

# Organización del computador

Arquitectura de von Newmann

# En el comienzo de todo, Hilbert creó un programa

- > Ante la crisis fundacional que vivía la matemática a comienzos del siglo 20, en el International Congress of Mathematicians de 1928, **David Hilbert** formula un programa:



# En el comienzo de todo, Hilbert creó un programa

- ➤ Ante la crisis fundacional que vivía la matemática a comienzos del siglo 20, en el International Congress of Mathematicians de 1928, **David Hilbert** formula un programa:
  - ➤ **Formalizar la matemática:** toda sentencia debe ser escrita en un lenguaje formal y manipulada por reglas bien definidas



# En el comienzo de todo, Hilbert creó un programa

- > Ante la crisis fundacional que vivía la matemática a comienzos del siglo 20, en el International Congress of Mathematicians de 1928, **David Hilbert** formula un programa:
  - >**Formalizar la matemática:** toda sentencia debe ser escrita en un lenguaje formal y manipulada por reglas bien definidas
  - >**Compleitud:** El formalismo permite probar todas las cosas verdaderas



# En el comienzo de todo, Hilbert creó un programa

- ➤ Ante la crisis fundacional que vivía la matemática a comienzos del siglo 20, en el International Congress of Mathematicians de 1928, **David Hilbert** formula un programa:
  - ➤ **Formalizar la matemática:** toda sentencia debe ser escrita en un lenguaje formal y manipulada por reglas bien definidas
  - ➤ **Compleitud:** El formalismo permite probar todas las cosas verdaderas
  - ➤ **Consistencia:** El formalismo no permite probar ninguna contradicción...



# En el comienzo de todo, Hilbert creó un programa

- > Ante la crisis fundacional que vivía la matemática a comienzos del siglo 20, en el International Congress of Mathematicians de 1928, **David Hilbert** formula un programa:
  - >**Formalizar la matemática:** toda sentencia debe ser escrita en un lenguaje formal y manipulada por reglas bien definidas
  - >**Compleitud:** El formalismo permite probar todas las cosas verdaderas
  - >**Consistencia:** El formalismo no permite probar ninguna contradicción...
  - >**Conservación:** Cualquier resultado sobre “objetos reales” obtenido a partir de razonar sobre “objetos ideales” (como conjuntos no numerables) puede ser demostrado sin usar “objetos ideales”
  - >**Decidibilidad:** Debe existir un algoritmo para determinar la verdad o falsedad de toda proposición del lenguaje





## → Teoremas de incompletitud de Gödel

- Son dos teoremas de la lógica que prueban las limitaciones inherentes de cualquier sistema axiomático formal que contenga la aritmética básica. Este resultado fue publicado en 1931 [1], y son sumamente importantes en la lógica matemática y la filosofía de la matemática.
- Estos dos teoremas constituyen la demostración de la imposibilidad de la existencia de un sistema axiomático completo y consistente para la aritmética y, consecuentemente, para la matemática.



---

[1] Kurt Gödel, 1931, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I", Monatshefte für Mathematik und Physik, v. 38 n. 1, pp. 173–198.

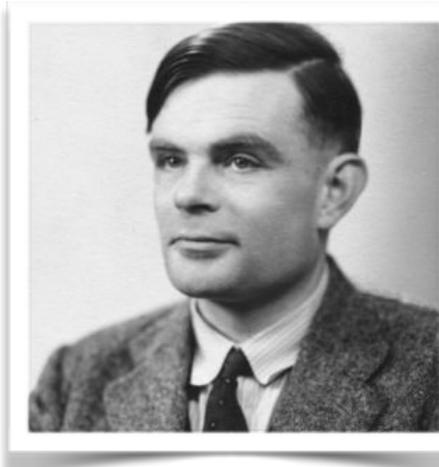


## → Hilbert's Entscheidungsproblem (decision problem) [2]

→ En 1936 Alonzo Church [3] prueba que no existe una solución general para el problema de la decisión si la noción de “cálculo efectivo” es la capturada por expresibilidad en lambda cálculo.



→ Independientemente, también en 1936, Alan Turing [4] prueba el mismo resultado pero tomando “cálculo efectivo” como función computable por una a-machine (por automatic machine), posteriormente llamada máquina de Turing.



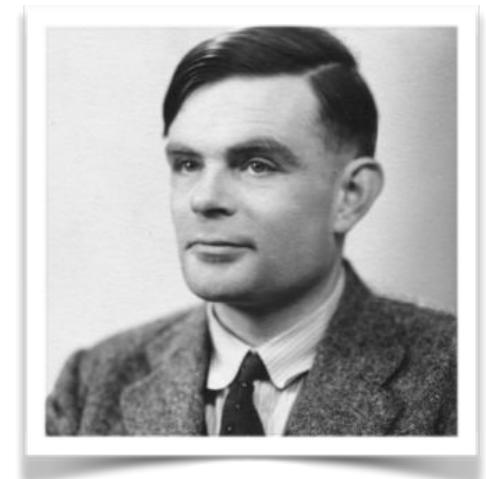
[2] David Hilbert and Wilhelm Ackermann (1928). *Grundzüge der theoretischen Logik* (*Principles of Mathematical Logic*). Springer-Verlag

[3] Alonzo Church, "An unsolvable problem of elementary number theory", *American Journal of Mathematics*, 58 (1936), pp 345–363

[4] Alan Turing, "On computable numbers, with an application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society*, Series 2, 42 (1936-7), pp 230–265. Errata appeared in Series 2, 43 (1937), pp 544–546.

# Entonces Turing dijo: “¡Que haya cálculo efectivo!”

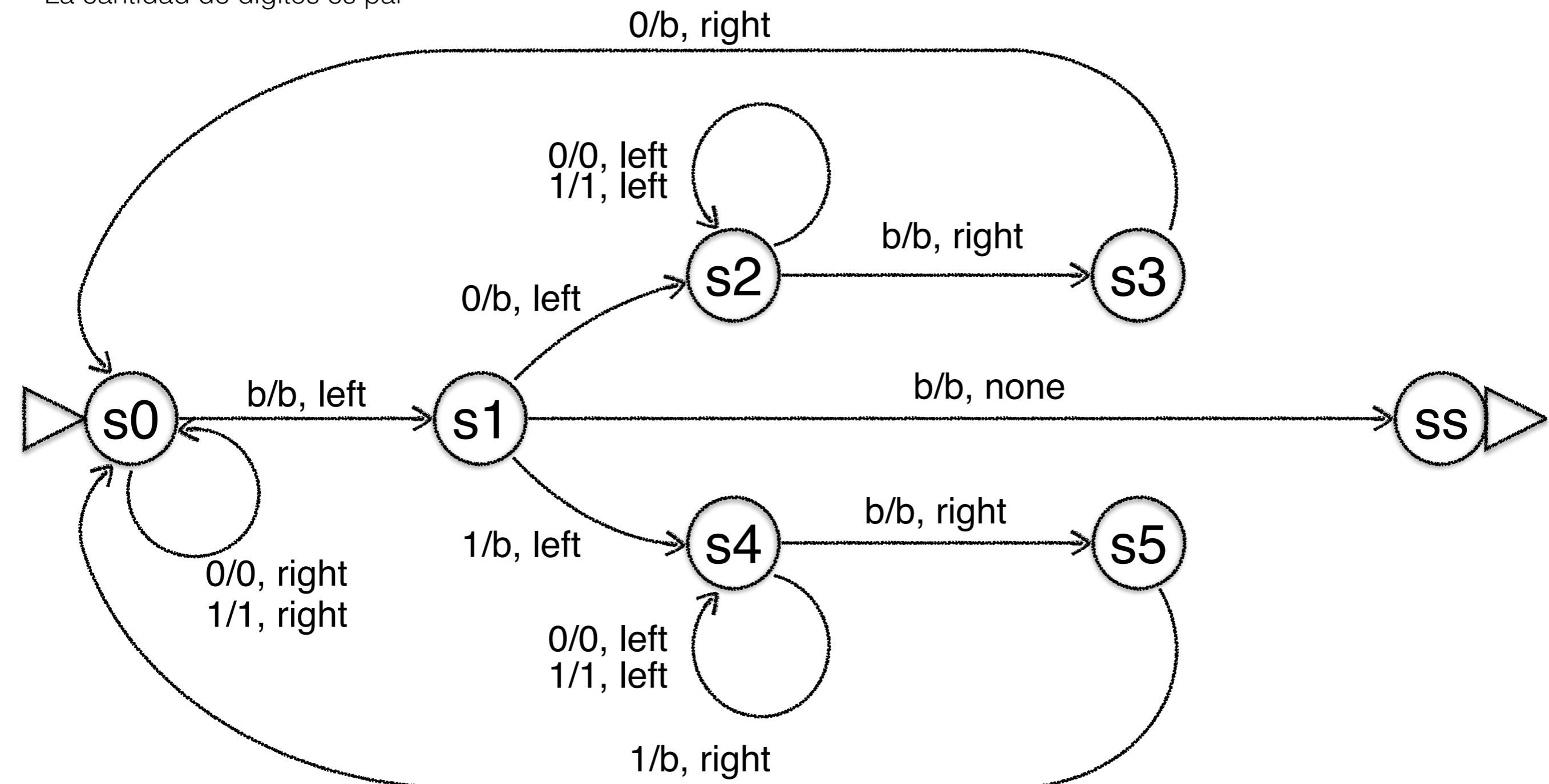
- ➤ Informalmente, una **máquina de Turing** modela matemáticamente un mecanismo de manipulación de símbolos. Esta cuenta con:
- ➤ **Cinta**: secuencia infinita de celdas capaces de contener un símbolo que puede ser b (blank) o un símbolo de D (alphabet)
- ➤ **Cabeza**: puede leer el contenido de una celda o escribir uno nuevo, y puede moverse hacia la derecha / izquierda a la celda contigua en esa dirección
- ➤ **Registro de Estado**: guarda el estado actual de la máquina tomado de un conjunto finito de estados posibles
- ➤ **Tabla de instrucciones**: dado **a**) el estado actual de la máquina y **b**) el contenido de la celda que se encuentra debajo de la cabeza, la máquina: **1)** escribe un símbolo en la celda, **2)** mueve la cabeza hacia la izquierda o derecha y **3)** cambia el estado en el registro de estado.



# Ejemplo: Palíndromo

## Simplificaciones:

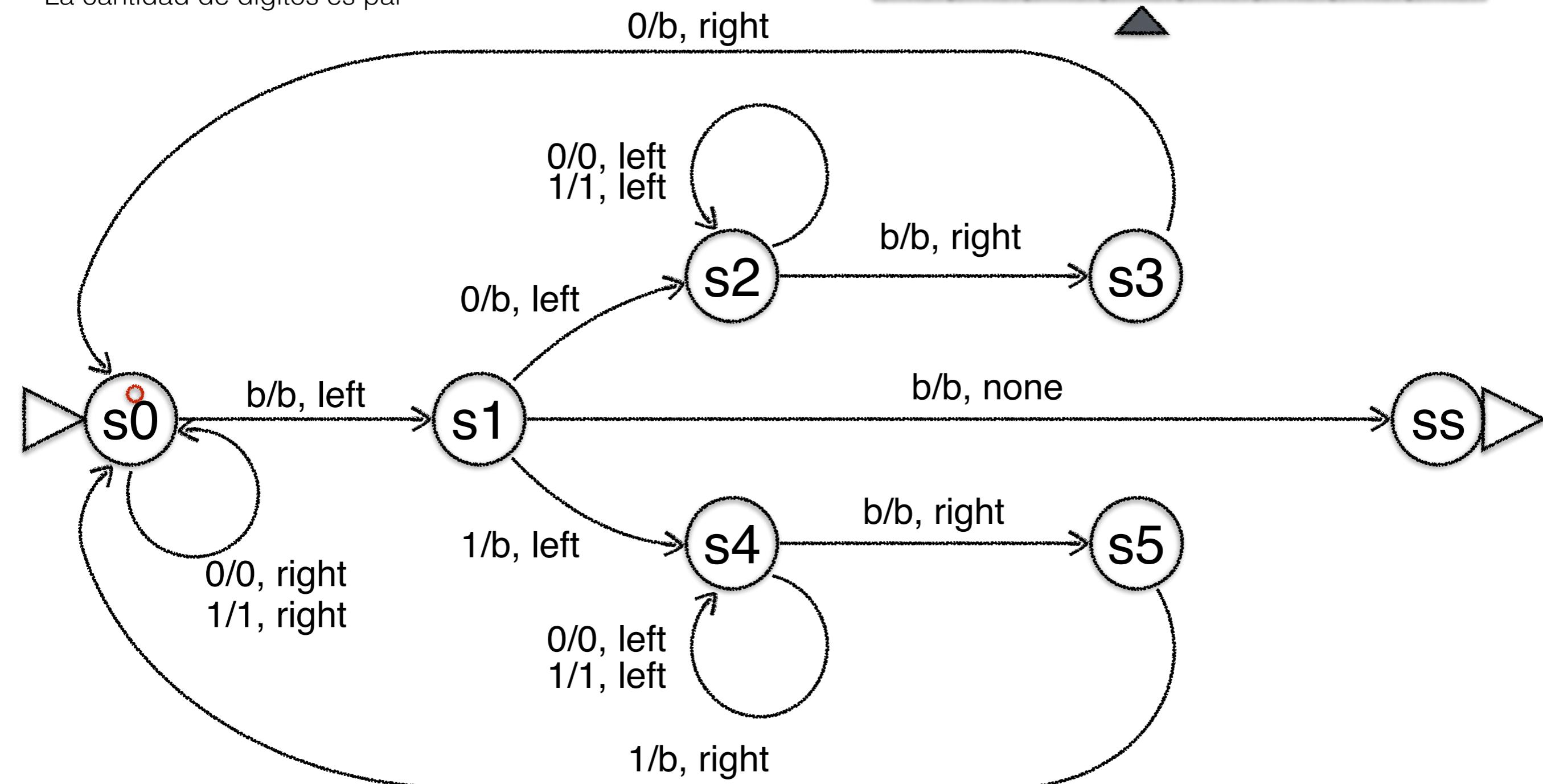
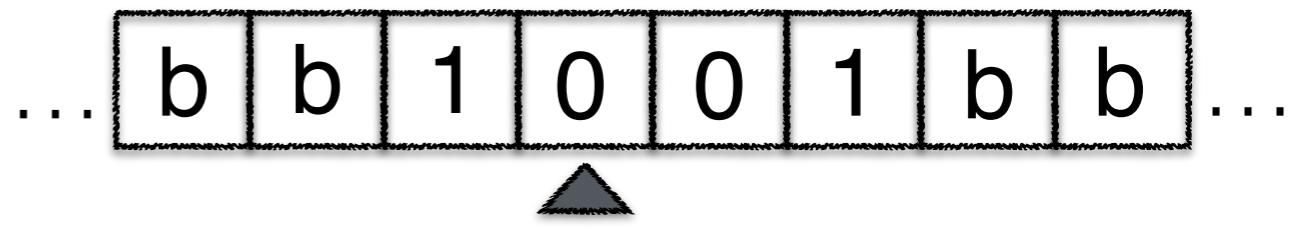
- La cabeza se encuentra en uno de los dígitos de la entrada
- La cantidad de dígitos es par



# Ejemplo: Palíndromo

## Simplificaciones:

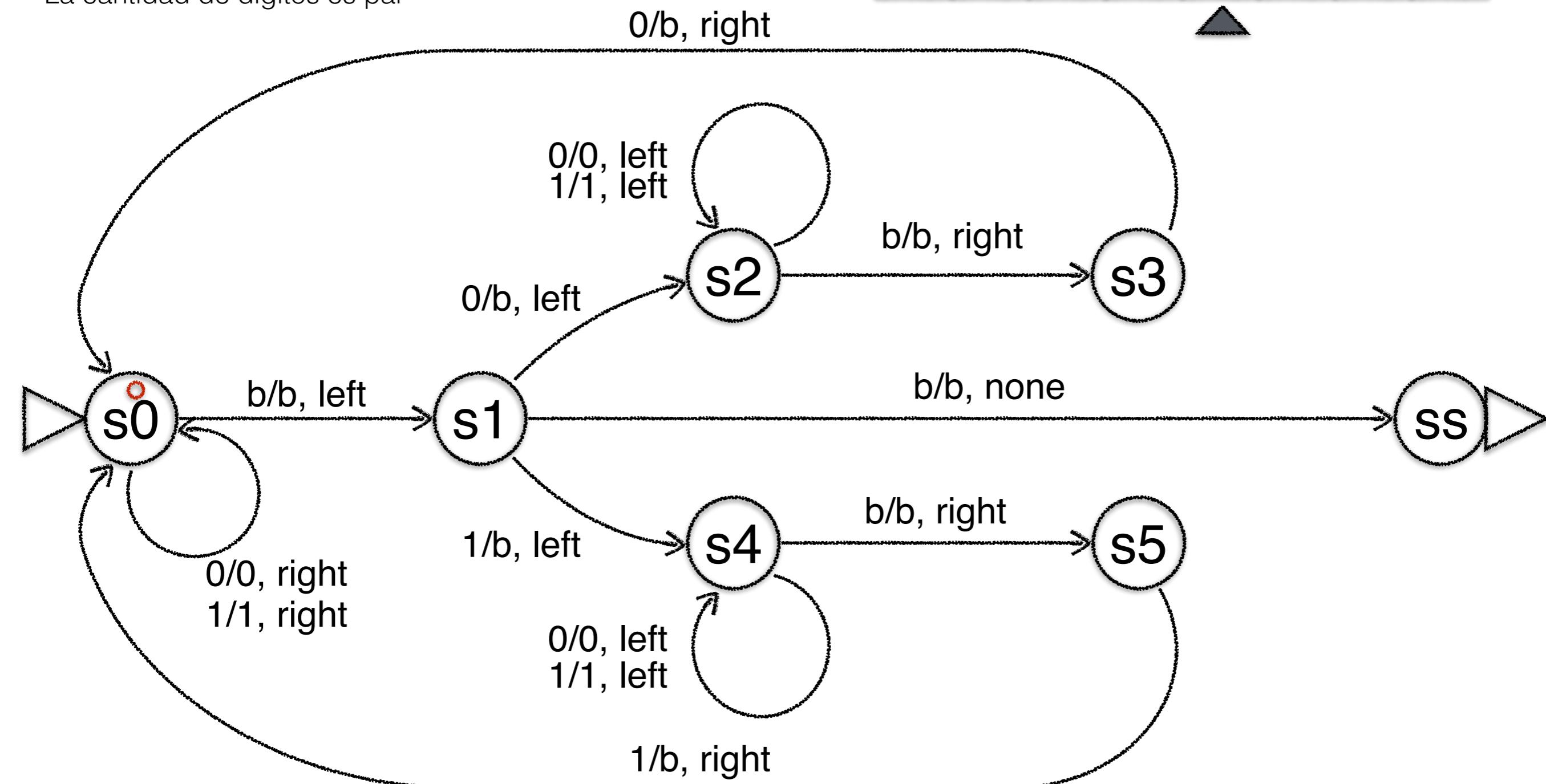
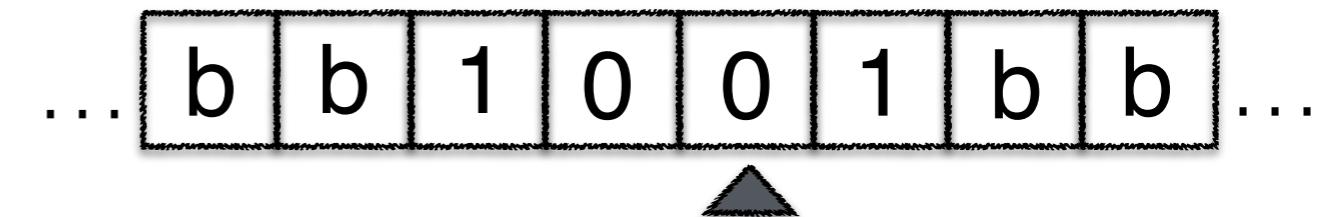
- La cabeza se encuentra en uno de los dígitos de la entrada
- La cantidad de dígitos es par



# Ejemplo: Palíndromo

## Simplificaciones:

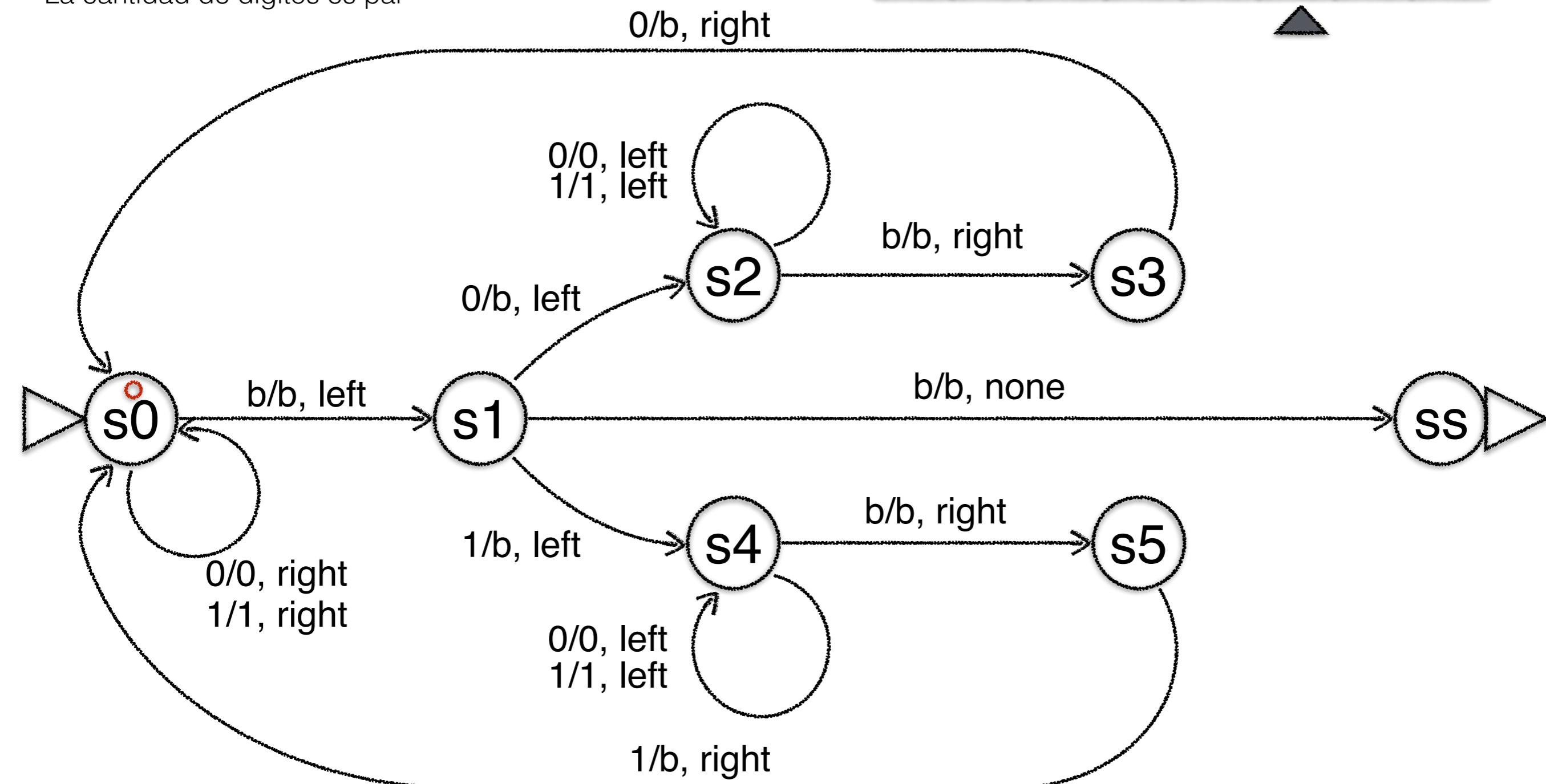
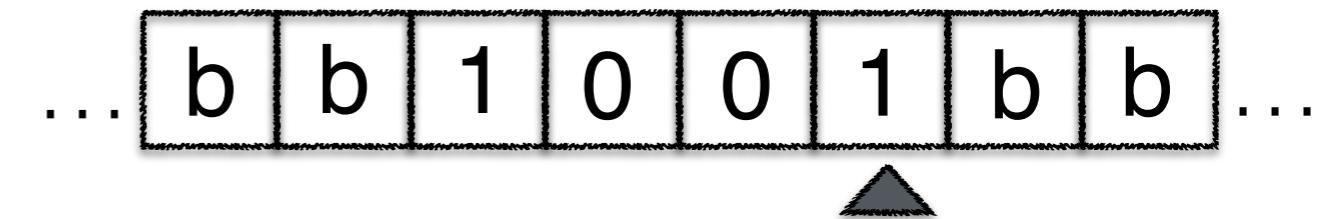
- La cabeza se encuentra en uno de los dígitos de la entrada
- La cantidad de dígitos es par



# Ejemplo: Palíndromo

## Simplificaciones:

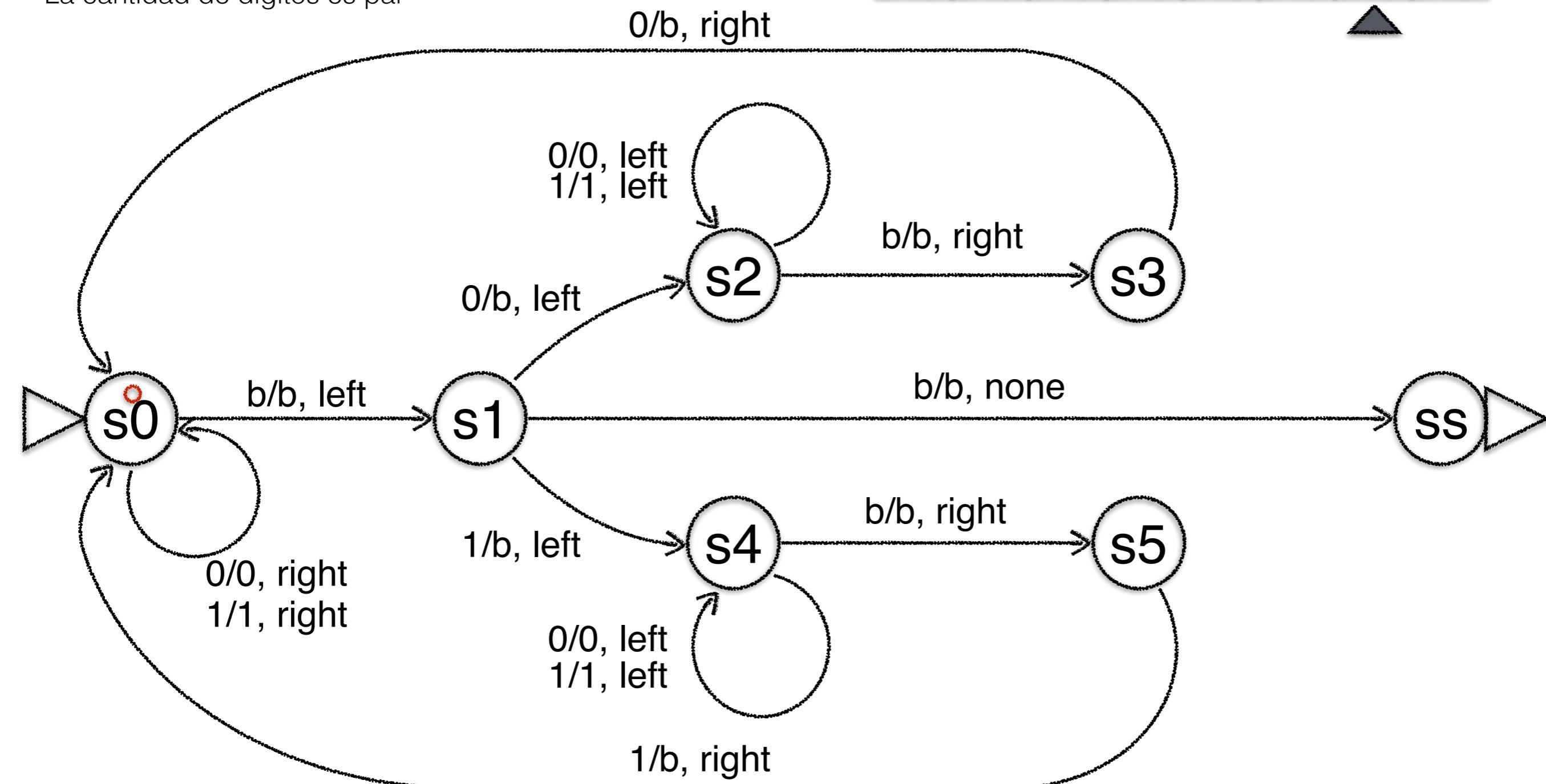
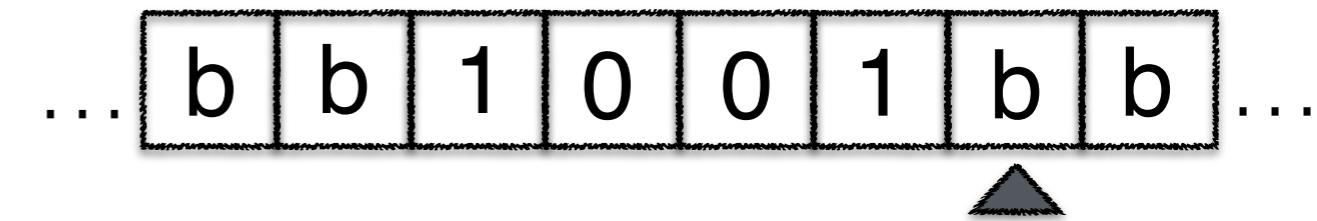
- La cabeza se encuentra en uno de los dígitos de la entrada
- La cantidad de dígitos es par



# Ejemplo: Palíndromo

## Simplificaciones:

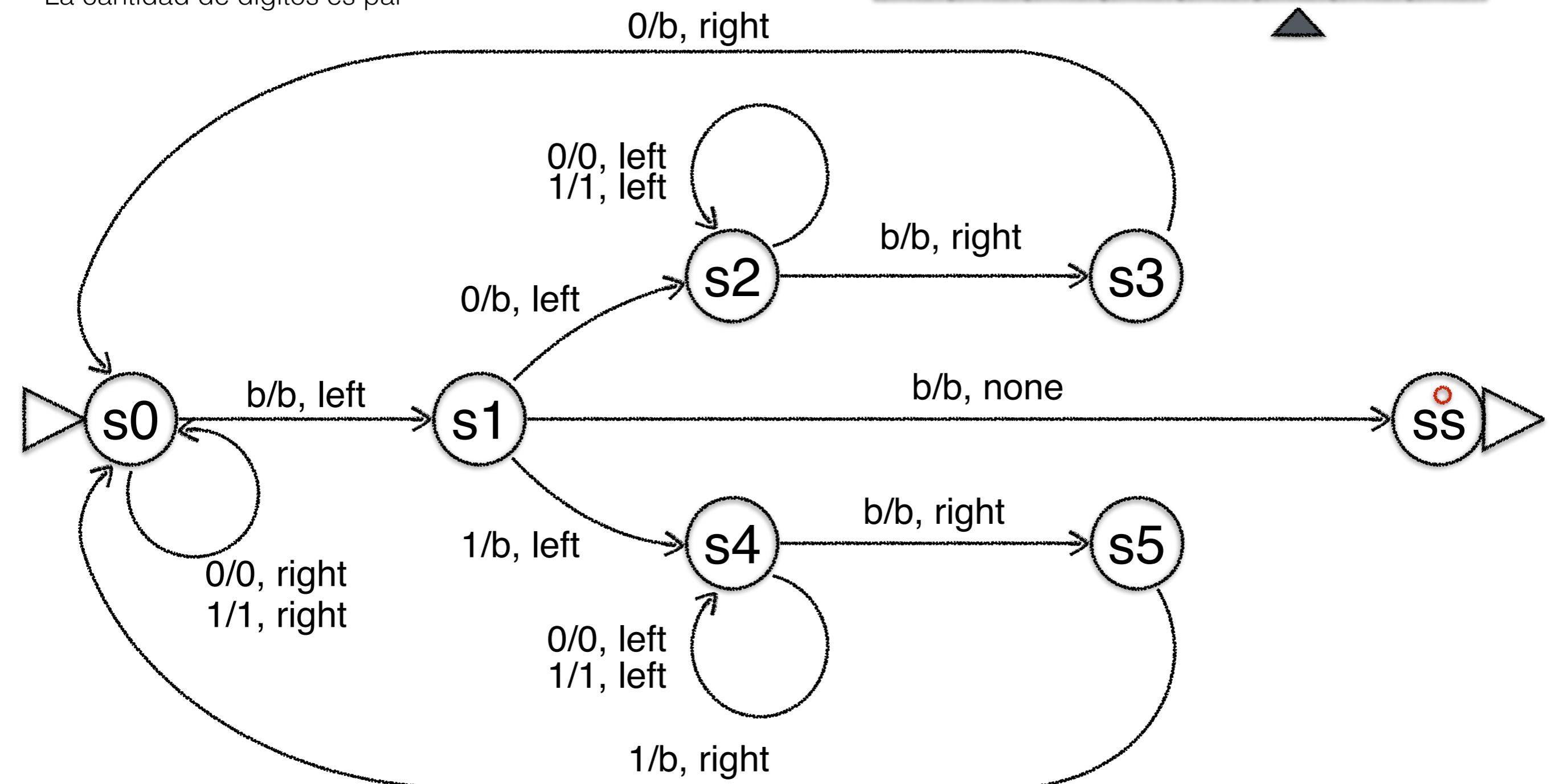
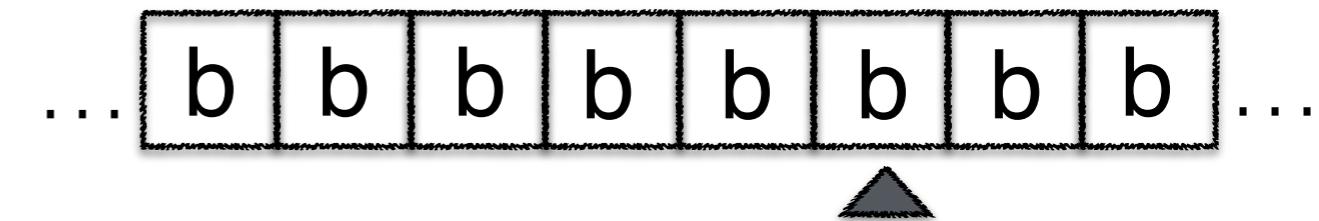
- La cabeza se encuentra en uno de los dígitos de la entrada
- La cantidad de dígitos es par



# Ejemplo: Palíndromo

## Simplificaciones:

- La cabeza se encuentra en uno de los dígitos de la entrada
- La cantidad de dígitos es par

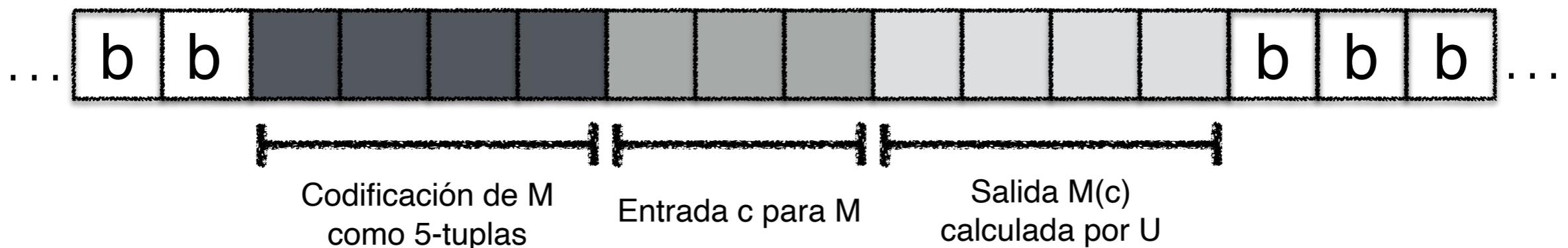


# Máquina universal de Turing

Ya en [4] Alan Turing escribió:

It is possible to invent a single machine which can be used to compute any computable sequence. If this machine **U** is supplied with the tape on the beginning of which is written the string of quintuples separated by semicolons of some computing machine **M**, then **U** will compute the same sequence as **M**.

Es posible inventar una máquina única que puede ser usada para computar cualquier secuencia computable. Si esta máquina **U** es abastecida con una cinta al comienzo de la cual están escritas las tiras de quíntuplas separadas por punto y coma de alguna máquina de cómputo **M**, entonces **U** va a computar la misma secuencia que **M**.



# El modelo de Von Newman

- \* Programar se trataba de interconectar con cables las componentes de la máquina...  
... y cada vez que se quería ejecutar un programa se debía reconectar según lo documentado
- \* Ejecutar un programa distinto implicaba modificar el cableado de la máquina
- \* Programar era muy parecido a la ingeniería electrónica
- \* Mauchly y Eckert (diseñadores de ENIAC) documentaron la idea de almacenar programas como base para la construcción de EDVAC (**nunca lo publicaron**)

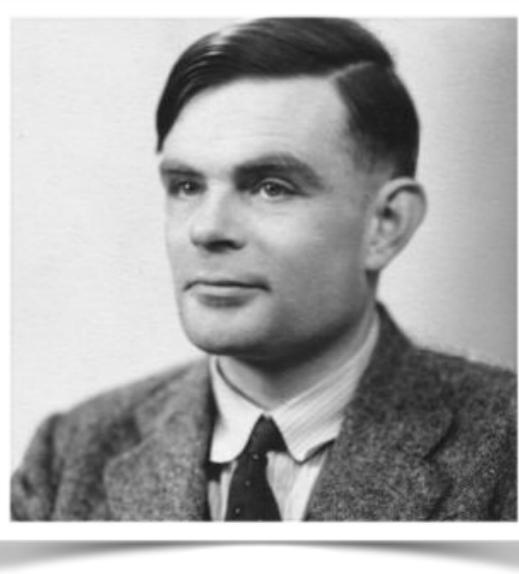
# El modelo de Von Newman

## John Von Neumann

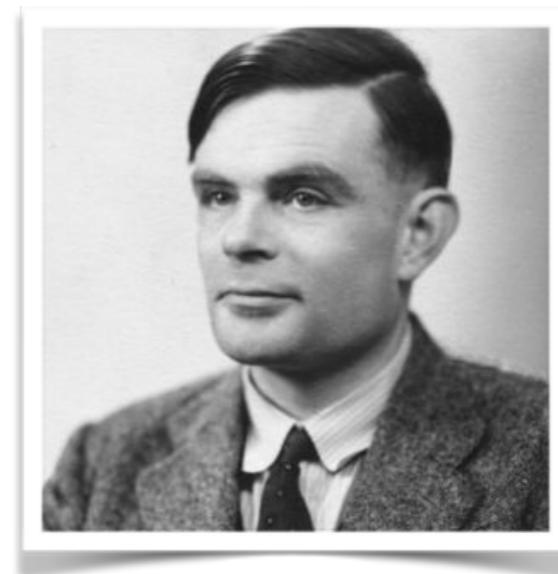
\*Publicó y popularizó la idea de que una computadora debiera tener una memoria en la cual almacenar los programas (The First Draft)



\*Desde un punto de vista teórico, esta idea responde a la máquina universal de Turing y data de 1936, publicada por **Alan Turing** (On Computable Numbers, with an Application to the Entscheidungsproblem)



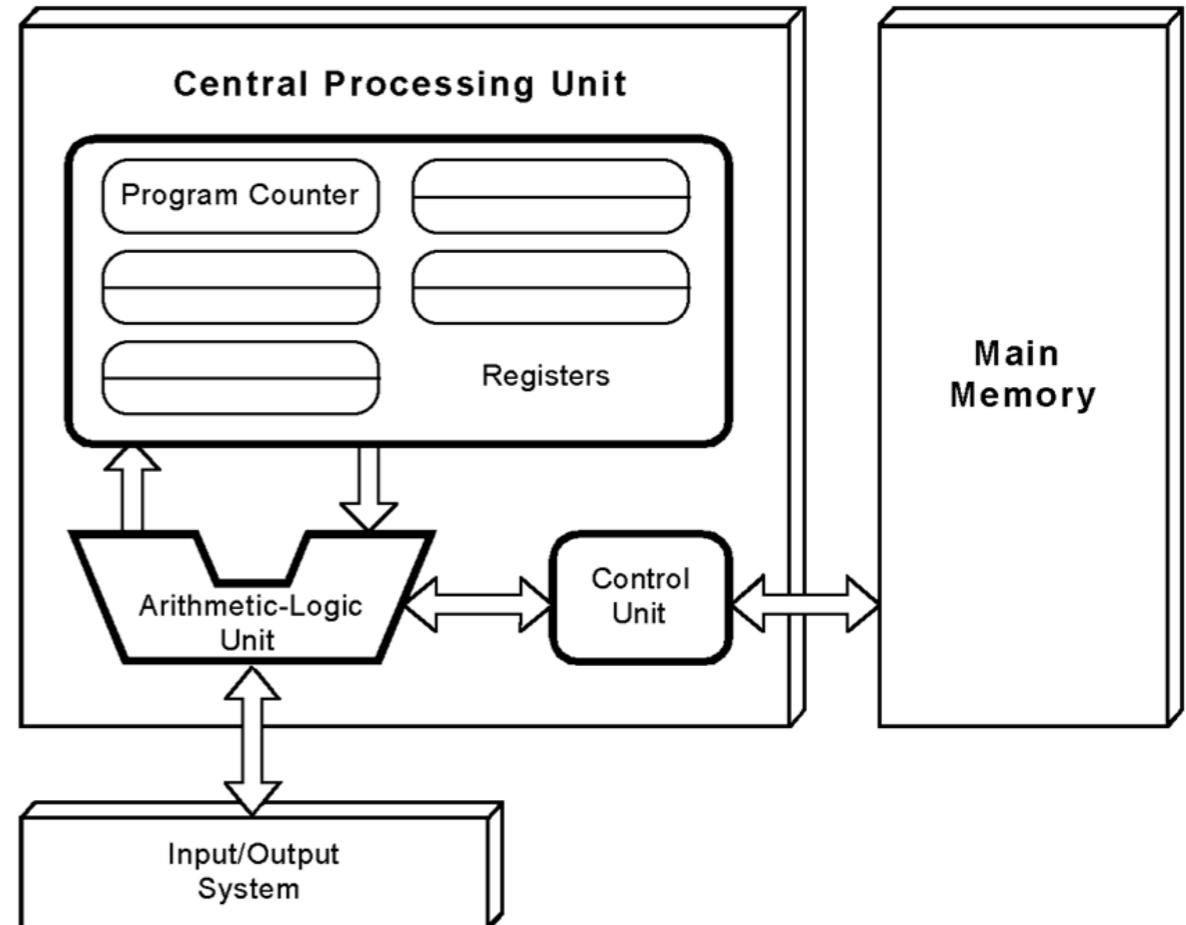
# Von Newman / Turing



- \* Los programas y los datos se almacenan en la misma memoria sobre la que se puede leer y escribir
- \* La operación de la máquina depende del estado de la memoria
- \* El contenido de la memoria es accedido a partir de su posición
- \* La ejecución es secuencial (a menos que se indique lo contrario)

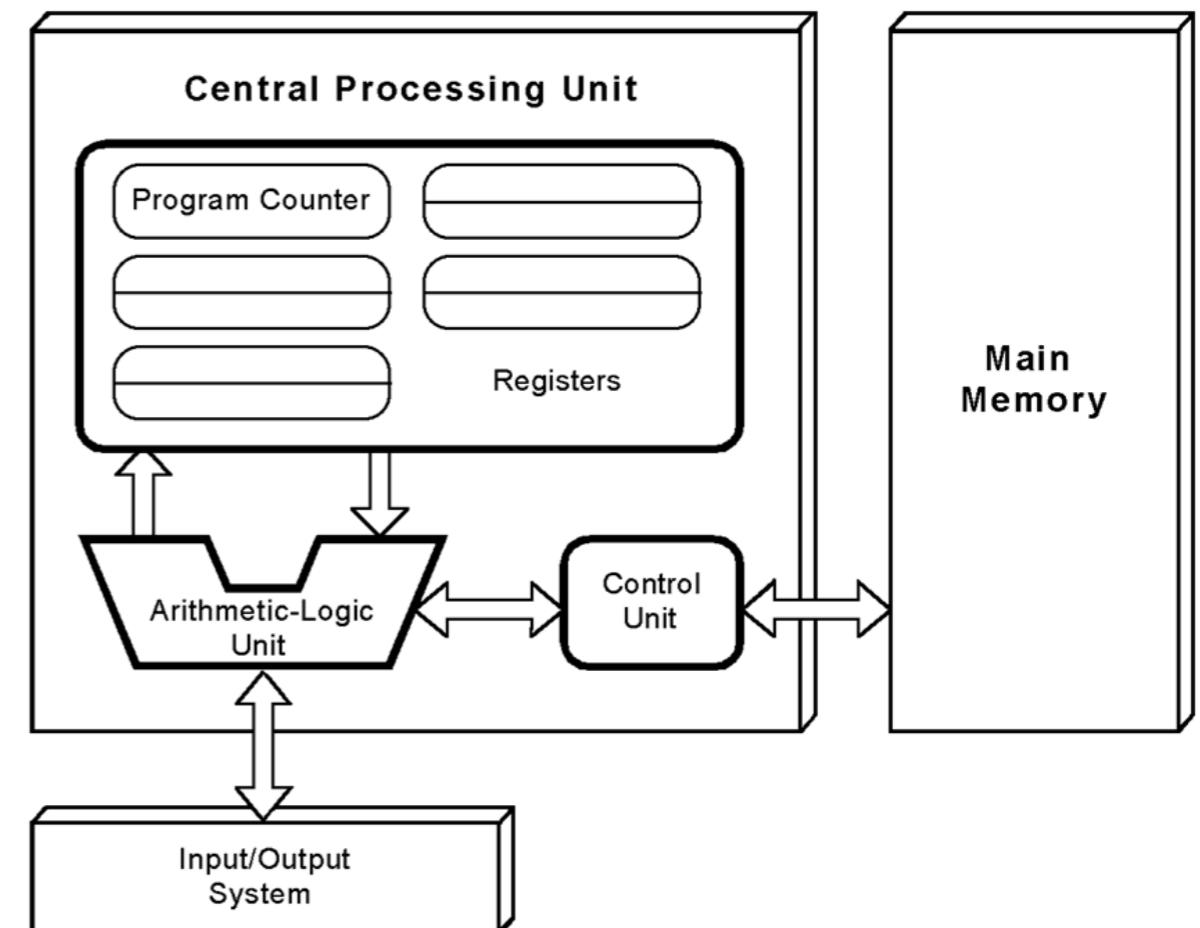
# Arquitectura de von Newmann

- 3 componentes principales:
  - **CPU:** Unidad de control, Unidad Aritmética lógica, Registros
  - **Memoria:** Almacenamiento de programas y datos
  - **Sistema de Entrada y Salida**
- Procesamiento secuencial de instrucciones
- Datos almacenados en sistema binario
- Sistema de interconexión de componentes:
  - Conecta la Unidad de Control con la Memoria mediante un camino único
  - La unicidad del camino fuerza la alternación entre ciclos de lectura / escritura y ejecución
  - Esta alternación se llama cuello de botella de von Newmann (von Newmann bottleneck)<sup>[\*]</sup>



[\*] El término “cuello de botella de von Neumann” fue acuñado por John Backus en su conferencia de la concesión del Premio Turing ACM de 1977.

# Arquitectura de von Newmann



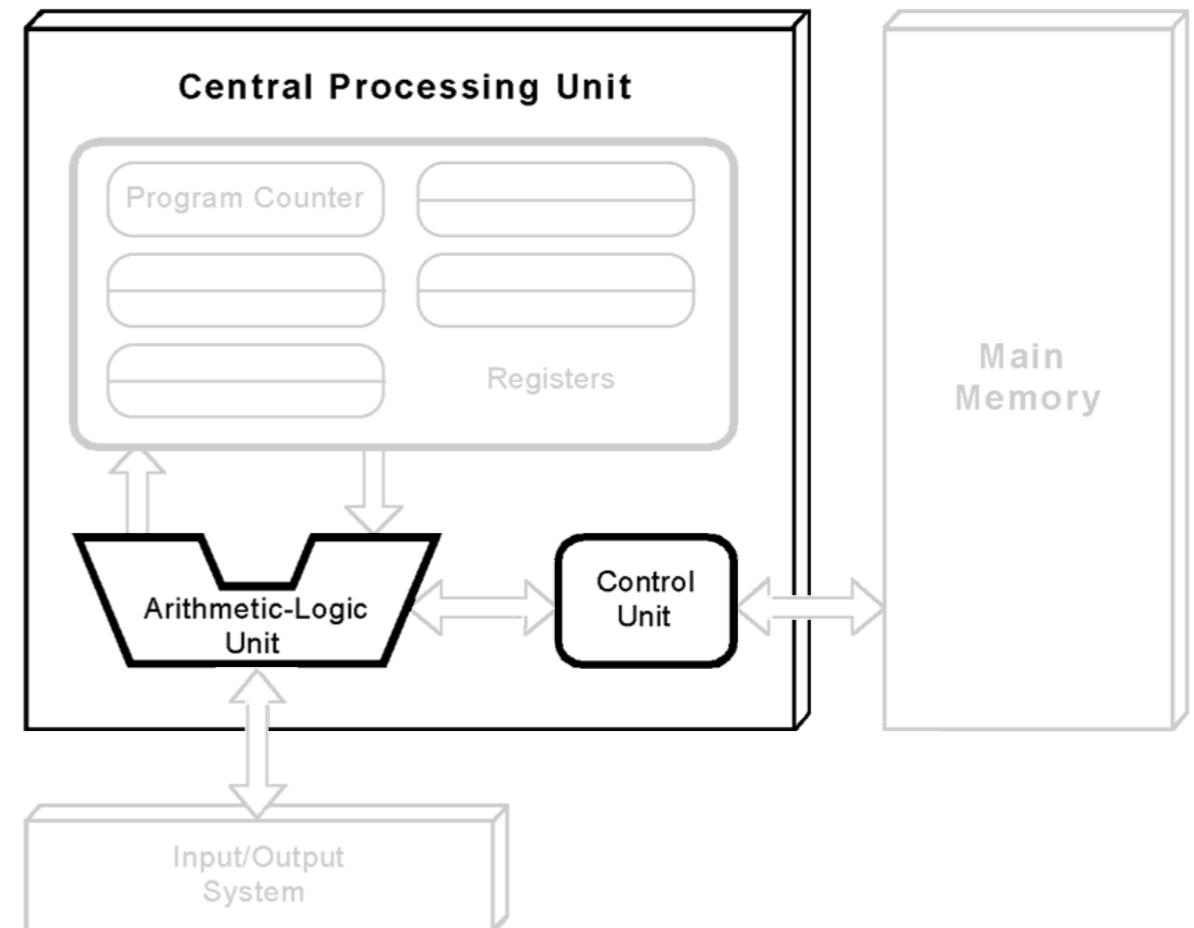
# Arquitectura de von Newmann

## → Unidad de Control (UC):

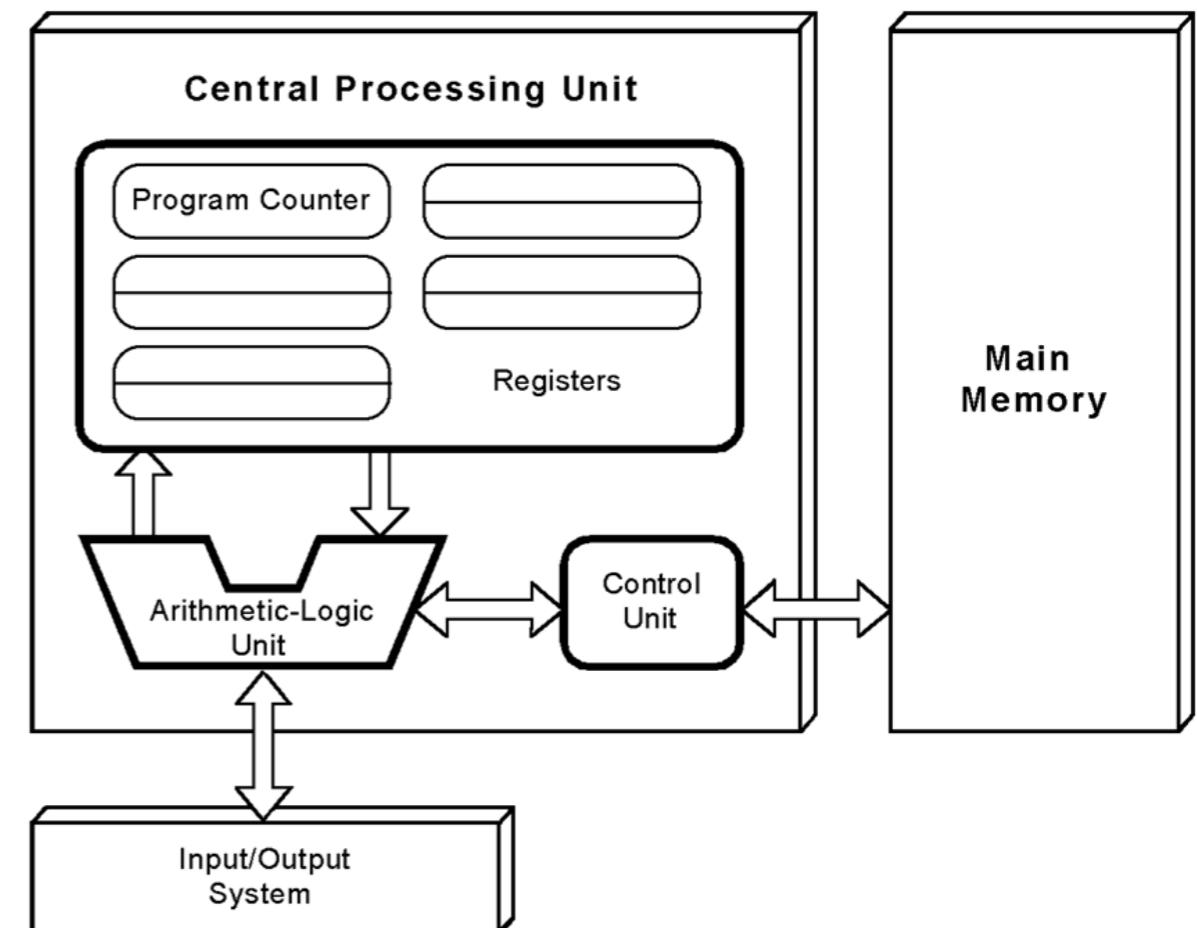
- Coordina la labor de los componentes de forma que las instrucciones y los datos puedan ser transferidos entre la memoria y los registros
- Decodifica instrucciones y las transforma en órdenes a los componentes
- Puede estar implementada en hardware o microprogramada

## → Unidad Aritmética Lógica (ALU):

- Realiza las operaciones matemáticas y lógicas sobre los datos
- Calcula los desplazamientos en memoria en el caso de direccionamientos complejos



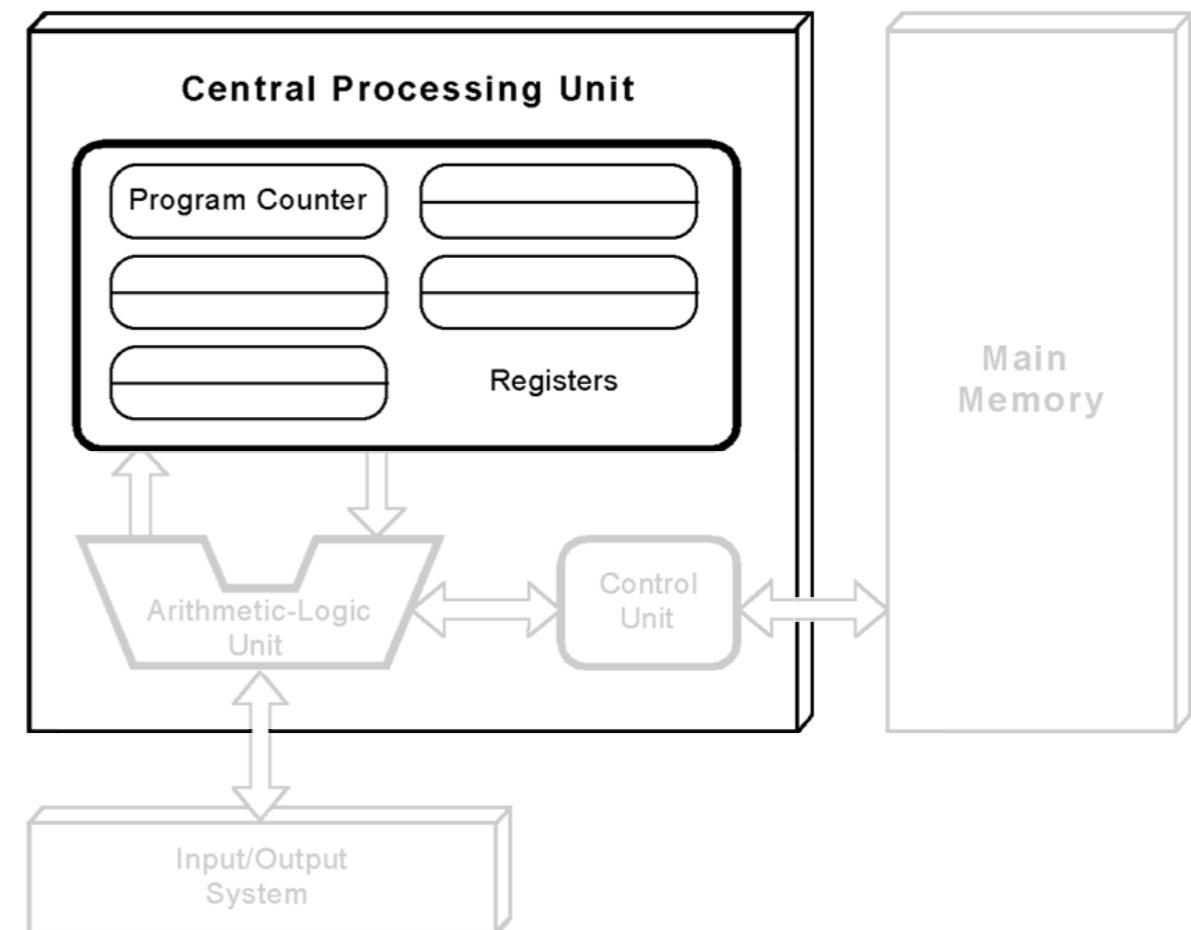
# Arquitectura de von Newmann



# Arquitectura de von Newmann

## →Registros:

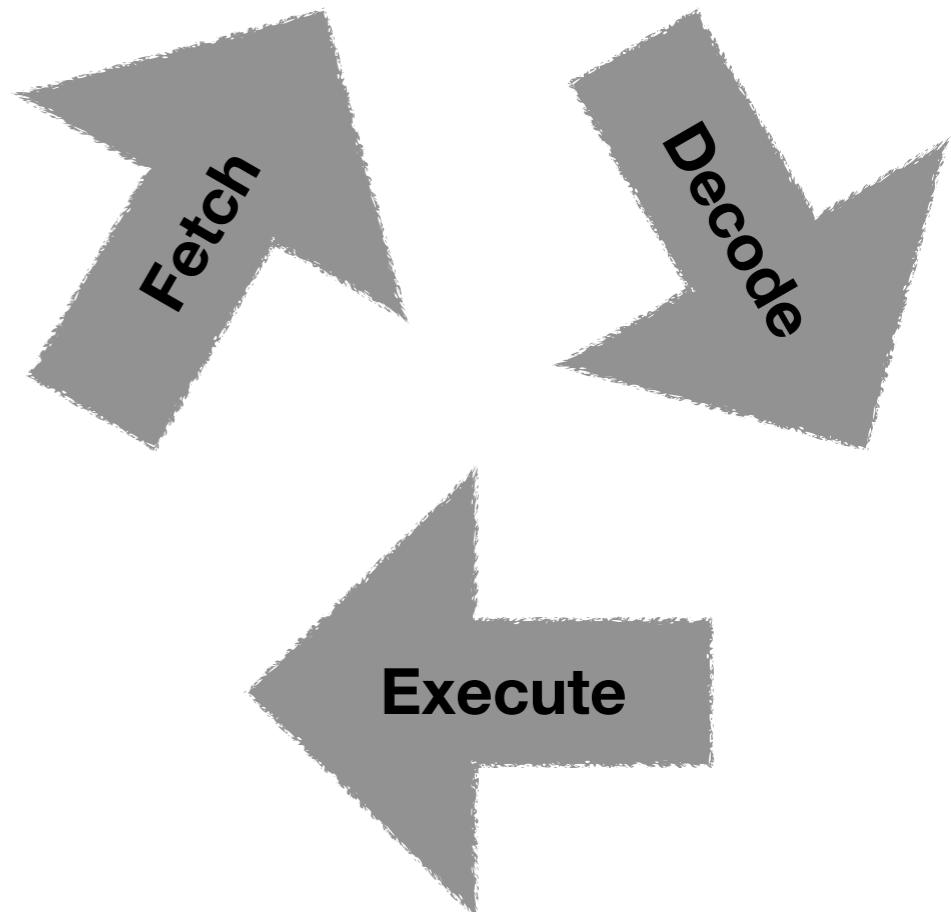
- Pequeñas unidades de memoria de acceso muy rápido
- Tamaño fijo dependiente de lo que almacenan: program counter, puntero a memoria, datos, instrucción
- Propósito específico: program counter, puntero a memoria, instrucción
- Propósito general: datos (tienen el tamaño de una palabra)



# Arquitectura de von Newmann

## → Ciclo de instrucción (UC):

- |         |  |
|---------|--|
| Fetch   | - Se obtiene la instrucción apuntada por el <b>PC</b> de la <b>Memoria</b>   |
| Decode  | - La <b>ALU</b> incrementa el <b>PC</b>  |
| Execute | - Se decodifica la instrucción<br>- Se obtienen los operandos de la <b>Memoria</b> y se los coloca en <b>Registros</b> |
|         | - La <b>ALU</b> realiza la operación<br>- Se coloca el resultado en la <b>Memoria</b>                                  |



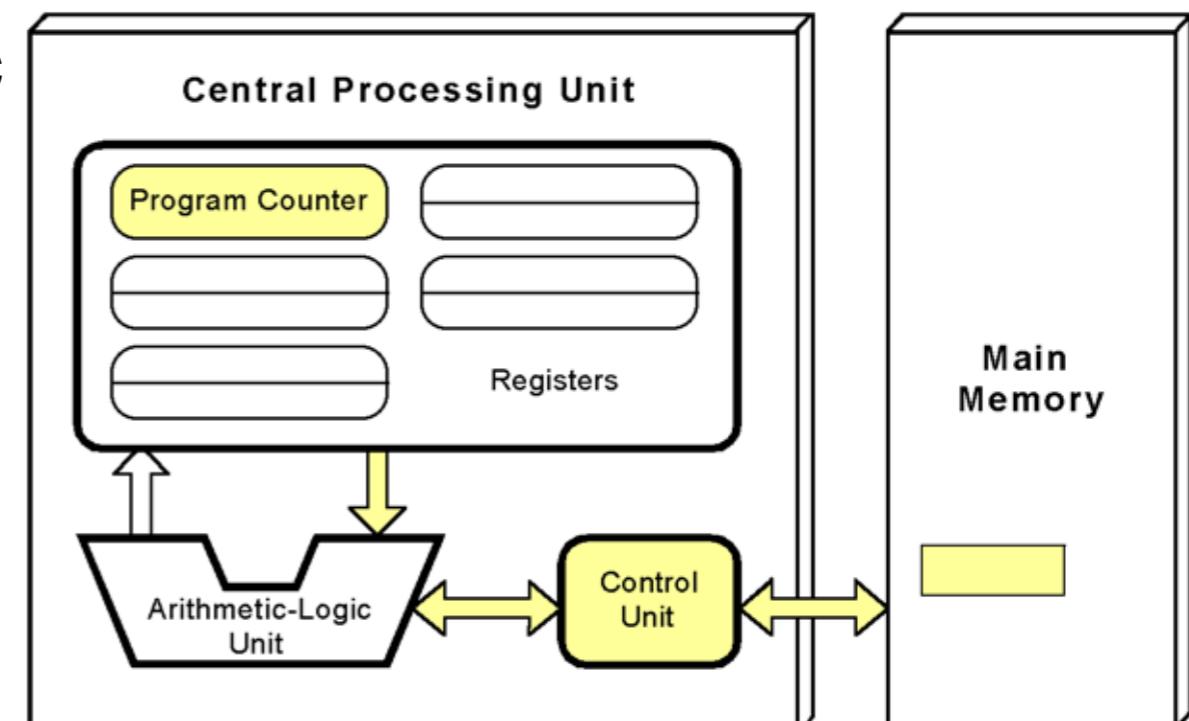
# Arquitectura de von Newmann

- > **Ciclo de instrucción (UC):**
  - Fetch
    - Se obtiene la instrucción apuntada por el **PC** de la **Memoria**
    - La **ALU** incrementa el **PC**
  - Decode
    - Se decodifica la instrucción
    - Se obtienen los operandos de la **Memoria** y se los coloca en **Registros**
  - Execute
    - La **ALU** realiza la operación
    - Se coloca el resultado en la **Memoria**

# Arquitectura de von Newmann

## → Ciclo de instrucción (UC):

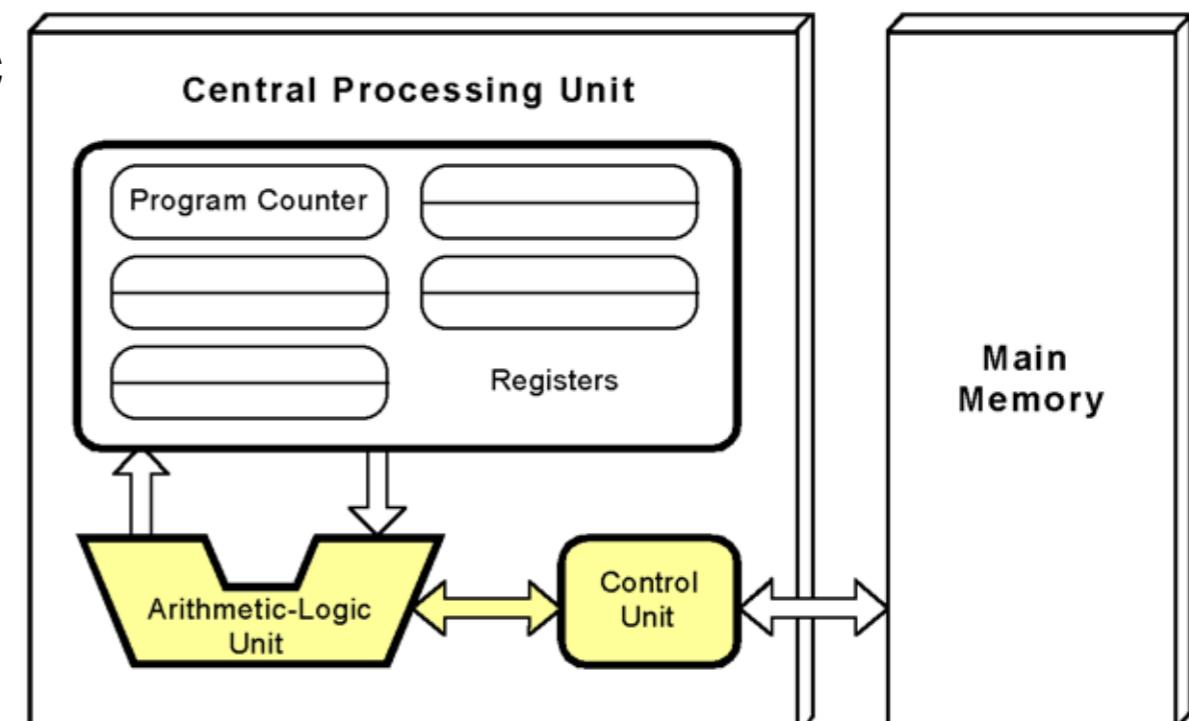
- |                |   |
|----------------|---|
| <b>Fetch</b>   | <ul style="list-style-type: none"><li>- Se obtiene la instrucción apuntada por el <b>PC</b> de la <b>Memoria</b></li><li>- La <b>ALU</b> incrementa el <b>PC</b></li></ul>  |
| <b>Decode</b>  | <ul style="list-style-type: none"><li>- Se decodifica la instrucción</li><li>- Se obtienen los operandos de la <b>Memoria</b> y se los coloca en <b>Registros</b></li></ul> |
| <b>Execute</b> | <ul style="list-style-type: none"><li>- La <b>ALU</b> realiza la operación</li><li>- Se coloca el resultado en la <b>Memoria</b></li></ul>                                  |



# Arquitectura de von Newmann

## → Ciclo de instrucción (UC):

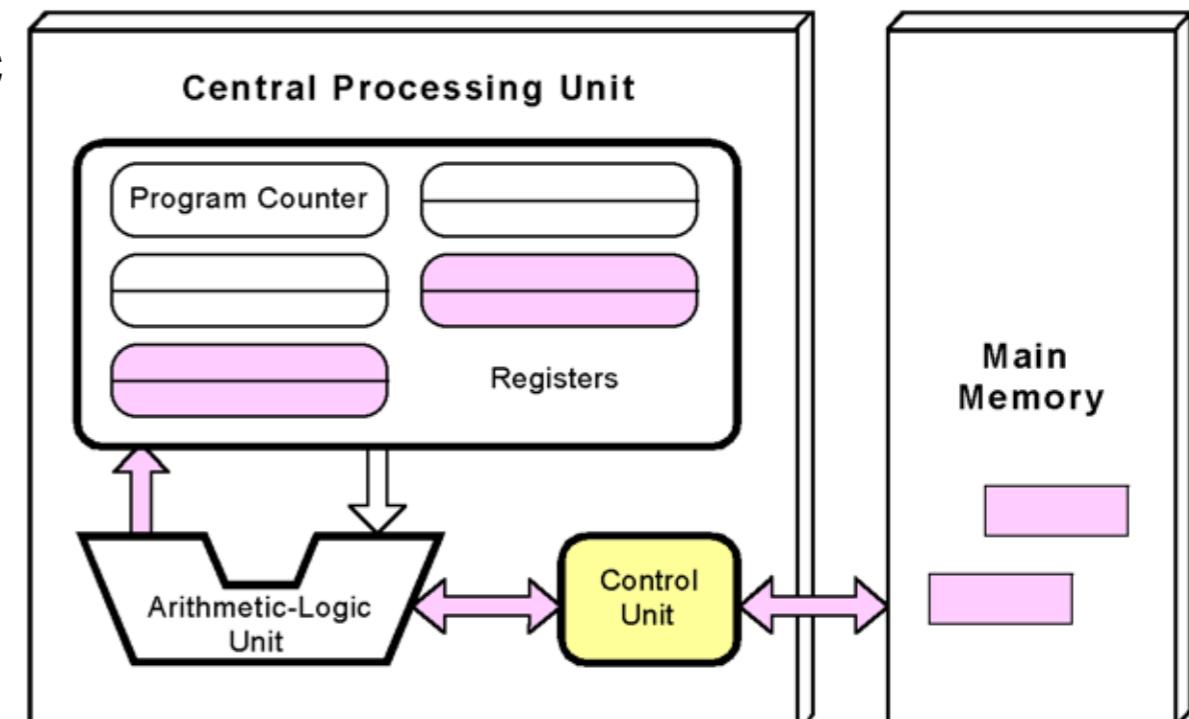
- Se obtiene la instrucción apuntada por el **PC** de la **Memoria**
- La **ALU** incrementa el **PC**
- Se decodifica la instrucción
- Se obtienen los operandos de la **Memoria** y se los coloca en **Registros**
- La **ALU** realiza la operación
- Se coloca el resultado en la **Memoria**



# Arquitectura de von Newmann

## → Ciclo de instrucción (UC):

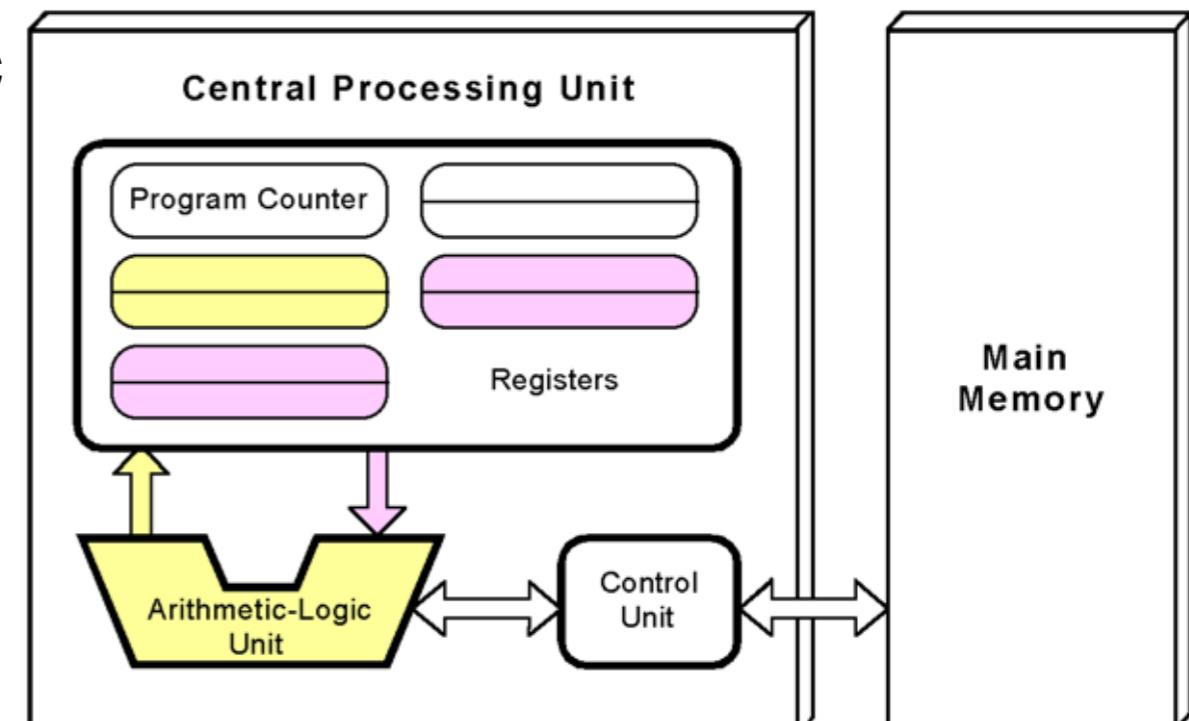
- Se obtiene la instrucción apuntada por el **PC** de la **Memoria**
- La **ALU** incrementa el **PC**
- Se decodifica la instrucción
- Se obtienen los operandos de la **Memoria** y se los coloca en **Registros**
- La **ALU** realiza la operación
- Se coloca el resultado en la **Memoria**



# Arquitectura de von Newmann

## → Ciclo de instrucción (UC):

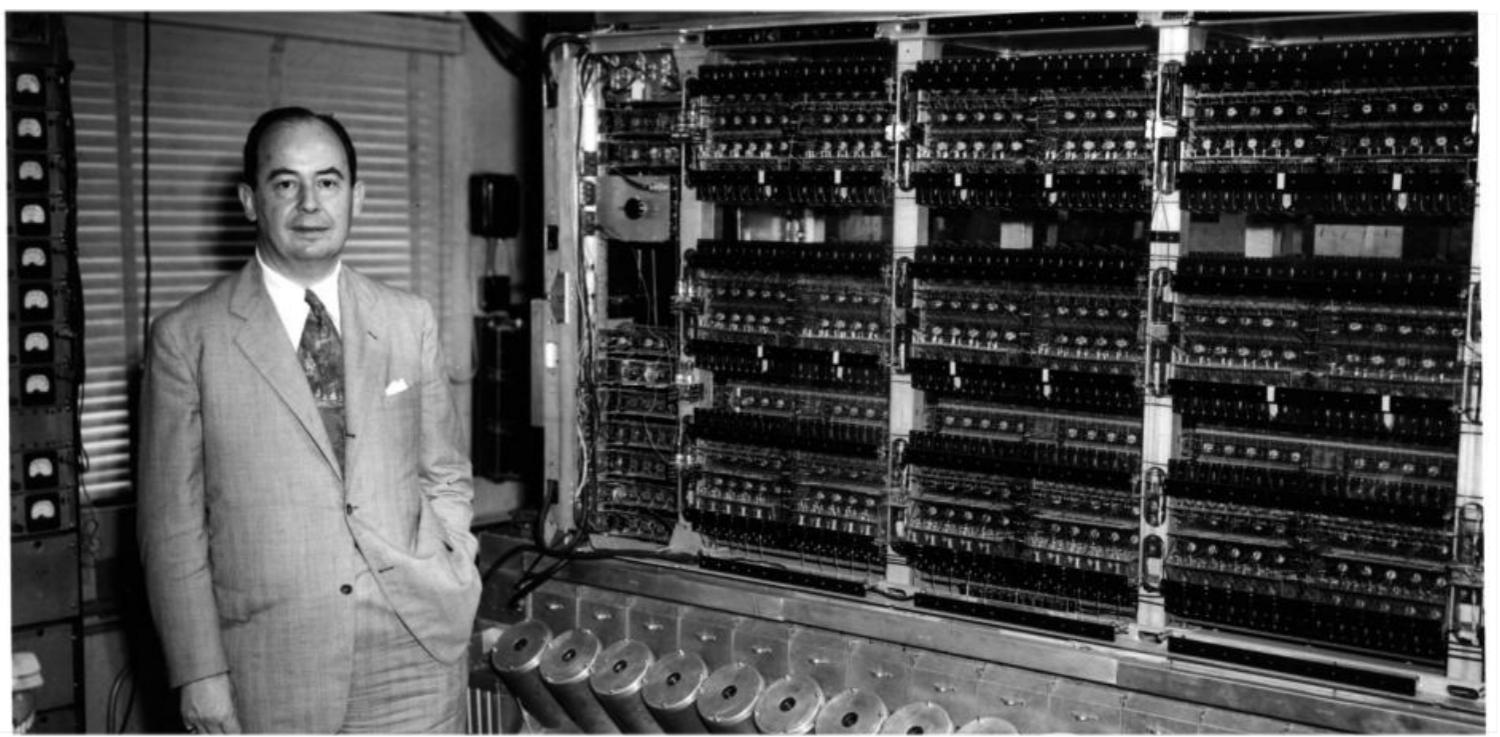
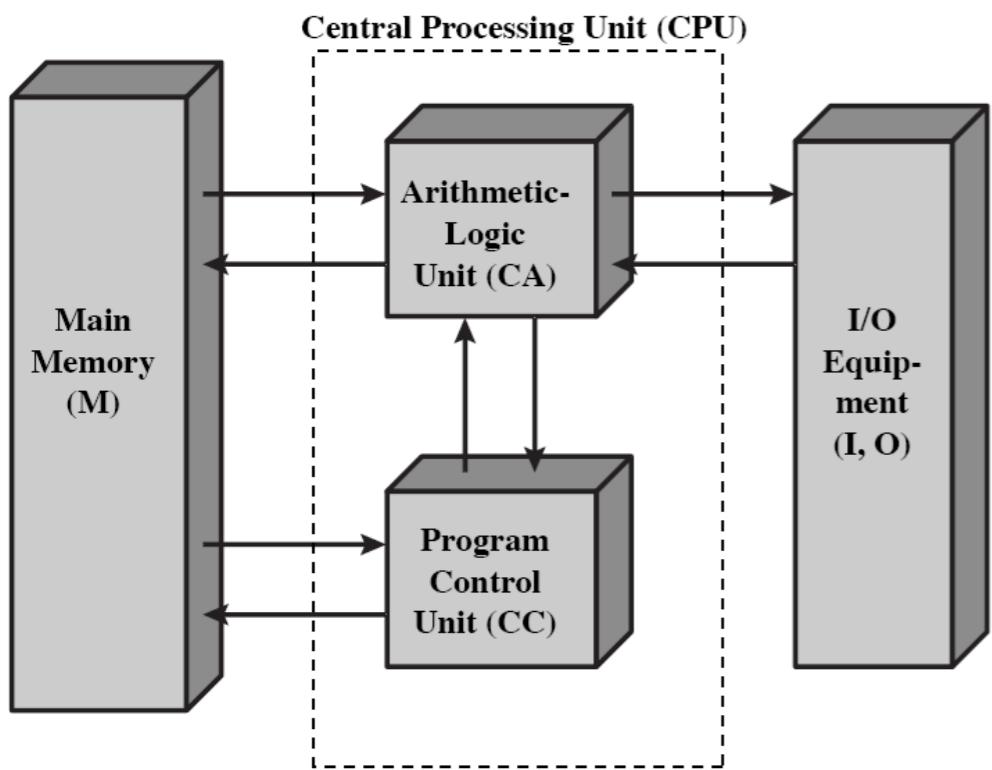
- Se obtiene la instrucción apuntada por el **PC** de la **Memoria**
- La **ALU** incrementa el **PC**
- Se decodifica la instrucción
- Se obtienen los operandos de la **Memoria** y se los coloca en **Registros**
- La **ALU** realiza la operación
- Se coloca el resultado en la **Memoria**



# Primera implementación de Arquitectura de von Neumann

IAS machine (1945-1951)

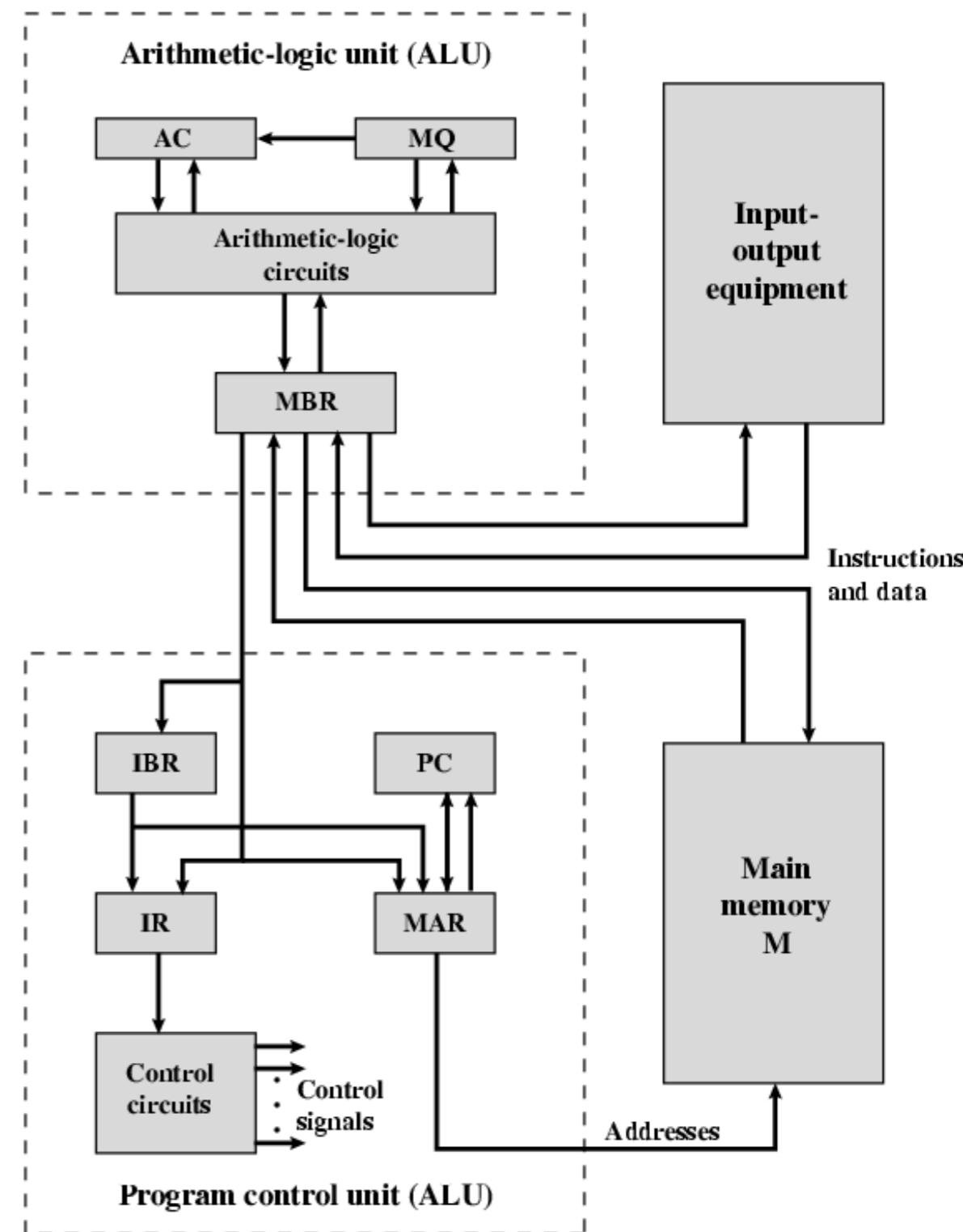
- \* Diseñada por el Institute of Advanced Studies sobre la base de The First Draft
- \* Fue utilizada para simular los efectos de la detonación de una bomba de hidrógeno
- \* Resolvió en 10 minutos una simulación climática que a ENIAC le tomó 36 horas.



# Primera implementación de Arquitectura de von Newmann

## →Registros:

- MBR: Memory buffer register
- MAR: Memory address register
- IR: Instruction register
- IBR: Instruction buffer register
- PC: Program counter
- AC: Accumulator
- MQ: Multiplier quotient

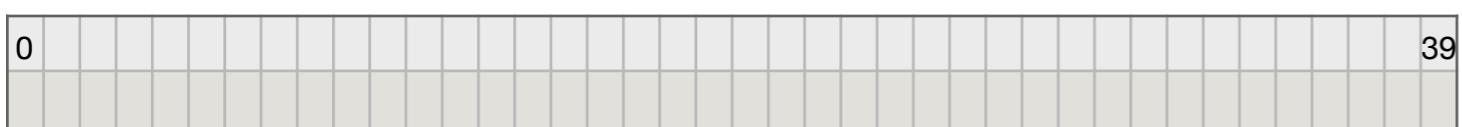


# Primera implementación de Arquitectura de von Neumann

## → Representación de la información:

- Memoria: 1000 palabras de 40 bits
- Representación binaria
- 2 instrucciones por palabra

Dato



Magnitud

Signo

Instrucciones

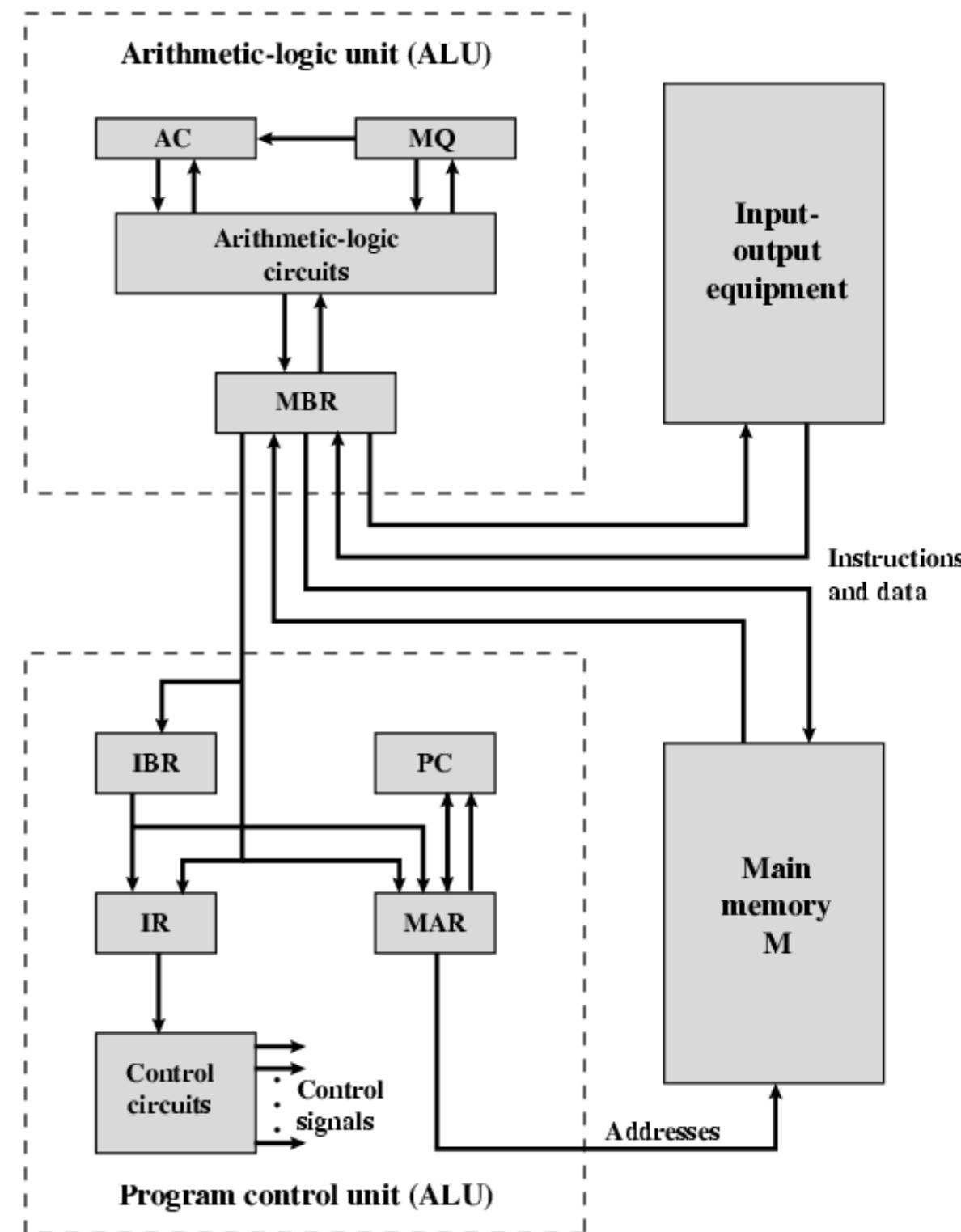


Código de operación

Operando (Dirección)

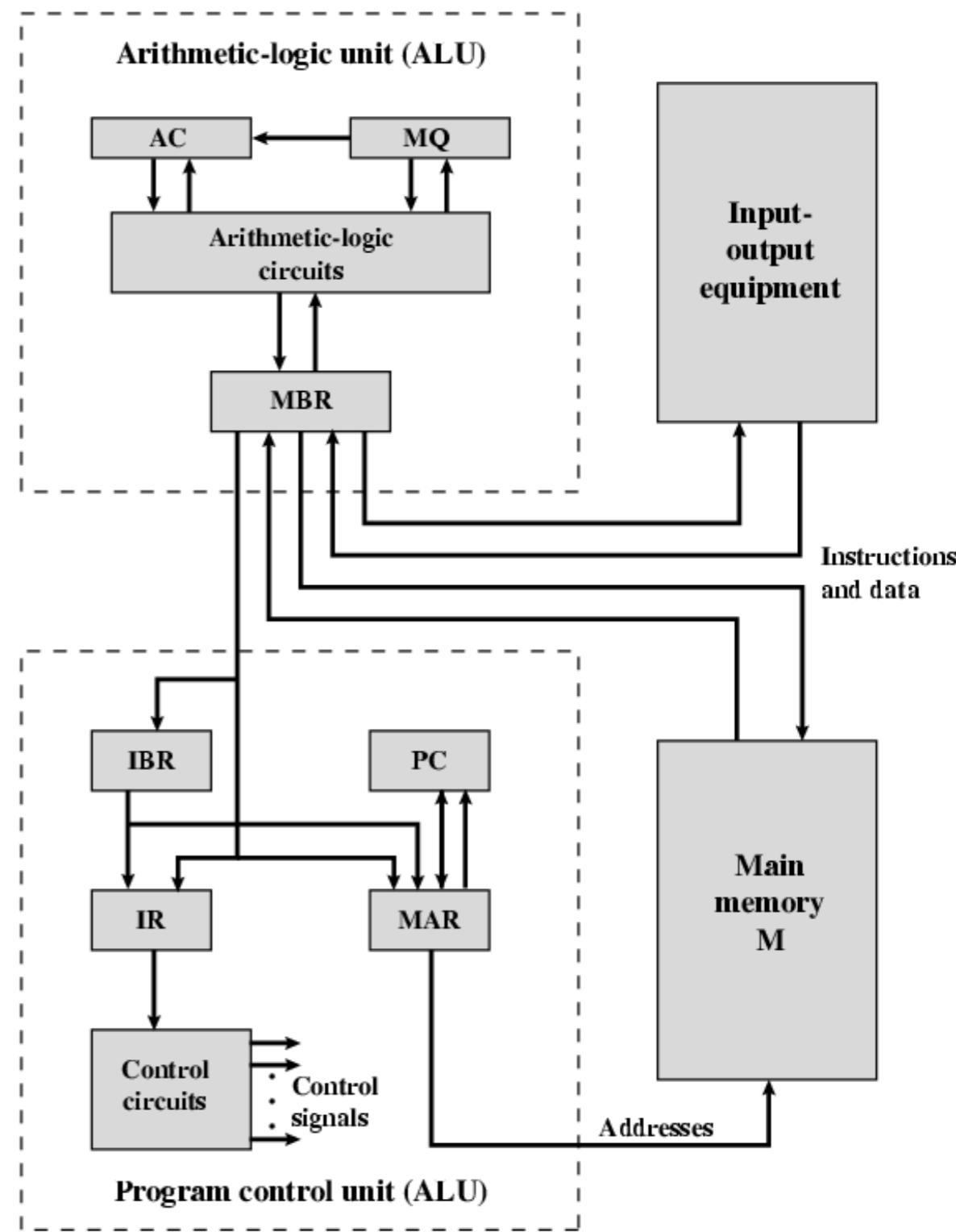
Código de operación

Operando (Dirección)



# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

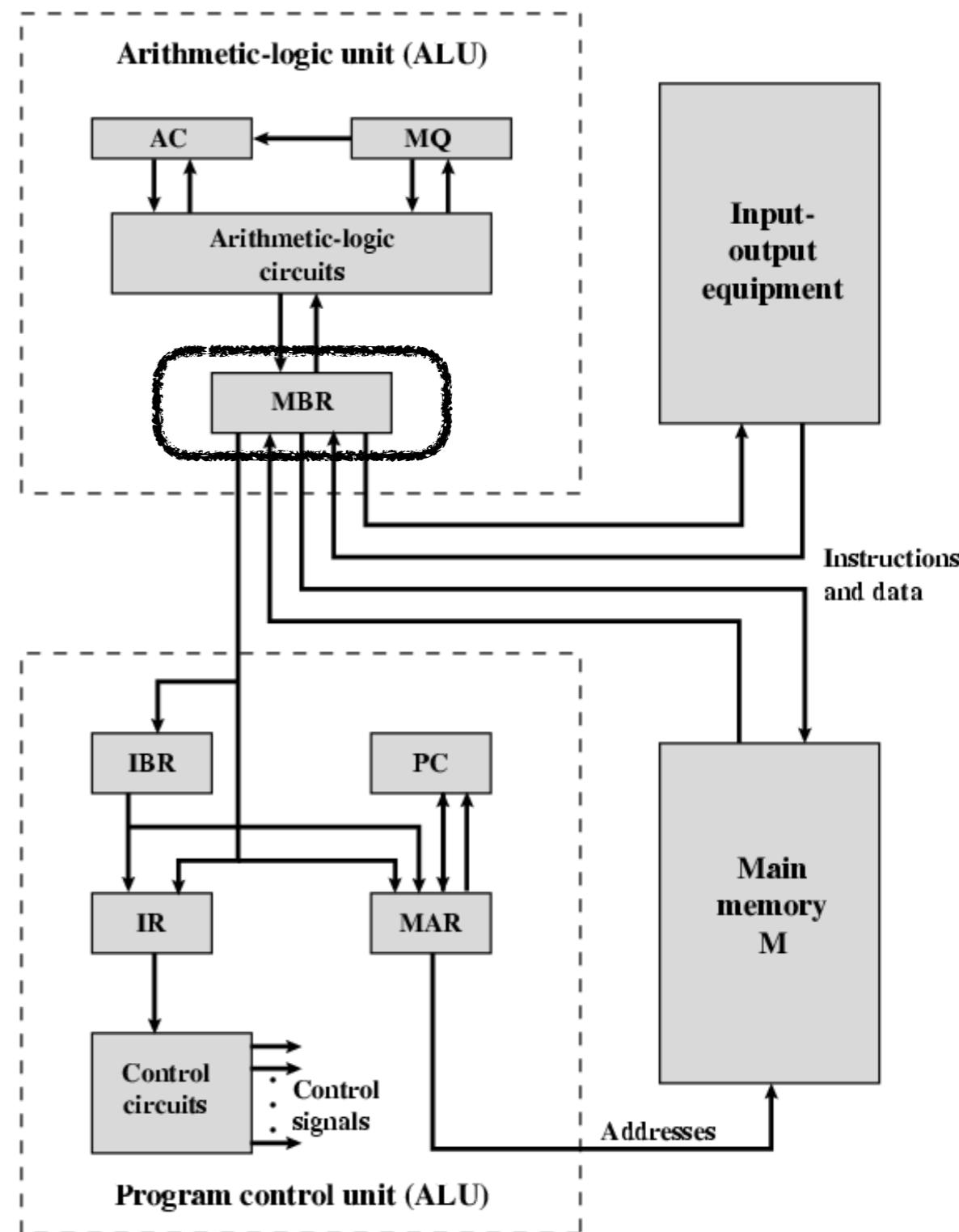


# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

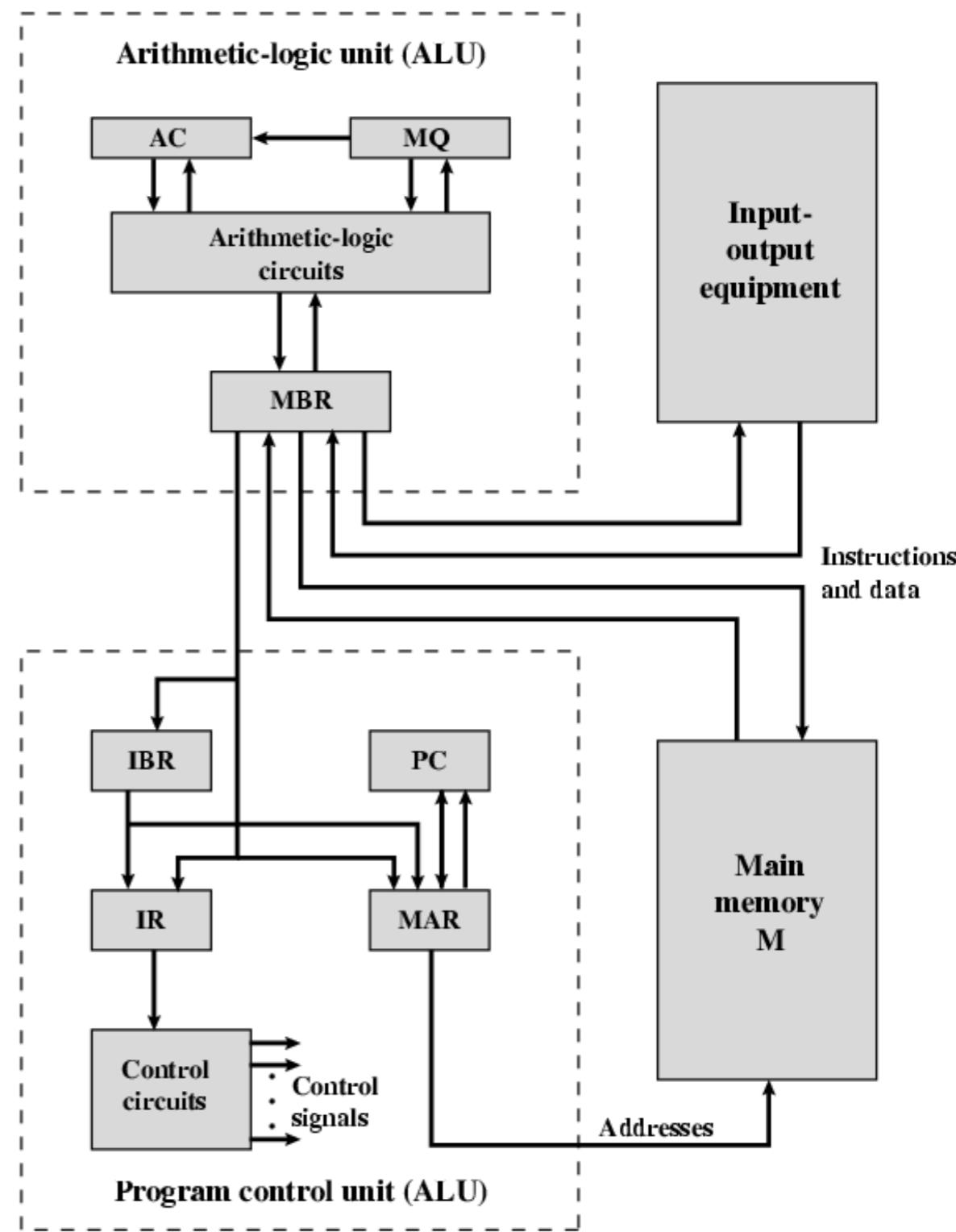
### → MBR (Memory buffer register):

- 40 bits de longitud,
- contiene la palabra que proviene de la memoria / debe ser almacenada en la memoria



# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

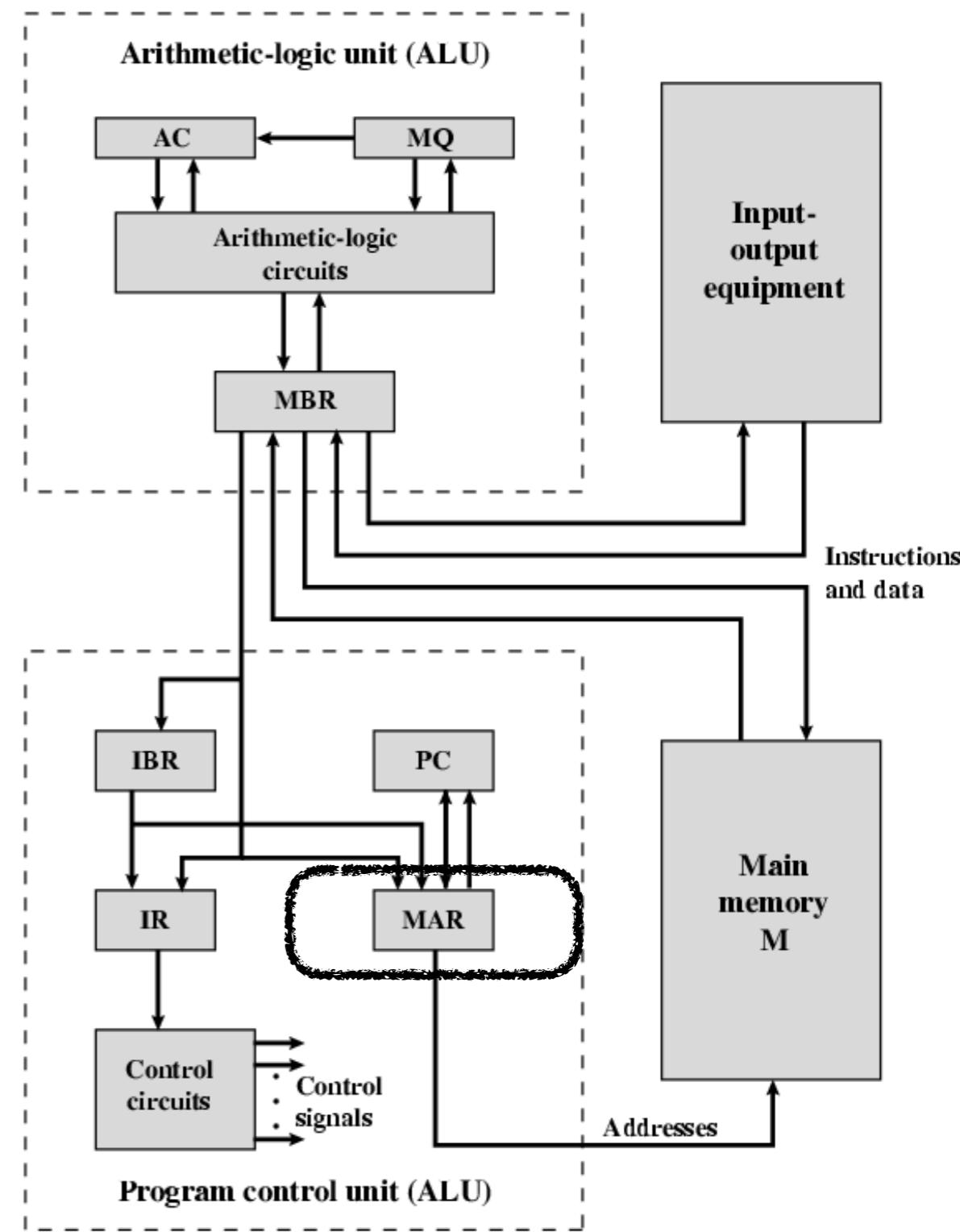


# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

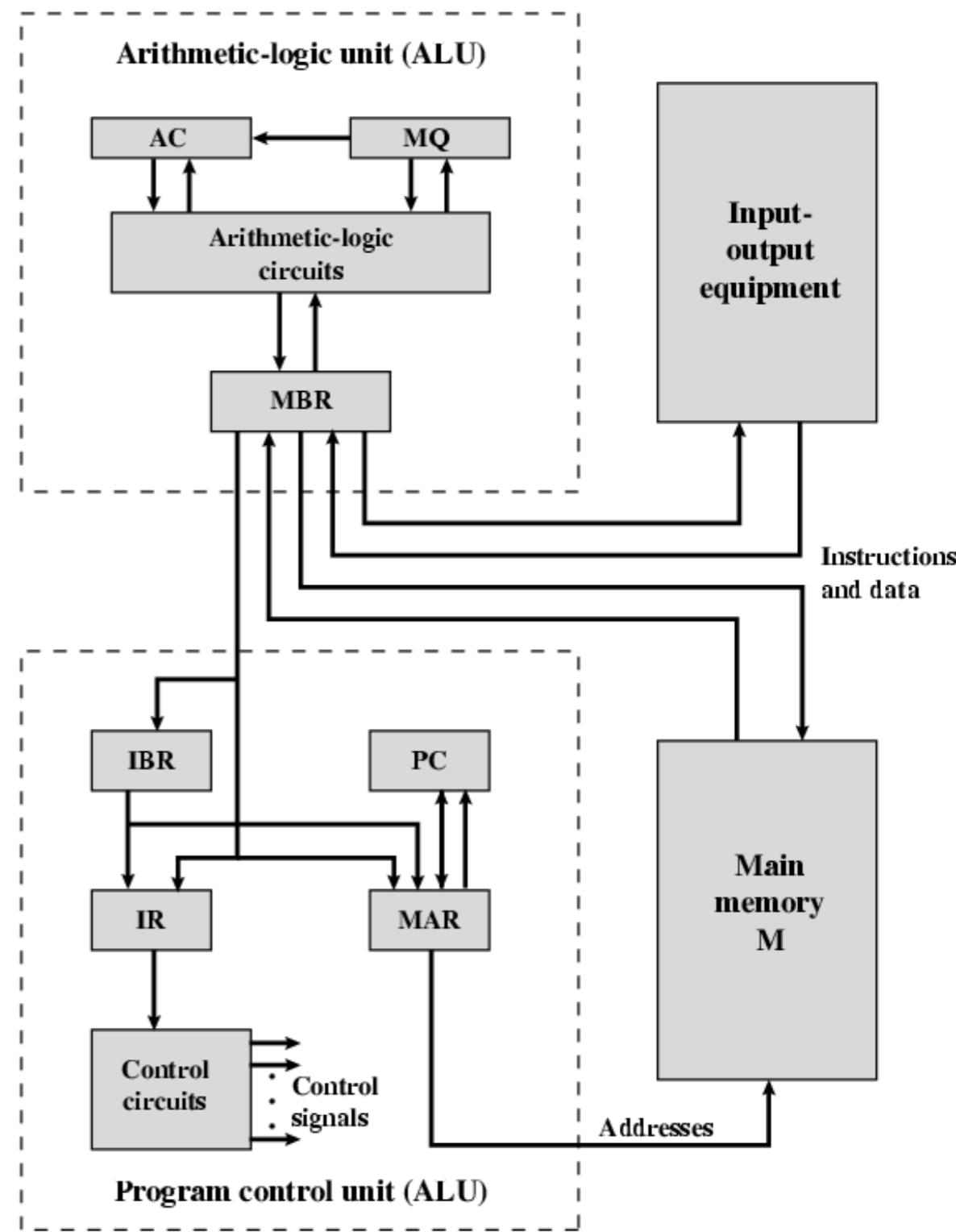
### → MAR (Memory address register):

- 12 bits de longitud,
- contiene la dirección de la palabra que será leída de la memoria / almacenada en la memoria



# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

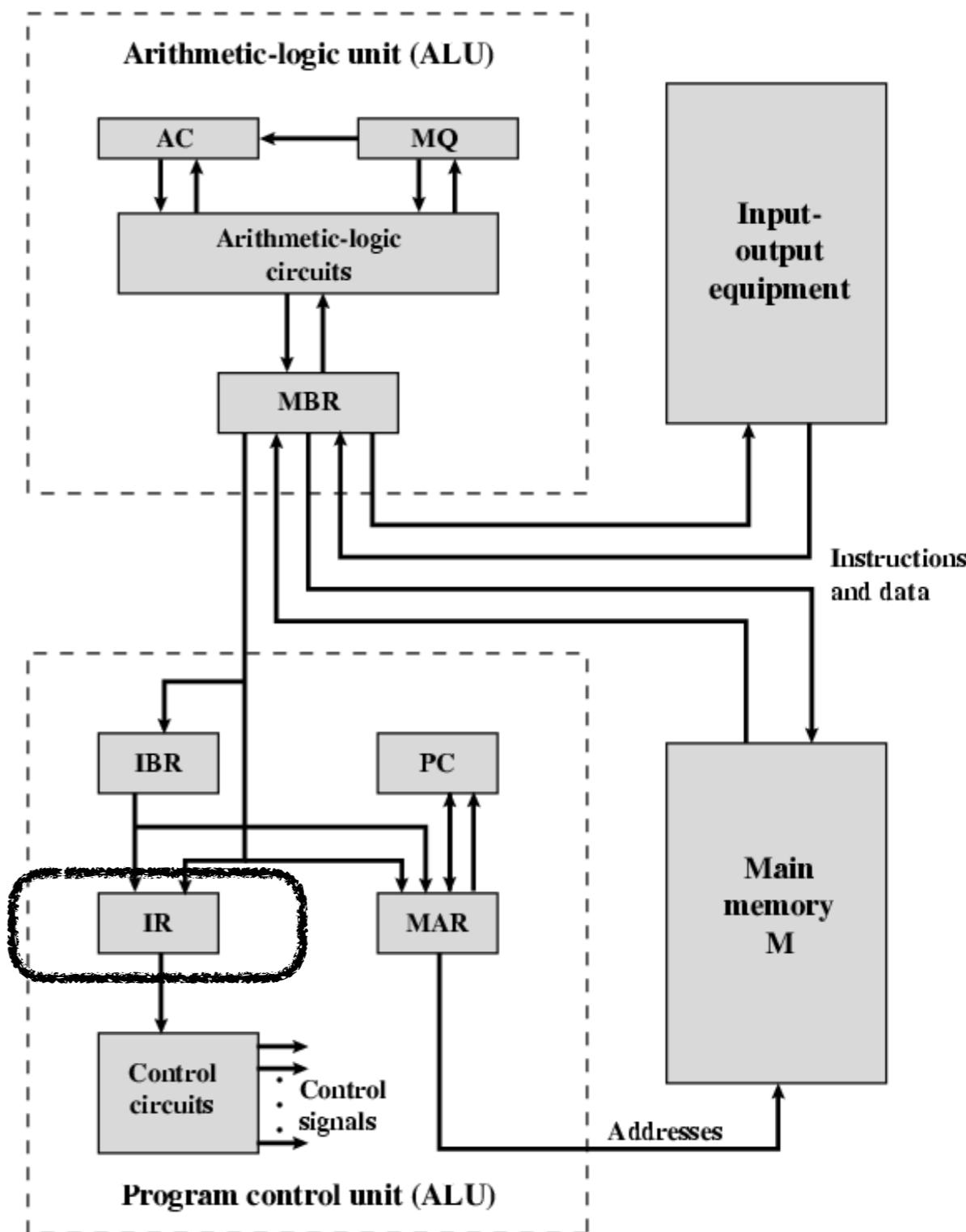


# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

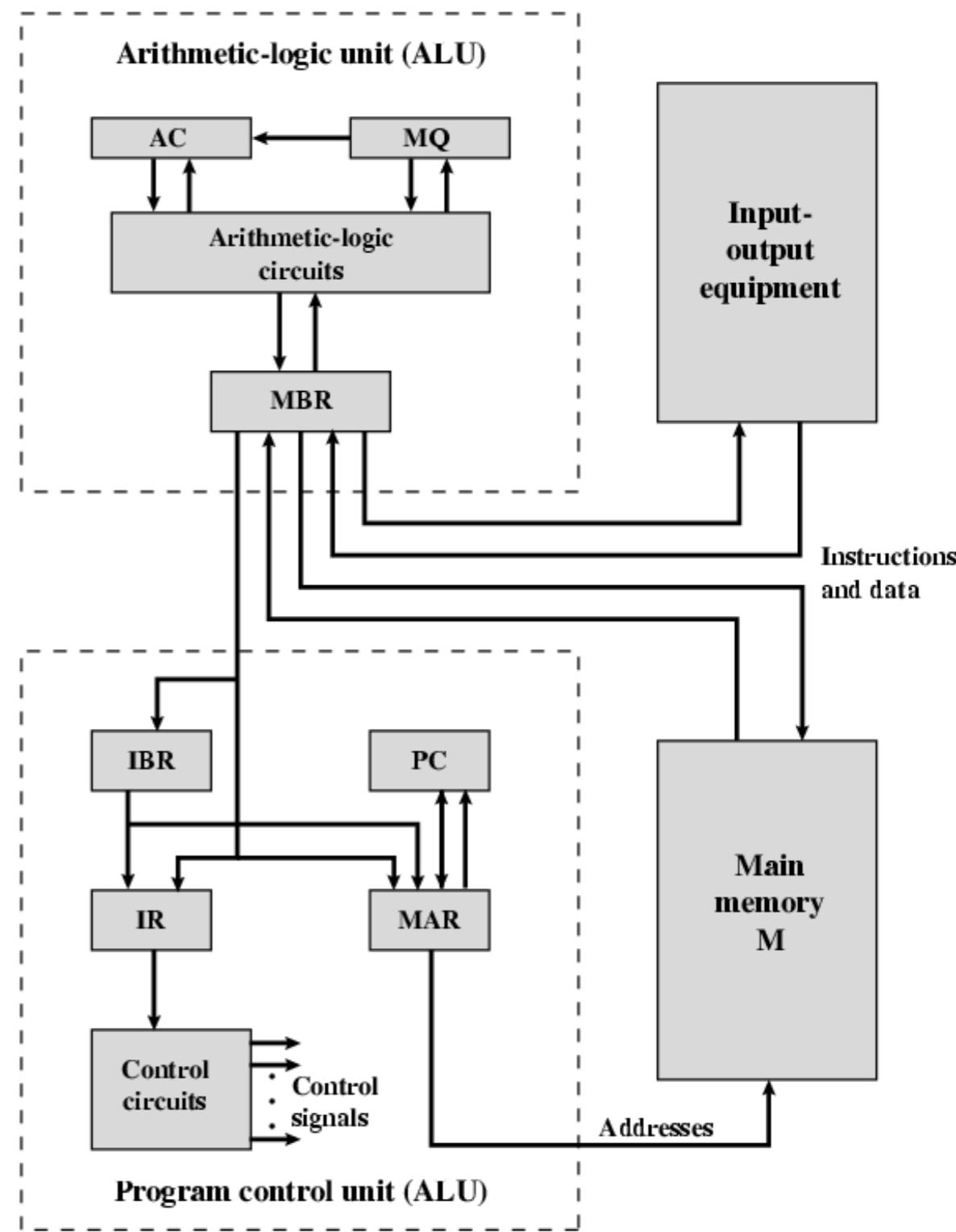
### → IR (Instruction register):

- 8 bits de longitud,
- contiene el código de operación de la operación que debe ser ejecutada



# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

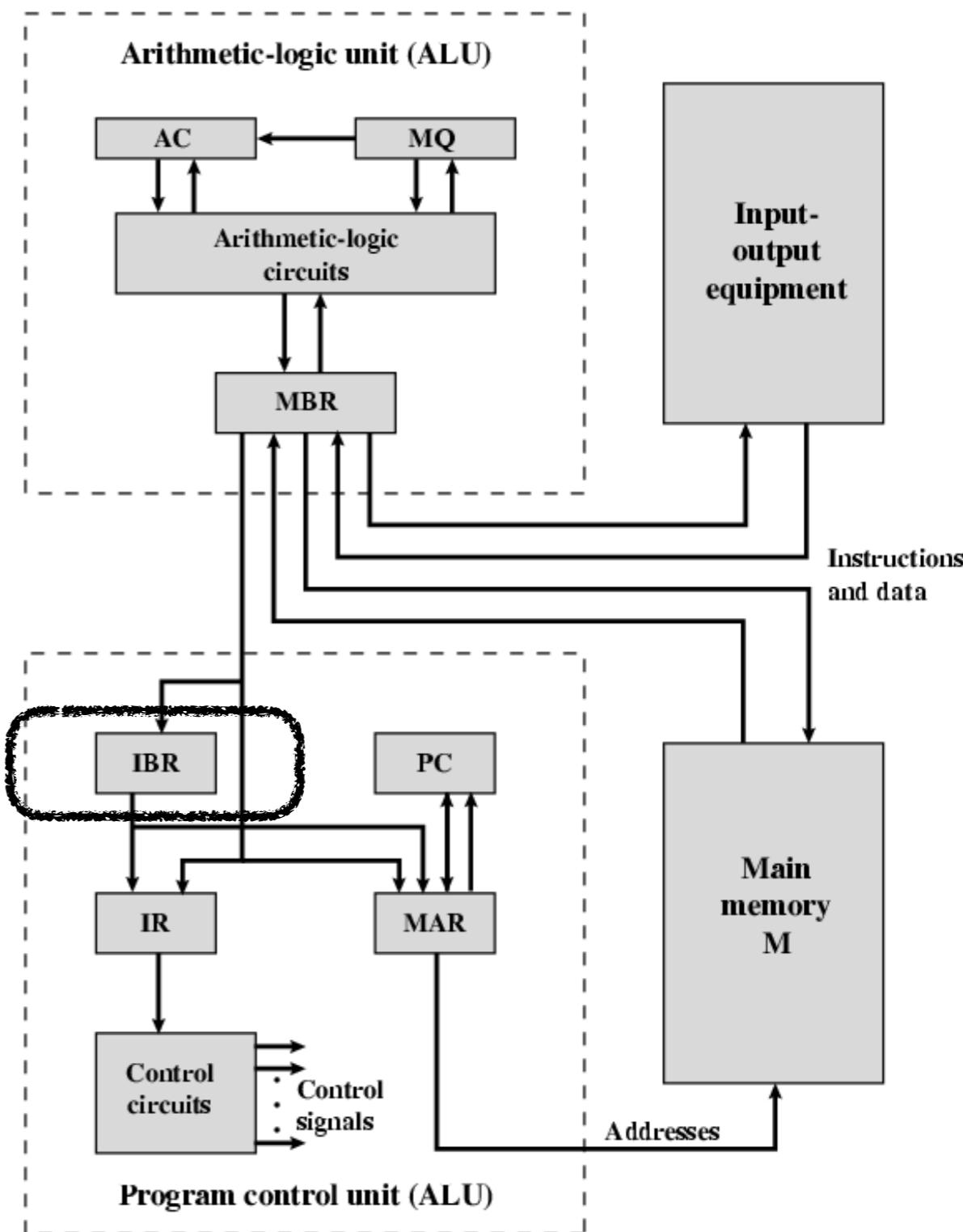


# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

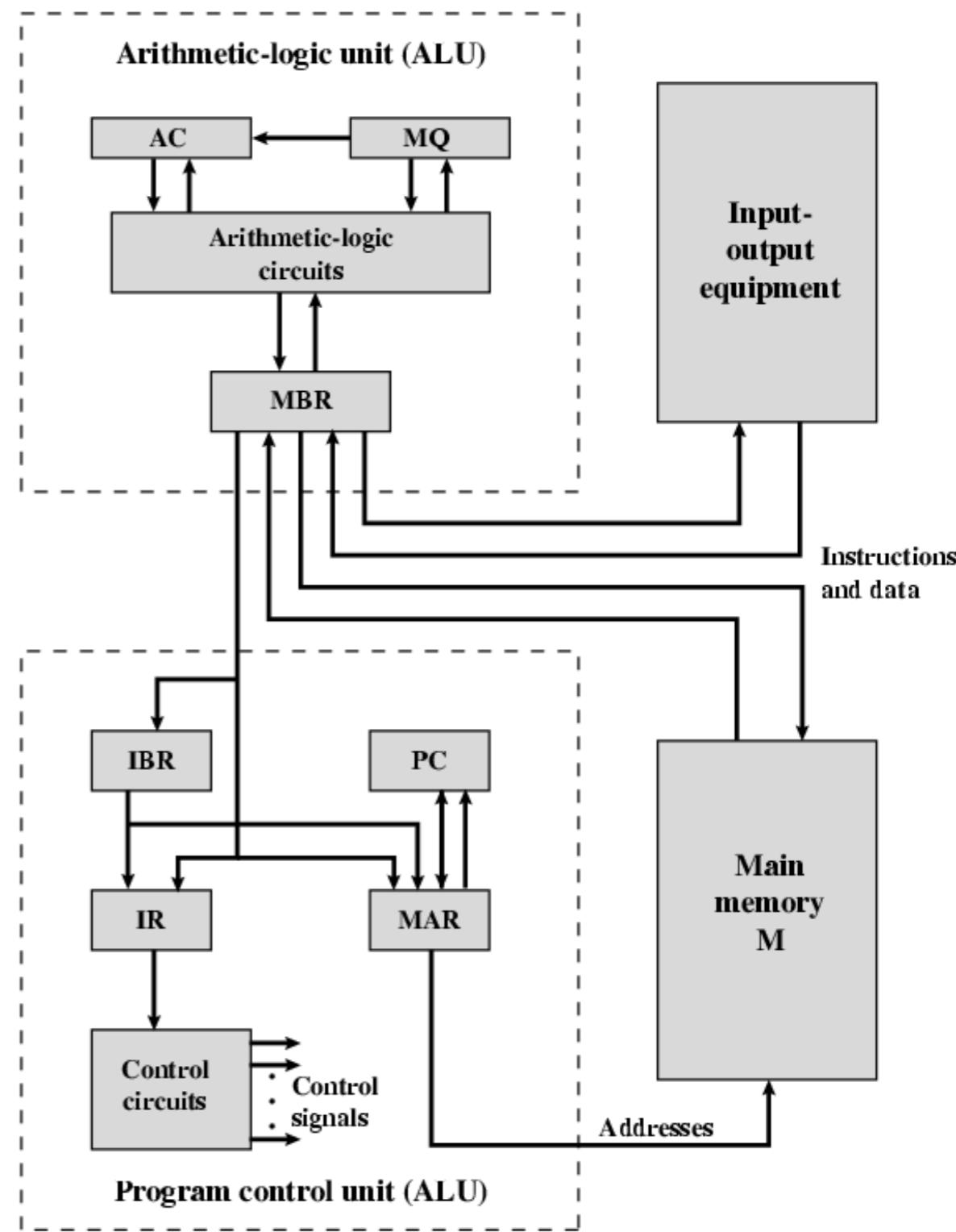
### →IBR (Instruction buffer register):

- 20 bits de longitud,
- contiene temporalmente la parte derecha de la palabra de instrucción leída de memoria



# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

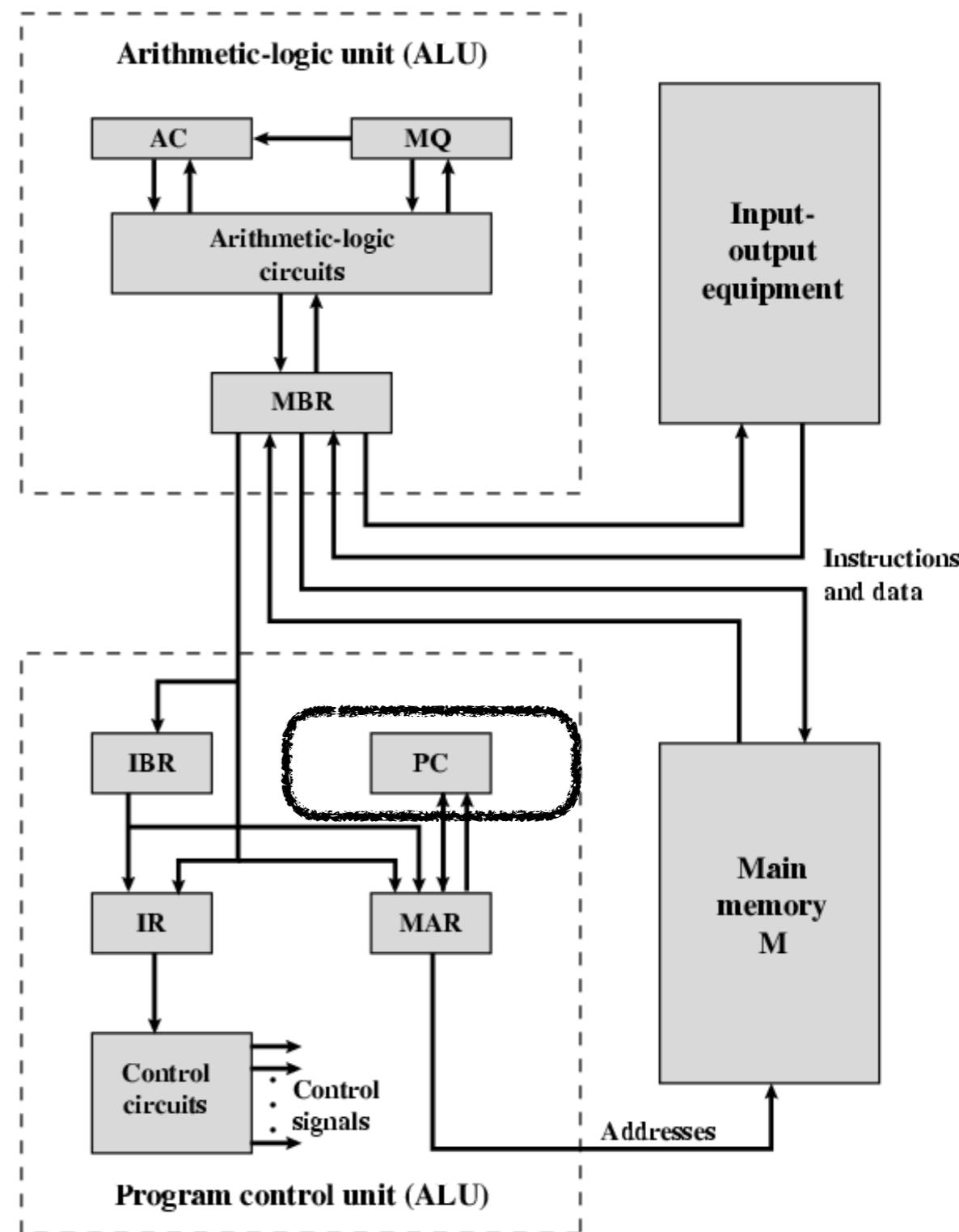


# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

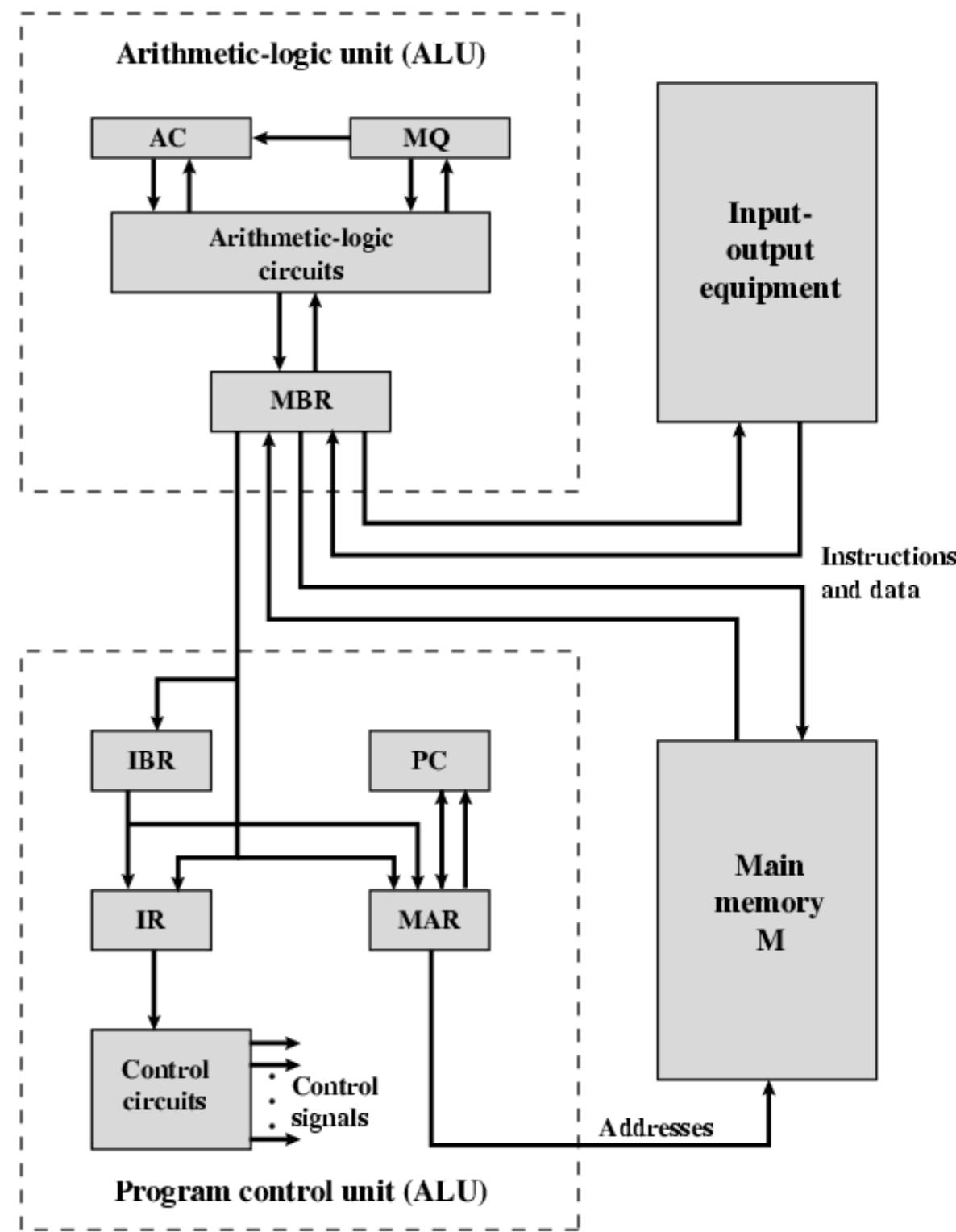
### → PC (Program counter):

- 12 bits de longitud,
- contiene la dirección de la próxima palabra de instrucción a leer de la memoria (i.e. de las dos instrucciones siguientes)



# Primera implementación de Arquitectura de von Newmann

## Detalle de la arquitectura

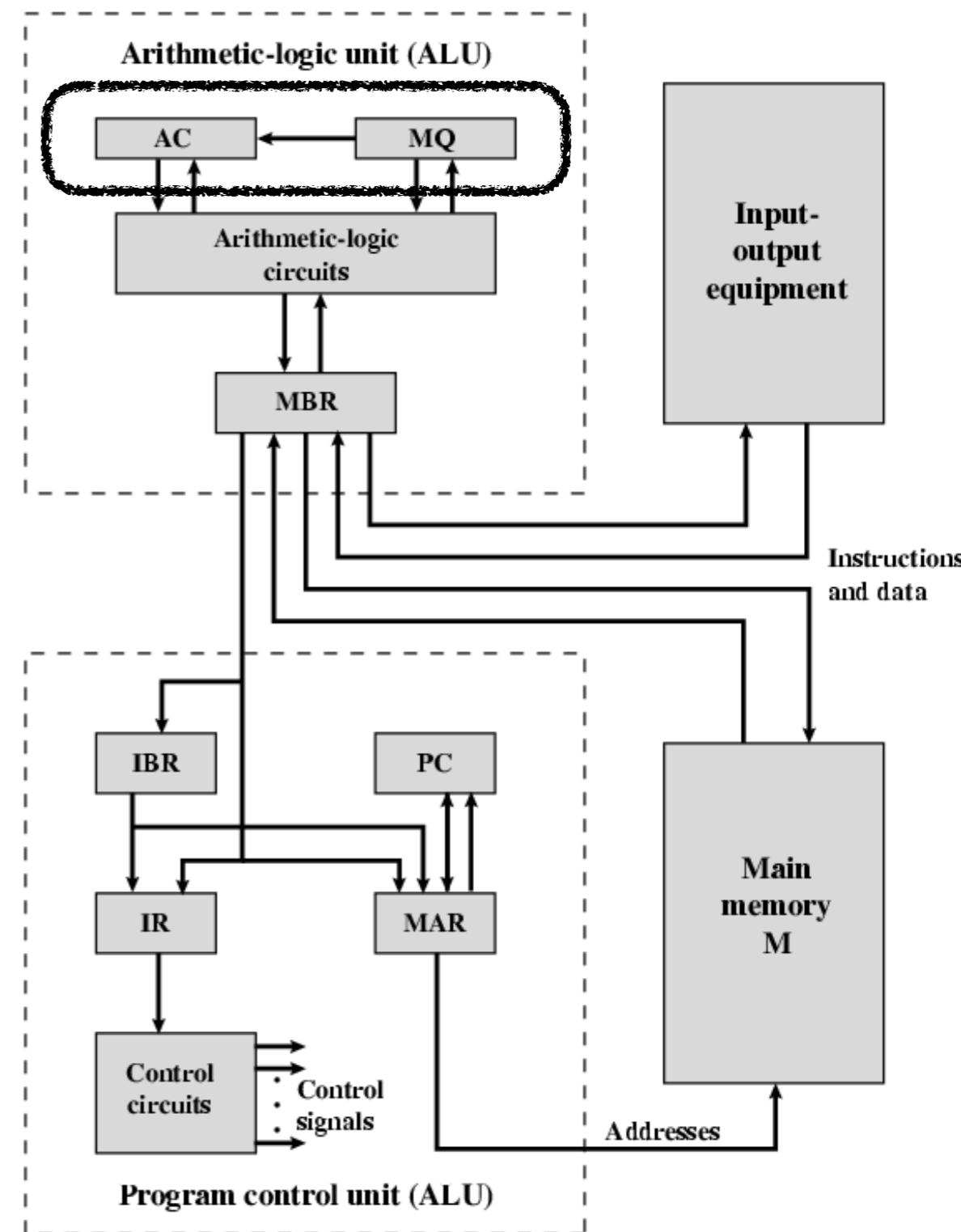


# Primera implementación de Arquitectura de von Newmann

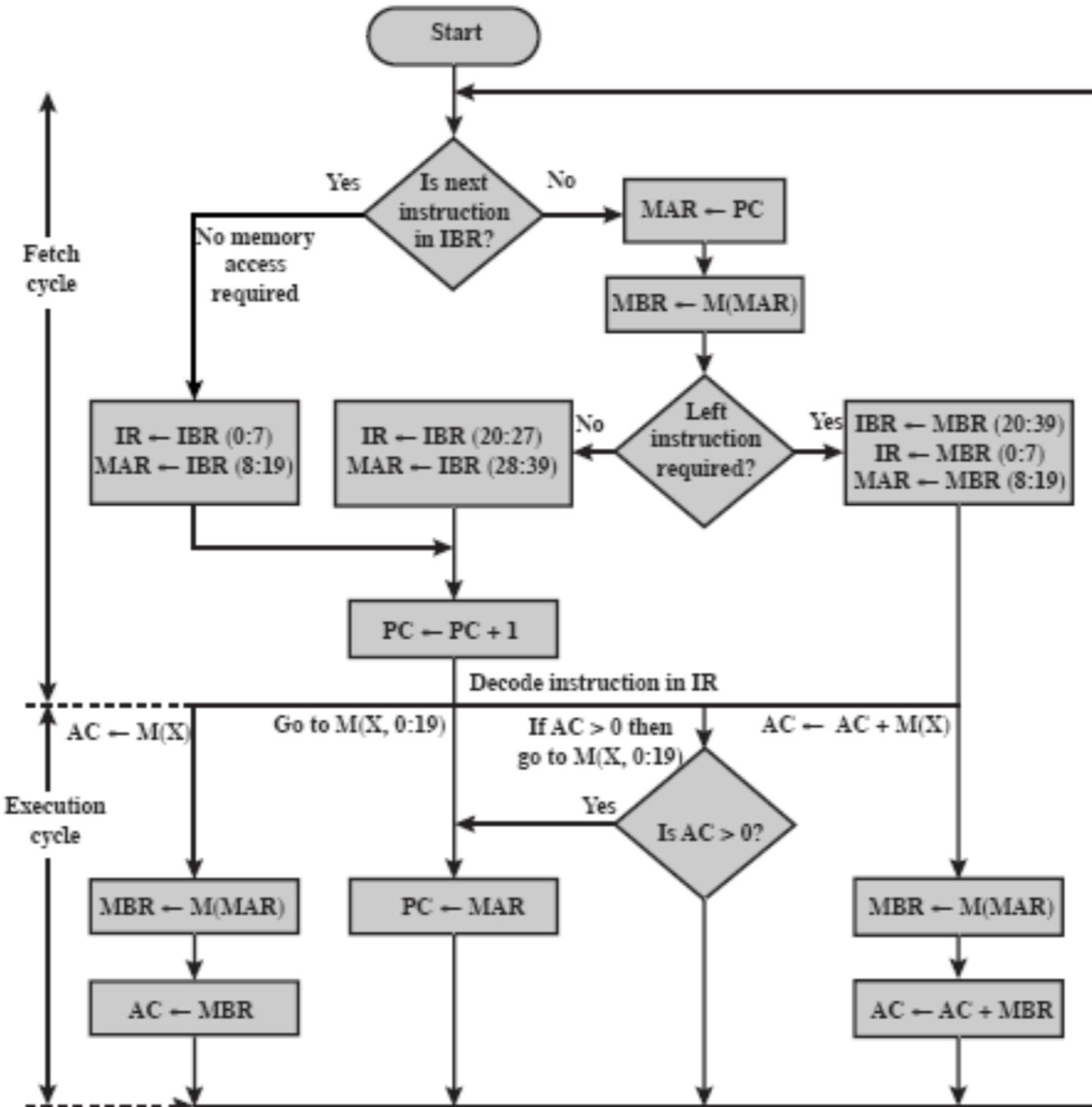
## Detalle de la arquitectura

### → AC y MQ (Accumulator y Multiplier quotient):

- 40 bits de longitud,
- almacenan los operandos y resultados de la **ALU** temporalmente hasta que estos sean colocados en la memoria. Son necesarios dos porque porque el producto de dos magnitudes de 40 bits resulta en una magnitud de 80 bits. **AC** contiene los bits más significativos y **MQ** los menos significativos

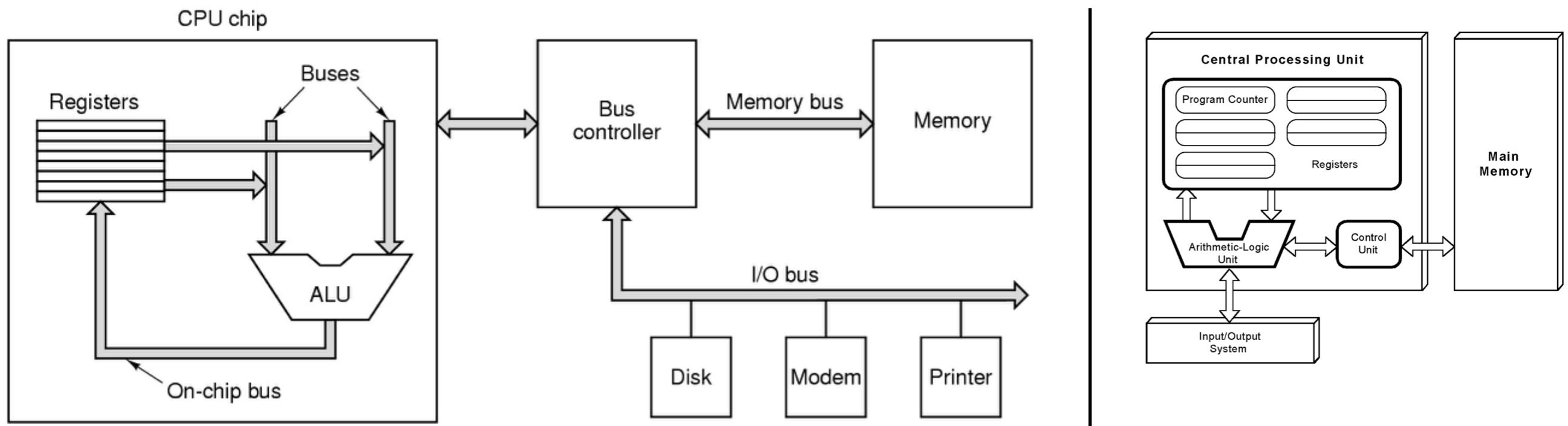


# Primera implementación de Arquitectura de von Newmann



$M(X)$  = contents of memory location whose address is  $X$   
 $(i:j)$  = bits  $i$  through  $j$

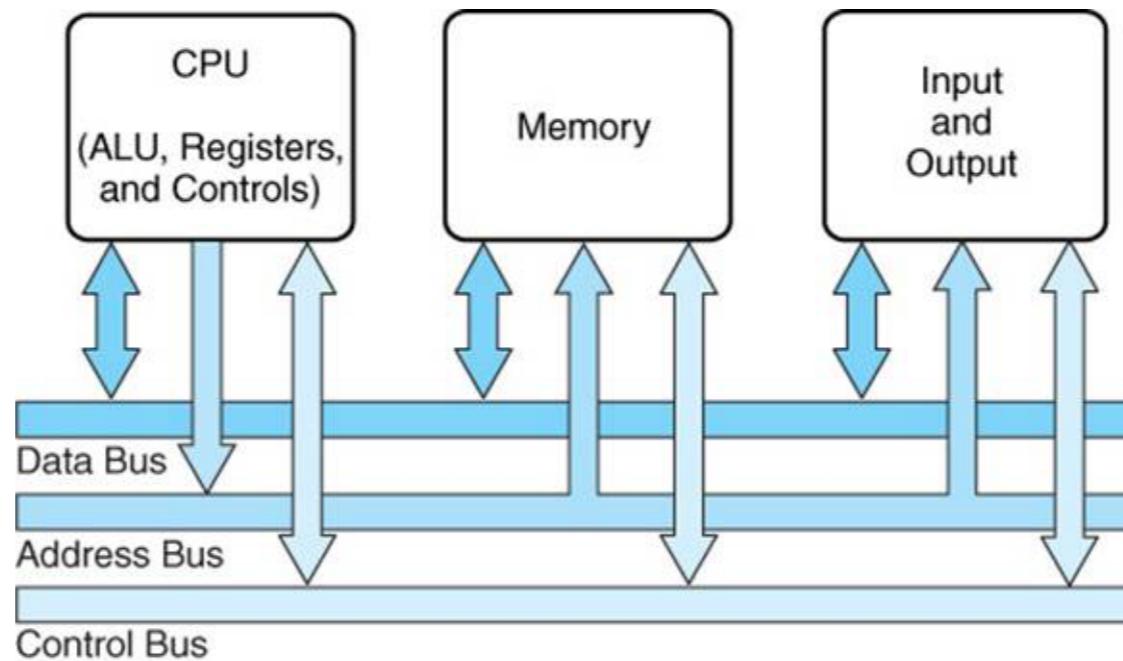
# Primera implementación de Arquitectura de von Neumann



## → Bus:

- vía de comunicación que interconecta componentes,
- funcionan como “broadcast” (todos reciben lo que alguien envía)
- solo las componentes involucradas e la comunicación deben “prestar atención”
- agrupan varias señales

# Primera implementación de Arquitectura de von Neumann



## → Data bus:

- tantas líneas físicas como bits tiene una palabra
- transfiere información entre las componentes; por ejemplo, es el canal por el cual la **ALU** envía / recibe los operandos hacia / desde la **Memoria**

## → Address bus:

- tantas líneas físicas como bits necesarios para direcciones todas las palabras en la **Memoria**
- identifica la posición de **Memoria** a ser leída / escrita

## → Control bus:

- tantas líneas físicas como sean necesarias
- señales auxiliares que permiten la coordinación de las componentes: R/W, Memory Ready, Reloj, Interrupciones, etc.

# Jerarquía de máquina

Nivel 6	Usuario	Programa ejecutables
Nivel 5	Lenguaje de alto nivel	C++, Java, Python, etc.
Nivel 4	Lenguaje ensamblador	Assembly code
Nivel 3	Software del sistema	Sistema operativo, bibliotecas, etc.
Nivel 2	Lenguaje de máquina	Instruction Set Architecture (ISA)
Nivel 1	Unidad de control	Microcódigo / hardware
Nivel 0	Lógica digital	Circuitos, compuertas, memorias

- Cada nivel funciona como una máquina abstracta que oculta la capa anterior
- Cada nivel es capaz de resolver determinado tipo de problemas a partir de comprender un tipo de instrucciones específico
- La capa inferior es utilizada como servicio

# Jerarquía de máquina

## Nivel 6 - Usuario Programas ejecutables

- ➔ Ejecución de aplicaciones utilizadas por los usuarios
- ➔ Se piensa en términos de aplicaciones con un propósito específico que cuentan con una interfaz para que los usuarios puedan disponer de su funcionalidad

# Jerarquía de máquina

Nivel 5 - Lenguaje de alto nivel  
C++, Java, Python, etc.

- ➤ Programación de aplicaciones que serán utilizadas por los usuarios
- ➤ Se piensa en términos de lenguajes de programación de propósito general, es decir, que permiten el desarrollo de cualquier aplicación
- ➤ Algoritmos y estructuras de datos

# Jerarquía de máquina

## Nivel 4 - Lenguaje ensamblador Assembly code

- ➤ Lenguaje de programación muy cercano a la arquitectura de la computadora
- ➤ El código en este lenguaje suele ser producido por compiladores como lenguaje intermedio o por programadores más especializados

# Jerarquía de máquina

Nivel 3 - Software del sistema  
Sistema operativo, bibliotecas, etc.

- ➤ Infraestructura del sistema para la ejecución de procesos, protección de recursos
- ➤ Brinda servicios de acceso a dispositivos Entrada / Salida

# Jerarquía de máquina

## Nivel 2 - Lenguaje de máquina Instruction Set Architecture (ISA)

- ➤ Interpretación de cadenas de bits como instrucciones que pueden ser comprendidas por la unidad de control
- ➤ Son específicas de la arquitectura de la máquina pues asume la existencia de ciertos registros, señales, etc.
- ➤ No necesita ser compilada ni ensamblada; una vez en memoria puede ser ejecutada directamente

# Jerarquía de máquina

## Nivel 1 - Unidad de control Microcódigo / hardware

- La unidad de control implementa el ciclo de instrucción de forma que es quien realmente ejecuta los programas
- Puede ser microprogramada o implementada en hardware

	Implementada en hardware	Microprogramada
Velocidad	Compuertas lógicas (rápida)	Interpretación de programas (lenta)
Complejidad	Gran cantidad de componentes	Programación
Versatilidad	Inmodificable	Modifiable
Flexibilidad	Atado a la arquitectura	Variedad de implementaciones

# Jerarquía de máquina

Nivel 0 - Lógica digital

Circuitos, compuertas, memorias

- ➤ Circuitos digitales (funciones y memorias)
- ➤ Compuertas lógicas y cables interconectándolas
- ➤ Manipulación explícita de la representación de los bits  
(+|- 5v)