Programación I

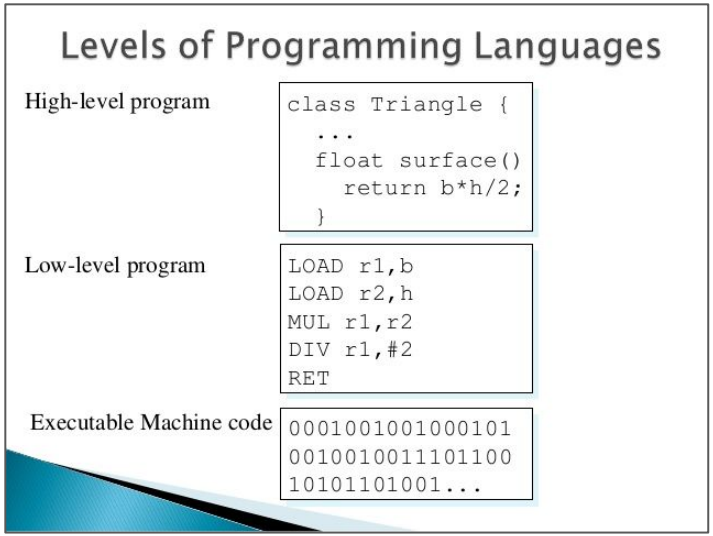
Organización de Computadoras

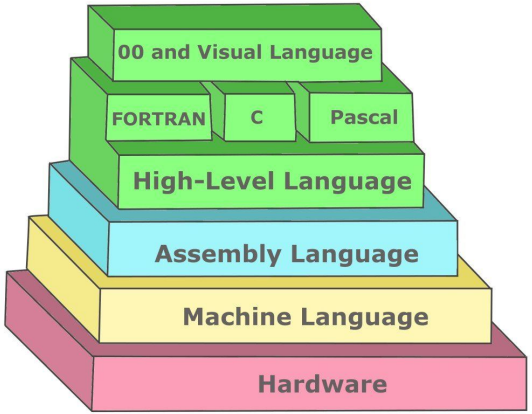
Primera Parte

Programación I

Contenidos de la materia dividido en dos partes principales:

- Conceptos de “alto nivel”
  - Algoritmos, modelización, tipos y estructuras de datos, eficiencia
  - Abstracción respecto del hardware
- Conceptos de “bajo nivel”
  - Organización de computadoras: computadora y sus partes, ciclo de instrucción, circuitos secuenciales, representación de números
  - Relación directa con el hardware





Temas de la primera parte

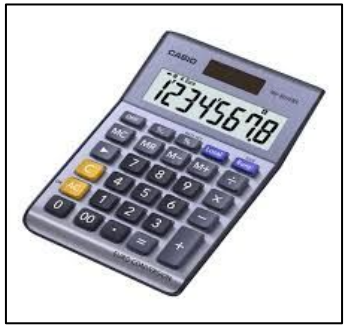
- Introducción y definiciones ←
- Evolución Histórica
- Arquitectura Von Neumann
- CPU
- Ciclo de instrucción
- Lógica digital

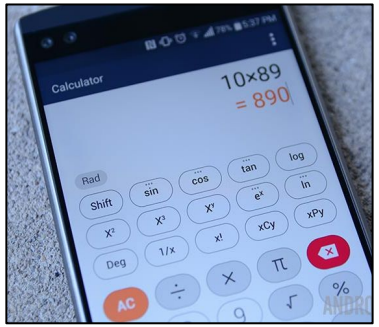
Definiciones y conceptos básicos.

SOFTWARE
HARDWARE

*“Hardware y software son **funcionalmente** equivalentes”*

- Máquinas:
  - De propósito **específico**
  - De propósito **general**





Definiciones y conceptos básicos.

- Computadora
  - Máquina
  - Digital
  - Sincrónica
  - Cálculo númerico
  - Cálculo lógico
  - Controlada por un programa
  - Comunicación con el mundo exterior

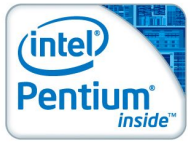
Definiciones y conceptos básicos.

- Arquitectura
  - Aquellos atributos **visibles al programador**
  - Conjunto de instrucciones, número de bits usados para representación de datos, mecanismos de E/S, técnicas de direccionamiento.
  - Ejemplo: *¿Existe la instrucción de multiplicación?*

- Organización
  - Cómo son implementados estos atributos (**transparente al programador**)
  - Señales de control, interfaces, tecnología de memoria
  - Ejemplo: *¿Existe una unidad de multiplicación por hardware o se realiza por sumas repetidas?*

Definiciones y conceptos básicos.

- Arquitectura y Organización
  - Toda la familia **Intel x86** comparte la misma **arquitectura**.
    - Esto brinda compatibilidad de código
  - La **organización** difiere entre las diferentes versiones (optimización para mejorar la performance)

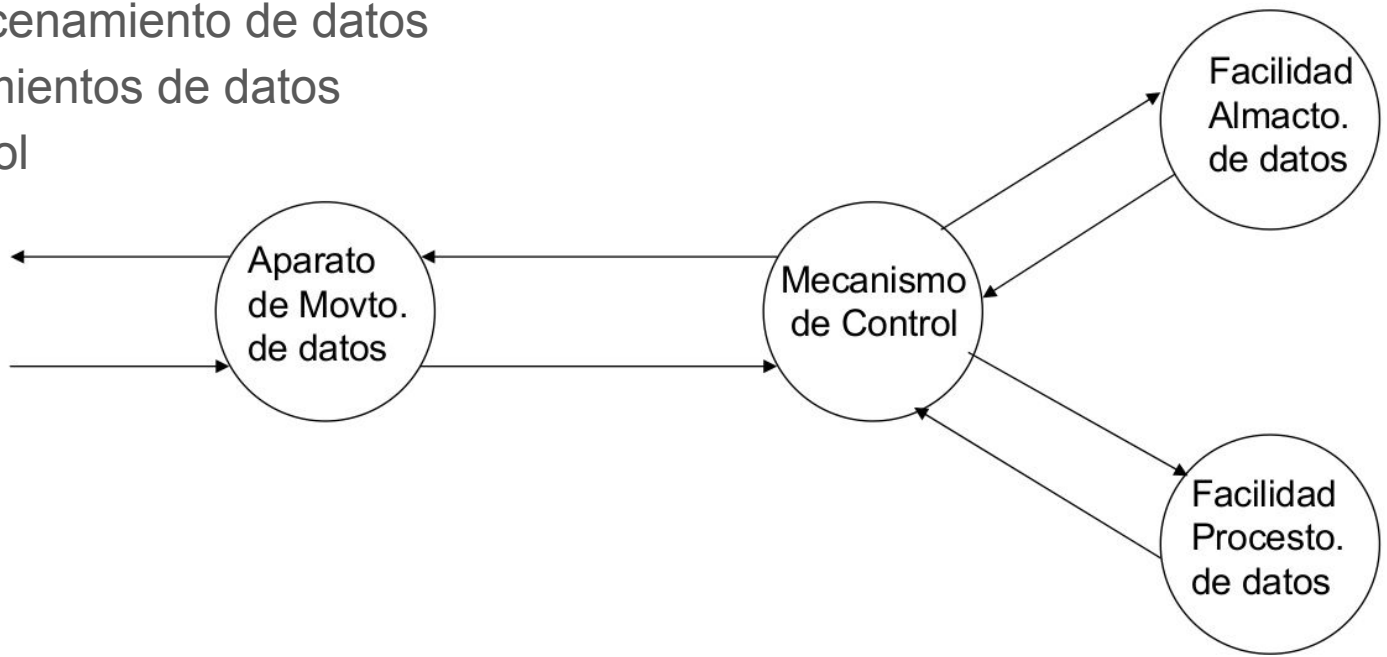


Definiciones y conceptos básicos.

- Función
  - Función es la operación de los componentes individuales como parte de la estructura
- Estructura
  - Estructura es el modo en el cual los componentes se relacionan entre sí.

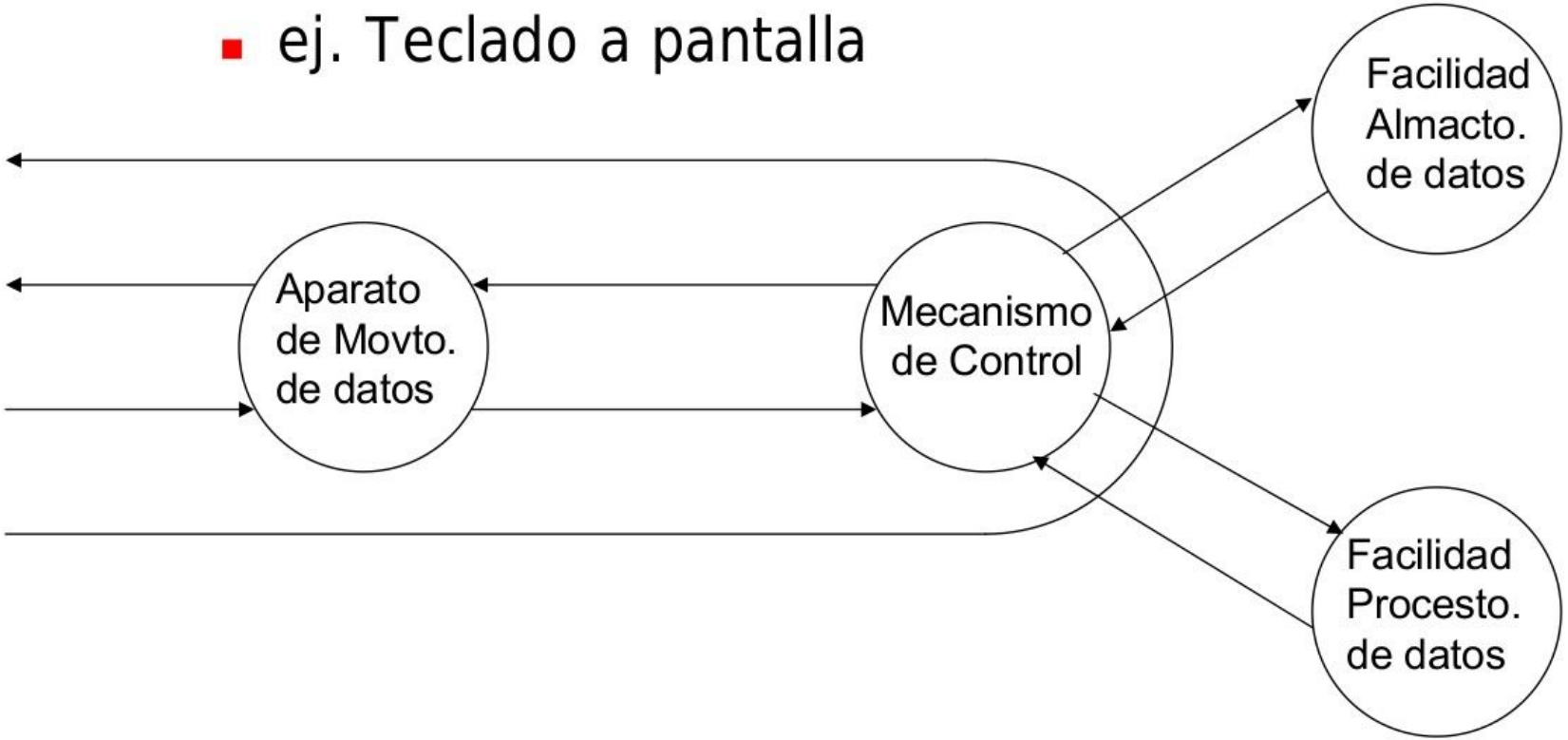
Visión funcional

- Funciones de las computadoras
  - Procesamiento de datos
  - Almacenamiento de datos
  - Movimientos de datos
  - Control



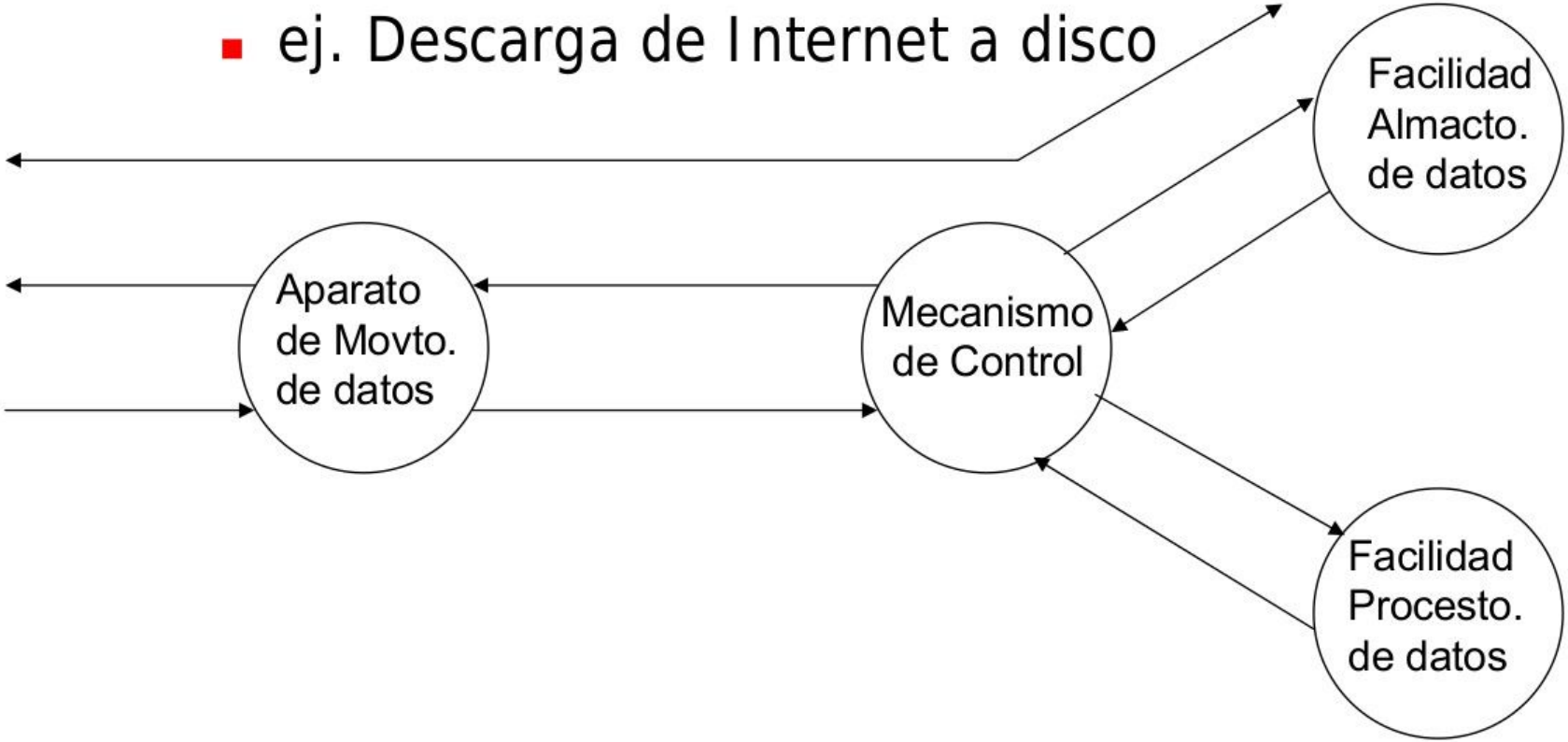
Visión funcional

- ej. Teclado a pantalla



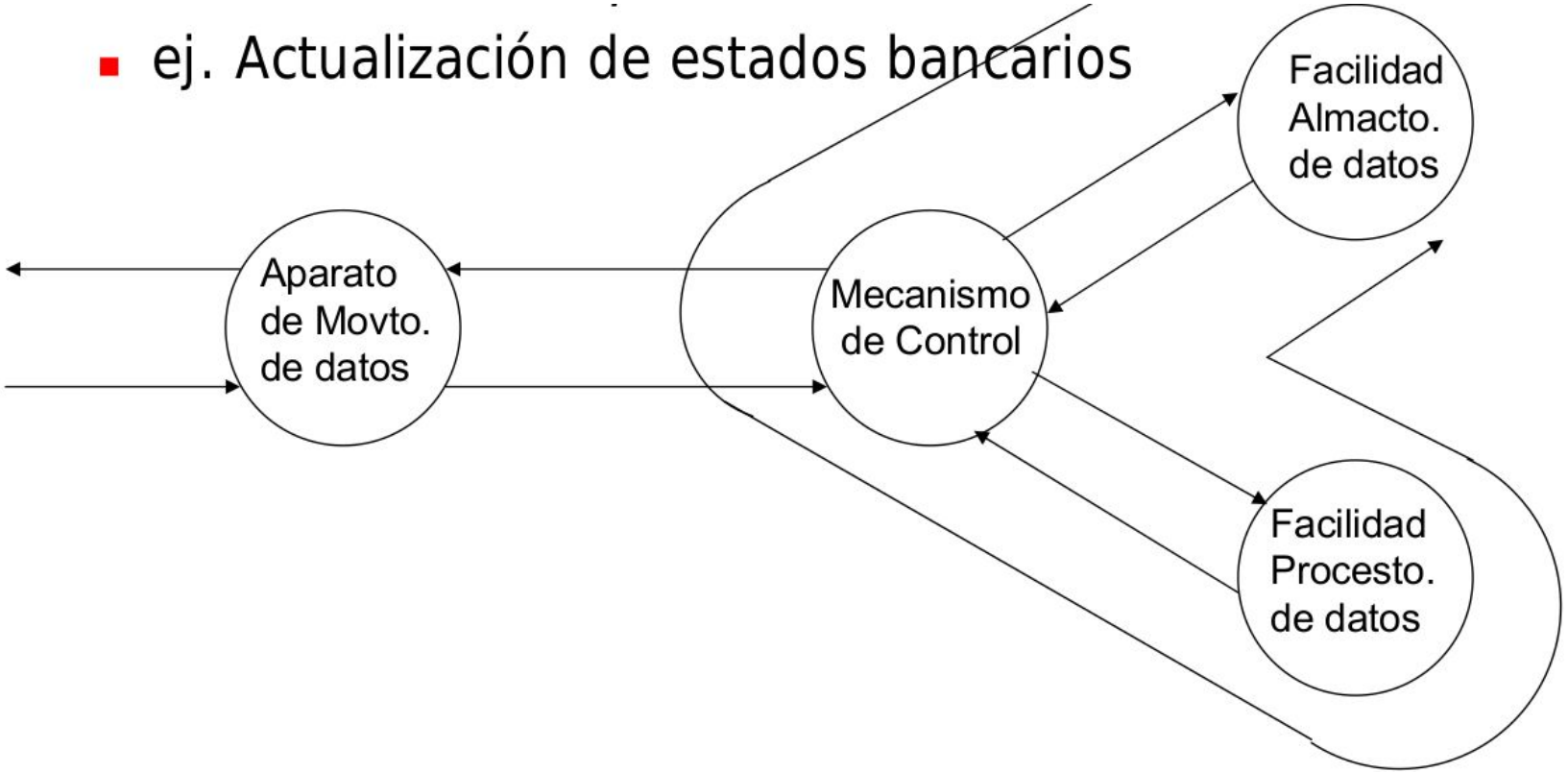
Visión funcional

- ej. Descarga de Internet a disco

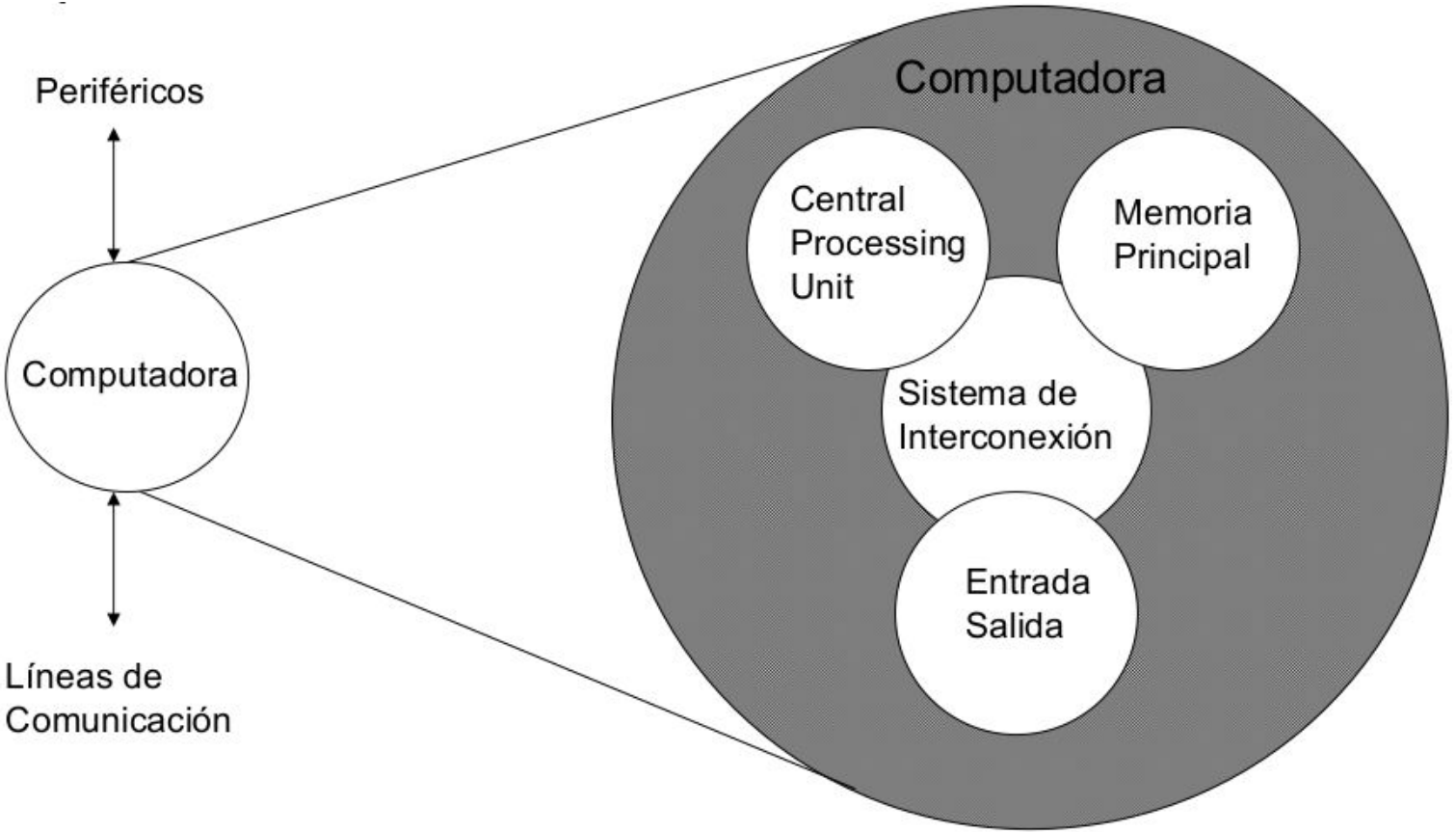


Visión funcional

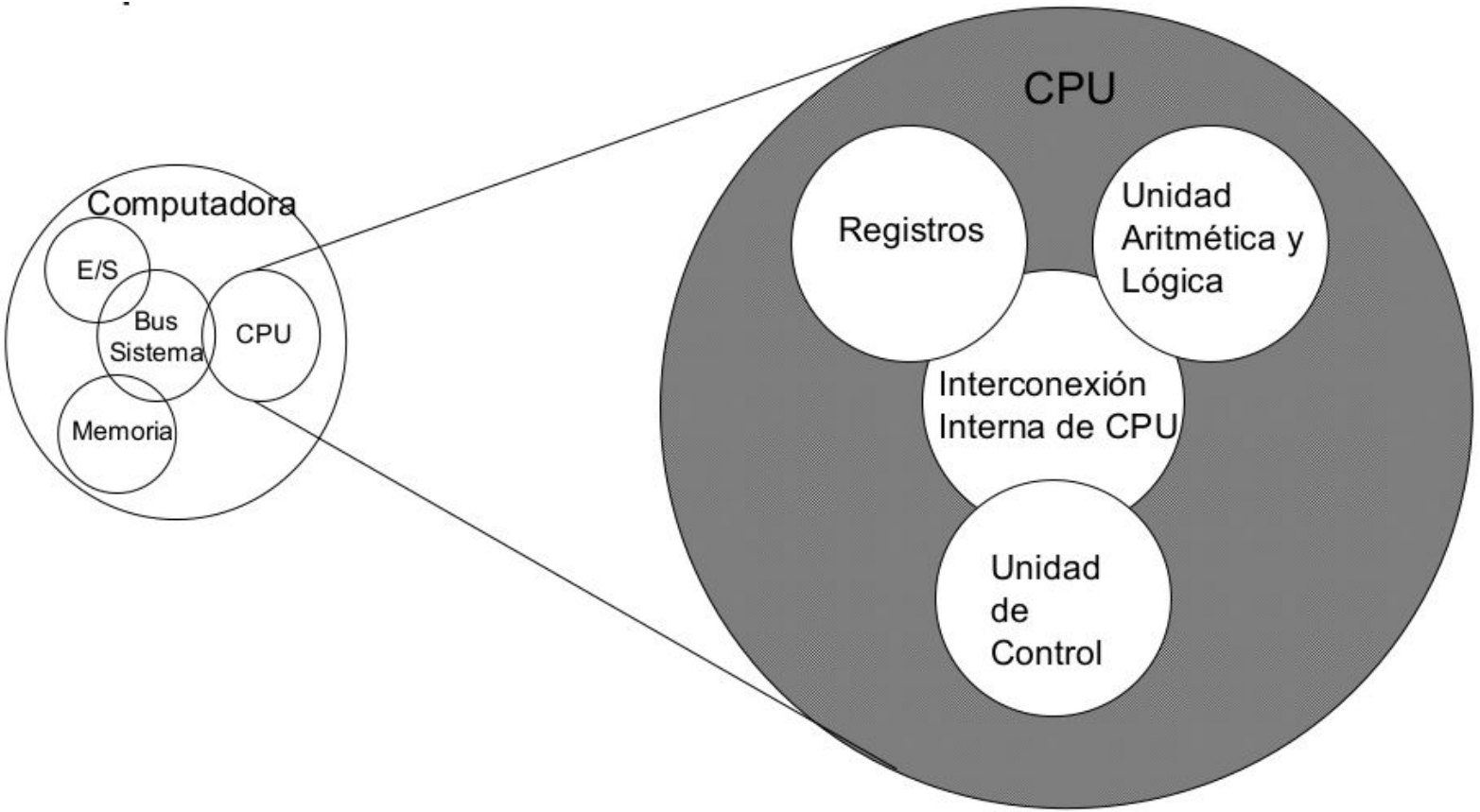
- ej. Actualización de estados bancarios



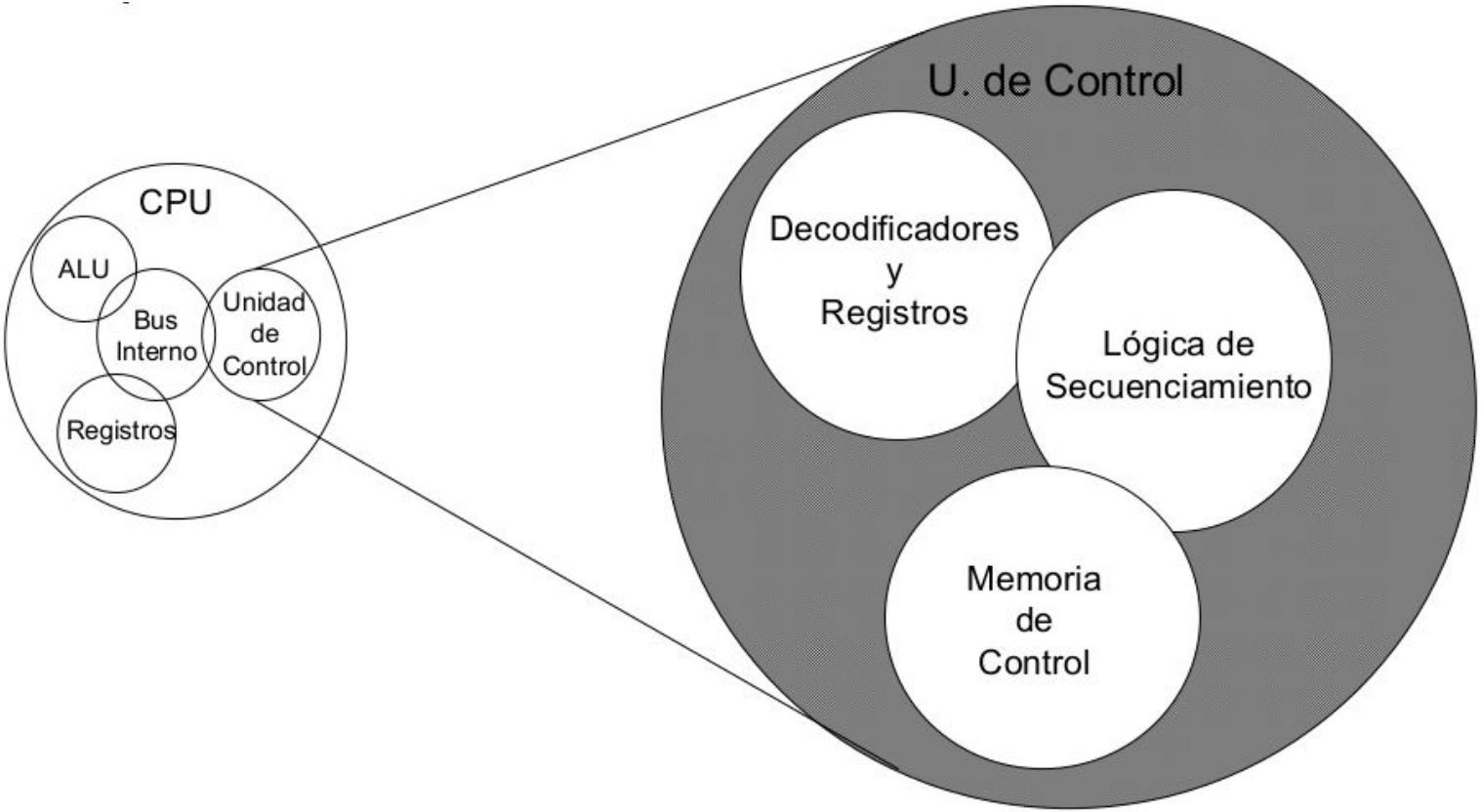
Visión estructural - Computadora



Visión estructural - CPU



Visión estructural - Unidad de control



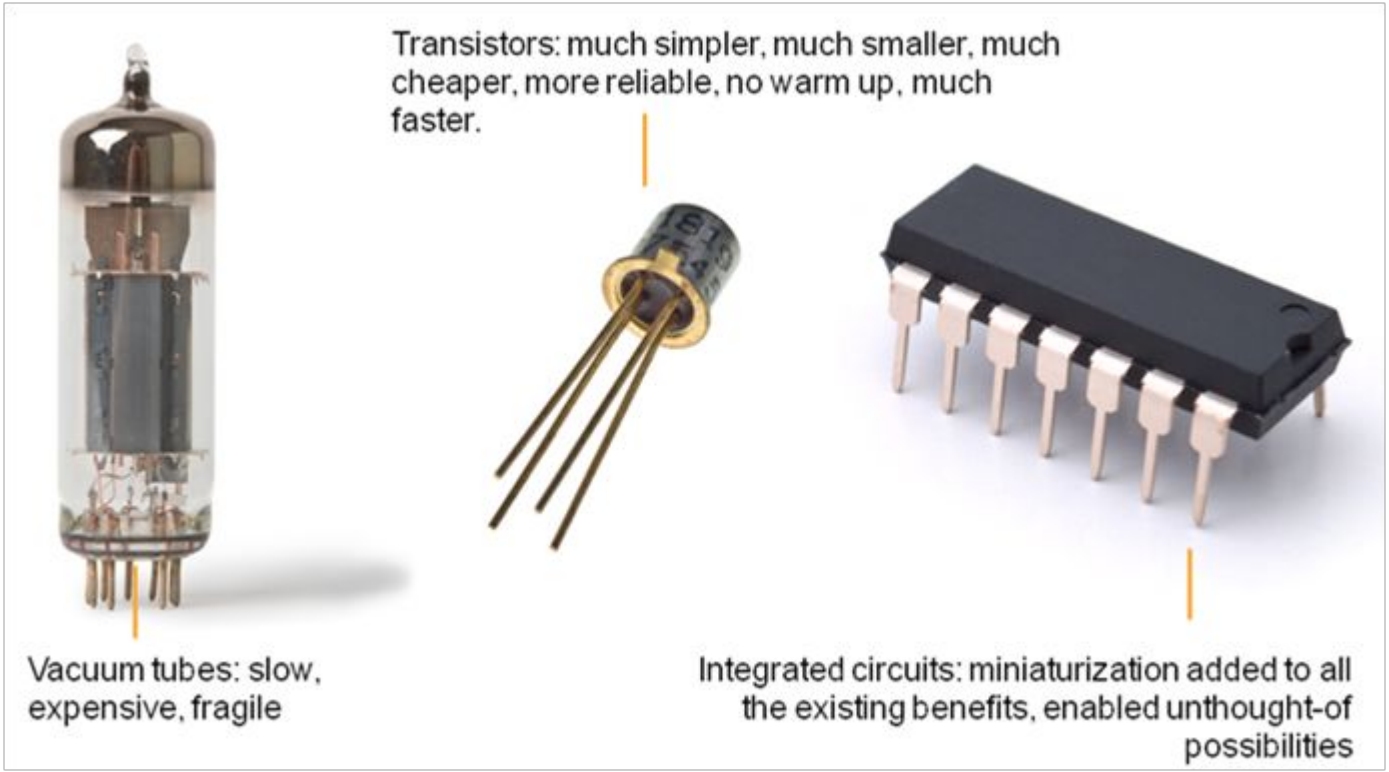
Temas de la primera parte

- Introducción y definiciones
- Evolución Histórica ←
- Arquitectura Von Neumann
- CPU
- Ciclo de instrucción
- Lógica digital

Evolución histórica. Generaciones

- Primera generación: Válvulas de vacío
  - 1940 a 1956
- Segunda generación: Transistores
  - 1956 a 1963
- Tercera generación: Circuitos integrados
  - 1964 a 1971
- Cuarta generación: Microprocesadores
  - 1972 a 2010
- Quinta generación: ¿I.A. / Multicore?
  - 2010 a ????. Varía la semántica con respecto a las generaciones anteriores

Evolución histórica. Generaciones



Evolución histórica. Primera generación.

ENIAC: Electronic Numerical Integrator and Computer

Universidad de Pennsylvania

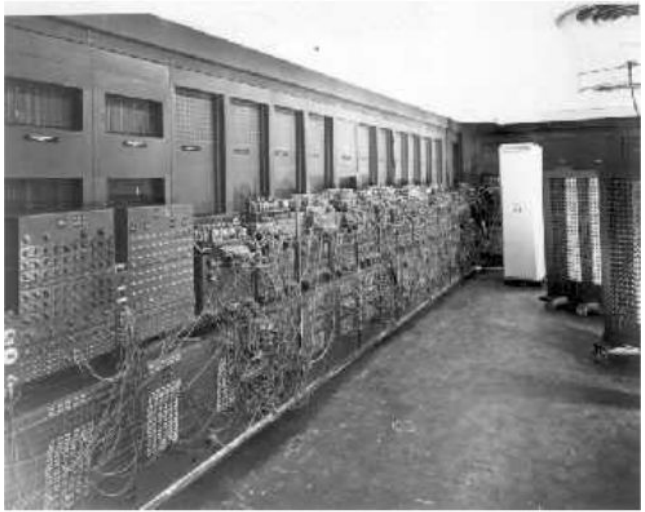
Diseñada para calcular trayectorias de proyectiles en la WWII.

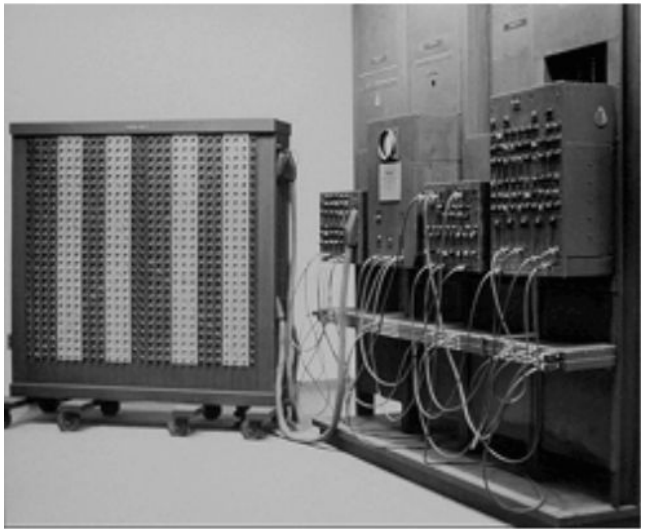
Desarrollo en 1943, finalizada en 1946 (Tarde para su uso en WWII).

Utilizada hasta el año 1955.

Evolución histórica. ENIAC.

- **Decimal**
- 20 acumuladores de 10 dígitos
- Programada manualmente por 6000 llaves
- 17468 tubos de vacío
- 32 toneladas de peso
- Ancho: 2,4 m Largo: 30 m
- 140 kWh de potencia
  - Computadora actual: 0,1 kWh
  - Aire acondicionado: 1,5 kWh
  - Microondas: 0,65 kWh
- 5000 sumas/s 360 productos/s





Evolución histórica. Primera generación.

EDVAC. Electronic Discrete Variable Automatic Computer

Universidad de Pennsylvania

Programa almacenado

Operativa desde 1949 hasta 1961

Evolución histórica. EDVAC.

- **Binaria**
- **Programa almacenado**
- 6000 válvulas
- 56 kWh de potencia.
- 45,5 m² de superficie
- 7850 kg.

Evolución histórica. Primera generación

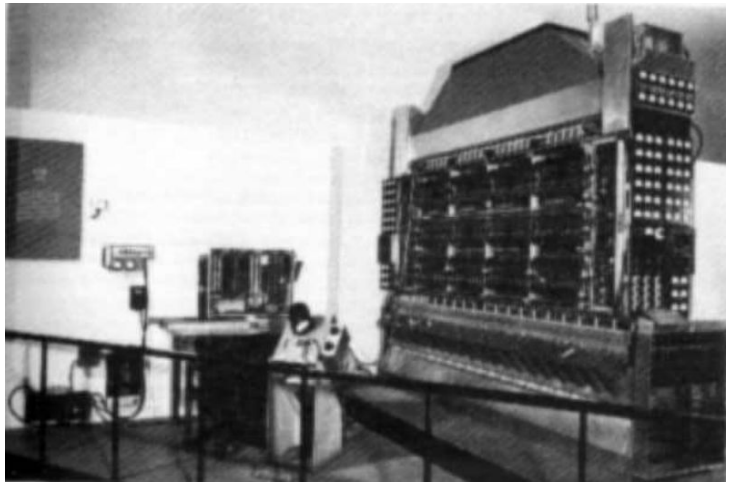
IAS. Institute for Advanced Study.

Instituto para el Estudio Avanzado (Princeton, Nueva Jersey)

Construcción desde 1942 hasta 1951

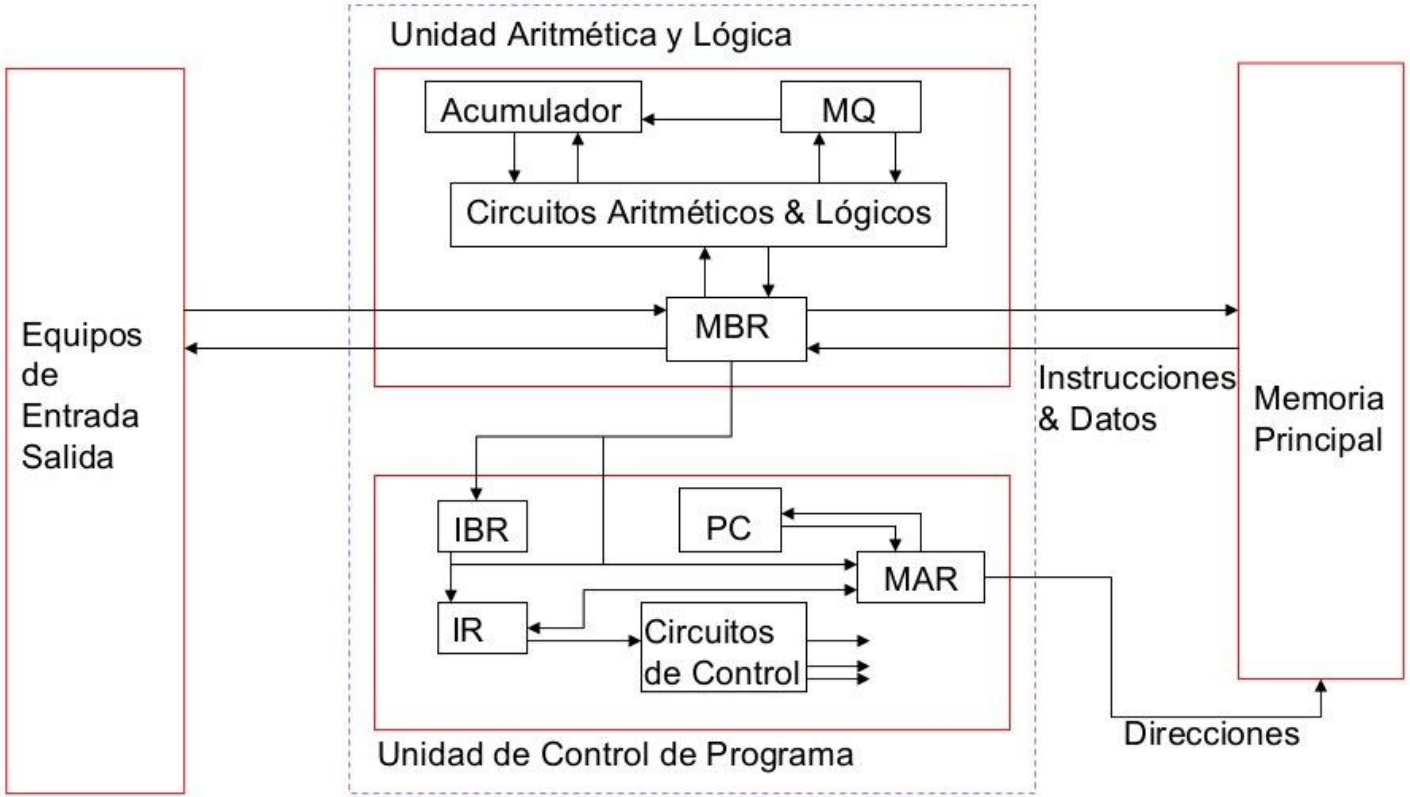
Operativa desde 1952

Evolución histórica. IAS



- **Binaria**
- 1700 tubos de vacío
- 450 Kg
- Sumas en 62 microsegundos 713 microsegundos
- Multiplicaciones en
- Memoria con 1024 palabras de 40 bits
  - Almacenamiento de 2 instrucciones de 20 bits
- Set de registros (almacenamiento en CPU)
  - Registro Buffer de Memoria (MBR)
  - Registro de Direcciones de Memoria (MAR)
  - Registros de Instrucción y Buffer de Instrucción
  - Registro Contador de Programa (Program Counter)
  - Registros Acumulador y Multiplicador/Cociente (MQ)

Evolución histórica. IAS



Evolución histórica. Primera generación.

UNIVAC. Universal Automatic Computer

Primera computadora comercial (1949)

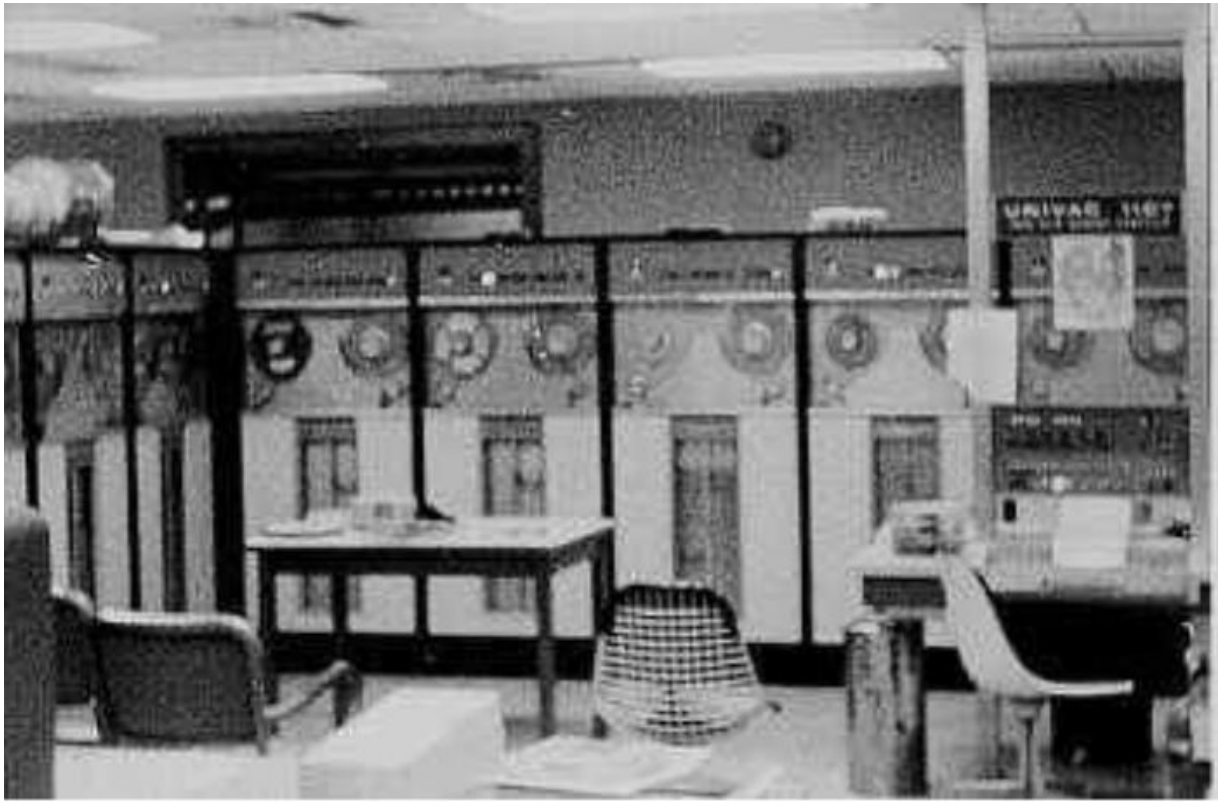
Primera en utilizar un compilador para traducir idioma de programa en idioma de máquinas.

Máquina decimal con 12 dígitos por palabra.

Principal avances: Sistema de **cintas magnéticas** que podían leerse hacia adelante y hacia atrás. Procedimientos de comprobación de errores.

Memoria de líneas de retardo de mercurio y tecnología a válvulas de vacío.

Evolución histórica. UNIVAC



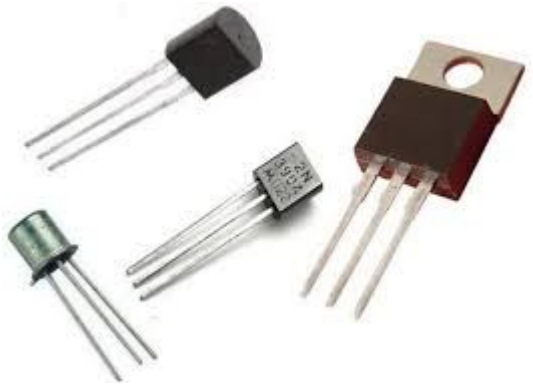
Evolución histórica. 1ra Generación

IBM. International Business Machines

- Equipos de procesamiento con **tarjetas perforadas**
- 1953: el 701
  - Primer computador con programas almacenados de IBM
  - Aplicaciones **científicas**
- 1955: el 702
  - Aplicaciones de **gestión**
- Primeros de una serie de computadores 700/7000

Evolución histórica. Segunda generación.

- Transistores
  - Sustituyen a los tubos de vacío
  - Más pequeños
  - Más baratos
  - Generan menos el calor
  - Dispositivos de estado sólido
  - Hechos con silicio
  - Inventados en 1947 en los Laboratorios Bell

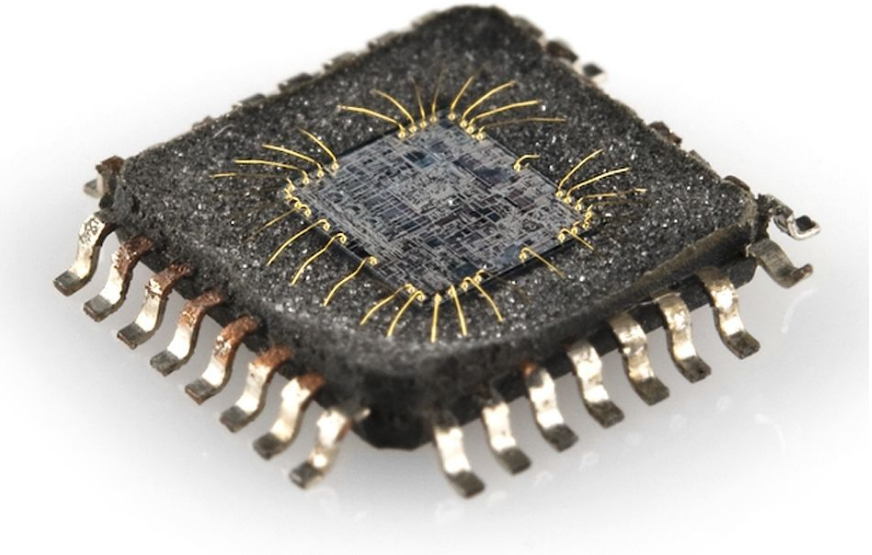


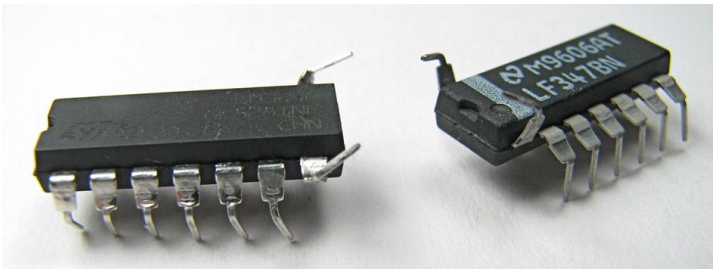
Evolución histórica. Tercera generación y siguientes

- Circuitos integrados (varios transistores en un solo chip)
  - Integración a pequeña escala (Small Scale Integration): desde 1965
    - Menos de 10 componentes en un chip
  - Integración a media escala (Medium Scale Integration): desde 1968
    - 100 a 500 componentes por chip
  - Integración a gran escala (Large Scale Integration): 1971
    - 500 a 20.000 componentes por chip
  - Integración a muy gran escala (Very Large Scale Integration): desde 1980
    - 20.000 a 1.000.000 componentes por chip
  - Integración a ultra escala (Ultra Large Scale Integration): desde 1984
    - 1.000.000 y más

Evolución histórica. Tercera generación y siguientes

- Circuitos integrados





Evolución histórica. Tercera generación y siguientes

DEC PDP-8 (1965)

Primer minicomputador

No necesita una habitación con aire acondicionado

Lo bastante pequeño para colocarlo en una mesa de laboratorio

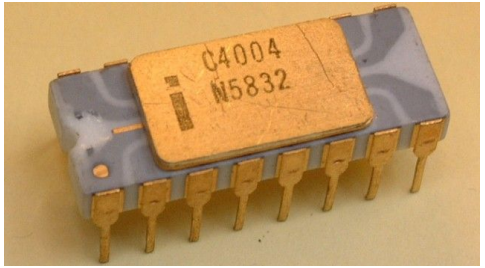
16.000 dólares (vs IBM 360: 100.000 dólares o más)

Estructura de Bus

Evolución histórica. Tercera generación y siguientes

Microprocesadores

- 1971: **Intel 4004**
  - Primer microprocesador. Capacidad de solo 4 bits
  - Todos los componentes de la CPU en un solo chip
  - 2300 transistores
  - En 1972 evoluciona al 8008 de 8 bits
  - Ambos diseñados para aplicaciones específicas
- 1974: **Intel 8080**
  - Primer microprocesador de Intel de uso genérico
- 1975: **MOS 6502**
  - CPU muy económica, utilizada en gran cantidad de consolas y computadoras personales. Y muy popular →



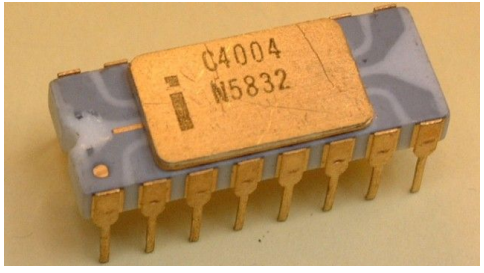
Intel 4004

MOS 6502

Evolución histórica. Tercera generación y siguientes

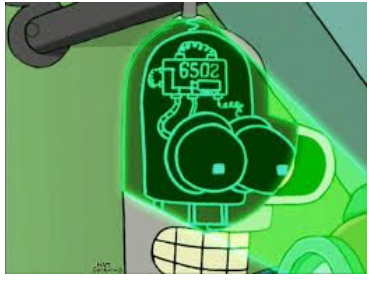
Microprocesadores

- 1971: **Intel 4004**
  - Primer microprocesador. Capacidad de solo 4 bits
  - Todos los componentes de la CPU en un solo chip
  - 2300 transistores
  - En 1972 evoluciona al 8008 de 8 bits
  - Ambos diseñados para aplicaciones específicas
- 1974: **Intel 8080**
  - Primer microprocesador de Intel de uso genérico
- 1975: **MOS 6502**
  - CPU muy económica, utilizada en gran cantidad de consolas y computadoras personales. Y muy popular →



Intel 4004

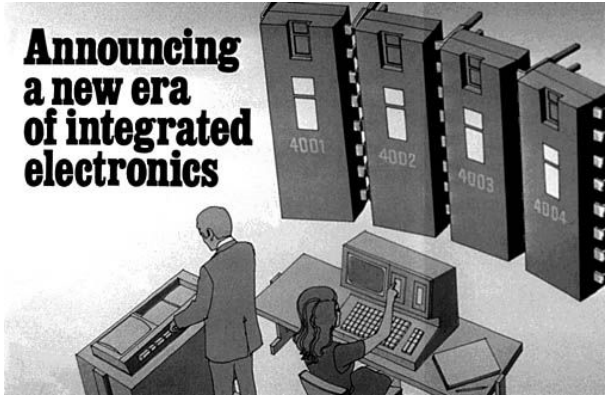
MOS 6502

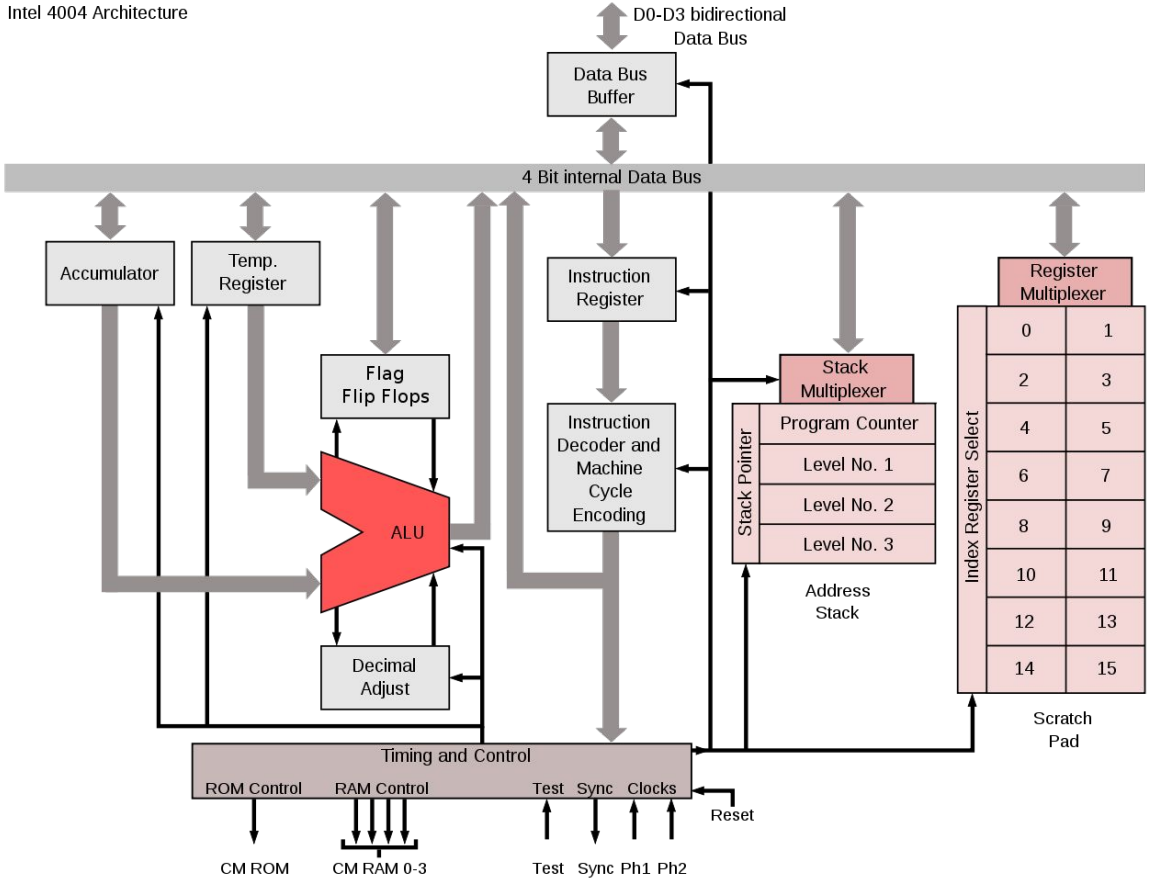


Evolución histórica. Tercera generación y siguientes

Microarquitectura

Intel 4004





Evolución histórica. Tercera generación y siguientes

- Microprocesadores. El inicio de la computación personal.

Commodore 64

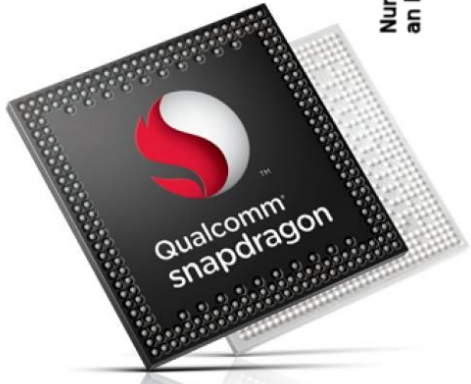
ZX Spectrum

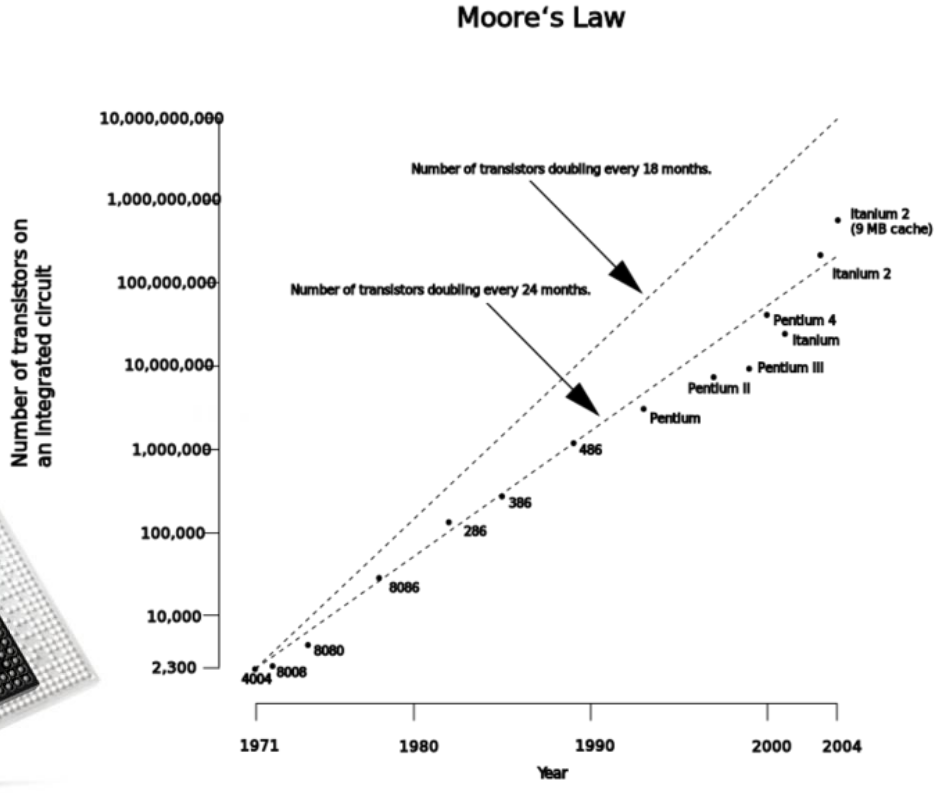
Apple II

Evolución histórica. Tercera generación y siguientes

- Microprocesadores

Actualmente, más de 1.000.000.000 de transistores en un solo chip

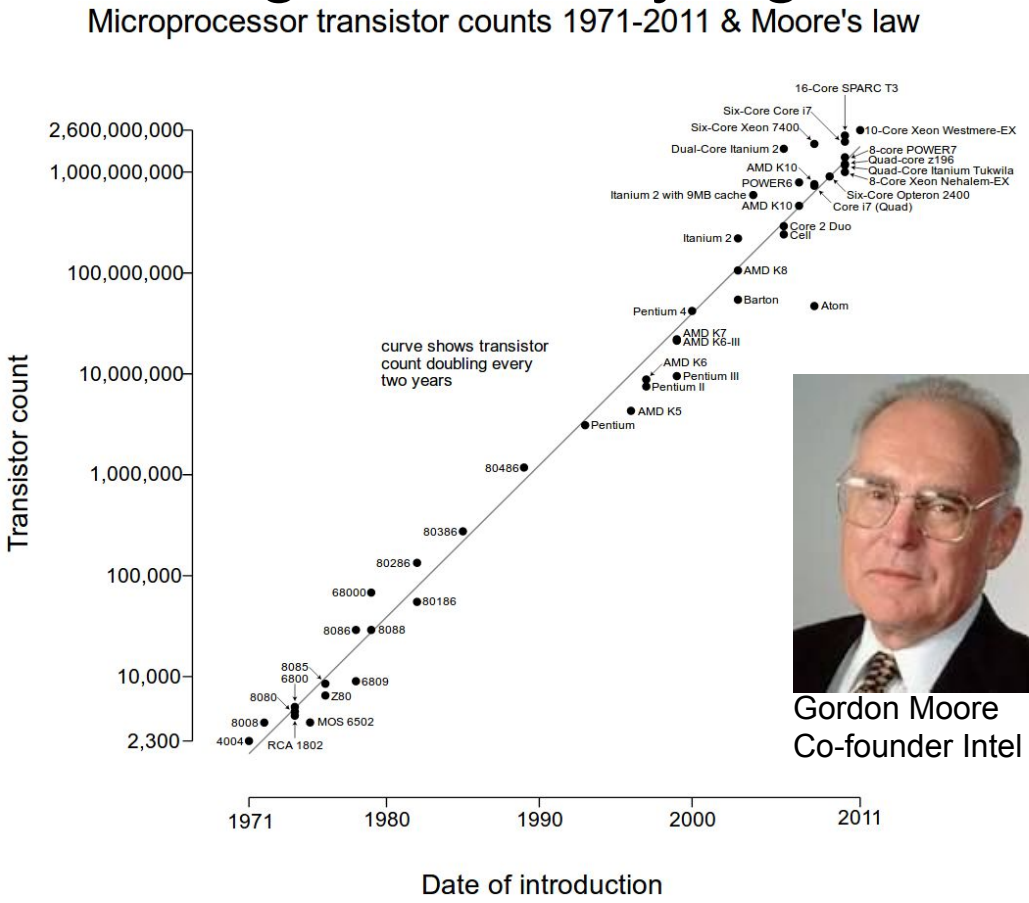




Evolución histórica. Tercera generación y siguientes

"Ley" de Moore (1965):

"El número de transistores en un microprocesador se duplica cada dos años"



Gordon Moore
Co-founder Intel Co.

Evolución histórica. Tercera generación y siguientes

¿Se acerca el fin de “ley” de Moore?

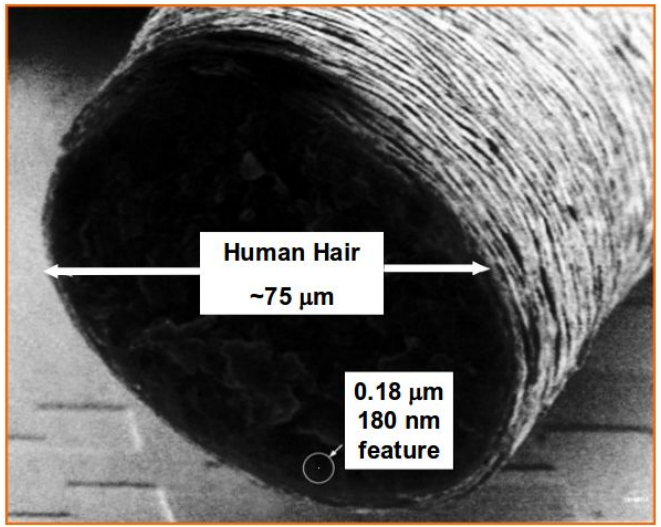
Tecnología actual: menos de **10 nm** (millonésima parte de un mm)!. Comparar con:

- Tamaño promedio de un átomo: 0,32nm
- Grosor promedio de un pelo: 75.000nm

Condicionantes:

- Limitaciones en las técnicas de fotolitografía
- Electrones pueden “saltar” (quantum tunneling)

¿Cuánto más se puede reducir?



10.000 nm – 1971
3.000 nm – 1975
1.500 nm – 1982
1.000 nm – 1985
800 nm – 1989
600 nm – 1994
350 nm – 1995
250 nm – 1997
180 nm – 1999
130 nm – 2002
90 nm – 2004
65 nm – 2006
45 nm – 2008
32 nm – 2010
22 nm – 2012
14 nm – 2014
10 nm – 2016
7 nm – 2018
5 nm – 2020

Temas de la primera parte

- Introducción y definiciones
- Evolución Histórica
- Arquitectura Von Neumann ←
- CPU
- Ciclo de instrucción
- Lógica digital

Arquitectura Von Neumann

John von Neumann (1903-1957)

Matemático nacido en Hungría

Contribuciones en varias ramas de la ciencia: ciencias de la computación, física cuántica, hidrodinámica, teoría de juegos, análisis funcional, etc.

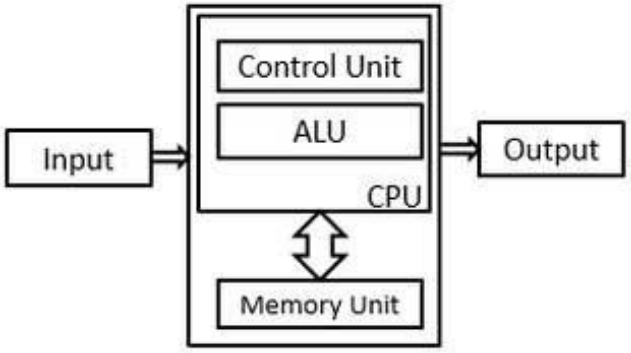
Participó también del Proyecto Manhattan.

“It would appear that we have reached the limits of what it is possible to achieve with computer technology, although one should be careful with such statements, as they tend to sound pretty silly in 5 years.”

Arquitectura Von Neumann

5 componentes principales:

- **Unidad de entrada**: provee las instrucciones y los datos
- **Unidad de memoria**: donde se almacenan datos e instrucciones
- **Unidad aritmético-lógica**: procesa los datos
- **Unidad de control**: dirige la operación
- **Unidad de salida**: se envían los resultados



Von Neumann Model

Arquitectura Von Neumann

Aspectos más importantes.

- Utilización del **sistema binario**
  - Simplifica la implementación de funciones.
  - Disminuye la probabilidad de fallos.
- Instrucciones y datos residen en memoria:
  - Ejecución del **programa** en forma **secuencial**.
  - Aumenta la velocidad.
- La memoria es direccionable por localidad **sin importar el dato almacenado**

Arquitectura VN. Programa almacenado.

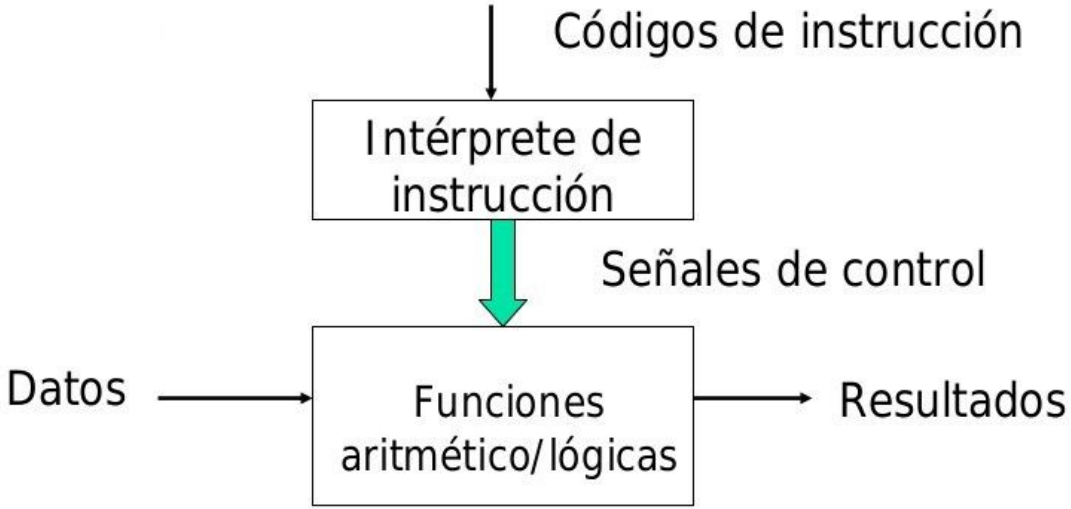
Antes

Programación en hardware: cuando cambiamos las tareas, debemos cambiar el hardware

Arquitectura VN. Programa almacenado.

Ahora

Programación en software: en cada paso se efectúa alguna operación sobre los datos



Arquitectura VN. Programa almacenado.

Programa

- Para cada paso se necesita un nuevo conjunto de señales de control.
- Las instrucciones proporcionan esas señales de
- control.
- No hay que cambiar el hardware.
- Aparece el nuevo concepto de **programación**:
  - Secuencia de pasos.
  - Se hace una operación aritmético/lógica por cada paso.
  - Diferentes señales de control se necesitan para cada operación.
    - La Unidad de Control obtiene la información de cada instrucción

Arquitectura V.Neumann vs Arquitectura Harvard

Cuello de botella Von Neumann

- Canal de transmisión de los datos entre CPU y memoria (buses) es **compartido**.

- CPU ***no puede leer instrucciones y datos al mismo tiempo***, ya que instrucciones y datos usan el mismo sistema de comunicación.

- Velocidad de comunicación entre memoria y CPU es más baja que velocidad de procesamiento de la CPU → ***reduce considerablemente el rendimiento*** del equipo.

- Mejoras?
  - Memorias adicionales (por ejemplo, memoria **cache** entre memoria principal y CPU)
  - Comunicaciones independientes para datos e instrucciones (**Harvard Architecture**)
  - Otros (por ejemplo, predicción de saltos)

Arquitectura V.Neumann vs Arquitectura Harvard

Arquitectura Harvard

La CPU puede tanto ***leer una instrucción*** como realizar un ***acceso a la memoria de datos al mismo tiempo***.

Las características de la ***memoria de datos*** puede ***diferir*** con respecto de las características de la ***memoria de instrucciones***

Arquitectura Harvard modificada

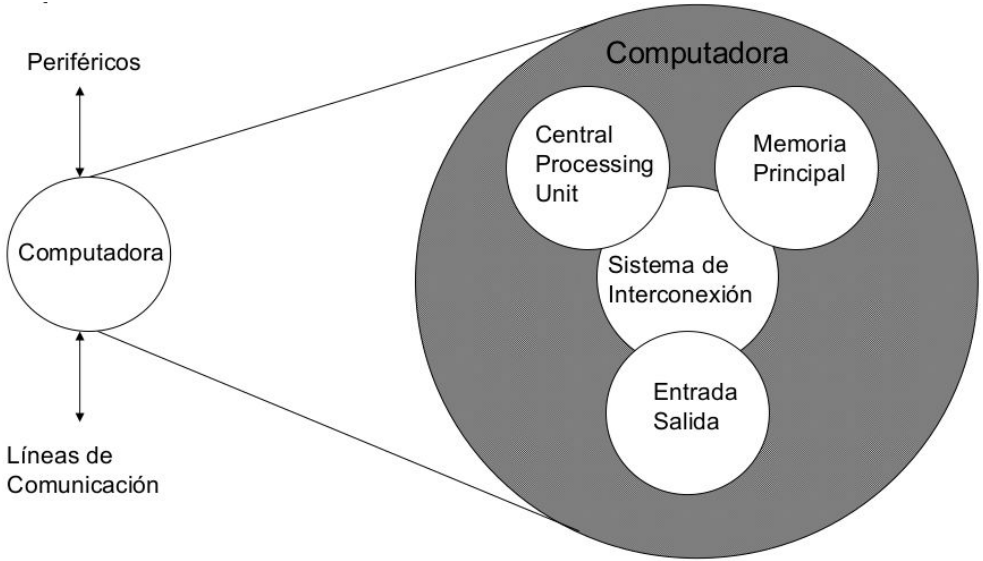
Actualmente, la mayoría de los procesadores implementan ***vías de comunicación independientes*** a fin de optimizar la performance, bajo una arquitectura llamada ***Harvard modificada***, la cual flexibiliza la separación entre instrucciones y datos.

Temas de la primera parte

- Introducción y definiciones
- Evolución Histórica
- Arquitectura Von Neumann
- CPU ←
- Ciclo de instrucción
- Lógica digital

CPU

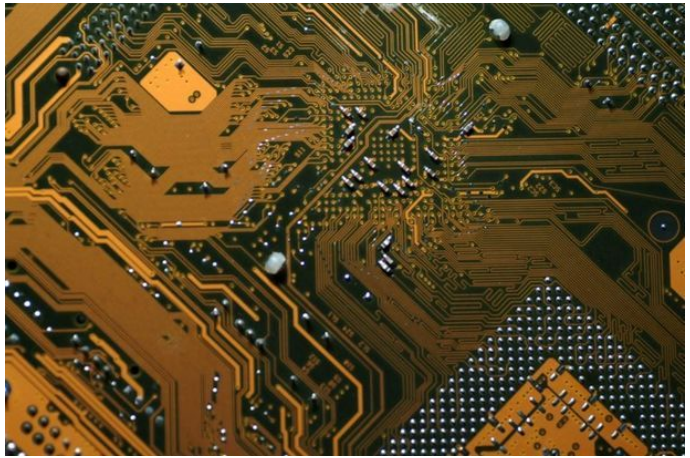
- Un sistema de cómputo está constituido por 3 subsistemas:
  - CPU
  - Memoria
  - E/S
- Los componentes deben poder comunicarse entre sí → Sistema de Interconexión



¿Cómo intercambia información la CPU?

CPU y buses

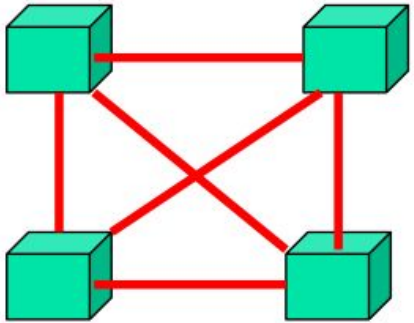
¿Qué es un bus?



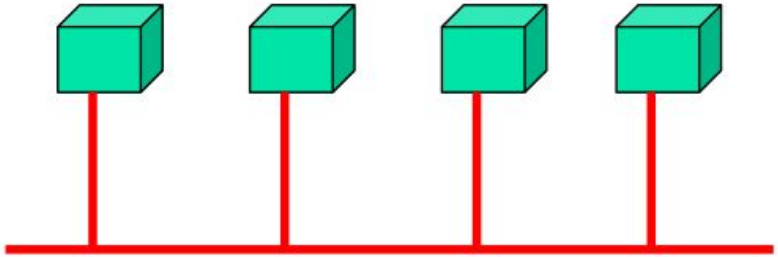
- Un camino de comunicación que conecta dos o más dispositivos.

- Agrupados en un número de canales por bus
  - Bus de 32 bits son 32 canales separados de un solo bit cada uno.

CPU y buses

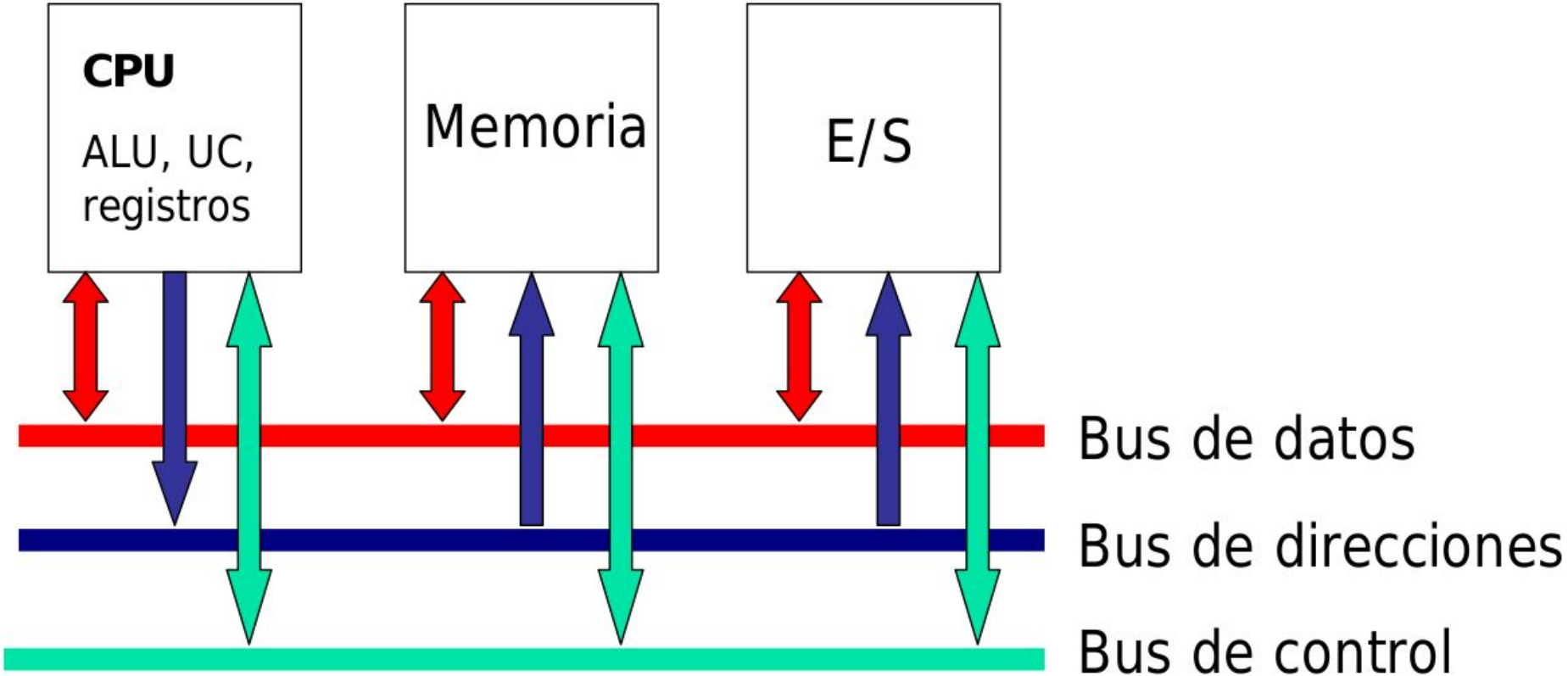


Conexiones independientes entre los distintos dispositivos



Conexiones a través de un medio compartido

CPU y buses


CPU
ALU, UC,
registros
Memoria
E/S
Bus de datos
Bus de direcciones
Bus de control

CPU y buses

Si el bus es compartido por diferentes elementos, éstos deben tener identidades distintivas: **direcciones**.

La **dirección de memoria** identifica una celda de memoria en la que se almacena información.

CPU y buses

Bus de direcciones

- Identifica el origen o el destino de los datos
  - La CPU necesita leer/escribir desde/hacia una ubicación en particular de la memoria
- El ancho del bus determina la máxima capacidad de memoria del sistema
  - ej. 8080 tiene un bus de direcciones de 16 bits dando un espacio de direcciones de 64k (2^16=65536 direcciones)

CPU y buses

Bus de datos

- Transporta datos
  - No hay diferencia entre “dato” e “instrucción” en éste nivel.
- El ‘ancho’ es un valor determinante de las prestaciones.
  - 8, 16, 32, 64 bits
  - A mayor número de bits, más información a la vez puede viajar a lo largo del bus

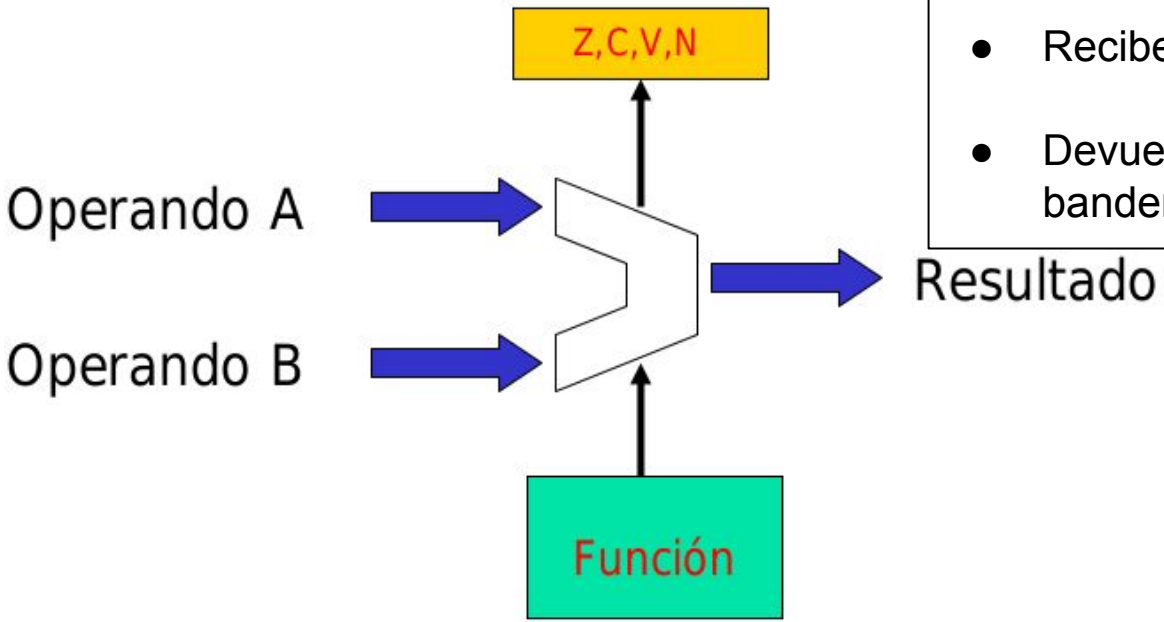
CPU y buses

Bus de control

- Información de control y temporizado
- Señales de lectura/escritura de Memoria o E/S
- Señales de selección o habilitación
- Señales de Reloj (Clock)
- Señales de pedido de Interrupción

CPU y ALU

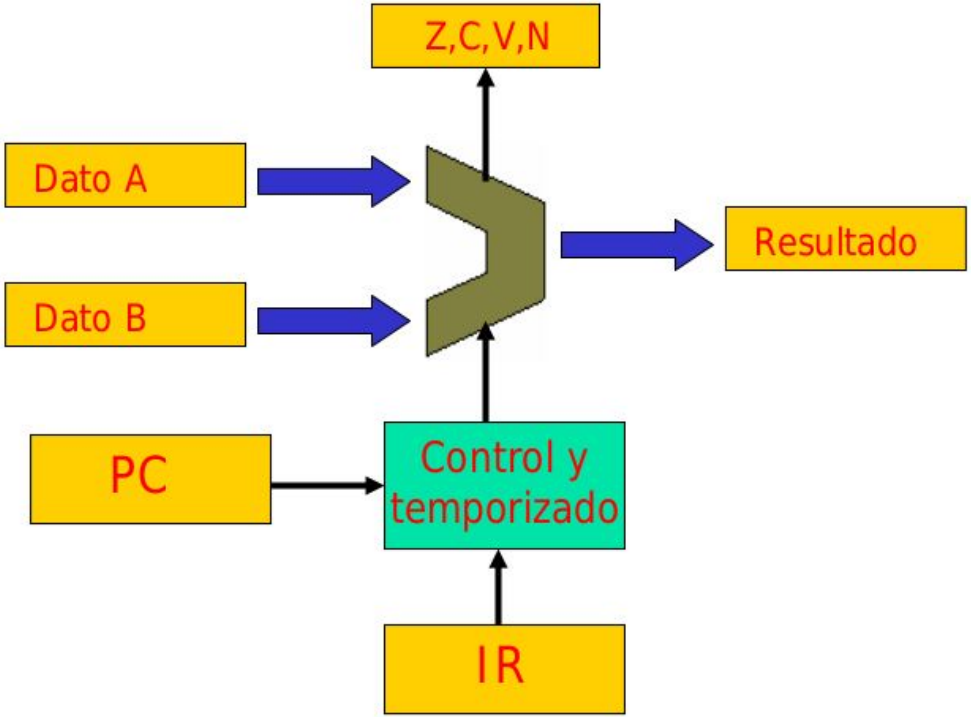
ALU: Arithmetic Logic Unit



- Realiza las operaciones lógicas y matemáticas a partir de las instrucciones definidas en el programa en ejecución.
- Recibe los operandos a procesar
- Devuelve tanto resultados como banderas de condición (flags).

CPU y ALU

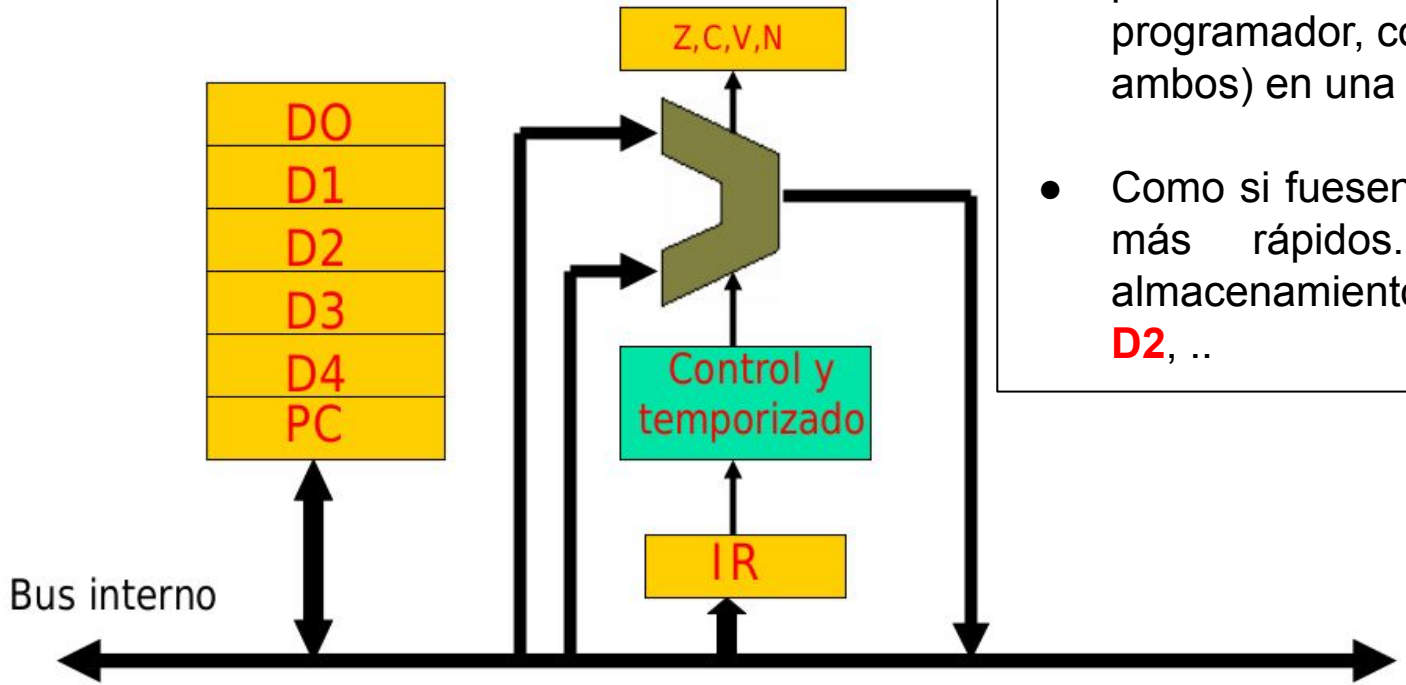
ALU: Arithmetic Logic Unit



- La instrucción se almacena en un registro de la CPU llamado **IR**.
- El bloque control lee **IR** para saber qué hacer, dónde están los operandos y dónde poner el resultado.
- ¿Cómo sabe la CPU dónde encontrar la próxima instrucción? → Hay un registro llamado **PC** (Program Counter).
- Inicialmente, el **PC** contiene la dirección de la primera instrucción, y luego es incrementado.
- El PC también es conocido como **IP** (Instruction Pointer).

CPU y ALU

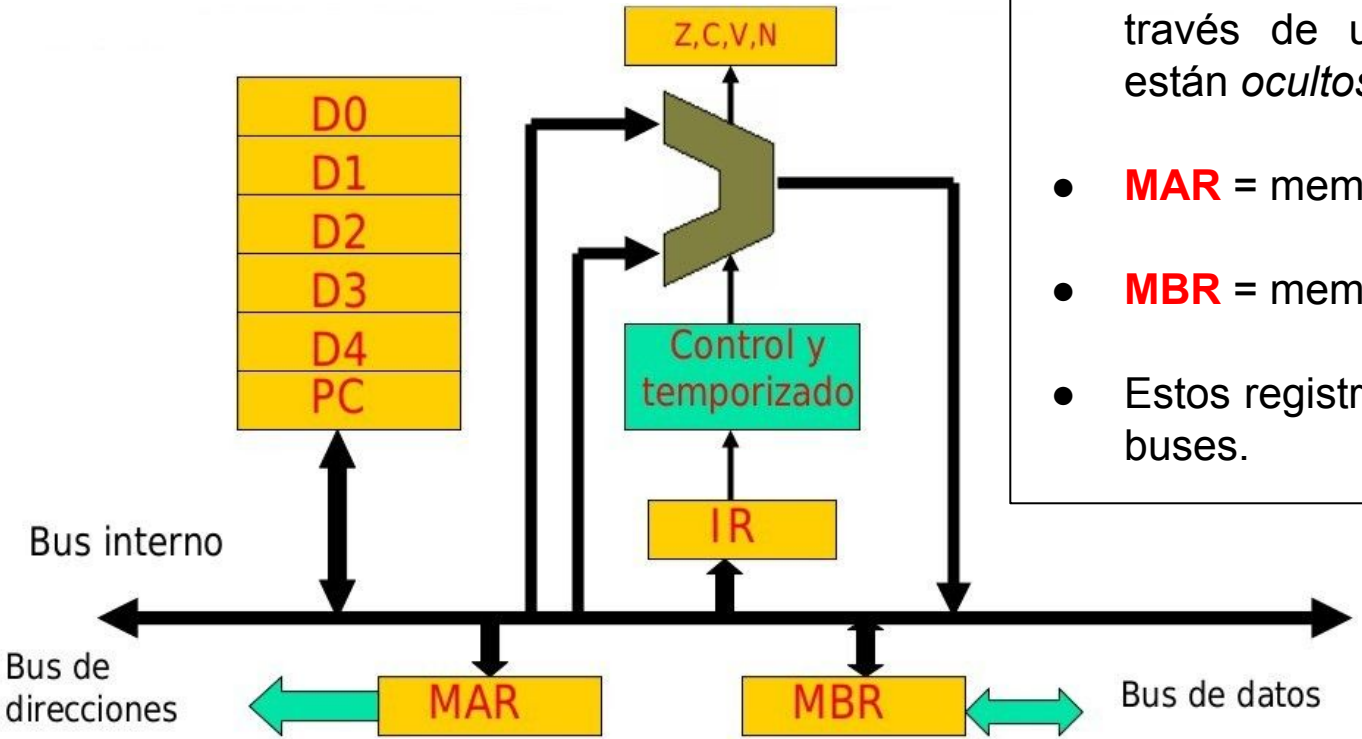
ALU: Arithmetic Logic Unit



- Todas las CPU tienen registros internos de propósito general que pueden ser referenciados por el programador, como fuente o destino (ó ambos) en una instrucción.
- Como si fuesen memoria, pero mucho más rápidos. Son lugares de almacenamiento temporario: **D0**, **D1**, **D2**, ..

CPU y ALU

ALU: Arithmetic Logic Unit



- La CPU interactúa con la memoria a través de un par de registros que están *ocultos* al programador.
- **MAR** = memory address register
- **MBR** = memory buffer register
- Estos registros están conectados a los buses.

Temas de la primera parte

- Introducción y definiciones
- Evolución Histórica
- Arquitectura Von Neumann
- CPU
- Ciclo de instrucción ←
- Lógica digital

Ciclo de Instrucción

Ejecución de programas

- Un programa está compuesto de **instrucciones** almacenadas en memoria
- La CPU **procesa** las instrucciones
  - Debe traerlas desde memoria **una por vez**
  - Debe **ejecutar** cada operación ordenada

Ciclo de Instrucción

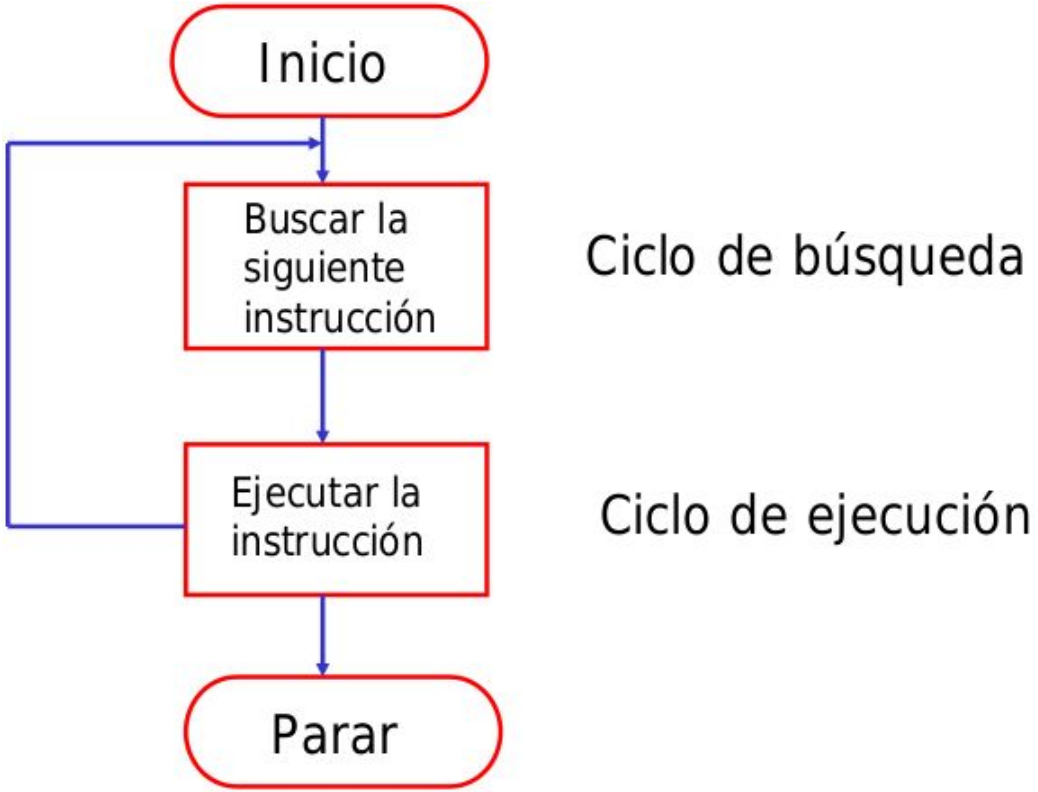
Ejecución de programas

- Podemos descomponer el procesamiento de instrucciones en dos etapas:
  - **Búsqueda**: leer desde memoria
    - Común a todas las instrucciones.
  - **Ejecución**: dependiendo de la instrucción
    - Puede implicar varias operaciones

Ciclo de Instrucción

- El procesamiento requerido para una sola instrucción se llama **ciclo de instrucción**.
- Dos etapas: **ciclo de búsqueda** y **ciclo de ejecución**.
- La ejecución del programa se interrumpe sólo si la máquina se **apaga**, hay un **error** ó una instrucción que **interrumpa** a la computadora.

Ciclo de Instrucción



Ciclo de Instrucción

- Al principio de cada ciclo, la CPU busca una instrucción en memoria.
- El contador de programa (**PC**) almacena la dirección de la próxima instrucción a buscar.
- La CPU, después de buscar cada instrucción, incrementa el valor contenido en **PC**; así podrá buscar la siguiente instrucción en secuencia.

Ciclo de Instrucción

- La instrucción buscada se carga dentro del registro **IR** (Instruction Register) de la CPU.
- La instrucción está en la forma de un código binario que especifica las acciones que tomará la CPU.
- La CPU interpreta cada instrucción y lleva a cabo las acciones requeridas.

Ciclo de Instrucción

- En general las acciones caen en 4 tipos
  - **CPU – Memoria**
    - datos pueden transferirse entre memoria y CPU
  - **CPU – E/S**
    - datos pueden transferirse entre CPU y entrada/salida.
  - **Procesamiento de datos**
    - CPU efectúa operaciones aritméticas o lógicas en datos.
  - **Control**
    - alterar la secuencia de ejecución de instrucciones.

Ciclo de Instrucción. Ejemplo

El siguiente ejemplo muestra la ejecución de un fragmento de programa que tiene tres instrucciones:

- Cargar en el registro D el contenido de la posición de memoria **940H**
- Sumar el contenido de la posición de memoria **941H** al registro D y guardar el resultado en D
- Almacenar el valor del registro D en la posición de memoria **941H**

Ciclo de Instrucción. Ejemplo

- Cargar en el registro D el contenido de la posición de memoria **940H**
- Sumar contenido de la posición de memoria **941H** al registro D y guardar resultado en D
- Almacenar el valor del registro D en la posición de memoria **941H**

- [CPU Reg. D] ← [Memoria 940H]
- [CPU Reg. D] ← [Memoria 941H] + [CPU Reg. D]
- [Memoria 941H] ← [CPU Reg. D]

- Similar a: **[941H] ← [940H] + [941H]**
- Similar a: B = A + B
  - Donde **A** es una variable almacenada en la celda 940H y **B** en 941H

Ciclo de Instrucción. Ejemplo

Conjunto de instrucciones disponibles en la CPU:

- CodOp **1** = 0001b = cargar D desde la memoria
- CodOp **2** = 0010b = almacenar D en memoria
- CodOp **5** = 0101b = sumar D con un dato en memoria

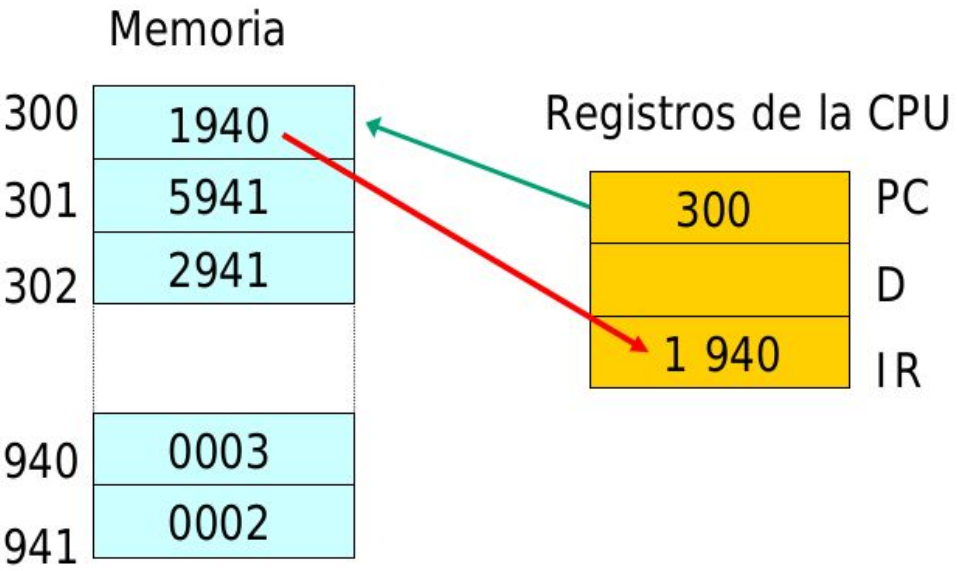
- Consideremos que cada posición de memoria almacena 16 bits.
  - Los primeros 4 bits indican la operación a realizar
  - Los siguientes 12 bits indican una dirección de memoria.

En binario: $b_{15}b_{14}b_{13}b_{12}b_{11}b_{10}b_9b_8b_7b_6b_5b_4b_3b_2b_1b_0\ B$

En hexadecimal: $h_3h_1h_2h_0\ H$

Ciclo de Instrucción. Ejemplo

CodOp 1 = cargar D desde la memoria
CodOp 2 = almacenar D en memoria
CodOp 5 = sumar D con un dato en memoria

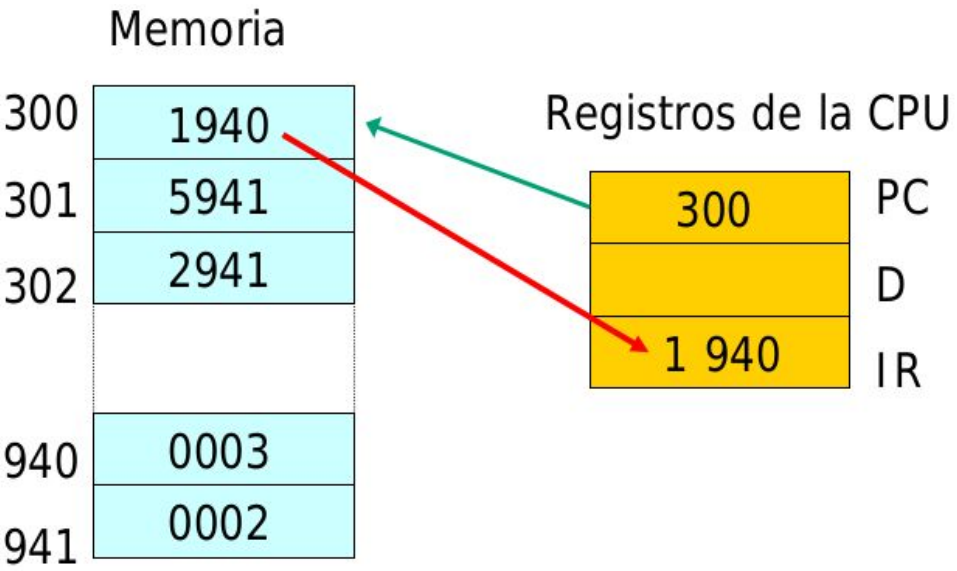


En memoria:

- Segmento datos:
  - 2 datos, en celdas **940H** y **941H**
- Segmento código:
  - 3 instrucciones (expresadas en hexadecimal) a partir de la celda **300H**

Ciclo de Instrucción. Ejemplo

CodOp 1 = cargar D desde la memoria
CodOp 2 = almacenar D en memoria
CodOp 5 = sumar D con un dato en memoria

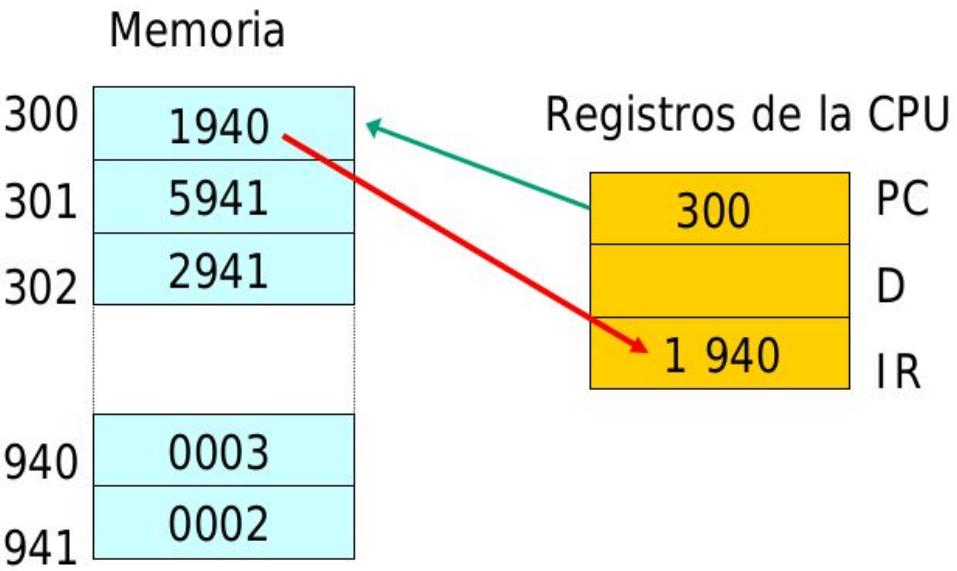


El contador de programa (**PC**) contiene **300H** como la dirección de la primera instrucción.

El contenido de esta dirección se carga en el registro de instrucción (**IR**)

Ciclo de Instrucción. Ejemplo

CodOp 1 = cargar D desde la memoria
CodOp 2 = almacenar D en memoria
CodOp 5 = sumar D con un dato en memoria



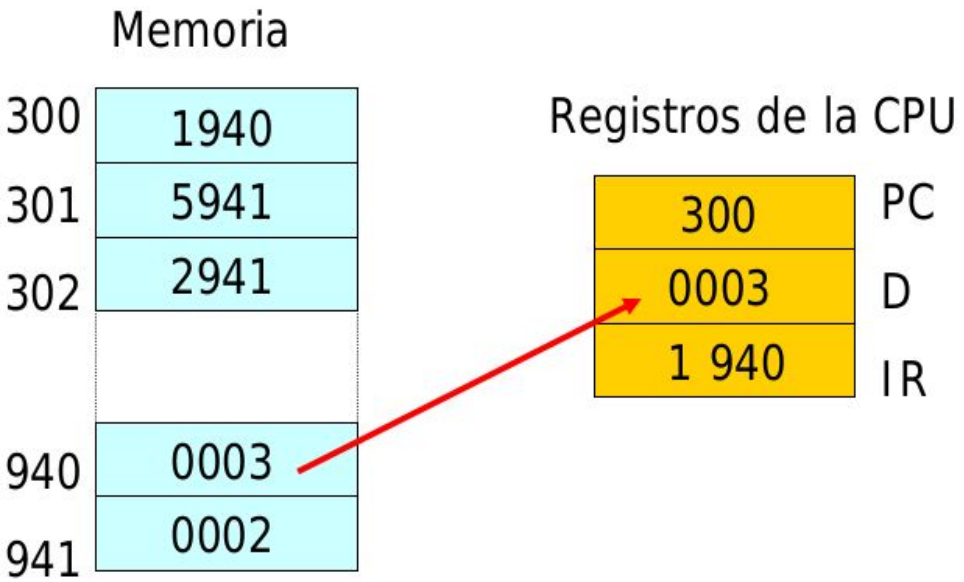
Instrucción **1940H**: Los primeros 4 bits en **IR** indican CodOp **1**.

El registro D se cargará con un dato proveniente de la dirección especificada en los restantes 12 bits de la instrucción.

En este caso la dirección es **940H**.

Ciclo de Instrucción. Ejemplo

CodOp 1 = cargar D desde la memoria
CodOp 2 = almacenar D en memoria
CodOp 5 = sumar D con un dato en memoria



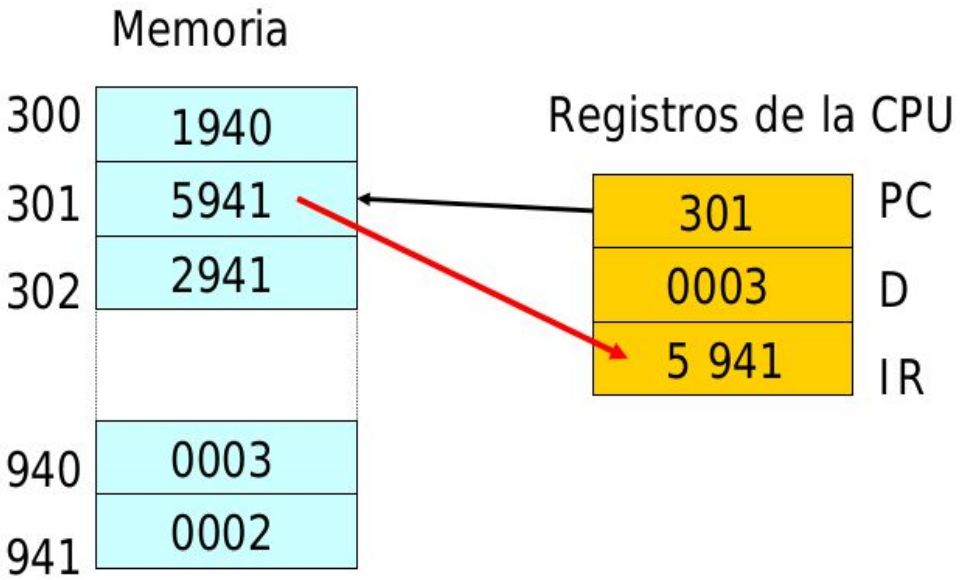
Instrucción **1940H**: Los primeros 4 bits en **IR** indican CodOp **1**.

El registro D se cargará con un dato proveniente de la dirección especificada en los restantes 12 bits de la instrucción.

En este caso la dirección es **940H**, que contiene el valor **3**.

Ciclo de Instrucción. Ejemplo

CodOp 1 = cargar D desde la memoria
CodOp 2 = almacenar D en memoria
CodOp 5 = sumar D con un dato en memoria

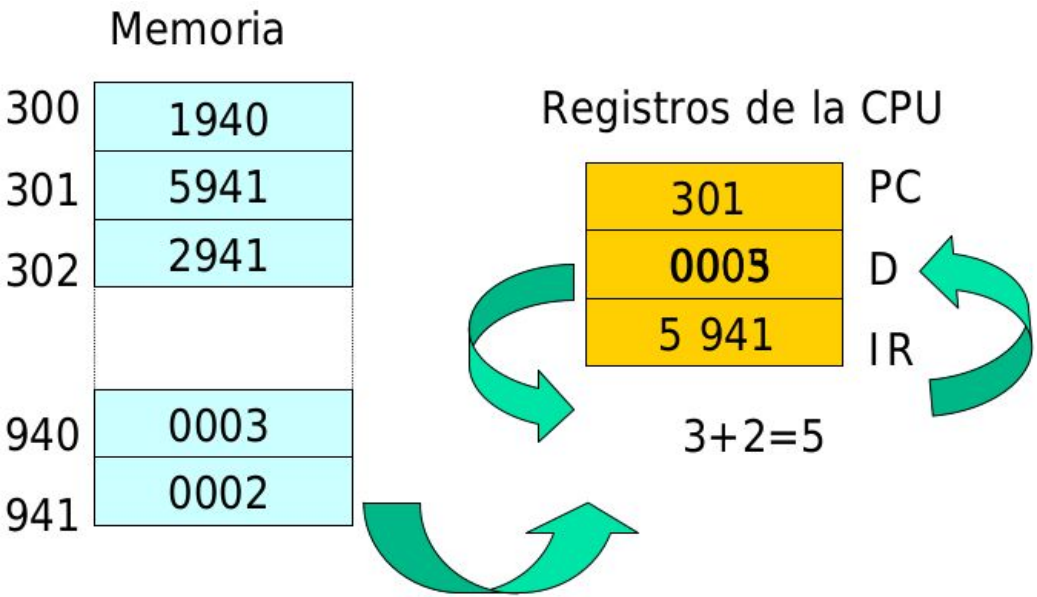


Se incrementa el contador de programa y se busca la siguiente instrucción en la dirección **301H**.

El **5** en **IR** indica que se debe sumar el contenido de una dirección de memoria especificada, en este caso la dirección es **941H**, con el contenido del registro D y almacenar el resultado en D.

Ciclo de Instrucción. Ejemplo

CodOp 1 = cargar D desde la memoria
CodOp 2 = almacenar D en memoria
CodOp 5 = sumar D con un dato en memoria

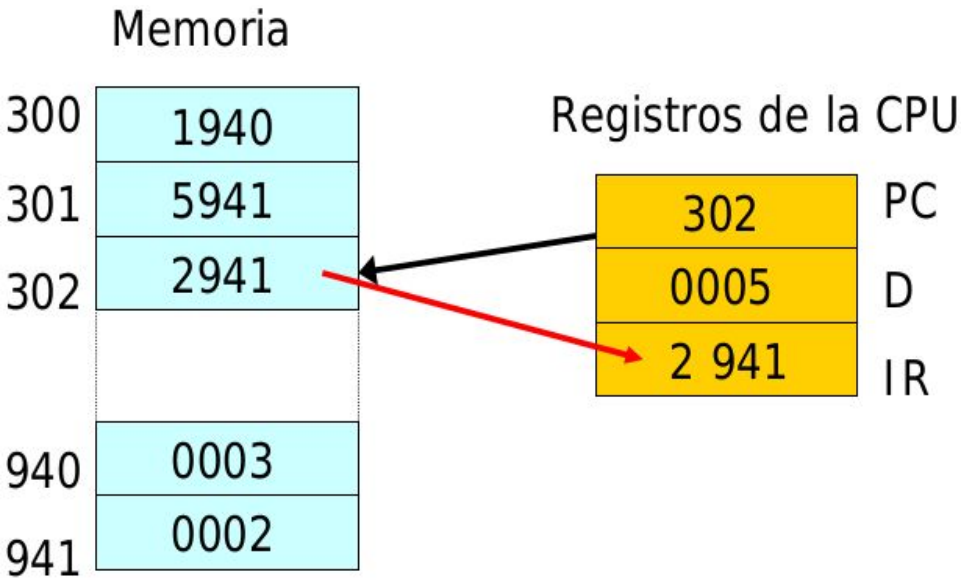


Se incrementa el contador de programa y se busca la siguiente instrucción en la dirección **301H**.

El **5** en **IR** indica que se debe sumar el contenido de una dirección de memoria especificada, en este caso la dirección es **941H**, con el contenido del registro D y almacenar el resultado en D.

Ciclo de Instrucción. Ejemplo

CodOp 1 = cargar D desde la memoria
CodOp 2 = almacenar D en memoria
CodOp 5 = sumar D con un dato en memoria

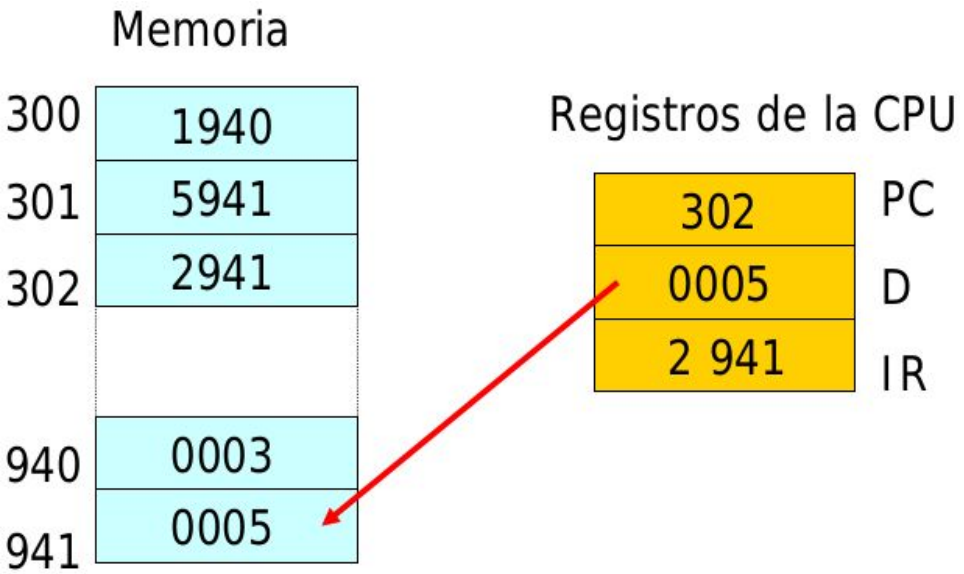


Se incrementa el **PC** y se busca la siguiente instrucción en **302H**.

El **2** en **IR** indica que el contenido del registro **D** se almacene en la dirección **941H**, que está especificada en los bits restantes de la instrucción.

Ciclo de Instrucción. Ejemplo

CodOp 1 = cargar D desde la memoria
CodOp 2 = almacenar D en memoria
CodOp 5 = sumar D con un dato en memoria

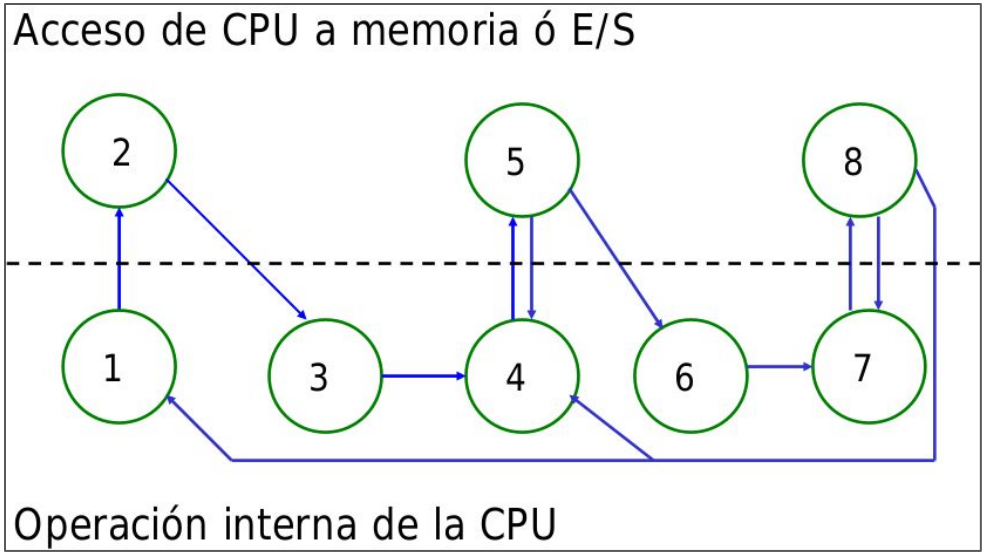


En este ejemplo, en total se necesitaron 3 ciclos de instrucción, cada uno con un ciclo de búsqueda y un ciclo de ejecución.

Con este ejemplo podemos ahora tener una visión más detallada del ciclo de instrucción básico.

Ciclo de Instrucción. Accesos a memoria o E/S

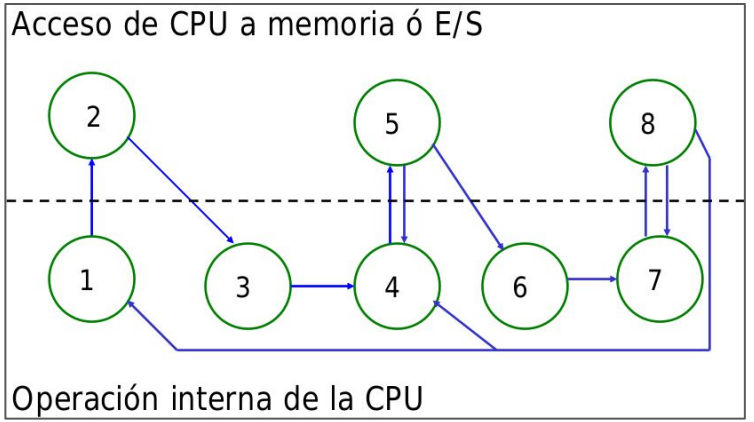
¿Cuántos accesos a memoria se realizan en un ciclo de instrucción?



Dependiendo la instrucción, algunos estados pueden no estar y otros pueden repetirse.

Ciclo de Instrucción. Accesos a memoria o E/S

1. cálculo dirección instrucción: determina la dirección de la siguiente instrucción a ejecutarse
2. búsqueda instrucción: lee la instrucción de su posición de memoria a la cpu.
3. decodificación de la instrucción: analiza la instrucción para determinar el tipo de operación a realizar y los operandos que se usarán.
4. cálculo dirección operando: si la operación implica la referencia a un operando en la memoria ó e/s, entonces se determina la dirección.
5. búsqueda del operando: busca el operando en la memoria ó e/s.
6. operación sobre los datos: ejecuta la instrucción.
7. cálculo dirección resultado.
8. almacenamiento resultado.



Temas de la primera parte

- Introducción y definiciones
- Evolución Histórica
- Arquitectura Von Neumann
- CPU
- Ciclo de instrucción
- Lógica digital ←

Lógica digital

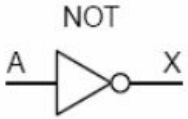
Un circuito digital es uno en el que están presentes **dos** valores lógicos.

Las compuertas son dispositivos electrónicos que **pueden realizar distintas funciones** con estos dos valores lógicos.

Una compuerta lógica es la **mínima operación digital** que se puede realizar

Las compuertas básicas son **NOT**, **AND**, **OR**, **XOR** y sus derivados **NAND**, **NOR**, **XNOR**

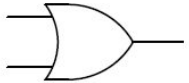
Lógica digital. Compuertas lógicas.



A	X
0	1
1	0

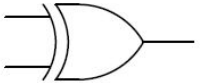
AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1



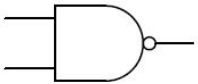
OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



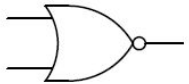
XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



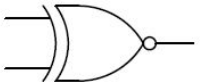
NAND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



NOR

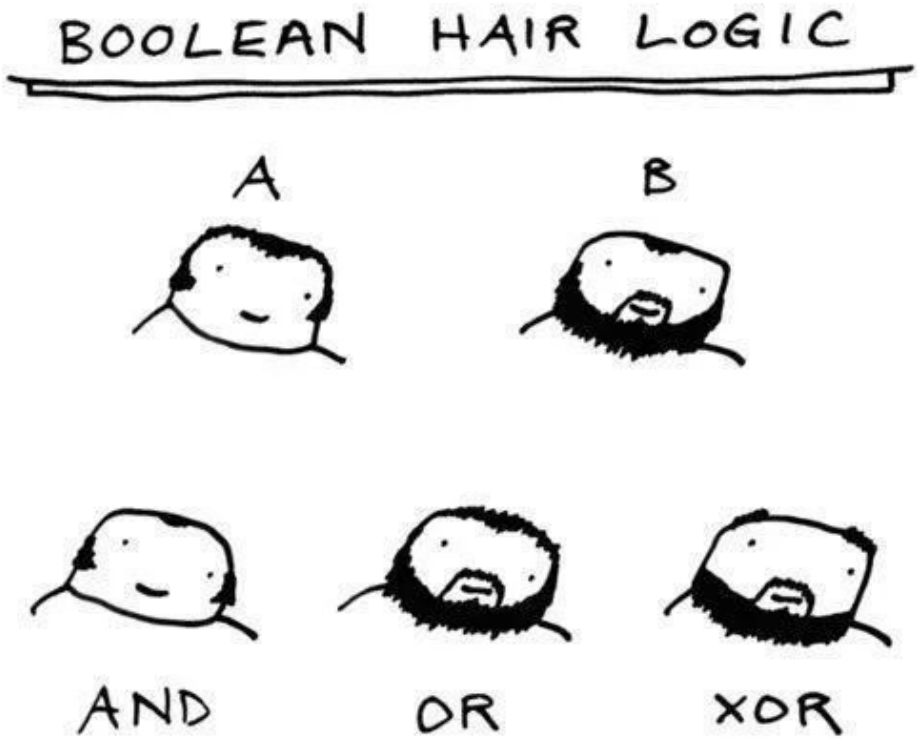
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0



XNOR

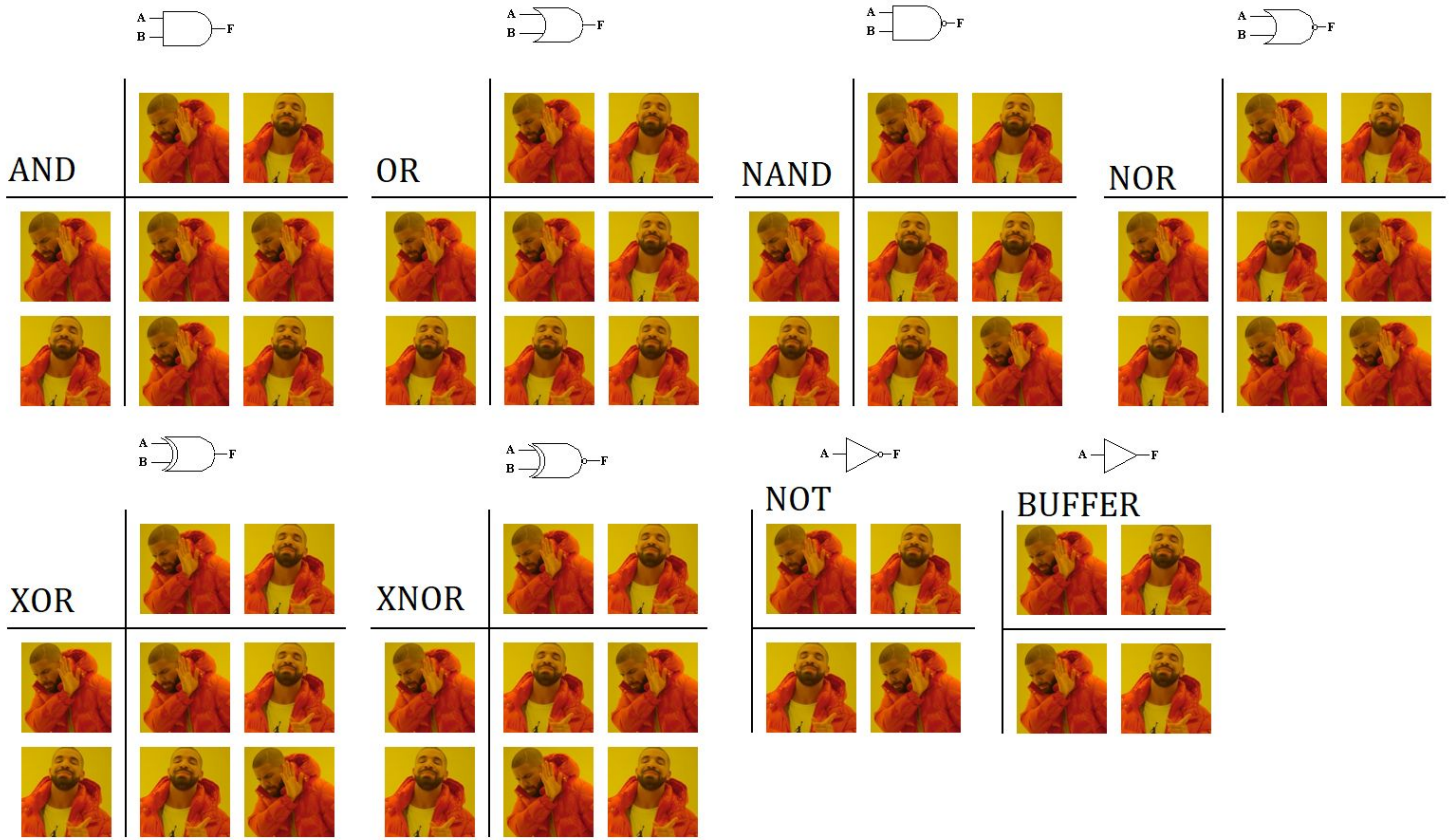
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Lógica digital. Compuertas lógicas.



Lógica digital. Compuertas lógicas.

Logic gates truth table.
Drake version.



Lógica digital. Algebra booleana.

Para describir los circuitos que pueden construirse combinando compuertas, se requiere **un nuevo tipo de álgebra**, donde las variables y funciones sólo puedan adoptar valores **0** ó **1**: **álgebra booleana**.

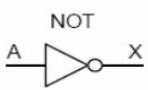
Una función booleana de **n** variables (entradas) tiene $\mathbf{2^n}$ combinaciones de los valores de entrada.

Una función booleana puede describirse con una tabla de $\mathbf{2^n}$ renglones, donde cada renglón indica una combinación única de entradas: **tabla de verdad**.

George Boole (1815-1864)

Lógica digital. Algebra booleana.

Las compuertas pueden expresarse como funciones, por ejemplo **F = A.B**



A	X
0	1
1	0

AND

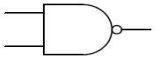
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



NAND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

NOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

XNOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

$\overline{A}$ = **NOT** A

A.B = A **AND** B

A+B = A **OR** B

A⊕B = A **XOR** B

$\overline{A.B}$ = A **NAND** B

$\overline{A+B}$ = A **NOR** B

$\overline{A \oplus B}$ = A **XNOR** B

Lógica digital. Algebra booleana.

Propiedades

Identidad	$1.A=A$	$0+A=A$
Nula	$0.A=0$	$1+A=1$
Idempotencia	$A.A=A$	$A+A=A$
Inversa	$A.\overline{A}=0$	$A+\overline{A}=1$
Conmutativa	$A.B=B.A$	$A+B=B+A$
Asociativa	$(AB).C=A(BC)$	$(A+B)+C=A+(B+C)$
Distributiva	$A+B.C=(A+B).(A+C)$	$A.(B+C)=AB+AC$
Absorción	$A.(A+B)=A$	$A+A.B=A$
De Morgan	$\overline{A.B}=\overline{A}+\overline{B}$	$\overline{A+B}=\overline{A}.\overline{B}$

RECORDAR

$\overline{A}$ = **NOT** A

A.B = A **AND** B

A+B = A **OR** B

A⊕B = A **XOR** B

Lógica digital. Algebra booleana.

Recordar

- En un **AND**, basta que una de sus entradas sea **0** para que la función valga **0**.
- En un **OR**, basta que una de sus entradas sea **1** para que la función valga **1**.
- Aplicar **XOR** con **1** invierte el valor de la variable.
- Aplicar **XOR** con **0** deja el valor de la variable como estaba.

$X.0=0 \qquad X.1=X$

$X+0=X \qquad X+1=1$

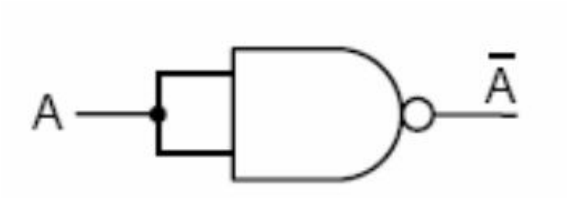
$X \oplus 0=X \qquad X \oplus 1=\overline{X}$

Lógica digital. Algebra booleana.

Leyes de De Morgan

Ejemplo: Implementar una compuerta **NOT** mediante compuerta **NAND**

$$F=\overline{A.B}=\overline{A.A}=\overline{A}$$

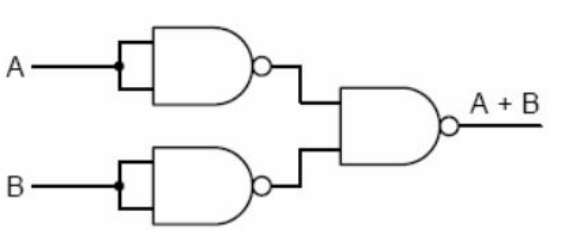


Lógica digital. Algebra booleana.

Leyes de De Morgan

Ejemplo: Implementar una compuerta **OR** mediante compuertas **NAND**

$$F = A + B = \overline{\overline{A + B}} = \overline{\overline{A} \cdot \overline{B}}$$

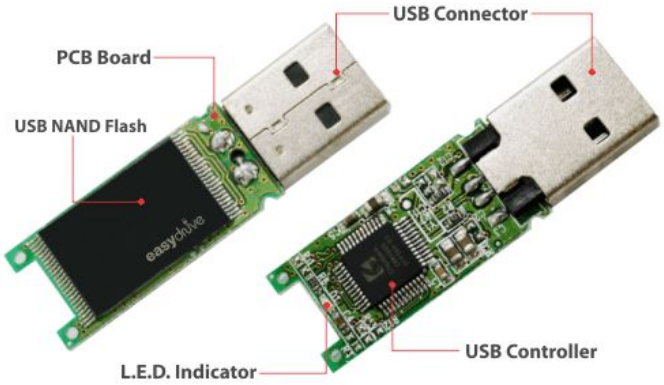


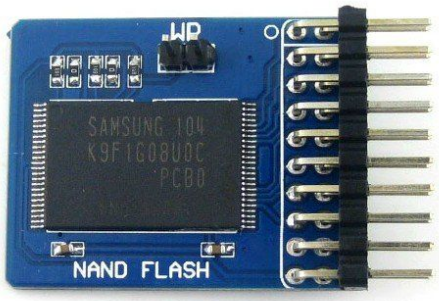
- Tambien es posible implementar **OR**, **XOR** mediante **NAND**.
- **NAND** y **NOR** permiten implementar cualquier otra compuerta lógica.

Lógica digital. Algebra booleana.

NAND y **NOR** permiten implementar cualquier otra compuerta lógica.

Fabricación masiva y abaratamiento de costos



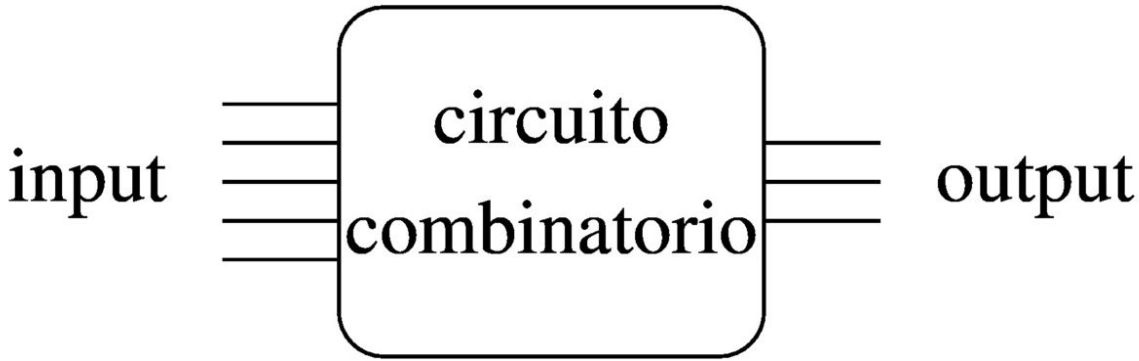


Lógica digital. Circuitos combinatorios.

Las compuertas lógicas pueden **combinarse** para generar **circuitos combinatorios.**

A partir del estado de las entradas, se obtiene un estado en particular en la salida.

La salida dependerá del estado de las entradas y del circuito combinatorio definido.

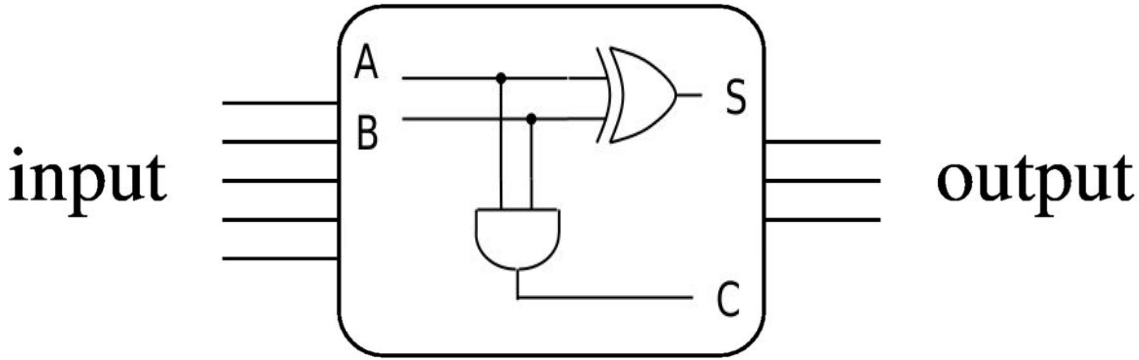


Lógica digital. Circuitos combinatorios.

Ejemplo: Semi-sumador

Es un circuito que permite realizar la suma aritmética entre 2 bits.

En este caso, la entrada son las líneas **A** y **B**, mientras que la salida es **S** y **C**.

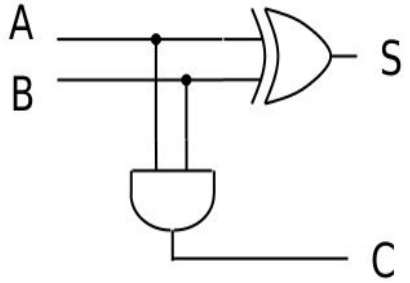


Lógica digital. Circuitos combinatorios.

Ejemplo: Semi-sumador

- **S** representa la suma aritmética de los bits A más B (ambos con un tamaño de sólo 1 bit)
- **C** es el valor del posible acarreo (caso 1+1=2, o 10 en binario)

A B	S	C
0 0	0	0
0 1	1	0
1 0	1	0
1 1	0	1



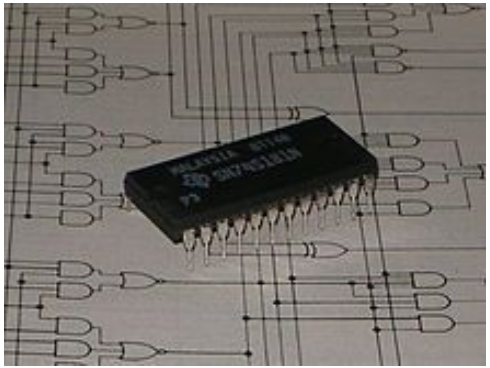
Suma aritmética

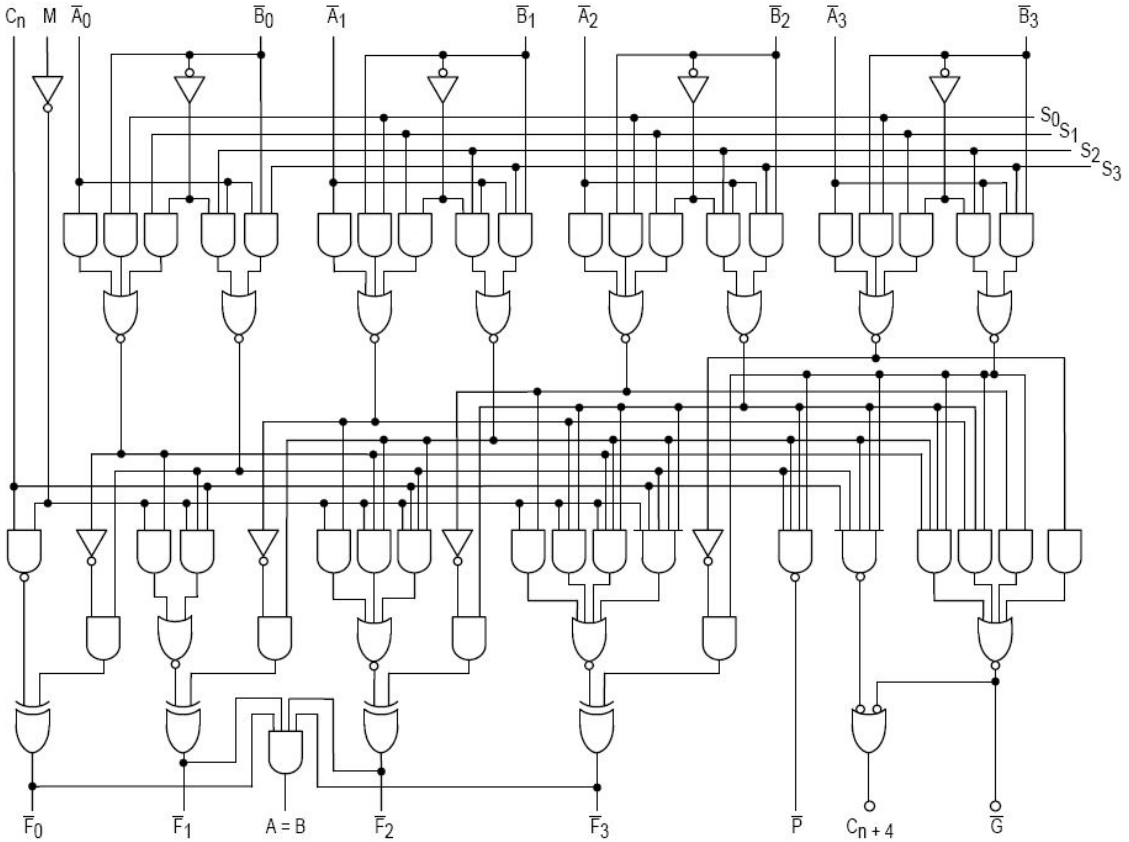
A + B =	C	S
0 + 0 =	0	0
0 + 1 =	0	1
1 + 0 =	0	1
1 + 1 =	1	0

Lógica digital. Circuitos combinatorios.

Ejemplo: **Intel 74181** (1970).
Primer ALU en un solo chip.
Operandos de 4 bits.
16 operaciones lógicas
16 operaciones aritméticas

- Sumas / Restas
- Incrementar / Decrementar
- Operaciones lógicas
- Desplazamiento (shifting)





Lógica digital. Diseñando circuitos.

¿Cómo diseñar un circuito o función booleana que tenga un propósito en particular?

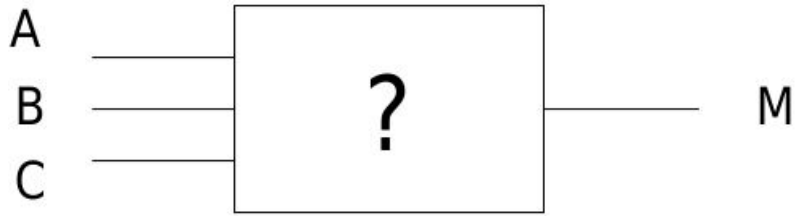
Suma de productos: mecanismo que permite convertir un requerimiento coloquial (o formal) en una función (y por consiguiente en un circuito).

Pasos:

- Escribir la tabla de verdad para la función que se quiera implementar
- Dibujar un AND para cada término que tiene un 1 en la salida
- Invertir las entradas necesarias
- Unir todas las AND a una OR

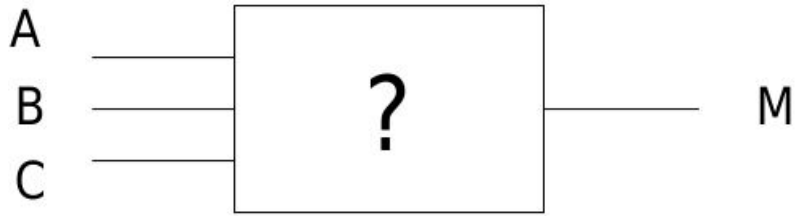
Lógica digital. Diseñando circuitos.

Ejemplo: construir la tabla de verdad e implementar el circuito de una función booleana M, de tres entradas A, B y C, tal que M=1 cuando la cantidad de ‘1’ en A, B y C es ≥ 2 y M=0 en otro caso.



Lógica digital. Diseñando circuitos.

Ejemplo: construir la tabla de verdad e implementar el circuito de una función booleana M, de tres entradas A, B y C, tal que M=1 cuando la cantidad de ‘1’ en A, B y C es ≥ 2 y M=0 en otro caso.

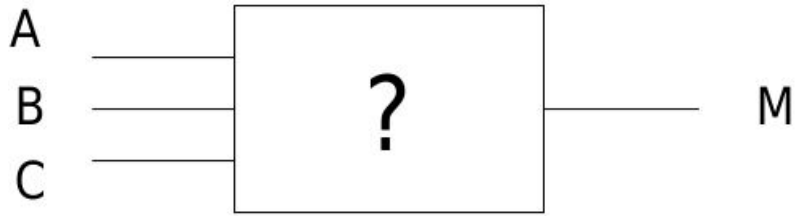


$$M = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

A	B	C	M	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	←
1	0	0	0	
1	0	1	1	←
1	1	0	1	←
1	1	1	1	←

Lógica digital. Diseñando circuitos.

Ejemplo: construir la tabla de verdad e implementar el circuito de una función booleana M, de tres entradas A, B y C, tal que M=1 cuando la cantidad de ‘1’ en A, B y C es ≥ 2 y M=0 en otro caso.



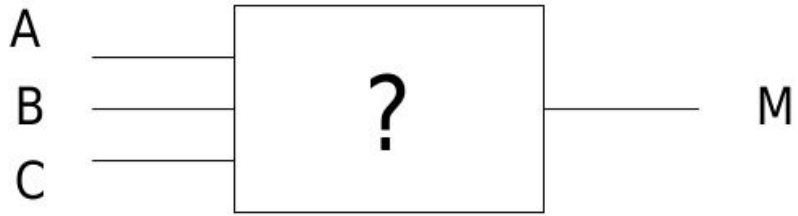
A	B	C	M	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	←
1	0	0	0	
1	0	1	1	←
1	1	0	1	←
1	1	1	1	←

$$M = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

- Hay tantos términos como 1s en la tabla
- Cada término vale 1 para una única combinación de A, B y C
- Las variables que valen 0 en la tabla aparecen aquí negadas

Lógica digital. Diseñando circuitos.

Ejemplo: construir la tabla de verdad e implementar el circuito de una función booleana M, de tres entradas A, B y C, tal que M=1 cuando la cantidad de ‘1’ en A, B y C es ≥ 2 y M=0 en otro caso.



$$M = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

- Hay tantos términos como 1s en la tabla
- Cada término vale 1 para una única combinación de A, B y C
- Las variables que valen 0 en la tabla aparecen aquí negadas