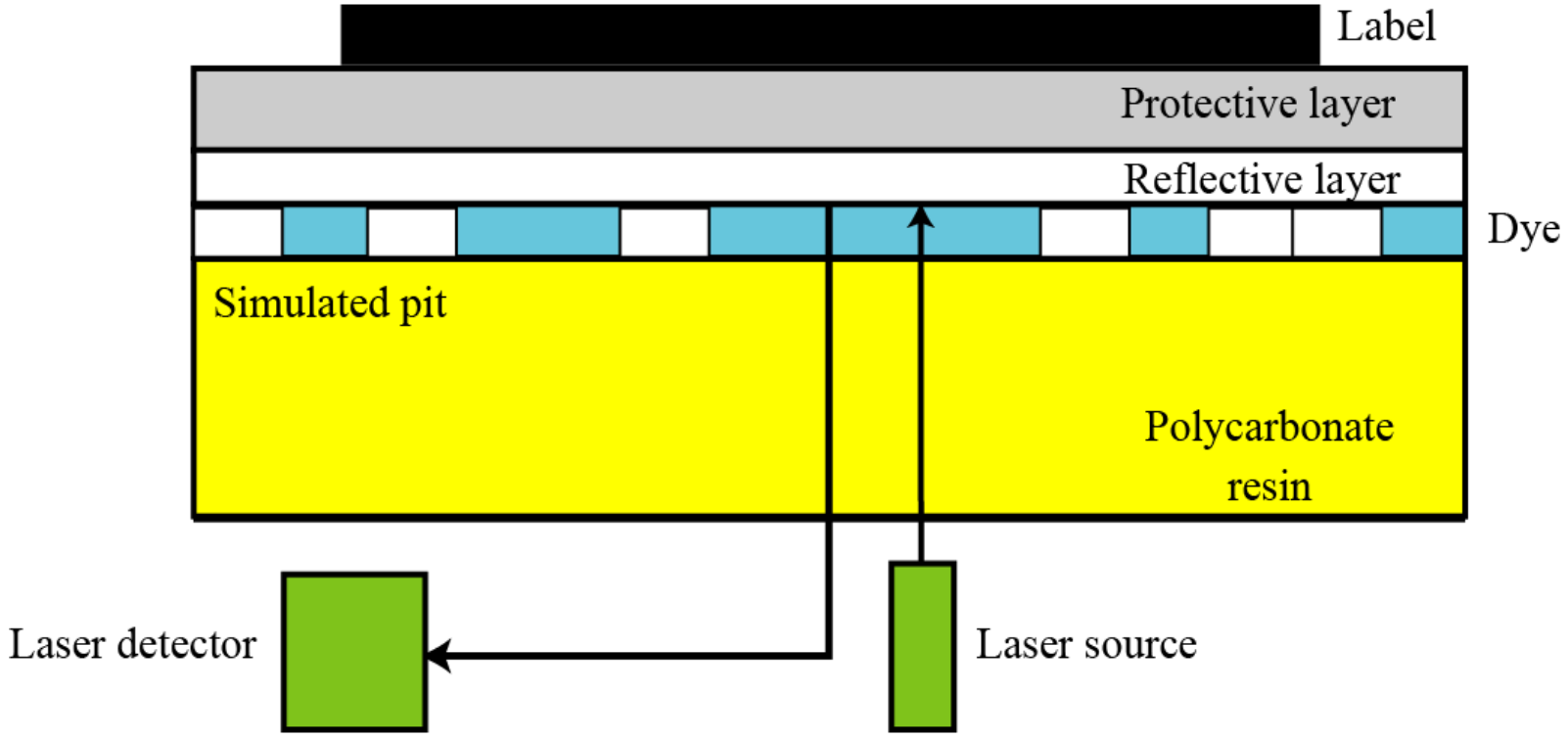
Creacion y uso del CD-ROM

Speed	*Data rate*	*Approximation*
1*x*	153,600 bytes per second	150 KB/s
2*x*	307,200 bytes per second	300 KB/s
4*x*	614,400 bytes per second	600 KB/s
6*x*	921,600 bytes per second	900 KB/s
8*x*	1,228,800 bytes per second	1.2 MB/s
12*x*	1,843,200 bytes per second	1.8 MB/s
16*x*	2,457,600 bytes per second	2.4 MB/s
24*x*	3,688,400 bytes per second	3.6 MB/s
32*x*	4,915,200 bytes per second	4.8 MB/s
40*x*	6,144,000 bytes per second	6 MB/s

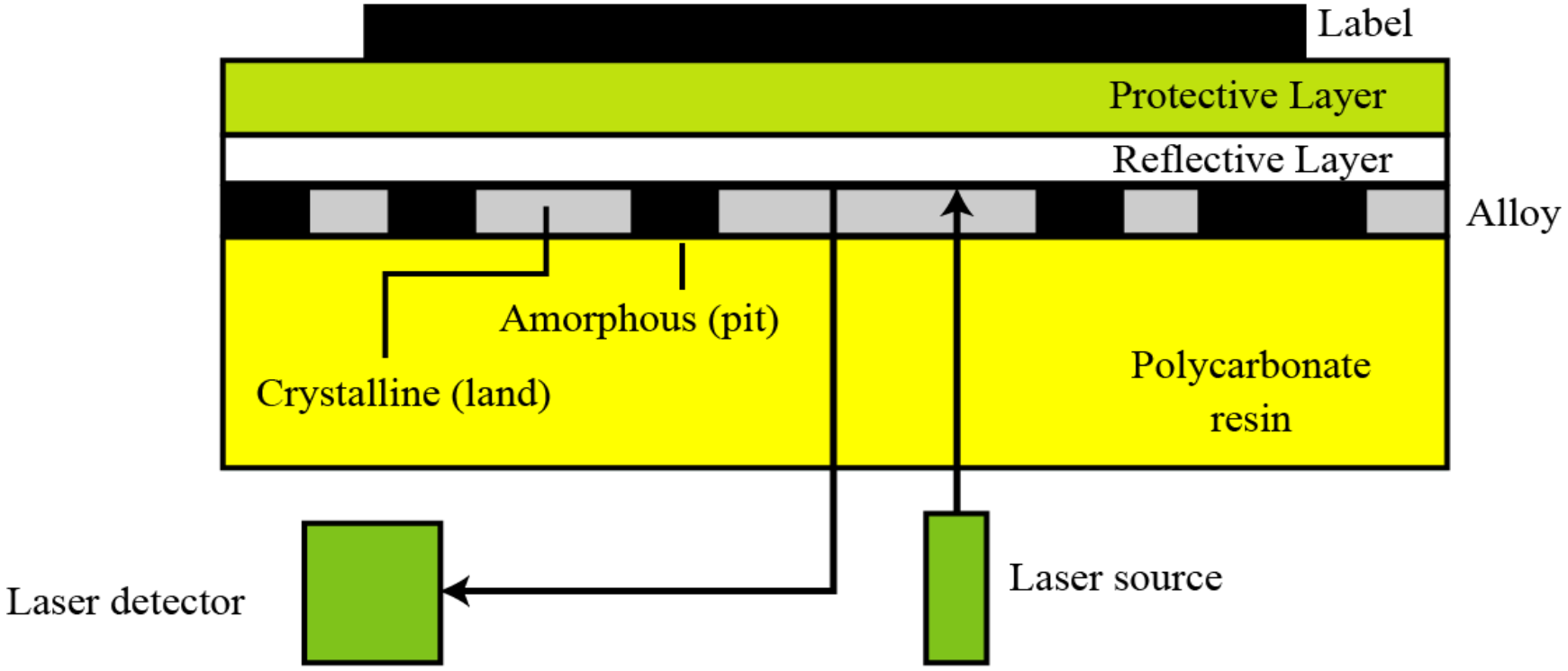
Velocidades de CD-ROM



Formato de CD-ROM



Haciendo un CD-ROM



Haciendo un CD-RW

Feature	*Capacity*
Single-sided, single-layer	4.7 GB
Single-sided, dual-layer	8.5 GB
Double-sided, single-layer	9.4 GB
Double-sided, dual-layer	17 GB

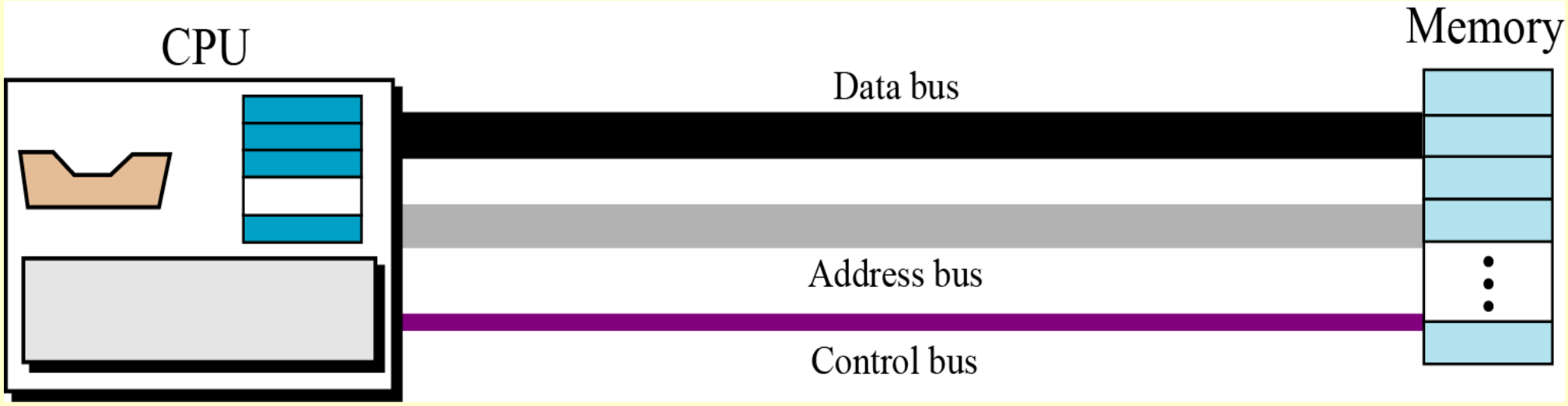
Capacidades del DVD

INTERCONEXIÓN DE SUBSISTEMAS

Las secciones anteriores describen las características de los tres subsistemas (CPU, memoria principal, y E/S) en un equipo independiente. En esta sección, se explora cómo estos tres subsistemas están interconectados. La interconexión juega un papel importante ya que la información debe ser intercambiada entre los tres subsistemas.

Conectando CPU y memoria

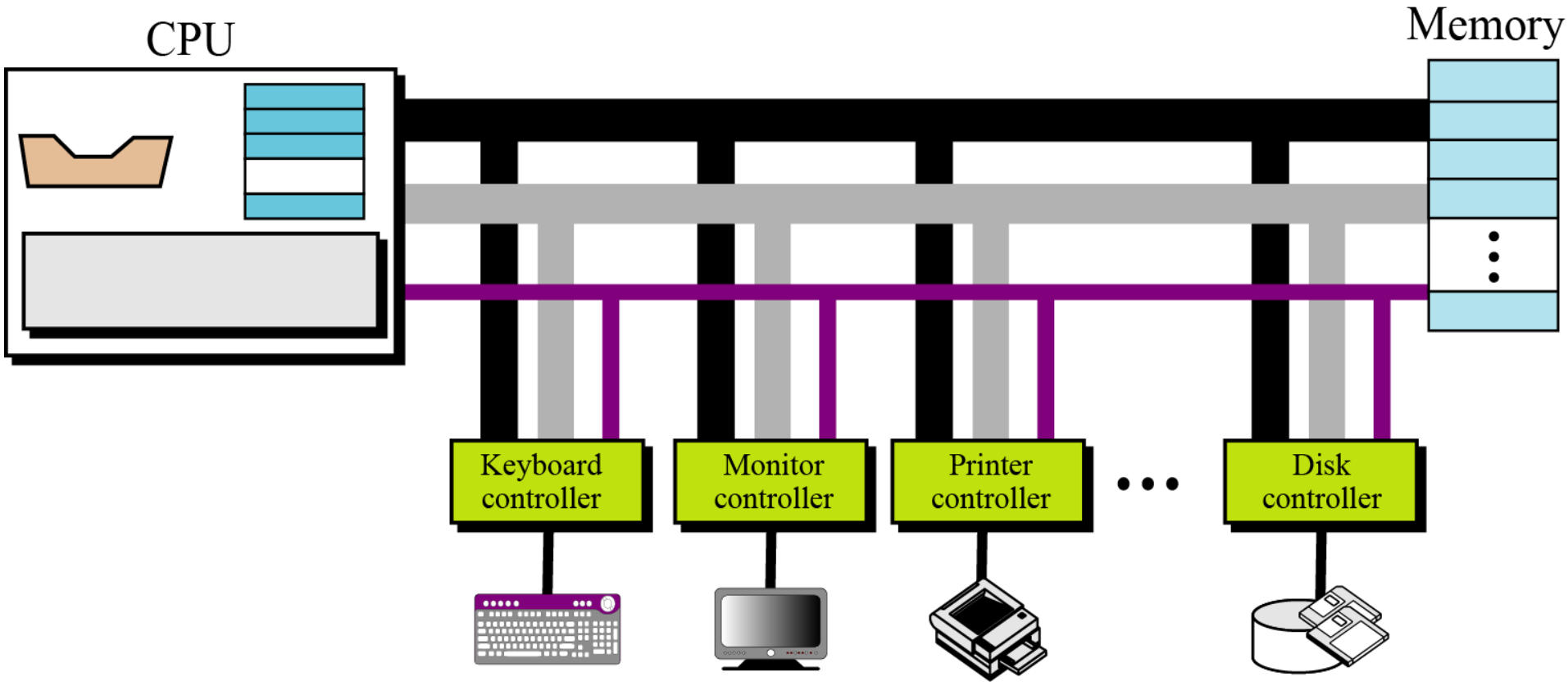
La memoria y el CPU estan normalmente conectados por tres grupos de conexiones, cada una llamada una **bus**: *data bus*, *address bus* y *control bus*.



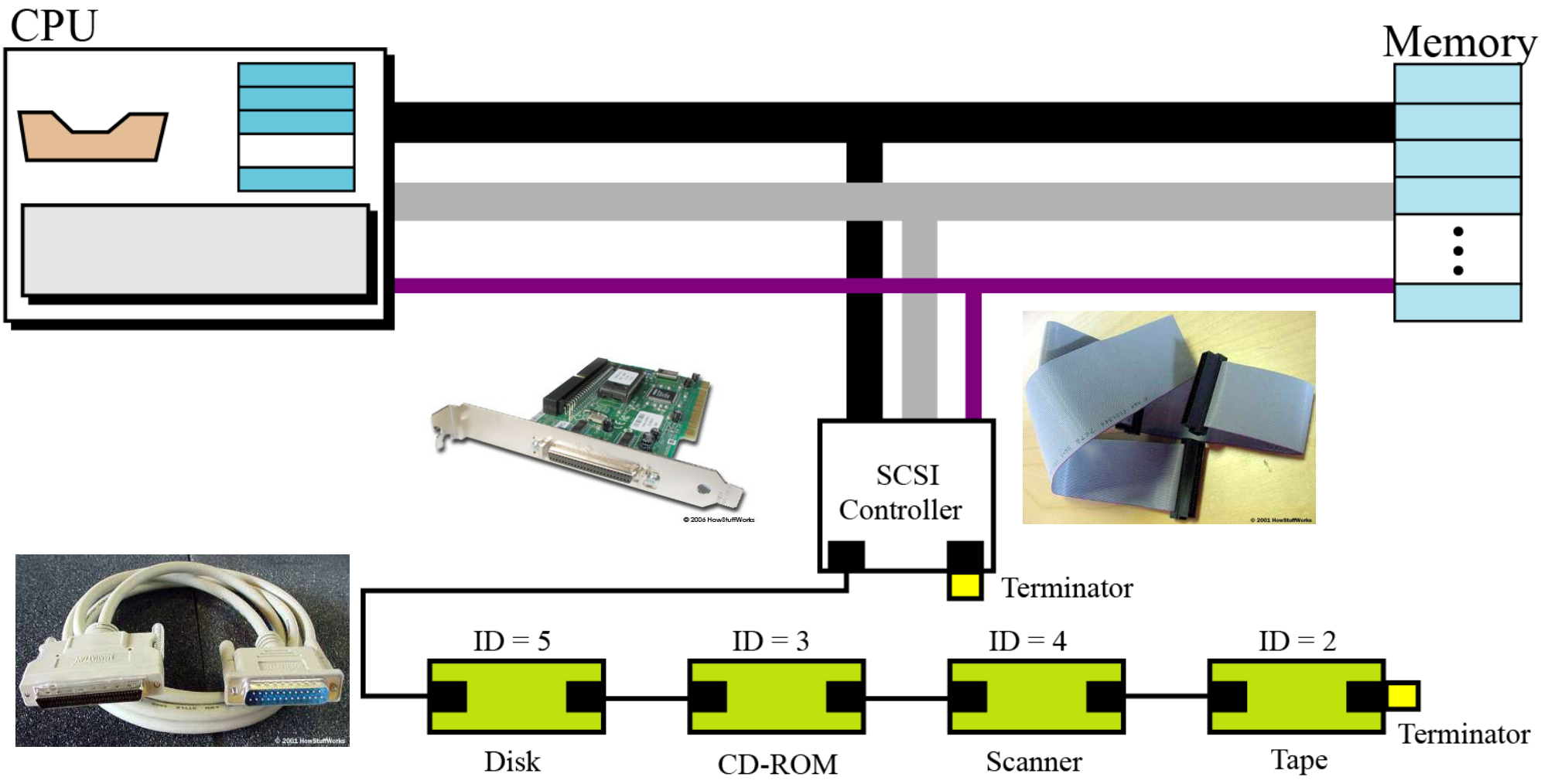
Conectando CPU y memoria usando tres buses

Conectando dispositivos E/S (I/O)

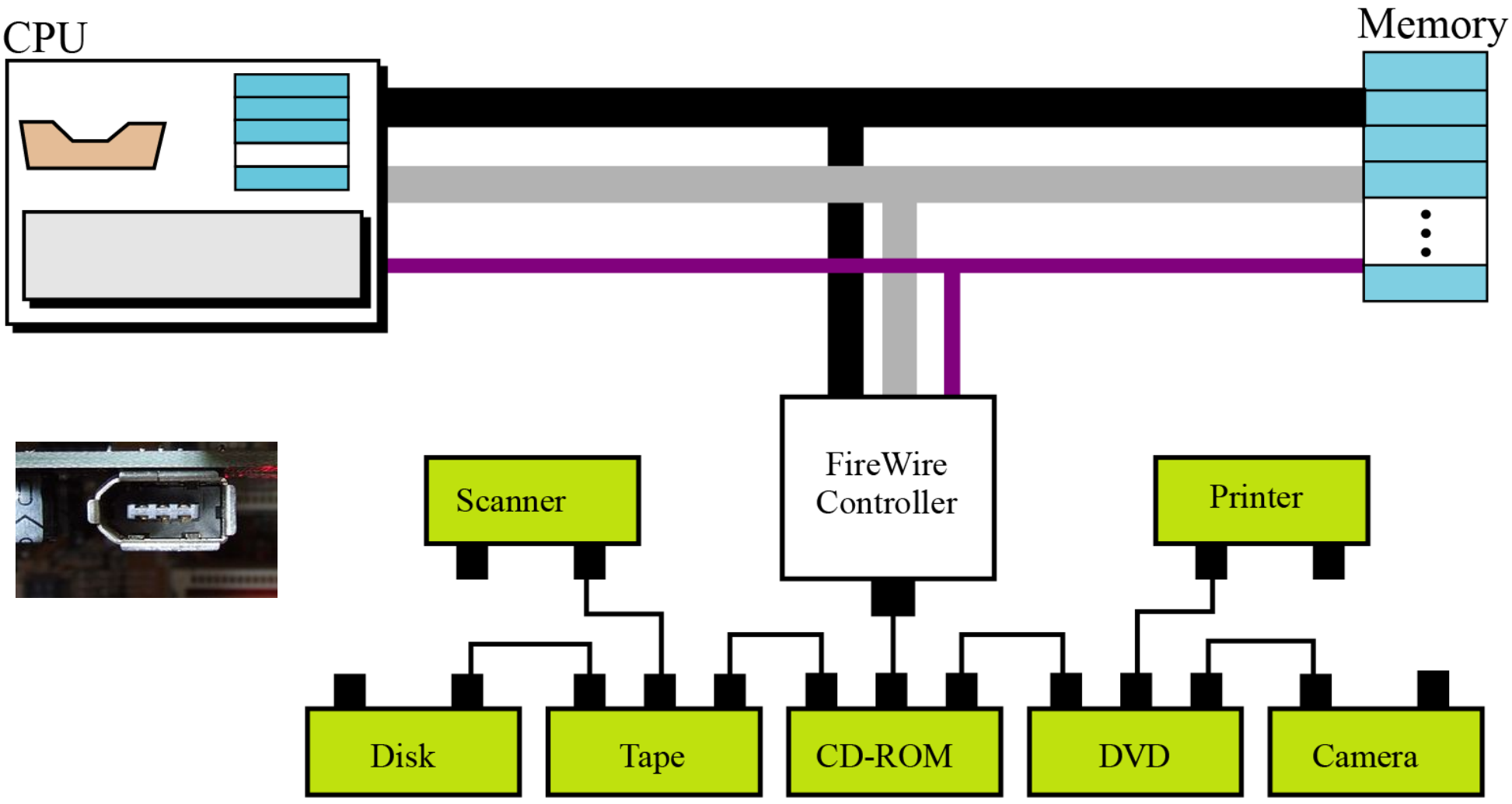
Dispositivos I/O no pueden ser conectados directamente a los buses que conectan la CPU y la memoria, porque la naturaleza de dispositivos I/O es diferente de la naturaleza de la CPU y la memoria. Dispositivos I/O son electromecánicos, magnéticos o dispositivos ópticos, mientras que la CPU y la memoria son dispositivos electrónicos. Dispositivos I/O también operan a una velocidad mucho más lenta que la CPU/memoria. Hay una necesidad de algún tipo de intermediario para manejar esta diferencia. Dispositivos de entrada/salida por consiguiente, se adjuntan a los buses a través de controladores de entrada/salida o interfaces. Hay un controlador específico para cada dispositivo de entrada/salida.



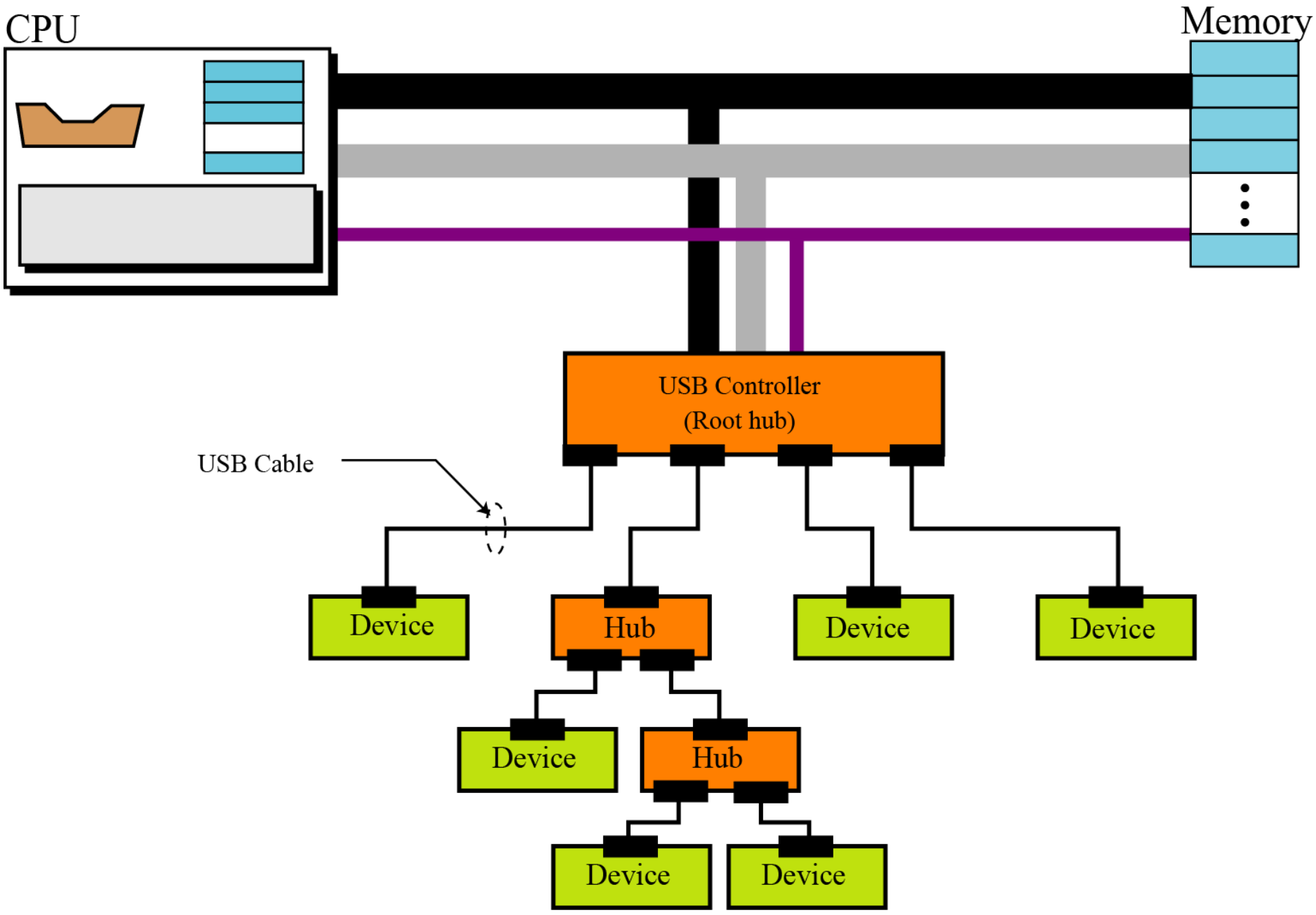
Conectando dispositivos I/O a los buses



Controlador SCSI (Small Computer System Device)



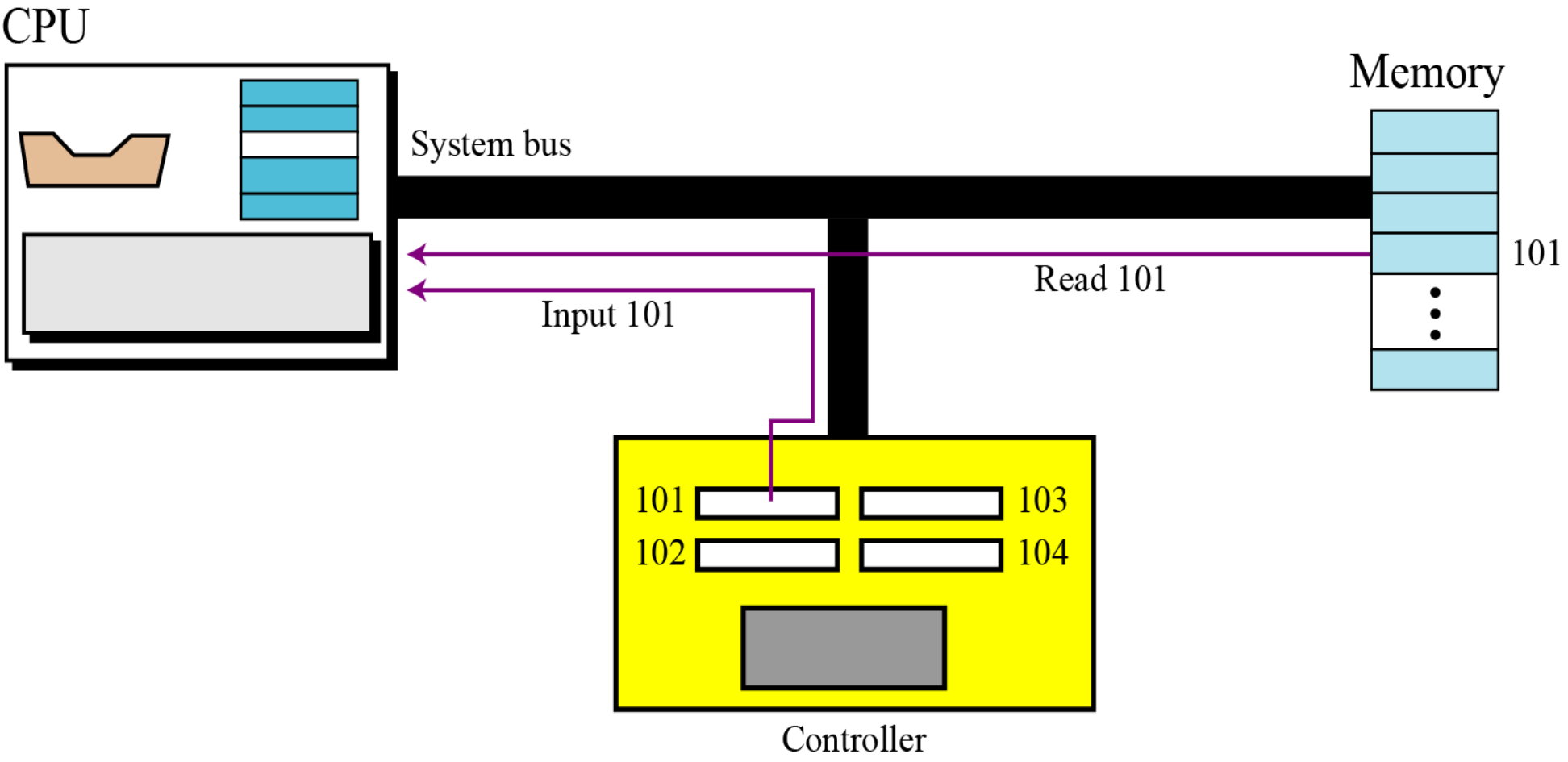
Controlador FireWire (iLink / IEEE 1394 -> peer-to-peer)



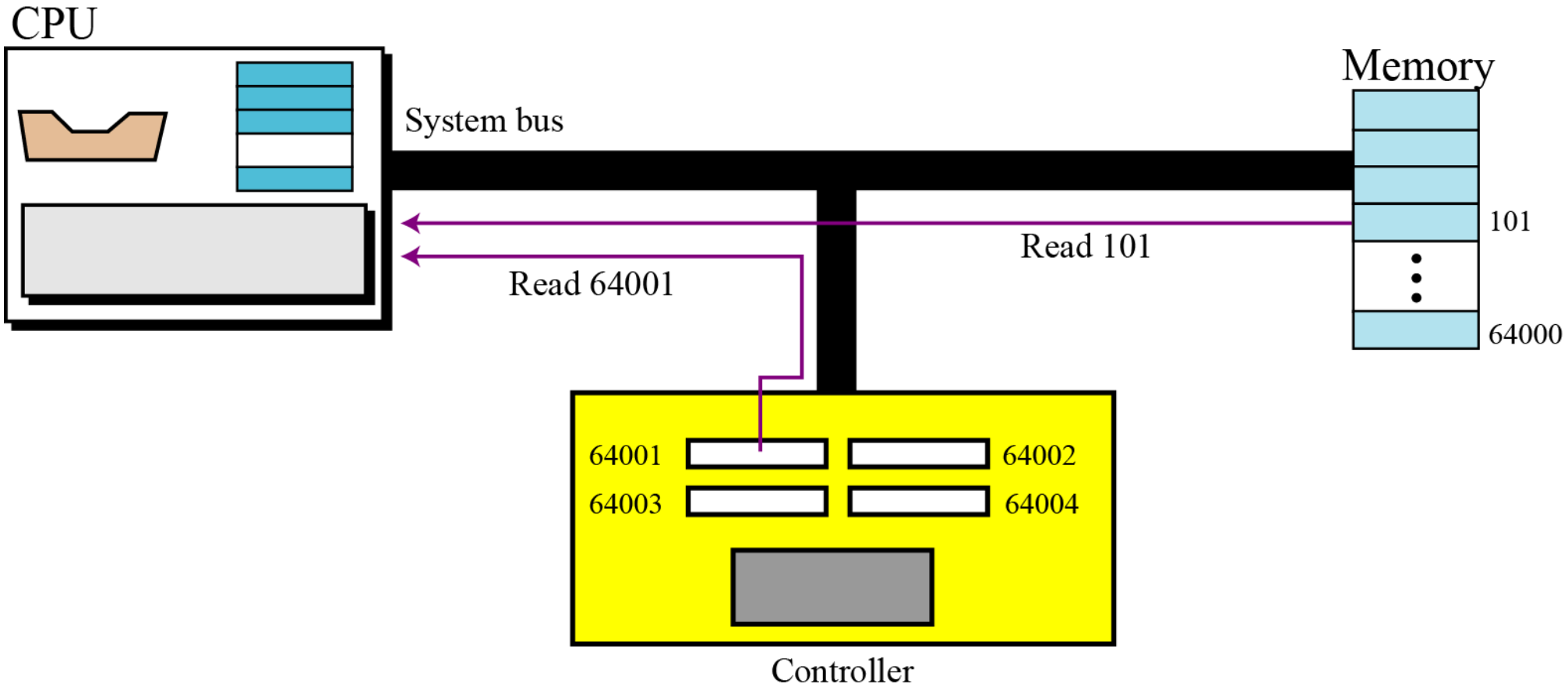
Controlador USB (Universal Serial Bus) → slave-master

Direccionamiento de dispositivos E/S (I/O)

La CPU suele utilizar el mismo bus de datos para leer o escribir datos en la memoria principal y de I/O. La única diferencia es la instrucción. Si la instrucción se refiere a una palabra en la memoria principal, la transferencia de datos es entre la memoria principal y la CPU. Si la instrucción se identifica un dispositivo de entrada-salida, la transferencia de datos entre el dispositivo I/O y la CPU. Hay dos métodos para el manejo de direccionamiento de dispositivos I/O: I/O aislado y memoria-mapeada I/O.



Direccionamiento de I/O aislado



Direccionamiento de I/O memoria-mapeado

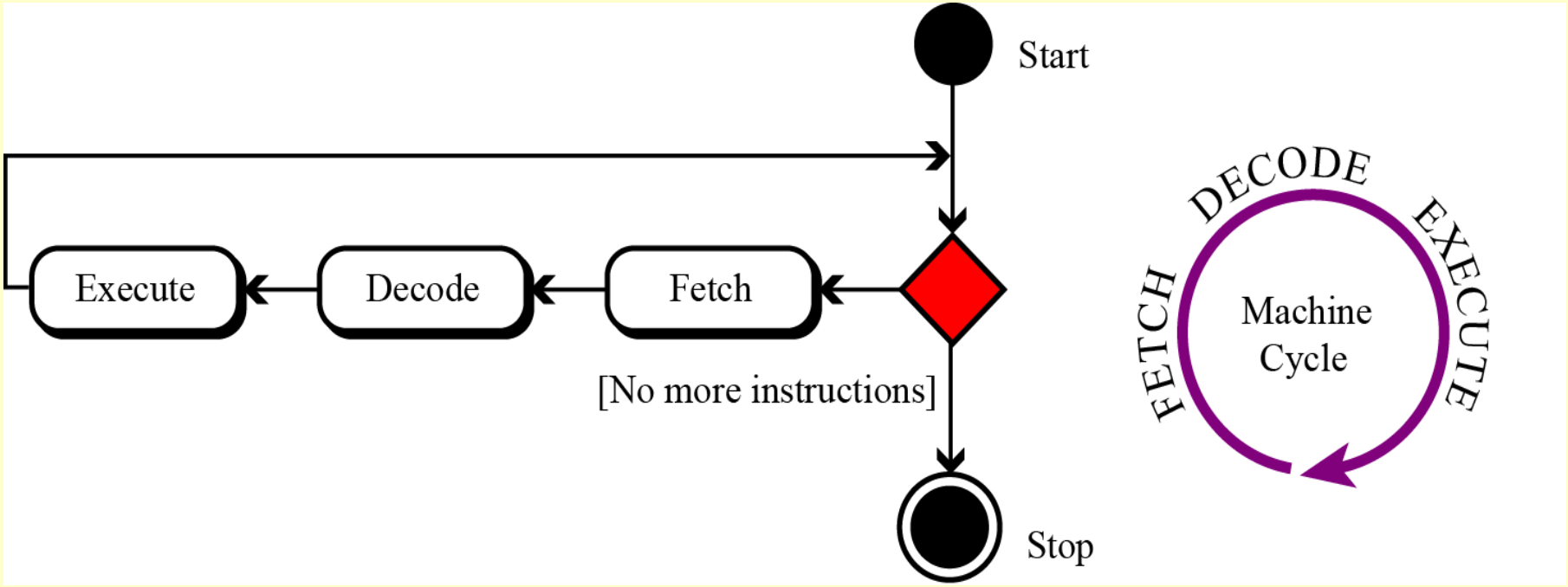
PROGRAMA DE EJECUCIÓN

Hoy en día, las **computadoras de propósito general** propósito general utilizan un conjunto de instrucciones llamado **programa** para procesar datos. Un computador ejecuta el programa para crear datos de salida a partir de datos de entrada. Tanto el programa como los datos se almacenan en la memoria.

Al final de este capítulo se dan algunos ejemplos de cómo un hipotético simple ordenador ejecuta un programa.

Ciclo de máquina

La CPU utiliza repetidamente ciclos de máquina para ejecutar las instrucciones en el programa, uno por uno, de principio a fin. Un ciclo simplificado puede constar de tres fases: buscar (***fetch***), decodificar (***decode***) y ejecutar (***execute***).

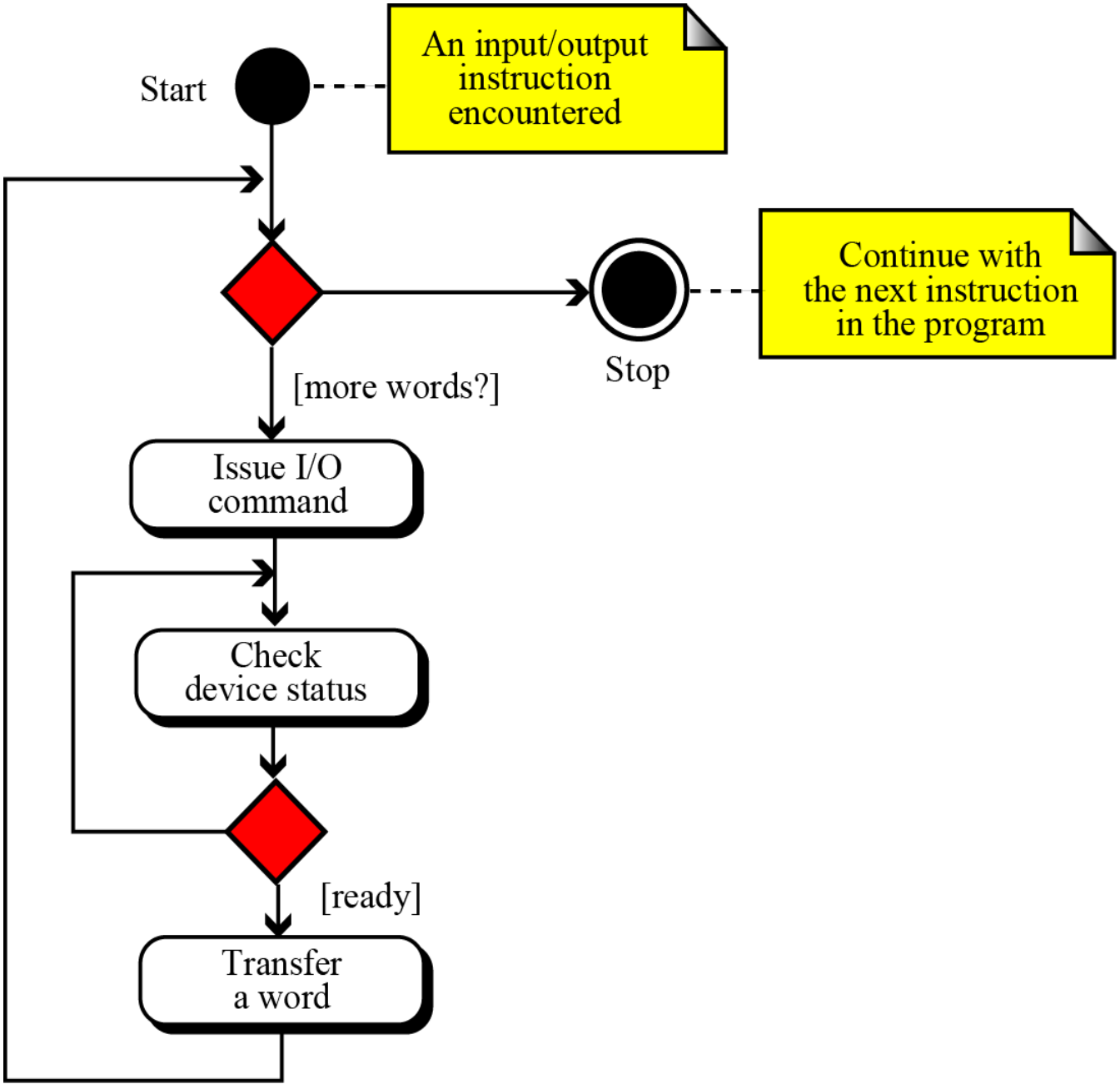


Los pasos de un ciclo

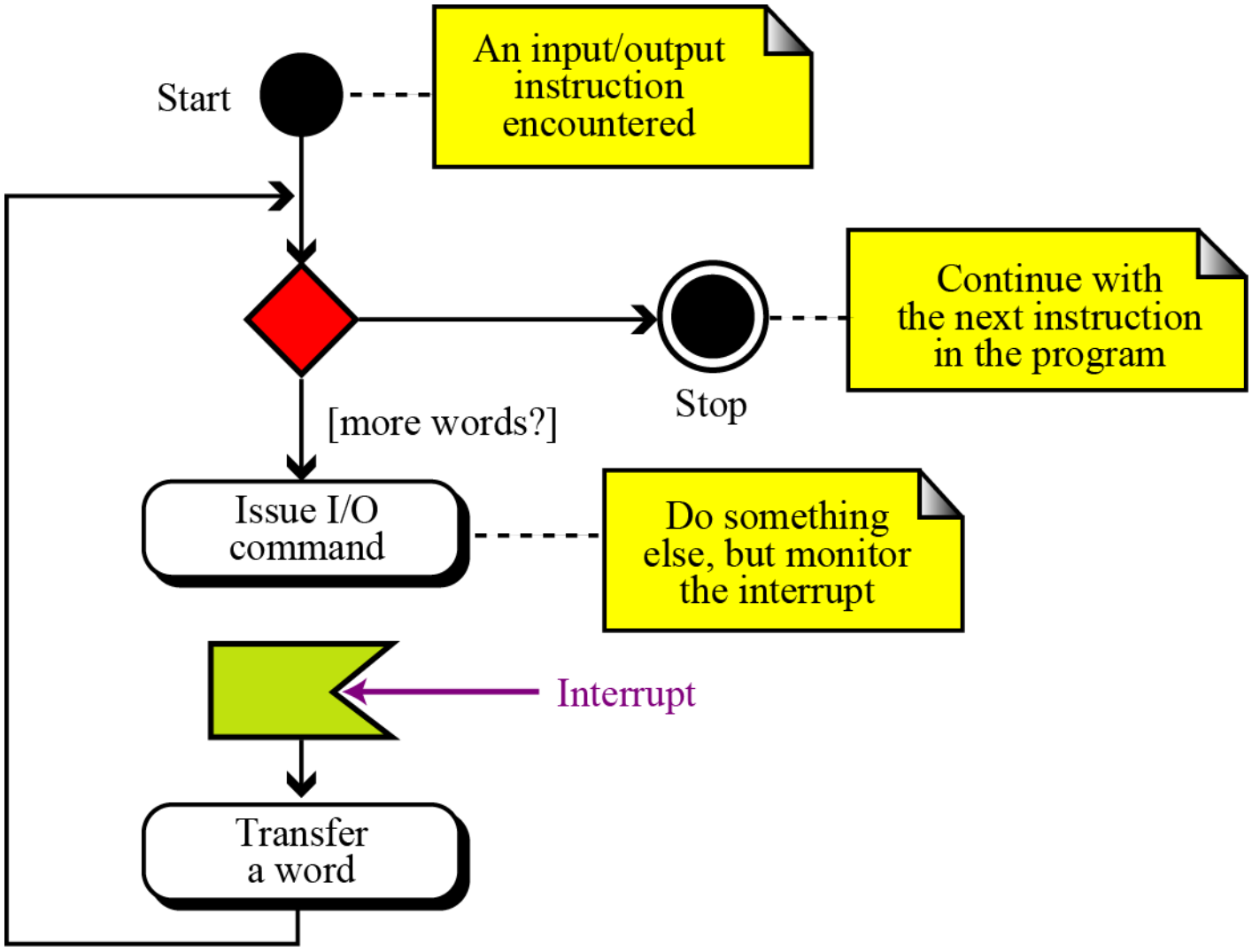
Operación entrada/salida

Los comandos son necesarios para transferencia de datos desde dispositivos I/O al CPU y la memoria. Debido a que los dispositivos I/O operan a velocidades mucho más lentas que el CPU, el funcionamiento de la CPU debe ser de alguna manera sincronizada con los dispositivos de entrada-salida. Tres métodos se han ideado para esta sincronización: *programmed I/O*, *interrupt driven I/O*, y *direct memory access (DMA)*.

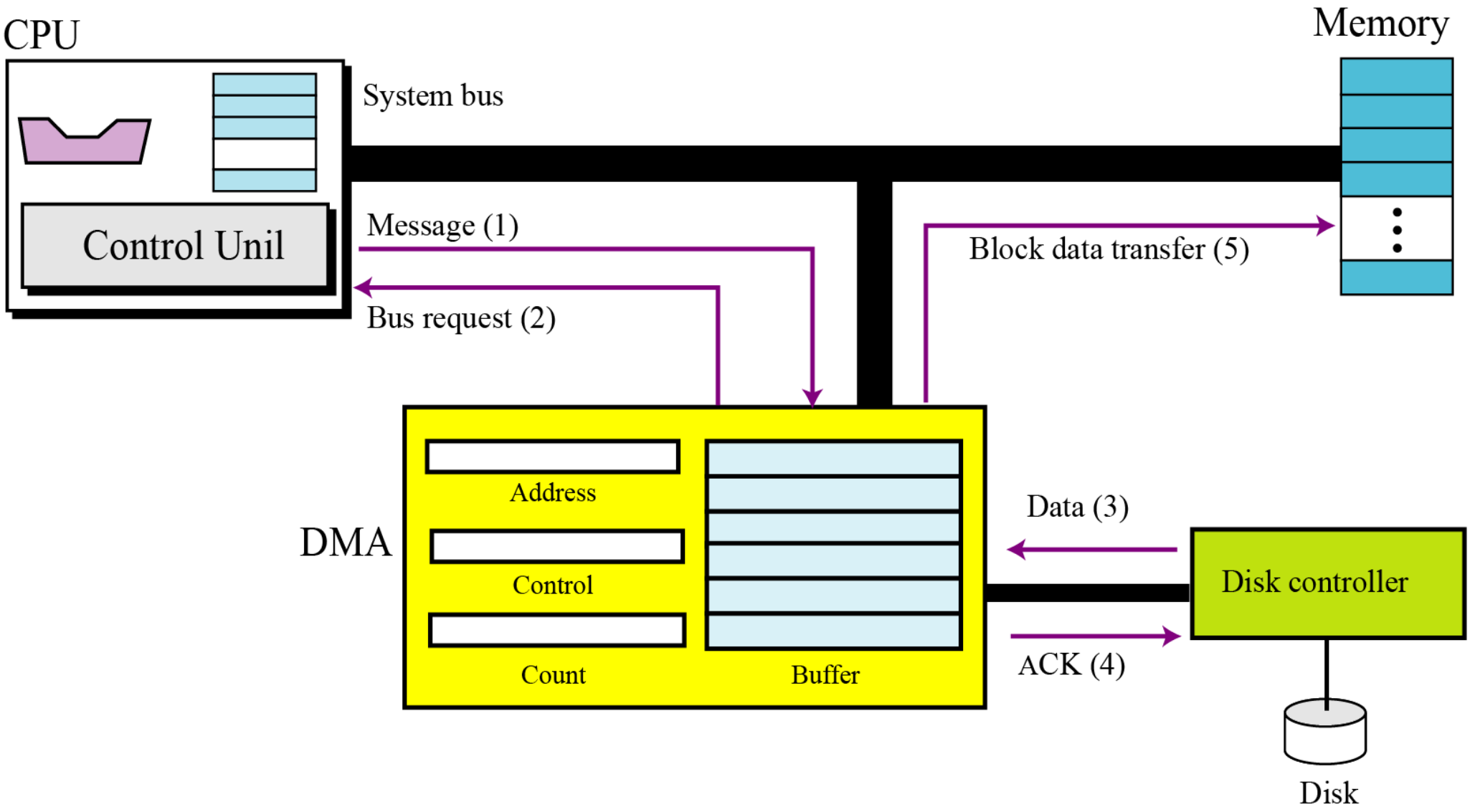
- **Programmed I/O**
- **Interrupt driven I/O**
- **Direct memory access (DMA)**



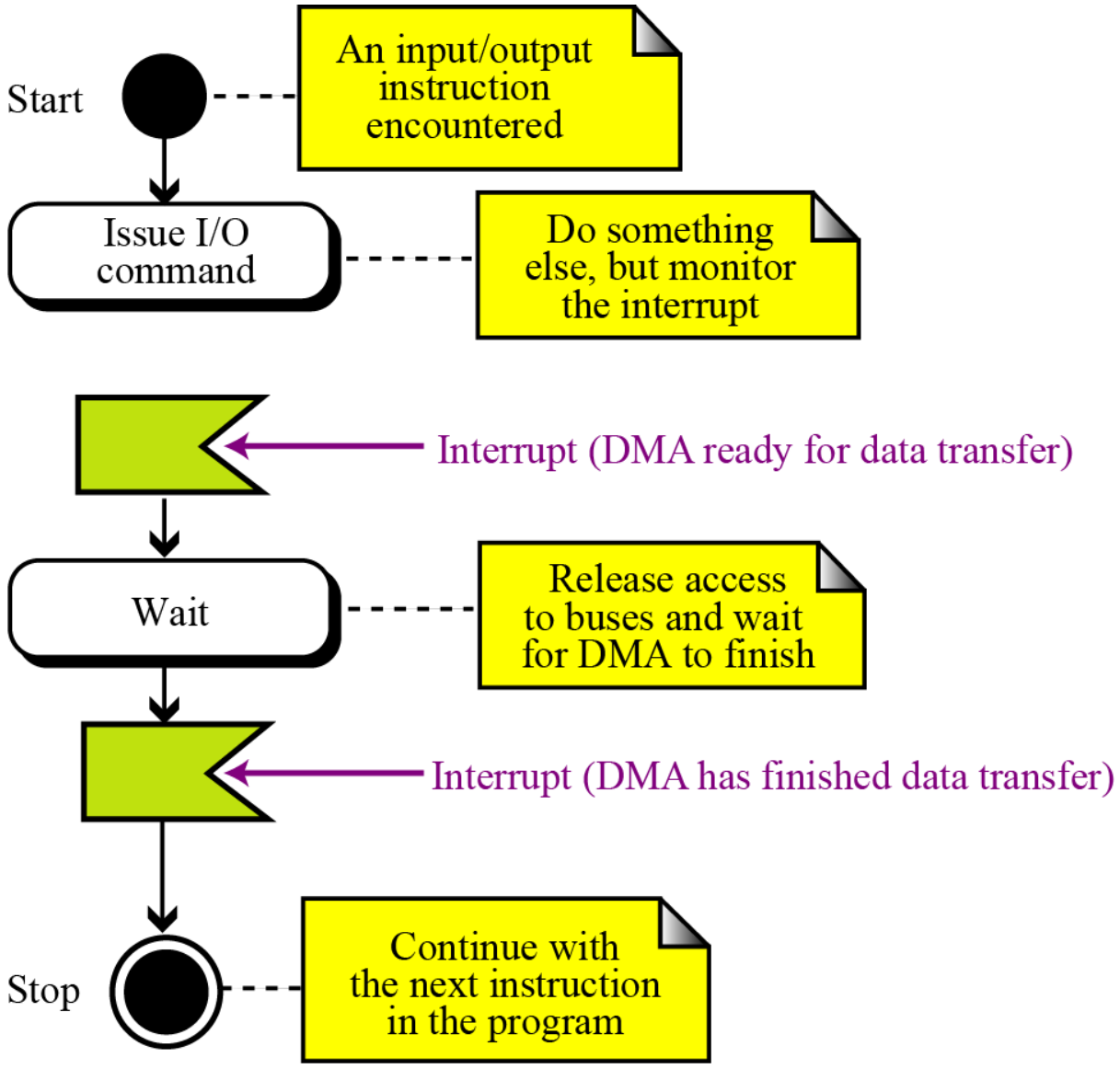
Programmed I/O



Interrupted-driven I/O



Conexión DMA al bus general



Entrada/salida DMA

DIFERENTES ARQUITECTURAS

La arquitectura y la organización de los ordenadores ha pasado por muchos cambios en las últimas décadas. Aqui discutiremos algunas arquitecturas comunes y la organización que difiere de la arquitectura simple del computador, que discutimos antes.

CISC

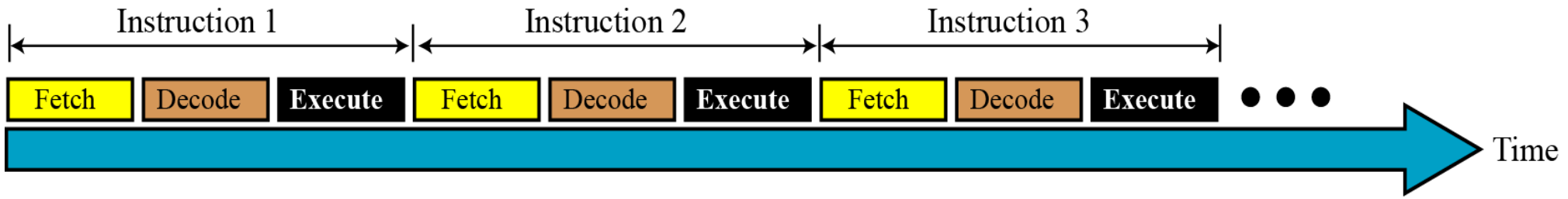
CISC (Complex Instruction Set Computer) es sinónimo de complejo conjunto de instrucciones del computador. La estrategia detrás de las arquitecturas CISC es tener un gran conjunto de instrucciones, incluyendo las más complejas. Programación de computadores basados en CISC es más fácil que en otros diseños porque hay una sola instrucción, tanto para tareas simples como complejas. Los programadores, por lo tanto, no tiene que escribir una serie de instrucciones para realizar una tarea compleja.

RISC

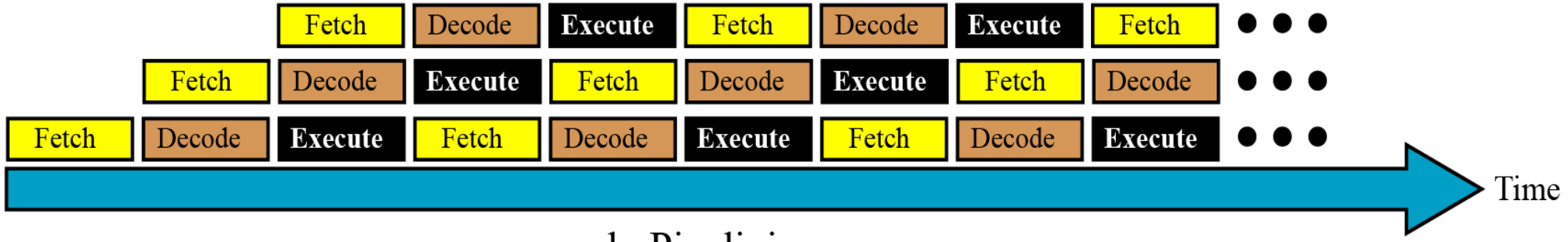
CISC (Reduced Instruction Set Computer) es sinónimo de conjunto reducido de instrucciones del computador. La estrategia detrás de la arquitectura RISC es tener un pequeño conjunto de instrucciones que hacen un número mínimo de operaciones simples. Instrucciones complejas son simuladas utilizando un subconjunto de instrucciones simples. Programación en RISC es más difícil y consume mas tiempo que en otros diseños, porque la mayoría de las instrucciones complejas son simuladas utilizando instrucciones sencillas.

Pipelining (canalización)

Hemos aprendido que un equipo utiliza tres fases, fetch, decode y execute, para cada instrucción. En las primeras computadoras, estas tres fases necesitaban ser hechas en serie para cada instrucción. En otras palabras, instrucción n tiene que terminar todas estas fases antes que la instrucción de n + 1 pueda comenzar sus propias fases. Las computadoras modernas utilizan una técnica llamada **pipelining** para mejorar el rendimiento (el número total de instrucciones realizadas en cada período de tiempo). La idea es que si la unidad de control puede hacer dos o tres de estas fases al mismo tiempo, la siguiente instrucción puede comenzar antes de que la anterior haya finalizado.



a. No pipelining

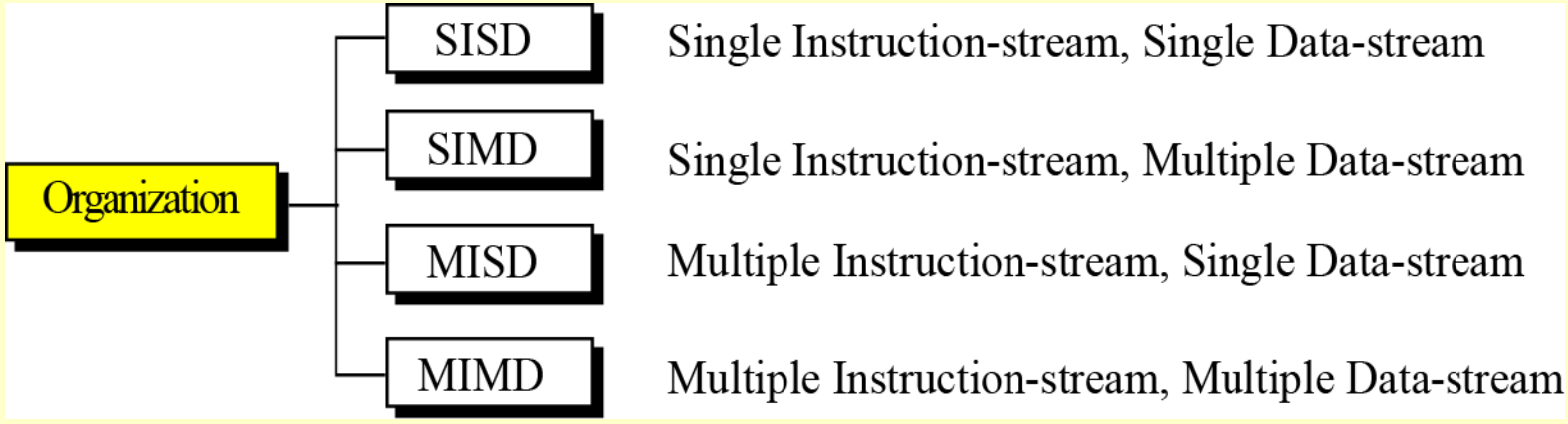


b. Pipelining

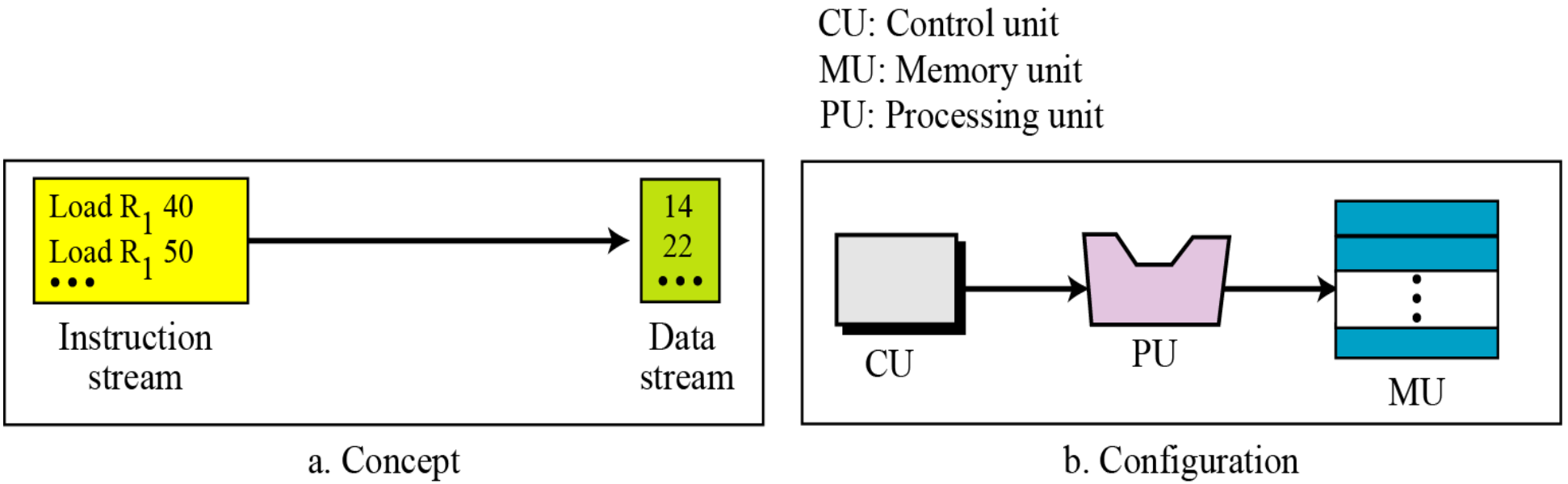
Pipelining

Procesamiento paralelo

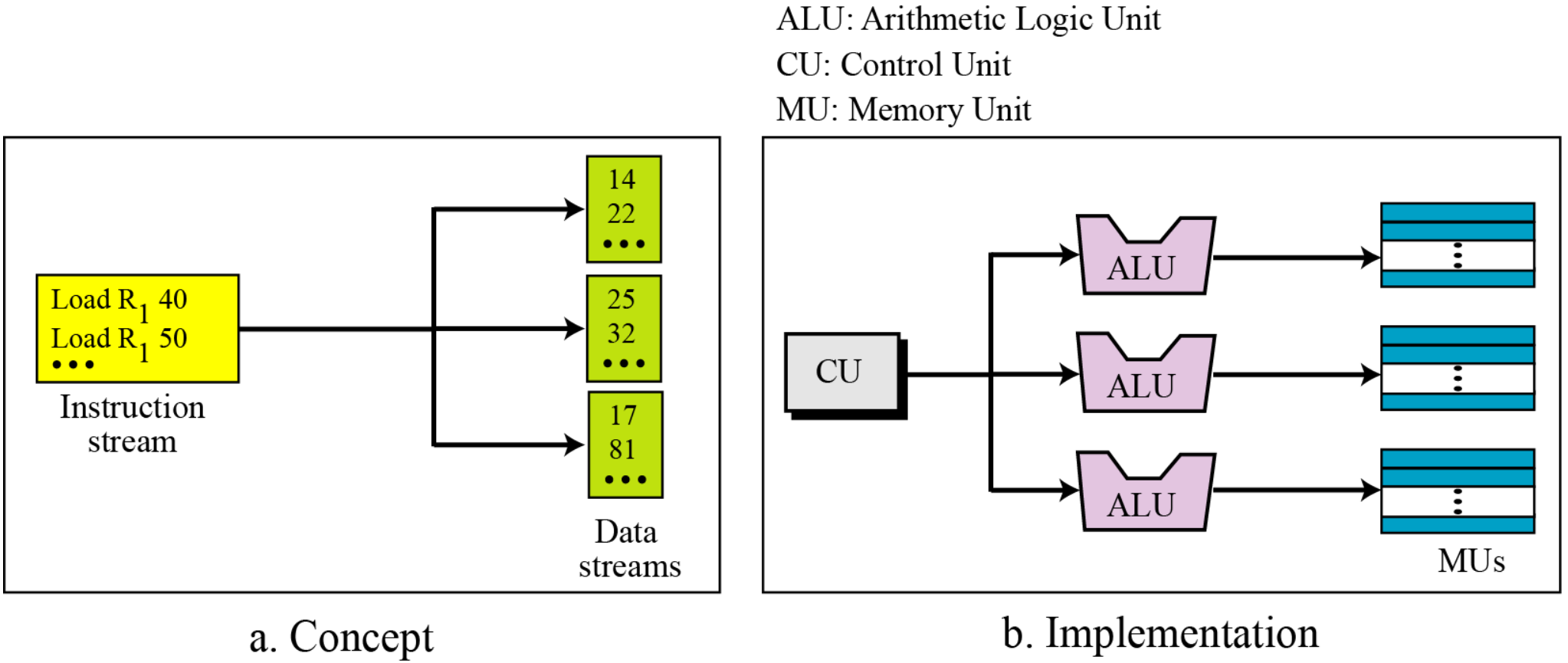
Tradicionalmente, un equipo tenía una sola unidad de control, una única unidad aritmética lógica y una simple unidad de memoria. Con la evolución de la tecnología y la caída en el costo del hardware del computador, hoy podemos tener un solo equipo con múltiples unidades de control, varias unidades aritméticas lógicas y múltiples unidades de memoria. Esta idea se conoce como **procesamiento paralelo**. Al igual que el pipelining, el procesamiento paralelo puede mejorar el rendimiento.



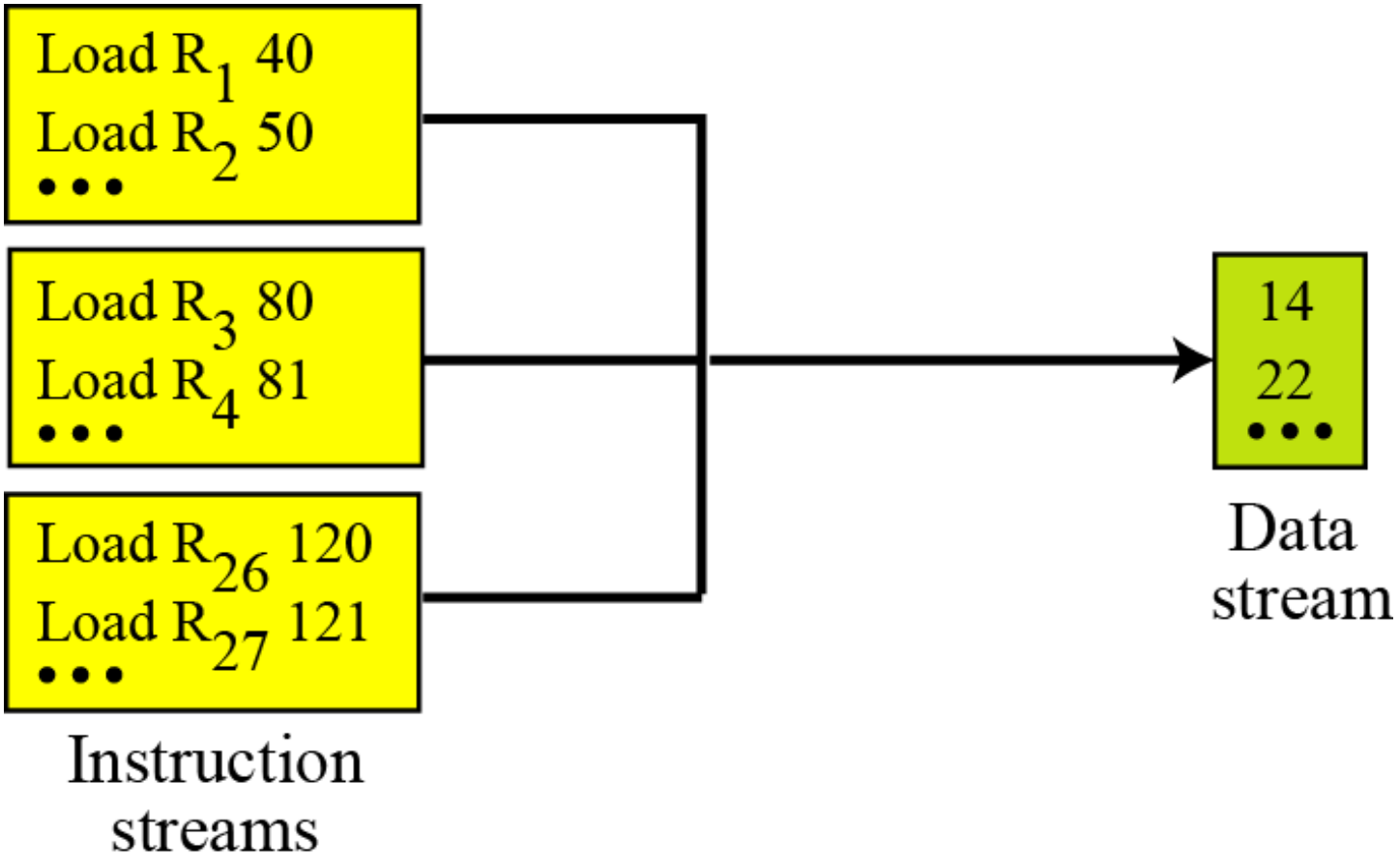
Una taxonomía de la organización del computador



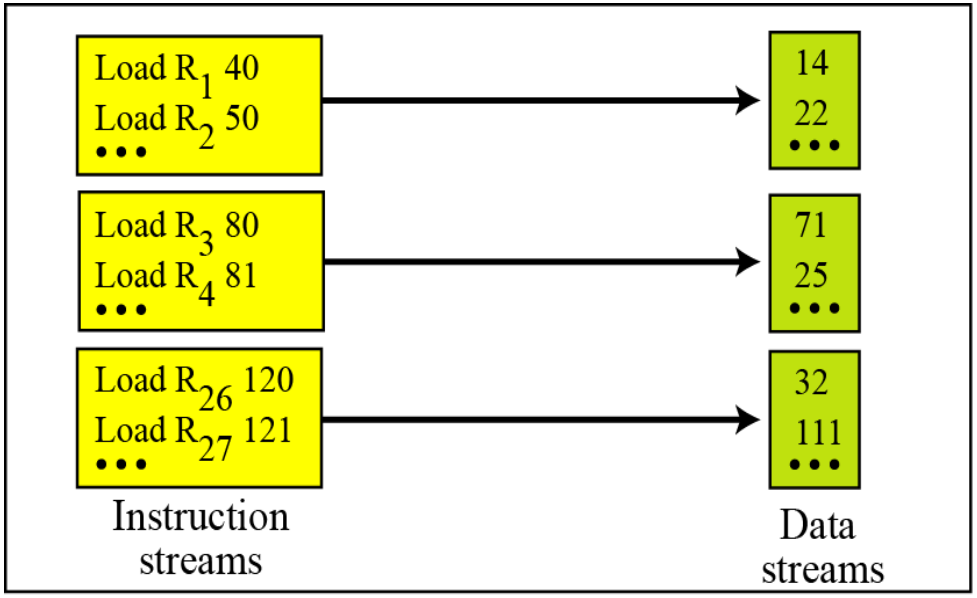
Organización SISD



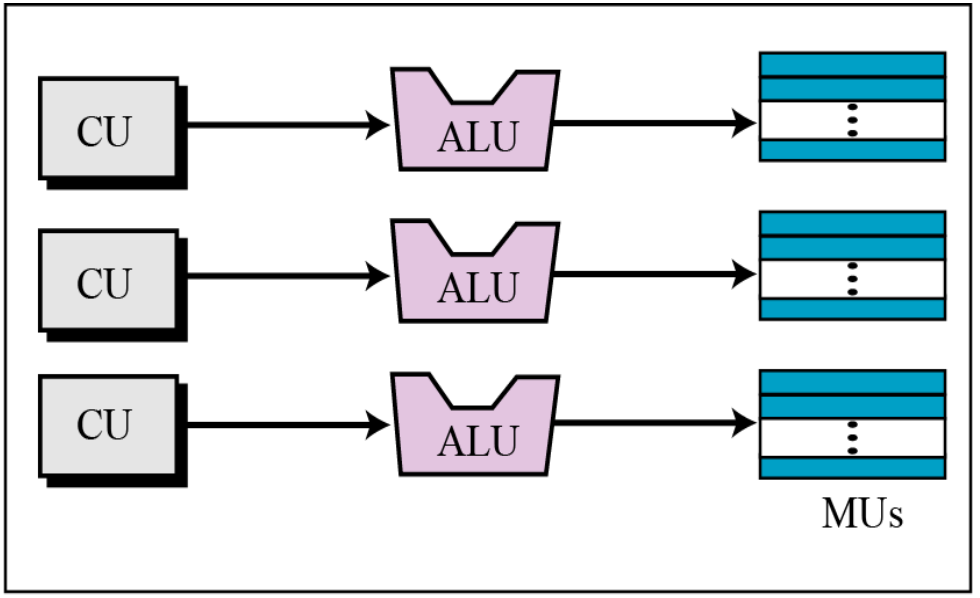
Organización SIMD



Organización MISD



a. Concept

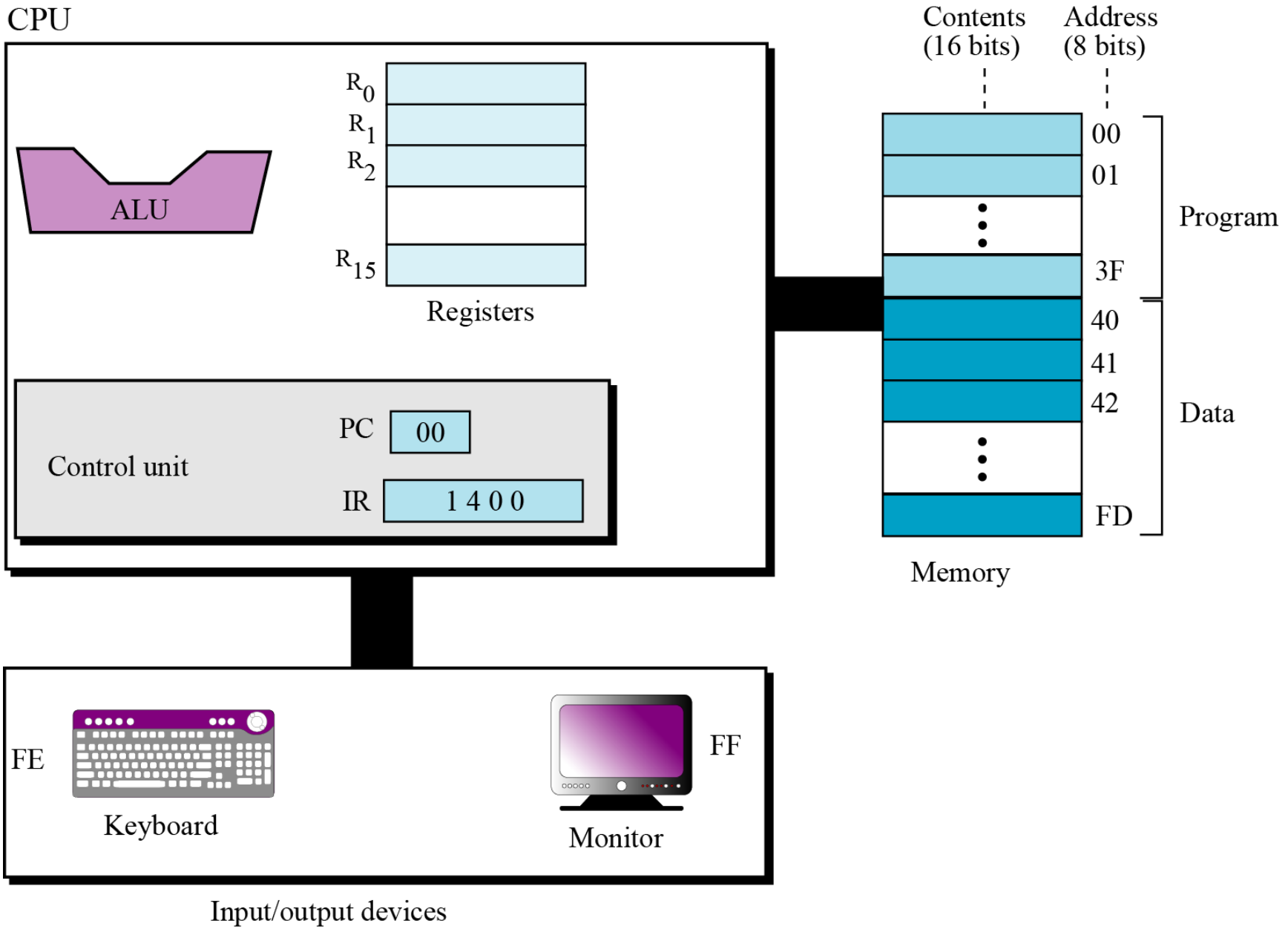


b. Implementation

Organización MIMD

UN SIMPLE COMPUTADOR

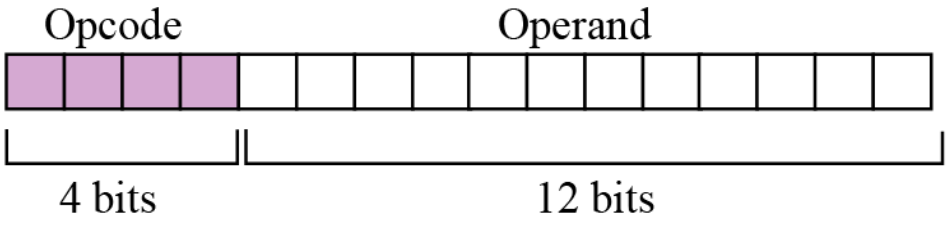
Para explicar la arquitectura de los ordenadores, así como su proceso de instrucción, se introduce un simple (no-realista) computador. Nuestro sencillo computador tiene tres componentes: la CPU, la memoria y un subsistema de entrada/salida.



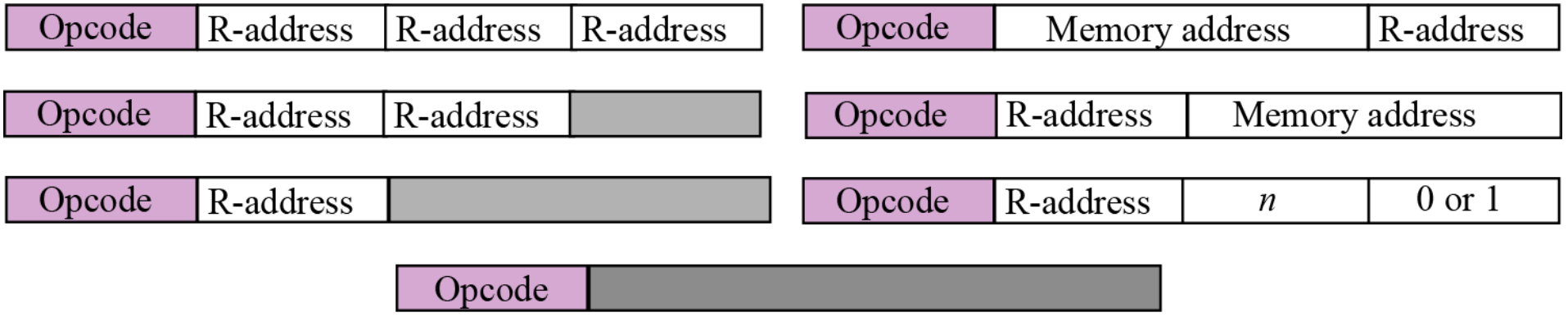
Componentes de un computador simple

Conjunto de instrucciones

Nuestro sencillo computador es capaz de tener un conjunto de instrucciones de dieciséis instrucciones, a pesar de que está utilizando sólo catorce de estas instrucciones. Cada instrucción del computador se compone de dos partes: el **código de operación** (opcode) y el **operando**(s). El opcode especifica el tipo de operación que se realizará en el operando(s). Cada instrucción consta de dieciséis bits dividido en cuatro campos de 4 bits. El campo mas a la izquierda contiene el opcode y los otros tres campos contiene el operando o la dirección del operando(s).



a. Instruction format



b. Instruction types

Formato y diferentes tipos de instrucciones

Procesando las instrucciones

Nuestro simple computador, como la mayoría de los equipos, utiliza ciclos de máquina. Un ciclo se compone de tres fases: buscar, decodificar y ejecutar. Durante la fase de buscar, la instrucción cuya dirección está determinada por el PC se obtiene de la memoria y se carga en el IR. El PC se incrementa para apuntar a la siguiente instrucción. Durante la fase de decodificación, la instrucción en el IR es decodificada y los operandos necesarios se buscan en el registro o en la memoria. Durante la fase de ejecución, la instrucción se ejecuta y los resultados se colocan en la posición adecuada de memoria o de registro. Una vez que la tercera fase se ha completado, la unidad de control se inicia el ciclo de nuevo, pero ahora el PC está apuntando a la siguiente instrucción. El proceso continúa hasta que la CPU llega a una instrucción HALT.

Instruction	Code	Operands			Action
	d_1	d_2	d_3	d_4	
HALT	0				Stops the execution of the program
LOAD	1	R_D	M_S		$R_D \leftarrow M_S$
STORE	2	M_D		R_S	$M_D \leftarrow R_S$
ADDI	3	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
ADDF	4	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
MOVE	5	R_D	R_S		$R_D \leftarrow R_S$
NOT	6	R_D	R_S		$R_D \leftarrow \overline{R_S}$
AND	7	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ AND R_{S2}
OR	8	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ OR R_{S2}
XOR	9	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ XOR R_{S2}
INC	A	R			R $\leftarrow$ R + 1
DEC	B	R			R $\leftarrow$ R – 1
ROTATE	C	R	n	0 or 1	Rot_n R
JUMP	D	R	n		IF $R_0 \neq$ R then PC = *n*, otherwise continue

Key: R_S, R_{S1}, R_{S2}: Hexadecimal address of source registers
R_D: Hexadecimal address of destination register
M_S: Hexadecimal address of source memory location
M_D: Hexadecimal address of destination memory location
n: hexadecimal number
d_1, d_2, d_3, d_4: First, second, third, and fourth hexadecimal digits

Un ejemplo

Vamos a mostrar cómo nuestro ordenador simple puede agregar dos números enteros A y B y crear el resultado como C. Se supone que los enteros se encuentran en formato complemento de dos. Matemáticamente, se muestra esta operación como:

$$C = A + B$$

Suponemos que los dos primeros números enteros se almacenan en posiciones de memoria $(40)_{16}$ y $(41)_{16}$ y el resultado debe ser almacenado en posición de memoria $(42)_{16}$. Para hacer la simple adición se necesita cinco instrucciones:

1. Load the contents of M_{40} into register R_0 ($R_0 \leftarrow M_{40}$).
2. Load the contents of M_{41} into register R_1 ($R_1 \leftarrow M_{41}$).
3. Add the contents of R_0 and R_1 and place the result in R_2 ($R_2 \leftarrow R_0 + R_1$).
4. Store the contents R_2 in M_{42} ($M_{42} \leftarrow$ R2).
5. Halt.

En el lenguaje de nuestro simple computador, esas cinco instrucciones son codificadas como:

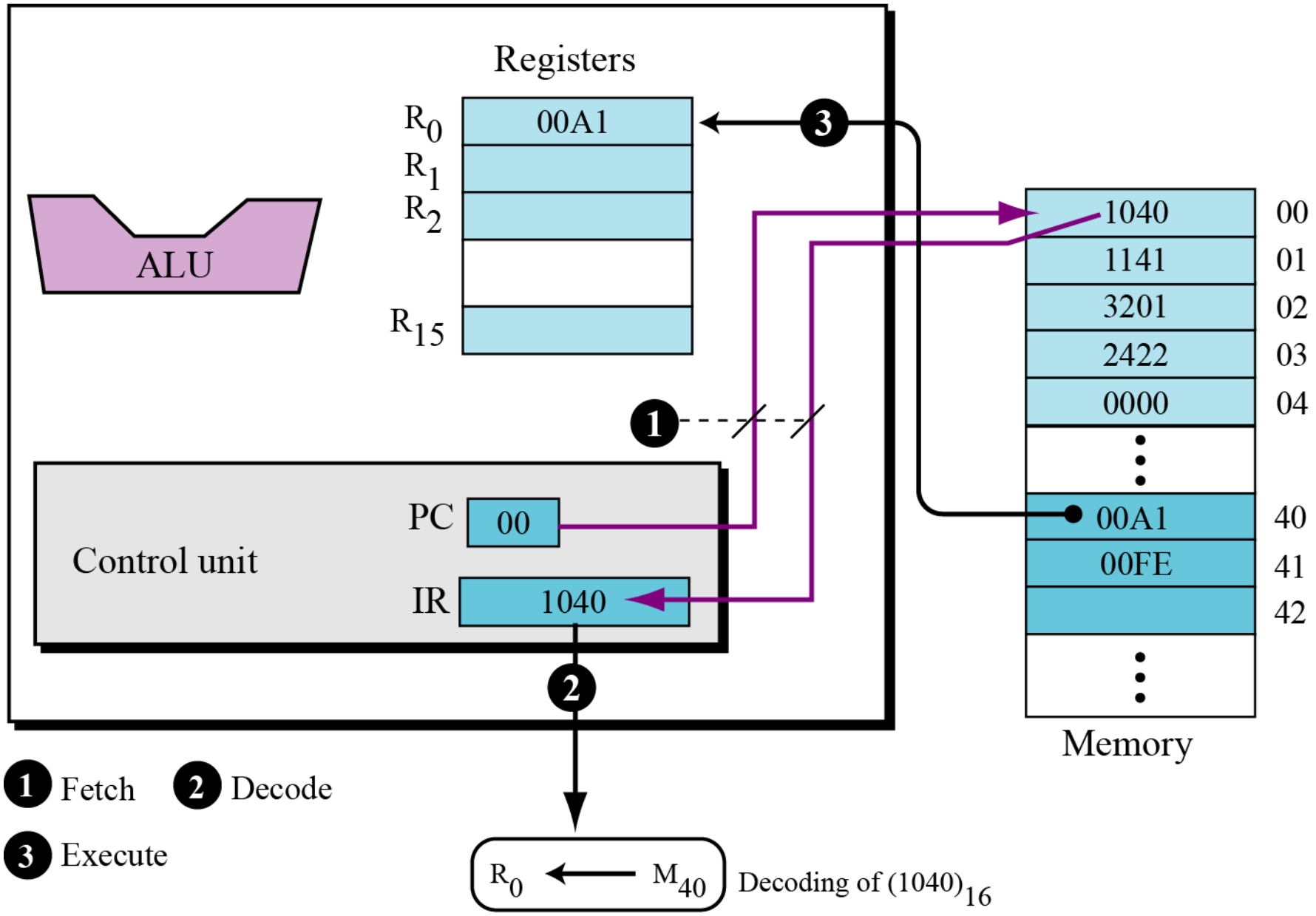
Code	*Interpretation*			
$(1040)_{16}$	**1**: LOAD	0: R_0	40: M_{40}	
$(1141)_{16}$	**1**: LOAD	1: R_1	41: M_{41}	
$(3201)_{16}$	**3**: ADDI	2: R_2	0: R_0	1: R_1
$(2422)_{16}$	**2**: STORE	42: M_{42}		2: R_2
$(0000)_{16}$	**0**: HALT			

Almacenando programa y datos

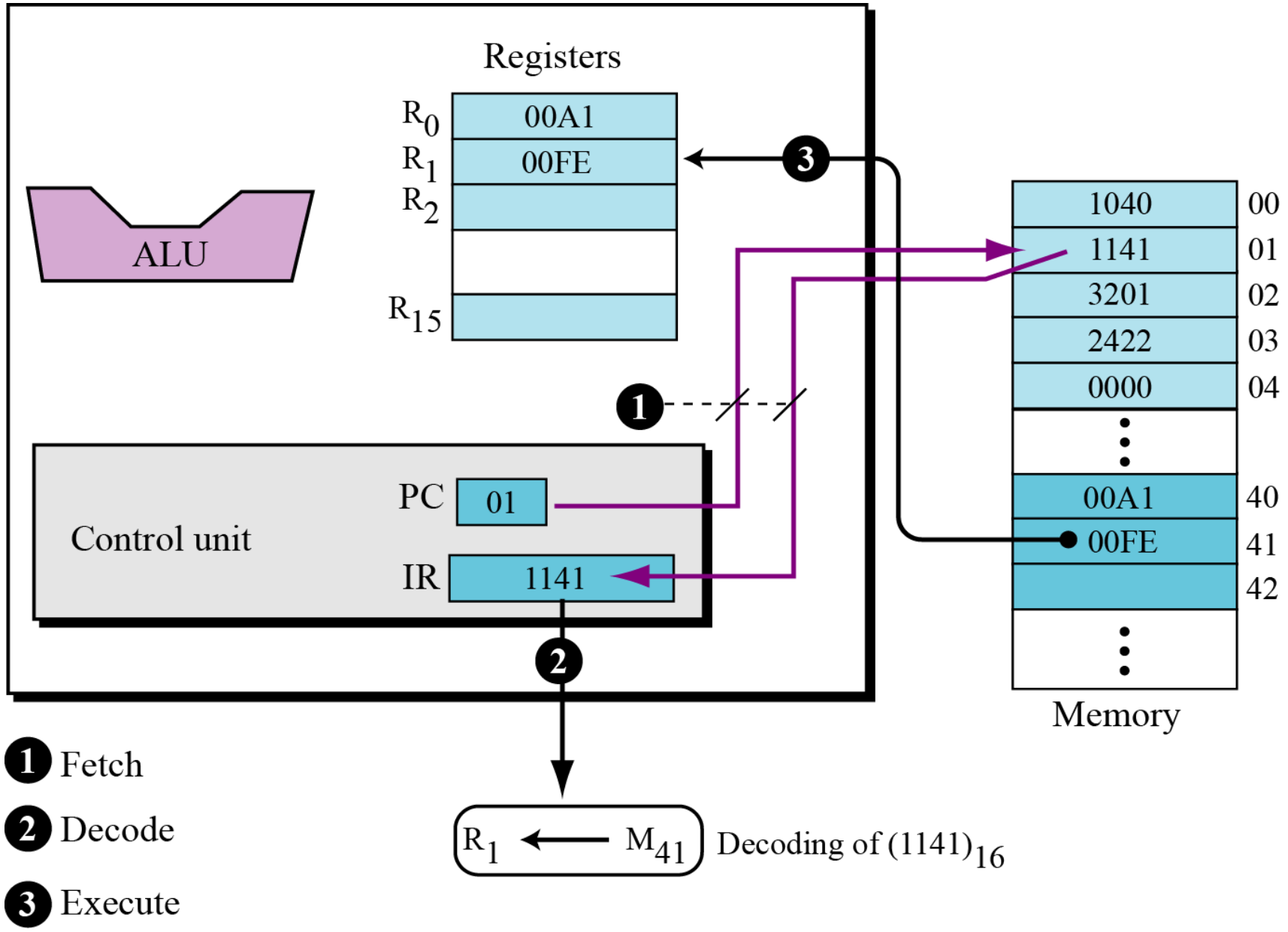
Podemos almacenar el programa de cinco líneas en la memoria a partir de la localización $(00)_{16}$ a $(04)_{16}$. Ya sabemos que los datos necesitan ser almacenados en ubicaciones de memoria $(40)_{16}$, $(41)_{16}$, y $(42)_{16}$.

Ciclos

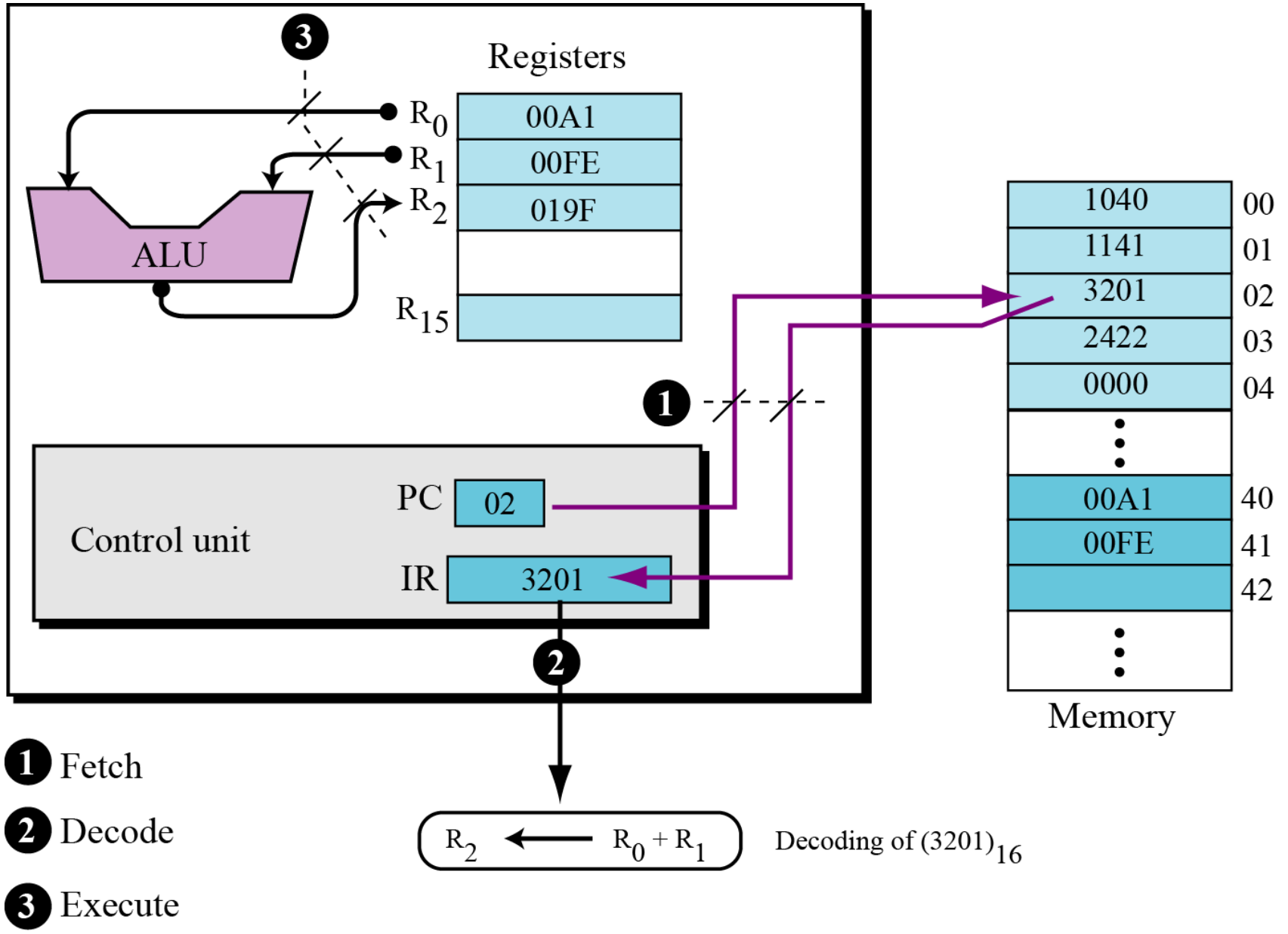
Nuestro computador utiliza un ciclo por instrucción. Si tenemos un pequeño programa con cinco instrucciones, tenemos cinco ciclos. También sabemos que cada ciclo se hace normalmente de tres pasos: buscar, decodificar, ejecutar. De momento, supongamos que tenemos que añadir 161 + 254 = 415. Los números se muestran en la memoria en hexadecimal es, $(00A1)_{16}$, $(00FE)_{16}$, y $(019F)_{16}$.



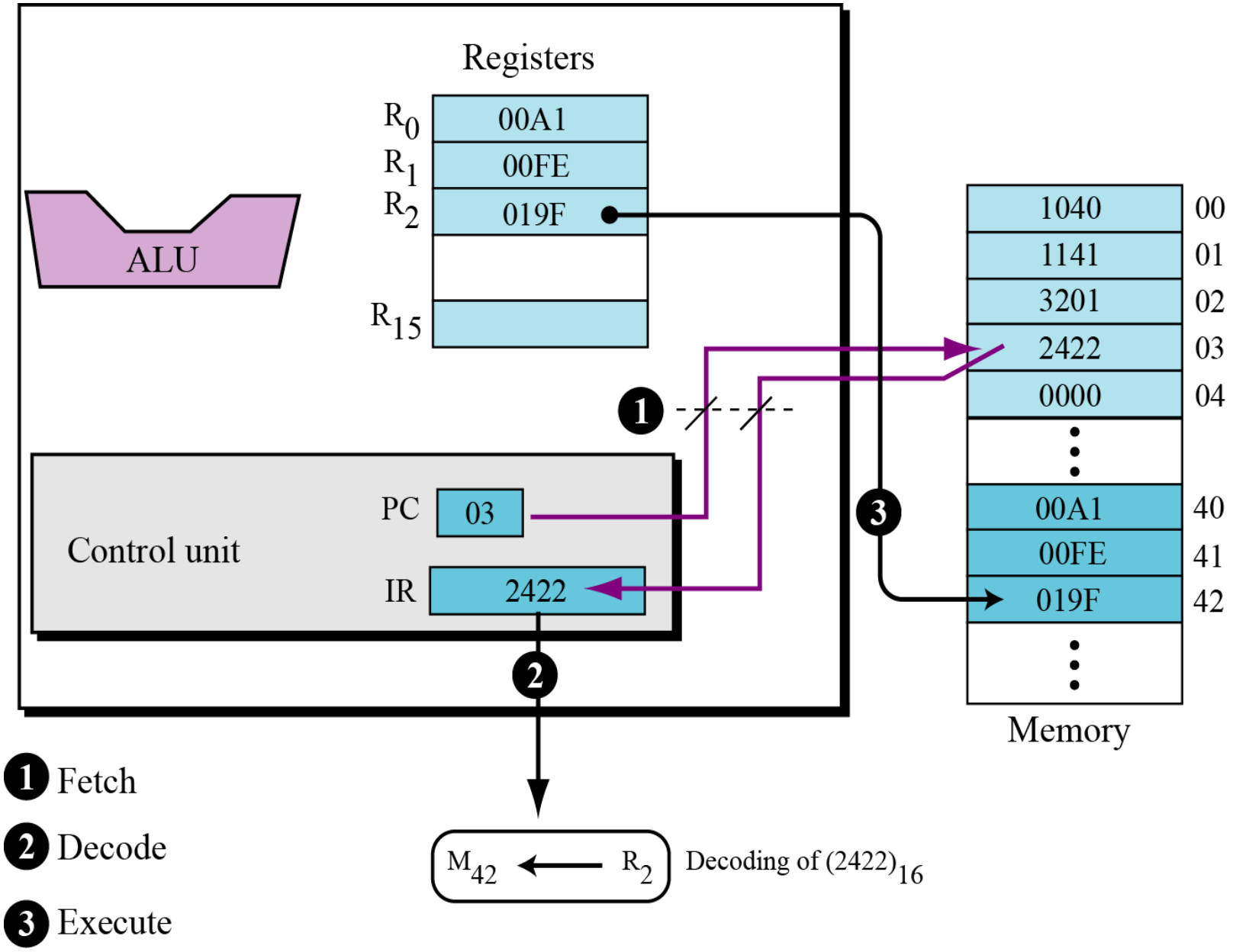
Situación de ciclo 1



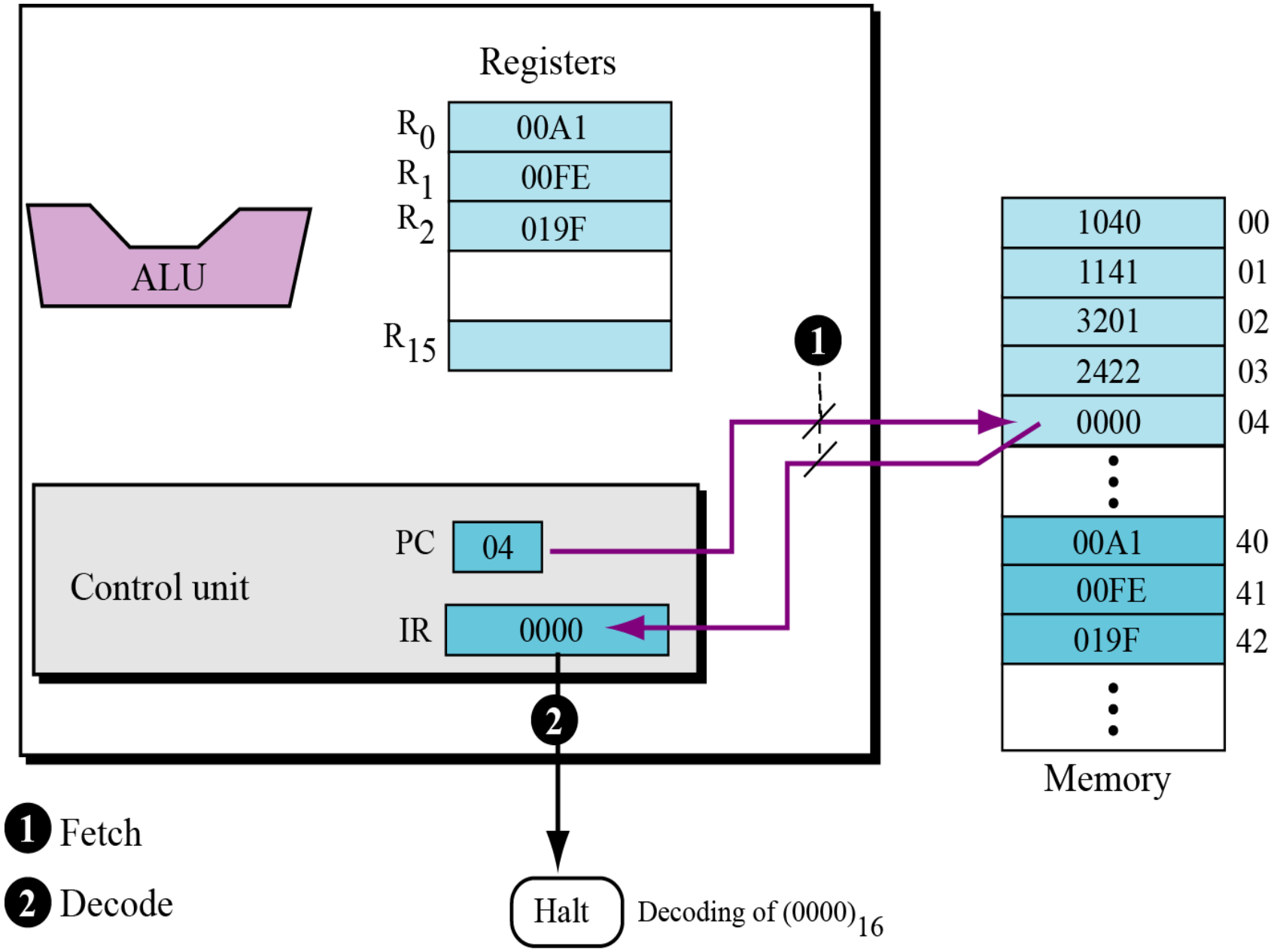
Situación de ciclo 2



Situación de ciclo 3



Situación de ciclo 4



Situación de ciclo 5

Otro ejemplo

En ejemplo anterior supusimos que los dos enteros que se suman ya estaban en la memoria. También asumimos que el resultado de la adición se llevará a cabo en la memoria. Podemos preguntar cómo almacenar los dos enteros que queremos sumar en la memoria, o cómo usar el resultado cuando se almacena en la memoria. En una situación real, proporcionamos los dos primeros enteros a la memoria mediante un dispositivo de entrada como el teclado, y presentamos el tercer entero a través de un dispositivo de salida como un monitor. Obtención de datos a través de un dispositivo de entrada normalmente se llama una operación de lectura, mientras que el envío de datos a un dispositivo de salida normalmente se llama una operación de escritura. Para hacer nuestro programa anterior más práctico, es necesario modificarlo de la siguiente manera:

1. Read an integer into M_{40}.
2. $R_0 \leftarrow M_{40}$.
3. Read an integer into M_{41}.
4. $R_1 \leftarrow M_{41}$.
5. $R_2 \leftarrow R_0 + R_1$.
6. $M_{42} \leftarrow R2$.
7. Write the integer from M_{42}.
8. Halt.

En nuestro ordenador podemos simular las operaciones de lectura y escritura utilizando las instrucciones cargar (LOAD) y almacenar (STORE). Además, LOAD y STORE leen datos de entrada a la CPU y escriben datos del CPU. Necesitamos dos instrucciones para leer los datos en la memoria o escribir datos fuera de la memoria. La operación de lectura es:

$R \leftarrow M_{EF}$	Because the keyboard is assumed to be memory location $(EF)_{16}$
$M \leftarrow R$	

La operación de escritura es la siguiente:

$R \leftarrow M$	
$M_{FF} \leftarrow R$	Because the monitor is assumed to be memory location $(EF)_{16}$

La operación de entrada siempre debe leer datos de un dispositivo de entrada en la memoria: la operación de salida siempre debe escribir los datos de la memoria a un dispositivo de salida.

El programa es codificado como:

1	$(1FFE)_{16}$	5	$(1040)_{16}$	9	$(1F42)_{16}$
2	$(240F)_{16}$	6	$(1141)_{16}$	10	$(2FFF)_{16}$
3	$(1FFE)_{16}$	7	$(3201)_{16}$	11	$(0000)_{16}$
4	$(241F)_{16}$	8	$(2422)_{16}$		

Operaciones de 1 a 4 son para la entrada y operaciones 9 y 10 son para la salida. Cuando ejecutamos este programa, espera a que el usuario ingrese dos enteros en el teclado y presione la tecla enter. El programa entonces calcula la suma y muestra el resultado en el monitor.