INTRODUCCIÓN Y CODIFICACIÓN

ORGANIZACIÓN Y ARQUITECTURA DE UNA COMPUTADORA

La primera pregunta que surge a un lector que recién comienza sus estudios en la ciencia informática es: **¿Qué es una computadora?** Es una máquina electrónica, analógica o digital, dotada de una memoria de gran capacidad y de métodos de tratamiento de información, capaz de resolver problemas matemáticos y lógicos mediante la utilización automática de programas informáticos. Este dispositivo electrónico está diseñado para aceptar datos de entrada y realizar operaciones sobre ellos (organizadas en una secuencia lógica y predeterminada por un algoritmo), para elaborar resultados que se puedan obtener como salidas.

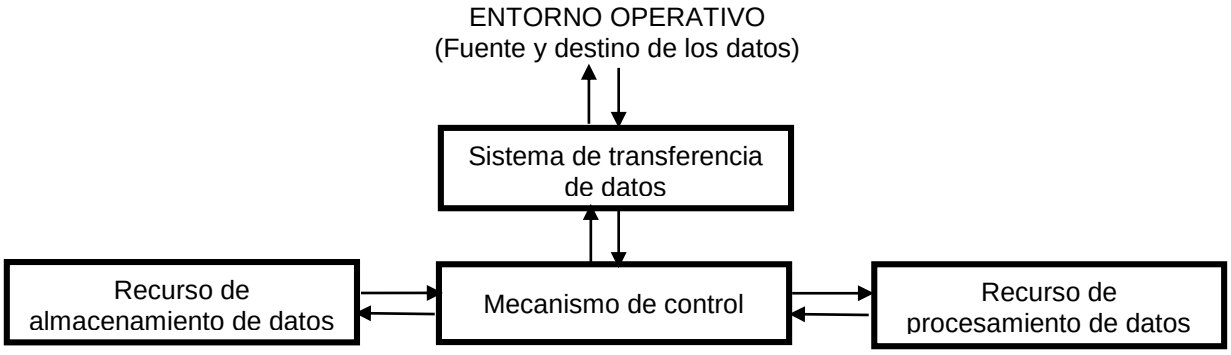
Un algoritmo computacional se determina por una secuencia de operaciones finita que permite resolver un problema computacional. Se representa con instrucciones que la computadora puede interpretar y ejecutar.

Al conjunto de instrucciones que representa un algoritmo se lo denomina programa; expresado de otra manera, un programa es la representación de un algoritmo en un lenguaje de programación.

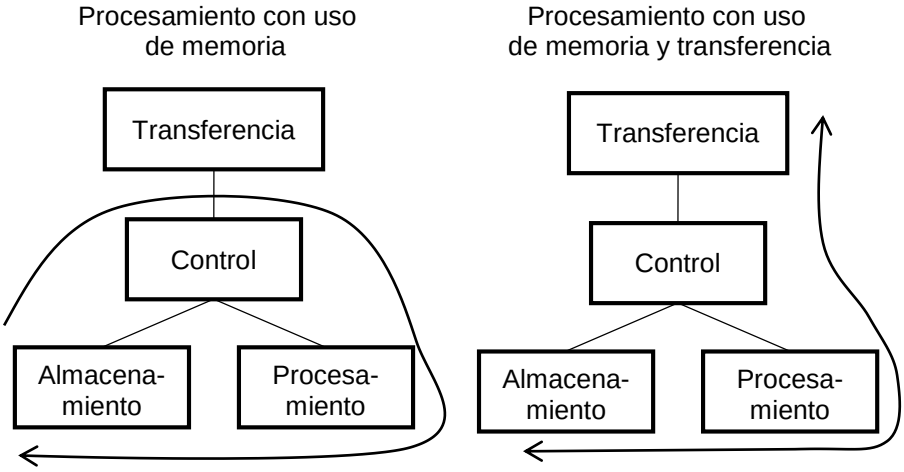
¿CÓMO FUNCIONA?

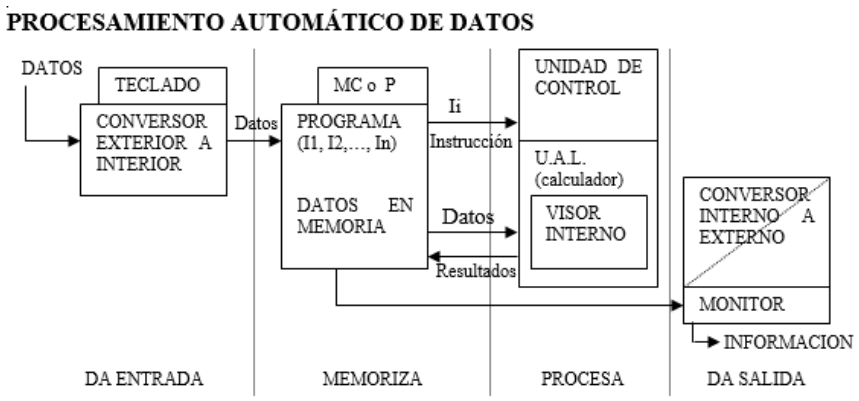
Tanto la estructura como el funcionamiento de un computador son en esencia sencillos. En términos generales hay solo cuatro:

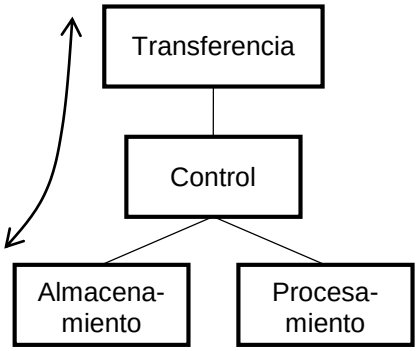
- Procesamiento de datos
- Almacenamiento de datos
- Transferencia de datos
- Control



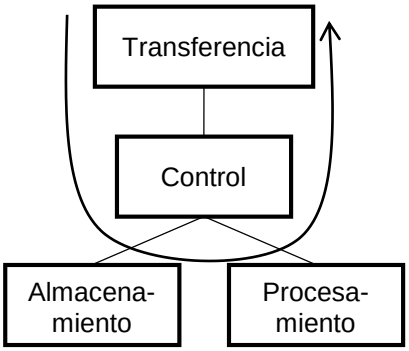
El computador, por supuesto, tiene que ser capaz de **procesar datos.** Los datos pueden adoptar una gran variedad de formas, y el rango de los requisitos de procesado es amplio.







También es esencial que un computador **almacene datos**. Incluso si el computador está procesando datos al vuelo (es decir, los datos se introducen, se procesan, y los resultados se obtienen inmediatamente), el computador tiene que guardar temporalmente al menos aquellos datos con los que está trabajando en un momento dado.



El computador tiene que ser capaz de **transferir datos entre él mismo y el mundo exterior.** El entorno de operación del computador se compone de dispositivos que sirven bien como fuente o bien como destino de datos. Cuando se reciben o se llevan datos a un dispositivo que está directamente conectado con el computador, el proceso se conoce como entrada-salida (E/S), y este dispositivo recibe el nombre de periférico. El proceso de transferir datos a largas distancias, desde o hacia un dispositivo remoto, recibe el nombre de *comunicación de datos.*

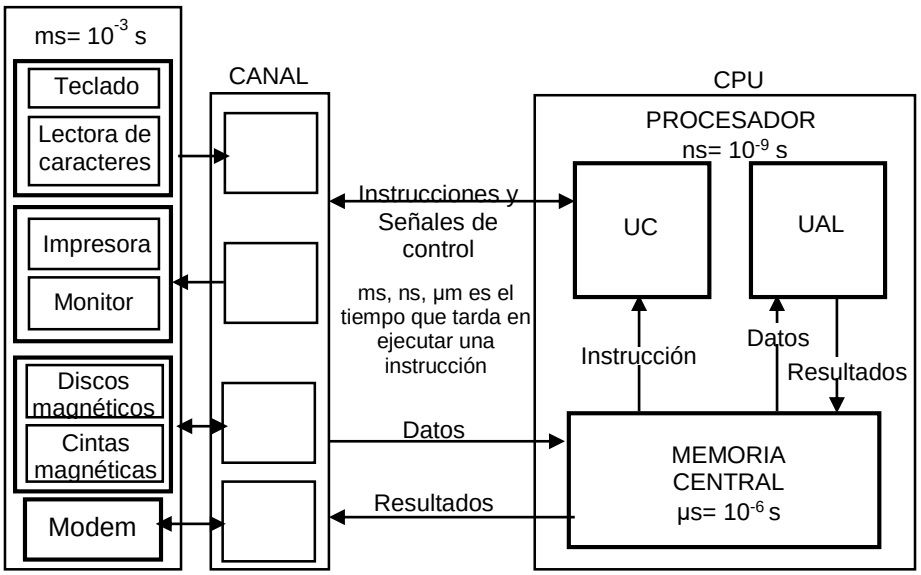
La "relación" entre los distintos componentes y su diseño y tecnología, sea en un nivel de detalle como el presentado o en uno menos abstracto, se define como organización de una computadora.

ESTRUCTURA

El computador es una entidad que interactúa de alguna manera con su entorno externo. En general, todas sus conexiones con el entorno externo pueden ser clasificadas como dispositivos periféricos o líneas de comunicación.

- **Unidad Central de Procesamiento (CPU, Central Processing Unit):** controla el funcionamiento del computador y lleva a cabo sus funciones de procesamiento de datos. Frecuentemente se le llama simplemente procesador.
- **Memoria principal:** almacena datos.
- **E/S:** transfiere datos entre el computador y el entorno externo.
- **Sistema de interconexión:** es un mecanismo que proporciona la comunicación entre la CPU, la memoria principal y la E/S.
- **Unidad de control:** controla el funcionamiento de la CPU y por tanto del computador.
- **Unidad aritmético-lógica (ALU, Arithmetic Logic Unit):** lleva a cabo las funciones de procesamiento de datos del computador.
- **Registros:** proporcionan almacenamiento interno a la CPU.
- **Interconexiones CPU:** son mecanismos que proporcionan comunicación entre la unidad de control, la ALU y los registros.

Interrupciones: son eventos externos producidos por dispositivos entrada/salida (E/S).



Ciclo de la máquina -> Búsqueda e instrucción
Búsqueda y operando
Tratamiento del cálculo
Preparación de la siguiente instrucción

PERIF. DE ENTRADA: Proporcionarán al sistema los datos provenientes del exterior La función es sensar (detectar) la existencia de señales que representan símbolos (letras, números, etc.) y convertir esa señal eléctrica detectada en señales eléctricas binarias internas.

PERIF. DE SALIDA: Envían al exterior los resultados almacenados en memoria. Las señales eléctricas binarias internas que codifican los datos provenientes de la memoria deben convertirse en el periférico en señales adecuadas para el almacenamiento, visualización o transmisión.

MEMORIA PRINCIPAL O CENTRAL: almacena instrucciones y datos provenientes desde los periféricos y guarda resultados intermedios y finales. En general un periférico no puede pasar información a otro sin que intervenga la

M.C.- PROCESADOR CENTRAL: Lugar donde se ejecutan las instrucciones contenidas en MC Si el procesador central está constituido por un único circuito integrado se denomina microprocesador (mP). El procesador tiene 3 secciones principales:

*UC: Unidad de control

*UAL: Unidad Aritmético Lógica.

*Registros auxiliares

CANALES (Interfases de Adaptación y Control de Periféricos y Unidades de Acceso Directo a Memoria -IACP y UADM-) Los canales logran la simetría y también coordinan cómo impacta en el periférico. Son dispositivos intermedios entre los periféricos y la memoria. Su función es adaptar las diferencias de velocidad de funcionamiento existente entre la CPU y los periféricos.

Analicemos las diferencias de velocidad:

El tiempo de acceso a memoria para leer o escribir una información está en el orden de 10 a 100 ns. (1 ns = 10-9 s) por lo tanto en 1 s se podrían realizar 107 escrituras o lecturas.

En un disco, teniendo en cuenta que posee elementos mecánicos, necesita de 1 a 5 ms. Para que la cabeza de lectura se posicione sobre la pista y sector correspondiente y luego la información puede transmitirse a aproximadamente 500.000 a un millón de transferencias /segundo.

En una impresora rápida se pueden lograr 6000 transferencias/segundo.

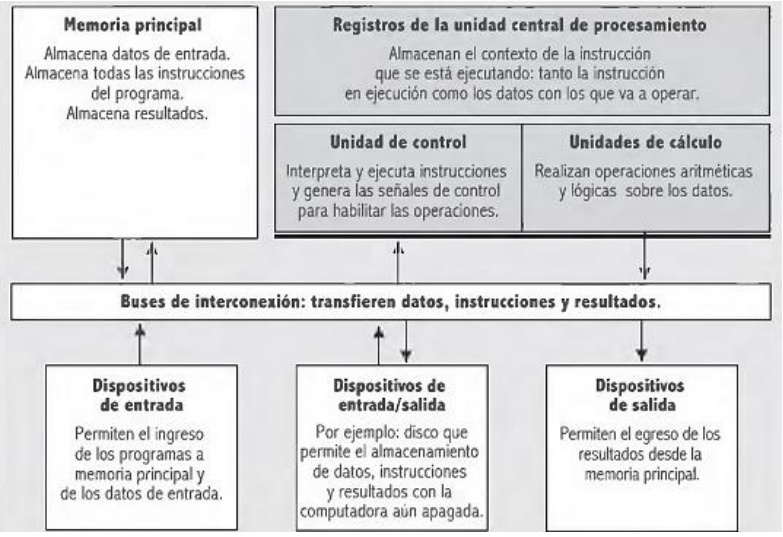
Los canales (IACP) deben adaptar estas disímiles velocidades y para ello poseen una memoria buffer de capacidad limitada.

MEMORIA BUFFER: Utilizan algunos periféricos para adaptarse a la velocidad de la CPU e igualmente los canales, dados a que algunos periféricos y canales no trabajan a la misma velocidad que el procesador necesitan de la memoria buffer.

NOCIONES SOBRE EL HARDWARE

Podemos admitir que el hardware se nutre de dos disciplinas: la TECNOLOGIA (fundamentalmente electrónica) y la LÓGICA. La tecnología aporta los avances en los componentes (transistores, resistencias, elementos de memoria, dispositivos ópticos, etc.) y la lógica aporta el estudio de cómo ensamblar esos componentes para construir registros, sumadores, circuitos de selección o unidades de mayor complejidad como ser memorias, la unidad aritmético – lógica, etc.

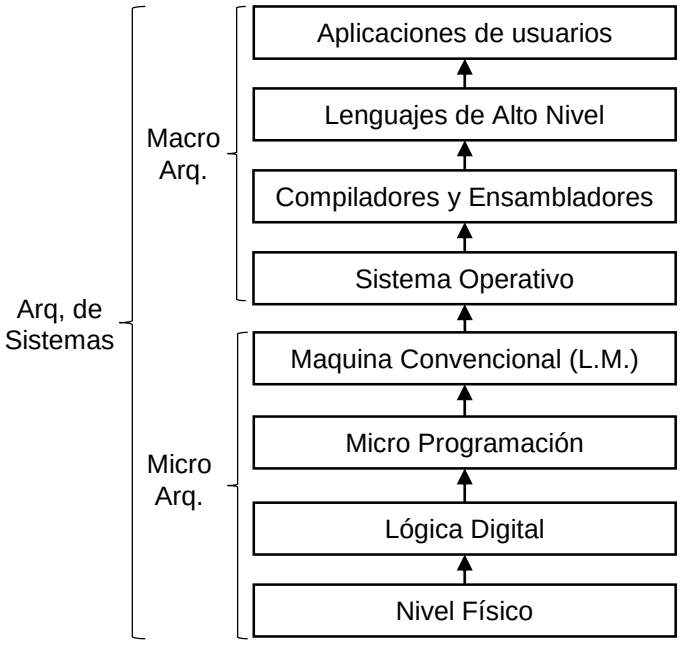
La ARQUITECTURA es otra disciplina, de importancia cada vez mayor, que apoya la construcción de sistemas informáticos de complejidad cada vez mayor.

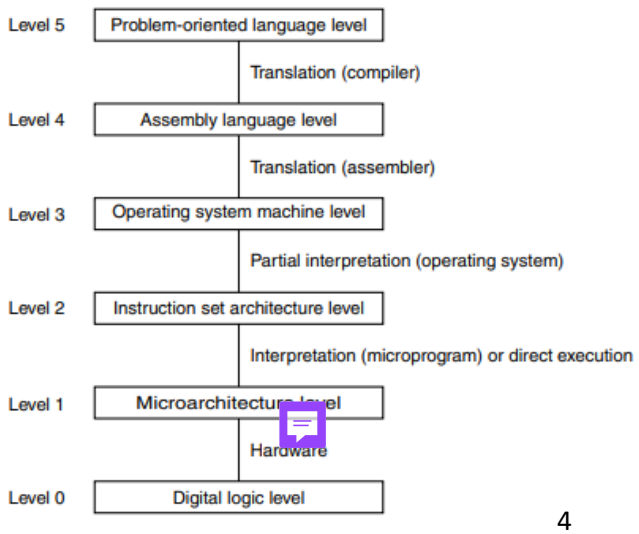


El **set de instrucciones** de una computadora permite representar los algoritmos que solucionan los problemas.

En el concepto de arquitectura de computadoras se considera la descripción de las características visibles relativas a las prestaciones que una determinada configuración interna de computadoras puede brindar. La organización de una computadora permite la identificación de los componentes desde el punto de vista de su estructura, la manera en que se relacionan entre sí y las cuestiones de índole tecnológico.

El concepto de arquitectura de computadoras incluye los aspectos relacionados con el formato del conjunto de instrucciones que el procesador pueda ejecutar, la representación interna de los datos y el estudio de los módulos de hardware que sostienen la dinámica del conjunto, desde la perspectiva del sistema informático.





The **digital logic level**, the interesting objects are called gates. Although built from analog components, such as transistors, gates can be accurately modeled as digital devices. Each gate has one or more digital inputs (signals representing 0 or 1) and computes as output some simple function of these inputs, such as AND or OR. Gates can also be combined to form the main computing engine itself.

The next level up is the **microarchitecture level**. At this level we see a collection of (typically) 8 to 32 registers that form a local memory and a circuit called an ALU (Arithmetic Logic Unit), which is capable of performing simple arithmetic operations. The registers are connected to the ALU to form a data path, over which the data flow. On some machines the operation of the data path is controlled by a program called a microprogram. On other machines the data path is controlled directly by hardware.

At level 2 we have a level that we will call the **Instruction Set Architecture level or (ISA level**). When they describe the machine's instruction set, they are in fact describing the instructions carried out interpretively by the microprogram or hardware execution circuits.

The next level is usually a hybrid level. Most of the instructions in its language are also in the ISA level. In addition, there is a set of new instructions, a different memory organization, the ability to run two or more programs concurrently, and various other features. This is what we mean by ''hybrid'' level. Throughout this book we will call this level **the operating system machine level**.

There is a fundamental break between levels 3 and 4. The lowest three levels are not designed for use by the average garden-variety programmer. Instead they are intended primarily for running the interpreters and translators needed to support the higher levels. These interpreters and translators are written by people called systems **programmers** who specialize in designing and implementing new virtual machines.

Level 4, the assembly language level, is really a symbolic form for one of the underlying languages. This level provides a method for people to write programs for levels 1, 2, and 3 in a form that is not as unpleasant as the virtual machine languages themselves. Programs in assembly language are first translated to level 1, 2, or 3 language and then interpreted by the appropriate virtual or actual machine. The program that performs the translation is called an **assembler**

Level 5 usually consists of languages designed to be used by applications programmers with problems to solve. Such languages are often called high-level languages. Literally hundreds exist. A few of the better known ones are C, C++, Java. Programs written in these languages are generally translated to level 3 or level 4 by translators known as **compilers**, although occasionally they are interpreted instead.

In summary, the key thing to remember is that computers are designed as a series of levels, each one built on its predecessors. Each level represents a distinct abstraction, with different objects and operations present. The study of how to design those parts of a computer system that are visible to the programmers **is called computer architecture**.

CLASIFICACIÓN DE LAS COMPUTADORAS

ANALÓGICAS

Las primeras computadoras analógicas se implementaron para estudiar un modelo semejante (análogo) a una ecuación; el resultado se elaboró tras la medición del valor que asumían las salidas del modelo. Sin embargo, el sistema adolecía de dos desventajas importantes: por un lado, la falta de exactitud en la salida, a causa del carácter continuo de esta magnitud, y por el otro, el modelo construido, que representaba a una única aplicación y no servía para otra.

La ventaja más destacada que ofrece esta computadora es su capacidad de procesar datos no discretos (temperaturas, presión, altura), además de permitir la simulación de modelos en la etapa de desarrollo de un proyecto.

DIGITALES

Procesan "dígitos" binarios, "ceros" y "unos" que representan datos. Un dato es un binario que corresponde a un código preestablecido. Entre las ventajas que presentan podemos destacar que efectúan cálculos precisos y que el modelo se arma sobre un programa intercambiable, lo que posibilita no tener que cambiar partes físicas para modificarlo, sino sólo ingresar un programa nuevo.

Para procesar variables continuas y estudiar el modelo a fondo en este tipo de computadoras, es necesario ingresar datos con diferencias infinitesimales, motivo por el cual su utilización en estos casos resulta inadecuada.

HIBRIDAS

Los sistemas híbridos son una combinación de analógicos y digitales. Mientras que la porción analógica se encarga de tomar los datos continuos (temperatura, presión, etc.), la parte digital efectúa los cálculos. Estas computadoras se construyen para propósitos especiales; un ejemplo actual es el GPS de nuestro auto.

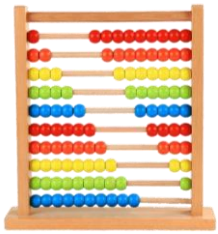
EVOLUCIÓN Y PRESTACIONES DE LOS COMPUTADORES

PUNTOS CLAVE

- La evolución de los computadores se ha caracterizado por un incremento de la velocidad del procesador, una disminución del tamaño de los componentes, un aumento del tamaño de memoria, y un aumento de la capacidad de E/S y de la velocidad.
- Otro factor responsable del gran aumento de la velocidad del procesador es la disminución del tamaño de los componentes del microprocesador; esto reduce la distancia entre componentes, y, por tanto, aumenta la velocidad. Todas estas técnicas se diseñan para mantener al procesador ocupado el mayor tiempo posible.
- Otro asunto crítico en el diseño de computadores es hacer un balance de las prestaciones de los distintos elementos, de forma que esta ganancia en prestaciones en un área no perjudique a otras áreas. En particular, la velocidad del procesador ha aumentado más rápidamente que el tiempo de acceso a memoria.

Generación	Años	Características
0	hasta 1945	Sistemas mecánicos y electro-mecánicos
1	1945 – 1955	Tubos al vacío, tableros
2	1955 – 1965	Transistores y sistemas por lotes
3	1965 – 1980	Circuitos integrados y multiprogramación
4	desde 1980	Computadores personales

GENERACIÓN 0 (Hasta 1945)



- Calculadoras mecánicas.
- Sistemas basados en relés.

Desde épocas remotas (alrededor de 3.000 años a.C.) el hombre trató de liberarse de hacer cálculos en forma manual. Es probable que la primera máquina típicamente digital que el hombre utilizó para resolver problemas aritméticos haya sido el **ábaco**. Este es un objeto que sirve para facilitar cálculos sencillos (sumas, restas, multiplicaciones) y operaciones aritméticas. Se trata de cierto número de cuentas engarzadas con varillas, cada una de las cuales indica una cifra del número que representa.

The **Pascalina**, made by Blaise Pascal (1623–1662). This device, built in 1642, it was entirely mechanical, using gears, and powered by a hand-operated crank. Pascal's machine could only do addition and subtraction operations; esta máquina fue considerada la primera calculadora digital, llamada así porque acumulaba las operaciones aritméticas -suma y sustracción- en un acumulador o contador de enteros. Su mecanismo se basaba en ruedas dentadas que tenían 10 posiciones (de 0 a 9)\ cada vez que una rueda pasaba de 9 a 0, la rueda inmediatamente a la izquierda avanzaba una posición.

En la primera mitad del siglo XVII John **Napier** introdujo el concepto de logaritmo, con el que la tarea de multiplicar se simplificó. A partir de ese momento, se comenzaron a construir las máquinas de cálculo llamadas **analógicas** o máquinas de medida.

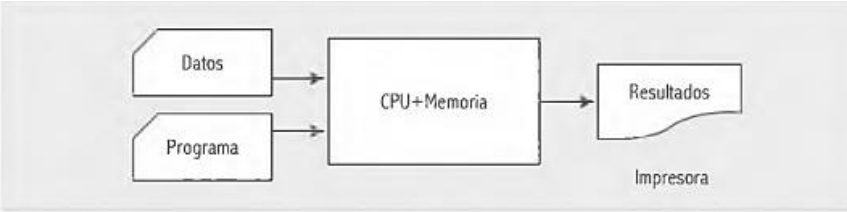
Charles Babbage (1792–1871), the inventor of the speedometer, designed and built his **difference engine.** This mechanical device, which like Pascal's could only add and subtract, was designed to compute tables of numbers useful for naval navigation. The entire construction of the machine was designed to run a single algorithm, the method of finite differences using polynomials. Although the difference engine worked reasonably well, Babbage quickly got bored with a machine that could run only one algorithm. He began to design a successor, called the **analytical engine**. The great advance of the analytical engine was that it was general purpose. It read instructions from punched cards and carried them out. By punching a different program on the input cards, it was possible to have the analytical engine perform different computations, something not true of the difference engine. Since the analytical engine was programmable in a simple assembly language, it needed software. To produce this software, Babbage hired a young woman named Ada Augusta Lovelace. **Unfortunately**, like many modern designers, Babbage never quite got the hardware debugged.

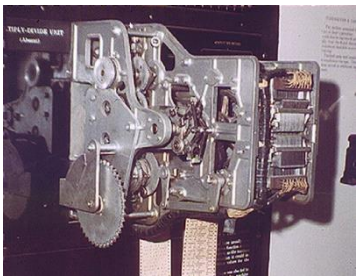
En el siglo XIX George Boole desarrolló un Álgebra que no utiliza números, sino que establece la relación entre conceptos lógicos. Se hizo un paralelismo entre las leyes del pensamiento y las operaciones algebraicas. Esto permitió la representación de conceptos lógicos en términos algebraicos y se denominó Lógica booleana. El **Álgebra de Boole** es un ente matemático que fundamenta los principios de la teoría de circuitos.

Mientras los trabajos sobre máquinas analógicas seguían desarrollándose, hubo una revolución en el campo digital cuando **Hermán Hollerith** inventó la técnica para procesar gran cantidad de datos por medio de **tarjetas perforadas**, para luego clasificar y analizar los datos perforados en ellas. Las ideas de Hollerith fueron tomadas y perfeccionadas por la empresa IBM (International Business Machines). IBM desarrolló un dispositivo básico conocido como tarjeta perforada de 80 columnas. Cada tarjeta era leída por una lectora que permitía detectar las perforaciones en el soporte mediante conmutadores eléctricos. Cuando estos contactos atravesaban las celdillas perforadas, la unidad interpretaba el dato según fuera la combinación de perforaciones en cada columna.

Desde alrededor de 1930 hasta la década de 1970 la tarjeta perforada desempeñó un papel importante en el procesamiento de datos y reemplazó en gran medida al procesamiento manual. El medio básico para el procesamiento era la tarjeta perforada que contenía 80 columnas por 12 filas.

Las intrucciones de un programa se ejecutan unas tras otras, siguen una lógica secuencial, salvo que haya una instrucción de salto.





After graduating, Aiken recognized the importance of being able to do calculations by machine. He went to the library, discovered Babbage's work, and decided to build out of relays the general-purpose computer that Babbage had failed to build out of toothed wheels. Aiken's first machine, the **Mark I**, was completed at Harvard in 1944. It had 72 words of 23 decimal digits each, and had an instruction time of 6 sec. Input and output used punched paper tape. By the time Aiken had completed its successor, the Mark II, relay computers were obsolete. The electronic era had begun.

GENERACIÓN 1 (1945-1955)

- Se utilizan tubos al vacío.
- Eran enormes (20.000 tubos) y lentas (un ciclo ≈ 1 seg.)
- Un solo grupo diseñaba, construía, programaba, operaba y mantenía cada máquina.
- Toda la programación se hacía en lenguaje máquina (alambrando tableros por ejemplo).
- No existían los sistemas operativos.
- En 1950 se introducen las tarjetas perforadas.

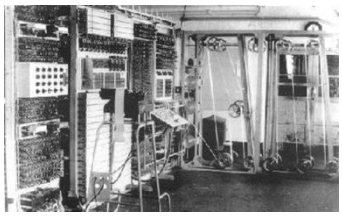
Estas computadoras estaban constituidas por válvulas de vacío, que disipaban gran cantidad de calor y ocupaban una superficie muy amplia. Las tareas se ejecutaban en forma secuencial, lo que implicaba que:

1. El programa, almacenado en tarjetas o cintas perforadas, era cargado en memoria principal por un programa llamado cargador, perteneciente al sistema operativo.
2. Se ejecutaba el programa instrucción por instrucción.
3. Se imprimían los resultados.

Las operaciones de entrada, procesamiento y salida de los datos se encontraban encadenadas en el tiempo, por lo que la duración del proceso era igual a la suma de todas las operaciones.

Ejemplos de máquinas de esta generación:

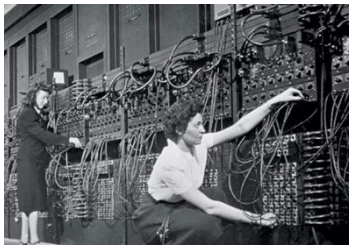
The stimulus for the electronic computer was World War II. During the early part of the war, German submarines were wreaking havoc on British ships. Commands were sent from the German admirals in Berlin to the submarines by radio, which the British could, and did, intercept. The problem was that these messages were encoded using a device called the **ENIGMA**, whose forerunner was designed by amateur inventor and former U.S. president, **Thomas Jefferson**.



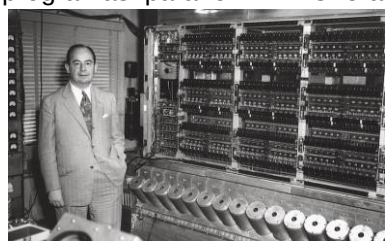
However, to break a coded message, a huge amount of computation was needed, and it was needed very soon after the message was intercepted to be of any use. To decode these messages, the British government set up a top secret laboratory that built an electronic computer called the **COLOSSUS**. The famous British mathematician **Alan Turing** helped design this machine.

El **ENIAC** (Electronic Numerical Integrator And Computer), diseñado y construido bajo la supervisión de John Mauchly y John Presper Eckert en la Universidad de Pennsylvania, fue el primer computador electrónico de propósito general del mundo. El proyecto fue una respuesta a necesidades militares durante la Segunda Guerra Mundial. However, the machine was not finished until 1946, when it was too late to be of any use for its original purpose.

La máquina que construyeron era enorme, pesaba treinta toneladas, ocupaba 15 000 pies cuadrados y contenía más de 18 000 tubos de vacío. Cuando funcionaba consumía 140 Kilowatios de potencia. También era bastante más rápida que cualquier computador electromecánico, ya que era capaz de efectuar 5 000 sumas por segundo. El ENIAC era una máquina decimal y no binaria. Es decir, los números estaban representados en forma decimal y la aritmética se realizaba también en el sistema decimal. Su memoria consistía en veinte «acumuladores», cada uno capaz de contener un número decimal de diez dígitos. Cada dígito estaba representado por un anillo de diez tubos de vacío. Uno de los mayores inconvenientes del ENIAC era que tenía que ser programado manualmente mediante conmutadores y conectando y desconectando cables.

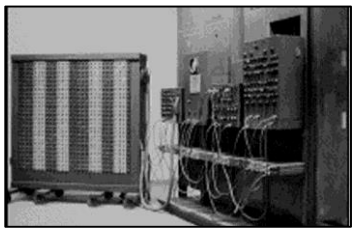


La **máquina de von Neumann**. La tarea de cargar y modificar programas para el ENIAC era extremadamente tediosa. Esta idea conocida como concepto del programa-almacenado, se atribuye a los diseñadores del ENIAC, sobre todo al matemático John von Neumann. La primera publicación de la idea fue en una propuesta de von Neumann para un nuevo computador en 1945, el EDVAC (Electronic Discrete Variable Computer). The von Neumann machine had five basic parts: the memory, the arithmetic logic unit, the control unit, and the input and output equipment.



The arithmetic logic unit and the control unit formed the ‘‘brain’’ of the computer. In modern computers they are combined onto a single chip called the **CPU** (Central Processing Unit). Inside the arithmetic logic unit was a special internal 40-bit register called the accumulator

En 1946 von Neumann y sus colegas empezaron, en el Instituto para Estudios Avanzados de Princeton, el diseño de un nuevo computador de programa-almacenado, que llamaron **IAS**.

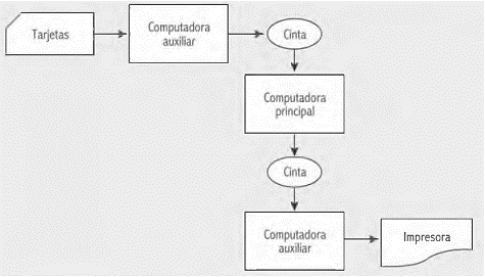


El **UNIVAC I** fue el primer computador comercial de éxito. Estaba diseñado, como su nombre indica, tanto para aplicaciones científicas como comerciales. El primer documento que describía el sistema mencionaba como ejemplos de tareas que podía realizar operaciones algebraicas con matrices, problemas de estadística, reparto de primas para las compañías de seguros de vida y problemas logísticos.

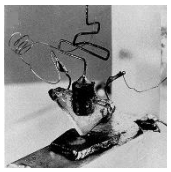
GENERACIÓN 2 (1955-1965)

- Se introducen los transistores.
- Distinción entre diseñadores, constructores, programadores, operadores y personal de mantenimiento.
- Mainframes en salas acondicionadas.
- Se escribían los programas en papel, luego se perforaban las tarjetas, cuarto de entrada, café, esperar la salida.
- Los operadores toman las tarjetas del programa y colocan también los del compilador.
- Se crea el proceso por lotes que agrupa trabajos.

Estaban constituidas por transistores y utilizaron circuitos impresos, lo que permitió reducir el tamaño con respecto a las anteriores. Posibilitaron la simultaneidad entre un cálculo y una operación de E/S. Este concepto en la práctica dio pocos resultados, debido, en gran medida, a la desproporción entre la velocidad de cálculo interno y las velocidades de E/S, que hacían que la CPU no se utilizara más que en un pequeño porcentaje de tiempo. El paliativo para este problema fue que las operaciones de E/S se realizaran utilizando como soporte de almacenamiento unidades de cinta magnética, mucho más rápidas que las lectoras de tarjetas y las impresoras. Para lograrlo, se copiaba la información contenida en el soporte tarjeta a soporte cinta magnética y de ésta a impresora con un procesador auxiliar.



Este procedimiento de explotación, se denominó **procesamiento por lotes**. En esta modalidad de procesamiento, cada lote estaba compuesto por varios trabajos y era preciso esperar que el lote cargado en la cinta magnética se procese por completo para obtener los resultados de cada trabajo.

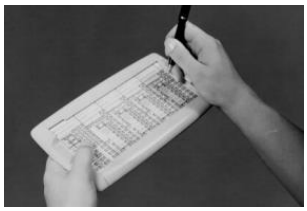


El uso del transistor define la segunda generación de computadores. Cada nueva generación se caracteriza por la mayor velocidad, mayor capacidad de memoria y menor tamaño que la generación anterior.

También hay otros cambios. En la segunda generación se introdujeron unidades lógicas y aritméticas y unidades de control más complejas, el uso de lenguajes de programación de alto nivel, y se proporcionó un software del sistema con el computador.

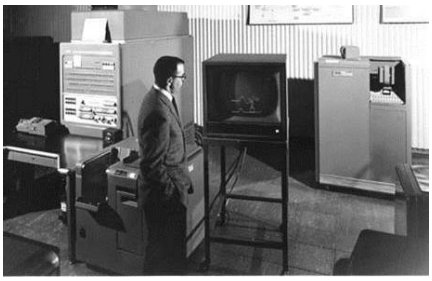
Se desarrollaron nuevos lenguajes de programación como COBOL y FORTRAN, los cuales eran comercialmente accesibles. Se usaban en aplicaciones de sistemas de reservaciones de líneas aéreas, control del tráfico aéreo y simulaciones de propósito general. La marina de los Estados Unidos desarrolla el primer simulador de vuelo, Computadora Whirlwind Surgieron las minicomputadoras y los terminales a distancia. Se comenzó a disminuir el tamaño de las computadoras.

Ejemplos de máquinas de esta generación:



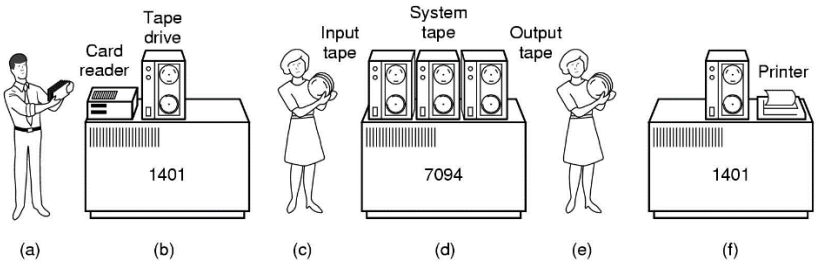
El **JOHNNIAC** fue un ordenador creado por RAND Corp. basado en la Arquitectura de von Neumann, de la que fue pionera la IAS machine. Clon de las IAS. Funcionaba con tarjetas.

La máquina **IBM 704** fue la primera computadora producida en masa con hardware basado en aritmética de coma flotante. Inicialmente desarrollado para ejecutar los lenguajes de programación FORTRAN y LISP, así como el MUSIC 1 (el primer programa de música para computadora). En 1959 se crea un programa capaz de jugar al ajedrez con la máquina, y dos años después es capaz de sintetizar el habla. Para la interacción humana con el equipo, los programas se introducen en tarjetas perforadas inicialmente en lugar de a la consola, y la salida legible se mostraba a través de una impresora.

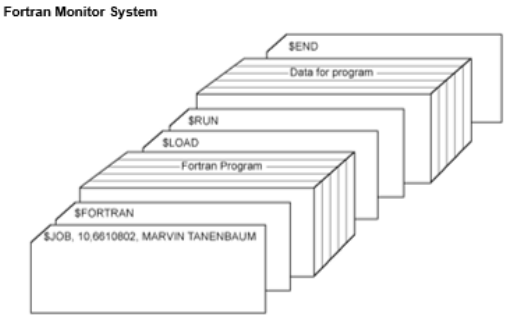


IBM 1401 También llamada Centro de Computo. Tenía 4KB de memoria expandible a 16KB. Buena para leer tarjetas, copiar cintas e imprimir resultados. Mala para cálculos numéricos. Se utilizaba con fines comerciales (bancos, etc.)

IBM 7094 Buena para hacer cómputos. Se utilizaba con fines científicos.



IBM 1401 - IBM 7094:
a) Los programadores llevan tarjetas
b) La 1401 lee un lote de tarjetas y los graba en la cinta
c) Un operador lleva la cinta a la 7094
d) La 7094 realiza los cómputos
e) Un operador lleva la cinta a una 1401
f) La 1401 imprime las salidas



ERMA, General Electric (1959)

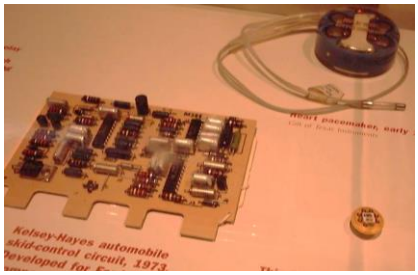
Primer video-juego. Estudiantes de MIT

Invención del mouse (1964).

GENERACIÓN 3 (1965-1980)

- Se introducen los circuitos integrados, lo cual es una gran ventaja en el precio y desempeño del computador.
- Se introduce el sistema IBM 360 altamente compatible.
- Se introduce la multiprogramación

- Se introducen los discos duros.
- Se introduce el tiempo compartido entre usuarios.



Tecnología de circuitos integrados (varios componentes electrónicos incluidos en un solo bloque de silicio). Estos circuitos eran del tipo SSI o Small Scale Integiation (pequeña escala de integración) y MSI o Médium Scale Integration (mediana escala de integración) y permitieron el incremento de la velocidad interna de la computadora y la reducción de la energía que utilizaban.

Moore observó que el número de transistores que se podrían integrar en un solo chip se duplicaba cada año y se predecía correctamente que esto continuaría en un futuro cercano. Las consecuencias de la ley de Moore son profundas:

1. El precio de un chip ha permanecido prácticamente invariable a través de este periodo de rápido crecimiento en densidad. Esto significa que el coste de la lógica del computador y de la circuitería de la memoria ha caído a una velocidad drástica.

2. Ya que los elementos de la lógica y la memoria están más próximos en chips más densamente encapsulados, la longitud de las interconexiones eléctricas ha disminuido, incrementándose así la velocidad operativa.

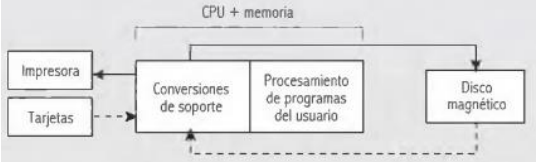
3. El computador es ahora más pequeño, lo que lo hace más adecuado para más entornos.

4. Hay una reducción de las necesidades de potencia y refrigeración.

5. Las interconexiones de los circuitos integrados son mucho más fiables que las conexiones soldadas. Con más circuitos en cada chip hay menos conexiones entre chips.

En esta etapa se pudo explotar la multiprogramación, método que consiste en que varios programas residan en forma simultánea en la memoria en estado de ejecución.

Las computadoras de 3a generación dividen su memoria “lógicamente” en dos zonas: una reservada a los "trabajos del usuario” y la otra a la “conversión de soportes y carga”. A simple vista podría afirmarse que las particiones corresponden a la computadora principal y a la auxiliar de la 2a generación, respectivamente. Sin embargo, hay una diferencia importante: la “carga por lotes” se sustituyó por la "carga continua”**.** Los trabajos se ponen en cola de espera en un disco magnético y el sistema operativo es el que se encarga de ejecutarlos según un nivel de prioridad. Los resultados, que son transferidos al disco, luego son extraídos por la impresora.

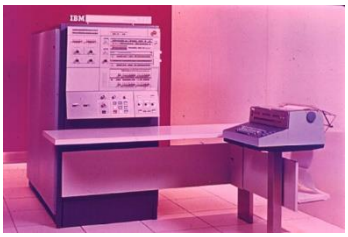


Se hace necesario establecer la diferencia entre la multiprogramación y el multiprocesamiento. El término multiprocesamiento se utiliza cuando se procesan varios programas, en forma simultánea, en CPU diferentes que se relacionan entre sí. A partir de la 3a generación, es posible la gestión denominada teleproceso o procesamiento a distancia. El teleproceso es un sistema de carga directa, con la ventaja de que los datos pueden ingresar y egresar por terminales remotas según una prioridad dada. Durante este período se desarrollan los primeros sistemas interactivos, que permiten que el usuario intervenga en la actividad del procesamiento. El diálogo se gestiona utilizando un terminal con menús o cuestionarios que el sistema formula y el usuario responde.

Cuando un sistema admite la atención de múltiples usuarios se denomina multiusuario. Todo sistema multiusuario es multitarea y, además, se ocupa de la administración de los recursos asignados a los usuarios.

Por último, en esta etapa se desarrolla el concepto de máquina virtual, que simplifica la labor del programador, ya que éste no conoce más que una máquina ficticia creada y controlada por el sistema operativo, que no presenta limitaciones de configuración (sobre todo en capacidad de memoria central), provocada en gran medida por la actividad compartida de varios usuarios.

Ejemplos de máquinas de esta generación:



IBM 360, la cual tiene terminales bobas, un Software compatible con IBM 7094, 1401 entre otros y aparece el byte de 8bits. e. El 360 fue el éxito de la década y consolidó a IBM como el dominante absoluto en las ventas de computadores, con una cuota de mercado por encima del setenta por ciento. El Sistema/360 fue la primera familia de computadores de la historia que se planeó. La familia abarcaba un amplio rango de prestaciones y precios. Este, no solamente dictó la carrera hacia el futuro de IBM, sino también tuvo un profundo impacto en toda la industria. Muchas de sus características se han convertido en un estándar para otros grandes computadores.

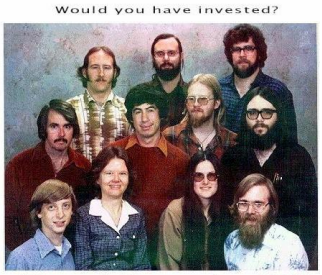
DEC PDP-8 es primer minicomputador, y el cual no necesita una habitación con aire acondicionado. Lo bastante pequeño para colocarlo en una mesa de laboratorio. No podía hacer todo lo que hacían los grandes computadores, pero a 16 000 dólares era suficientemente barato para que cada técnico de laboratorio tuviera uno.

Fundación Intel (1968) La compañía comenzó fabricando memorias antes de dar el salto a los microprocesadores. Hasta los años 70 fueron líderes gracias al competitivo mercado de las memorias DRAM, SRAM y ROM. El 15 de Noviembre de 1971 lanzaron su primer microprocesador: el Intel 4004 para facilitar el diseño de una calculadora. En lugar de tener que diseñar varios circuitos integrados para cada parte de la calculadora, diseñaron uno que según un programa almacenado en memoria podía hacer unas acciones u otras, es decir, **un microprocesado**r. Sin embargo, el ordenador personal no pasó a ser tal hasta la aparición de IBM, en el mercado. Algo que sucedió en dos ocasiones en los meses de junio de 1978 y de 1979. Fechas en las que respectivamente, hacían su aparición los microprocesadores 8086 y 8088, que pasaron a formar el denominado IBM PC, que vendió millones de unidades de ordenadores.

Laboratorio Bell desarrolla el lenguaje C (1972)

Appel I (1976) fue uno de los primeros computadores personales, y el primero en combinar un microprocesador con una conexión para un teclado y un monitor. Fue diseñado y hecho a mano por Steve Wozniak originalmente para uso personal. Un amigo de él, Steve Jobs, tuvo la idea de vender el computador.

Apple II (1978) tenía una arquitectura de 8 bits basada en el procesador 6502. Fue el primer microcomputador producido a gran escala. Fue popular entre los usuarios caseros, y fue ocasionalmente vendida también a usuarios de negocios. El Apple II fue uno de los integrado, común en los primeros computadores personales pero no muy común hoy en día. La Apple II tenía color y gráficos de alta resolución, capacidad de sonido y el lenguaje de programación Basic, inicialmente Integer BASIC, luego Applesoft BASIC. Comparado con microcomputadores anteriores, estas características estaban bien documentadas y eran de fácil aprendizaje. La Apple II despertó el comienzo de la revolución de la computadora personal, ya que su mercado meta era las masas en vez de únicamente ingenieros y aficionados.



Microsoft Corporation, 1978

La **Historia de Microsoft** comienza el 4 de abril de 1975, cuando fue fundada por Bill Gates y Paul Allen en Albuquerque. Después de leer el 1 de enero de 1975, en la revista Popular Electronics, que la compañía MITS había creado el Altair 8800, Bill Gates llamó a los creadores del nuevo microordenador y ofreció demostrar la implementación del lenguaje de programación BASIC para el sistema.5 Después de desarrollarlo en las ocho semanas siguientes junto con Allen, el intérprete funcionó en la versión parcial del programa y MITS acordó distribuir Altair BASIC.

La primera oficina internacional de la compañía fue fundada el 1 de noviembre de 1978 en Japón, llamada ASCII Microsoft (ahora llamado Microsoft Japón). El 1 de enero de 1979 la compañía se trasladó de Albuquerque a Bellevue, Washington. Steve Ballmer inauguró la sede de la compañía el 11 de junio de 1980; éste más adelante sustituiría a Gates como gerente.

GENERACIÓN 4 (1980)

- Usan LSI (large scale integration).
- Intel 8080 (8 bits) -> IBM PC (1980) con DOS.
- Intel 80286 (1983), 80386 (1985) y 80486 (1989).
- GUI (graphical User Interface) -> Macintosh
- Microsoft adopta GUI y desarrolla Windows (sobre DOS)
- Microsoft saca Windows 95 como sistema operativo, luego Windows 98 (basado en 16 bits), Windows NT (32 bits) con ideas de VAX VMS.
- Windows NT 5.0 se transformó en Windows 2000
- Windows 98 se transformó en Windows Me.
- Windows XP
- UNIX se mantiene más fuerte en computadores potentes
- Se basa en chips RISC de alto desempeño.
- Pentium Liunx es una alternativa a Windows
- Unix saca X Windows basado en GUI
- Se introducen sistemas operativos en red (varios computadores conectados) y sistemas operativos distribuidos (múltiples procesadores).

La tecnología aplicada en la fabricación de circuitos pertenece a la clasificación LSI o Large Scale Integration -escala de integración grande-, que permitió incluir una CPU completa en una sola pastilla, que se denomina microprocesador. En esta etapa el procesamiento se realiza en mayor medida en tiempo real Considerando los sistemas interactivos, se hace posible la consulta y la actualización de datos, así como el acceso a grandes bancos de datos que utilizan, incluso, unidades inteligentes distribuidas en redes.

Se desarrollan nuevas utilidades a partir de la sencilla comunicación usuario-máquina: enseñanza asistida por computadora, consultas telefónicas que entregan una respuesta oral al usuario, regulación automática de semáforos, etc.

Hasta esta etapa, los avances tecnológicos estuvieron concentrados en lograr mejorar el hardware de la computadora y así obtener equipos más pequeños, menos costosos y más rápidos. A partir de entonces, también se puso atención en la necesidad de mejorar el software para que permitiera una mayor velocidad de procesamiento.

Ejemplos de máquinas de esta generación:

Commodore 64 (1982) es una computadora doméstica de 8 bits presentando 64 kilobytes (65.536 bytes) de RAM, con gráficos y sonido muy por encima de otros equipos contemporáneos. Contaba con una paleta de 16 colores y un intérprete BASIC. Aceptaba la conexión directa de periféricos sin necesidad de una interfaz de conexión, (como alguno de sus más directos competidores) incorporando dos puertos de conexión de mandos de juego (joysticks), salida a televisión, salidas de vídeo compuesto y audio. Algunos cartuchos incorporaban lenguajes de programación como COBOL, o un BASIC más avanzado, o expansión de RAM, más algunas utilidades para congelar los juegos y poder copiarlos.

Disco Duro Seagate 40 MB (1978)

Disco Duro Memorex 10MB (1983)

AT&T y Laboratorio Bell desarrollan C++ (1983)

IBM PC/AT (1983)

Sony introduce el CD (1984)

Commodore Amiga (1985)

Macintosh (1984)

X Windows para UNIX (1984)

Windows 95 (1995)

GENERACIÓN 5 -> COMPUTACIÓN CUÁNTICA

- Tiene su origen en uno de los principios fundamentales de la mecánica cuántica, conocida como principio de superposición de estados.
- La unidad de información es el cubit, o bit cuántico.
- Un Cubit puede tener múltiples estados simultáneamente en un instante determinado, así reduciendo el tiempo de ejecución de algunos algoritmos de miles de años a segundos.
- Trabajan con compuertas cuánticas diseñadas para llevar a cabo operaciones lógicas cuánticas.
- Para evitar que la energía termina introduzca perturbaciones que provoquen un cambio de estado espontaneo en los qubits es necesario mantener los ordenadores cuánticos a una temperatura cercana al cero absoluto.
- Una computadora clásica primero debe analizar un camino, después ir al siguiente y así hasta encontrar la solución al problema. En cambio, una cuántica puede estar en todos los estados diferentes a la vez, recorrer todos los caminos a la vez, con lo que encontrará la solución de forma casi instantánea,
- El entrelazamiento, es la generación de pares de qubits entrelazados, lo que significa que dos miembros de un par existen en un solo estado cuántico. Al cambiar el estado de uno de los qubits se puede predecir el cambio de su par. Este fenómeno permite que dos partículas separadas entre sí por una distancia monstruosa sean capaces de “comunicarse” sin que exista nada, ningún canal de transición entre ambas. A es te fenómeno lo llamamos entrelazamiento cuántico.
- Aplicaciones a las que apunta: IA, Simulación Biomédica, Sistemas Financieros, Transmisión y encriptación de datos, etc.

En los procesadores actuales se ejecutan varias etapas de las instrucciones en paralelo y éste es un ejemplo de avance respecto de las máquinas anteriores, que procesaban cada etapa de una instrucción en un orden secuencial estricto. Sin embargo, las expectativas de esta nueva generación apuntan a que los cambios creen una revolución profunda y no una mejoría gradual de las computadoras anteriores. Se resalta

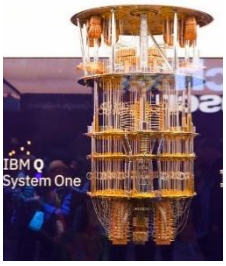
la necesidad de una interfaz hombre/máquina de elevada inteligencia, que demuestre cualidades de juicio similares a las del hombre.

Las características de las máquinas de 5a generación dependen de proyectos ya en marcha, en los que se diferencian tres áreas fundamentales de desarrollo: integración de microcircuitos en muy alta escala, nuevos métodos de procesamientos de datos, nuevas arquitecturas y avances en el diseño de memorias con mayor capacidad y velocidad.

El procesador de conocimiento es el corazón de esta máquina, el lenguaje de este núcleo es de inferencia lógica, de manera de dotarlo de capacidad de resolución de problemas. El procesador de base de datos representaría el centro neurológico del procesador de conocimiento. La complejidad de los proyectos en investigación supone la concreción de muchos proyectos individuales que no terminarán su evolución al mismo tiempo.

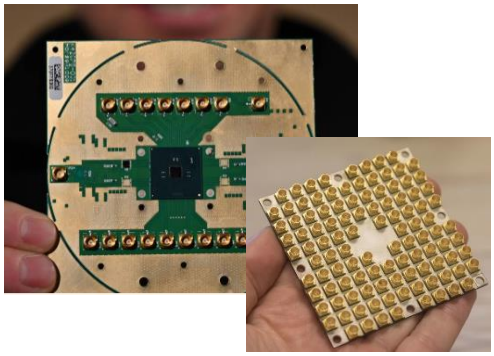
Se sustituyen lenguajes de alto nivel por otros de mayor capacidad para la manipulación de símbolos y recursos lógicos, semejantes a los precursores utilizados en los comienzos de la inteligencia artificial, Prolog y Lisp.

Ejemplos de máquinas de esta generación:



IBM Q System One (2019) es la primera computadora cuántica para uso comercial, de negocios e investigación, y ha sido creada por la empresa IBM. Combina tanto computación cuántica como "tradicional" para ofrecer un sistema de 20 qubits que utilizar en investigaciones y grandes cálculos. Esta solución que propone IBM, aunque sea de uso comercial, no puede utilizarse por cualquiera y en cualquier lugar.

D-Wave System. Los ordenadores cuánticos de D-Wave aprovechan la dinámica cuántica para acelerar y habilitar nuevos métodos de resolución de problemas de lógica, optimización discreta, ciencias materiales y la mejora del aprendizaje automático. Para ello, utiliza un proceso llamado Recocido cuántico. El recocido cuántico («quantum annealing») es una forma de computación cuántica para la resolución de problemas de optimización combinatoria que presenta grandes ventajas (speedups) en algunos algoritmos respecto a las implementaciones basadas en el recocido simulado clásico («simulated annealing»). Pero esta técnica no es ventajosa con un gran número de cubits sino se usan técnicas de corrección de errores.



INTEL desarrolla un chip que podría ser clave para la computación cuántica. El denominado como 'Horse Ridge' facilitará la escalabilidad y el uso comercial de esta tecnología llamada a revolucionar la informática. La compañía resume su avance explicando que la fabricación en masa de estos chips les permitirá diseñar, optimizar y probar los ordenadores cuánticos a nivel comercial. Horse Ridge recibe las órdenes a través de radiofrecuencia, por lo que podrá controlar los qubits desde dentro de la cámara criogénica, "traslada las instrucciones a pulsos electromagnéticos que pueden influir en el estado de los qubits"

PROCESAMIENTO DE DATO S Y SISTEMAS DE INFORMACIÓN

Los **datos** son conjuntos de símbolos que representan un objeto concreto o abstracto, mientras que la información es la consecuencia de procesar los datos para que tengan un significado. Así, la **información** permite disminuir la incertidumbre y facilita la toma de decisiones más acertadas. En las máquinas la información descriptiva se encuentra en la memoria constituyendo programas que indican la secuencia de operaciones a seguir para lograr el resultado deseado. También existe *información de control* que permite verificar si las acciones planificadas se realizan correctamente.

DATOS & INFORMACIÓN → *Ambos son representaciones simbólicas*
↓ ↓→ Conjunto de datos que *quien los necesita e interpreta* puede decidir una
Hechos ↓ acción entre varias.
Atributos └→ Puede ocurrir que un mismo mensaje provea información distinta
Valores para dos o más decisiones a tomar.

INFORMACIÓN —→ <u>REPRESENTACIÓN SIMBÓLICA</u>
└→ No es material ni tangible

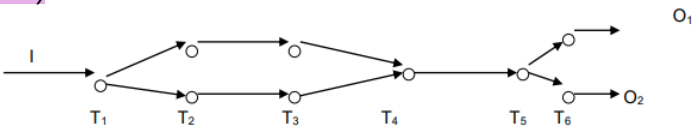
La información puede ser transmitida y almacenada (también puede ser procesada para obtener
Nuevas informaciones)
└→ TRANSMITIR ≠ COMUNICAR
└→ Hay **comunicación** cuando el receptor asigna *el mismo* significado que el que transmite.

Procesar datos implica que se relacionen entre sí o que formen parte de un contexto, o ambas situaciones. Los datos que procesa una computadora no tienen significado para ella, sino que lo tienen para el usuario que los recibe y puede interpretarlos en su propio contexto.

Un **proceso** especifica la actividad interna del sistema y estará representado por una secuencia de instrucciones pertenecientes a uno o varios programas relacionados, de manera que verifique la serie finita de operaciones definida en un algoritmo. Un proceso es un conjunto finito de pasos que deben llevarse a cabo en un cierto orden para resolver un problema. Los procesos pueden ser:

• *Secuenciales:* Operaciones, totalmente ordenadas en el tiempo que se realizan una por vez. Las máquinas con arquitectura Von Neumann realizan el procesamiento siguiendo este modelo

• *Concurrentes:* Hay operaciones que en su ejecución se superponen en el tiempo. Muchos procesadores actuales realizan procesos en forma simultánea, siempre y cuando que no utilicen los mismos recursos (memoria, buses, etc.)



Una **estructura de datos** es un conjunto de datos o símbolos pero organizado respondiendo a relaciones que no existen antes de procesarlos.

Los términos **sistema de información** o **sistema de informática** se utilizan para referirse al conjunto coordinado de elementos, datos y procesos, cuya interacción permite la obtención de la información. En un sistema de información deben coordinarse los procesos y los datos del sistema que administra la computadora y los procesos y los datos del usuario. Siempre todos cooperan entre sí para la elaboración de la información (procesamiento de datos o cómputo de datos).

Los **datos de entrada** son, internamente, una estructura binaria organizada que respeta un convenio dispuesto de antemano, denominado código. Éste permite establecer una **relación** entre dígitos binarios (símbolos de la computadora) y los símbolos que el usuario utiliza para construir datos: números, letras, caracteres especiales, auditivos, gráficos o visuales. La secuencia lógica de pasos se describe con la confección de un programa. Las **salidas** también son información codificada en binario (lenguaje poco amistoso para el usuario) y, para que se represente en símbolos fáciles de comprender, se decodifican. La decodificación convierte el código binario a código de usuario (letras, números, puntos, comas, etc.).

Un dato puede ser una cadena de caracteres o *stríng*, que ingresa por teclado y se ve en el monitor. Un stríng es una unidad formada por caracteres representados según un código alfanumérico preestablecido para esa computadora. Sin embargo, la mayoría de los procesos en un sistema informático toma y vuelca datos organizados en estructuras mayores denominadas archivos.

ARQUITECTURA DE COMPUTADORAS: LOS PRIMEROS CONCEPTOS

La palabra digital implica que el valor de los datos es discreto y limitado y se representa con combinaciones de 0 y 1, o sea, son **máquinas binarias**. Los datos son agrupaciones de bits que, como ya señalamos, al aplicar distintas técnicas de codificación se convierten en números y letras.

Una computadora es un sistema que incluye módulos de hardware y de software. El **hardware** es el conjunto de dispositivos electrónicos y electromecánicos que constituyen la estructura física de la computadora. Es la porción “dura”, lo tangible. En tanto que la parte lógica o “logicial”, se denomina **software,** que es el nombre que identifica al conjunto de programas para procesar datos en un equipo de computación, esto es, su parte lógica. Sin el software, el hardware no podría procesar dato alguno o quedaría limitado a una tarea fija. Un término más, que involucra ambos conceptos, es **firmware** y se utiliza para identificar los dispositivos físicos programados, como puede ser la programación de los múltiples usos que presta un electrodoméstico, o sea, que se fusionan los conceptos de hardware y software (no se borra al desconectar los circuitos de la fuente de alimentación).

Esos **circuitos integrados** constituyen las denominadas memorias ROM (Read Only Memory). Los programas y tablas de datos contenidos no pueden cambiarse. Solo pueden leerse. Los programas incluidos en firmware son de uso muy frecuente, tales como programas iniciadores de funcionamiento de máquina, de verificación de funcionamiento del hardware, programas traductores, etc.

En las computadoras digitales las entidades (datos e instrucciones) están constituidas por agrupaciones de bits**. Bit** es el acrónimo de binary digit (dígito binario) y es la unidad mínima de información. Internamente la computadora está compuesta por dispositivos electrónicos que actúan como conjuntos de llaves o interruptores, que permiten el paso de ciertos niveles de tensión. La "vida" de los bits depende de que la computadora esté conectada a una fuente de alimentación, de modo que un bit 1 se puede representar con un nivel de tensión.

Un byte es una combinación de Sbits. Así como el bit es el grano de arena, la unidad más pequeña en la construcción de un dato para procesar en la computadora, el byte, es el ladrillo, el bloque de construcción real de datos. Cuando se hace referencia a medio byte, o 4 bits, se utiliza el término **nibble.**

Las computadoras necesitan procesar pequeños grupos de bytes (2, 4, 8). La unidad de trabajo o procesamiento de CPU se denomina palabra de CPU. Su longitud tiene relación con la longitud de los registros asociados a la CPU y puede tener, o no, la misma cantidad de caracteres que el contenido de una posición de memoria (denominado también "palabra de memoria")

PRIMERA VISIÓN DE UNA ARQUITECTURA TIPO VON NEUMANN

Von Neumann, en 1945, propone los conceptos fundamentales en que se basan las computadoras tal como las conocemos actualmente.

1. **Programa almacenado:** En la memoria de la máquina se encuentran tanto los datos a procesar como los programas con la secuencia de instrucciones para el procesamiento.
2. **Ruptura de secuencia:** Von Neumann concibió la idea de hacer automáticas las operaciones de decisión lógica, dotando a las máquinas de la instrucción de **salto condicional**, teniendo en cuenta el resultado de una operación realizada con anterioridad seguiría una parte u otra del programa.

En una visión muy general un computador se compone de:

1. **Memoria Central:** Almacena los programas y datos
2. **Unidad Central de Proceso:** Ejecuta el programa
3. **Unidades de Entrada/Salida:** Permite el intercambio con el exterior

A veces se llama sistema informático al conjunto ordenador + elementos a él conectados.

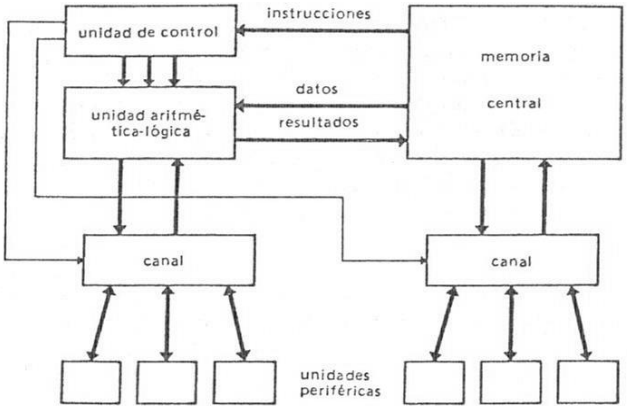


Fig. 2. Esquema general de un ordenador

ARQUITECTURA DE UNA UNIDAD CENTRAL DE PROCESO (CPU)

La **CPU** es el módulo físico más importante. Su capacidad de trabajo determina la capacidad de trabajo de la computadora.

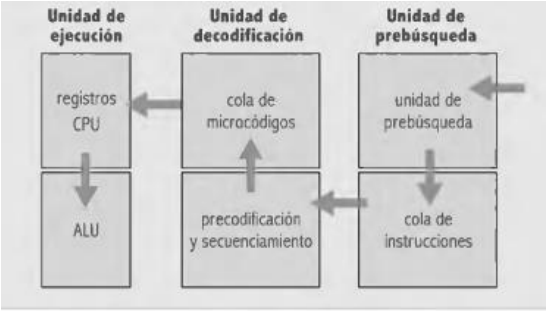


Fig. 1.8. Tres bloques funcionales de una CPU Intel.

La unidad de pre-búsqueda obtiene instrucciones de la memoria principal por medio de una unidad que actúa de intermediario con el colector de datos e instrucciones o "bus" (no dibujada en el esquema). Estas instrucciones se disponen en una cola (de instrucciones). Cada instrucción espera en la cola y es interpretada por la unidad de pre-decodificación, que la traduce a microcódigos. Se denomina microcódigo a un lenguaje de menor nivel que el código de máquina, que permite la representación de las micro-operaciones organizadas de manera secuencial en el tiempo para la ejecución de la instrucción.

La ejecución se realiza en la unidad de ejecución con el soporte de la unidad aritmético-lógica y de los registros de trabajo de la CPU; además, la unidad de ejecución se comunica con los componentes que sean necesarios, aun cuando éstos no pertenezcan a la CPU.

El módulo de cálculo y comparación, denominado unidad aritmético-lógica (**ALU** o Aríthmetic Logic Unit), se encarga de operar los datos que recibe de la memoria y obtener el resultado.

En la CPU también están incluidos **registros** o registers, que forman una pequeña memoria local. Éstos guardan información de manera transitoria para el procesamiento en curso (operandos, direcciones de memoria, etc.)

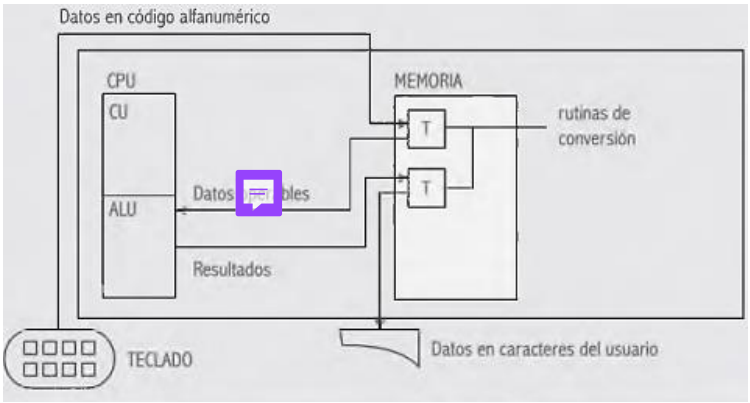
La CPU está asociada a la memoria principal (**PM** o Principal Memory) para obtener las instrucciones y los datos y almacenar los resultados; es de tecnología de semiconductores y también se denomina memoria central, interna o memoria de trabajo.

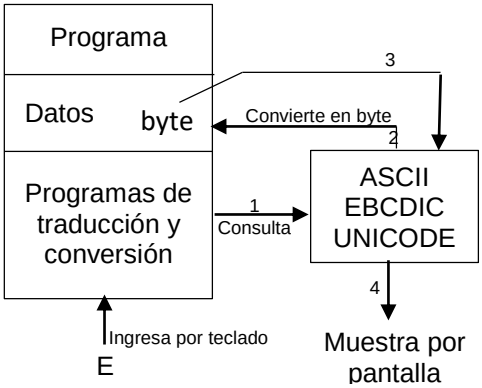
FLUJO DE DATOS DENTRO DE LA COMPUTADORA

Una computadora está constituida por una memoria, que almacena programas y datos en forma temporal, y un órgano ejecutor de estos programas conocido como unidad central de proceso (CPU).

Hay un *"mediador"* entre los programas y los datos que vienen "de afuera" y las necesidades internas para procesarlos. Este vínculo que pasa de lo estándar a lo particular en cada equipo es una rutina de conversión enlazada al código del programa. Las **instrucciones** de un programa como los datos que ingresan en la memoria desde el exterior lo hacen en un código alfanumérico de representación de caracteres, por ejemplo, el ASCII. Las instrucciones son transformadas por un **programa de traducción** (que puede ser un ensamblador, un traductor o un compilador) a código de máquina, que es el que entiende el procesador, para luego ejecutarse. Por lo tanto, aunque los datos ingresen desde el teclado en un código alfanumérico de caracteres, éstos se almacenarán en la memoria de distintas formas, según los requerimientos determinados por la declaración de los datos del programa.

Por otra parte, una computadora **necesita comunicarse** con el medio externo por medio de dispositivos de entrada o salida, y éstos se diseñan para conectarlos a una amplia gama de computadoras diferentes entre sí, lo que induce a pensar que los periféricos utilizan códigos estándar para la representación de información. La forma de ingreso o egreso de datos y programas en una computadora respeta estos códigos de representación de caracteres que abastecen las necesidades de intercambio de información a nivel general. Esto es independiente de las transformaciones que se realizan para su tratamiento interno.





Se denomina **formato** de una entidad binaria a la estructura y la cantidad de bits de un determinado tipo de dato para su tratamiento en la computadora Los formatos pueden ser de longitud fija (byte) o variable (en la cabeza me dice hasta donde es información válida), según la categoría de dato que represente. Es importante destacar que el que interpreta un grupo de bits bajo un formato determinado es un programa.

Tabla 3-1. Distintos tipos de representaciones de datos.

Representaciones alfanuméricas	ASCII UNICODE	
Representaciones decimales (BCD)	BCD puro o natural (*8421*) BCD exceso tres BCD *2421* o AIKEN	
Representaciones binarias	Números enteros	Coma o punto fijo sin signo (enteros-positivos) Coma o punto fijo con signo (enteros) Coma o punto fijo con complemento a la base (enteros) Coma o punto fijo con complemento restringido (enteros)
	Números reales	Coma o punto flotante (entera y fraccionaria) Coma o punto flotante con mantisa normalizada Coma o punto flotante *IEEE P754* Coma o punto flotante "exceso *64*"
Representaciones redundantes (detección de errores)	Códigos de paridad	Paridad vertical o a nivel carácter Paridad vertical o a nivel bloque Paridad entrelazada Código de Hamming

CODIFICACIÓN DE LA INFORMACIÓN

Para poder transmitir información existen dos problemas fundamentales: el hardware y el lenguaje de comunicación. Así como el habla, sería imposible sin lenguajes comunes, la comunicación entre computadoras sería imposible sin coordinación de códigos de caracteres.

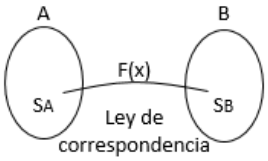
Todas las computadoras digitales actuales usan un **lenguaje binario** para representar la información, internamente. Debido a que algunos de los dispositivos con los que se deben comunicar las computadoras están diseñados para uso humano (específicamente teclados, monitores e impresoras), es importante que estos periféricos utilicen un código de comunicación compatible con la comunicación humana.

Actualmente es frecuente encontrar en los sistemas de cómputo, dispositivos provistos por distintos fabricantes. La posibilidad de conectar estos dispositivos existe únicamente si éstos utilizan un código común para la transmisión y recepción de la información.

Son claras las ventajas de conseguir que todas las computadoras utilicen el mismo código de comunicación. Aun cuando la calidad de los códigos varía enormemente, casi cualquier estándar universal sería mejor que ninguno. Si bien hay códigos que prácticamente son aceptados por todos los fabricantes, aún no existe un estándar que optimice el aprovechamiento tanto de los recursos del hardware como las nuevas teorías sobre codificación e información.

En sentido genérico **código** significa: Sistema de signos y de reglas que permite formular y comprender un mensaje.

La **codificación** es un proceso, u sistema, que permite establecer una **ley de correspondencia** ente dos alfabetos A y B; entre un símbolo de A y un símbolo de B. Consiste en establecer una ley de correspondencia, llamada código, entre las informaciones por representar y las posibles configuraciones binarias, de tal manera que a cada información corresponda una y generalmente solo una, configuración binaria. Es decir, este proceso convierte el alfabeto del usuario a una cadena o configuración binaria entendible para la computadora.



La codificación de la información puede ser:

- **Numérica**: - Conversión de números en diferentes bases
  - Conversión de números en número flotante
- **No numérica**: - Alfanumérica
  - Instrucciones
  - Redundantes -> detección y corrección de errores.

Decodificar: Aplicar inversamente las reglas de su código a un mensaje codificado para obtener la forma primitiva de este.

Decodificación es el proceso inverso al de codificación, se convierte a un código donde la cantidad de símbolos es menor, pero cada una contiene más información.

Transcodificación: Aplicación de un cambio de código a una información ya codificada. Es una codificación con un alfabeto intermedio de puente para llegar al destino. Por lo tanto, hay dos leyes de correspondencia.

Retomemos lo dicho anteriormente de los bits. La condición binaria es la que posee una **calidad bivaluada.** En el sistema binario de numeración esas condiciones están representadas por los dígitos 0 y 1. Se denomina bit al dígito binario, independientemente del valor asignado (0 o 1). Los dígitos binarios llevan, ambos, la misma cantidad de información, ya que la presencia de uno significa la ausencia del otro.

Un proceso fundamental en la codificación binaria es **determinar la cantidad de bits necesarios** para representar las informaciones. De manera que podamos identificar una entre varias posibles. Como un bit puede ser 1 o 0, podremos utilizarlos para seleccionar una información entre dos; con dos bits, una entre cuatro; tres bits una entre ocho; etc. Las posibilidades aumentan como potencias de dos:

Un bit → $2^1 = 2$ elecciones
Dos bits → $2^2 = 4$ elecciones
Tres bits → $2^3 = 8$ elecciones

Si quisiéramos conocer cuántos bits necesitamos para una de ocho situaciones utilizamos logaritmos en base 2: $I = \text{Log}_2 N$

Esto nos indica que cualquier código que utilicemos para representar los números del sistema decimal precisará, como mínimo, 4 Dígitos Binarios (Bits). Cuando el código utilice más dígitos que los que necesita lo denominaremos **código redundante**. Esta redundancia nos sirve para ver que el mensaje llegue a salvo, es decir, sin errores; siempre buscamos la mínima redundancia

Redundancia	8bits

CODIFICACIÓN DE LA INFORMACIÓN NUMÉRICA

Estudiaremos la representación de los símbolos del sistema decimal de numeración mediante símbolos binarios (bit).

CÓDIGOS PONDERADOS

Se denomina código ponderado a aquel que respeta, para la representación de cada dígito decimal, el "peso" que corresponde a cada dígito binario de acuerdo a la posición que ocupa. Ejemplos: BCD (8421), AIKEN(2421), 84-2-1. PERO, en cualquiera de los códigos numéricos (ponderados y no ponderados) un número decimal se codificará cada dígito decimal por separado.

Si bien los símbolos utilizados se corresponden con los del sistema binario, la representación codificada no tiene nada que ver con el sistema de numeración binario.

Dígito Decimal	B.C.D (8421)	Exceso de tres	A I K E N (2421)	8 4 -2 -1
0	0000	0011	0000	0000
1	0001	0100	0001	0111
2	0010	0101	0010	0110
3	0011	0110	0011	0101
4	0100	0111	0100	0100
5	0101	1000	1011	1011
6	0110	1001	1100	1010
7	0111	1010	1101	1001
8	1000	1011	1110	1000
9	1001	1100	1111	1111

CÓDIGOS NO PONDERADOS

En estos códigos la representación de cada dígito decimal es en principio arbitraria o responde a características que no son el "peso" de acuerdo a la posición que ocupan. Ejemplos: Código de GRAY, el cual tiene la particularidad que de un dígito decimal al siguiente cambia siempre un solo dígito por vez.

Construcción de GRAY

I. Se colocan los dos bits (0 y 1) y se traza una línea debajo de ellos (espejo), se copian los bits como si se reflejaran en dicho espejo.

0
1
1
0

II. se completa la siguiente columna con 1 por debajo del espejo y con 0 por encima del mismo.

0 0
0 1
1 1
1 0

III. Se repite la operación de reflejado con los cuatro números obtenidos.

0 0
0 1
1 1
1 0
1 0
1 1
0 1
0 0

IV. Se completa la tercer columna con 1 debajo del espejo y 0 por encima.

0 0 0
0 0 1
0 1 1
0 1 0
1 1 0
1 1 1
1 0 1
1 0 0

V. Se vuelve a realizar la operación hasta completar los diez dígitos.

0 0 0 0
0 0 0 1
0 0 1 1
0 0 1 0
0 1 1 0
0 1 1 1
0 1 0 1
0 1 0 0
1 1 0 0
1 1 0 1

→

0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1

CODIFICACIÓN DE LA INFORMACIÓN NO NUMÉRICA

CODIFICACIÓN DE LOS CARACTERES

Hay unas ciertas condiciones que se deben imponer para esta:

I) La representación debe englobar a las de las cifras en una de las formas descriptas (BCD, 2421, etc.) y permitir distinguir las cifras rápidamente de los otros caracteres.

II) La representación debe permitir añadir nuevos caracteres específicos para una aplicación determinada.

III) En el caso de las transmisiones, la representación debe incluir un sistema de redundancia que permita la detección de errores.

El concepto de **caracter** aparece como la cantidad de BITS necesarios para representar los diferentes símbolos del alfabeto (letras, cifras, signos de puntuación, etc.). De acuerdo al código utilizado cada caracter puede codificarse con un número variable de BITS, pero dentro de un sistema todos los caracteres se representan con el mismo número de bits.

El código **ASCII** se definió inicialmente con 6 bits, esto permitía representar 2^6 = 64 caracteres. Posteriormente la ANSI (Instituto Nacional Norteamericano de Normas) definió un nuevo ASCII (que se mantiene como norma) de 7 bits. Esta nueva definición permite codificar 128 caracteres; haciéndolo más apto, fundamentalmente para la transmisión donde parte de los "caracteres" codifican funciones de control. El código **EBCDIC** creado por IBM utiliza 8 bits para representar cada caracter.

La palabra es un conjunto de caracteres, fijo o variable, según el caso, que la computadora trata como unidad. Es una unidad de información de rango superior al caracter. Generalmente contiene un número entero de caracteres. La palabra es la unidad de información procesada por la máquina.

El término **BYTE** (octeto), se utiliza para describir un conjunto de 8 bits consecutivos que se toman como unidad. El BYTE muchas veces no es práctico para manipular datos en problemas de programación, recurriéndose al concepto de PALABRA (WORD), de acuerdo a la siguiente relación:

1 BYTE	=	8 bits
1 PALABRA	=	2 Bytes
1 DOBLE PALABRA	=	4 Bytes
1 CUDRUPLE PALABRA	=	8 Bytes
1 DECABYTE	=	10 Bytes

El KILOBYTE (KB): Unidad de capacidad de memoria que representa 2^{10} unidades de información, por lo tanto una memoria de 1 KB podrá almacenar 1024 caracteres.

CÓDIGO EBCDIC (EXPANDED BINARY CODE DECIMAL INTERCHANGE CODE)

Este código fue diseñado y utilizado exclusivamente por IBM. Su importancia radica en que sirvió como base para los códigos posteriores normalizados.

Utiliza 8 dígitos binarios para representar cada caracter. La correspondencia entre las informaciones por representar la correspondiente secuencia binaria se encuentra en tablas con distintos formatos. Con respecto a los caracteres alfabéticos y signos de puntuación no tiene características que lo destaquen, salvo que podemos encontrar pequeñas alteraciones al ser utilizados en países con distintos alfabetos.

- **BCD puro o natural:**

Se denomina así porque los pesos en cada bloque coinciden con el valor de los 4 primeros pesos del sistema binario puro.

Tabla 3-4. BCD puro.

Valor decimal	BCD puro 8421	Valor decimal	BCD puro 8421
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

- **BCD empaquetado**

Cada dígito decimal es representado internamente con 8 bits, distribuidos de la siguiente forma:

Zona (4bits)	Digito (4bits)

ZONA: Ocupa los 4 bits de orden superior del Byte, tiene una secuencia binaria fija para cualquier número: $1111_{(2} = F_{(16}$

DIGITO: En este espacio se representa el número decimal codificado en **BCD**

La misma ALU que opera en binario puro puede operar en BCD. Lo que cambia es la forma en que se producen las distintas etapas para llevar a cabo la operación. Estas etapas están definidas por instrucciones en el programa, que permiten la gestión de la operación, considerando como base del sistema decimal y no del binario.

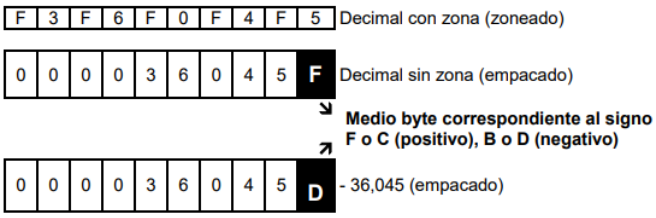
En ASCII los números se representan con octetos divididos en dos partes: zona y dígito. Los números así representados se denominan **"zoneados**"(unpacked). Si estos datos numéricos se operaran en la ALU como fueron ingresados, los bits de zona ocuparían, en principio, un lugar innecesario y provocarían resultados incorrectos al operarlos aritméticamente, lo que complicaría el cálculo.

El producto final al eliminar la zona y agrupar los dígitos en forma reducida se conoce como "**empaquetado**" y cada dígito queda representado empleando sólo 4 bits de "dígito", definido en el byte que representa el carácter numérico.

Un grupo de dígitos BCD constituye un número u operando empaquetado; el signo se incluye en la cabecera o la cola del número, por lo general, en la parte sobrante del octeto que representa el último octeto. Los operandos empaquetados ocupan un número entero de octetos y, por lo tanto, un operando signado, con cantidad par de dígitos decimales, deberá rellenarse con un cero en la posición más significativa.

Un número desempaquetado ocupa un byte, en el que los bits de orden superior son los correspondientes a la zona del código alfanumérico, y se utiliza para el ingreso o para que pueda "mostrarse" al usuario.

EJEMPLO: Queremos representar el número 36045 en EBCDIC



En las representaciones internas de memoria, de los números, el punto decimal (coma) no se representa, queda implícitamente considerado en el lugar correspondiente, nunca forma parte física de la cifra, es el programador quien debe tenerlo en cuenta cuando efectúe la salida de los resultados. Las operaciones "empaquetar" y "desempaquetar" se indican en forma explícita en el programa con instrucciones que las definan.

- **BCD exceso tres**

Se obtiene a partir del BCD puro, sumando un 3 binario a cada cifra decimal. Obsérvese que 0111 (4) es el complemento restringido de 1000 (5), o que 0011 (0) es el complemento restringido de 1100 (9).

Valor decimal	BCD exceso tres
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

"Código autocomplementario"

- **BCD AIKEN o 2421**

También se usa en gran medida por su simetría, que permite hallar con facilidad el complemento de un número, o sea que cuando una operación BCD involucra el complemento restringido de un dígito, éste se obtiene invirtiendo los bits de la combinación de la misma forma que se hace en sistema binario.

Valor decimal	BCD 2421
0	0000
1	0001
2	0010
3	0011
4	0100
5	1011
6	1100
7	1101
8	1110
9	1111

"Código autocomplementario"

CÓDIGO ASCII (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE).

Se trata de un código que utiliza 7 bits para representar cada carácter, permite determinar 2^7= 128 combinaciones distintas, suficientes para establecer una única relación carácter-combinación binaria. Parte de las configuraciones binarias son utilizadas para codificar **funciones de control**.

Zona (3bits)	Digito (4bits)

Tabla 3-2. Tabla de código ASCII (de 7 bits).

	HEX. N°	0	1	2	3	4	5	6	7
HEX. N°	BITS 654 3210	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

Significado en inglés de los caracteres de control de las columnas 0 y 1		
NUL, *null*	VT, *vertical tabulation*	SYN, *synchronous idle*
SOH, *start of heading*	FF, *form feed*	ETB, *end of transmission block*
STX, *start of text*	CR, *carriage return*	CAN, *cancel*
ETX, *end of text*	SO, *shift out*	EM, *end of medium*
EOT, *end of transmission*	SI, *shift in*	SUB, *substitute*
ENQ, *enquiry*	DEL, *data link escape*	ESC, *escape*
ACK, *acknowledge*	DC1, *device control 1*	FS, *file separator*
BEL, bell (*audible signal*)	DC2, *device control 2*	GS, *group separator*
BS, *backspace*	DC3, *device control 3*	RS, *record separator*
HT, *horizontal tabulation (punched card skip)*	DC4, *device control 4*	US, *unit separator*
LF, *line feed*	NAK, *negative acknowledge*	DEL, *delete*

Los primeros 32caracteres pueden parecer “raros", sin embargo, son de gran utilidad porque permiten cierto **control** de las actividades de la computadora. a. Por otro lado, las mayúsculas se consideran distintas a las minúsculas, aun cuando se trate de la misma letra, debido a que su forma de representación simbólica es diferente por completo (A vs. a)

El éxito de este código se basa en que cumple con todas las condiciones impuestas para la codificación de caracteres:

- Utiliza relaciones para establecer el código.
- Los valores correspondientes a las letras de alfabeto y a los restantes caracteres, siguen una secuencia binaria continua, la computadora no tiene que dejar su propio lenguaje binario para realizar operaciones secuenciales con esos caracteres.
- Agrupamiento de las funciones de control, con solo analizar los dos primeros bits de una combinación cualquiera codificada, la computadora puede determinar si se trata de una función de control (dos ceros) o de un caracter (uno de los dos no es cero).

Para la representación de los números, en este código, al igual en el EBCDIC, la representación de la información numérica contempla representar los símbolos del sistema decimal de numeración en forma binaria. Cada dígito se codifica con 7 bits que también se encuentran asociados en dos grupos ZONA Y DIGITO.

La zona es siempre 011 y dígito corresponde a la representación BCD del número.

Aquí surge el mismo inconveniente visto en el código EBCDIC, la dificultad para realizar operaciones matemáticas con los números representados con la ZONA, la solución es la misma que la indicada para el código anterior. Como la mayor parte de las máquinas utilizan 8 bits para cada dígito no surgen conflictos al producirse el EMPAQUE.

- **Código ASC II ampliado**

Si bien el ASCII se concibió como código de 7bits, su uso generalizado dio lugar a una ampliación a causa de la necesidad de agregar caracteres. Ésta es la razón por la que se incluye la tabla ASCII de 8bits, 4bits de zona y 4bits de dígito. Los primeros 128caracteres son los originales. Los segundos 128 son los agregados y aquí se puede hablar de caracteres "no oficiales”, dado que las tablas no guardan uniformidad; en su mayoría, representan caracteres gráficos y no son esenciales.

Tabla 3-3. Tabla de código ASCII ampliado (de 8 bits).

	HEX. N°	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
HEX. N°	BITS 7654 3210	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	NUL		SP	0	@	P	`	p	Ç	É	á	¦	À	Ð	a	=
1	0001		DC1	!	1	A	Q	a	q	ü	æ	í	¦	Á	Ñ	ß	±
2	0010		DC2	"	2	B	R	b	r	é	Æ	ó	¦	Â	Ò	G	=
3	0011		DC3	#	3	C	S	c	s	â	ô	ú	³	Ã	Ó	p	≥
4	0100		DC4	$	4	D	T	d	t	ä	ö	ñ	´	Ä	Ô	S	⌠
5	0101			%	5	E	U	e	u	à	ò	Ñ	µ	Å	Õ	s	⌡
6	0110			&	6	F	V	f	v	å	û	ª	¶	Æ	O		÷
7	0111	BEL		'	7	G	W	g	w	ç	ù	º		Ç	×	t	≈
8	1000	BS	CAN	(	8	H	X	h	x	ê	ÿ	¿		È	Ø	F	°
9	1001	HT	EM	)	9	I	Y	i	y	ë	O	⌐	¹	É	Ù	Θ	∙
A	1010	LF		*	:	J	Z	j	z	è	Ü	¬	°	Ê	Ú	O	·
B	1011	VT	ESC	+	;	K	[	k	{	ï	¢	½	»	E	¦	d	√
C	1100	FF		,	<	L	\	l	|	î	£	¼	¼	Ì	—	B	n
D	1101	CR		-	=	M	]	m	}	ì	¥	¡	½	Í	¦	f	²
E	1110	SO		.	>	N	^	n	~	Ä	₧	«	¾	Î	¦	e	■
F	1111	SI		/	?	O	_	o	DEL	Å	ƒ	»	¿	Ï	'	n	□

REPRESENTACIONES REDUNDANTES

CÓDIGOS DE DETECCIÓN Y/O CORRECCIÓN DE ERRORES.

El cambio de un bit en el almacenamiento o la manipulación de datos origina resultados erróneos. Para detectar e incluso corregir estas alteraciones se usan códigos que agregan "bits redundantes". De todas formas, la cantidad de errores que puedan producirse son mensurables probabilísticamente. El resultado de "medir" la cantidad de "ruido" que puede afectar la información se representa con un coeficiente conocido como tasa de error. De acuerdo con el valor de la tasa de error, se selecciona un código, entre los distintos desarrollados, para descubrir e incluso corregir los errores. Estos códigos reciben el nombre de códigos de paridad.

Un **proceso de comunicación** puede tener lugar e diversas formas e involucra el movimiento de un flujo de información a través de un **canal** definido en el sistema.

Un **sistema de comunicación ideal** se puede representar por tres partes esenciales:

Transmisor →Canal →Receptor

Sin embargo en la realidad no es tan así. En un Sistema de Comunicación Real, un canal de comunicación está sujeto a una diversidad de perturbaciones que resultan en una distorsión del mensaje que se está transmitiendo. Cualquier alteración de esas, se denominan ruido. La forma en la cual el ruido puede aparecer depende de las características particulares de cada canal.

Para mejorar la eficiencia se usan codificadores y decodificadores en ambos extremos, que permitan transformar el mensaje para que el receptor pueda detectar y corregir errores.

Transmisor →Codificador →Canal →Decodificación →Receptor
↑
Ruido

PARIDAD VERTICAL SIMPLE O A NIVEL CARÁCTER

Al final de cada carácter se incluye un bit, de manera que la suma de "unos" del carácter completo sea par (paridad par) o impar (paridad impar). Este tipo de codificación se denomina paridad vertical. Suele acompañar la transmisión de octetos en los periféricos y la memoria. Este código de detección de errores sólo se utiliza si la tasa de error en la cadena de bits que se han de transmitir no es superior a uno; esto es, si hay dos bits erróneos no permite detectarlos.

00110010 | 1 se agrega un uno que permita obtener paridad par
01001011 | 0 se agrega un cero para lograr paridad par

PARIDAD HORIZONTAL A NIVEL DE BLOQUE

Por cada bloque de caracteres se crea un byte con bits de paridad. El bit 0 será el bit de paridad de los bits 0 del bloque; el bit 1, el de paridad de los bits 1 del bloque, y así sucesivamente.

PARIDAD ENTRELAZADA

Utilizando en conjunto el código de paridad vertical con el horizontal se construye el código de paridad entrelazada, que además de detectar errores permite corregirlos.

Supongamos una transmisión de cuatro caracteres en código ASCII de 7bits, con paridad par. El primer grupo indica cómo deberían llegar los caracteres a su lugar de destino, de no haberse producido errores en la transmisión. El segundo grupo indica con un recuadro el bit que tuvo un error en la transmisión.

La forma de corregir el error es invertir el bit señalado como erróneo. Si hay más de un error en la misma fila o columna, quizá se pueda detectar, pero no se podrá determinar con exactitud cuál es el error.

					PH
	0	1	0	1	0
	0	1	1	1	1
	1	1	0	1	1
	1	0	1	1	1
	1	1	1	0	1
	1	0	1	1	1
	0	1	1	0	0
PV	0	1	1	1	1

sin error

					PH
	0	1	0	1	0
	0	$\boxed{0}$	1	1	1
	1	1	0	1	1
	1	0	1	1	1
	1	1	1	0	1
	1	0	1	1	1
	0	1	1	0	0
PV	0	1	1	1	1

con error

Adicionalmente se emplea un bit de paridad cruzada que se calculó a partir de los bits de paridad de filas y columnas. Los chequeos de PV y PH se usan para detectar y corregir los posibles errores que puedan se producir durante la transmisión de datos.

EJEMPLO: Se necesita transmitir 4 caracteres ASCII de 7 BITS con PARIDAD PAR de UNOS.

- La forma de detectar y corregir un error es intersectando fila / columna donde no aplique la paridad.
- Si hubiese mas de un error en la misma fila o columna, será posible detectar pero NO corregirlo
- Se comienza barriendo filas, si la paridad esta bien se pasa a la siguiente, pero si paridad no cumple, se barre columna, donde la columna de error de paridad, en la intersección se invierte el bit y se continua el proceso con la siguiente fila

CONTROL DOS ENTRE TRES

Para transmitir una información cualquiera de "n" bits, se envían 3 veces esos "n" bits, en forma sucesiva. El receptor de la información, al efectuar el análisis de la misma, puede presentársele tres situaciones distintas:

I. Las tres son idénticas. La información se toma como correcta.
II. Dos son iguales y una distinta. El código se comporta como **auto-corrector**, selecciona una de las dos iguales y la toma como correcta.
III. Las tres son distintas. El código se comporta como **auto-detector**, la máquina detecta que hay error pero no puede determinar cuál es la información correcta.

CÓDIGO DE HAMMING

Conceptos básicos:

Código binario: es una representación univoca de las cantidades, de tal forma que a cada una de estas se le asigna una combinación de símbolos binarios.

Distancia entre dos combinaciones binarias: viene dada por el número de bits que hay que cambiar en una de ellas para obtener la otra.

Distancia mínima de un código: es la menor de las distancias entre dos combinaciones binarias cualesquiera pertenecientes a dicho código.

Distancia de Hamming: el código Hamming se genera a partir de una distancia mínima de 3, ya que una distancia de 1 o de 2 es inconveniente para poder generar un código para detectar un error. Si se tuviese distancia 1, no tendríamos valores para saber si hay un error o no; si tenemos una distancia de 2, puede tener datos erróneos que pueden pasar como correctos.

La distancia mínima está dada por la siguiente ecuación: **Dm= 2x +1**

Donde Dm es la distancia mínima de un código para permitir la corrección de datos y x las líneas de datos.

$D_m = 2 * 0 + 1 = 1$

Distancia HAMMING = 1

Mensaje	
0	Válido
1	Válido

Se debe modificar 1 bit

$D_m = 2 * 1 + 1 = 3$

Distancia HAMMING = 2

Mensaje		
0	0	Válido
0	1	Invalido
1	0	Invalido
1	1	Válido

Se pueden detectar errores de 1 bit

(Caso bit Paridad)

$D_m = 2 * 3 + 1 = 7$

Distancia HAMMING = 3

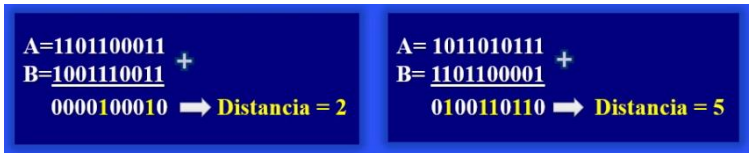
Mensaje			
0	0	0	Válido
0	0	1	Invalido
0	1	0	Invalido
0	1	1	Invalido
1	0	0	Invalido
1	0	1	Invalido
1	1	0	Invalido
1	1	1	Válido

Detecta y corrige errores de 1 bit

Distancia	Longitud Total (n)	Longitud Información (k)	Notación (n,k)	Eficiencia sobre Canal	Detección	Corrección int((m-1)/2)
m	(2^m)-1	(2^m)-1-m	(n,k)	k/n	(m-1)	(m-1)/2
1	1	0	(1,0)	0,0%	0	0
2	3	1	(3,1)	33,3%	1	0
3	7	4	(7,4)	57,1%	2	1
4	15	11	(15,11)	73,3%	3	1
5	31	26	(31,26)	83,9%	4	2
6	63	57	(63,57)	90,5%	5	2
7	127	120	(127,120)	94,5%	6	3
8	255	247	(255,247)	96,9%	7	3
9	511	502	(511,502)	98,2%	8	4
10	1023	1013	(1023,1013)	99,0%	9	4
11	2047	2036	(2047,2036)	99,5%	10	5
12	4095	4083	(4095,4083)	99,7%	11	5

Se denomina **Distancia de Hamming entre dos códigos** de igual longitud, al número de símbolos que se deben alterar para pasar de una configuración valida a otra. Es decir, el número de bits en que difieren dos configuraciones binarias validas de igual longitud.

Se puede calcular la distancia sumando ambas cadenas pero sin considerar el acarreo y finalmente contando la cantidad de 1 obtenidos:



Este código permite detectar y corregir los errores producidos en una transmisión con sólo agregar p bits de paridad, de forma que se cumpla la siguiente relación: $2^p \geq i + p + 1$ donde i es la cantidad de dígitos binarios que se han de transmitir y p es la cantidad de bits de paridad.

Si consideramos que tenemos **i dígitos de información** y **p bits de control** y con la premisa que los casos que se pueden presentar son de ningún error o un solo error por vez, tendremos i + p + 1 condiciones que deben identificarse usando los p bits de control de paridad.

Mediante la expresión anteriormente mencionada, podemos construir la siguiente tabla:

i	1	2	3	4	5	6	7	8	9	10	11	12	.....	Bits de información
p	2	3	3	3	4	4	4	4	4	4	4	5	.....	Bits de control
n	3	5	6	7	9	10	11	12	13	14	15	17	.....	Bits del mensaje

Los bits de control se colocan en las posiciones que corresponden a las potencias de dos. Para determinar el valor que tomará cada bit de paridad se construye una tabla de combinaciones binarias de tantas columnas como la cantidad de bits de paridad determinada para la codificación.

	p_2	p_1	p_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

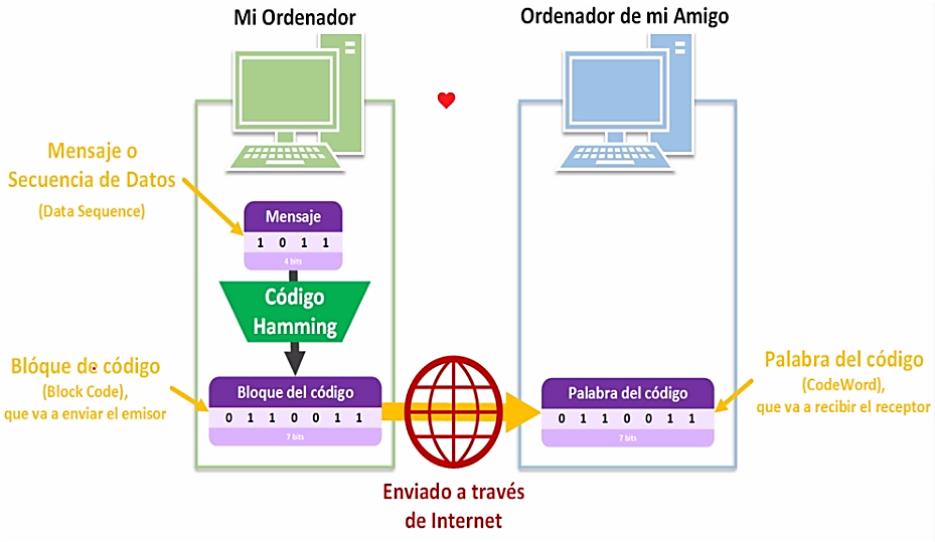
Cada bit de paridad controla la posición que coincide con un 1 en su columna en la tabla binaria. Es decir:
p_0 controla las posiciones: 1, 3, 5 y 7
p_1 controla las posiciones: 2, 3, 6 y 7
p_3 controla las posiciones: 4, 5, 6 y 7

Control y corrección.

Cuando el mensaje es enviado, el receptor calcula el valor de los bits de control de la siguiente forma:

$$c_0 = 1 \oplus 3 \oplus 5 \oplus 7$$
$$c_1 = 2 \oplus 3 \oplus 6 \oplus 7$$
$$c_2 = 4 \oplus 5 \oplus 6 \oplus 7$$

Si el bit de control es igual a 0, entonces es correcto; si es 1 es incorrecto. La posición del error se determina según los valores obtenidos en los bits de control ordenados en forma decreciente, es decir c2, c1 y c0. Si todos los bits de control son igual a 0, entonces no se detectó un error en el mensaje.



PRINCIPIO DE FUNCIONAMIENTO DEL COMPUTADOR

MEMORIA CENTRAL O PRINCIPAL

El programa a ser ejecutado debe encontrarse en memoria central, en el momento que se va a ejecutar. Almacena fundamentalmente dos tipos de información:

1. Instrucciones del programa (o informaciones descriptoras del tratamiento) que la maquina deberá ejecutar.
2. Datos (dicho a veces operandos o informaciones a tratar), con los que trabajará la máquina, de acuerdo a lo indicado en el programa.

Estos dos tipos de información tienen su correspondencia física en dos unidades peculiares de la máquina: **la unidad de control**, también llamada **unidad de instrucciones** o **unidad de gobierno**, para las instrucciones, y la ALU o **unidad de proceso**, para los datos.

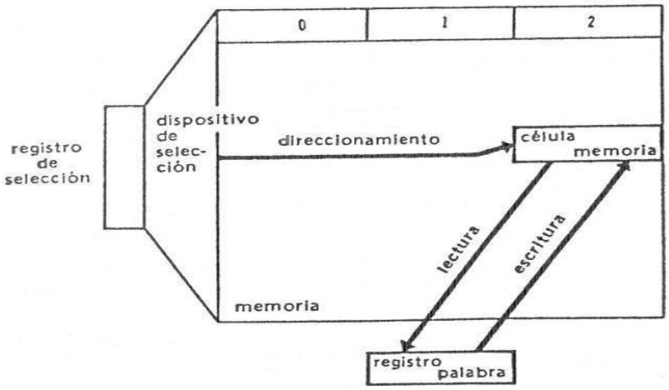


Fig. 3. Lectura y escritura en memoria

ALU

Manejada por la unidad de control (CU), ejecuta las operaciones con los datos almacenados en la memoria principal.

Para pedir al ordenador una operación aritmética, por ejemplo una suma, la instrucción debe facilitarle las siguientes informaciones:

1. La clase de operación a realizar, en este caso una suma, es el papel del **código de operación**.
2. La dirección de la célula de memoria que contiene el primer dato, o **primer operando**.
3. La dirección de la célula de memoria que tiene el **segundo operando**.
4. La dirección de la célula de memoria donde quiere almacenarse el resultado.
5. La dirección de la célula de memoria de la siguiente instrucción.

De acuerdo a como se resuelvan las cuestiones anteriormente enunciadas, tendremos 4 formatos de instrucción:

1. **Instrucciones de 4 direcciones**

Código de Operación	Dirección 1er Operando	Dirección 2do Operando	Dirección Resultado	Dirección Siguiente Instrucción

2. **Instrucciones de 3 direcciones**

Código de Operación	Dirección 1er Operando	Dirección 2do Operando	Dirección Resultado

Cuando se utiliza este formato de instrucción se supone que la siguiente instrucción se encuentra almacenada en la dirección de memoria siguiente a la que se está ejecutando.

La figura a representa una ALU capaz de ejecutar la operación anterior (de tres direcciones), la cual está rodeada de 3 registros donde se memorizan los dos operandos y el resultado. La instrucción de suma necesita cuatro accesos a la memoria central, que permiten sucesivamente buscar la instrucción, después el 1er operando, después el 2do operando y por último, almacenar el resultado. A las máquinas que utilizan este tipo de instrucción se les llama máquinas de tres direcciones.

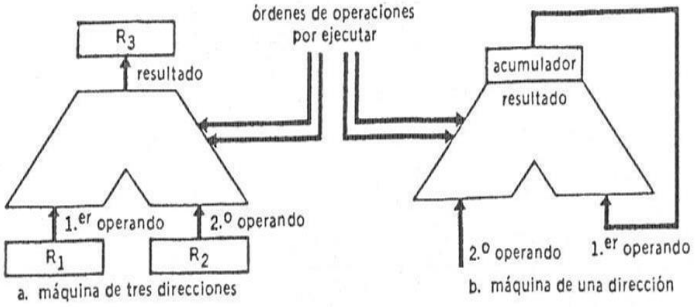


Fig. 4. Unidad aritmética-lógica

3. **Instrucciones de 2 direcciones**

Código de Operación	Dirección 1er Operando	Dirección 2do Operando

En las máquinas que utilizan este formato de instrucción el resultado generalmente se guarda en la dirección del primer operando. Es importante que el programador tenga en cuenta esto ya que ese operando se perderá.

4. **Instrucciones de 1 dirección**

Código de Operación	Dirección del Operando

Abacus es una máquina de una dirección. Su ALU posee un registro particular denominado **Acumulador**, que contiene tanto el 1er operando como el resultado. Esta característica permite instrucciones de una sola dirección: la del segundo operando.

En estas máquinas la operación de SUMA necesita 3 instrucciones para:

1. Cargar el primer operando en el Acumulador.
2. Sumar el segundo operando con el contenido del Acumulador.
3. Almacenar en memoria central el contenido del Acumulador.

Cada una de estas 3 instrucciones comportara un código de operación y una dirección:

	Código de Operación	Dirección
(1)	Carga	Dirección del 1er operando
(2)	Adición	Dirección del 2do operando
(3)	Almacenamiento	Dirección del resultado

La ALU esquematizada en la figura b, donde el acumulador sustituye a los registros R_1 y R_3 de la figura a. El segundo operando puede almacenarse durante la operación en el registro de palabra asociado a la memoria. Este es el caso del Abacus.

5. Instrucciones sin dirección

Este caso se presenta fundamentalmente en las máquinas con MICROPROCESADOR en las cuales la longitud de la palabra es generalmente de 1 byte y se ocupa íntegramente para el código de operación, la dirección del operando se encuentra en la dirección o direcciones siguientes al de la instrucción.

UNIDAD DE CONTROL

Esta unidad se encarga de extraer y analizar las instrucciones de la memoria central, luego establece las conexiones eléctricas correspondientes dentro de la ALU y extrae los datos de la MC implicados por la instrucción. Por último ordena a la ALU el tratamiento de los datos extraídos.

Eventualmente, suele almacenar el resultado en la memoria central. Básicamente consta de dos registros:

1. **Contador de Instrucciones** o contador de programa o contador ordinal: contiene la dirección de la próxima instrucción por ejecutar.
2. **Registro de Instrucción:** contiene la instrucción extraída de la memoria. El registro de instrucción de Abacus tiene 2 partes: una para el código de operación, que define el tipo de instrucción a ejecutar (suma, multiplicación, salto, etc.) y otra parte, que contiene la dirección del operando.

Otro elemento fundamental de la unidad de control es un órgano llamado **generador de secuencias** o **secuenciador**, que es el encargado de analizar el código de operación de la instrucción, y en base a ello distribuir las microórdenes al conjunto de las unidades del ordenador (memoria, ALU, etc.) para hacer ejecutar las distintas Fases de la instrucción.

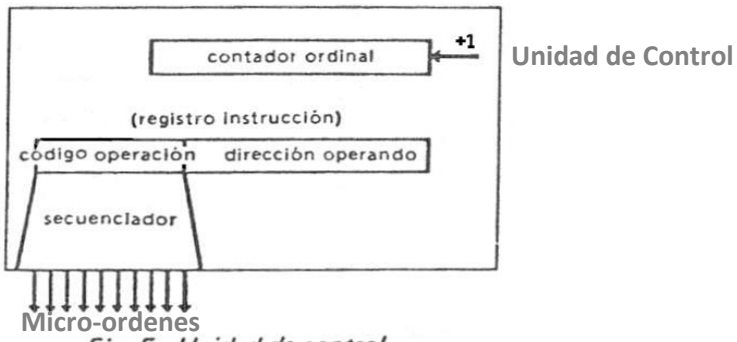


Fig. 5. Unidad de control

La unidad de control y unidad aritmética forma un todo en la mayoría de los ordenadores, que se suele denominar **unidad central** o **unidad central de proceso** o **procesador central**.

LAS UNIDADES PERIFERICAS

Son los elementos que permiten a la máquina el intercambio con el mundo exterior. Podemos citar: unidades de cinta, unidades de disco (rígido, flexible, óptico), teclados, monitores, impresoras, conversores analógico/digitales o dígito/analógicos, módem, etc.

Las memorias auxiliares: que sirven como soporte de almacenamiento de gran capacidad y medio de comunicación en el interior del sistema. Están representadas fundamentalmente por cintas magnéticas, discos magnéticos y ópticos.

La mayoría de estas unidades constan de dos partes:

1. Una parte electrónica, llamada **controlador** o *unidad de control del periférico*.
2. Una unidad electromecánica que, gobernada por la primera, lee o escribe informaciones.

Las unidades periféricas se conectan, bien a la unidad central, bien directamente a la memoria a través de unidades especializadas en la gestión de las transferencias de información. Estas unidades de intercambio se llaman canales.

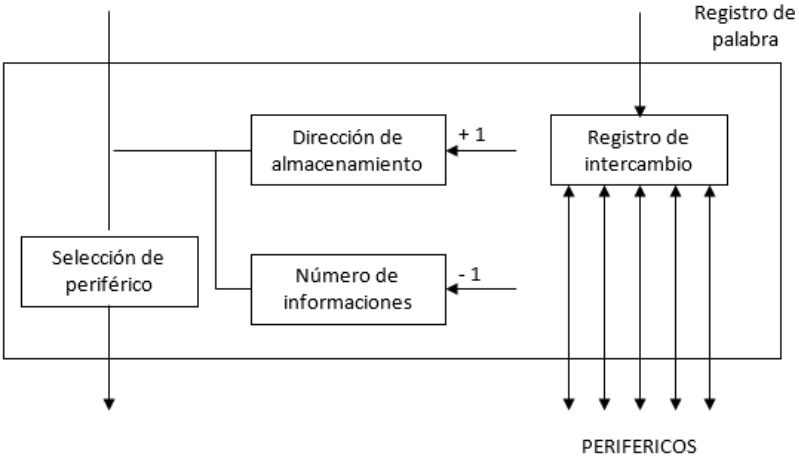
Como se mencionó anteriormente la unidad de control gobierna la ejecución de las operaciones pedidas por el programa. Si la operación es un cálculo, es la ALU quién lo realiza, si es una transferencia de informaciones con el exterior (instrucción de entrada/salida), se cede el control a un canal.

EL CANAL

Cuando se efectúan intercambios entre el exterior y la máquina o viceversa, actúa la unidad denominada CANAL; dispositivo especializado para gestionar las operaciones de entrada – salida (I/O).

En las máquinas actuales, las transferencias de información entre el procesador central (CPU) y los periféricos pueden realizarse en forma “simultánea” con el desarrollo de un programa de cálculo. Esta simultaneidad es posible a causa de la gran diferencia de velocidad que existe entre los periféricos y el interior de la máquina (CPU y MC). Por consiguiente los canales o INTERFACES DE ADAPTACIÓN Y CONTROL DE PERIFERICOS (IACP), deben adaptar esas disímiles velocidades, para lo cual cuentan con registros especializados para este fin.

Las informaciones producidas, como resultado del procesamiento, son almacenadas en memoria central en forma secuencial. Para inicializar tal transferencia, especiales instrucciones de entrada – salida deben suministrar al canal, por una parte, la dirección de la unidad periférica implicada y, por otra, la dirección para el almacenamiento de la primera información y el número de informaciones por transferir. En lo sucesivo, el canal se ocupara totalmente de la gestión de la transferencia; por cada información transferida, añadirá 1 a la dirección de almacenamiento y restara 1 al número de informaciones por transferir. El canal advertirá a la unidad de control en el momento en que todas las informaciones hayan sido transferidas.



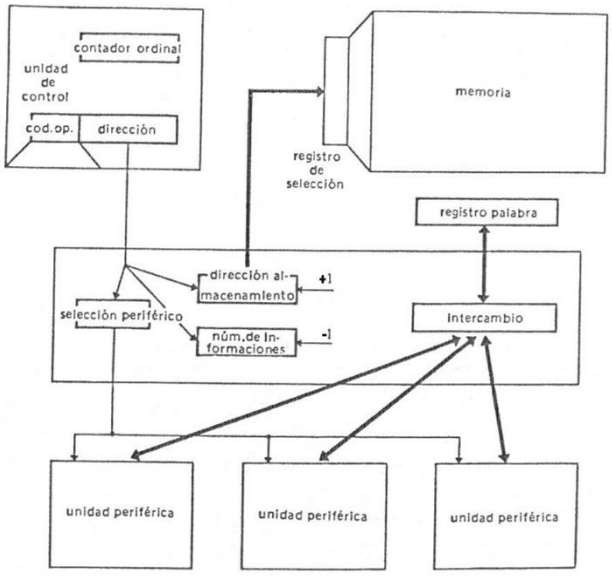


Fig. 10. Operaciones de entrada-salida

PROGRAMA

Se trata de un conjunto de instrucciones, que deben encontrarse en la memoria central, almacenadas secuencialmente o no; esto nos indica que las instrucciones pueden estar o no en direcciones sucesivas de memoria. En este primer análisis supondremos que las instrucciones se encuentran almacenadas en forma sucesiva en memoria. Esquemáticamente dividiremos a las instrucciones **en tres clases**:

1. **Instrucciones de procesamiento:** sobre operandos contenidos en memoria. Operaciones aritméticas, lógicas o de almacenamiento – escritura en memoria.
2. **Instrucciones de ruptura de secuencia.**
3. **Instrucciones de intercambio** entre el medio exterior y el ordenador.

DESARROLLO DE UNA INSTRUCCIÓN DE PROCESAMIENTO

El desarrollo de una instrucción de procesamiento en un computador de una dirección, como Abacus, puede descomponerse en tres fases:

1. Fase de búsqueda y análisis de la instrucción
2. Fase de búsqueda y procesamiento del operando
3. Fase de almacenamiento del resultado.
4. Fase de preparación para la próxima instrucción.

- **Búsqueda de la Instrucción:** La Unidad de Control ordena la transferencia del contenido del contador ordinal (es decir, la dirección de la instrucción por ejecutar) al registro de selección de memoria, y envía a la memoria una orden de lectura. Una vez terminada la operación de lectura, la instrucción queda disponible en el registro de palabra. Entonces la unidad de control ordena la transferencia del contenido de este registro al de instrucción. Los circuitos especializados de la unidad de control pueden ya analizar el código de operación de la instrucción. Esta primera fase es común a todos los tipos de instrucción.

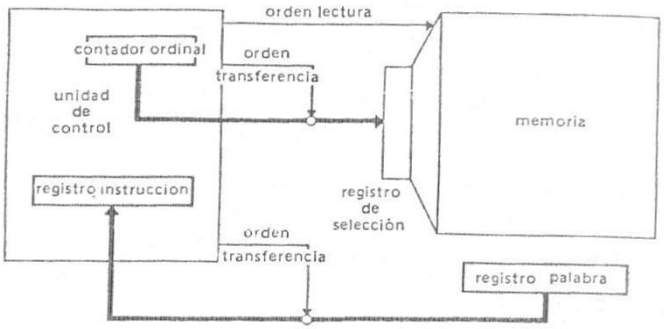


Fig. 6. Fase de búsqueda de la instrucción

- **Búsqueda y Procesamiento del Operando:** Una vez analizado el código de operación de la instrucción, la unidad de control sabe que se trata de una instrucción de procesamiento, con búsqueda previa del operando. La dirección del operando se encuentra en la zona de dirección del registro de instrucción. La unidad de control ordena su transferencia al registro de selección de memoria y acto seguido ordena una operación de lectura en la memoria. Al finalizar dicha operación, el operando buscado queda disponible en el registro de palabra. La unidad de control posiciona los circuitos de la ALU para realizar el procesamiento solicitado por el código de operación y ordena la transferencia del operando a la ALU. El resultado del procesamiento queda almacenado en el acumulador. Obsérvese que un posible procesamiento pudiera ser simplemente una transferencia del operando al acumulador.

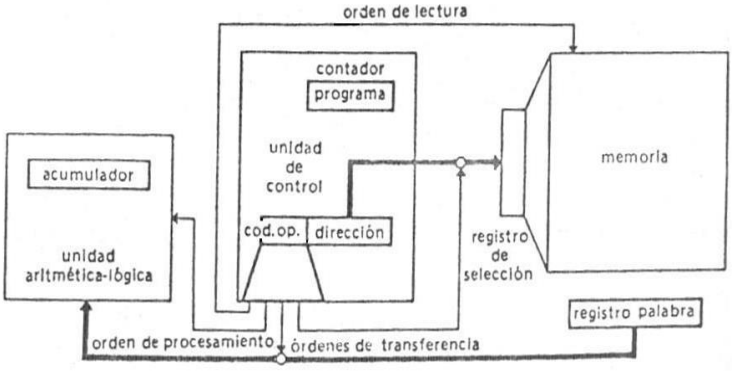


Fig. 7. Búsqueda y procesamiento del operando

• **Almacenamiento del Resultado:** La dirección de almacenamiento del operando se encuentra en la zona de dirección del registro de instrucción: la unidad de control ordena su transferencia al registro de selección de la memoria. El operando por almacenar está en el acumulador: la unidad de control ordena su transferencia al registro de palabra de memoria. Luego de esto el secuenciador emite la orden de escritura y el resultado queda almacenado en memoria central.

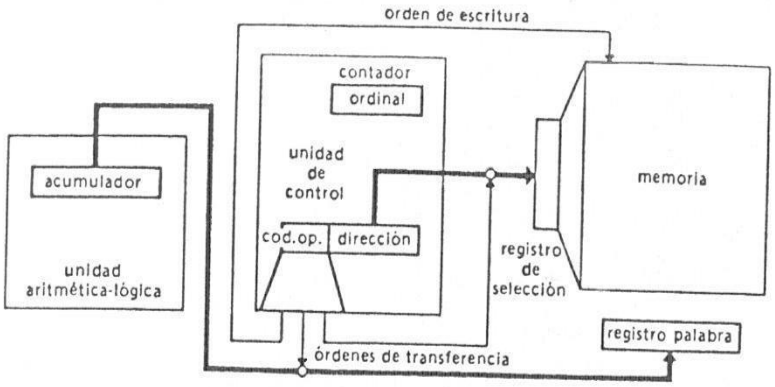


Fig. 8. Almacenamiento del operando

• **Preparación para la Próxima Instrucción:** Consiste en aumentar en una unidad el contenido del Contador Ordinal de la unidad de control, de manera que contenga la dirección de la próxima instrucción. Esta operación se realiza luego del análisis del código de operación de la instrucción, ya que el incremento se producirá automáticamente para todas las instrucciones que no sean de ruptura de secuencia.

INSTRUCCIÓN DE RUPTURA DE SECUENCIA

Este tipo de instrucciones, también llamado instrucción de bifurcación o de salto, permite modificar el desarrollo secuencial del programa. La ruptura de secuencia puede ser **condicional** o **incondicional**.

El salto puede ser **incondicional**, la dirección de la siguiente instrucción está contenida en la propia instrucción de salto y es transferida al contador ordinal.

En el caso de salto **condicional** esta no tendrá efecto más que si se satisface una determinada condición, normalmente relacionada con el contenido del acumulador; si la condición no se satisface, el programa continuará en secuencia. El código de operación de la instrucción establece la condición y en la zona de dirección de la instrucción indica el emplazamiento de la próxima instrucción por ejecutar en el caso de satisfacerse la condición. Si la respuesta de la ALU es que la condición se ve satisfecha, la unidad de control ordena la transferencia de la dirección, contenida en la instrucción, al contador ordinal e inhibe la suma de una unidad al contador ordinal, en caso contrario, se ordena incrementar en 1 el contador.

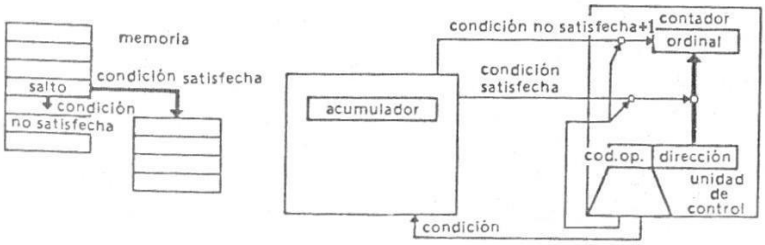


Fig. 9. Instrucción de salto condicional

INSTRUCCIÓN DE INTERCAMBIO CON EL EXTERIOR

Para que el canal pueda encargarse, en forma autónoma, de las transferencias desde y hacia el exterior, es necesario que la CU le suministre la dirección del periférico implicado, la dirección de la primera información y la cantidad de informaciones por transferir. Luego el canal se ocupará totalmente de la gestión, como se indicó cuando se describió el canal.

LAS INTERRUPCIONES

Las **interrupciones** son órdenes que dimanan del medio exterior y que piden al ordenador ejecutar un programa asociado a la orden. El programa en curso se ve interrumpido para permitir la ejecución del programa solicitado por la interrupción, considerado ahora como prioritario. Acabado este último, se reanuda la ejecución del programa interrumpido. Es gracias a las interrupciones, por poner un ejemplo, como los canales avisan a la unidad de control que una operación de entrada – salida ha llegado a su fin.

CONFIGURACION DE UN SISTEMA INFORMATICO

Se llama configuración de un sistema de procesamiento de la información a la lista, posiblemente acompañada de sus características, de las unidades que lo componen, y la forma en que se deben interconectar.

COMPUTADORES DIGITALES Y SISTEMAS DIGITALES

Los computadores se usan para cálculos científicos, procesamientos de datos comerciales y de negocios, control de tráfico aéreo, dirección espacial, campo educacional y en muchas otras áreas, también cabe aclarar que los computadores digitales han hecho posible muchos avances en distintas áreas y que no se hubiesen podido lograr por otros medios. La propiedad más impactante de un computador es su generalidad. Puede seguir una serie de instrucciones, llamadas **programa,** que operan con datos dados. El usuario puede determinar y cambiar los programas y datos de acuerdo a una necesidad específica. Como resultado de esta f1exibilidad, los computadores digitales de uso general pueden realizar una serie de tareas de procesamiento de información de amplia variedad.

El computador digital de uso general es el ejemplo más conocido de **sistema digital**. Otros ejemplos incluyen conmutadores telefónicos, voltímetros digitales, contadores de frecuencia, máquinas calculadoras, y máquinas teletipos. Es característico de un sistema digital la manipulación de **elementos discretos** de información. Tales elementos discretos pueden ser impulsos eléctricos, los dígitos decimales, las letras de un alfabeto, las operaciones aritméticas, los símbolos de puntuación o cualquier otro conjunto de símbolos significativos. La yuxtaposición de elementos discretos de información representan una cantidad de información. Por ejemplo, las letras d, o y g forman la palabra dog. Los dígitos 237 forman un número.

De la misma manera una secuencia de elementos discretos forman un lenguaje, es decir una disciplina que con lleva información. Los primeros computadores fueron usados principalmente para cálculos numéricos, en este caso los elementos discretos usados son los dígitos. De esta aplicación ha surgido el término computador digital. Un nombre más adecuado para un computador digital podría ser “sistema de procesamiento de información discreta”.

Los elementos discretos de información se representan en un sistema digital por cantidades físicas llamadas **señales.** Las señales eléctricas tales como voltajes y corrientes son las más comunes. Las señales en los sistemas digitales electrónicos de la actualidad tienen solamente dos valores discretos y se les llama **binarios.** El diseñador de sistemas digitales está restringido al uso de señales binarias debido a la baja confiabilidad de los circuitos electrónicos de muchos valores. Debido a la restricción física de los componentes y a que la lógica humana tiende a ser binaria, los sistemas digitales que estén restringidos a usar valores discretos, lo estarán para usar valores binarios.

Las cantidades discretas de información podrían desprenderse de la naturaleza del proceso o podrían ser cuantificadas a propósito de un proceso continuo. Por ejemplo, un programa de pago es un proceso que contiene nombres de empleados, números de seguro social, salarios semanales, impuestos de renta, etc. El cheque de pago de un empleado, se procesa usando valores discretos, tales como las letras de un alfabeto (nombres), dígitos (salarios) y símbolos especiales tales como $. Por otra parte, un científico investigador podría observar un proceso continuo pero anotar solamente cantidades específicas en forma tabular. El científico estará cuantificando sus datos continuos. Cada número en su tabla constituye un elemento discreto de información.

Un **computador análogo** realiza una **simulación** directa de un sistema físico. Cada sección del computador es similar al de alguna parte específica del proceso sometido a estudio. Las variables en el computador análogo están representadas por señales continuas que varían con el tiempo y que por lo general son voltajes eléctricos. Las señales variables son consideradas similares con aquellas del proceso y se comportan de la misma manera.

Una calculadora electrónica es un sistema digital similar al computador digital que tiene como elemento de entrada el teclado y como elemento de salida una pantalla numérica. Inclusive Algunas calculadoras tienen forma de imprimir y además facilidad de programación. Sin embargo un aparato más poderoso que una calculadora; puede usar muchos otros dispositivos de entrada y salida, puede realizar no solamente cálculos aritméticos y operaciones lógicas sino que puede ser programado para tomar decisiones basadas en condiciones internas y externas.

Un computador digital es una interconexión de módulos digitales. Un procesador combinado con la unidad de control forma un componente llamado unidad central de proceso o CPU. Un CPU encapsulado en una pastilla de circuito integrado se denomina **microprocesador.** La unidad de memoria, de la misma forma que la parte que controla la interconexión entre el microprocesador y los elementos de entrada y salida, puede ser encapsulada dentro de la pastilla del microprocesador o puede encontrarse en pastillas pequeñas de circuitos integrados. Un CPU combinado con una memoria y un control de interconexión formará un computador de tamaño pequeño denominado **microcomputador.**

Los elementos de entrada y salida son sistemas digitales especiales manejables por partes electromecánicas y controladas por circuitos electrónicos digitales.

En síntesis un computador digital manipula elementos discretos de información y que estos elementos se presentan en forma binaria. Los operandos, usados en los cálculos pueden ser expresados en el sistema de números binarios. Otros elementos discretos, incluidos los dígitos decimales, se representan con códigos binarios. El procesamiento de datos se lleva acabo por medio de los elementos lógicos binarios, usando señales binarias. Las cantidades se acumulan en los elementos de almacenamiento binario.

CIRCUITOS DIGITALES

Sistema digital es cualquier sistema que pueda generar, procesar, transmitir o almacenar señales mediante dígitos y que solo admite valores discretos, es decir, que solo admite un conjunto limitado de números o valores.

ÁLGEBRA DE BOOLE

El Álgebra de Boole constituye una formalización apropiada para representar información digital y proporciona un modelo matemático para determinar la respuesta de los circuitos lógicos, independientemente del tipo de dispositivos que constituyen dichos circuitos (transistores, reales, etc.).

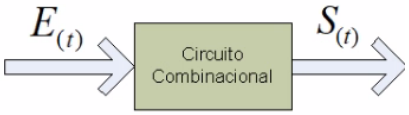
Algunas de las aplicaciones del álgebra booleana en conmutación son las siguientes:

1. Expresión algebraica de los requerimientos que debe cumplir un circuito lógico.
2. Análisis y síntesis de circuitos combinacionales.
3. Comparación de distintas realizaciones circuitales.
4. Minimización del número de cables y dispositivos en los circuitos.
5. Codificación de información digital.
6. Herramienta auxiliar para analizar y sintetizar circuitos secuenciales.

Llamaremos **circuitos lógicos o de conmutación**, a un circuito que trate señales lógicas

Decimos que un **circuito es combinacional** cuando las señales de salida no dependan más que de las señales de entrada, por lo que no necesita de módulos de memoria. Y será **secuencia**l si las señales de salida dependen de las de señales de entrada y del tiempo, estos poseen elementos de memoria que recojan la información de la "historia pasada" del sistema

El estudio de los **circuitos combinacionales** se basa en el álgebra de boole y el de los circuitos secuenciales en la teoría de autómatas finitos: tiene una cantidad de estados determinado que no es infinito. Estos circuitos son aquellos en los que sus valores de salida dependen de las combinaciones de valores en sus entradas en el mismo instante de tiempo t. Este tipo de circuitos no tiene en cuenta la variable tiempo, es decir, a menos que se redefina su estructura interna, un circuito combinacional entrega siempre el mismo valor de salida para una combinación dada de entradas.



ELEMENTOS DEL ALGEBRA DE BOOLE

Variables Lógicas: variables binarias que solo pueden tener un valor entre dos distintos, por convención las llamaremos 0 (cero, o falso) y 1 (uno, o verdadero); A, B, C… ε {0, 1}.

Funciones Lógicas o booleanas: Son funciones de n variables lógicas que toman valores en el conjunto 0, 1 => F (A1, A2, A3,…, An) ε {0, 1}

1. **Función inversa o complementación:** Función que permite invertir cualquier secuencia
   F(A) = A'
2. **Función intersección o producto lógico:** Operación que se materializa mediante la compuerta AND
   F (A, B) = A**.** B o A ^ B
3. **Función reunión o suma lógica:** Operación que se materializa mediante la compuerta OR
   F (A, B) = A + B o A v B

Para realizar las operaciones lógicas se opera con cada par de bits de A y B y se procesan, continuando con el siguiente para y así sucesivamente.

Tabla de verdad

Es una tabla, con todas las combinaciones posibles de las variables, que muestra la relación entre los valores que las variables pueden tomar y el resultado de la operación para un circuito digital determinado. Cualquier función de Boole puede ser representada por una tabla de verdad. En número de filas en la tabla es de 2^n donde n es el número de variables binarias de la función. Las combinaciones de unos y ceros se pueden obtener fácilmente para cada fila de los números binarios contando desde 0 a 2^{n-1}. Para cada fila de la tabla hay un valor para la función igual a 1 ó 0.

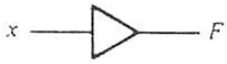
Operadores lógicos

Compuerta AND: Designamos la salida de una compuerta AND como el producto lógico de las entradas; en este caso la salida será 1 si y solo si todas las entradas son 1, caso contrario es igual a 0. La compuerta AND puede tener más de 2 entradas.

Compuerta OR: De acuerdo a la tabla la compuerta OR realizará la suma lógica de los valores de las entradas; la salida será 0 si y solo si todas las entradas son 0, caso contrario la salida es igual a 1. También la compuerta OR puede tener más de 2 entradas.

Compuerta NOT – INVERSOR: La compuerta NOT entrega a la salida el complemento de la señal de entrada. Es el único operador con una sola entrada.

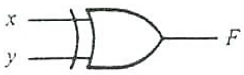
Compuerta BUFFER o SEPARADOR: El símbolo triángulo designa para sí solo un circuito separador (buffer). Un circuito separador produce la función de transferencia pero no produce ninguna operación lógica particular ya que el valor binario de la salida es igual al valor binario de la entrada. Este circuito se usa solamente para amplificación de señal de potencia y es equivalente a dos inversores conectados en cascada.



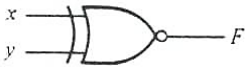
Compuerta NAND: La compuerta NAND surge de invertir la salida de una compuerta AND, por lo tanto, cada resultado de la compuerta NAND será el complemento de su correspondiente de la compuerta AND.

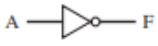
Compuerta NOR: La compuerta NOR es una compuerta OR con un inversor en su salida, como consecuencia complementa cada resultado que ésta genera, de modo de realizar una suma lógica negada.

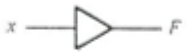
Compuerta XOR (OR EXCLUSIVO): Esta compuerta es similar a la compuerta OR, pero excluye la combinación de ambos, X y Y igual a 1, de acuerdo a esto, la salida de la compuerta XOR es igual a 1 si al menos una y a lo sumo una de las dos entradas es igual a uno, pero no ambas.



Compuerta XOR NEGADO (NOR EXCLUSIVO): La salida de la compuerta XOR NEGADO es igual a 1 cuando las dos variables son iguales, es decir, cuando ambas son 1 o ambas son 0. También se llama **equivalencia**, y se puede ver que esta compuerta es el complemento de la OR EXCLUSIVO.



Nombre	Símbolo gráfico	Función algebraica	Tabla verdad
AND		$F = A \cdot B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = (\overline{AB})$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (\overline{A + B})$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

Nombre	Símbolo gráfico	Función algebraica	Tabla verdad
Buffer		$F = x$	x | F 0 | 0 1 | 1
XOR		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
$\overline{\text{XOR}}$		$F = xy + x'y'$ $= x \odot y$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

MAPAS DE KARNAUGH

Si se quiere simplificar, los mapas de Karnaugh son una forma conveniente de representar una función booleana con pocas variables (de cuatro a seis). El mapa es un conjunto de 2^n cuadrículas, que representan las posibles combinaciones de los valores de n variables binarias.

FORMAS NORMALES O CANÓNICAS DE UNA FUNCIÓN

Una función se puede representar por una serie de términos canónicos Se **llama término canónico** de una función lógica a "todo producto o toda suma en los que aparecen todas las variables que componen una función, en su forma directa o inversa". O sea que hay productos canónicos y sumas canónicas.

Se llama **forma normal o canónica** a aquella expresión donde cada término de la función contiene todas las variables en su forma normal o negada.

MINITÉRMINOS Y MAXITÉRMINOS

Una variable binaria puede aparecer ya sea en su forma (X) o en su forma complementada (X**'**). Si consideramos dos variables binarias X e Y con su operador AND, ya que cada variable puede aparecer en cualquier forma, hay cuatro combinaciones posibles: X'Y', X'Y, XY' y XY. Cada uno de esos cuatro términos AND se denomina un **Minitérmino** de un Producto normalizado. En forma semejante pueden combinarse **n** variables para formar **2^n** minitérminos o términos mínimos.

De manera similar, las **n** variables formando un término OR, con cada variable tildada o no tildada, darán **2^n** combinaciones posibles, denominadas **Maxitérminos** de las sumas normalizadas. Cada maxitérmino es el complemento de su correspondiente minitérmino y viceversa.

A las funciones de Boole expresadas como Suma de Minitérminos o Producto de Maxitérminos se dice que está en **forma canónica**.

			Minitérminos		**Maxitérminos**	
x	**y**	**z**	**Termino**	**Designación**	**Termino**	**Designación**
0	0	0	x' y' z'	m_0	x + y + z	M_0
0	0	1	x' y' z	m_1	x + y + z'	M_1
0	1	0	x'y z'	m_2	x + y' + z	M_2
0	1	1	x' y z	m_3	x + y' + z'	M_3
1	0	0	x y' z'	m_4	x' + y + z	M_4
1	0	1	x y' z	m_5	x' + y + z'	M_5
1	1	0	x y z'	m_6	x' + y' + z	M_6
1	1	1	x y z	m_7	x' + y' + z'	M_7

Una propiedad importante del álgebra booleana es que cualquier función booleana puede expresarse como una suma de minitérminos (por “suma” se entiende la aplicación del operador OR en los términos) y como un producto de maxitérminos (por “producto” se entiende que se aplica el operador AND a los términos).

Suma de Minitérminos:

Son los minitérminos cuya suma define la función de boole, es decir, que son aquellos que dan los 1 de la función en una tabla de verdad. Como la función puede ser 1 ó 0 para cada minitérmino, y ya que hay 2^n minitérminos, se pueden calcular las funciones posibles que pueden formarse con n variables como 2^{2n}.

Minitermino o Producto Canónico: Es el producto de las variables en juego o sus negaciones individuales que hacen que el producto valga uno.

Producto de Maxitérminos:

Cada una de las 2^{2n} funciones de n variables binarias pueden expresarse como un producto de maxitérminos, o sea, son aquellos que dan los 0 de la función en una tabla de verdad.

Maxitermino o Suma Canónica: Es la suma de las variables en juego o sus negaciones individuales, que hacen que la suma valga cero.

<u>ENTONCES</u>

La **Forma Normal Disyuntiva (FND)** se obtiene a partir de la tabla de verdad, en las filas donde la función vale 1, tomando el producto de todas las variables, reemplazando así los 0 por sus respetivas negadas y los 1 por sus correspondientes variables sin negar. Luego se realiza la suma de los minitérminos así determinados.

A	B	C	Minitérminos	S
0	0	0	$\overline{A}.\overline{B}.\overline{C}$	1
0	0	1	$\overline{A}.\overline{B}.C$	0
0	1	0	$\overline{A}.B.\overline{C}$	0
0	1	1	$\overline{A}.B.C$	1
1	0	0	$A.\overline{B}.\overline{C}$	0
1	0	1	$A.\overline{B}.C$	1
1	1	0	$A.B.\overline{C}$	1
1	1	1	$A.B.C$	0

2. Una forma normal disyuntiva (FND) es una disyunción de conjunciones de literales:

(L11 y L12 y .. y L1n) o (L21 y L22 y .. y L2n) o (Lm1 y Lm2 y .. y Lnm)

La **Forma Normal Conjuntiva (FNC)** se obtiene a partir de la tabla de verdad en las flas donde la función vale 0, se forman las sumas de todas las variables reemplazando los unos de cada combinación por sus respectivas variables negadas y los 0 por sus correspondientes variables sin negar. Luego se realiza el producto de los maxitérminos así obtenidos.

A	B	C	Maxitérminos	S
0	0	0	$A+B+C$	1
0	0	1	$A+B+\overline{C}$	0
0	1	0	$A+\overline{B}+C$	0
0	1	1	$A+\overline{B}+\overline{C}$	1
1	0	0	$\overline{A}+B+C$	0
1	0	1	$\overline{A}+B+\overline{C}$	1
1	1	0	$\overline{A}+\overline{B}+C$	1
1	1	1	$\overline{A}+\overline{B}+\overline{C}$	0

1. Una forma normal conjuntiva (FNC) es una conjunción de disyunciones de literales:

(L11 o L12 o .. o L1n) y (L21 o L22 o .. o L2n) y (Lm1 o Lm2 o .. o Lnm)

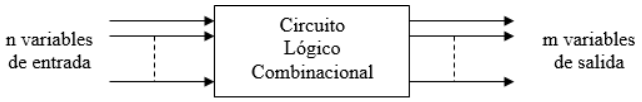
CIRCUITOS COMBINACIONALES

Los circuitos lógicos para sistemas digitales pueden ser combinacionales o secuenciales. Un circuito combinacional consta de **compuertas lógicas** cuyas salidas se determinan en cualquier momento de la combinación presente de las entradas sin tomar en cuenta las entradas previas. Este realiza una operación de procesamiento de información específica completamente lógica por medio de un conjunto de funciones boole.

Un circuito combinacional consiste en variables de entrada, compuertas lógicas y variables de salida. Las compuertas lógicas aceptan señales en las entradas y generan señales en las salidas. Este proceso transforma la información binaria de los datos de entrada dados a datos de salida requeridos. Los datos de salida y de entrada se representan por medio de señales binarias (1, 0).

Las n variables binarias de entrada vienen de una fuente externa, las m variables de salida van a un destino externo. En muchas aplicaciones la fuente y el destino son registros acumuladores, localizados en algún componente externo. Por definición un registro externo no debe influenciar el comportamiento de un circuito combinacional ya que si lo hace el sistema total se convierte en un circuito secuencial.

Para n variables de entrada, hay 2^n combinaciones posibles de valores de entrada binaria. Para cada combinación de entrada posible, hay una y solo una combinación de salida posible. Un circuito combinacional puede describirse por m funciones booleanas, una para cada variable de salida. Cada función de salida



se expresa en términos de las n variables entrada. Cada variable de entrada a un circuito combinacional puede tener una o dos conexiones.

Los circuitos combinacionales se emplean para generar decisiones de control binarias y para proporcionar los componentes digitales requeridos para el procesamiento de datos. Los circuitos combinacionales pueden describirse mediante una tabla de verdad.

CIRCUITOS SECUENCIALES

Un circuito es secuencial cuando sus salidas dependen del valor de las entradas y del estado anterior del circuito representado, que puede tener una o más salidas. Por lo tanto, la representación de estos circuitos debe tener en cuenta que las funciones de salida deben contemplarse como entradas; por ello se afirma que el circuito **tiene uno o más lazos de realimentación**.

Los circuitos secuenciales usan elementos de memoria (celdas binarias) además de las compuertas lógicas. Las salidas de un circuito secuencial son una función no solamente de las entradas externas sino del presente estado de los elementos de la memoria. El estado de los elementos de memoria, a su vez, es una función de las entradas previas. *Como consecuencia las salidas de un circuito secuencial dependen no solo de las entradas presentes, sino también, de las entradas del pasado y el comportamiento del circuito debe especificarse por una secuencia de tiempos de las entradas y estados internos*. Otra manera más general y útil de ver esto es que la salida actual de un circuito secuencial depende de la entrada actual y del estado actual del circuito.

El circuito que debemos diseñar, por ejemplo para el funcionamiento de un ascensor, recibe el nombre de circuito lógico secuencial, conocido como **máquina de estados finitos**, el cual toma una entrada y un estado actual para generar una salida y un nuevo estado. Esta máquina de estados finitos se distingue de un circuito combinacional en que la historia de las entradas del circuito secuencial tiene influencia sobre sus estados y sobre su salida. Este concepto es importante en la implementación de circuitos de memoria, así como en el diseño de unidades de control en una computadora.

Un diagrama de bloques de un circuito secuencial consta de un circuito combinacional al cual se le conectan elementos de memoria para formar un camino de realimentación. Los elementos de memorias son capaces de almacenar dentro de ellos información binaria. La información binaria almacenada en los elementos de memoria en cualquier momento dado define el estado del circuito secuencial.

Hay dos tipos de circuitos secuenciales. Su clasificación depende del temporizado de sus señales.

1. Un circuito secuencial **<u>sincrónico</u>** es un sistema cuyo comportamiento puede definirse a partir del conocimiento de sus señales en instantes discretos de tiempos.

Un sistema lógico secuencial sincrónico, puede usar señales que afecten los elementos de memoria solamente en **instantes de tiempo discreto**. Una forma de lograrlo es usar pulso de duración limitada a través del sistema de tal manera que la amplitud de un pulso representa la lógica 1 y otra amplitud de pulso (o la ausencia de un pulso) represente lógica 0.

El pulso de reloj se aplica a las compuertas AND conjuntamente con las señales que especifican los cambios requeridos en los elementos de memoria. Los circuitos secuenciales sincrónicos son el tipo más usado, también se les suele llamar circuitos secuenciales temporizados.

La sincronización se logra por un dispositivo de tiempo llamado generador maestro de tiempo el cual genera un tren periódico de pulsos de reloj. Los pulsos de reloj se distribuyen a través del sistema de tal manera que los elementos de memoria son afectados solamente con la llegada del pulso de sincronización.

2. El comportamiento de un circuito secuencial **<u>asincrónico</u>** depende del orden en que cambien las señales de entrada (es decir, cambios se producen únicamente por activación de alguna de las entradas) y puedan ser afectadas en cualquier instante de tiempo dado. Los elementos de memoria comúnmente usados en los circuitos secuenciales asincrónicos son mecanismos retardadores de tiempo. Así, un circuito secuencial asincrónico puede tomarse como un circuito combinacional con realimentación.

TEORÍA DE AUTÓMATAS FINITOS

Funciones de transición

S(t+1) = F [H(t), E(t)] *expresión genérica*

S(t+1) = F [Q(t), E(t)]

Q(t+1) = G [Q(t), E(t)]

/Qt= Estado AF ent; Et= Entradas AF en t; F= función de salida en t+1

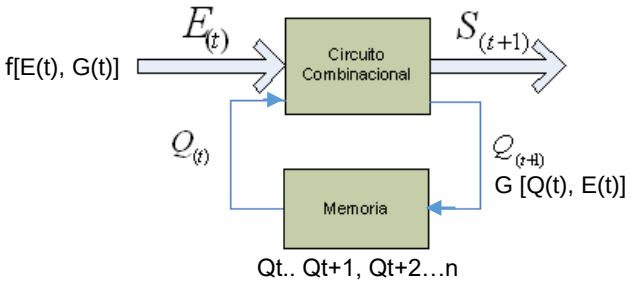


Tabla de estados:

Es una matriz que permite ordenar en columnas los estados actuales, las estradas y los estados futuros, en función al diagrama de transición definido. Incluye: entradas, estado anterior, salida/s y estado siguiente

La secuencia de tiempo de las entradas, salidas y estados de los flip – flops pueden enumerarse en una tabla de estado. Ella consiste en 3 secciones llamadas estado presente, estado siguiente y salida.

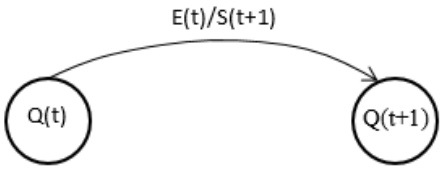
El **estado presente** designa los estados de los flip – flops antes de la ocurrencia de un pulso de reloj. El **estado siguiente** muestra los estados de los flip – flops después de la aplicación del pulso de reloj y la sección de **salida** lista los valores de las variables de salida durante el presente estado.

A	B	x	$A^{\prime}$	$B^{\prime}$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

Generalmente un circuito secuencial con m flip – flops y n variables de entrada tendrá 2m filas, una para cada estado. Las secciones del siguiente estado y de salida tendrán cada una 2n columnas, una para cada combinación de entrada.

Diagrama de transición:

Grafo orientado que permite ver cómo se produce cada cambio de estado en función a una entrada y una posible salida.



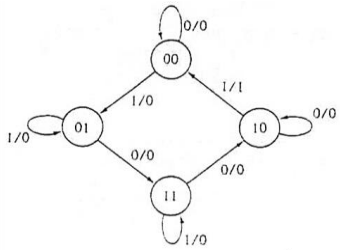


Figura 6-16 Diagrama de estado

La información disponible en la tabla de estado puede representarse gráficamente en un diagrama de estado. En este diagrama se representa un estado por un círculo y la transición entre estados se indica por líneas dirigidas que conectan los círculos.

El número binario dentro de cada círculo identifica el estado representado por el círculo. Las líneas dirigidas se marcan con dos números binarios separados por /. El valor de entrada que causa la transición de estado se marca primero; el número después de la / da el valor de la salida durante el presente estado. Una línea dirigida que conecta un círculo a sí mismo indica que no hay cambio de estado. El diagrama de estado suministra la misma información que la tabla de estado, es decir, que no hay diferencias excepto en la forma de presentación.

Tabla de excitación:

Es una matriz que permite ordenar en columnas los estados actuales, las entradas, los estados futuros y las entradas a los elementos de memoria Qt que registran cada estado del autómata.

J_A	K_A
0	X
0	X
0	X
1	X
X	0
X	0
X	0
X	1

Tabla Excitación Elementos de memoria

Cada tabla consiste en dos columnas, Q (t) y Q (t + 1), y una columna para cada entrada para mostrar cómo se logra la transición requerida (o cambio de estado). Hay cuatro transiciones posibles del presente estado al siguiente. Las condiciones de entrada requeridas para cada una de las cuatro se derivan de la información disponible en la tabla característica (o tabla de funcionamiento). El símbolo X en las tablas representa la condición de no importa, es decir, no importa que la entrada sea 1 ó 0.

Ecuaciones de estado

Una ecuación de estado (también conocida como una ecuación de aplicación) es una expresión algebraica que especifica las condiciones para la transición de estado de un flip – flop. El lado izquierdo de la ecuación denota el estado siguiente del flip – flop y el lado derecho una función de Boole que especifica las condiciones del presente estado que hacen el siguiente estado igual a 1. La ecuación de estado se deriva directamente de la tabla de estado.

BIESTABLES O FLIP-FLOPS

Un biestable es una celda binaria capaz de almacenar un bit. Tiene dos salidas, una para el valor del bit almacenado y otra que representa su complemento. Su denominación sugiere que el biestable tiene sólo dos estados posibles de funcionamiento, permaneciendo en cualquiera de ellos si los niveles lógicos de las entradas no fuerzan su cambio. El biestable es un arreglo de compuertas, caracterizado por tener lazos de realimentación que permitan mantener o memorizar el efecto de combinaciones anteriores en las entradas.

La forma más sencilla de un circuito secuencial es un biestable. Hay varios tipos de biestables, y todos ellos comparten dos propiedades:

- El biestable es un dispositivo con dos estados. Está en uno de dos estados, en ausencia de entrada, recordando el último estado. Entonces, el biestable puede funcionar como una memoria de un bit.
- El biestable tiene dos salidas, que son siempre complementarias. Normalmente se denominan Q y $\overline{Q}$.

Flip – Flop: se llama así porque el sonido de las memorias de las maquinas viejas eran compuertas de biestables electromecánicos (relais). Circuito que conduce o no conduce.

Un circuito flip – flop puede mantener un estado binario en forma indefinida aún luego de que las entradas pasen a un estado inactivo, hasta que se cambie por una señal de entrada para cambiar estados. La diferencia principal entre los diversos flip – flop esta en el número de entradas que poseen y en la manera en la cual las entradas afectan el estado binario.

Características

- El flip-flop es un sistema con dos estados estables (0 o 1 estados binarios). A cada representación del estado se le asigna una función. El estado es una situación interna del circuito. El estado de un circuito esta determinado por los valores que los biestables puedan estar en ese momento.
- La salida de un flip – flop queda determinada tanto por las entradas actuales como por la historia de las mismas.
- El circuito del flip-flop puede recordar o almacenar un BIT de información binaria.
- El flip-flop tiene dos señales de salida, de las cuales una es el complemento de la otra.
- El flip – flop se mantiene en uno de los estados, mientras no sea exitado por señal alguna.
- La mayoría de los flip – flop tienen entradas de impulso y salidas de nivel.

Según en qué instante pueden actuar sus entradas y cambiar de estado (y en correspondencia cambiar el valor de sus salidas), los flip – flops pueden ser:

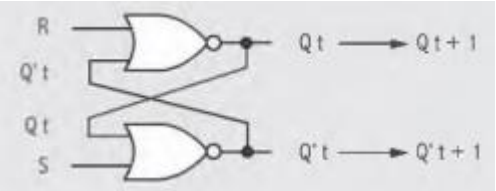
- **Asincrónico:** sus entradas pueden actuar en cualquier instante para cambiar el estado del circuito, y en conformidad cambiar el valor de sus salidas.

• **Sincrónico:** sus entradas pueden actuar sobre el estado del circuito sólo en determinados instantes, definidos en relación a pulsos que llegan por otra entrada de sincronismo conocida como reloj o “clock”. La entrada CP o CLK recibe pulsos de sincronismo que son generados por lo general a una frecuencia regular por algún tipo de oscilador, de donde viene el nombre “reloj”.

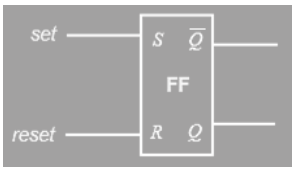
Flip – Flop RS	Asincrónico Sincrónico
Flip – Flop JK	Asincrónico Sincrónico
Flip – Flop D	Sincrónico
Flip – Flop T	Asincrónico Sincrónico

BIESTABLE R-S ASINCRÓNICO

El gráfico del biestable R –S asincrónico reemplaza al expuesto antes, ya que resultan más visibles las interconexiones entre las dos compuertas NOR. Siendo:



• R la entrada de reseteo o puesta a 0.
• S la entrada de seteo o puesta a 1.
• Qt el almacenamiento del valor del biestable antes de ser activado.
• Qt + 1 e1 almacenamiento del estado posterior del biestable.
• Q't el almacenamiento del complemento del valor del biestable antes de ser activado.
• Q't + 1 el almacenamiento del estado posterior del complemento del biestable antes de ser activado



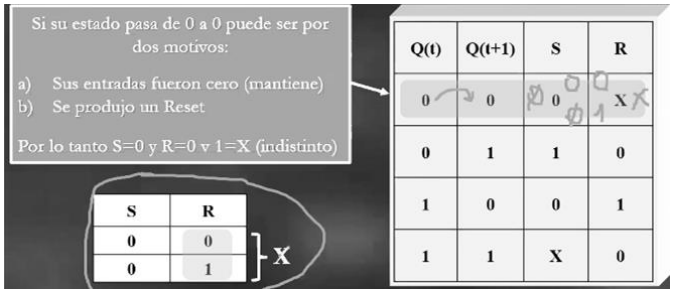
Debido a las distintas nomenclaturas aceptadas en diversos manuales, se debe considerar que la salida Q puede denominarse Qt y que el estado posterior de Q puede denominarse indistintamente Qt+1 o Q'. De la misma manera, no Q es equivalente a Q y, por lo tanto, Q ' es el estado posterior de Q

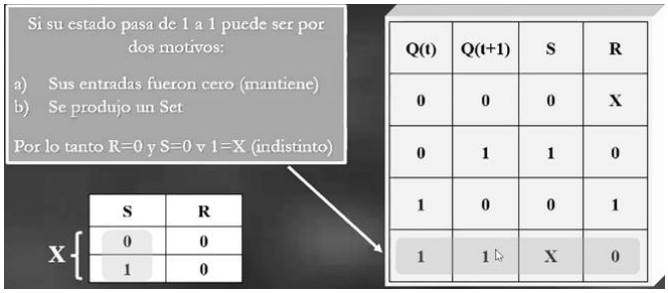
La entrada **Set** almacena un uno, mientras que **el Reset** almacena un cero. En ausencia de señal de entrada, el flip-flop mantiene el valor almacenado, lo que lo convierte en un componente ideal para almacenar un valor lógico de manera temporal. Se verifica el siguiente comportamiento:

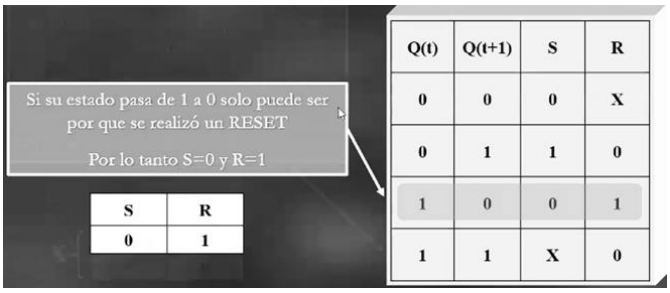
• Si R=S=0 => Q’=Q (Mantiene su valor)
• Si R=S=1 => Q=0 o 1 (INDETERMINADO)
• Si R= 1; S=0 => Q=0 (Reset)
• Si R=0; S=1 => Q=1 (Set)

Tabla de Excitación

Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

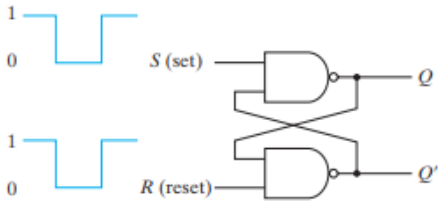






En el caso del flip-flop básico **NAND**, debemos recordar que la salida de una NAND es 0 si todas las entradas son 1, y es 1 cuando alguna entrada es 0.

- Normalmente funciona con ambas entradas en 1 (a diferencia del latch NOR que trabaja con ambas en 0) a menos que el estado del biestable tenga que cambiar, para lo cual se envía un 0 por la entrada correspondiente.
- La condición indefinida se da cuando ambas entradas son 0, lo que produce que ambas salidas NAND sean 1. Luego, cuando las entradas vuelven a 0, el estado es indeterminado, y dependerá de aquella entrada que permanezca por mayor tiempo en 0 antes de hacer la transición a 1.



S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(Después de $S = 1, R = 0$)
0	1	1	0	
1	1	1	0	(Después de $S = 0, R = 1$)
0	0	1	1	

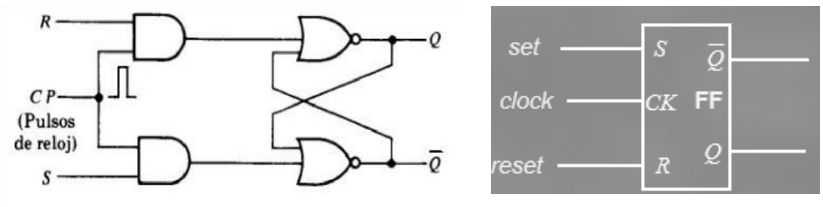
a) Diagrama lógico b) Tabla de función

BIESTABLE R-S SINCRÓNICO (TEMPORIZADO)

En los sincrónicos o temporizados, la presencia de un pulso en una entrada especial determina el instante a partir del cual las entradas modifican su estado. Los pulsos de sincronismo son generados por un **sistema de reloj** y, por eso, llamaremos a la señal que lo representa Ck (clock).

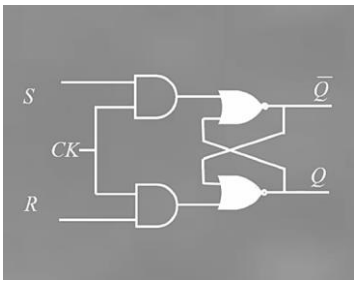
La señal Ck oscila entre 0 y 1 a intervalos regulares, de manera que un biestable sincrónico es accionado cuando Ck = 1. Así, si varios biestables están relacionados entre sí por la señal de sincronismo, se modifican en el mismo instante.

El flip-flop RS síncrono **NOR** consiste en un flip-flop básico NOR y dos compuertas AND:



Las salidas de las compuertas AND permanecen en cero mientras el pulso de reloj (en inglés CP) sea 0, independientemente de los valores de S y R. Cuando el pulso de reloj vaya a 1, la información de las entradas S y R entra al flip-flop básico afectando las salidas, pudiéndose dar los siguientes casos:

- S = 1 y R = 0 cuando CP va a 1, se pasa al estado de puesta a uno.
- S = 0 y R = 1 cuando CP va a 1, se pasa al estado de puesta a cero.
- Con S = 1 y R = 1, el pulso de reloj causará que ambas salidas vayan momentáneamente a 0, y luego, cuando se quite el pulso, el estado del flip-flop será indeterminado o impredecible, es decir, podría resultar 0 o 1, dependiendo de cuál de las dos entradas (S o R) permanezca el mayor tiempo. Esta condición indeterminada hace que el flip-flop RS sea difícil de manejar y por lo tanto se usa rara vez en la práctica.

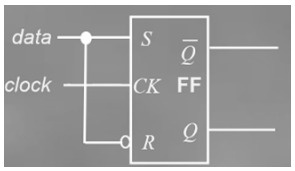


CK	S	R	Q
⎍	0	0	Q
⎍	0	1	0
⎍	1	0	1
⎍	1	1	?

R	S	Q	Q(t + 1)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	indeterminado
1	1	1	indeterminado

BIESTABLE D SINCRÓNICO

El biestable D a veces se denomina biestable de datos porque es, en efecto, almacén para un bit de datos. La salida del biestable D es siempre igual al valor más reciente aplicado a la entrada. Por tanto, recuerda y produce la última entrada. También se le llama biestable de retardo, porque retrasa un cero o uno aplicado a la entrada durante un pulso de reloj.

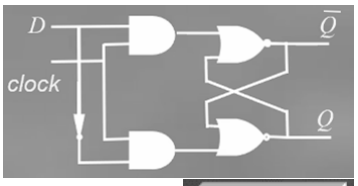


Es una modificación del RS síncrono. **Es básicamente un flip-flop RS de una sola entrada con un inversor en la entrada R,** lo que ofrece una manera de eliminar el estado indefinido en el flip-flop RS, ya que asegura que las entradas R y S nunca sean iguales a 1 al mismo tiempo.

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Tabla de Excitación

Para evitar la indeterminación del RS cuando ambas entras son igual a uno, se opta por unir la entrada del Set a la del Reset a través de una compuerta lógica Not; de esta manera el flip-flop D almacena el calor lógico presente en su entrada (cero o uno según corresponda).

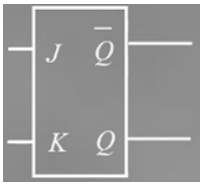


Debido a la ausencia de entrada en D se interpreta como Cero Lógico, es necesario su diseño de tipo sincrónico para así evitar esta situación de Reset espontaneo. **La entrada de Ck es interpretada como Carga** en bancos de memoria.

CK	D	Q
⎍	0	0
⎍	1	1

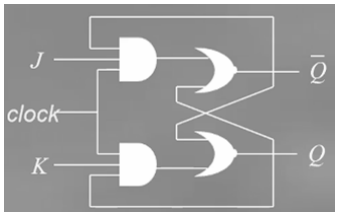
BIESTABLE J -K SINCRÓNICO

Como el biestable S-R, tiene dos entradas. Sin embargo, en este caso, todas las combinaciones posibles de los valores de entrada son válidos. Las entradas J y K se comportan como las entradas S y R respectivamente para poner a uno o cero al flip-flop. Cuando las entradas J y K son ambas 1, el estado del flip-flop se complementa.



El funcionamiento es el siguiente:

- La salida Q se aplica junto con K y CP a una compuerta AND de manera que el flip-flop se ponga a cero durante un pulso de reloj solamente si Q fue 1 previamente.
- De manera similar la salida Q se aplica junto con J y CP a una compuerta AND de manera que el flipflop se ponga a uno con un pulso de reloj, solamente si Q fue 1 previamente.



- Cuando J = K = 1, el pulso de reloj activa solo una de las compuertas AND, aquella conectada a la salida igual a 1. Entonces si Q = 1, la salida de la AND superior se convertirá en 1 cuando se aplique el pulso de reloj, y el flip-flop se pondrá a cero. Si Q = 1, la salida de la AND inferior se convierte en 1 poniendo a uno al flip-flop. En cualquier caso, el estado se complementa.

Cuando J = K =0
Mantiene el valor almacenado en Q_{t+1}

J	K	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

R

S

Cuando J = K = 1 (Complemento)
pone $Q_{t+1} = \overline{Q_t}$

Si su estado pasa de 0 a 0 puede ser por dos motivos:

a) Sus entradas fueron cero (mantiene)
b) Se produjo un Reset

Por lo tanto S=0 y R=0 v 1=X (indistinto)

Q_t	Q_{t+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

J	K
0	0
0	1

} X

Tabla de Excitación

Si su estado pasa de 0 a 1 puede ser por dos motivos:

a) Se hizo un Set (J=1 y K=0)
b) Se complementó (J=K=1)

Por lo tanto J=1 y K=0 v 1=X (indistinto)

Q_t	Q_{t+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

J	K
1	0
1	1

} X

Tabla de Excitación

Si su estado pasa de 1 a 0 puede ser por dos motivos:

a) Se hizo un Reset (J=0 y K=1)
b) Se complementó (J=K=1)

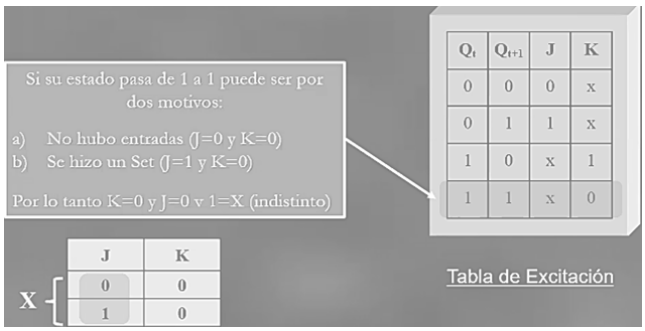
Por lo tanto K=1 y J=0 v 1=X (indistinto)

Q_t	Q_{t+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

X {

J	K
0	1
1	1

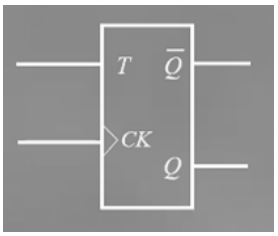
Tabla de Excitación



Q_t	Q_{t+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

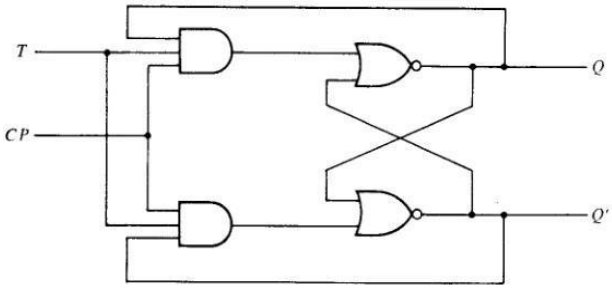
J	K
0	0
1	0

BIESTABLE T SINCRÓNICO



Versión de una sola entrada del biestable JK, se obtiene uniendo las entradas J y K de dicho flip-flop. El nombre T se deriva de la habilidad del flip-flop de variar ("toggle") o cambiar estado. Se lo conoce también como “de disparo”. Independientemente del presente estado del flip-flop, este asume el estado de complemento cuando ocurre el pulso de reloj mientras que la entrada T sea 1.

Cuando T=0 (J=K=0), un borde de reloj no modifica la salida, mantiene el valor reservado. Cuando T=1 (J=K=1), un borde de reloj complementa la salida. El flip-flop complementador es útil para diseñar contadores binarios.

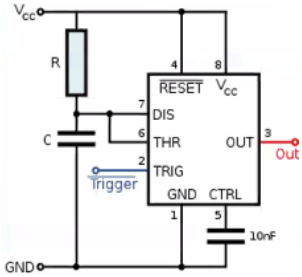


Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

EL MONOESTABLE

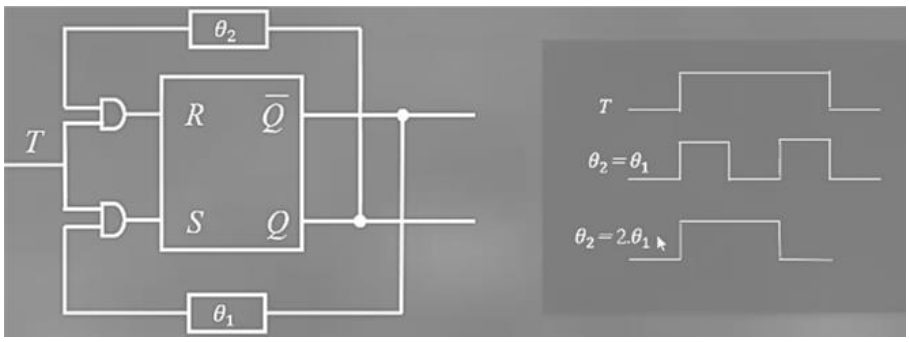
Es un sistema de dos estados con solo una posición estable. Consiste de un RS donde la señal se manda siempre a S, donde Q la manda a R y la retroalimentación se ve realentizada (por θ) por un temporizador

Se lo puede implementar a través de un circuito multivibrador que realiza una función secuencial consistente en que al recibir una excitación exterior, cambia de estado y se mantiene en el durante un periodo que viene determinado por una constante de tiempo. Transcurrido dicho periodo, la salida del monoestable vuelve a su estado original. Por lo tanto, **el monoestable es un sistema de dos estados con solo uno de ellos estables (y otro casi estable**)

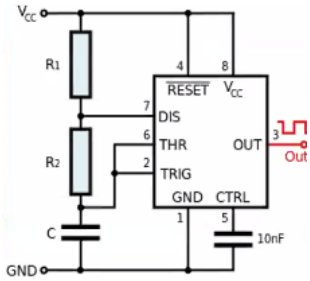


LA BÁSCULA

Es un sistema con ninguno de sus dos estados **estables.** Puede realizarse uniendo las dos entradas de un Biestable JK y en electrónica se los conoce como **Astable**. En el gráfico, θ1 maneja S y θ2 maneja R.



Se lo puede implementar a través de un circuito multivibrador **que no tiene ningún estado estable**, lo que significa que posee dos estados inestables entre los que conmuta, permaneciendo en cada uno de ellos un tiempo determinado (manejado por los tiempos de carga y descarga de cada condensador).



REGISTROS

Son elementos de memoria de almacenamiento temporal de información, instrucciones y datos, durante el procesamiento. A una orden procedente en general de la unidad de control, una información puede ser transferida de un registro a otro, sin que se altere el contenido del primero. Se denomina registro a un conjunto de biestables que funcionan como unidad, permitiendo almacenar información en el transcurso de un proceso.

Un registro representa un **arreglo de flip-flops** organizados a través de una lógica combinacional y/o secuencial que lo capacita para realizar operaciones específicas sobre la información binaria contenida. Un registro es una **función digital**, que suele utilizarse **para retener información binaria temporariamente**. La principal característica de los registros es que poseen una gran velocidad de acceso a los datos y permiten almacenar los datos rápidamente, pero cuentan con poca capacidad de almacenamiento, se encuentran en el microprocesador que guardan la información temporalmente.

Como mínimo debe contener la lógica necesaria para la su lectura y grabación con el objeto de poder inicializarte. Las lógicas más sencillas permiten operaciones básicas sobre todo el conjunto de flip-flops tales como la puesta a cero, el complemento, y la transferencia. Un registro puede almacenar uno o más elementos discretos de información, pero la misma configuración de bits puede ser interpretada, de manera diferente para diferentes elementos de información.

Algunas lógicas más elaboradas los capacitan para desplazar su información a derecha izquierda o de manera cíclica a través de una señal de gobierno y la lógica correspondiente. Otras permiten realizar contadores y Descontadores como parte de la lógica de control de determinados órganos.

Un **contador** es básicamente un registro que pasa por una sucesión predeterminada de estados. Las compuertas del contador están conectadas de tal manera que producen la sucesión prescrita de estados binarios.

Finalmente un arreglo de Registros puede verse como una Memoria y la lógica asociada al mismo permite la carga y lectura de palabras en función a una dirección o ubicación dada por la lógica de de decodificación de direcciones

Operaciones elementales sobre los registros

- Un registro puede limpiarse o *ponerse a cero*, enviándole un impulso por la entrada R de cada biestable.
- El contenido de un registro puede *complementarse*, enviándole un impulso por la entrada C (de complemento) de cada biestable. Para complemento a la base -1 se toma los Q'.
- El contenido de un registro puede *desplazarse* a izquierda o a la derecha.
- El contenido de un registro puede *incrementarse o decrementarse*.

Operaciones entre registros

- Transferir el contenido de un registro a otro.
- Sumar o restar el contenido de un registro con el contenido de otro.

Tipos de registros

- **Registros de datos:** usados para guardar números enteros. Antiguamente existía un único registro donde se guardaba toda la información, llamado acumulador.

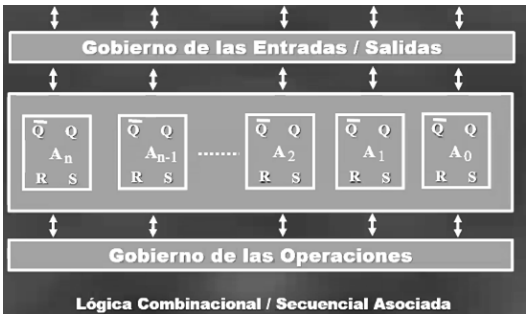
Registros de memoria: son usados para guardar exclusivamente direcciones de memoria. Muy usados en Arq. Harvard, ya que muchas veces las direcciones tienen un tamaño de palabra distinto que la de datos.

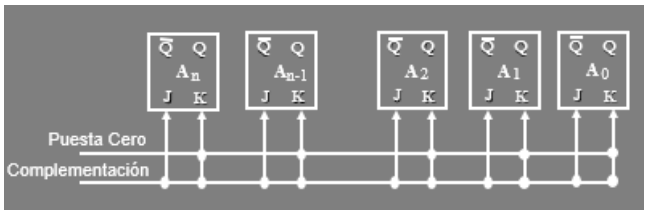
Registros de propósito general (GPR): pueden guardar tanto datos como direcciones. Son fundamentales en la Arq. Von Neumann, la mayor parte de las computadoras modernas lo usan.

Registros de coma flotante: son usados para guardar datos en formato de coma flotante,

Registros de constantes: no son editables, son solo de tipo lectura y son generados a partir de valores por hardware.

Registros de propósito específico: guardan información específica del estado del sistema, como el puntero de pila o el registro de estado, es decir, guarda información específica del estado del sistema, hardware y software también es encargado de supervisar el registro de estados de procesos.



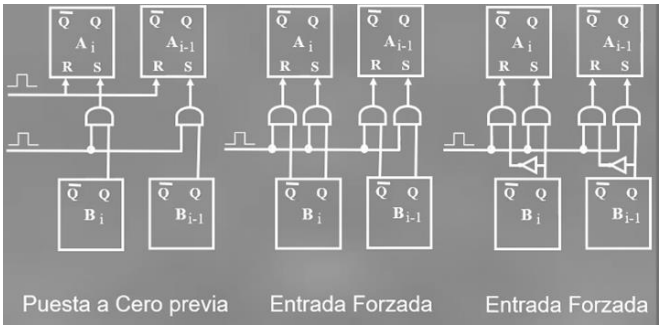


Registro con carga paralela

Algunos registros cumplen la función de transferir la información que viene por las líneas de entrada en paralelo a salidas en paralelo. La transferencia de información nueva a un registro se describe como carga del registro. Si todos los bits del registro se cargan simultáneamente, con un pulso de reloj común, se dice que la carga se efectúa en paralelo.

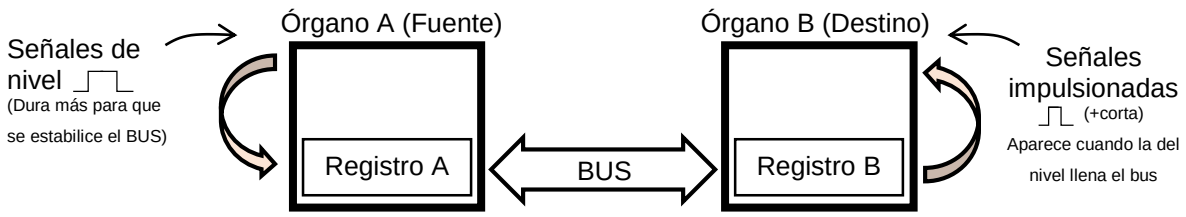
Una señal de control, llamada validación de dato de entrada, controla la escritura en los registros de los valores provenientes de las líneas de señales.

Puede transferirse la información de un registro B a otro A, bien por dos impulsos sucesivos, uno de puesta a cero y el otro de transferencia, o bien por introducción forzada de los 1 y 0.



BUSES

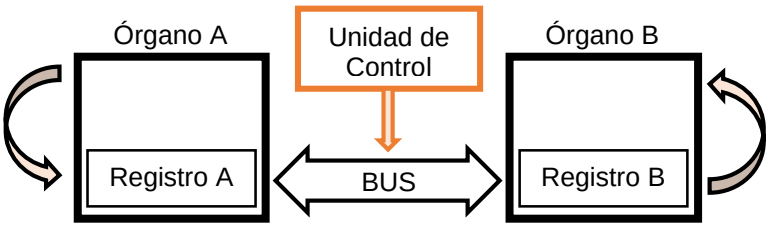
BUS es una palabra que significa transporte y consiste en un arreglo de líneas lógicas dedicadas a interconectar Órganos de computador entre sí, con el objeto de transportar las configuraciones binarias presentes en sus registros. Es decir, se utilizan para interconectar varios registros (unos considerados como registros fuente de información y otros como destinatarios de información)



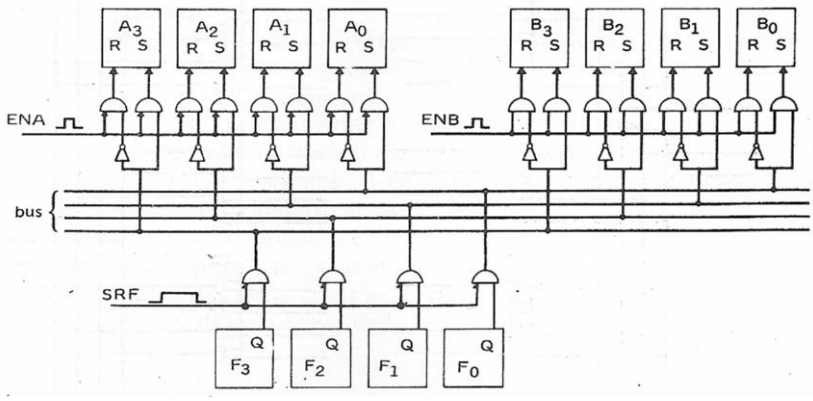
En la práctica son un conjunto de conductores eléctricos en forma de pistas metálicas impresas sobre la tarjeta madre del computador, por donde circulan las señales que corresponden a los datos binarios del lenguaje maquina con la que opera el Microprocesador.

Un bus es un conjunto de hilos en donde las informaciones binarias son mantenidas bajo forma de tensión por uno de los registros fuente. Un bus consta de uno o dos hilos por bit, en unos de los cuales está la magnitud cierta y en el otro la complementada. El bus puede alimentar independientemente o simultáneamente a varios registros destinatarios.

Las señales que representan bits sobre el bus son digitales y pueden transferir el dato en serie o en paralelo. El grado de paralelismo del bus unido a la velocidad que admite para lograr la transferencia se denomina caudal del bus. Cuando el ciclo de bus está controlado por el reloj del sistema, el bus realiza una transferencia sincrónica; en cambio, cuando su operación está controlada por los dispositivos conectados a él realiza una transferencia asincrónica.



La UC gobierna el BUS, las señales de gobierno de nivel permiten estabilizar los valores lógicos del Registro A sobre el BUS. ALU controla y manda las señales de gobierno de nivel



En el bus se establecen los niveles de tensión correspondientes a los contenidos de los biestables del registro F y se mantienen con

Señal SRF → Apertura de las puertas

Una vez se estabilizan los niveles en el bus

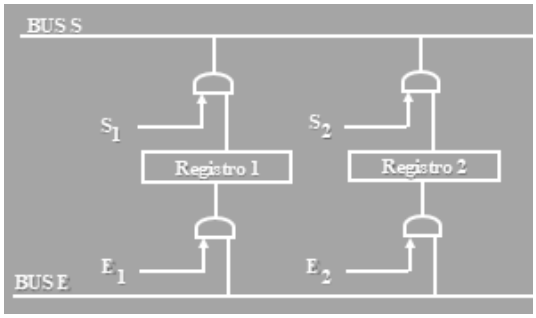
Impulso de Señal ENB → abre las puertas de entrada a los biestables del registro B, introduciendo la información mantenida en el bus por el registro F.

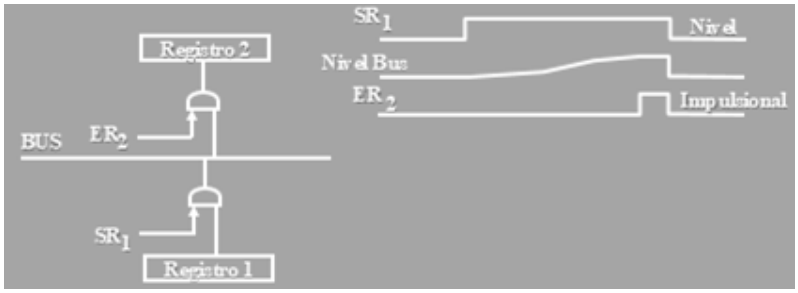
Señal SRF → puede volver el valor cero

Señal impulsional de gobierno EN (ENA y ENB) → valida la información a la entrada del registro → señal de muestreo

El tiempo de llenado de bus es el que le lleva para que las señales estén a lo largo de todos los hilos.

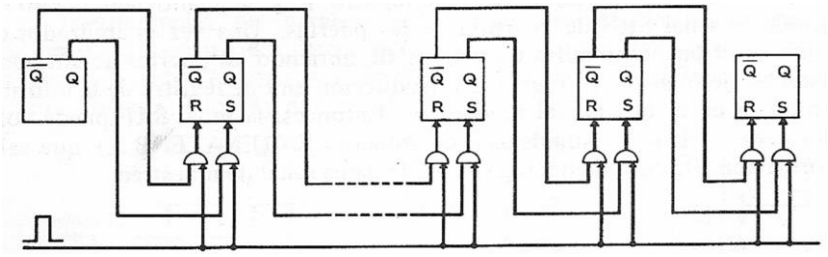
Registro situados entre dos buses





Registros especiales

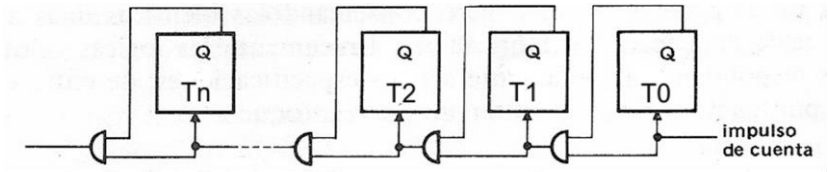
<u>Registros de desplazamiento</u>: permiten desplazar la información a derecha, a izquierda o en ambos sentidos, según la información de control y la lógica combinacional que lo gobierne. Tienen capacidad de carga y de lectura, al igual que un cableado secuencial que les permite que un valor almacenado se pueda ir de derecha a izquierda (por ejemplo).



Todos los flip – flops reciben un pulso de reloj común el cual causa el desplazamiento de un estado al siguiente. La salida Q de un flip – flop dado, se conecta a la entrada S, mientras que la salida Q' a la entrada R, del flip – flop a la derecha (en caso de desplazamiento a derecha). Cuando se trabaja con flip – flop D, la salida Q va conectado a la entrada D, del flip – flop de la derecha.

<u>Registros contadores:</u> permiten incrementar su contenido binario en una unidad por cada señal que cuenta. Es una cadena de flipflop, de manera que cada posición de cadena binaria puede pasar de 1 a 0 y de 0 a 1 según el impulso de la cuenta; los ciclos varían según la posición del biestable, cuando cumple su ciclo, induce al de mayor orden. Las compuertas en un contador se conectan de tal manera que se produce una secuencia preestablecida de estados binarios en el registro.

Los pulsos de entrada, pueden ser pulsos de reloj, o ellos pueden originarse en una fuente externa y pueden ocurrir en intervalos establecidos de tiempo o aleatoriamente. En un contador, la secuencia de estados puede seguir una cuenta binaria, que es la más simple y directa, o cualquier otra secuencia de estados. Un contador que sigue la secuencia binaria se llama **contador binario.**



Contador de n bits

Un contador de n bits consiste en n flip – flops y puede contar de 0 hasta $2^n - 1$. La única entrada al circuito puede ser el pulso de cuenta, o bien, una entrada de habilitación. Mientras que las salidas se especifican directamente con los estados presentes de los flip – flops.

El siguiente estado del contador depende enteramente de su estado presente y la transición de estado ocurre cada vez que ocurre un pulso de reloj. Debido a esta propiedad, un contador puede especificarse por

medio de una lista de secuencia de cuenta, es decir, la secuencia de los estados binarios que se le suceden. Esto es muy importante ya que la secuencia de cuenta da toda la información necesaria para diseñar el circuito.

Contadores de Impulsos

Son registros cuyo contenido se incrementa en 1 cada vez que se presenta un impulso a la entrada de cuenta. Son capaces de contar el número de impulsos recibidos. Trabaja de la siguiente manera: el impulso ataca al biestable de menor peso posicional.

Si está a cero, debe pasar a 1; si está a 1, debe pasar a 0, pero debe propagar el impulso al biestable inmediato con objeto de tomar en cuenta el arrastre. Por consiguiente, el biestable debe ser complementado en todos los casos y debe propagarse el impulso solamente en aquellos casos en que el biestable estaba a 1 antes de la complementación (condición para que haya arrastre).

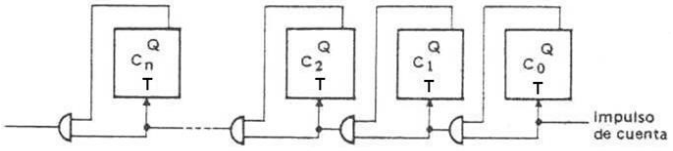


Fig. 76. Contador binario

Contador Binario Creciente – Decreciente

Un contador binario con una cuenta invertida se llama contador binario decreciente. En este contador la cuenta binaria se disminuye en 1 con cada pulso de cuenta de entrada.

Una lista de una secuencia de cuenta de un contador binario decreciente muestra que el bit de menor orden debe ser complementado con cada pulso de cuenta. Cualquier otro bit en la secuencia es complementado, si el bit previo de menor orden va de 0 a 1. Por lo tanto, podemos decir que cuando Q va de 0 a 1, Q' irá de 1 a 0 y se complementará el siguiente flip – flop como se requiere.

En un contador binario sincrónico creciente – decreciente el flip – flop en la posición de menor orden se complementa con cada pulso. Un flip – flop en cualquier otra posición se complementa con un pulso siempre y cuando todos los bits de menor orden sean iguales a cero. Un contador capaz de contar hacia arriba o hacia abajo se muestra en la figura.

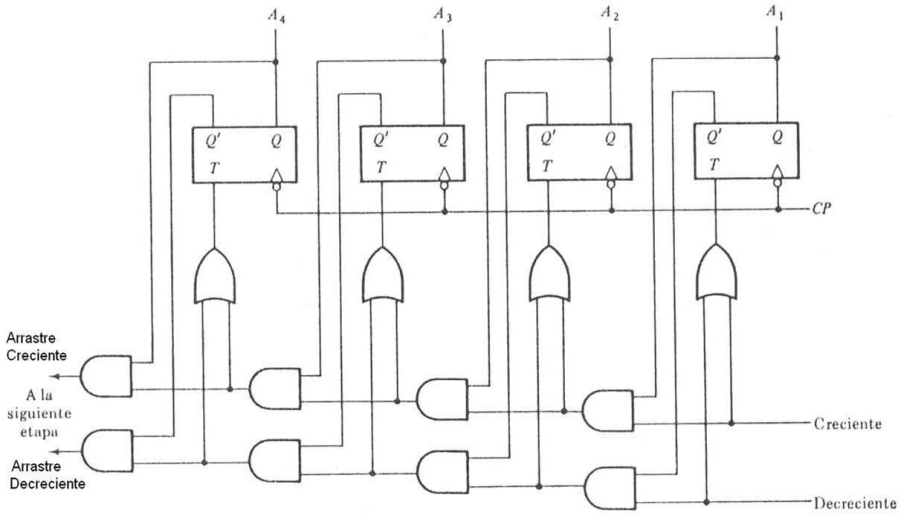


Figura 7-18 Contador binario creciente-decreciente de 4 bits

Cuando la entrada de control creciente es 1, el circuito cuenta hacia arriba, ya que las señales de T se determinan a partir de los valores previos de las salidas normales de Q.

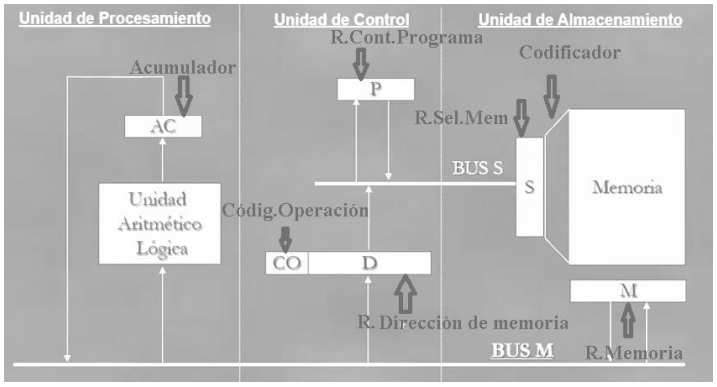
Cuando la entrada del control decreciente es 1, el circuito contará hacia abajo, ya que las salidas complementadas Q' determinan los estados de las entradas T. Cuando ambas señales creciente y decreciente son 0, el registro no cambia de estado pero permanece en la misma cuenta.

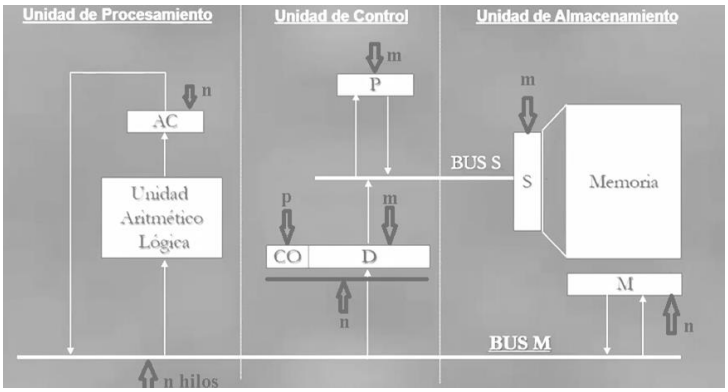
ARQUITECTURA ABACUS DE MEINADIER

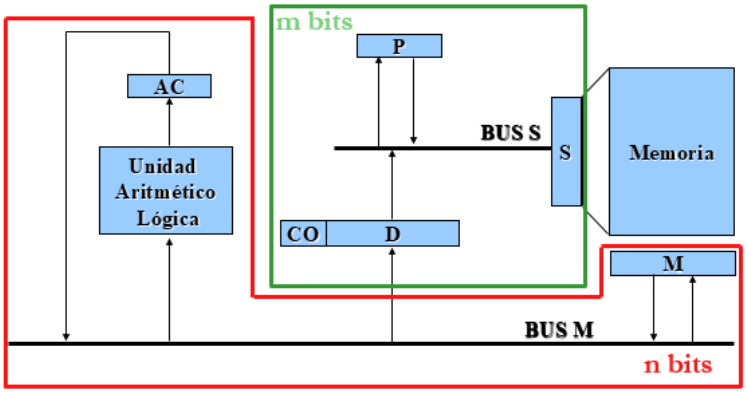
Abacus es una arquitectura simple presentada por Meinadier en su libro con fines puramente académicos. Está diseñada como una máquina sincrónica, con un registro de instrucción de una dirección (código de operación y dirección del operando) en tal sentido los cálculos se realizan contra el valor almacenado en el registro acumulador de la UAL.

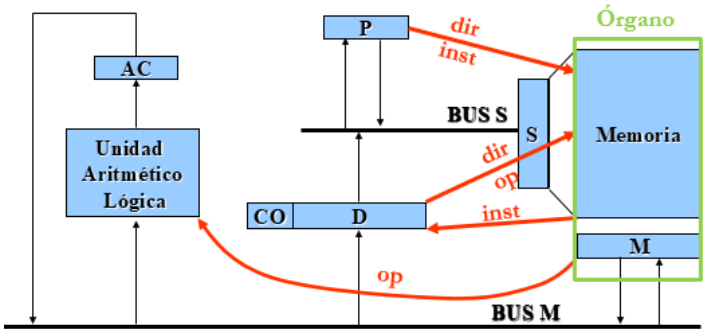
El reloj sincroniza a todos sus componentes produce un batido por cada θ segundos, siendo el ciclo base de memoria de 2.

Posee un bus M o de instrucción y operandos, y otro bus S de direcciones operandos. El primero lleva la próxima instrucción al registro de instrucciones para ser decodificada. En caso de que ésta deba buscar un operando, el campo DIR proveerá al bus S el valor de la ubicación de la operando en memoria, el que será leído y llevada a la UAL para su cálculo. Siendo n en las siguientes figuras; la suma entre p y m, los cuales son los bits de información.









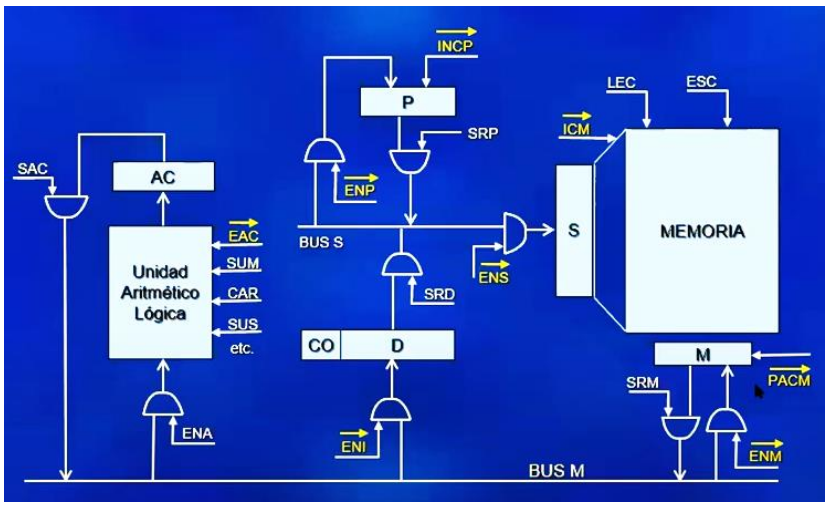
El bus de dirección S de m bits, permite transferir el contenido de P a S en el caso de la búsqueda de la siguiente instrucción, o bien el contenido de D del registro de Instrucción a S en el caso de la Búsqueda del Operando de la instrucción en curso para su resolución

La Memoria principal está compuesta por 2^m registros o palabras de n bits. Pueden ser seleccionadas independientemente por su posición en relación al valor del registro S a la hora de Lectura o Escritura. Se relaciona directamente con el registro M de intercambio o palabra, y a los restantes Órganos a través del Bus de Datos e Instrucción M

Registro de Instrucción i de n bis compuesto por: Código de Operación (p bits) y Dirección (m bits) El campo D permitirá direccionar 2^m operandos

Registro Acumulador de n bits. Opera contra el contenido del registro M y el resultado es escrito sobre AC, sobre-escribiendo su contenido. Ejemplo: si C.O es SUMA entonces (M)+(AC)→AC

Recordando señales de impulso y de nivel se describirán las **señales de gobierno** (característica más importante) de esta arquitectura Abacus, las de amarillo son de impulso y las blancas de nivel. La unidad de control es aquella que controla las señales de gobierno para que se realicen las respectivas instrucciones. Los cálculos deben ser de nivel porque no alcanza con un pulso para realizar la operación.



Instrucciones:

SAI: Salto incondicional, estricto.

SAC: Salto condicional, depende del signo del AC(-salta; +anula operación).

Ruta de instrucción:

FASE 1: Búsqueda de instrucción.

INCP: Para pasar a la siguiente instrucción.

SRP: Sal.Reg.P habilita para que se pase la instrucción al BUS S.

ENS: Ent.Reg.S copia lo del bus S y lo pasa a S.

ENP: Ent.Reg.P copia lo del bus S y lo pasa a P. Se la utiliza para los saltos cond-incond.

ICM: Inicio ciclo de memoria, activa el decodificador y marca el incio de un ciclo de memoria.

LEC: Lectura, lleva más de un ciclo de memoria.

ESC: Escritura, lleva más de un ciclo de memoria.

PACM: Puesta a cero registro memoria.

ENM: Ent.Reg.M, copia el bus de M y lo pone en el registro memoria.

SRM: Sal.Reg.Mem, a través de una lectura la palabra leída se mueve al registro de memoria y se la pasa al busM.

ENA: Ent.Ac, le puede llegar una instrucción(es porque está en el ciclo de instrucción) u operando(ya hubo una instrucción que requiere de un operando). PD: Hubiera sido impulsional pero necesitaría de un registro en ALU.

ENI: Ent.Reg.D, permite copiar el contenido del bus M al registro de instrucción.

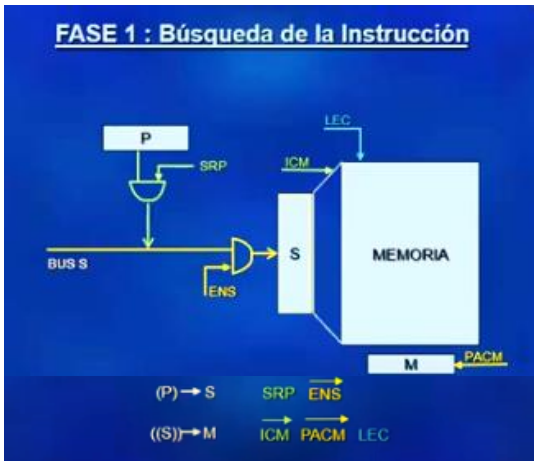
SRD: Permite mover el contenido de la dirección del operando sobre el bus S; y también permite mover el contenido D al bus S y por medio de ENP moverlo hasta P y realizar el salto condicional.

EAC: Mueve el resultado de la ALU a AC.

SAC: Sal.Reg.AC, pone el busM el valor de AC.

CAR: Carga el AC con una dirección de memoria.

Cálculos: SUS(sustracción), SUM(suma) OR, XOR, AND, NOT.

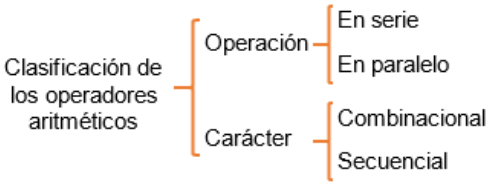


UNIDAD ARITMÉTICO LÓGICA

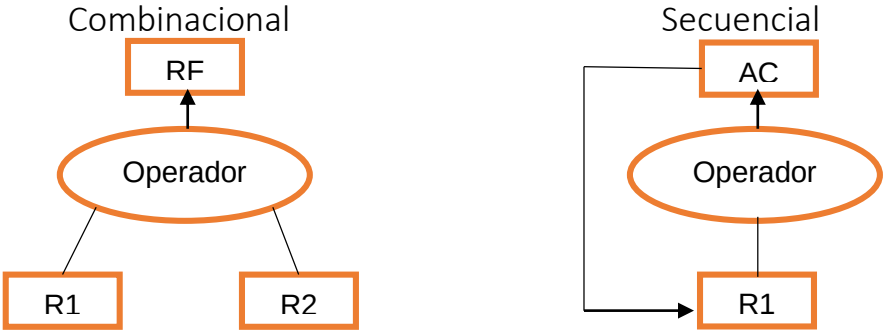
Es la encargada de ejecutar todo tipo de instrucciones de cálculo. Para cumplir esta función dispone de operadores o unidades funcionales, destinadas a realizar una o varias operaciones aritméticas o lógicas. Es un órgano compuesto de muchísimos niveles de circuitos lógicos combinaciones y secuenciales dedicados a resolver operaciones de cálculos aritméticos y lógicos entre las unidades funcionales dedicadas a esta tarea. Es el motor fundamental para determinar la performance del computador.

La unidad aritmético-lógica (U.A.L.) se compone de una unidad capaz de ejecutar todo el surtido de instrucciones del calculador o de *varias unidades funcionales* u *operadores*, cada uno especializado en la ejecución de una o varias clases de operaciones.

Un operador aritmético es circuito lógico encargado de realizar un cálculo entre dos registros. Clasificación según su régimen de operación y su carácter:



Carácter, la operación se hace con o sin acumulador



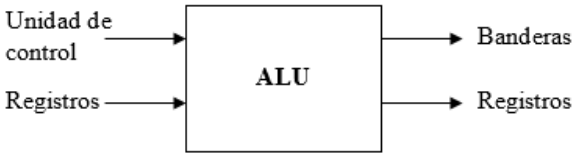
Los operadores pueden clasificarse en función del carácter combinacional o secuencial de su concepción. Los primeros ejecutan las operaciones en una sola fase y por esto pueden ser gobernados por señales de nivel, que permanecen activadas durante toda la operación.

En los segundos, las operaciones se ejecutan en varias fases gobernadas por impulsos distribuidos por un dispositivo de control (que puede ser la unidad de control del computador o un órgano del operador). Los operadores constan entonces de dispositivos de memorización de los resultados parciales. Una solución intermedia consiste en disponer de un operador combinacional asociado a un registro acumulador o totalizador, que sirve para la memorización del primer operando durante la operación y para la memorización del resultado, después.

Ventaja en serie: con un solo sumador y en función al tiempo haces entrar las parejas de bits de una cadena y vas sumando, pero tenés un solo circuito. En paralelo es más veloz que en serie pero es más caro.

Habitualmente se asocian al operador aritmético indicadores que suministran informaciones acerca de la última operación realizada. Se encuentran indicadores de error: desbordamiento o sobre-pasamiento de

capacidad, división por cero, etc. e indicadores del estado del acumulador: positivo o negativo, igual a cero, etc. Estos indicadores constituyen lo que se ha dada en llamar código de condición, porque representan informaciones a verificar con ocasión de las instrucciones de salto condicional.



Todos los otros elementos del sistema computacional – UC, registros de memoria, E-S – principalmente llevan datos a la ALU para que esta los procese y luego devuelva los resultados. Está basada en el uso de dispositivos lógicos digitales simples que pueden almacenar dígitos binarios y ejecutar operaciones de lógica simple de Bolean.

Los datos se presentan a la ALU en registros, y los resultados de una operación se almacenan en registros. Éstos son localidades de almacenamiento temporal dentro de la CPU que están conectados por vías de señal a la ALU. La ALU también pondrá banderas como resultado de una operación; los valores de as banderas se almacenan también en registros dentro de la CPU. La unidad de control proporciona señales que controlan la operación de la ALU, y el movimiento de los datos hacia dentro y hacia fuera de la misma.

Cada operación debe tener un indicador sobre la operación. Informan a la arquitectura de éxito o no de una operación o diversas anomalías que puedan ocurrir. Indicadores del Operador Aritmético:

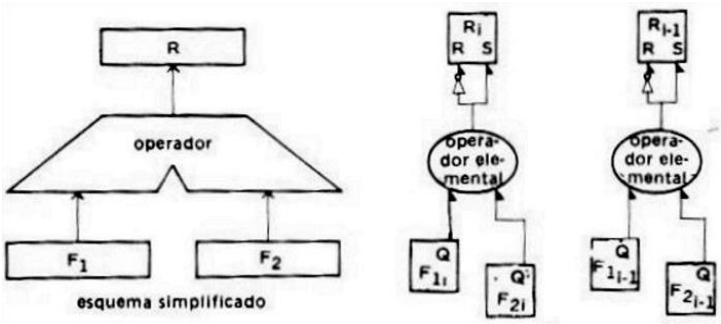
- **Última operación realizada**. Permite informar si fue realizada o no la última operación, sirve para saber si se encuentra disponible o no el operador.
- **Error de Desbordamiento**. Indican que el valor, el resultado del acumulador se excede por un digito y es necesario hacer un desplazamiento para acomodarlo.
- **Error de Desborde de Capacidad**. Es cuando directamente el registro se ve totalmente desbordado, entontes la unidad de control debe tomar alguna acción para poder preservar ese desbordamiento que excede a más de un bit.
- **Error de División por Cero**. La ALU entra en error si intenta de dividir algo por 0.
- **Estado Acumulador Positivo**.
- **Estado Acumulador Negativo**.
- **Estado Acumulador Igual a 0**.

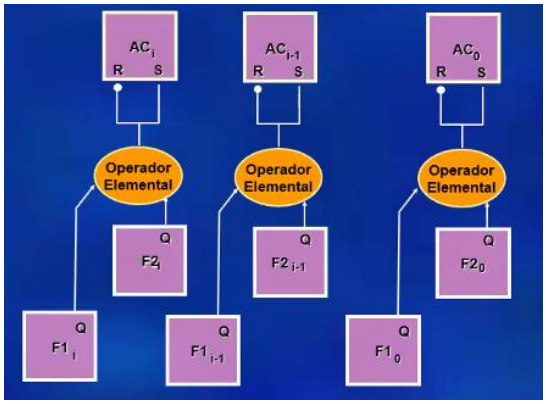
Son fundamentales para tomar decisiones de salto por positivo, por negativo o por cero. Es la única base para actuar con inferencias lógicas. Son importantísimos al tener en cuenta su diseño

Unidad Aritmético-Lógica Elemental

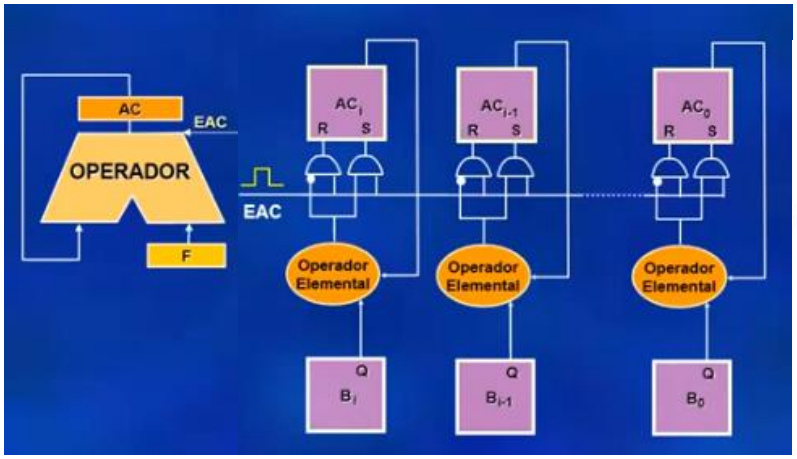
Para el caso de operaciones entre bits del mismo peso, se encuentran dos montajes posibles de los operadores.

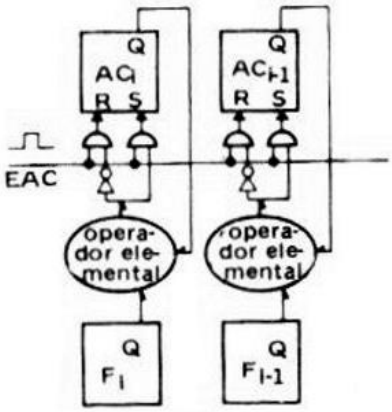
Operador Combinacional: Los operadores se montan entre dos registros fuentes F1 y F2 (o entre dos buses fuentes) para los operandos y un registro R para el resultado (o un bus para el resultado). Obsérvese que, sobre este esquema, el resultado solo es válido en tanto los operandos fuentes permanezcan posicionados.

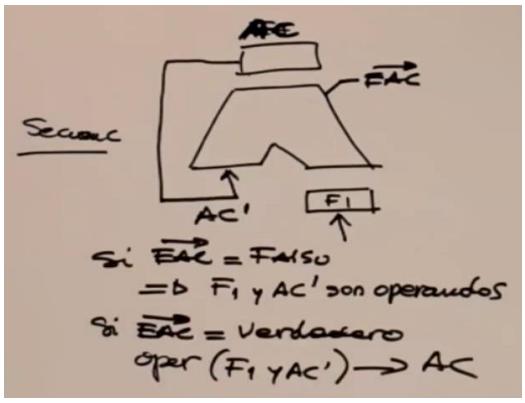




Operadores con Acumulador (secuencial): no exigen más que un registro fuente para almacenar uno de los operandos, ya que el otro es memorizado por el acumulador durante todo el tiempo de la operación. Cuando a la salida de los operadores elementales quedan establecidos los niveles lógicos, un impulso EAC introduce el resultado en el acumulador (EAC=Verdadero). EAC=Falso si el operador está calculando sobre valor del acumulador en el instante anterior.







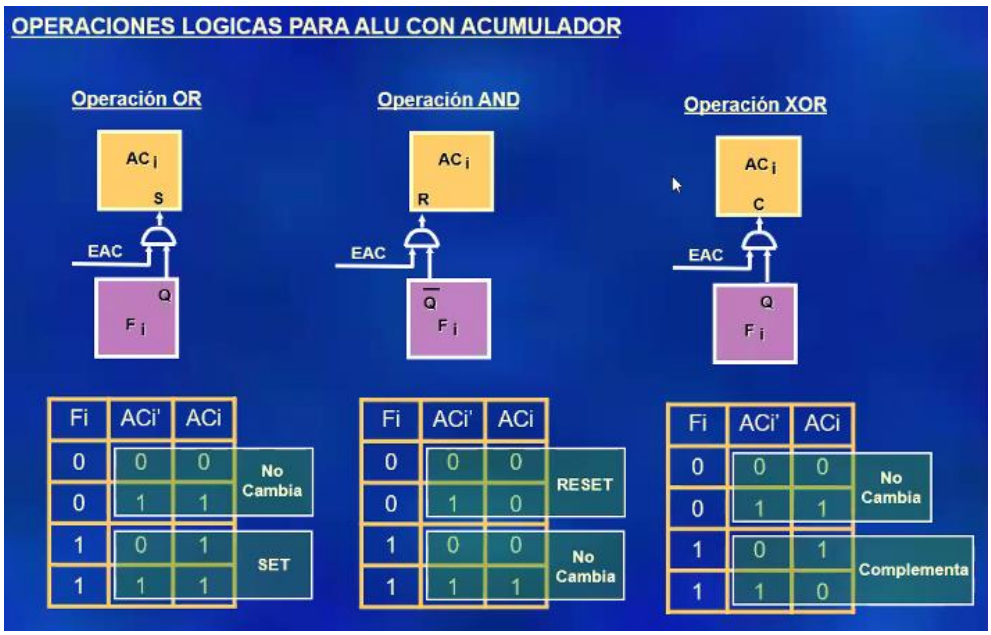
Operaciones Lógicas

Fuera de la complementación, que se consigue por circuitos NOT en el caso del operador combinacional y por la complementación del acumulador, en el caso del operador con acumulador, las demás operaciones se reducen esencialmente a operaciones AND, OR y ORX.

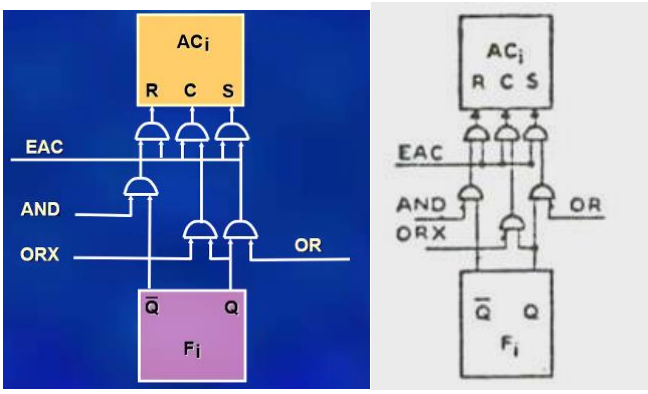
Operador Combinacional: *Utiliza el mismo esquema del operador combinacional* visto anteriormente sustituyendo los operadores elementales por el circuitos AND, si se trata de la operación de intersección, OR si se trata de la operación reunión, ORX (OR exclusivo) si de la operación diferencia simétrica.

Operador con Acumulador: Se complica porque no tiene dos entradas, una de sus entradas es el mismo acumulador. La organización presentada con la figura del operador con acumulador anterior puede simplificarse, si se parte de la idea siguiente: en vez de ejecutar la operación con los bits homólogos del registro fuente y del acumulador, se observa que basta, según el valor del bit fuente Fi, con dejar inalterado el bit del acumulador o forzarle a un valor no dependiente más que de Fi. Por ejemplo: para el AND, el bit acumulador continua inalterado si Fi = 1, pero debe ponerse a cero si Fi = 0.

En sí, no existe un FF de RCS, es una especie de FF JK. C lo que hace es complementar el contenido del acumulador.

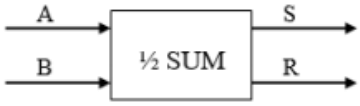


Operador Lógico completo: EAC Capta el momento del resultado.



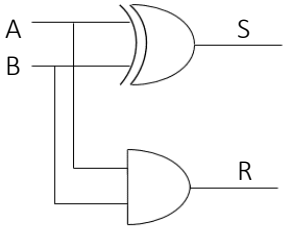
Semisumador

El semisumador es un circuito lógico que realiza la suma de dos bits del mismo peso sin acarreo. Con dos salidas, una como la suma (S) y otra como el acarreo (R). Siempre va a estar ubicado en la primera posición de una suma (por no poseer el acarreo del anterior) Sirva para hacer la suma de los primeros bits de una cadena.

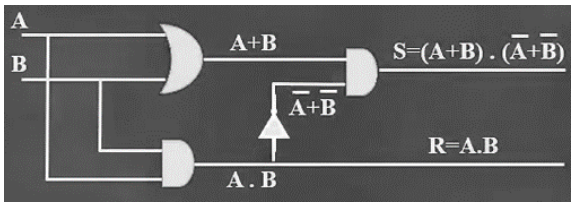


Símbolo

Entradas		Suma	
A	B	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Forma simplificada



Forma complicada

Ecuaciones:

$$S = (\overline{A} + B).(A + \overline{B}) = A \oplus B$$

$$R = A.B$$

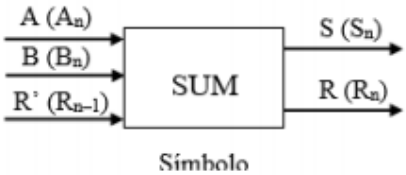
Sumador completo

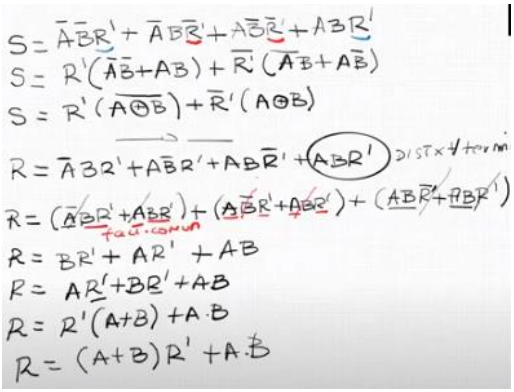
Para realizar la suma de dos dígitos binarios es preciso tener en cuenta, en cada posición n, los dos dígitos de dicha posición y el arrastre de la posición $n-1$.

Llamamos etapa de sumador al operador que realiza la adición de 2 dígitos binarios y del arrastre eventual que consiste en 3 entradas (A, B, R') y 2 salidas (S, R).

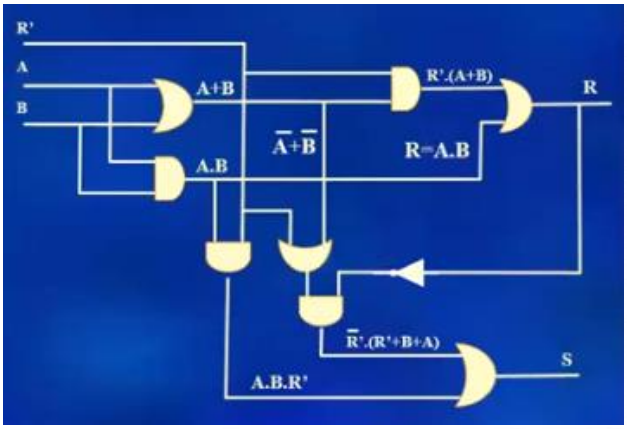
La salida S vale 0, si todos los bits de entrada son ceros, en cambio, es igual a 1 si solamente una entrada es igual a 1 o si tres entradas son 1. Por otra parte, la salida R tiene un bit de arrastre de 1 si dos de las tres entradas son iguales a 1.

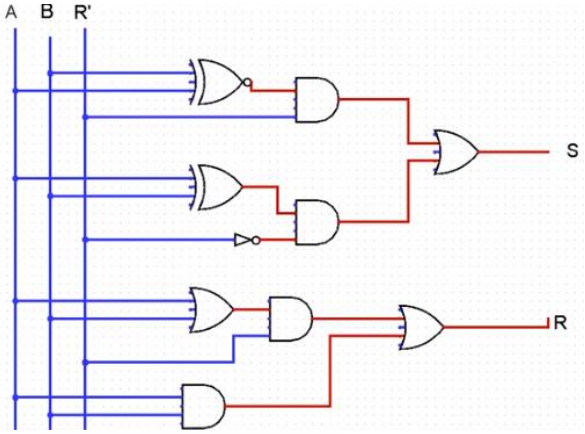
Entradas			Salidas	
A	B	R'	R	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1





Para construir una etapa de sumador a partir de 2 semisumadores: sumemos A y B en un primer semisumador, luego al resultado S_1 le sumamos R' proveniente de la etapa anterior; obtenemos el resultado S de la suma y dos arrastres: R_1 generado en la etapa, R_2 propagado por la misma; los reunimos mediante un OR lógico y obtenemos el arrastre de la suma.





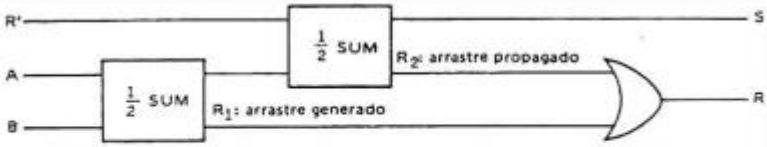


Fig. 62. Una etapa del sumador realizada con ayuda de semisumadores

Suma: $S = (A \oplus B) \oplus R'$

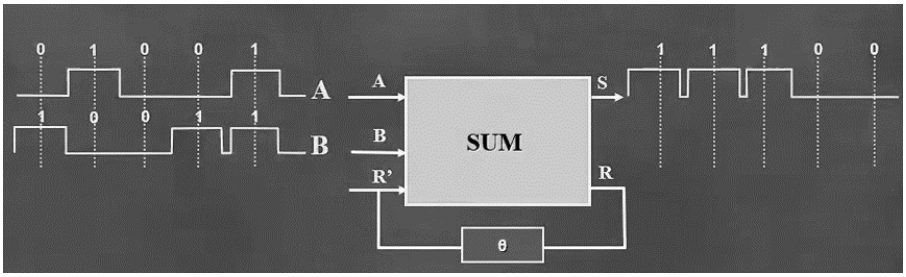
Arrastre: $R = (A \oplus B)\, R' + AB$

Sumador en serie

Las operaciones en los computadores digitales se hacen principalmente en paralelo ya que éste es un modo más rápido de operación. Las operaciones en serie son más lentas y requieren menos equipo. El circuito sumador en serie usa solamente un circuito sumador completo y un elemento acumulador para conservar el

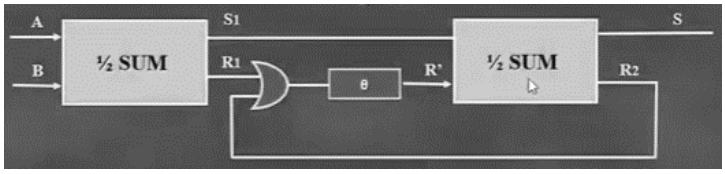
arrastre de salida generado. El método en paralelo usa n circuitos sumadores completos y todos los bits de A y B se aplican simultáneamente.

Las entradas A y B son trenes de pulso, las cuales contienen las cadenas que se pretenden sumar. Ambas cadenas están perfectamente sincronizadas. El acarreo queda retardado por un temporizador a la espera del próximo tren de pulso (la próxima parejita de bits). Tiene un único sumador y suma en cada batido.



Ventaja: Simpleza y economía del circuito con un solo sumador de dos compuertas (ORX, ORX) más una suma y multiplicación lógica (AND y OR) logro hacer un sumador completo con un circuito que permita llevar el acarreo.

De una forma más económica según Meinadier, también puede evitar el lag.



Desventaja: Consume muchos ciclos de CPU y mantiene ocupado al procesador lo que dure la suma, dependiendo del tren de entrada N ciclos y si lo guardo N+1.

¿Cómo funciona? Primeramente, los dos números binarios son agregados en serie y se almacenan en dos registros de desplazamiento. Los bits se suman un par al tiempo de forma secuencial, por medio de un circuito sumador completo (FA). El bit de arrastre del circuito sumador completo se transfiere al flip-flop D. La salida de este flip-flop se usa como arrastre de entrada para el siguiente par de bits significativos. El contenido de los dos registros de desplazamiento se desplaza a la derecha por un período de tiempo de palabra. Los bits de suma de la salida S del sumador completo pueden ser transferidos a un tercer registro de desplazamiento. En este caso, lo transferimos al registro A, por lo tanto, utilizamos el registro A para almacenar el sumando y los bits de suma.

Los bits de suma de salida S ingresan por la entrada serial (SI) del registro A, siempre que la señal de desplazamiento esté habilitada, produciéndose así el desplazamiento hacia el exterior de los bits del registro A, por la salida serial (SO) de dicho registro.

De la misma manera la entrada serial (SI) del registro B es capaz de recibir un número binario nuevo mientras que los bits de este registro se desplazan hacia fuera durante la suma.

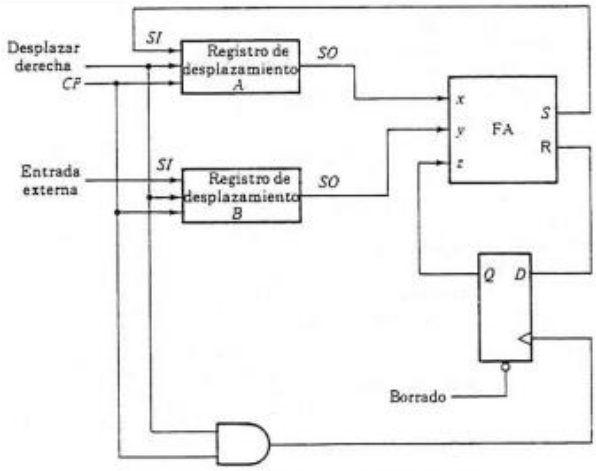


Figura 7-10 Sumador en serie

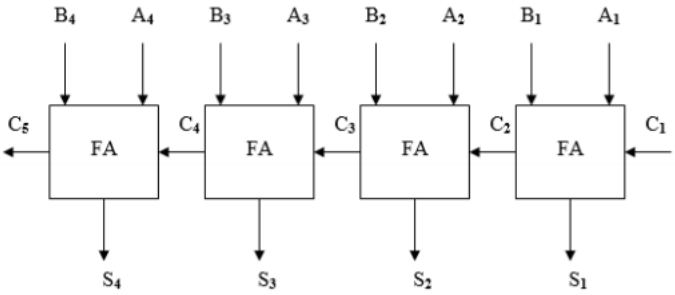
Entonces, la operación del sumador en serie consiste en:

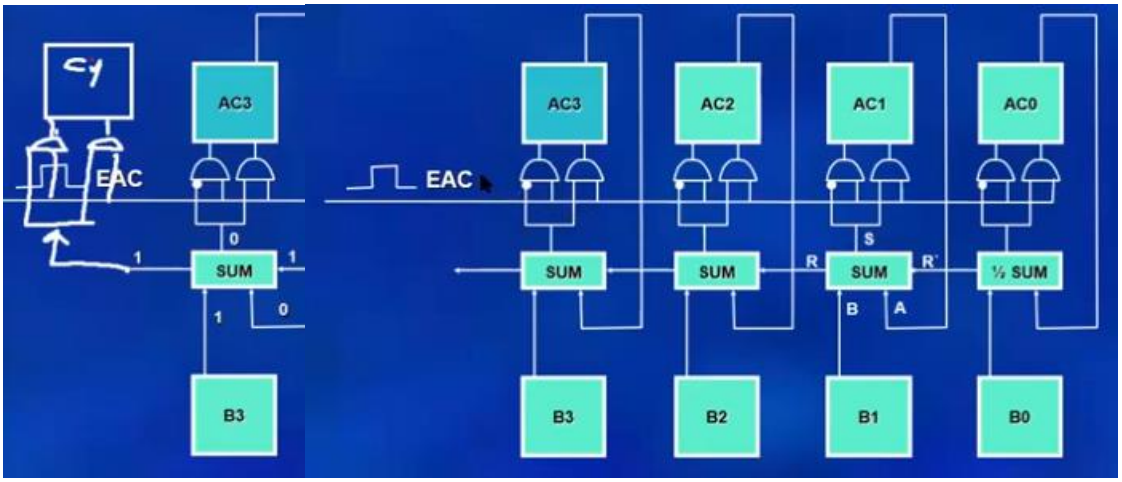
1. Los registros A y B almacenan los sumandos. El flip-flop D se pone a cero.
2. Las salidas seriales SO de A y B suministran un par de bits significativos para el sumador completo en X e Y.
3. La salida Q del flip flop da el arrastre de entrada Z.
4. El control de desplazamiento a la derecha habilita ambos registros y al flip flop del bit de arrastre.
5. El siguiente pulso del reloj ambos registros se desplazan a la derecha, el bit suma S entra en el flip flop de la extrema izquierda del registro A y el arrastre de salida se transfiere al flip flop D.

El proceso de desplazamiento a derecha continúa hasta que el control de desplazamiento a la derecha se inhabilita. Si se quiere realizar otra suma con el contenido del registro A, primero se debe transferir en serie el número deseado al registro B.

Sumador en paralelo

Un sumador paralelo binario es una función digital que produce una suma aritmética de dos números binarios en paralelo. Este consiste en sumadores completos conectados en cascada con la salida de arrastre de un sumador completo conectado al arrastre de entrada del siguiente sumador completo ubicado a la izquierda. Un sumador completo de n bits requiere n sumadores completos conectados en cascada. Es mucho más caro, pero puede resolver la suma en un batido.





Comparando sumador en serie vs sumador en paralelo.

El sumador en paralelo debe usar registros con capacidad de carga en paralelo, mientras que el sumador serial usa registros de desplazamiento. El número de circuitos del sumador completo en el sumador en paralelo es igual al número de bits en los números binarios, mientras que el sumador en serie requiere solamente un circuito sumador completo y un flip-flop para el arrastre.

Excluyendo los registros, el sumador en paralelo es un circuito combinacional, mientras que el sumador en serie es un circuito secuencial. El circuito secuencial en el sumador serial consiste en un circuito sumador completo y un flip-flop que acumula el arrastre de salida. Esta es una operación en serie típica porque el resultado de una operación de un tiempo de bit puede depender no solamente de las entradas presentes sino de las entradas previas (arrastre).

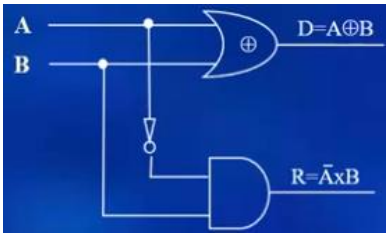
Semisustractor

El semisustractor resta del dígito X el dígito Y. El semisustractor es un circuito combinacional que resta dos bits, tiene 2 salidas, una genera el resultado de la diferencia D y la otra indica si se ha prestado un 1 (B: Borrow) Se designa el bit minuendo con X y el bit sustraendo con Y. La salida prestada B es 0 siempre y cuando $X \geq Y$, y será 1 para $X = 0$ e $Y = 1$. La salida D es exactamente la misma que la salida S del semisumador.

Diferencia: $D = X \oplus Y$
Prestado: $B = X' Y$

X	Y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Símbolo

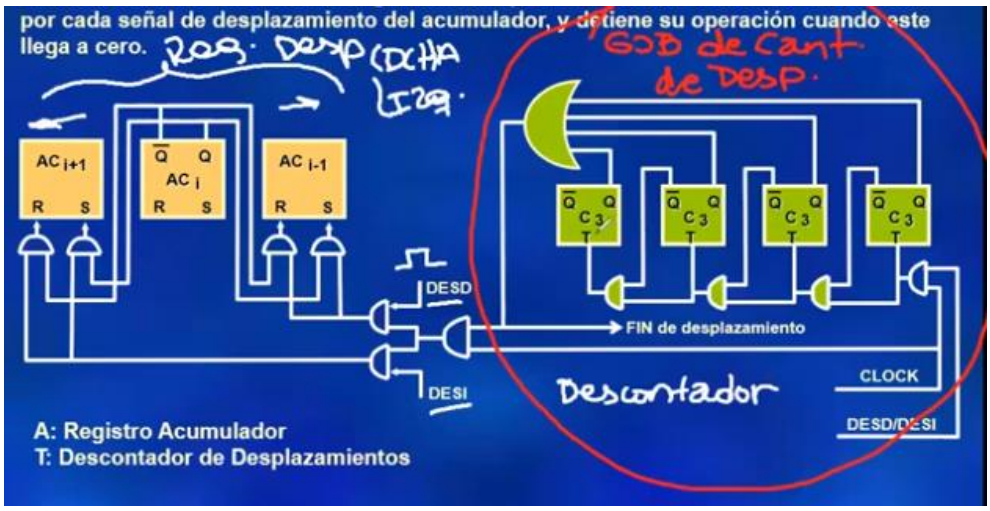


No se lo utiliza ya que con el sumador puedo realizar la misma operación con artificios lógicos (complemento a la base del sustraendo).

Gobierno Secuencial de Desplazamiento

Este circuito es el gobierno de cantidad de desplazamiento. Consiste en un registro de desplazamiento le permite ir a derecha o a izquierda, depende desde dónde se active (DESD o DESI). El valor binario en el descontador le dice cuántas veces se puede desplazar.

Es un acumulador montado como un registro desplazamiento. El contador se ve retrasado por cada señal de desplazamiento del acumulador, y detiene su operación cuando este llega a 0 (es decir, detiene la posibilidad de desplazar a izquierda o derecha).



Unidad Aritmético-Lógica para ABACUS (ALU de 1 bit)

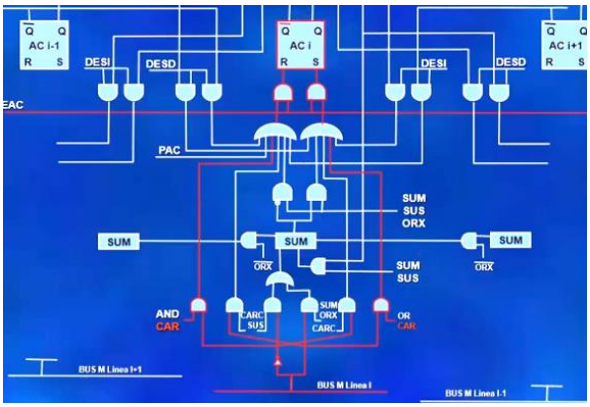
Toma como entrada al Bus M porque no tiene transferencias con el exterior ninguna información, sino hay que poner un registro. Hoy en di a es un conjunto de buses para evitar el cuello de botella.

La unidad aritmética de Abacus está montada entre un bus fuente M, donde se mantienen los niveles lógicos representativos del segundo operando, y un acumulador AC que mantiene los niveles lógicos correspondientes al primer operando hasta que la señal de muestreo EAC introduce el resultado en el acumulador.

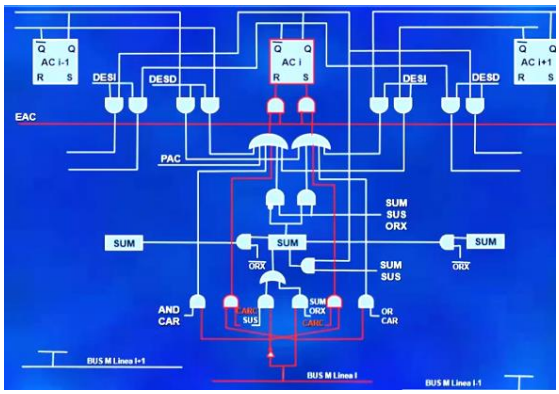
Consta de un sumador en paralelo y de un conjunto de puestas para ejecutar las operaciones lógicas y distribuir las informaciones de acuerdo con las operaciones aritméticas a efectuar. Además, el acumulador está montado como registro de desplazamiento.

Las diferentes operaciones están gobernadas por señales lógicas, procedente normalmente del generador central de secuencias del ordenador. Todas estas señales son de nivel, de tal suerte que al cabo de un cierto tiempo después del posicionamiento el resultado de la operación se estabiliza bajo la forma de niveles lógicos, que serán introducidos en el acumulador por el impulso EAC. En la figura siguiente se representa el i-simo elemento de esta unidad. Las diferentes operaciones que permite son:

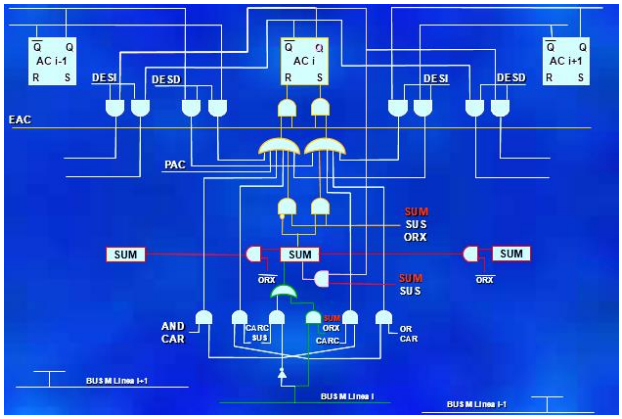
CAR *Carga:* transferencia (forzada) al acumulador de la información presente sobre el bus M



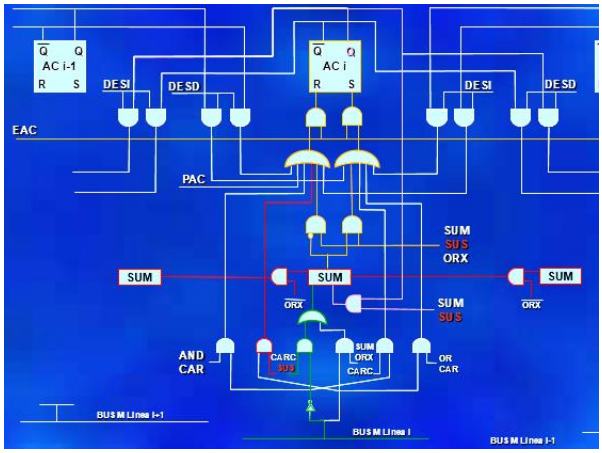
CARC *Carga con complementación:* el mismo proceso, pero después de complementar cada dígito.



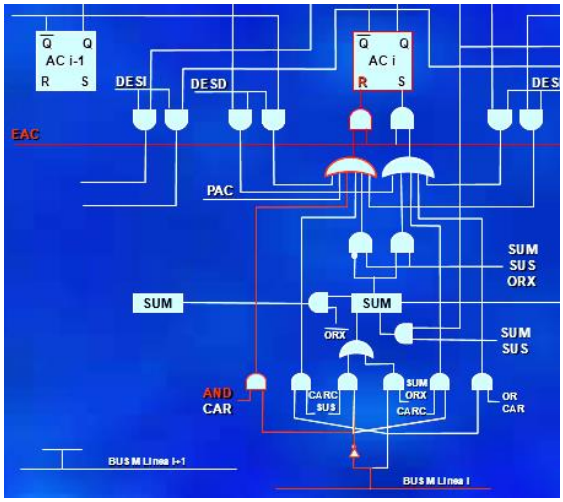
SUM *Adición:* suma al contenido del acumulador, la información presente sobre el bus M. Esta adición es operada por el sumador en paralelo y la duración de la misma es el tiempo de propagación de los arrastres.



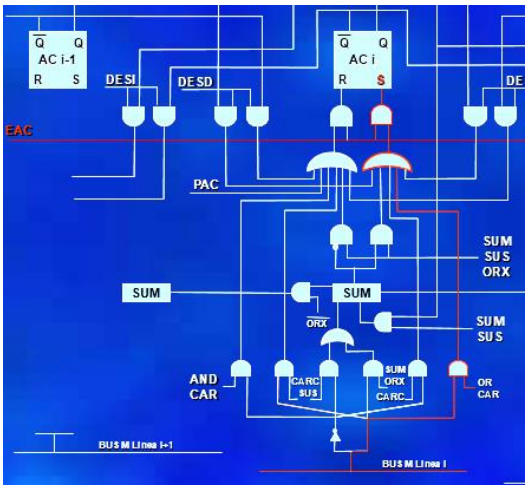
SUS *Sustracción:* se resta la información en el bus M del contenido del acumulador. Se realiza complementando la información del bus M y después sumándola en el sumador paralelo. La operación es correcta si Abacus opera en complemento a 1. Si no, se precisa posicionar un nivel 1 a la entrada de arrastre a la etapa de sumador de menor peso.



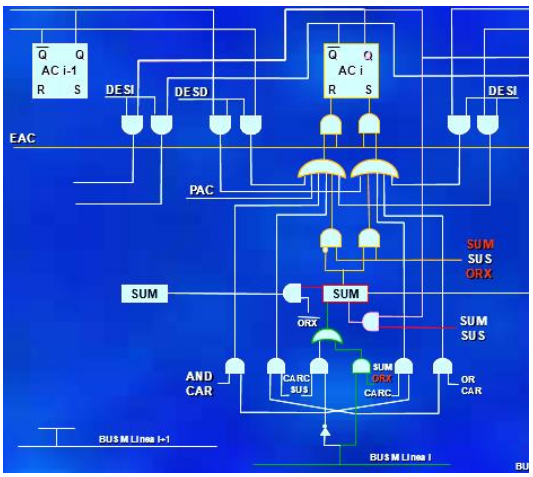
AND *intersección lógica:* entre la información en el bus M y el contenido del acumulador.



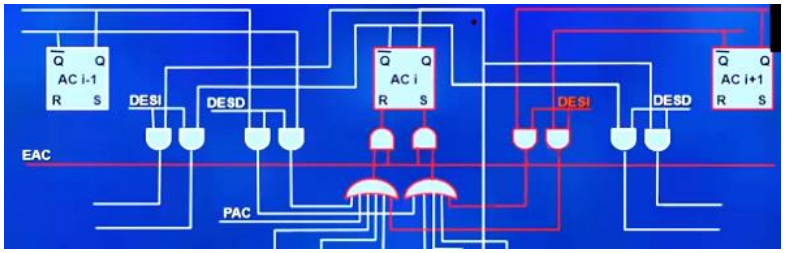
OR *reunión lógica:* entre la información en el bus M y el contenido del acumulador. Estas dos últimas operaciones se realizan según el esquema de la etapa de operador lógico.



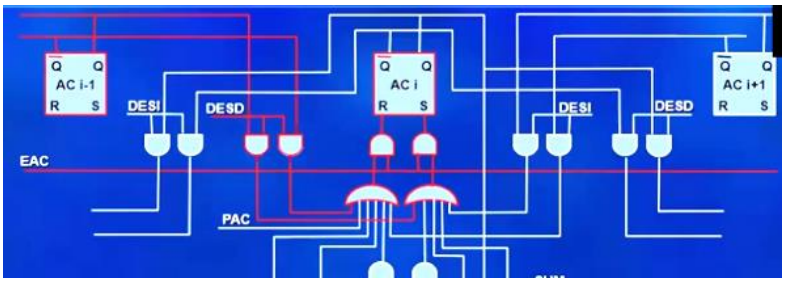
ORX *OR exclusivo:* entre la información en el bus M y el contenido del acumulador. Esta operación se ejecuta aquí por parte del sumador, inhibiendo la propagación de los arrastres.



DESI *Desplazamiento a izquierda:* desplazamiento del contenido del acumulador una posición binaria a la izquierda.



DESD *Desplazamiento a derecha:* desplazamiento del contenido del acumulador una posición binaria a la derecha.

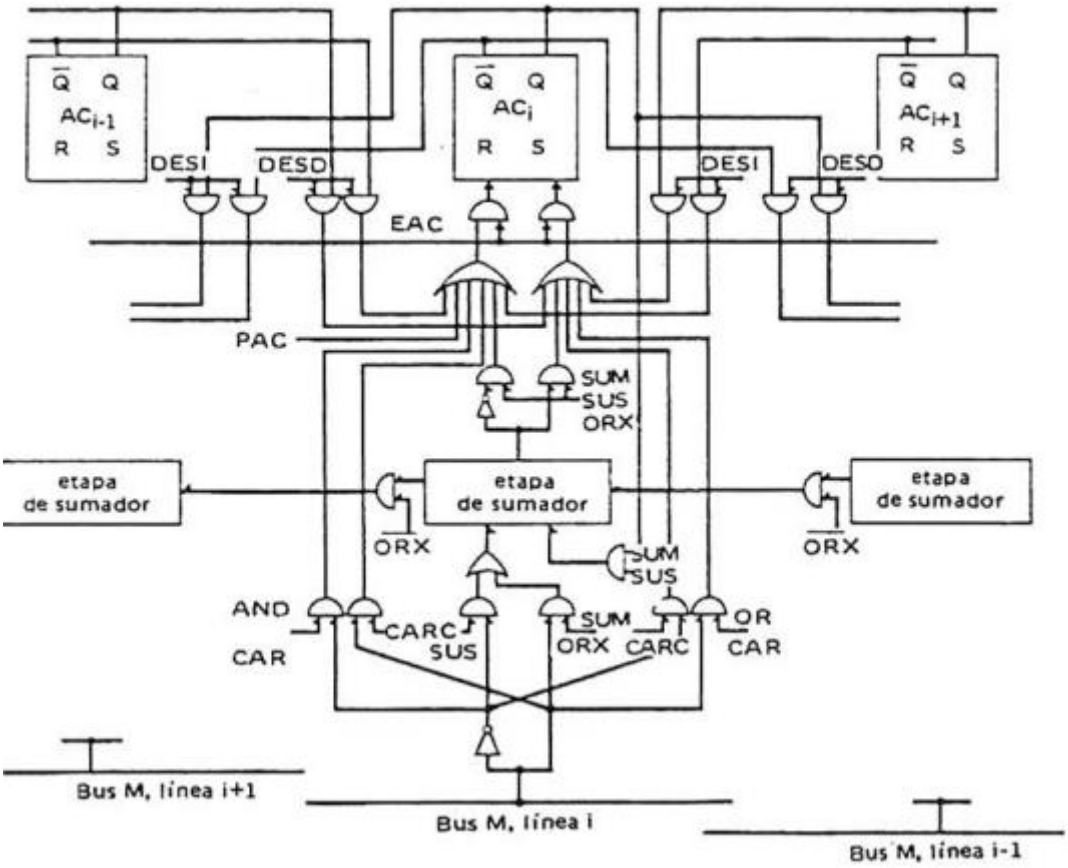


Los indicadores asociables a la unidad aritmética de Abacus podrían ser los siguientes:

DEB Indicador de desbordamiento.

S Indicador de signo.

CE Indicador de cero. Puede cablearse mediante un NOR recibiendo todas las salidas Q de los biestables del acumulador, si Abacus trabaja en complemento a 2. (En complemento a 1 se precisaría un OR detrás de un NOR para todas las salidas Q y de un NOR para todas las salidas Q').



Números Binarios con Signo

Cuando un número binario de punto fijo es positivo, el signo se represente con un cero y la magnitud con un número binario positivo.

Si el número es negativo, el signo se representa con un uno y el resto puede ser representado por cualquiera de las siguientes maneras:

+6	0 000110		-6	1 111010
+ +9	0 001001		+ +9	0 001001
+15	0 001111		+3	0 000011
+6	0 000110		-9	1 110111
+ -9	1 110111		+ -9	1 110111
-3	1 111101		-18	1 101110

Los dos números de los cuatro ejemplos se suman, con sus bits de signo incluidos. Cualquier arrastre del bit de signo se descarta y los resultados negativos se producen automáticamente en la forma de complemento de 2

Sobrecapacidad o Desbordamiento:

Debido a la longitud fija del tamaño de los registros del computador, estos deben verificar situaciones de sobre capacidad e indicar esta situación en un biestable diseñado a tal efecto.

Puede haber Sobrecapacidad o Desbordamiento después de una suma si un número es positivo y el otro es negativo.

Puede haber Sobrecapacidad o Desbordamiento después de una suma si ambos números son positivos o negativos.

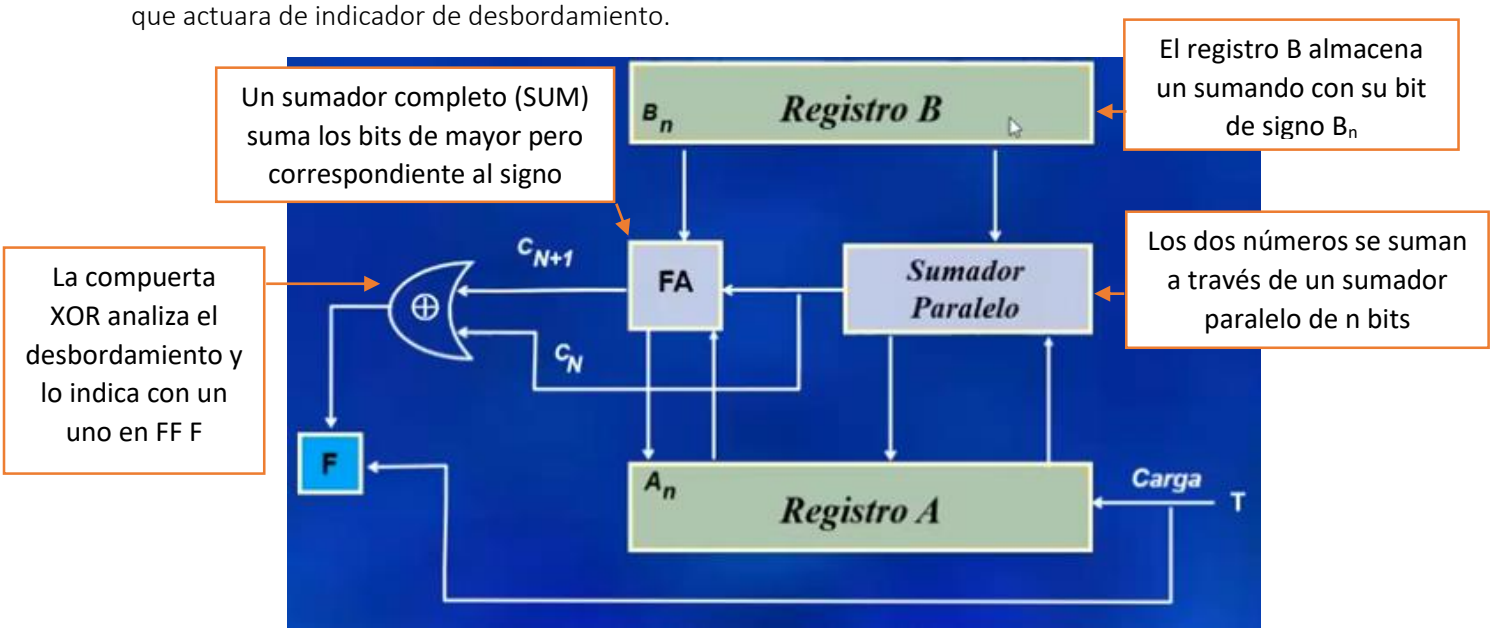
La condición de Sobrecapacidad o Desbordamiento se detecta observando el arrastre producido por el bit más significativo de la Magnitud y el del Bit del Signo. Si son distintos se produce una condición de Sobrecapacidad debiendo indicarlo en el Bit Desbordamiento

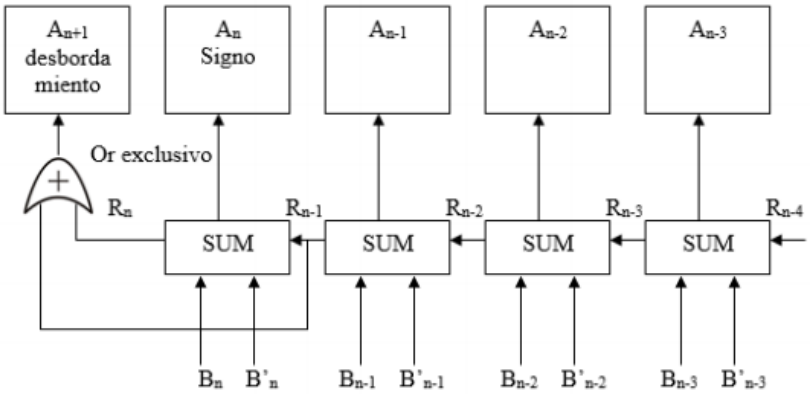
Sumador/Sustractor de Números Algebraicos Binarios en Punto Fijo

Para poder desarrollar un circuito sumador/sustractor es necesario:

- Expresar las magnitudes positivas en valor absoluto indicar el bit de mayor peso su signo a través de un 0
- Expresar las magnitudes negativas por complemento absoluto e indicar el bit de mayor peso su signo a través de un 1
- Verificar situaciones de desborde a través de analizar el arrastre del dígito de mayor peso de la magnitud junto del arrastre de la suma de los signos, detectando situaciones donde alguno de ellos sea igual a 1
- De esta manera es posible planear un circuito que permita la suma o resta a través de sumadores, ajuste el signo de la operación y detecte indique en un biestable la condición de arrastre de la operación.

Sustraer un número de otro equivale a sumarle su opuesto. La sustracción se reduce a la suma algebraica siempre que se sepa calcular el opuesto de un número, lo que generalmente se hace por complementación. Este método evita tener que situar un Sustractor junto al sumador. Nos limitaremos al estudio de la suma de dos números algebraicos al caso de la representación por complementación auténtica (complemento a 2). Supongamos que nuestra unidad aritmética procesa números representados por n bits, con el bit de signo situado en la primera posición a la izquierda. Asociaremos al acumulador un n+1- esimo bit, que actuara de indicador de desbordamiento.





Para estudiar la operación de suma algebraica se elabora un cuadro de tres entradas correspondientes a las tres configuraciones posibles de signo de los operandos y considerando el arrastre $n-1$ procedente de la etapa inmediatamente derecha del bit de signo. La presencia de este arrastre implica un desbordamiento en el caso de un resultado esperado negativo.

El indicador de desbordamiento debe posicionarse solo en dos casos: cuando no hay arrastre de orden $n-1$ pero hay arrastre de orden n o bien cuando hay arrastre de orden $n-1$ pero no de orden n.

Condiciones de signo	2 operandos > 0 El resultado debe ser positivo	1 operando > 0 1 operando < 0. No puede haber desbordamiento	2 operandos < 0 El resultado debe ser negativo
Representación inicial (registros de 5 bits)	0---- 0----	1---- 0----	1---- 1----
Resultado si no hay arrastre procedente de $n-1$	0---- Válido: resultado positivo	1---- Válido: resultado negativo	10--- No válido puesto que resultado positivo, hay desbordamiento.
Resultado si hay arrastre procedente de $n-1$	1---- No válido: puesto que resultado negativo, hay desbordamiento	10--- Válido: olvidándose del arrastre. Resultado positivo	11--- Válido: olvidándose del arrastre. Resultado negativo

Aritmética Binaria en Coma Flotante

Recordemos que los números en coma flotante se representan en la forma $S\mathrm{M}\alpha^E$, donde S es el signo del número, M la mantisa, E el exponente y α la base (generalmente 2 o 16 en aritmética binaria). Las operaciones se llevan a cabo sobre números normalizados, esto es, sobre números cuyo valor de exponente está ajustado para que la mantisa tenga el mayor número posible de dígitos significativos (cuando el valor de binario es su signo y el bit más significativo de su mantisa son distintos.). Por consiguiente, los operadores en aritmética flotante deben, no solamente ejecutar la operación correspondiente, sino también normalizar el resultado obtenido.

S	E	M

La primera fase consiste en *alinear las mantisas* con el objeto de enfrentar los bits del mismo peso. Supongamos a sumar los números A = 12000 x 10^{-1} y B = 80000 x 10^{2}

A

+	0	0	1	2	0	0

y

B

+	2	8	0	0	0	0

No puede operarse directamente la suma de las mantisas, se comparan los exponentes y se detecta que el primero de los números es más pequeño, sobre este número se ejecuta una doble operación:

incremento del exponente en una unidad y desplazamiento de la mantisa una posición a la derecha. De nuevo se comparan y se recomienza hasta obtener identidad de exponentes. En ese momento puede efectuarse la suma de las mantisas:

A	+	2	0	0	0	1	2
B	+	2	8	0	0	0	0

La segunda fase consiste en *ajustar los signos*. Consideremos la sustracción de A –B. En la representación en valor absoluto y signo, se deben primeramente comparar las mantisas A y B, comprobar que la de B es superior a la de A y deducir de ello que la operación pertinente es sustraer la mantisa de A de la de B y al resultado darle el signo menos.

A	+	2	0	0	0	1	2
- B	+	2	8	0	0	0	0

		M_B	8	0	0	0	0
		- M_A	0	0	0	1	2
R	-	2	7	9	9	8	8

Debe tomarse así en cuenta en esta fase la clase de operación (suma o sustracción), el signo de los operandos y el resultado de la comparación de mantisas. El problema se simplifica si el conjunto signo-mantisa fuera representado en complementación cuando el número sea negativo.

La tercera fase es la *normalización del resultado*

Puede que el resultado bruto de la operación no esté normalizado, por dos razones:

- Hay desbordamiento de capacidad en la suma, en cuyo caso se deberá desplazar la mantisa una posición a la derecha, aumentando al mismo tiempo en una unidad el exponente.
- La diferencia de las mantisas implica un número de n ceros a la izquierda del resultado, en cuyo caso se deberá desplazar la mantisa n posiciones a la izquierda, restando simultáneamente n del exponente.

1er operando:	+	2	8	0	0	0	0
2do operando:	+	2	8	0	0	0	0
Operación:		1	6	0	0	0	0
Normalización:	+	3	1	6	0	0	0

1er operando:	+	2	8	0	0	0	0
2do operando:	-	2	7	9	9	9	9
Operación:			0	0	0	0	1
Normalización:	+	-2	1	0	0	0	0

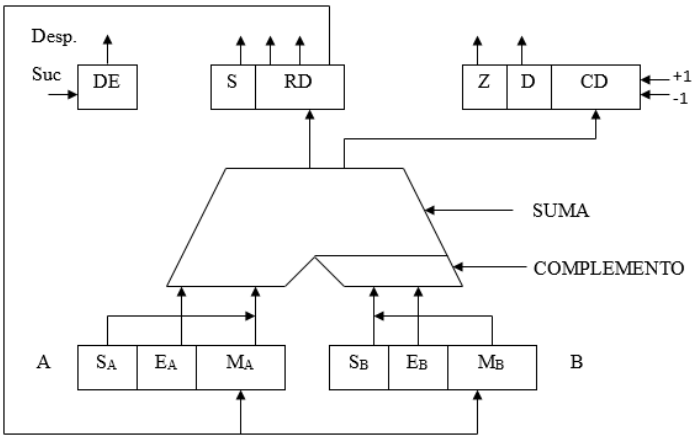
Funcionamiento de un Operador de Suma Flotante

Operando	S	E	M

S → 1 bit signo del operando S = 0 (signo más); S = 1 (signo menos)

E → e bits exponente; representa una potencia de 2 y está comprendido entre -2^{e-1} y $2^{e-1}-1$; el exponente -2^{e-1} está representado por E = 0 y el exponente $2^{e-1}-1$ está representado por E = $2^{e}-1$.

M → m bits mantisa; la mantisa de un operando negativo está representada en complemento a 2. Un operando normalizado se caracteriza por el hecho de que el bit de signo y el bit de mayor peso de la mantisa tienen valores diferentes.



Este operador utilizado comprende dos registros A y B para los operandos y un sumador m + 1 bits que ejecuta las sustracciones por complementación previa del segundo operando.

Se emplea de una parte, para comparar los exponentes por sustracción cuyo resultado va a un registro CD montado como contador-descontador binario con bit de desbordamiento, y por otra parte, para las operaciones con signo y mantisa, cuyo resultado va a un registro RD montado como registro de desplazamiento a derecha e izquierda con bit de desbordamiento (en general se utiliza el sumador de la unidad aritmética, al que se le añade eventualmente un sumador de exponentes).

La operación se divide en varias fases:

1. **Comparación de los exponentes**: Se resta E_B de E_A sumando a E_A E_B complementado, y el resultado va a CD. Si el resultado es nulo, las dos mantisas están correctamente alineadas y se pasa inmediatamente a la fase de suma o sustracción de las mantisas. Si no, el contenido de CD permitirá contar los desplazamientos necesarios para alinear las mantisas.

2. **Alineamiento de mantisas**: Esta operación se realiza sobre el operando cuyo exponente es menor. Si $E_A < E_B$ el resultado de la sustracción de los exponentes es negativo y se caracteriza por el valor 1 del bit de desbordamiento. El complemento del contenido de CD indica el número de desplazamientos por efectuar. CD se usará entonces en funciones de contador, y los desplazamientos se sucederán hasta un nuevo desbordamiento.

El desplazamiento de la mantisa se efectúa en RD, entonces se envían los contenidos de S_A de M_A yuxtapuestos a RD y se opera por desplazamiento aritmético hacia la derecha (lo que permite reproducir el signo en los bits no significativos y por lo tanto conservar la representación complementada).

En el caso $E_A > E_B$, el desplazamiento aritmético a la derecha afecta a S_B y M_B, pero en esta ocasión el registro CD se emplea como descontador, deteniéndose los desplazamientos cuando CD=0 (nótese que después de todo, pueden detenerse al cabo del m-ésimo desplazamiento ya que a partir de entonces la mantisa pierde todo su significado).

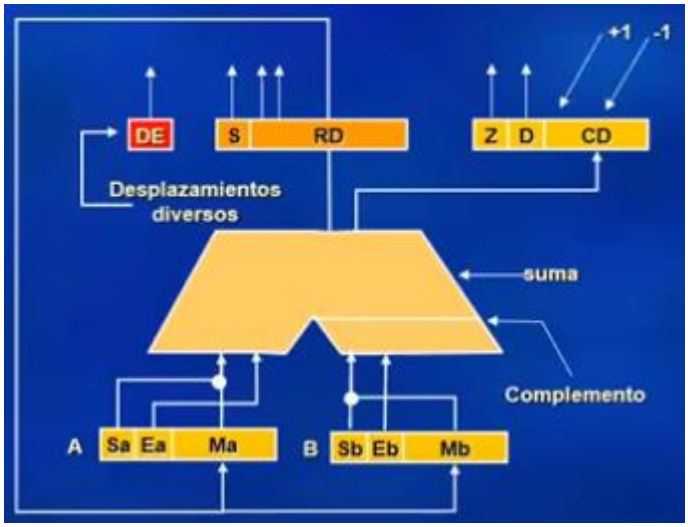
Esta fase finaliza preservando el mayor exponente en CD y devolviendo la mantisa contenida en RD al registro A o B que corresponda.

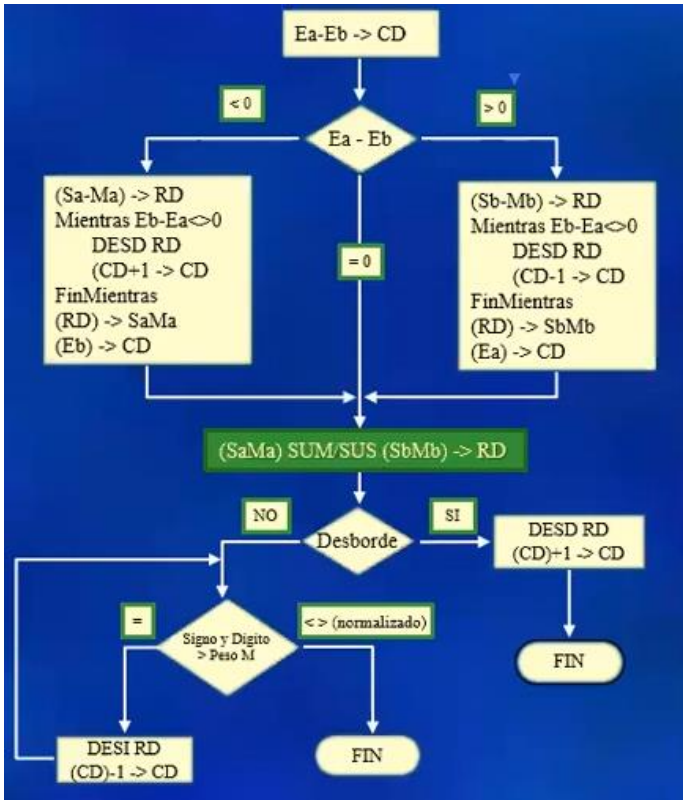
3. **Suma-sustracción de las mantisas**: Estas operaciones se ejecutan sobre el conjunto formado por el signo y mantisa, yuxtapuestos. La sustracción se obtiene sumando S_A M_A el complemento auténtico de S_B M_B. El resultado a RD, tomando en cuenta el signo y un eventual desbordamiento.

Las operaciones de normalización se realizan mediante desplazamientos sobre el registro RD, que contiene el signo y la mantisa del resultado, y sobre el registro CD, que contiene el exponente.

4. **Normalización en el caso de desbordamiento**: Consiste en desplazar la mantisa del resultado en una posición a la derecha, siendo sustituido el bit de mayor peso por el de desbordamiento. El signo queda inalterado. El contenido del registro CD se ve aumentado en una unidad.

5. **Normalización en caso que la mantisa del resultado contenga dígitos no significativos a su izquierda**: esta eventualidad es detectable haciendo un test sobre el signo y el digito de mayor peso de la mantisa. Si son diferentes, el número ya estaba normalizado. Si no, es preciso realizar desplazamientos aritméticos de la mantisa hacia la izquierda hasta que sean diferentes. A cada desplazamiento se resta una unidad del contenido de CD. El proceso se detiene incondicionalmente al cabo de **m** desplazamientos.





S: Signo del operando (0 es + y 1 es -)

E: Exponente

M: Mantisa

CD: Contador/Descontador con:
(D) Desborde / **(Z)** Indicador de Cero

Si Z=1 indica resultado 0

Si Z=0 se analiza campo D

Si D=1 resultado Ea-Eb<0
(resultados negativos se caracterizan por tener desborde)

Si D=0 resultado Ea-Eb>0

RD: Registro de Desplazamiento derecha o izquierda con **(DE)** Bit de Desborde

Multiplicación y división en coma flotante

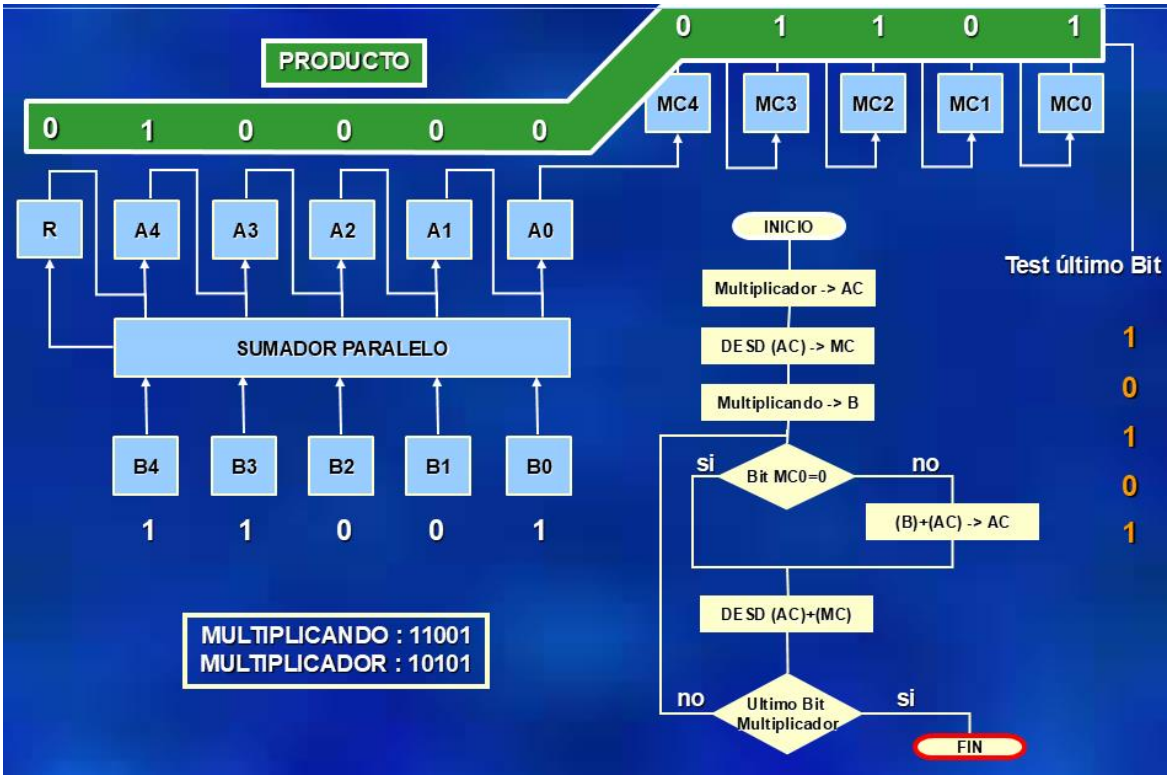
La multiplicación flotante es reducible a la multiplicación de las mantisas y a la suma de los exponentes, en tanto que la división se reduce a la división de las mantisas y a la sustracción de los exponentes. No hay operación preliminar de comparación de exponentes o de alineamiento de las mantisas. La normalización es sencilla si se compara con la suma. La multiplicación de dos números comprendidos entre ½ y 1 da un resultado entre ¼ y 1, de tal suerte que la normalización supondrá, como máximo, un desplazamiento de una posición a la izquierda. La división dará como resultado entre 2 y ½ , de forma que su normalización implicará, como máximo, un desplazamiento de una posición a la derecha. Se ejecutan estas operaciones sobre las mantisas representadas en valor absoluto.

Multiplicador Secuencial por Suma - Desplazamiento

Hay que mencionar que aquí nos limitaremos a las operaciones con los números en valores absolutos, dejando aparte el tratamiento de los signos. Consideremos el siguiente ejemplo numérico:

```
   1011     Multiplicando
   1101     Multiplicando
   1011
  0000      Productos parciales
 1011
1011
10001111    Producto
```

Los productos parciales son iguales al multiplicando si el bit correspondiente del multiplicador es 1, nulos en caso contrario. Una primera operación consiste, entonces, en verificar sucesivamente cada bit del multiplicador, de donde se deducen los productos parciales que, convenientemente desplazados, se totalizan en un acumulador. Nótese que si el multiplicador y el multiplicando tienen n bits cada uno, el acumulador necesario deberá poseer capacidad para 2n bits; por consiguiente, el resultado se obtiene en doble longitud. La unidad aritmética capaz de realizar una multiplicación debe estar dotada de un anexo al acumulador, generalmente llamado "Multiplicador-Cociente", abreviadamente MC, porque contiene el multiplicador en el caso de la multiplicación, el cociente en el caso de la división.



La operación se inicializa de la siguiente manera:

1) Carga del multiplicador en el acumulador.
2) Desplazamiento a derecha del contenido del conjunto formado por el acumulador y el registro MC, cuyo resultado es poner 0 en el acumulador y el multiplicador en MC.
3) Carga del multiplicando en B (que, eventualmente, pudiera ser sustituido por el RPM).

A continuación se ejecuta la operación siguiendo las líneas marcadas por el siguiente organigrama de principio.

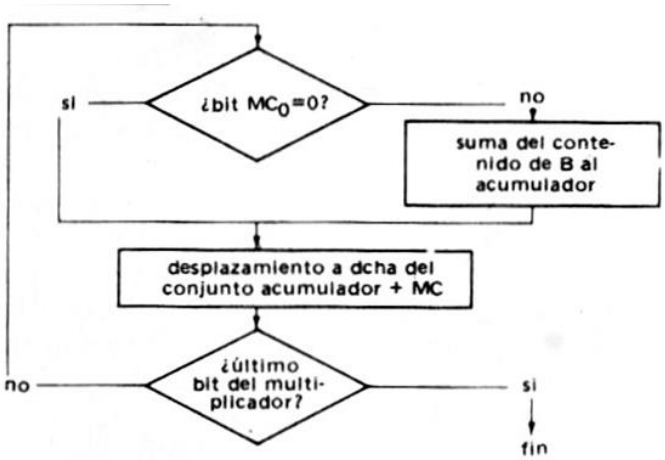


Fig. 12. Organigrama de la multiplicación

Al final de la operación, el resultado de la multiplicación ocupa en doble longitud el conjunto acumulador + MC.

TECNICAS DE MULTIPLICACION RAPIDA

Dos vías principales permiten aumentar, por medios lógicos, la velocidad de la multiplicación.

(1) Basándose en la técnica tradicional de suma-desplazamiento y disminuyendo el número de operaciones parciales sucesivas.
(2) Realizando un operador celular para una multiplicación casi paralela.

1. Mejora de la técnica de Suma-Desplazamiento: pueden aportarse modificaciones a la técnica anterior. Entre las diversas soluciones propuestas, escogemos una que se refiere a la condensación de operaciones por realizar cuando se tiene una sucesión de ceros o una sucesión de unos. Por consiguiente se presupone que pueda verificarse el número de 0 o de 1 sucesivos.

- Caso de una sucesión de 0. La mejora consiste en dotar al acumulador de circuitos para el desplazamiento de varias posiciones en una sola operación. En estas condiciones, la sucesión de 0 será tratada por un solo desplazamiento.

- Caso de una sucesión de 1. Se emplea el método llamado de "suma y sustracción". En el esquema tradicional una serie de 1 contiguos da lugar a una serie de adiciones, cada una seguida de un desplazamiento.

El método consiste en sustituir esta serie de adiciones por una adición y una sustracción. Por ejemplo, se la operación; multiplicando x 01111100; esta operación implica 5 adiciones, pero el multiplicador puede también escribirse: 10000000 – 00000100; de donde se deducen las reglas de la operación para el caso de presentarse una serie de 1 son las siguientes:

(1) Al primer 1 de la serie, se resta el multiplicando del contenido del acumulador.

(2) Al primer 0 después de la serie, se suma el multiplicando con el contenido del acumulador, tras haber ordenado un desplazamiento de tantas posiciones como 1 hay en la serie.

El esquema se puede mejorar aún más, por ejemplo, si se tiene un cero entre dos series de 1, ello conduce normalmente a una suma para tomar en cuenta al 0 y a una sustracción para el siguiente 1. De hecho, es posible agrupar estas dos operaciones en una sola sustracción al nivel del 0.

2. Multiplicación celular en paralelo: En el método de Suma-Desplazamiento, eventualmente mejorado, la multiplicación consiste en hacer un determinado número de operaciones sucesivas, cada una de las cuales debe concluirse antes de pasar a la siguiente. Pero es factible un circuito combinacional con dos entradas (el multiplicando y el multiplicador) que de el resultado a la salida en una sola operación. La red celular de la figura siguiente realiza la multiplicación:

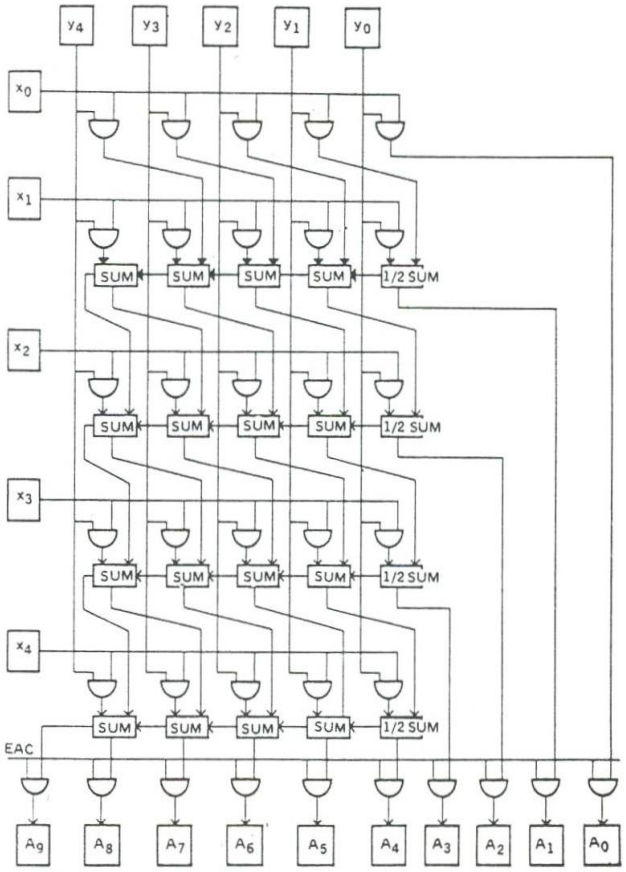
					Y_4	Y_3	Y_2	Y_1	Y_0
				x	X_4	X_3	X_2	X_1	X_0
A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0

Esta operación es ejecutada en paralelo. La señal EAC muestrea los niveles a la entrada del registro A.

Si se exceptúan la fila y la comuna 0, la célula está constituida por una puerta AND y una etapa de sumador. Si fijamos nuestra atención en la célula correspondiente a la fila i y a la columna j, la puerta AND está cerrada si el bit X_i del multiplicador vale 0: el sumador se limita a transmitir a la fila inferior el resultado parcial precedentemente obtenido, desplazándolo una posición.

En caso contrario, la puerta se abre: el sumador adiciona el nuevo producto parcial Y_j al resultado parcial obtenido en la fila superior, después practica el desplazamiento de una posición transmitiéndolo a la fila inferior.

Esta sucesión, mucho más onerosa que la anterior, puesto que se multiplica por n el número de etapas de sumador, ofrece la ventaja de ser muy rápida: aproximadamente el doble del tiempo de una suma de n dígitos, ya que un arrastre atraviesa como máximo *2n+1* etapas de sumador.



Multiplicación celular en paralelo

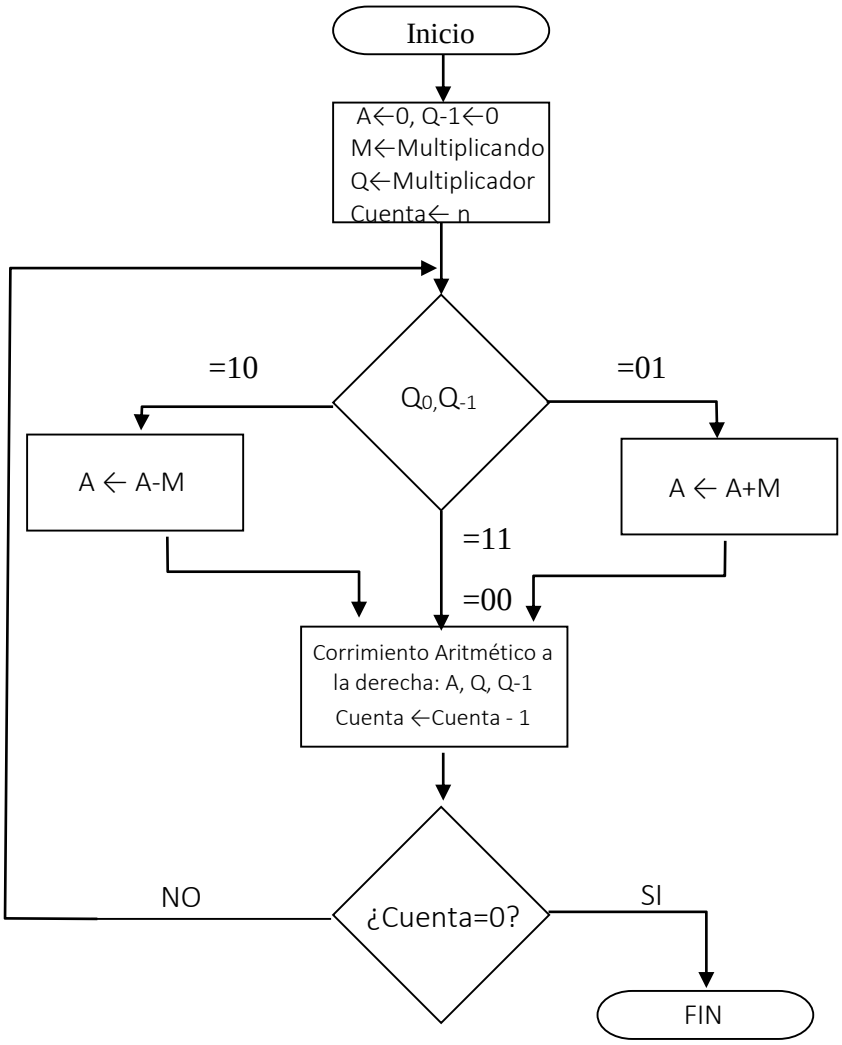
Multiplicación en Complemento a 2 (STALLING)

En el caso de complemento a dos, cuando tenemos el multiplicador o multiplicando o ambos con signo negativo debemos operar de diferente manera. Existen varias soluciones para este dilema. Una de ellas es convertir el multiplicador y el multiplicando en números positivos, realizar la multiplicación, y después tomar el complemento a 2 del resultado si y solo si el signo de los dos números originales difiere. Los implantadores han preferido utilizar técnicas que no requieren este paso final de transformación. Uno de los más comunes es el algoritmo de Booth.

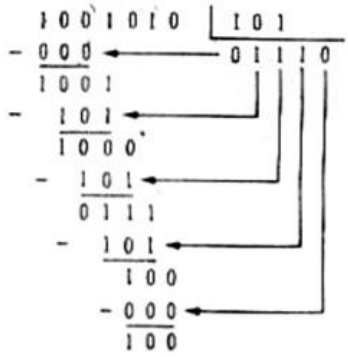
El algoritmo de Booth se puede describir de la siguiente manera: El multiplicador y el multiplicando se colocan el los registros Q y M, respectivamente. También existe un registro de 1 bit colocado de manera lógica a la derecha del bit menos significativo (Q0) del registro Q y designado Q-1.

Los resultados de la multiplicación aparecerán en los registros A y Q. Los registros A y Q_{-1} se inicializarán en 0. Como antes la lógica de control analiza los bits del multiplicador, uno a la vez. Ahora, cada bit es examinado, el bit a su derecha también es examinado. Si los dos bits son el mismo (1-1 o 0-0), entonces todos los bits de los registros A, Q y Q_{-1} se corren un bit a la derecha. Si los dos bit difieren, entonces el multiplicando se suma a o se resta del registro A, de acuerdo a si los dos bit son 0-1 o 1-0.

Siguiendo a la suma o la resta, el corrimiento a la derecha ocurre. En cualquier caso el corrimiento a la derecha es tal que el bit más significativo de A, denominado An_{-1}, no solo se recorre a An_{-2}, sino que también permanece en An_{-1}. Esto se requiere para preservar el signo del número en A y Q. Esto se conoce como un *corrimiento aritmético*, puesto que preserva el bit del signo.



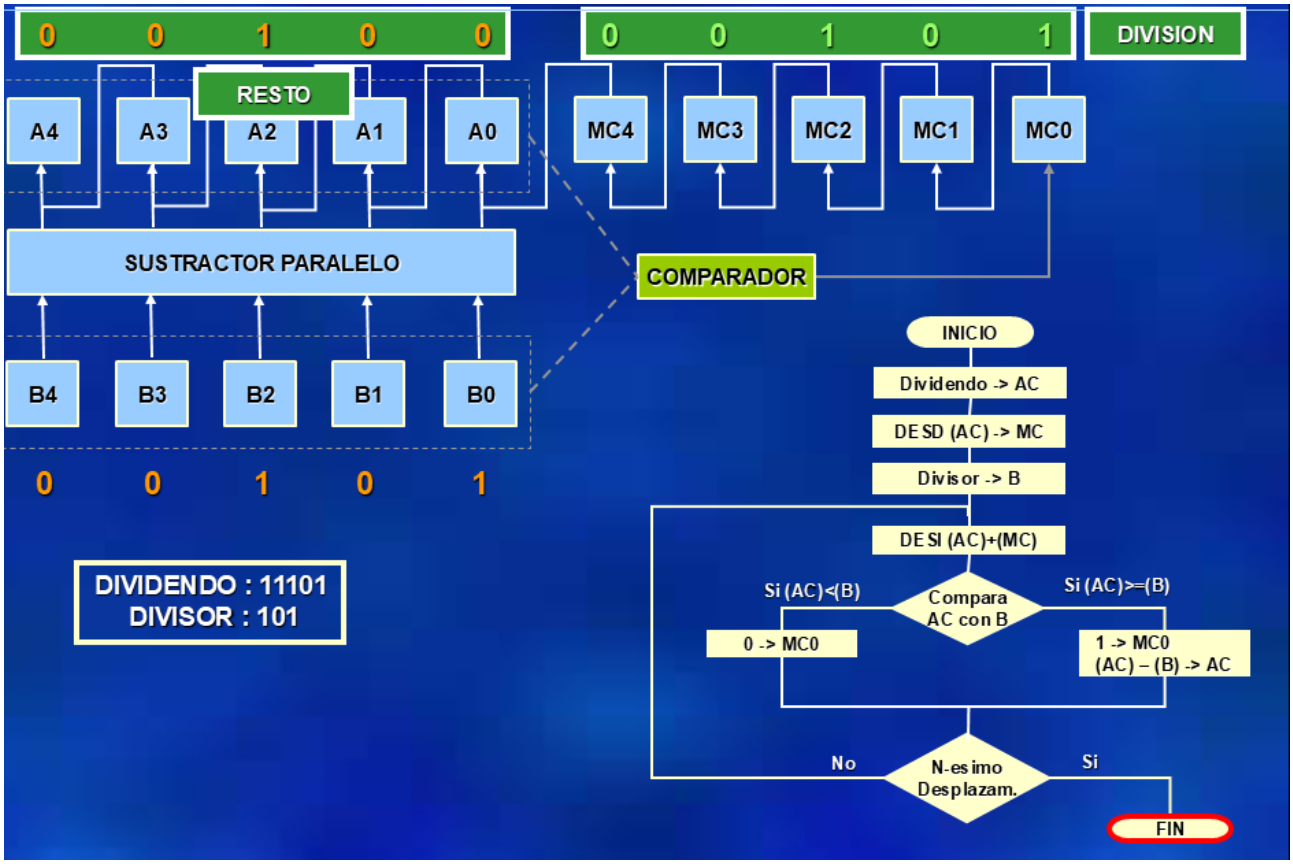
Divisor por Sustracción y Desplazamiento



En la multiplicación, los productos parciales eran iguales al multiplicando o nulos, según que el correspondiente bit del multiplicador valiera 1 ó 0. Se iban totalizando después de desplazar una posición hacia la izquierda a cada producto parcial en relación con el anterior.

En la división, se resta del dividendo el divisor o cero, según que el correspondiente bit del cociente valga 1 o 0. Se recomienza después de practicar un desplazamiento a la derecha del divisor en relación con el dividendo. Por tanto, multiplicación y división son operaciones muy semejantes. Basta cambiar sumas en sustracciones e invertir el sentido de los desplazamientos. Sin embargo, en la división aparece una dificultad complementaria: es necesario añadir a cada etapa una operación de comparación entre los bits de mayor peso de dividendo y divisor a fin de determinar el bit correspondiente del cociente.

Para efectuar una división tomaremos de nuevo la unidad aritmética utilizada con la multiplicación (casi siempre es común la unidad para ambas operaciones); pero transformaremos el sumador en sustractor, invertiremos el sentido del desplazamiento y añadiremos el dispositivo de comparación.



Generalmente la operación se inicializa de la manera siguiente:

1) Carga del dividendo en el Acumulador
2) Desplazamiento a derecha del contenido del conjunto acumulador + MC, con objeto de poner a cero el acumulador y cargar el dividendo en el MC.
3) Carga del divisor en B.

La operación comenzará por un desplazamiento a izquierda del conjunto acumulador + MC, lo que tiene por consecuencia poner el bit de mayor peso del dividendo en A_0 y liberar el biestable MC_0. Se compara

el contenido del acumulador con el contenido del registro B. Si (AC)<(B), se pone MC_0 a cero sin tocar el acumulador, si (AC)>=(B), se resta del acumulador el divisor, se pone MC_0 a 1, etc. Al final de la operación, el registro multiplicador cociente alberga al cociente y el resto aparece en el acumulador.

Por lo mismo que hemos obtenido en la multiplicación un resultado en doble longitud, puede presuponerse aquí un dividendo de doble longitud. Basta cargar al principio los bits de mayor peso del dividendo en el acumulador, y los bits inferiores al MC. En estas condiciones, se corre el riesgo de cometer un error si el contenido del acumulador es mayor que el divisor: el cociente excedería entonces a la capacidad del MC y se perderían sus bits de mayor peso.

Se evita este error comparando inicialmente los contenidos de acumulador y registro B. A este nivel es cuando pueden detectarse las divisiones por cero. También puede convenirse en no dividir más que por números enmarcados a izquierda, de manera que siempre el bit de mayor peso del divisor sea 1. (Este será el caso en la división de números flotantes).

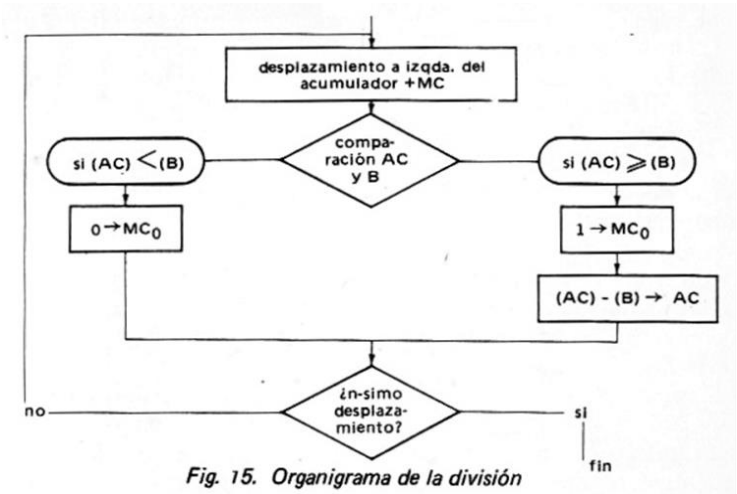


Fig. 15. Organigrama de la división

División con Restauración

En el punto anterior hemos dado por supuesto que contábamos con un dispositivo para comparar directamente los valores numéricos de los contenidos del acumulador y del registro B. En realidad, casi nunca se cuenta con él por razones de precio, por lo que puede hacerse la comparación intentando restar el divisor del dividendo y comprobando el signo del resultado. Este es, además, el procedimiento que empleamos cuando nos vemos obligados (desgraciadamente) a operar una división sin máquina ni regla de cálculo. Pero, al contrario de lo que sucede en el papel, la unidad aritmética pierde el antiguo valor del dividendo cuando el intento no surte efecto. En tal caso, es necesario restaurarlo, añadiendo el divisor al dividendo antes de hacer un nuevo intento, de ahí el algoritmo de la división con restauración.

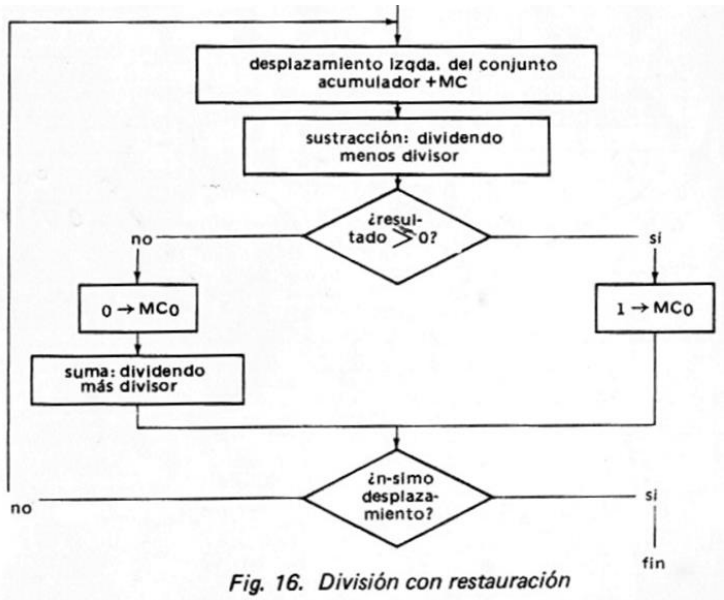
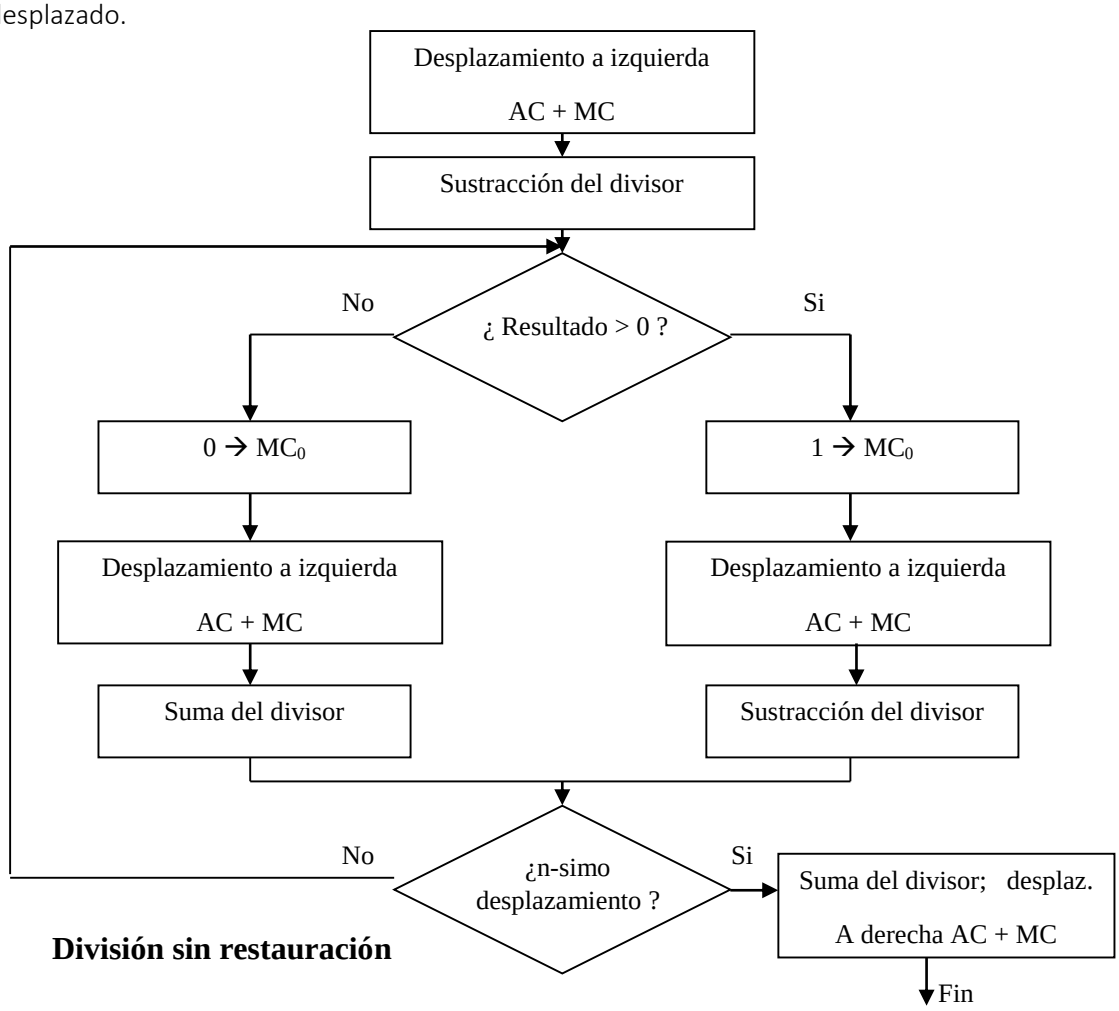


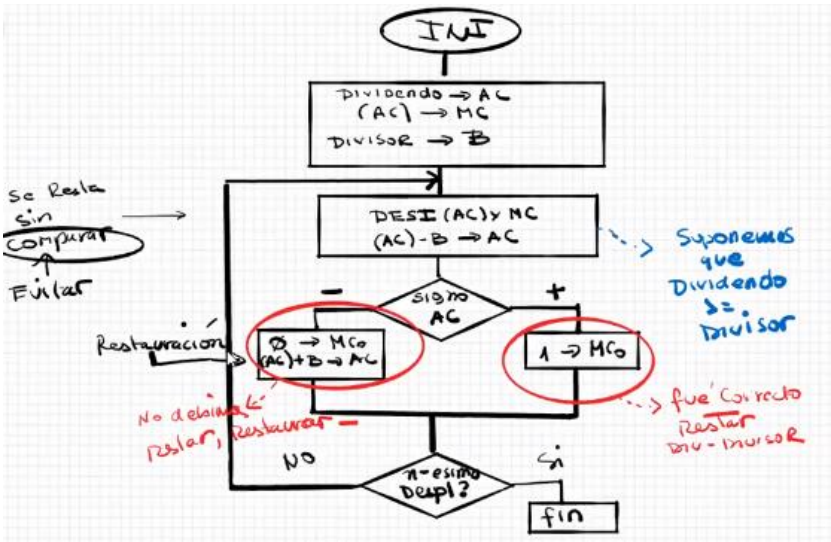
Fig. 16. División con restauración

División sin Restauración

Corresponde a un algoritmo que permite ahorrar la fase de restauración después de una sustracción con resultado negativo, lo que supone una ganancia de tiempo. Sea α el contenido del acumulador y β el divisor antes de la fase de restauración. Esta consiste en hacer α + β, sistemáticamente seguida de una sustracción del divisor desplazado (que vale β/2) y el resultado es (α + β) - β/2 = α + β/2. Es posible combinar la restauración y la sustracción que la sigue en una sola operación: la suma del divisor desplazado. En otros términos, un intento dando resto positivo ira seguido, al paso siguiente, de la sustracción del divisor desplazado, mientras que un intento resultando en resto negativo deberá seguirse de la suma del divisor desplazado.



División sin restauración



Consideraciones de ALU:

En general, el diseño de la ALU juega un papel muy importante en los ciclos de cálculo de computador, incidiendo en gran porcentaje sobre la performance de la arquitectura del computador.

El tipo y cantidad de operadores cableados microprogramados será preponderante en la relación costo/performance.

Observamos que ciertas funciones se han optado por cablear con el tiempo y en función de los resultados obtenidos, tal el caso de las operaciones en coma flotante, que hasta los 90 estaban microprogramadas y era una opción la incorporación de un Co Procesador adicional para estas operaciones que mejoraba notablemente la performance global.

En lo particular, la ALU de ABACUS consiste en un dispositivo que cablea operaciones básicas tales como suma/sustracción y algunas operaciones lógicas básicas. A su vez desarrolla esquemas que pueden implementar operaciones más complejas como el manejo de números con coma, y la multiplicación y división algebraica binaria.

A su vez prescinde de un registro fuente de entrada (Bi obligando al registro palabra de memoria a sostener todo el tiempo los valores lógicos del Bus M en pos del cálculo solicitado por la instrucción en curso.

MEMORIA

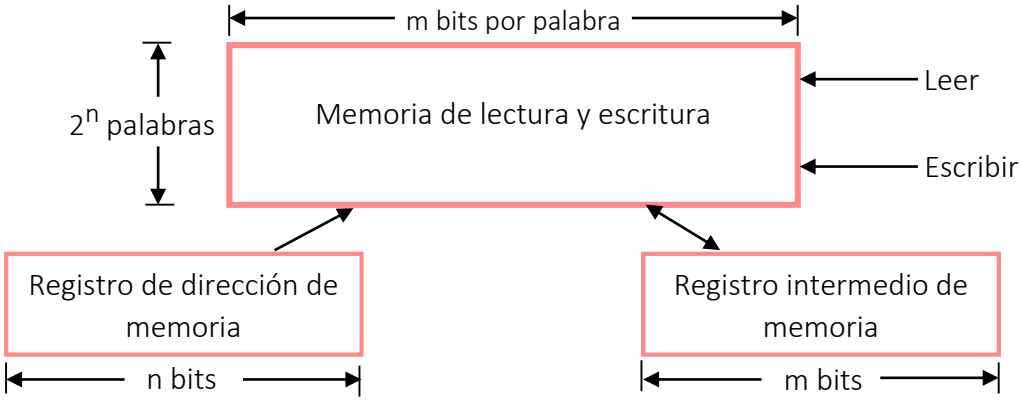
Es un dispositivo electrónico capaz de almacenar de manera temporal una información binaria de tal forma que todo órgano que requiera servirse de él, pueda acceder a la información solicitada en cualquier momento. Forma parte integral de la máquina y está directamente controlado por ella. De esta manera no consideramos memoria a las cintas y discos magnéticos, pero sí una vez que están montados en el dispositivo de lectura. Debe pertenecer a la máquina.

La memoria está formada por un conjunto de células o registros, capaces de almacenar información (en general una palabra). La información guardada podrá ser un dato o una instrucción, sin embargo, las instrucciones están en una zona y los datos en otra. Los registros de almacenamiento en una unidad de memoria se llaman registros de memoria.

Cada célula de memoria tiene un número que la identifica (su dirección), que es habilitada por la unidad de control, ya que conoce a cada célula por su número. Esta habilitación nos permite leer o escribir información contenida en una dirección determinada.

La memoria debe ser de lectura no destructiva, por lo tanto una celda de memoria tendrá que poder ser leída todas las veces que sea necesaria sin que se altere su contenido. Solo deberá modificarse el contenido cuando escribimos una nueva palabra. Las memorias que pueden ser leídas y escritas se llaman memoria de lectura y escritura, en cambio, las memorias que tienen programas o datos permanentes almacenados se denominan memorias de solo lectura.

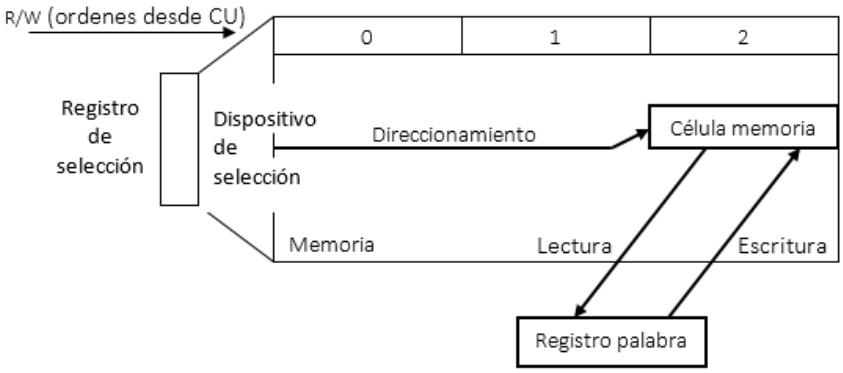
Para realizar una lectura o escritura, la unidad de control proporciona la dirección de la célula implicada a un registro asociado a la memoria central, llamado registro de dirección de memoria, o también conocido como registro de selección de memoria. Este registro está formado por n elementos binarios (generalmente flip-flops), en los que 2 n es el número de palabras que pueden ser almacenados en la memoria.



Memoria de lectura y escritura

El dispositivo de selección de memoria analiza el contenido del registro de dirección y sensibiliza la célula implicada, bien para una lectura o para una escritura. Si se tratase de una lectura, primero se limpia registro de palabra, y luego la información almacenada en la célula sensibilizada será transferida a este registro. El registro de palabra está asociado a la memoria central, y también se conoce como registro de intercambio que tiene tantos elementos de almacenamiento binario como bits en cada palabra de memoria (m bits). En el caso de una escritura, previamente habrá sido preciso cargar este mismo registro con la información que se quiera transferir a la célula en cuestión. Se dice a la memoria que se va a escribir, por medio de un 1 en la línea de escribir. La memoria, entonces almacenará el contenido del registro intermedio de memoria, en la posición especificada en el registro de dirección de memoria.

La operación de lectura no destruye la información almacenada en la célula (lectura no destructiva). La operación de escritura destruye la información almacenada, sustituyéndola por una nueva información.



Lectura y escritura en la memoria

Punto de memoria

Es el soporte físico que permite almacenar la mínima unidad de información binaria. Deberá contar con la lógica necesaria para poder leer o escribir información binaria en él, tal el caso que los antiguos Núcleos de Ferrita o el Biestable Electrónico.

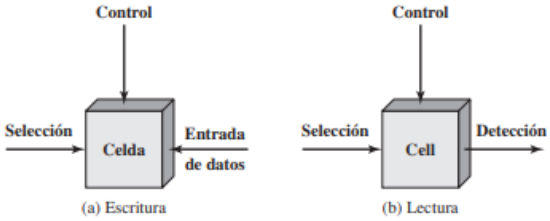


Figura 5.1. Funcionamiento de una celda de memoria.

Prácticamente, la totalidad de las memorias emplean el almacenamiento binario, es decir, que la información más elemental registrada es el bit; a cuyo soporte físico llamamos punto de memoria (almacena un bit). El punto de memoria debe poseer circuitos electrónicos de manera que pueda ser perfectamente definido e individualizado y constará, además del dispositivo de almacenamiento, de los de lectura y escritura.

Los elementos de almacenamiento pueden ser:

- Electrónico – F. F. } Memoria Central
- Eléctrico – condensador (DRAM) } Memoria Central
- Magnético – Punto magnetizable } Memoria Auxiliar
- Deformaciones de superficie – Discos ópticos } Memoria Auxiliar

Palabra de Memoria

Es el conjunto de bits que la maquina trata como unidad básica de información. Mínima unidad de información que la máquina puede almacenar, transferir y transformar (definición de Vicente). Cada palabra se almacena en un registro de memoria. Una palabra es una entidad de n bits que se mueven hacia adentro y afuera del almacenamiento como una unidad. Una palabra de memoria puede representar un operando, una instrucción, o un grupo de caracteres alfanuméricos o cualquier información codificada binariamente.

En general el procesador mueve a través de los buses un número de bits igual al tamaño de la palabra que puede procesar. En general la longitud de la palabra es múltiplo de 8 bits (1 byte). La longitud de la palabra decide una serie de aspectos destacables de la arquitectura de la máquina. Así el procesador central (CPU)

interpretara a los datos e instrucciones almacenadas en memoria como grupos de bits del tamaño de una palabra.

Los registros de la CPU que almacenarán informaciones en el transcurso del procesamiento, los de la ALU y los asociados a la MC, deben tener un tamaño mínimo a la longitud de la palabra.

Características de las memorias

Volatilidad: se dice que la información almacenada es volátil si corre el riesgo de verse alterada cuando se produce un corte en la alimentación y no volátil en caso contrario. **Las memorias electrónicas** son de este tipo. Las **memorias magnéticas** son no volátiles. *La memoria central es volátil*, mientras que las memorias auxiliares (también llamadas externas o secundarias) son no volátiles.

Lectura y Escritura: Son las operaciones básicas de las memorias. Se realiza escritura cuando se registran informaciones en la memoria y lectura cuando se extraen informaciones previamente registradas. La lectura puede ser:

- **Destructiva:** La información leída se borra de la memoria, era el caso de los núcleos magnéticos y es el de las memorias RAM Dinámicas; en estas se debe re-grabar la información sistemáticamente luego de cada lectura.
- **No destructiva**: caso de las memorias de biestables (RAM Estáticas), y memorias de semiconductores. La escritura puede exigir o no un borrado previo. En las memorias de semiconductores no obligan forzosamente a un borrado previo a la escritura, mientras que en las memorias de núcleos la escritura exige un borrado previo.

Modo de Acceso:

- **Aleatorio, Directo o Selectivo:** en las memorias con punto de memoria electrónico (biestables por ejemplo) se tiene acceso directamente a cualquier información, cuya dirección sea previamente conocida, en una cantidad de tiempo que es **independiente a la posición** de la información en la estructura de la memoria.
- **Secuencial:** En las informaciones guardadas en cinta magnéticas, es necesario desenrollar completamente la cinta, leyendo registro por registro hasta encontrar el buscado. En este tipo de memorias, el tiempo de acceso al registro buscado **dependerá fundamentalmente de la posición** de este en la cinta.

  En los discos magnéticos el acceso a la información es selectivo (directo) en cuanto a la elección de la pista, y secuencial en el interior de la pista.

Tiempo de Acceso: Tiempo que transcurre entre el instante en que se lanza una operación de lectura en memoria y el instante en que se dispone, en el RPM, la primera información buscada. En las memorias actuales, es de aproximadamente de 10ns (nano segundos) para las RAM estáticas y de 60ns para las RAM dinámicas.

Tiempo de Escritura: Es el tiempo que transcurre entre el momento en que se proveen, en los registros asociados correspondientes de la memoria, la información a guardar y su dirección, y el instante en que la información queda realmente almacenada.

En el tiempo de acceso para la lectura es mayor en la RAM dinámica fundamentalmente porque en estas memorias la lectura es destructiva y por lo tanto, la lectura debe incluir necesariamente un ciclo de reescritura.

Velocidad de Transferencia: Es la velocidad a la que se pueden transferir datos a, o desde, una unidad de memoria. Para memorias de *acceso aleatorio* coincide con el inverso del tiempo de ciclo. *Para otras* memorias se utiliza la siguiente relación: $T_N = T_A + N/R$

Donde:

T_N = Tiempo medio de escritura o de lectura de N bits
T_A = Tiempo de acceso medio
N = Número de bits
R = Velocidad de transferencia, en bits por segundo (bps)

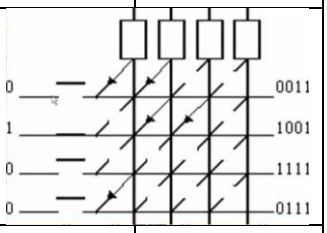
Caudal: así se llama al número máximo de informaciones leídas o escritas por unidad de tiempo. Generalmente se expresa en Kilo-informaciones o Mega-informaciones transferidas por segundo. Se relaciona con la velocidad.

Tiempo de Ciclo de Memoria: Incluye el tiempo de lectura y de reescritura.

Capacidad: número de palabras o de bits que la memoria puede almacenar, teniendo en cuenta el direccionamiento binario, las capacidades de memoria se expresan habitualmente como potencias de 2: 1024, 4096 palabras, etc.

Para simplificar se ha convenido utilizar de capacidad, la Kb (kilo) palabra que representa 2^{10} = 1024 unidades de información, 1Mb (mega) = 2^{20}, 1Gb (giga) = 2^{30}, 1Tb (tera) = 2^{40}.

Tipos de Memoria

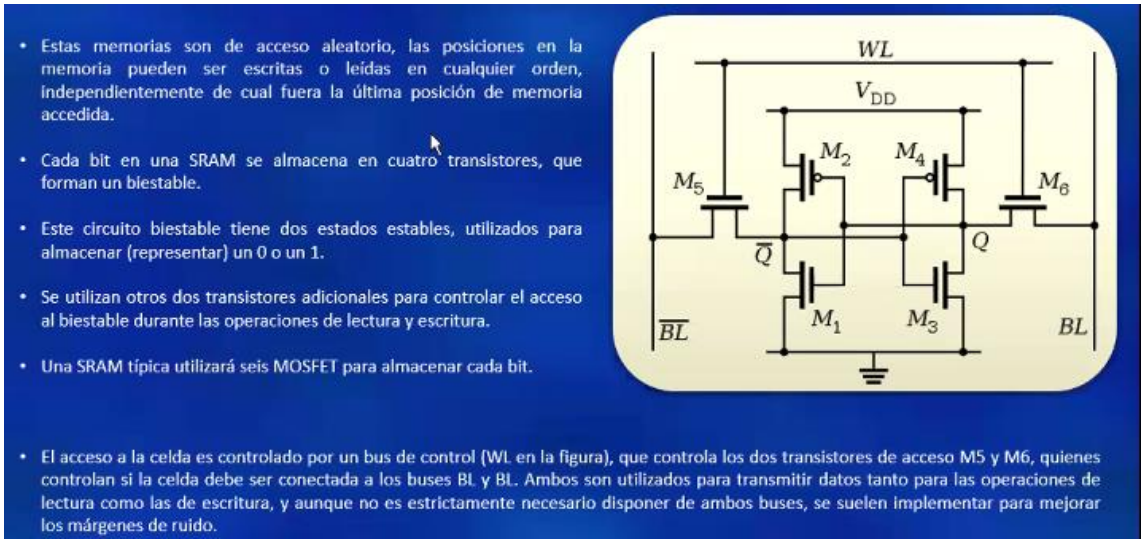
Tipo de memoria	Clase	Borrado	Mecanismo de Escritura	Volatilidad
M. de Acceso aleatorio (RAM)	Memoria de Lectura/Escritura	Eléctricamente por Bytes	Eléctricamente	Volátil
Es como una plancha de ravioles. Borrado por bytes, porque no podes borrar media palabra, significa que el borrado se hace por palabra (por fila completa). Ante la ausencia de electricidad de borra todo.				
Memoria de solo Lectura (ROM)	Memoria de solo Lectura	No disponible	Mediante Máscaras	No Volátil
Read only memory. No permite su borrado, una vez que se grabó ese chip mantiene esa información hasta que el soporte físico se degrada. *Escritura mediante máscaras*, quema el material, es decir indica donde hay un 1 o un 0 (un ejemplo es un CDROM), *por cada iodo: si no está quemado representa un 0, si está quemado representa un 1.* Esta escritura lo que hace es dañar el iodo, *se graban en líneas*. *Aplicaciones:* donde lo queremos que influencia ambiental. 				
ROM Programable (PROM)	Memoria de solo Lectura	No disponible	Eléctricamente	No Volátil
Program read only memory. Las PROM se graban en grabadoras PROM y tienen la capacidad de ser grabadas por una única vez. No hacía falta una línea de producción (como en el caso de las ROM)				
ROM Borrable (EPROM)	Memoria de sobre todo Lectura	Luz Ultravioleta, chip completo	Eléctricamente	No Volátil
Ereaseble program read only memory. Más que nada son para leer, pocas veces son para escribir ya que no son prácticas; en otras palabras es que *permite reescribirlas*. Permite borrado mediante luces ultra violetas y se escribe eléctricamente. Es como una chance de poder reescribir.				
Memoria FLASH	Memoria de sobre todo Lectura	Eléctricamente por bloques	Eléctricamente	No Volátil
Sobre todo lectura *porque se desempeñan muy bien en lectura, pero en escritura produce un desgaste en el punto de memoria.* Se borran y graban en bloques. Ejemplos: pendraive, microSD. Simulan una estructura de disco rígido hoy en día.				
PROM Borrable eléctricamente (EEPROM)	Memoria de sobre todo Lectura	Eléctricamente por Bytes	Eléctricamente	No Volátil
Electrical eraseble program read only. Se borra por byte, se borra por palabras (se asemeja a un disco rígido)				

Clasificación Tecnológica: existen dos tipos de memoria que se comunican directamente con la CPU: la memoria de acceso aleatorio (RAM) y la memoria de solo lectura (ROM). Las memorias de solo lectura son dispositivos de *lógica programable*. La información que está contenido en estos dispositivos, debe especificarse de alguna manera y luego se introduce incorporándose al hardware. A este proceso se lo denomina *programación de la unidad*.

Tipos de memoria (apunte):

- Memorias Bipolares: En estas, el elemento de almacenamiento (flip-flop) y los demás elementos están fabricados con transistores de unión PN convencionales. Estas son memorias muy rápidas pero de mayor precio. (Registros y Caché)

- Memorias MOS Estáticas: se fabrican utilizando transistores de efecto de campo MOS para hacer los circuitos de los FF. Estas memorias tienen menor velocidad que las bipolares pero tiene menor precio, consumen menos potencia y permiten mayor densidad de empaque (más elementos por unidad de volumen). Las celdas MOS ocupan entre un medio y un cuarto del área de una celda bipolar y por lo tanto ofrecen una ventaja de costo considerable.



Tiene pequeñísimos transistores y muchos (como mínimo tiene 6 semi-conductores (este regula la base entre el catado y el ánodo, es como una llave)) No son magnéticas, la información la guardan bajo los conceptos teóricos de un biestable. M5 Y M6 controlan el acceso al biestable durante la esc y el read.

El acceso a esta celda está controlado por dos bloques (BL Y (BL)') se los implementa para disminuir el marguen de ruido, pueden o no estar. MORRIS MANO, QUIROGA Y WEB

Ventaja de este punto de memoria=> velocidad. Son memorias dedicas a registros del hardware y consumen mucha corriente eléctrica para ser veloces.

- Memorias MOS Dinámicas: también se fabrican utilizando transistores MOS pero en lugar de usar FF como elemento de almacenamiento del punto de memoria, se usa la carga depositada en un condensador fabricado en circuitos integrados y la presencia o ausencia de esta carga determina el estado de la celda.

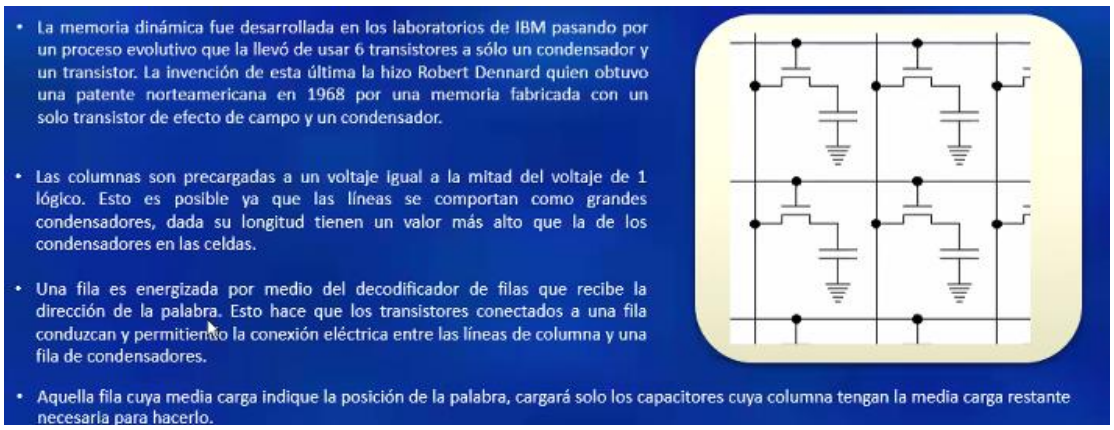
En los condensadores la carga desaparece con el tiempo, por perdidas a través del dieléctrico, entonces es necesario "refrescar" periódicamente esta carga y por esto se llaman memorias dinámicas.

Este tipo de memorias son más lentas, su costo es menor, consumen menos potencia y permiten mayor densidad de empaque y por lo tanto estos circuitos de memorias se utilizan ampliamente.

La ventaja principal de la memoria MOS dinámica radica en la sencillez de la celda individual. Una celda de memoria RAM dinámica es más simple que una estática y por lo tanto más pequeña. Otra ventaja es que no es necesario aplicar potencia a las celdas, cuando no van a ser leídas o escritas. Esto hace mayores las

densidades de empaque de cada circuito. Por lo tanto las RAM dinámicas son más densas (celdas más pequeñas = más celdas por unidad de superficie) y más baratas que las correspondientes RAMs estáticas. La desventaja radica en la necesidad de refrescar estas celdas cada pocos milisegundos dado que la carga se fuga continuamente de los condensadores. Generalmente se requieren circuitos externos para controlar la escritura de refresco, como resultado se requieren ciclos adicionales, aunque éstos ocupan sólo un pequeño porcentaje de los tiempos totales de operación. En memorias grandes, el coste fijo de la circuitería de refresco se ve más que compensado por el menor coste de las celdas RAM dinámicas. Así pues, las RAM dinámicas tienden a ser las preferidas para memorias grandes.

Las memorias con flip-flops MOS o bipolares se llaman memorias estáticas. Un último detalle es que las RAMs estáticas son generalmente más rápidas que las dinámicas.



Son ideales. Tienen capacitores que funcionan como punto de memoria. El transistor por punto de memoria gobierna la lectura y grabación del punto de memoria (del capacitor) cuando se empieza a descargar el capacitor, el transistor hace algo que no llegue a anotar :(

Desventaja=> dependen de los capacitores, es más lento que los transistores, hay cierta latencia para cargarse en los capacitores. Dinámica es más lenta que las estáticas.

Estáticas para nivel 0 y 1 requieren alimentación constante y muy veloz. Dinámicas para nivel 2 Tienen refrescos eléctricos (gracias a los transistores) cuando van perdiendo su capacidad (es más lenta igual u.u)

- Memorias CMOS: Los CMOS utilizan elementos de canal n y p en el mismo sustrato. Como resultado, esto involucra un proceso más complejo. Los CMOS tienen mejoras en su salida en potencia y velocidad sobre los dispositivos MOS canal n o p, pero su costo es mayor.
- Memorias de silicón sobre zafiro (SOS): Los SOS son similares a los CMOS. Estos dispositivos están formados sobre una superficie aisladora de zafiro. Esto reduce la capacitancia, aumentando la velocidad. Sin embargo, los SOS tienen un costo mayor.
- Memorias con lógica de inyección integrada (IIL): Los circuitos IIL eliminan las resistencias de carga y las fuentes de corrientes de los circuitos TTL. Esto reduce el consumo de potencia sobre las memorias bipolares, dando una densidad de empaque mayor que en éstas. Como resultado, la IIL mezcla la alta velocidad de las memorias bipolares con la densidad de empaque de los MOS. Estas tienen un costo intermedio.

Las memorias ROM se pueden clasificar en:

→ ROM: Graban una vez por el fabricante.
→ PROM: Se graban una vez por el usuario.
→ EPROM: Se graban varias veces por el usuario, el borrado se realiza con luz UV.
→ EEPROM: Se graban varias veces por el usuario, el borrado se realiza eléctricamente posición a posición.
→ FLASH: Se graban varias veces por el usuario, el borrado se realiza eléctricamente de una sola vez.

Jerarquía de Memorias

En un ordenador hay una jerarquía de memorias atendiendo al tiempo de acceso y a la capacidad que normalmente son factores contrapuestos por razones económicas y en muchos casos tamicen físicas. Comenzando desde el procesador al exterior, es decir en orden creciente de tiempo de acceso y capacidad.

Las restricciones de diseño de la memoria de un computador se pueden resumir en tres cuestiones:

1. ¿Cuánta capacidad?
2. ¿Cómo de rápida?
3. ¿De qué coste?

La cuestión del tamaño es un tema siempre abierto. Si se consigue hasta una cierta capacidad, probablemente se desarrollarán aplicaciones que la utilicen.

La cuestión de la rapidez es, en cierto sentido, fácil de responder. Para conseguir las prestaciones óptimas, la memoria debe seguir al procesador. Es decir, cuando el procesador ejecuta instrucciones, no es deseable que tenga que detenerse a la espera de instrucciones o de operandos.

La última de las cuestiones anteriores también debe tenerse en cuenta. *En la práctica, el coste de la memoria debe ser razonable con relación a los otros componentes*. Como es de esperar, existe un compromiso entre las tres características claves de coste, capacidad y tiempo de acceso.

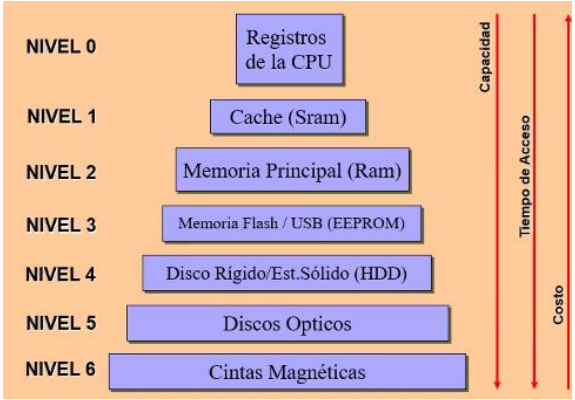
En todo el espectro de posibles tecnologías se cumplen las siguientes relaciones:

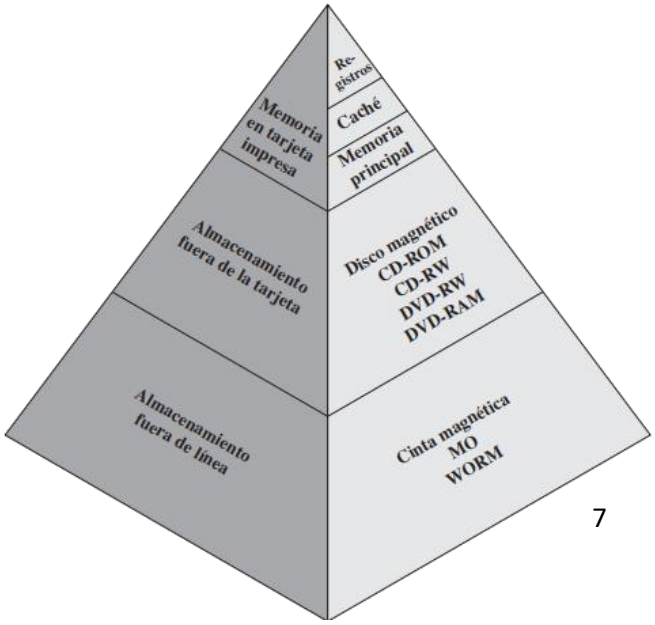
→ A menor tiempo de acceso, mayor coste por bit.
→ A mayor capacidad, menor coste por bit.
→ A mayor capacidad, mayor tiempo de acceso.

El dilema que enfrenta el diseñador está claro: desearía utilizar tecnologías de memoria que proporcionen gran capacidad, tanto porque esta es necesaria como porque el coste por bit es bajo. Sin embargo, para satisfacer las prestaciones requeridas, el diseñador necesita utilizar memorias costosas, de capacidad relativamente baja y de tiempos de acceso reducidos. La respuesta a este dilema es no contar con un solo componente de memoria, sino emplear una jerarquía de memoria.

1. Disminuye el coste por bit.
2. Aumenta la capacidad.
3. Aumenta el tiempo de acceso.
4. Disminuye la frecuencia de accesos a la memoria por parte del procesador.

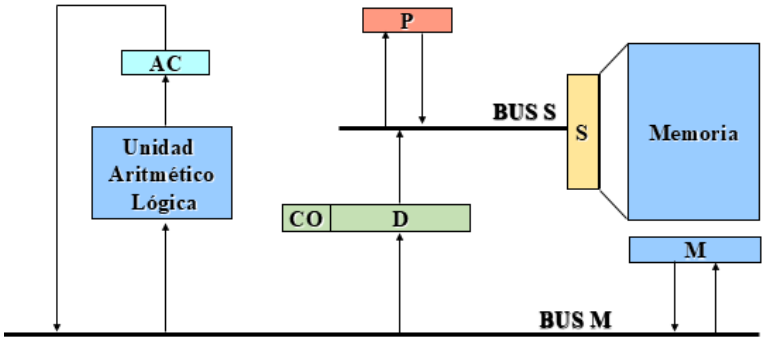
Así pues, las memorias más pequeñas, costosas y más rápidas, se complementan con otras más grandes, más económicas y más lentas.





Nivel 0 → Hardware. registros de la arquitectura. Son caros, más cache, más registros tiene, más caro son. Tardan nanosegundos en leer. **Registros del procesador**. Estos registros interaccionen continuamente con la CPU (porque forma parte de ella). Los registros tienen un tiempo de acceso muy pequeño y una capacidad mínima, normalmente igual a la palabra del procesador (de 1 a 8 bytes).

Los registros intermedios constituyen un paso intermedio entre el procesador y la memoria, tienen un tiempo de acceso muy breve y muy poca capacidad. Interactúan con los buses, permiten mover y operar con la información. Intermediario entre procesador y la memoria.

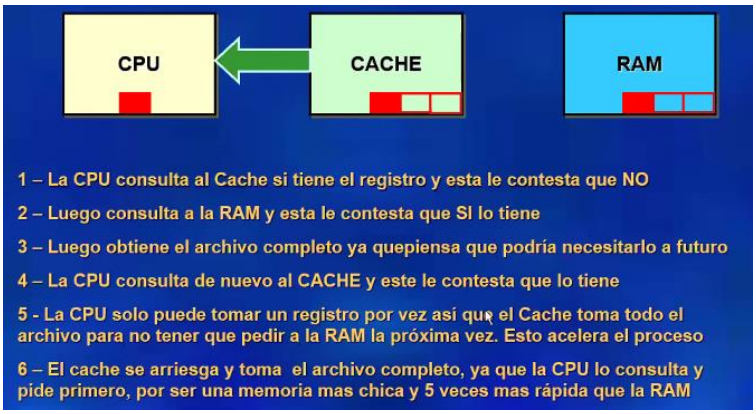


Nivel 1 → Memoria cache SRAM

Memorias de pequeña capacidad, son más veloces que la RAM, pero más caras. Tiene una altísima velocidad, y menor capacidad de memoria, que permite amortiguar las asimetrías de velocidad y tiempos de accesos. Libera la RAM de estar leyendo continuamente, sirven como amortiguador. Las memorias caché o ante-memoria, es un buffer inteligente, corresponden a una jerarquía de memoria situada entre la CPU y la memoria principal. Su función es mejorar el rendimiento del Procesador Central sin aumentar el costo del sistema. Está preparada para servir de apoyo al procesador y es capaz de trabajar a valocidades muy elevadas, es por ello que apoya a la CPU para mejorar su rendimiento.

Para ello actúa como un sistema de almacenamiento instrucciones y de datos capaz de comunicarse con el procesador a gran velocidad. Normalmente la memoria caché se almacenan datos e instrucciones que el procesador va a necesitar para completar diferentes tareas a corto plazo. Con ellos evitan dos cosas muy importantes que podrían reducir el rendimiento.

Principio básico de funcionamiento. Se ahorran ciclos de memoria. Quiere liberar a la RAM, entonces la CP consulta a la Cache porque es más veloz que la RAM y esta le dice que no, entonces le pregunta a la RAM y esta le pasa los datos a la cache. Luego la CPU pregunta a la cache, y *está ya va a tener la información*; entonces le devuelve el dato que quiere el CPU (*se libera el cuello de botella de la RAM*) Es como una cámara de vigilancia.



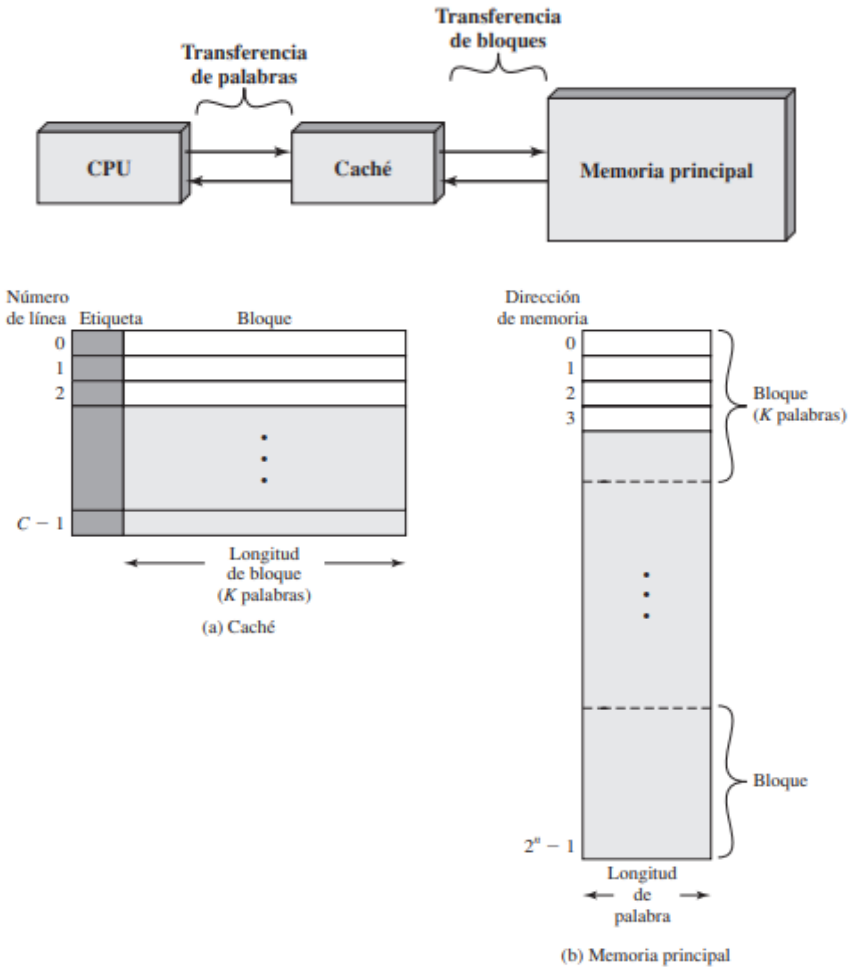


Figura 4.4. Estructura de memoria caché/principal.

El caché recibe de la memoria principal DRAM, a una cierta velocidad, una copia de las instrucciones que serán ejecutadas próximamente por la CPU, y los datos que serán procesador por ellas. Esta información, es mayormente la que ya fue accedida recientemente, pues la estadística indica que es la que con mayor probabilidad será accedida una y otra vez. De este modo la CPU leerá del caché instrucciones y datos, al triple o más de velocidad, que de la memoria principal sin acceder a ella, dado que se trata de una memoria SRAM (de acceso mas rápido que la DRAM). Esto último implica que no existan pulsos de espera (“wait states”).

Luego pasarán más lentamente a la memoria principal desde el caché los resultados que la CPU escribió velozmente en este. Un subsistema circuital denominado “controlador de caché” asegura que lo anterior se cumplirá, en más del 90% de los accesos (aunque esta cifra depende del software ejecutado), o sea que la CPU sólo accederá a la memoria principal en un 10% de todos los accesos requeridos. El porcentaje anterior se logra, almacenando con anticipación en el caché instrucciones y datos que la CPU solicitará próximamente. Por lo tanto, el manejo de la memoria caché está a cargo de un hardware de control, no requiriéndose software alguno.

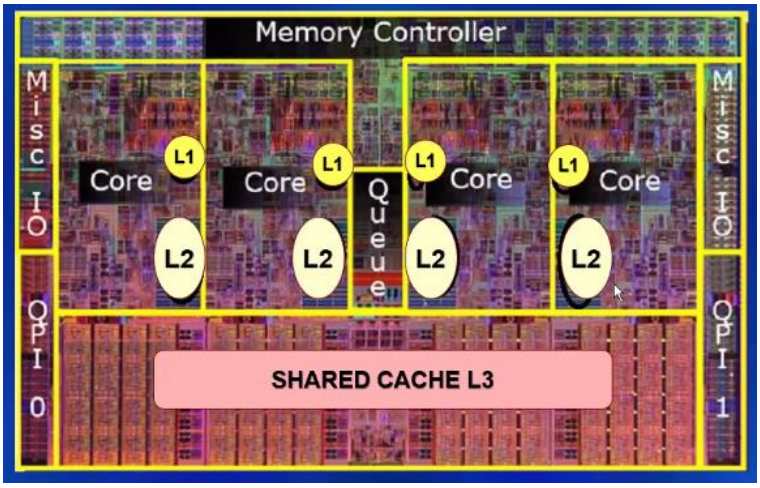
Cache tiene técnicas de cómo ir limpiando (como un índice de recurrencia, entonces los temas menos consultados, son "pisados") Tiene una lógica tipo LIFO, FIFO, etc.

Se clasifican en L1, L2 y L3.

→ L1 Es el nivel más básico, rápido y caro, y el de menos capacidad, está más cerca del procesador. Se caracteriza por cachear cálculos e información, libera al núcleo el tener que hacer el cálculo o acceder recurrentemente la memoria.

→ L2 Presenta un buen equilibrio en cercanía, capacidad y velocidad, pero el tiempo de acceso es más lento.

→ L3 El Shared L3, tiene una memoria por cada núcleo, guarda cálculos recurrentes; entonces se evita hacer cálculos (o un acceso de memoria) hechos anteriormente. Guarda información de memoria compartida de los núcleos. Comparte información de los núcleos de los procesadores, tienen una mayor capacidad pero son más lentos y están más alejados de los núcleos. Permiten integrar el chache a nivel de NUCLEOS. Si el núcleo 1 quiere determinada información que el núcleo 3 ya lo hizo, L3 le permite acceder a esa data.



Primero busca en L1, luego en L2 y luego en L3, si ninguna la tiene, se fija en la RAM. Esa cadena de consulta genera un retardo, llamaremos a eso latencia. Es el tiempo que le lleva al sistema llegar a la información que quiera; esta latencia aumenta a medida que pasamos de L1 a RAM. En caso de que ninguna tenga la información buscada, se tiene que realizar un ciclo de trabajo completo, con todo lo que ello supone el nivel de rendimiento.

Para mejorar el tiempo de acceso a memoria, los procesadores operan con dos niveles de caché, el primer nivel (Level 1) está constituido por el caché interno que el microprocesador lleva incorporado junto con su controlador de caché. A este caché se accede tan rápidamente como a los registros de la CPU.

Un segundo nivel (Level 2) lo constituye el caché externo al procesador (256 kb y 30 ns), conformado por los chips SRAM opcionales, que pueden insertarse en zócalos de la "mother". Cada vez que la CPU ordena leer la MC, el caché interno provee la información si está presente (hit) en él. De no estarlo (miss) en 30 ns la proveerá el caché externo, pasando también una copia al caché interno. Si la información no está en el caché interno ni el externo, se deberá recurrir a la MC a través del bus, lo cual implica el costo de tiempo adicional que insume un estado de espera.

Asimismo, en los dos niveles de caché se escribe dicha información, que estaba ausente en los mismos. Se deduce, que en general existirán copias de una misma información en los niveles de caché y MC. De lo anterior también resulta, que el caché de segundo nivel permite una menor utilización directa de la MC por parte de la CPU, descongestionando el bus que une a ambos.

Esto permite que la MC sea accedida por otros dispositivos, mientras la CPU accede a los niveles de caché para leer y escribir información. Son de acceso aleatorio, muy rápidas y de baja capacidad (1kb a 512kb). Se fabrican con transistores bipolares o HCMOS, son estáticas y su tiempo de acceso oscila entre 5 y 10ns, son más costosas.

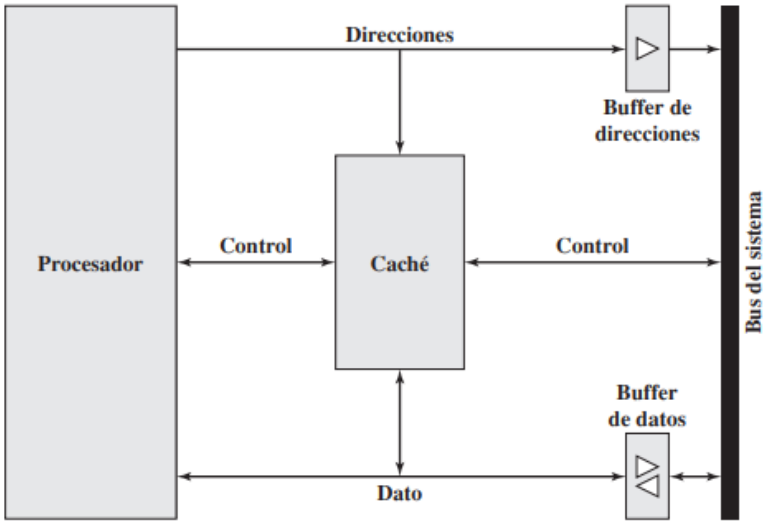


Figura 4.6. Organización típica de caché.

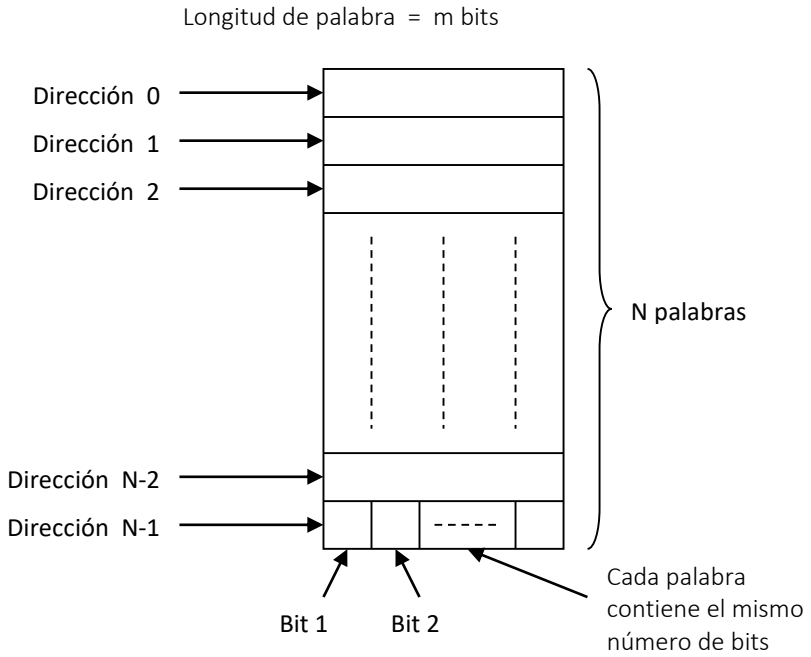
Nivel 2 → Memoria Principal RAM

Esta altamente vinculada con la arquitectura e interactúa de manera directa con la unidad central de proceso.

En este nivel residen los programas y los datos. La CPU lee y escribe datos en él aunque con menos frecuecnia que en los niveles inferiores. Tiene un tiempo de acceso relativamente rápido y de gran capacidad. Las extensiones de MC son memorias de la misma naturaleza que la MC, ampliando su capacidad de forma modular. El tiempo es similar, o un poco mayor, al de la MC y su capacidad puede ser algunas veces mayor.

Es posible leer o escribir en cualquier dirección de la memoria "a la vez", esto es si el retardo para encontrar una posición determinada es exactamente igual que para encontrar cualquier otra, la memoria se denomina memoria de acceso aleatorio (RAM).

Las memorias RAM son volátiles, es decir que si se interrumpe la energía se pierden los datos y sólo pueden utilizarse como almacenamiento temporal. Existen dos tipos de RAM: las estáticas y dinámicas. Tanto las RAMs estáticas como las dinámicas son volátiles. La RAM dinámica tiene un micro-capacitor como punto de memoria.



La figura muestra una memoria principal organizada en palabra de longitud fija. Como indica la figura una memoria está dividida en N palabras, donde N es generalmente una potencia de 2, y a cada palabra se asigna una dirección, o posición en la memoria. Cada palabra tiene el mismo número de bits, llamados **longitud de la palabra**.

Las direcciones o números de dirección en la memoria, van consecutivamente, partiendo de la dirección 0 llegando hasta la dirección más grande.

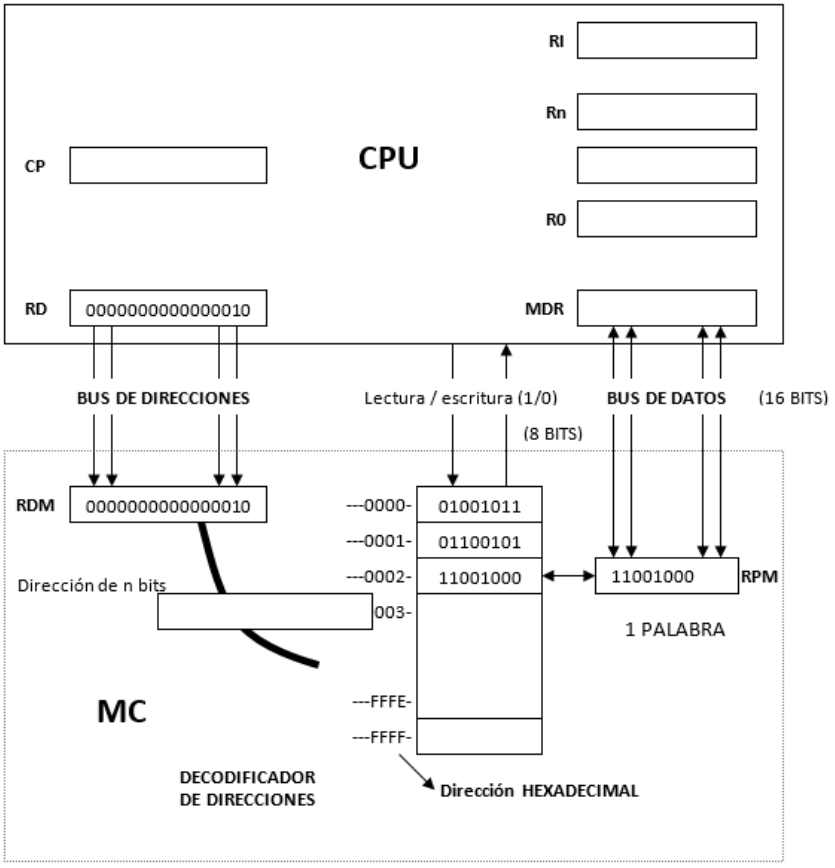
Existen diferencias entre el contenido de una dirección de memoria y la dirección en sí misma. La memoria es como un gran gabinete con muchos cajones, los cuales corresponden a las direcciones de memoria. En cada cajón hay una palabra y la dirección de cada palabra se escribe en la parte externa del cajón. Si escribimos o almacenamos una palabra en la dirección 17, es como si colocáramos la palabra en el cajón marcado con 17. Posteriormente, leer en la dirección 17, es como mirar este cajón para ver su contenido. Sólo cambiaremos el contenido de una dirección cuando almacenamos o escribimos una nueva palabra.

Cada dirección o posición contiene un número fijo de bits, número que ha sido llamado **longitud de palabra de memoria**. Así una memoria con 4096 posiciones, cada una con una dirección diferente y con cada posición que almacena 16 bits, se llama memoria de 4096 palabras de 16 bits, o memoria de 4k de 16 bits.

(Dado que las memorias vienen generalmente con un número de palabras igual a 2^n para cualquier n, si una memoria tiene 2^{14} = 16.384 palabras, se hará referencia a ella como una memoria de 16k y dado que siempre se entiende que la memoria tiene la capacidad de 2^n completa. Por lo tanto, una memoria de 2^{15} palabras de 16 bits cada una se denomina memoria de 32k de 16 bits).

La unidad de memoria se especifica por el número de palabras que contiene y el número de bits que hay en cada palabra. Las líneas de direcciones seleccionan una palabra en particular. Cada palabra almacenada en la memoria es identificada por su dirección.

En la memoria habrá tantas direcciones como posiciones pueda tener una memoria.



En la figura tenemos representada una memoria de palabras de 8 bits, de 65536 posiciones. Los registros de uso general R0 a Rn y RI son de 8 bits, mientras que RD, CP y RPM son de 16 bits. Lógicamente los buses tendrán la misma cantidad de conductores que la cantidad de biestables de los registros que unen.

La descripción de la memoria RAM comprende dos partes:

- El funcionamiento del punto de memoria.
- La organización general de la memoria, esencialmente ligada a la técnica de selección.

Los computadores usan invariablemente memorias de lectura y escritura de acceso aleatorio como memoria principal de alta velocidad y para su respaldo memorias de baja velocidad como reserva para mantener datos auxiliares.

Las memorias de acceso aleatorio más usadas son los circuitos integrados y las de núcleos magnéticos, así mismo, ambos están organizados de una manera similar.

Nivel 4 → Discos Magnéticos

Son memorias que residen en dispositivos externos al ordenador, en ellas se archivan programas y datos para su uso posterior. También se usan estas memorias para el apoyo de la MC en caso de que esta sea insuficiente (memoria virtual).

Estas memorias suelen tener una gran capacidad pero pueden tener un tiempo de acceso muy lento. Dentro de ellas también se pueden establecer varios niveles de jerarquía.

Nivel 5 → Discos Ópticos

Es básicamente un pedazo de plástico lleno de marcas o huecos que un rayo láser puede leer. Cuando más marcas podemos hacer (y leer) en su superficie, mayor será la capacidad de datos que podremos almacenar en él.

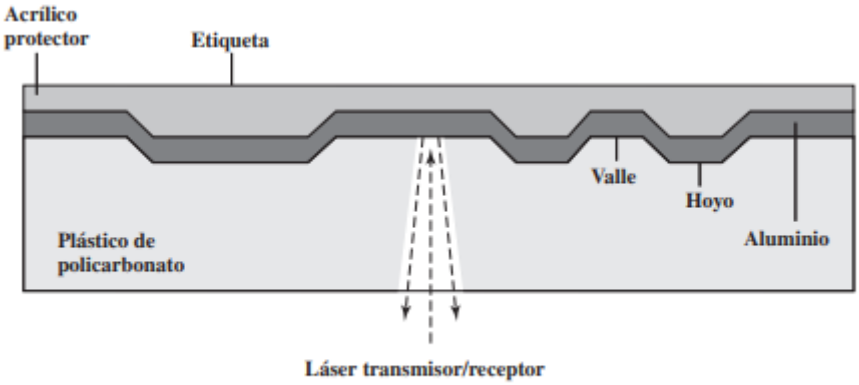


Figura 6.11. Funcionamiento de un CD.

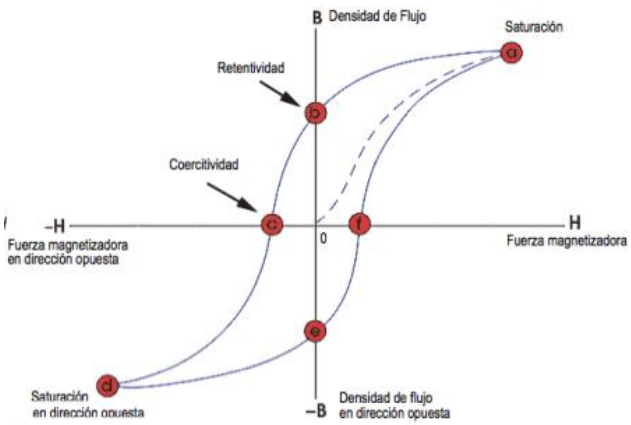
Nivel 6 → Cintas Magnéticas

Los sistemas de cinta usan las mismas técnicas de lectura y grabación que los discos. El medio es una cinta de poliéster flexible (parecido al usado en ropa) cubierta por un material magnetizable. Las cintas magnéticas fueron el primer tipo de memorias secundarias. Se usan todavía ampliamente como los miembros de la jerarquía de memoria de menor coste y de menor velocidad.

Almacenamiento en Núcleos Magnéticos

El dispositivo básico de almacenamiento en una memoria de núcleos magnéticos consiste en una pequeña pieza toroidal (en forma de anillo), llamada **núcleo magnético o anillo de ferrita**. El núcleo magnético, almacena información binaria, es generalmente una pieza sólida de material cerámico ferromagnético.

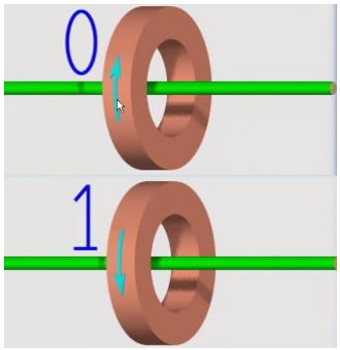
Ciclo de Histéresis: La corriente que transita al través del alambre produce una inducción eléctrica que produce un B. Este ciclo, plantea que si aumenta H, entonces el campo B comienza a magnetizarse; este tiene un límite, el cual está dado en el propio material (punto de saturación) A medida que bajamos H, B baja pero en una nueva línea, es decir que el material quedó imantado por la inducción, es el campo remanente que queda en el material ante la ausencia de la inducción producida.



Ahora invertimos, vamos a los números negativos. Obliga a B a descender, o sea, que para poder llevarlo a su posición original de no magnetismo, tenemos que ejercer una fuerza que lo obligue a ir a su estado normal (el punto de inducción remanente). Cuando B=0 Y H negativo me hace un campo coercitivo, un imán permanente debería tener mucha fuerza coercitiva. Seguimos aplican aún más -H, entonces generamos un -B y llega a un punto de saturación pero en los ejes negativos.

Si vamos a H=0 ahora volvemos a una inducción remanente con los polos invertidos a la que tuvimos al principio. Vamos a aumentando H => una corriente eléctrica coercitiva para poder volver.

FeO => el anillo puede magnetizarse. La forma de anillo las cargas recorren una forma propicia a que mantengan una carga magnética. **Si lo queremos magnetizar** => le pasamos un cable y este se magnetiza en el sentido del campo B inducido. **ESCRITURA.** En otras palabras: para actuar sobre la magnetización e un anillo de ferrita se le enhebra un hilo conductor y se le hace pasar por él una corriente eléctrica. Al aplicar una intensidad suficiente, el anillo se magnetiza en el sentido del campo magnético inducido y, al suprimir ducha corriente inductora, la magnetización permanece.



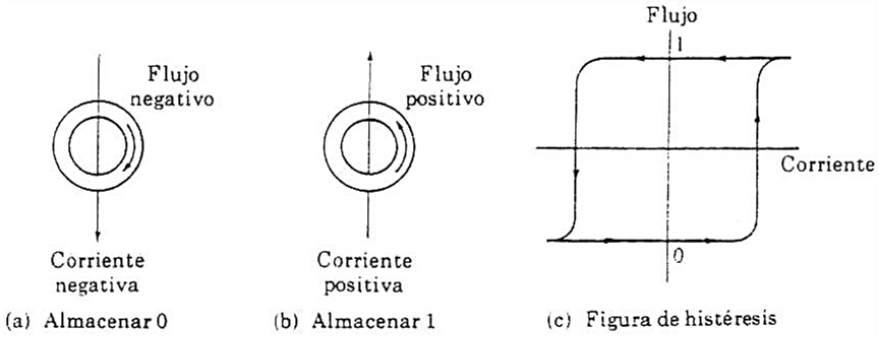
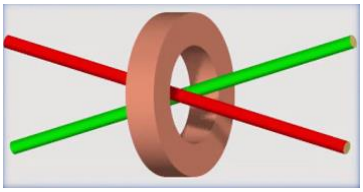


Figura 7-31 Almacenamiento de un bit en un núcleo magnético

El gráfico muestra un núcleo magnético, en un tamaño mucho mayor a su tamaño normal. El arrollamiento de entrada (hilo conductor), se muestra enhebrado a través del núcleo. Si pasa corriente a través del arrollamiento, se producirá un flujo magnético, con una dirección que depende de la dirección de la corriente en el arrollamiento. El núcleo es en forma de anillo, forrado de un material de alta permeabilidad, de tal forma que presenta una trayectoria de baja reluctancia al flujo magnético. Dependiendo de la dirección

de la corriente a través del arrollamiento de entrada, el núcleo se magnetiza en el sentido del movimiento de las agujas del reloj o en el sentido contrario.



Para averiguar el estado o sentido de magnetización del anillo se enhebra un segundo hilo (que no debe ser paralelo al hilo de escritura para así evitar inducciones indirectas entre ambos) que hará las veces de hilo de lectura. Para obtener el estado, se aplica una corriente eléctrica que fuerce la magnetización del anillo al estado 0. Ese hilo tiene una Intensidad que intente magnetizar el anillo => lo obliga a llevar a B=0 (lo desmagnetiza). Eso es para el proceso de LECTURA. Para leer el contenido, hay que borrarlo. Se arregla mandando otro pulso, pero son memorias muy lentas ya que ocupa dos ciclos. Destruir y escribir de nuevo, es decir: si dicho anillo estaba en el estado, al pasar al 0 su magnetización cambia bruscamente, lo cual induce una corriente apreciable en el hilo de lectura. Si el anillo estaba ya en 0, la magnetización apenas cambia y la corriente inducida es mucho menor. Midiendo esa corriente se puede saber cuál era el estado del anillo.

Un núcleo magnético emplea tres cantidades físicas: corriente, flujo magnético y voltaje. La señal que excita al núcleo es un pulso de **corriente** en un alambre que pasa a través del núcleo. El pulso de corriente que excita al núcleo, es generado por un circuito accionador (DR = Driver).

La información binaria almacenada se representa por la dirección del **flujo magnético** dentro del núcleo. La información binaria de salida se extrae de un alambre que encadena el núcleo, en la forma de un pulso de **voltaje**.

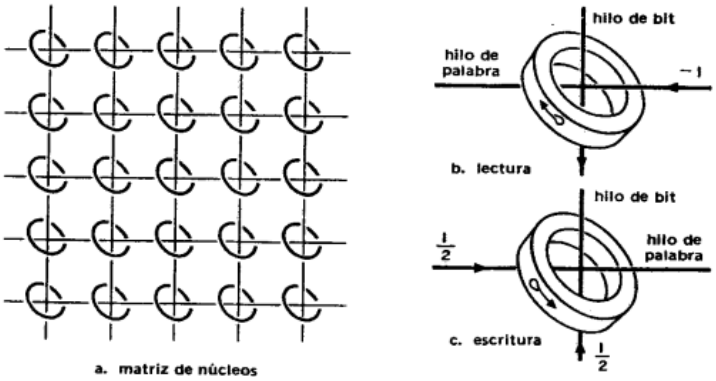
Las propiedades fundamentales de estas memorias son las siguientes:

a) Memoria estática con direccionamiento cableado, tipo RAM.
b) No volátil, pues, si se deja de alimentar, las polarizaciones de las ferritas se mantienen invariables.
c) De lectura destructiva.
d) La velocidad de los primeros prototipos era de 20 us y se ha llegado a modelos de 275 ns.
e) La capacidad de estas memorias varía de unos pocos K a unos pocos Megas. Se construían con módulos de 4 K. Valores típicos de anchos de palabra han sido 8, 16, 32 y 36 bits.

La conexión a los transductores se realiza mediante hilos de cobre barnizados, que pasan por el interior de las ferritas. Esta parte del proceso de fabricación consistía en "coser" las ferritas y se hacía a mano. Era fácil que, al coserlas, hubiera roturas de ferritas, lo que obligaba a difíciles reparaciones.

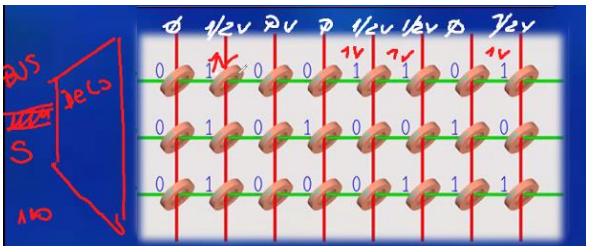
Implementación 2D

Va a haber un hilo de palabra x cada fila, no por cada anillo como antes (Gobierna una palabra completa), y un hilo por columna (gobierna el bit de la palabra seleccionada). El hilo que pasa por todos los aniñños de una misma posición, horizontalmente, se llama el hilo de palabra y el que pasa por los anillos homólogos, verticalmente, se llama hilo de bit. Para una memoria de 1KB se necesitan 1024 hilos de palanras y 8 hilos de bit. *Menos hilos => menos compleja la memoria*



Limitación => El plano

Ventaja => Simplicidad de construccion de la memoria. El plano nos permite hacerla materialmente de manera sencilla (anillos en la misma direccione hilos verticales y horizontales)

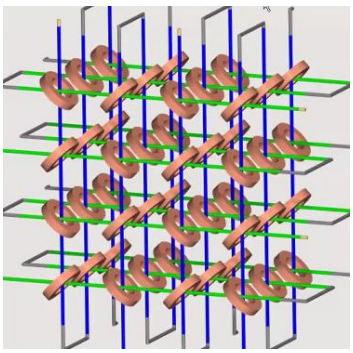


Corrientes coincidentes → implica en mandar desde el hilo de palabra el valor medio del valor eléctrico necesario para inducir en donde debe ir un 1. Así en donde hay un 1 se produce un 1V entonces se magnetizan y llegan al punto de saturación y se estabiliza la carga remanente que es la que dice que ahí hay un 1

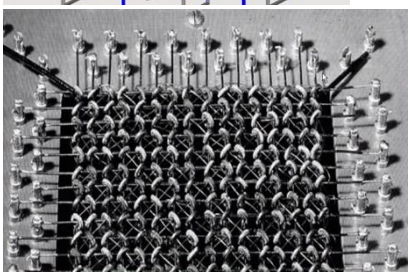
ESCRITURA → Primero ponemos la palabra (cuya dirección coincida con la que estoy buscando) completo en 0 (NO LA MEMORIA) entonces la sensibiliza con una corriente negativa para ponerlas a 0 SE lo conoce como BORRADO de la palabra; esto consume dos ciclos. Luego se le manda 1/2 V para poner a 1 (corrientes coincidentes). **Es decir**: Primero se pone la palabra en cero y luego se hace pasar otra corriente electrica positiva por el hilo de l apalabra y, simultaneamente, otras iguales por los hilos de bits cuyos anillos hay que poner a 1. La intensidad de esass corrientes está establecida de forma que una, por si sola, no es capaz de hacer que los anillos que atraviesa se pongan a 1; pero si dos juntas. Los anillos que reciben el efecto combinado de la corriente en el hilo de palabra con la del hilo de bit se pondrán a 1.

LECTURA → Requiere de dos ciclos por ser destructiva. Entonces para la corriente negativa para llevarlos a 0. Si estaba magnetizado, si produce una corriente inducida sobre el hilo rojito => esa pequeña carga se interpreta como si fuese un 1. El segundo paso es reescribir para recuperar los datos que borramos antes. **Es decir:** Se hace pasar una corriente negativa por el hilo de la palabra elegida, que lleve a todos los anillos al estado 0 (lectura destructiva). Entonces lo que tienen un 0 siguen igual, en cambio los que tenían un 1 pasan a 0 induciendo una corriente apreciable en su hilo de bit; pero como esta corriente es menor que la necesaria para forzar una conmutación no afecta al estado de los demás anillos atravesados por el mismo hilo de bit.

Implementación 3D



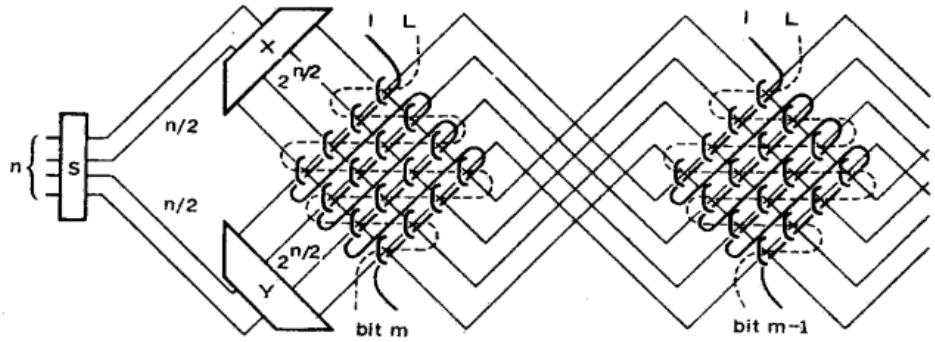
Con el fin de disminuir aún más la cantidad de hilos, la 3D plantea superponer planos colocándolos alternados, una al derecho y otro al revés. Finalmente se conectan en serie los hilos de selección homólogos uniéndolos en lados alternos, de modo que se forme una especie de zigzag con cada hilo. Un decodificador de direcciones X (Horizontal) e Y (Vertical) logran a través de corrientes coincidentes, seleccionar una palabra espacial.



La palabra se forma en el espacio => vamos a llamar plano a cada uno de los nivele => la palabra está constituida por la unión de los planos. La longitud de la palabra está relacionada con la cantidad de planos (8 bits => 8 planos). La capacidad de memoria está dada por la cantidad de palabras. Necesito un deco por fila y otro por columna.

Ventaja => mucho más pequeña y comprimida y muchas menos conexiones que la 2D

Desventajas =>



Esquema de organización 3D de una memoria

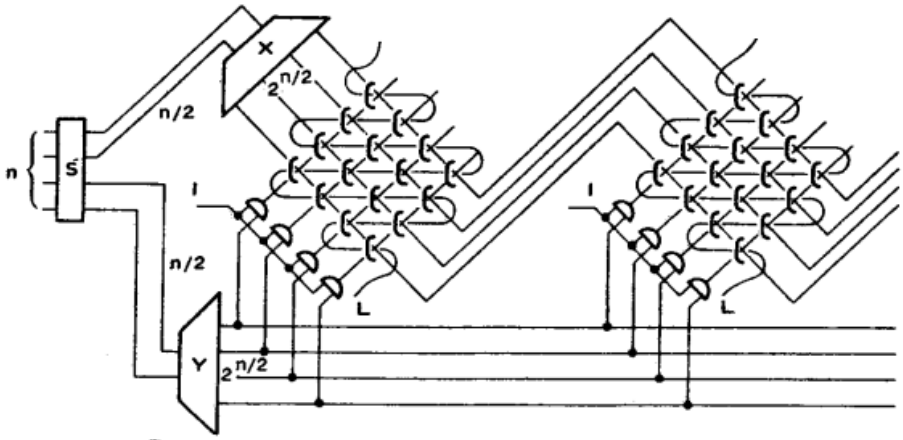


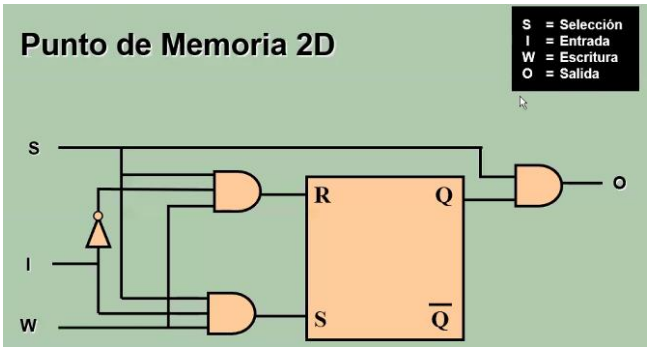
Fig. 11. Esquema de principio de una memoria $2^{1}/_{2}$ D

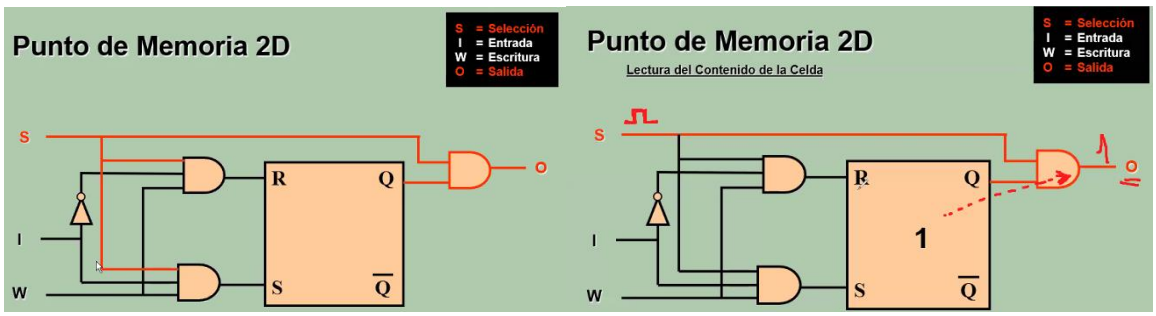
Organizaciones de Memoria basadas en semiconductores (biestables)

Organización 2D

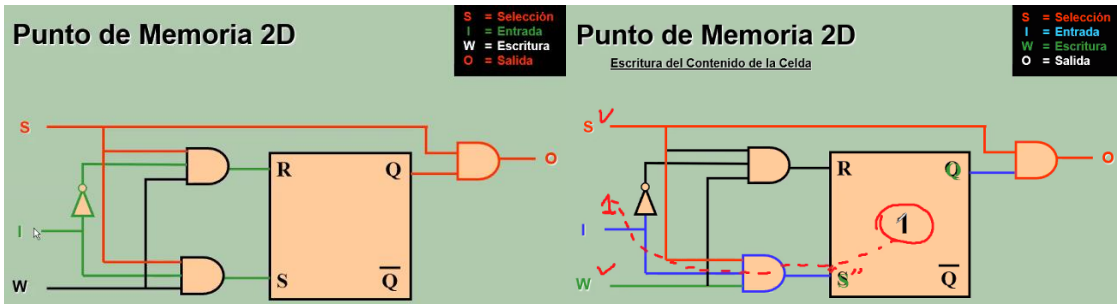
Es una memoria de tipo volátil cuyo punto de memoria está compuesto por un ff. Para que esta celda sea útil, deberá poder ser seleccionada mediante el registro de dirección de memoria y se debe tener un método para controlar cuando las celdas seleccionadas deban ser leídas o escritas. Es decir, tiene que presentar diferentes funciones para que pueda interactuar con la matriz del punto de memoria para poder leer o escribir.

Selección -> se selecciona desde el punto de vista lógico a un punto que contenga una palabra, cuatro son las variables S,I,W,O





S selecciona el punto de memoria para que sobre este se puede ejecutar una acción. El hecho de seleccionar produce de manera espontánea por el punto O el la lectura, la cual es transmitida y transportada a O.

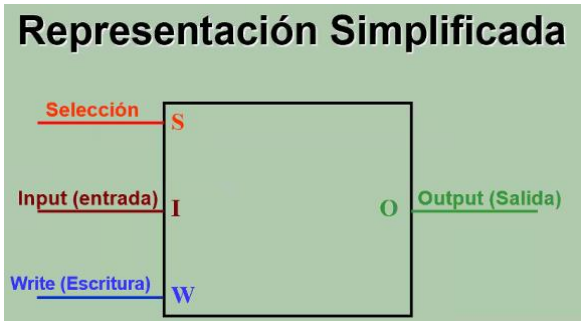


S+I | I => (0 reset, 1 set) es la Input que se quiere guardar en el ff

S+I+W W escribe sobre el punto de memoria la información presente en I

Ante la presencia de una señal impulsional, necesaria para que se activa la salida Q y pueda salir la información por O

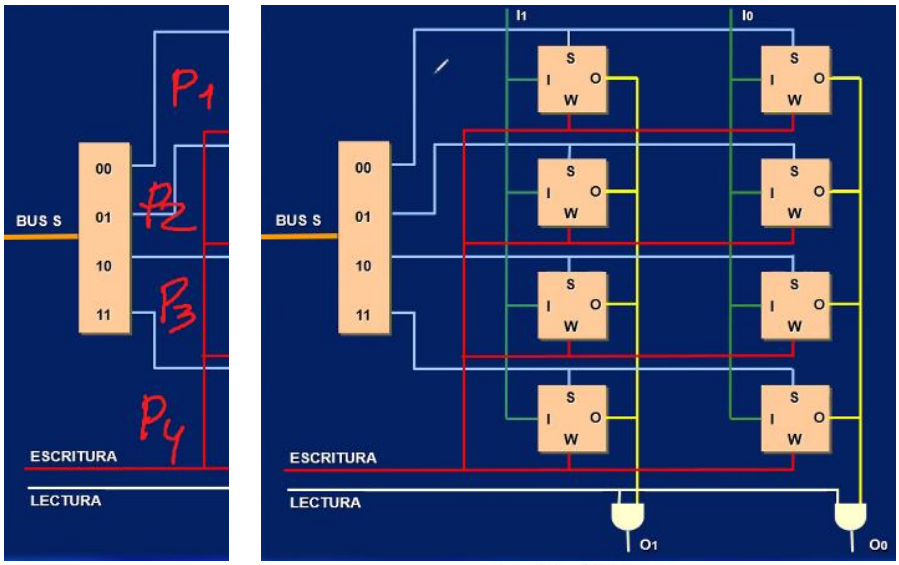
Para escribir un 1, S debe estar seleccionado y después ingresamos el valor lógico (set en caso de 1) se activa el SET cuando activamos W (se graba de forma lógica en el punto de memoria y el 1 sigue en la entrada) <=> S y W son verdaderos. Tener en cuenta: apenas activamos S se produce una lectura, para evitar que se escriba y lea al mismo tiempo, (escritura => lectura espontanea, por el hecho de S) se coloca una periferia combinacional para evitar la lectura espontanea



El registro de dirección de memoria selecciona la celda de memoria (flip-flop) para ser leída o escrita, por medio de un decodificador que selecciona tres flip-flop para cada una de las direcciones que puede tener el registro de dirección de memoria.

El esquema muestra que el decodificador tiene una entrada para cada flip-flop (bit) que va a ser decodificado. Si hay dos bits de entrada, se tendrán 4 líneas de salida, una para cada estado (valor) que pueda tomar el registro de entrada.

4 palabras de 2 bits c/u. 2 hilos de bus puede direccionar 4 palabras (2^2) La entrada u la salida son por el mismo lugar, el registro palabra de memoria es un rectángulo (M) que se comunica con la memoria. (↑ I0,I1,I2,I3 (W) ↓ O3,O2,O1,O0 (S))



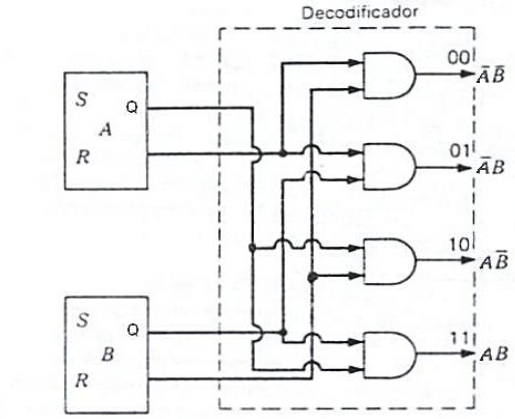
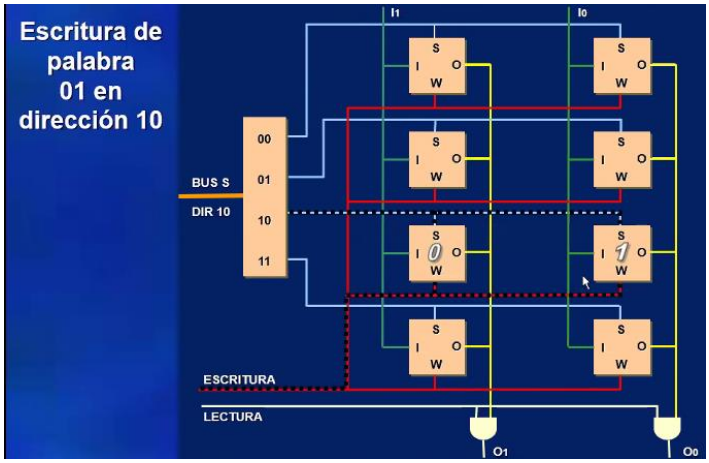
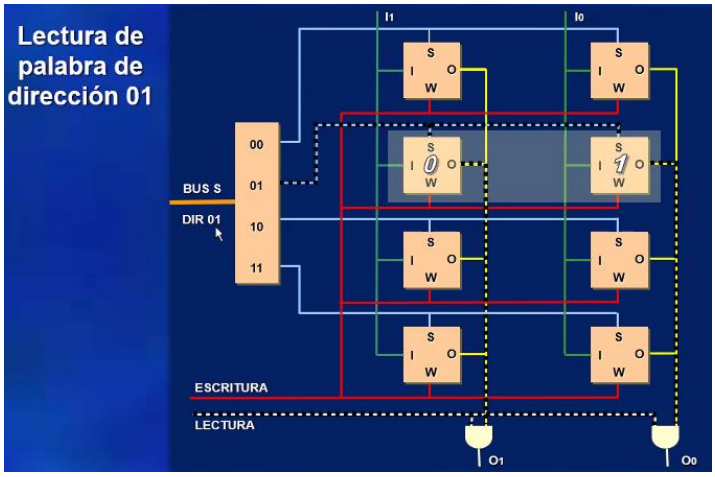


Fig. 7.5. (a) Decodificador de cuatro salidas.

La selección permite la escritura. Para la **escritura** la palabra que se va a escribir es cargada en el registro de palabra de memoria; colocando un 1 en la línea de habilitación de escritura, la información disponible en las líneas I_1, I_2 quedará almacenada en las celdas de la palabra. Para escribir una determinada configuración binaria en una palabra de memoria *(no es necesario que este puesta a cero*) se envía un 1 para seleccionar la fila correspondiente, la información por los hilos de bit (I_1, I_2) y un 1 por el hilo de escritura que alcanza a todos los biestables. Únicamente los biestables sensibilizados por el hilo de palabra almacenaran la información.



El decodificador, mediante las líneas correspondientes, sensibilizará a la palabra seleccionada y colocando un 1 en la línea de habilitación de **lectura**, el contenido de las 2 celdas de la palabra se transferirán, por las líneas O1, O2 al RPM, donde podrán ser leídas. La operación de lectura (*no es destructiva*) consiste en enviar un 1 por el hilo de selección (o palabra) que se desea leer y un 1 por el hilo de habilitación de lectura.



Por ejemplo, si la segunda fila de la memoria contiene 11 en las 2 celdas de memoria, y si el RDM contiene 01, la segunda línea de salida del decodificador (marcada como 01), será 1 y serán seleccionadas las compuertas de entrada de estas 2 celdas de memoria. Si la línea LEER está en 1, las salidas de estas 2 celdas de memoria de la segunda fila estarán en 11, a la entrada de las compuertas AND, de la parte inferior de la figura, que transmitirán el valor 11 como salida de la memoria.

Si la línea ESCRIBIR está en 1 y el RDM contiene nuevamente 01, la segunda fila de flip-flop tendrá seleccionadas las líneas de entrada. Los valores de entrada I_1, I_2 quedarán almacenados en los flip-flop de la segunda fila.

Es la memoria más sencilla, consiste en una matriz de biestables, cada uno de ellos alcanzados por tres hilos, un hilo para seleccionar la fila, un hilo para la entrada de información y un hilo para habilitar la escritura. Los biestables de una misma fila memorizan una palabra. Se dice que una memoria así concebida está organizada por palabras y que su selección es lineal o en dos dimensiones (2D).

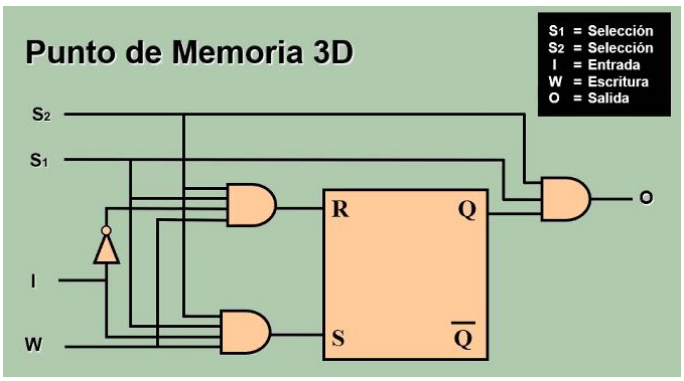
Esta es una memoria completa, con capacidad total de lectura y escritura. La memoria almacenará los datos por un período indefinido y operará tan rápido como lo permitan las compuertas y los flip-flop. En las memorias 2D la organización es muy sencilla pero el decodificador del sistema de selección resulta muy complejo a medida que aumenta el tamaño de la memoria. El gran número de componentes a utilizar en los circuitos decodificadores es la principal objeción a este tipo de memorias.

Ventajas: tiene un punto de memoria sencillo y su arquitectura también, es económica en cantidad de elementos

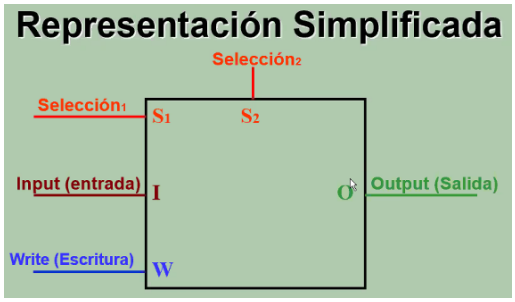
Desventaja: crece de forma exponencial el espacio y el costo si queremos ampliar la memoria.

Organización 3D

Plantea planos en distintas posiciones espaciales que cada plano representa un tamaño x de memoria. Implica un punto más complejo en lo que se refiere a las entradas, contempla una línea más de entrada S => las dos AND Tienen 4 entradas y la de salida AND tiene 3. Pasa de 8 entradas de la 2D a 11 entradas. Las 11 entradas la convierte en un punto de memoria del doble de carro.



Funciona igual que la 2D, con la diferencia que para la lectura se seleccionan S1 y S2 simultáneamente y la escritura S1,S2,I y W. S1 sería para las filas y S2 para las columnas

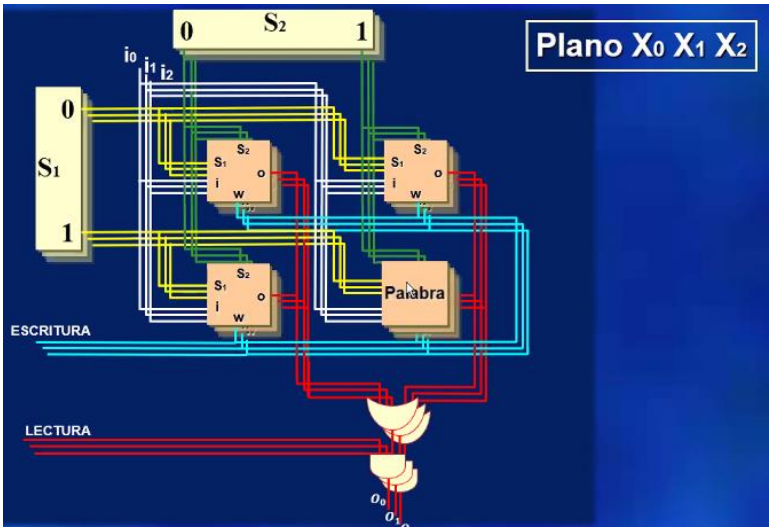


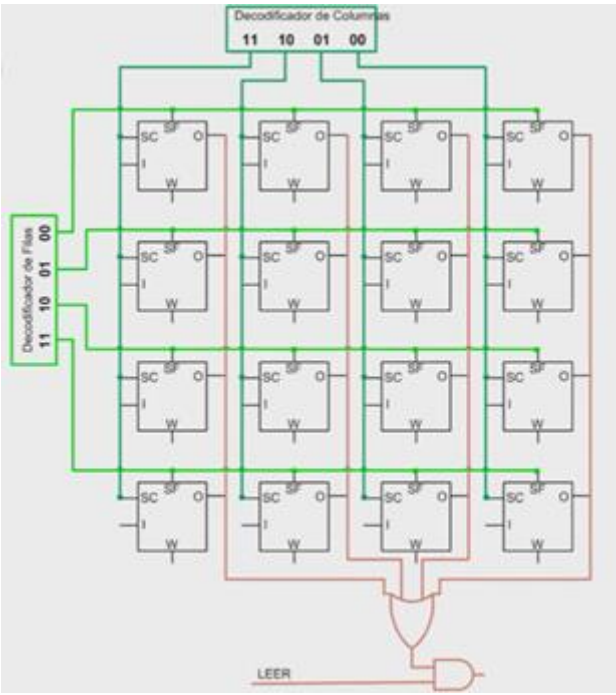
En este tipo de organización de memoria, parte de la decodificación la realiza la propia organización de la memoria. La memoria se divide en **m** matrices de 2^n biestables, donde **m** es el número de bits de la palabra de memoria y **n** el número de bits del RSM. Cada biestable de la i-sima matriz corresponde al bit de peso **i** de una de las 2^n palabras de la memoria.

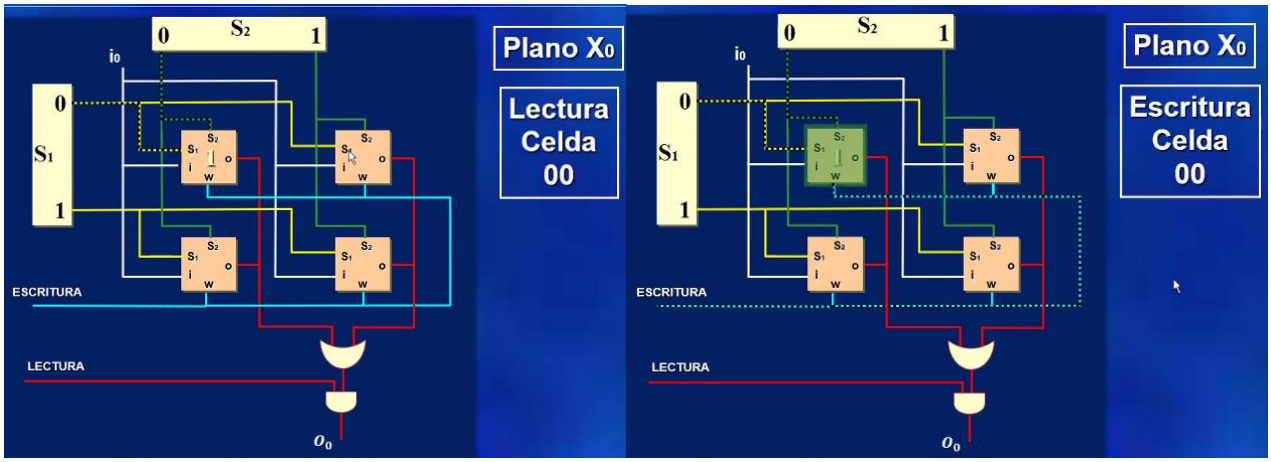
Cada biestable es alcanzado por 4 hilos, 2 hilos para la selección de filas y columnas, un hilo para la entrada de información y un hilo para habilitar la escritura.

En síntesis, a la celda básica de la memoria 2D le agregamos otra entrada de selección (S_1 y S_2), ambas deben estar en 1 para seleccionar una celda determinada. Se precisan 2 decodificadores para esta memoria, que tiene 16 palabras de 1 bit por palabra. El RDM tiene 4 bits y, por lo tanto, 16 estados.

4 Palabra de 1bitc/u, cada plano es una posición de la palabra (1 bit, un plano, 2bit, dos planos) aquellos puntos 3d que tengan los mismos s1 y s2 son una palabra. Cada memoria necesita un deco para las filas y otro para las columnas. Si hay 8 planos, y selecciona para leer en celda 00, SE SELECCIONAN LOS 8 PLANOS SIMULTANEAMENTE!!





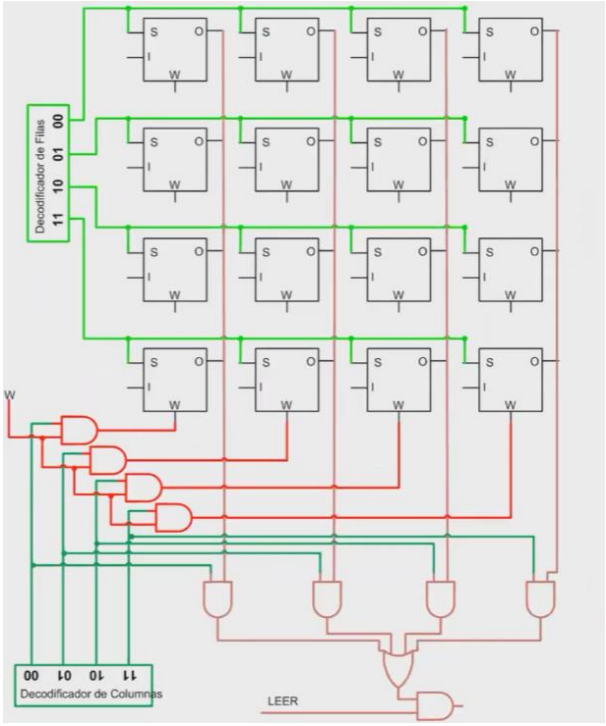


Acá *no hay crecimiento exponencial*, porque el deco está dividido.

Organización 2½D

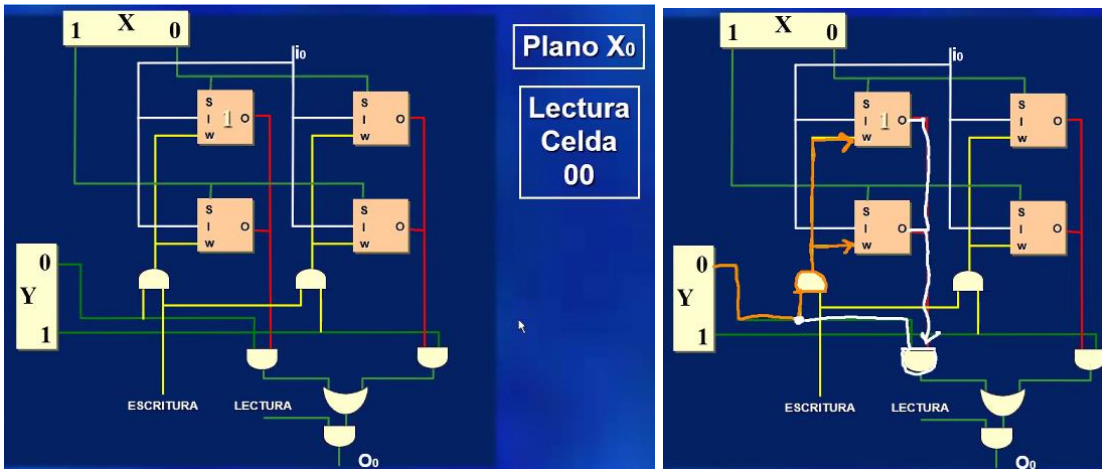
Es una organización 3d construida a partir de puntos de memoria 2d. Aprovecha los dos puntos fuertes de cada uno.

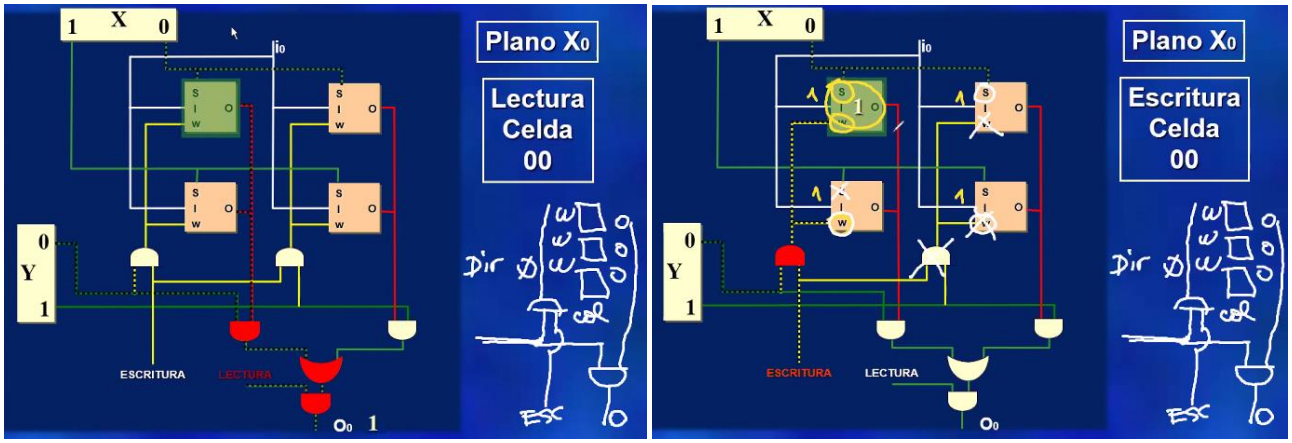
La estrategia consista que el deco de fila slecciona la final por la entrada S, el de columna como NO tiene S selecciona de forma combinacional => dos compuertas AND por cada plano para la lectura; las de escritura son dos y son para todos los planos y se desvia el cableado. El esquema de selección usa un control en las entradas LEER y ESCRIBIR para así lograr la organización 3D. Es también una selección por corrientes coincidentes, con la única diferencia respecto a la selección 3D que la habilitación para escritura no se sitúa a nivel de los biestables, sino al de las líneas de selección de columna.



Ventaja: Planos y económica.

Desventaja:





PRO Y CONTRA DE LAS ORGANIZACIONES DE MEMORIA

El sistema básico de selección lineal 2D es la organización más sencilla, sin embargo, el decodificador del sistema de selección resulta muy grande a medida que el tamaño de la memoria aumenta. Si la memoria de 16 palabras de 1 bit fue diseñada usando el sistema 2D, será necesario un decodificador con 16 x 4 entradas, y por lo tanto se utilizarían 64 diodos. En cambio para un sistema 3D se requerirán 2 decodificadores de 2 entradas y cuatro salidas, cada uno con 8 diodos, o sea, requerirán 16 diodos para ambos decodificadores. Este es un ahorro muy apreciable, que se extiende a memorias muy grandes.

En una memoria de tres dimensiones, sin embargo, la simplificación en la complejidad del decodificador se paga con la complejidad de la celda. En algunos casos, esta complejidad de la celda no es costosa, aunque es un problema y, por lo tanto, se usa una variación del esquema 3D, el 2 y 1/2D, que es un sistema de selección que usa dos decodificadores, pero las celdas de memoria son las básicas.

Direccionamiento

El direccionamiento de memoria es técnica que permite que la arquitectura pueda acceder a un apalabra de la memoria centra a través de una estrategia que permita de forma específica acceder a esa palabra memoria en la lectura. Son técnicas de acceso a la memoria a la hora de la lectura-Nivel de lógica digital, todo el valor agregado se tiene que cablear.-

Los datos afectados durante la ejecución de una instrucción están alojados en memoria principal, de modo que la instrucción hace referencia durante su ejecución a uno o más bytes que constituyen el dato u operando. Una instrucción que hace referencia a un dato lo involucra en una operación, que puede ser la lectura o la aplicación de un algoritmo que lo modifique.

Es decir, el direccionamiento son las diversas formas que disponen las instrucciones para determinar el valor de un operando o la posición de un operando o una instrucción. Se llama *objeto a la instrucción*, operando o resultado que se desea direccionar. *El objeto puede residir* en la propia instrucción, en un registro o en la memoria principal y la finalidad de los modos de direccionamiento es especificar el lugar donde se encuentra el objeto. En bastantes ocasiones la instrucción no contiene al objeto o a su dirección real, puesto que así se obtienen las siguientes ventajas:

a) Ahorro de espacio. Las instrucciones conforman programas cortos, que se leen y ejecutan más rápidamente.
b) Código reubicable y reentrante. Código reubicable es el que se puede ejecutar en cualquier zona de memoria y código reentrante es aquél que puede llamarse a sí mismo.
c) Estructuras de datos. Se simplifica notablemente el manejo de estructuras de datos (tablas, matrices, colas, listas, etc.) mediante ciertos modos de direccionamiento.

No se habla de ventajas y desventajas, es ilógico, ya que esto se toma como estrategias.

Direccionamiento Normal (directo o absoluto)

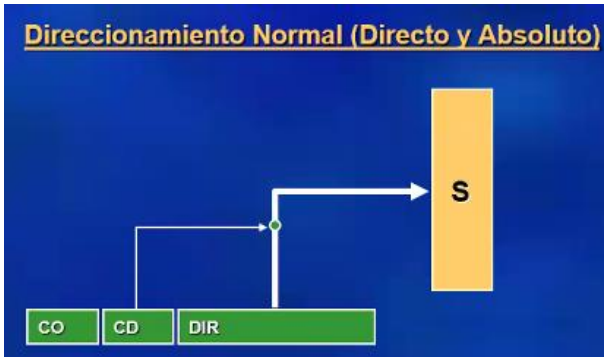
Si la maquina **solo consta de direccionamiento directo** implica mucho trabajo => es necesario diseñar circuitos que permitan disponer de un abanico de posibilidades de técnicas de direccionamiento que se adecuen mejor a distintos escenarios.

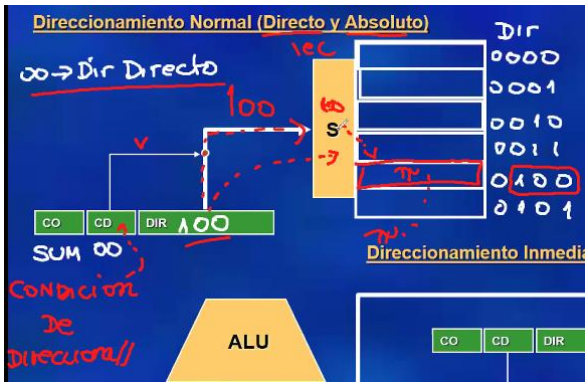
Cuando el modo de direccionamiento es directo, la referencia al dato que especifica el campo DATA queda sin alteraciones y el tiempo de captación del dato depende sólo del tiempo de un acceso a memoria.

Consiste en que la arquitectura del campo de condición de direccionamiento (CD) => puede seleccionar una de una abanico de técnicas disponibles (las cuales están cableadas). Es la técnica por

excelencia y la más sencilla, la palabra seleccionada su posición coincide con el campo DIR. Selecciona la palabra de memoria cuya ubicación sea coincidente con el valor binario del campo DIR

S → registro selección de memoria



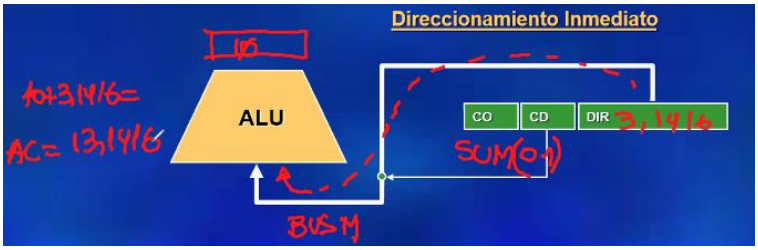


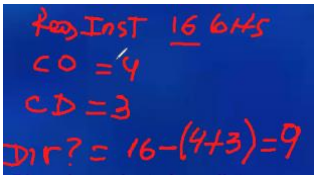
Sin embargo, hay dos modos que determinan que no se acceda a memoria para obtener el dato. El primero se llama direccionamiento implícito y el segundo, direccionamiento inmediato.

Direccionamiento Inmediato

Es Inmediato dado un DC se dice que la arquitectura hace un dir. Inmediato cuando obliga al DIR a no ir a S, si no ir a la ALU a operar ese contenido como si fuese un operando; puede contener el operando mismo y se calcula de manera directa con la ALU. Puede contener el operando mismo en DIR

Desventaja en tamaño del operando porque está en relación del tamaño del campo div pero la ventaja está en la velocidad y en la performance.



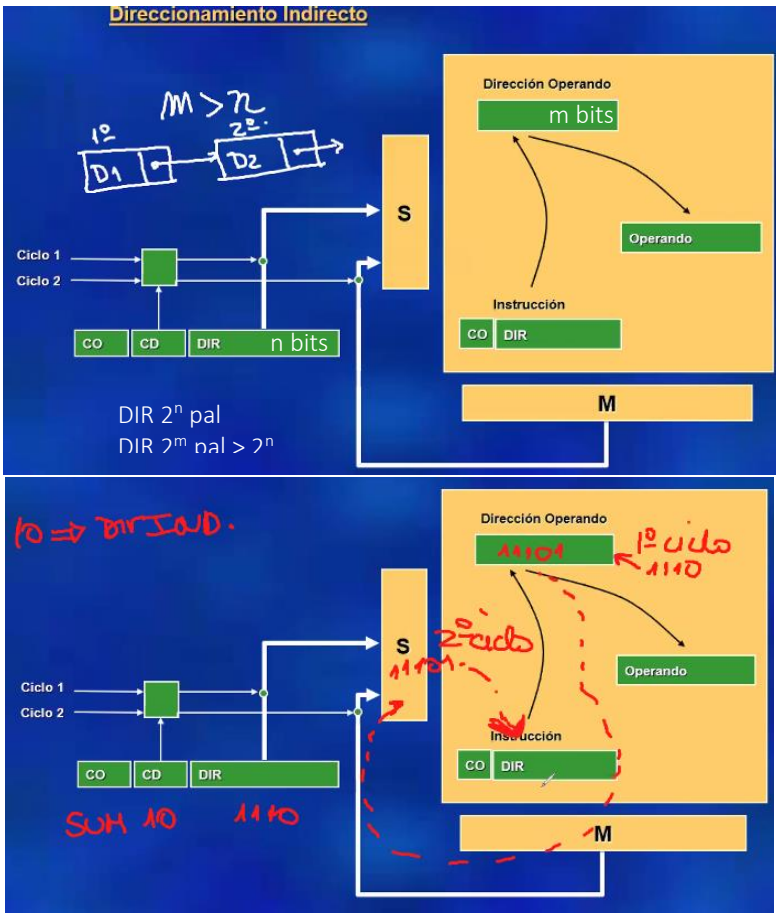


Las instrucciones que se adapten a este modo de direccionamiento involucran el dato en la instrucción en sí, pero ahora no en el campo código de operación sino en el campo DATA. Este tipo de instrucciones permite asignar un valor constante a un registro o una posición en memoria. La ventaja es que no se requiere el acceso a memoria para obtener el dato, la desventaja es que no se puede modificar el valor del operando durante el transcurso de la ejecución del programa ya que es una constante.

Direccionamiento Indirecto

Consiste en una técnica que permite acceder a un apalabra de memoria de manera indirecta, es decir la dirección del operando no es la dirección efectiva, si no que la esta hace referencia a una palabra de memoria que contenga la dirección del operando; 1º busca la dirección 2ª encuentra el operando (tiene dos ciclos)

La imagen nos muestra desde el ciclo de instrucción.



En el segundo ciclo se une el bus s con el m de forma temporal, porque m tiene la dirección del operando. Se unen con una señal de nivel (TMS= Transferencia de bus m a s) este "puente" se activa solo para el modo indirecto.

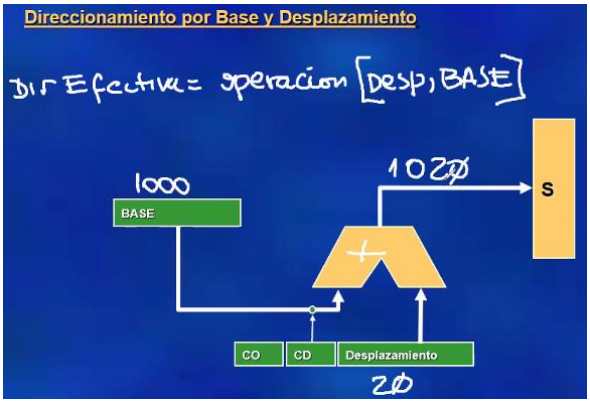
Ventaja es súper impórtate. DIR tiene una capacidad de n bits y puede direccionar 2^n bits (palabras) => la dirección de la palabra tiene m bits (siendo m>>n). Permite acceder a zonas muuuuy lejanas que no podrías solo por el bus S y permite hacer un manejo de arreglos, de punteros (es un recorrido lógico de la información de la memoria) permite manejar estructura de datos gracias a los punteros.

Desventaja consume muchos ciclos de memoria, encarece mucho la velocidad y tiempos de acceso.

Existen computadoras que tienen un nivel de indirección múltiple, esto es, se accede a una posición en memoria que indica con un bit que su contenido se usa como señalador o "puntero" de otra; a la vez, la nueva posición también es un pointer, y así sucesivamente hasta llegar a aquella cuyo bit indicativo de pointer sea cero. En esta última se halla entonces el dato buscado.

Direccionamiento por Base y desplazamiento

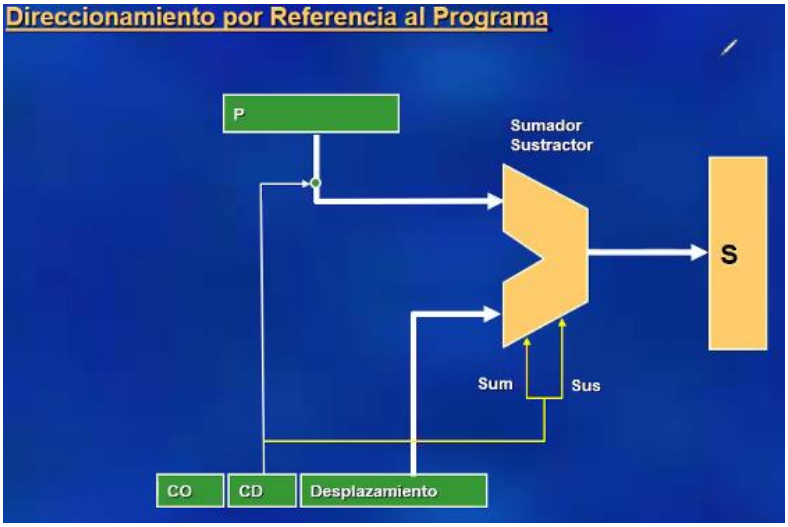
Plantea que la direccione efectiva surge de la operación de registro base y del contenido de DIR (entre desplazamiento y base).



Direccionamiento por Referencia al programa

Plantea que la dirección efectiva de la palabra de memoria surge de sumar o restar el campo DIR (desplazamiento) con el contador de programa. Esta técnica permite, basándose en el valor del conto de programa tiene un valor fijo de DIR, entonces es muy utilizada para hacer barridos de palabras consecutivas.

(respl)±(cant de prop) = Direccion efectiva de a palabra.

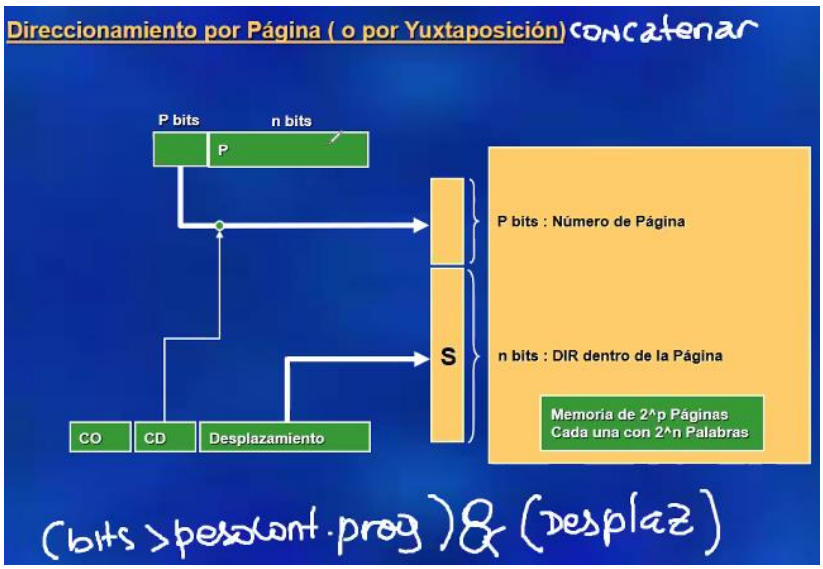


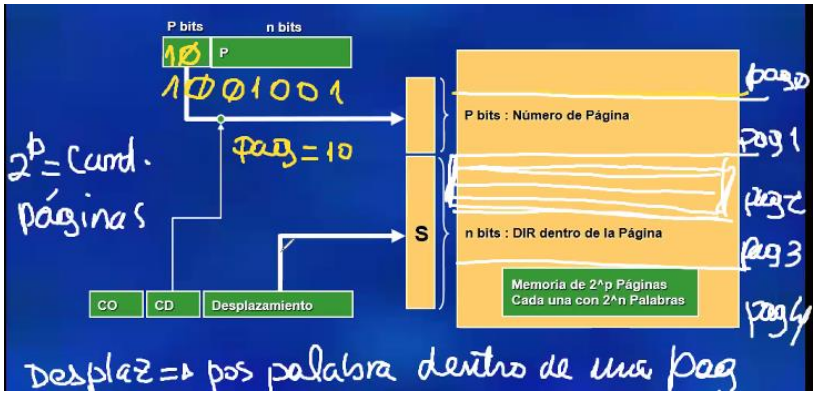
Direccionamiento por Pagina (yuxta=concatenar)

Concatena el contenido de los bits de mayor peso del contador de programa con el valor DIR. El contador de programa mantiene fijos los de mayor peso y acelera los de menor => la memoria se organiza en 2^p tal que p es la cantidad de bits de mayor peso del contador de programa. 2^p=cantidad de páginas (manera virtual de ver agrupamiento de palabras en una unidad de información)

Desplazamiento indica posición de la palabra dentro de una página.

Es fundamental porque permite la multiprogramación, el procesador puede hacer procesos concurrentes, hay distintos procesos que corren en distintas páginas.





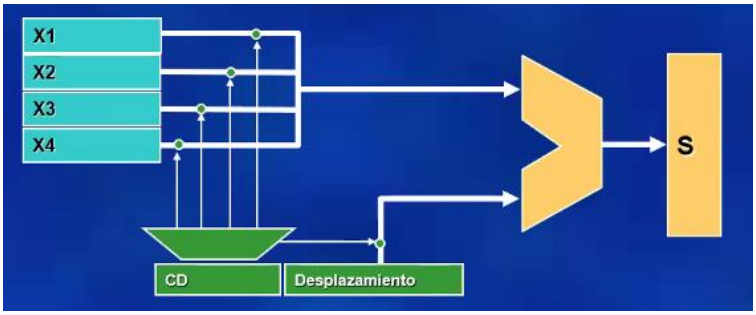
Direccionamiento Indexado

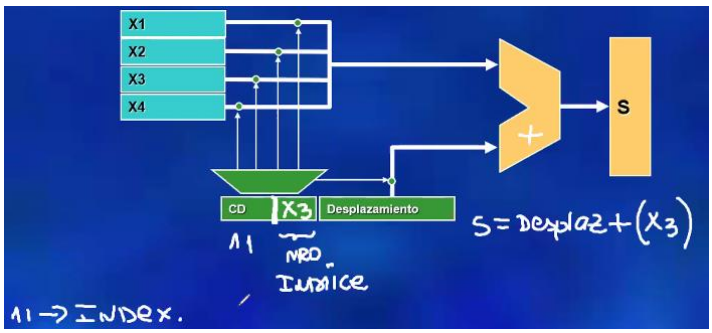
Es uno de los más usados. Consiste en muchos registros bases modificables por el programa. La dirección efectiva surge del operar del contenido del índice con el desplazamiento. En un programa se utilizan algoritmos que involucran índices, por ejemplo, para sumar datos en posiciones contiguas de memoria.

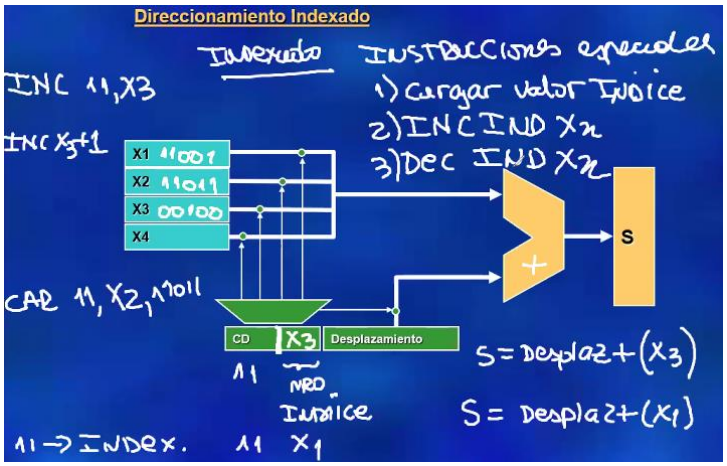
Hay instrucciones especiales -una para cargar el valor del índice -otra para incrementar el índice -otra para decrementar el índice;

El set de instrucciones debe proveer una instrucción de carga del registro índice que actúa como "señalador de elementos". Es común que una CPU contenga más de un registro índice, por lo que esto se debe reflejar en el código de la instrucción.

Hay tanto vectores como índices, a nivel físico es una limitación. En programación un vector es un conjunto o una agrupación de variables del mismo tipo, cuyo acceso se realiza por índices. La forma de acceder a los elementos del array es directa, o sea que el valor del dato deseado se obtiene a partir de su índice y no hay que ir buscándolo elemento por elemento







ORGANIZACIÓN Y BUSQUEDA DE LAS INFORMACIONES EN MEMORIA

Conceptos de Vector, Lista y Puntero

La organización más clásica de las informaciones en memoria consiste en almacenarlas sucesivamente, una detrás de otra. Muy generalmente este es el caso de las instrucciones sucesivas de un programa. Necesitará entonces, para buscar las informaciones sucesivas disponer de un **puntero**, cuyo valor sea igual sucesivamente a las direcciones de las informaciones almacenadas en secuencia. En lo que concierne a las instrucciones éste es, evidentemente, el papel del contador ordinal. Un conjunto de informaciones almacenadas en secuencia se denomina **vector**.

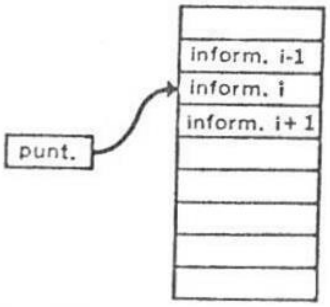


Fig. 11. Estructura de vector

Este método no encaja bien en el caso de las manipulaciones de listas de datos, tales como la extracción o la inserción de datos, la combinación de listas de datos, etc., porque estas exigen desplazamientos de las informaciones en memoria; se recurre a una **estructura de lista** en que cada dato va acompañado de un puntero, que contiene la dirección del próximo dato. Las operaciones de inserción, extracción, combinación, se llevan a cabo por simple manejo de punteros.

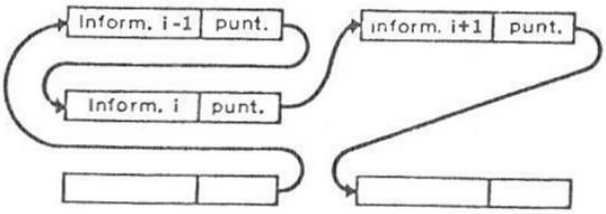


Fig. 12. Estructura de lista

Concepto de Tabla

Una tabla establece una correspondencia entre 2 tipos de información: la información significativa que se busca, y la información de entrada que permite localizarla. Esta última se llama también dirección **asociativa o etiqueta.**

La memorización de una tabla puede reducirse a la constitución de un simple vector, si la posición de la información significativa en el vector es calculable a partir de la información de entrada.

En el caso general, la tabla se almacenará bajo la forma de un vector, cada uno de cuyos elementos contendrá la información de entrada y la información significativa correspondiente. Así será el caso de la tabla de correspondencia entre el código nemotécnico de operación, empleado en lenguaje ensamblador y la representación binaria de máquina del código de operación.

La búsqueda en tabla exige sucesivas comparaciones de la información de entrada con todas las posibles informaciones de entrada, hasta llegar a la coincidencia.

Conceptos de Pila y de Cola de Espera

Estos dos conceptos designan organizaciones particulares de datos en la memoria, en las que el orden de utilización de las informaciones depende del orden en que han sido introducidas.

Cola de Espera: La cola de espera funciona según un principio análogo al que conocemos en los negocios: se tiene acceso a la información de acuerdo con el esquema "primer llegado, primero en ser atendido". Se necesitan dos punteros para definir el estado de una cola de espera, uno apuntando a la última palabra introducida, el otro a la próxima palabra por extraer.

Pila: El mecanismo de la pila recuerda a la pila de camisas en el armario: se accede a la última información almacenada, con el esquema "último en entrar, primero en salir". El estado de la pila se consigue con un solo puntero, pues basta con conocer el emplazamiento de la cima de la pila.

Las pilas y colas de espera pueden constituirse en memoria central por programación.

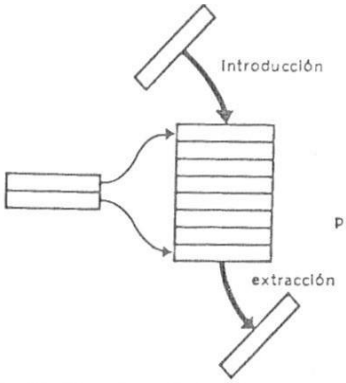


Fig. 13. Mecanismo de la cola de espera: primero en entrar, primero en salir

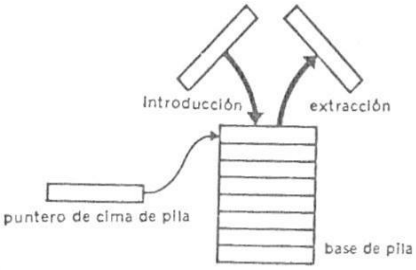


Fig. 14. Mecanismo de la pila: ultimo en entrar, primero en salir

PILAS

Básicamente una pila es un conjunto de posiciones consecutivas en la memoria, en las cuales se pueden colocar los operandos. El nombre de pila se deriva del hecho de que la memoria está organizada como una pila de platos (puede considerarse cada operando como si fuese un plato).

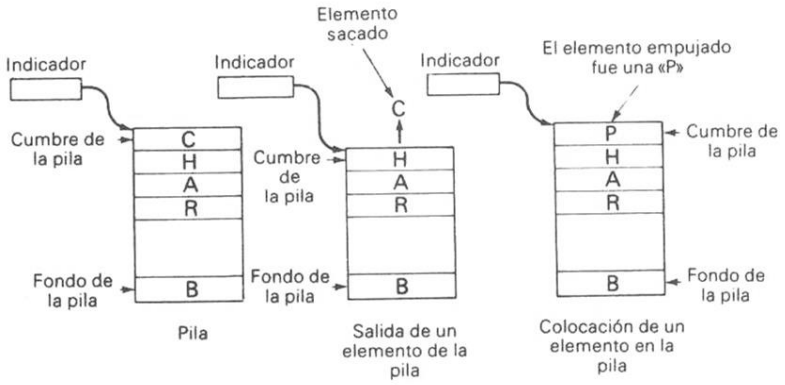


Fig. 10.2. Operaciones de la pila.

Las dos operaciones de la pila son la inserción y el desecho. La operación de inserción de un operando se llama **empujar** (push), es decir, añade un nuevo operando en la cumbre de la pila.

La operación de desecho de un operando se llama sacar (pop), el operando que se elimina es el que está en la cima de la pila. Se dice que el primer operando colocado en la pila está en el fondo de la pila. Sólo el operando que esté en la cumbre (top), está disponible de forma inmediata.

Las pilas se mantienen en una memoria como un conjunto de palabras. Cada palabra tiene, por lo tanto, una longitud fija (número de bits) y una dirección. El indicador de la pila es un registro que contiene la dirección del operando superior de la pila. El indicador de pila se incrementa o decrementa cada vez que se introduce o saca un operando.

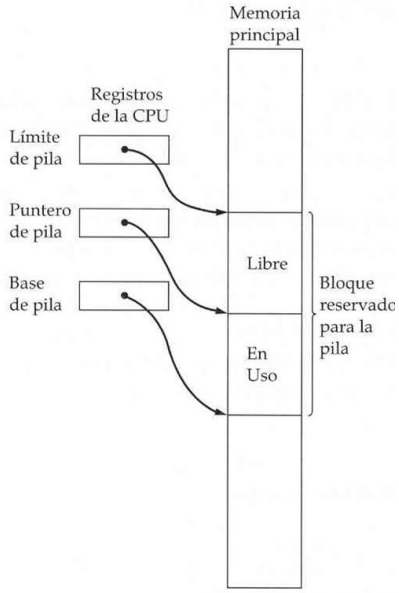
Si se da una instrucción de SUMAR a un computador con una arquitectura de pila, los dos operandos superiores de la pila serán sacados y sumados, y luego el resultado de la suma se colocará en la cumbre de la pila. Análogamente, para una instrucción de MULTIPLICAR ocurre exactamente lo mismo.

TABLA 9.11 Operaciones orientadas al uso de la pila

PUSH	Añade un nuevo elemento en la cabecera de la pila.
POP	Elimina el elemento de la cabecera de la pila.
Operación Unaria	Realiza una operación con el elemento de la cabecera de la pila. Sustituye el elemento de la cabecera con el resultado.
Operación Binaria	Realiza una operación con dos elementos de la cabecera de la pila. Elimina dichos elementos. Pone el resultado de la operación en la cabecera de la pila.

La implementación de una pila requiere que exista un cierto conjunto de posiciones utilizadas para almacenar los operandos de la pila, por lo tanto, en memoria principal (o en memoria virtual) se reserva un bloque de posiciones contiguas para la pila. Una parte de esta reserva va a estar parcialmente llena de operandos y otra va a estar disponible para que crezca la pila. Para un funcionamiento correcto se necesitan 3 direcciones, normalmente memorizadas en registros de la CPU:

- **Puntero de la pila:** contiene la dirección del tope o cabecera de la pila.
- **Base de la pila:** contiene la dirección de la posición de la base de la pila, esto es por si se intenta realizar un POP cuando la pila está vacía.



(a) Toda la pila en memoria

FIGURA 9.14 Organizaciones de pila usuales

- **Límite de la pila:** contiene la dirección del otro extremo de los bloques reservados. Si se intenta un PUSH cuando los bloques están utilizados en su totalidad, se informa de un error.

Su uso en el microprocesador está dirigido en su mayoría para el manejo de subrutinas (llamadas y retornos de estas) e interrupciones. La ventaja de una pila de memoria es que el procesador puede referirse a ella sin tener que especificar una dirección ya que la dirección está siempre disponible y actualizada automáticamente en el indicador de la pila.

Así, un procesador puede hacer referencia a una pila de memoria sin especificar una dirección. Por esta razón, las instrucciones que incluyen operaciones de pila se llaman **dirección cero, instrucciones implícitas,** o **instrucciones sin dirección**, son del tipo de instrucciones que no especifican ninguna posición en la memoria para el operando ya que la dirección del operando esta en el indicador de la pila.

También es necesario mover operandos de la memoria a la pila, o de la pila a la memoria, y la palabra instrucción para estas operaciones debe ser más grande debido a que se debe especificar la dirección de memoria. (Estas instrucciones serán como las instrucciones de una sola dirección, excepto que los operandos se mueven desde y hacia la pila en vez de hacia o desde el acumulador).

DECODIFICADORES

Cantidades discretas de información se presentan en sistemas digitales con códigos binarios. Un código binario de **n** bits es capaz de representar hasta 2^n elementos diferentes de información codificada. Un decodificador es un circuito combinacional que convierte la información binaria de **n** entradas a un máximo de 2^n líneas de salida., de las cuales solo una de estas 2^n líneas de salida será seleccionada por el decodificador (es decir tendrá el valor 1). Este circuito particular se llama decodificador de varios a uno, matriz decodificadora, o simplemente decodificador.

El siguiente esquema muestra un decodificador que es completamente paralelo en su construcción y diseñado para decodificar 3 entradas (flip-flops). Tiene $2^3 = 8$ líneas de salidas, y se seleccionara una sola línea de salida para cada uno de los 8 estados que pueden tomar las tres entradas (flip-flop), dicho esto este decodificador opera de la misma forma que el de 2 entradas, explicado en memoria 2D.

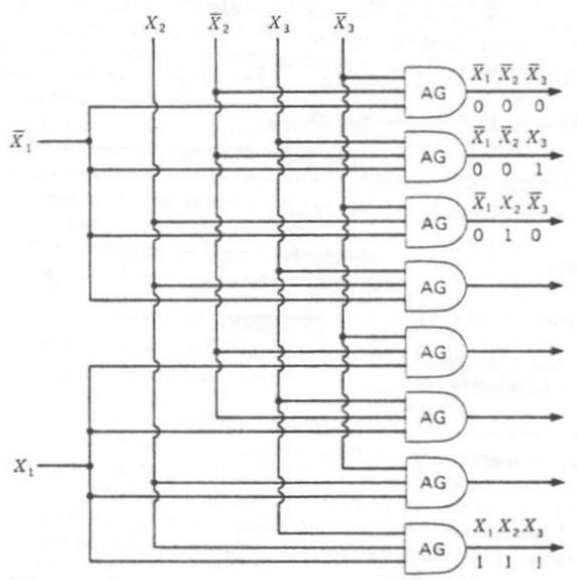


Fig. 7.5. (*b*) Decodificador paralelo.

Este tipo de decodificador se construye usando diodos (o transistores) para las compuertas AND. La regla es: el número de diodos (o transistores) usados en cada compuerta AND es igual al número de entradas a cada compuerta AND.

El número de compuertas AND es igual al número de líneas de salida, que es igual a 2^n (n = es el número de flip-flops de entrada que se están decodificando).

El número total de diodos es por lo tanto igual a **n x 2^n** y para la matriz decodificadora binaria de 3 entradas se necesitan 24 diodos. El número de diodos necesarios aumenta bruscamente con el número de entradas del circuito.

Para construir un decodificador paralelo necesitamos tantos operadores AND como salidas. Si a los operadores AND los materializamos con diodos, necesitamos 1 por cada entrada. Los operadores AND tendrán tantas entradas como biestables tenga el registro de selección.

Existe un decodificador **tipo árbol**, que decodifica cuatro flip-flops, y por lo tanto tiene 2^4 = 16 líneas de salida. Para construir este circuito en particular se necesitan 56 diodos, mientras que se necesitan 4 x 2^4 = 64 diodos para construir el decodificador tipo paralelo.

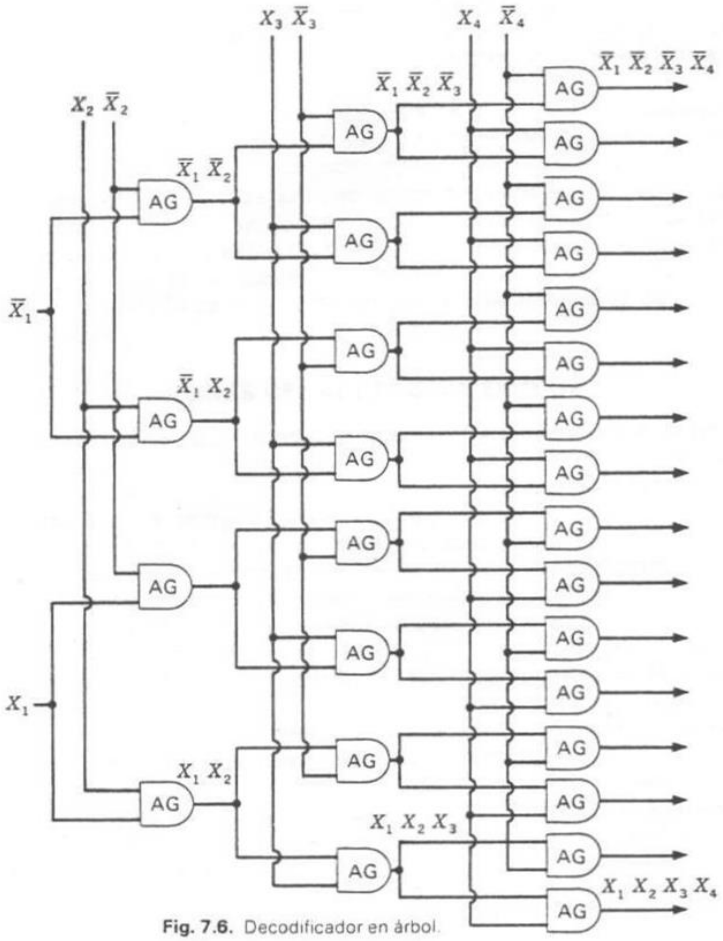


Fig. 7.6. Decodificador en árbol.

El circuito llamado **decodificador equilibrado multiplicativo**, requiere de solo 48 diodos, vemos que este decodificador necesita de un número mínimo de diodos para su construcción a diferencia de los explicados anteriormente. Sin embargo el circuito decodificador paralelo tiene la ventaja de que es más rápido y regular en construcción de los 3 tipos explicados.

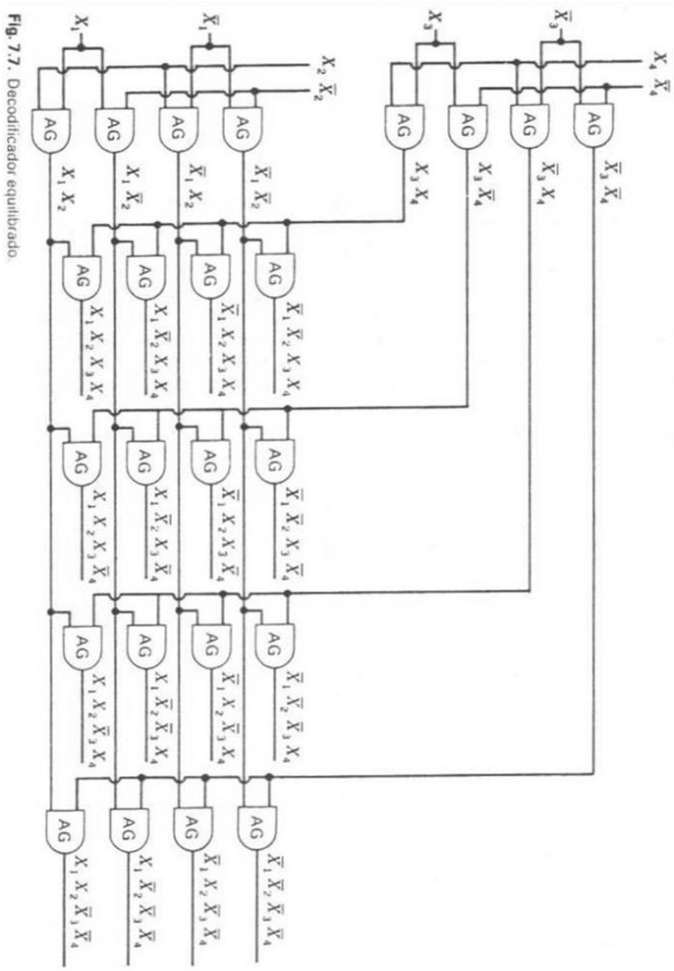


Fig. 7.7. Decodificador equilibrado.

En lo sucesivo simplemente dibujaremos el decodificador como una caja de n entradas y 2^n salidas, entendiéndose que uno de los 3 tipos se usa en la caja; sólo las entradas sin complementación están conectadas al decodificador, y los inversores están incluidos en el paquete. Por lo tanto para un decodificador de 3 entradas tendrá solo 3 líneas entradas y 8 líneas de salidas.

El Dispositivo de Selección de Memoria no es otra cosa que un circuito combinacional, habitualmente conocido como decodificador. Las entradas al circuito provienen del Registro de Selección.

Si el RSM tiene **n** biestables, el DSM podrá seleccionar **2^n** palabras de memoria. Este tipo de decodificadores pone en 1 la salida correspondiente, de acuerdo al valor contenido en el RSM.

Si quisiéramos construir un decodificador para un registro de selección de n biestables, con probabilidad de seleccionar 2^n palabras de memoria, necesitaremos:

2^n compuertas AND.

Cada compuerta AND tendrá **n** entradas

Para calcular la cantidad de diodos aplicaremos

$$\boxed{d = n \cdot 2^n}$$

<u>Decodificador BDC a Decimal:</u> los elementos de información en este caso son los diez dígitos decimales representados por el código BDC. El código tiene cuatro bits, por lo tanto, el decodificador deberá tener 4 entradas para aceptar el dígito codificado y diez salidas para cada uno de los dígitos decimales. Esto dará un decodificador de 4 a 10 líneas de BDC a Decimal.

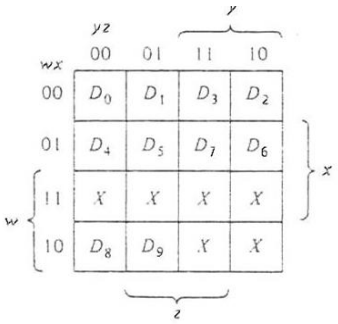


Figura 5-9 Mapa para simplificar un decodificador BDC a decimal

En vez de dibujar 10 mapas, se dibujara solamente un mapa y se escribirán cada una de las variables de salida D_0 hasta D_9, dentro de su cuadrado de término mínimo, como muestra la figura. Hay 6 combinaciones de entrada que nunca ocurren, por lo tanto se marcan los cuadrados correspondientes con X, de manera que podamos utilizarlos para simplificar al número mínimo de literales. Esta es sólo una de las tantas maneras de tratar las condiciones de no importa, lo que también podríamos hacer cuando una combinación de entrada no sea válida es que todas las salidas sean igual a 0.

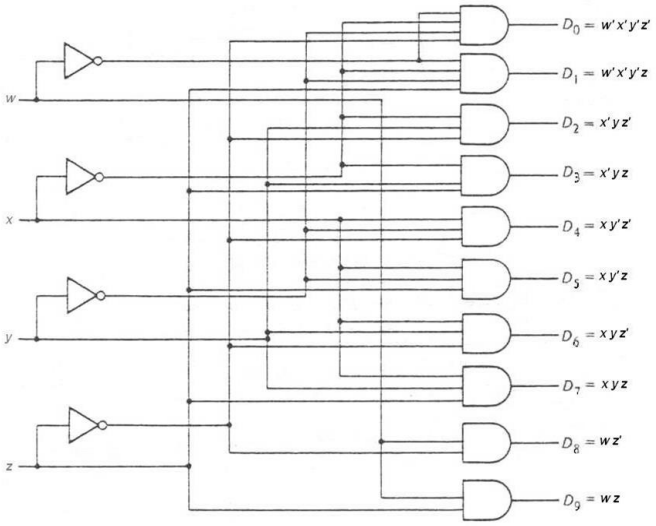


Figura 5-10 Decodificador BDC a decimal

CODIFICADORES

Un codificador es una función digital que produce una operación inversa a la del decodificador. Un codificador tiene 2^n (o menos) líneas de entrada y n líneas de salida. Las líneas de salida generan el código binario para las 2^n variables de entrada. La figura muestra un decodificador octal a binario que consiste en 8 entradas, una para cada uno de los 8 dígitos y tres salidas para generar el número binario correspondiente. Este se construye a partir de la siguiente tabla de verdad.

Entradas								Salidas		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Nótese que D_0 no se conecta a ninguna compuerta OR; la salida binaria debe ser sólo ceros en este caso. Una salida de sólo ceros se obtiene también cuando todas las entradas sean cero. Esta discrepancia puede resolverse agregando una salida más para indicar el hecho de que todas las entradas no son ceros. El codificador de la figura asume que solamente una línea de entrada puede ser igual a 1 en cualquier momento; de otra forma el circuito no tiene significado. Nótese que el circuito tiene ocho entradas y podría tener 2^8 = 256 combinaciones de entrada posibles. Solamente 8 de estas combinaciones tienen significado. Las otras combinaciones son condiciones de no importa.

Existen otros codificadores que establecen una prioridad de entrada para asegurar que solamente la línea de entrada de la más alta prioridad se codifica. Estos codificadores se llaman **codificadores de prioridad**. Por ejemplo, tomando la tabla de verdad dada, si la prioridad es dada a una entrada con un número de suscrito mayor con respecto a un número suscrito menor, entonces si ambos D_2 y D_5 son lógica 1 simultáneamente, la salida será 101 porque el D_5 tiene una mayor prioridad sobre D_2. Cabe aclarar que la tabla de verdad para un codificador de prioridad es diferente a la que se acaba de presentar.

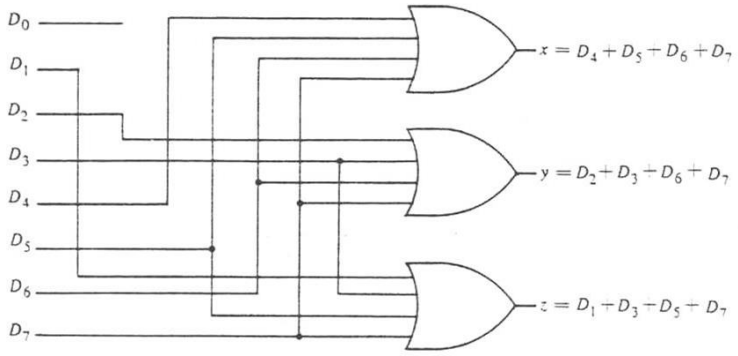


Figura 5-15 Codificador octal a binario

CONEXIÓN DE LOS CIRCUITOS DE MEMORIA A UN BUS

La tendencia actual, fundamentalmente en los sistemas con microprocesador, es conectar la memoria del computador a la CPU por medio de un BUS, este es un conjunto de alambres, que es compartido por todos los elementos de memoria que se utilizan. El Bus que conecta la memoria al microprocesador consiste en:

- Un conjunto de **líneas de dirección**, llevan la dirección de la palabra a seleccionar.
- Un conjunto de **líneas de datos**, para entrar datos a la memoria o sacar datos de ella.
- Un conjunto de **líneas de control**, para controlar las operaciones de lectura o escritura.

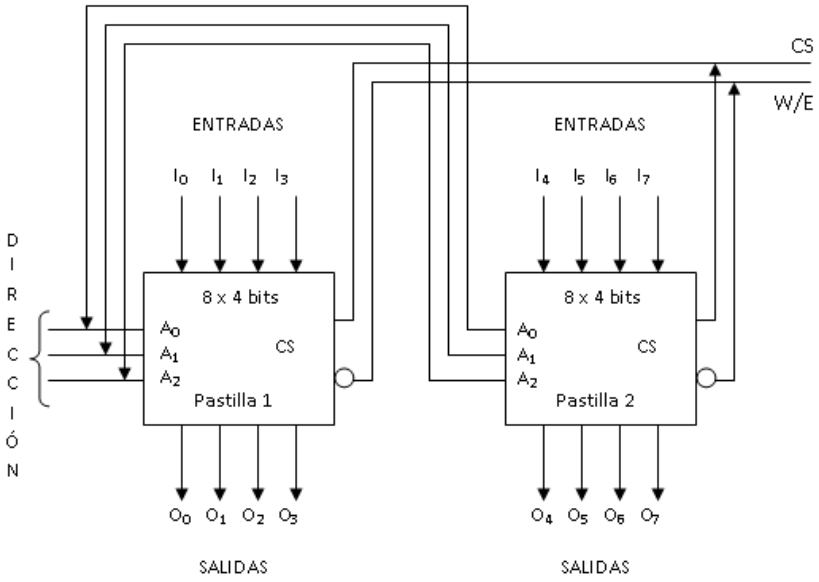
Figura 7.12 Bus para un sistema de computador. Líneas del bus.

La figura anterior ilustra un Bus para un microprocesador con tres líneas de dirección, tres líneas de entrada de datos, tres de salida de datos y 2 señales de control. Podemos deducir que la memoria que puede conectarse a ese Bus, tendrá como máximo 8 palabras de 3 bits por palabra. La línea L/E se pondrá en 1 o 0 según si la CPU indica que se lea o escriba en la memoria. La línea H/M (habilitación de memoria) se pondrá en 1 cuando en la memoria se va a leer o escribir, caso contrario será 0.

Con frecuencia surgen casos en los que no es adecuado el número de palabras de una pastilla (C. I. de memoria) o el número de bits por palabra o las dos cosas a la vez. El problema puede solucionarse colocando pastillas en paralelo. La figura muestra una conexión para incrementar el número de bits por palabra (pero no el número de palabras).

Si observamos vemos que hay en paralelo 2 pastillas de 8 palabras, de 4 bits por palabra, que dispuestas de esa forma da como resultado una memoria cuyo número de palabras sigue siendo 8, pero el número de bits se ha incrementado de 4 a 8. Los 3 bits de dirección se aplican en la entrada de dirección de ambas memorias. Los terminales CS de las pastillas se unen, lo mismo que los WE.

Las entradas de selección de pastillas y habilitación de escritura, seleccionan y habilitan simultáneamente las pastillas. La pastilla 1 acepta y almacena 4 bits (0, 1, 2, 3) y la pastilla 2, 4 bits más (4, 5, 6, 7). Por supuesto se pueden conectar en paralelo más pastillas adicionales para incrementar aún más el número de bits por palabra.



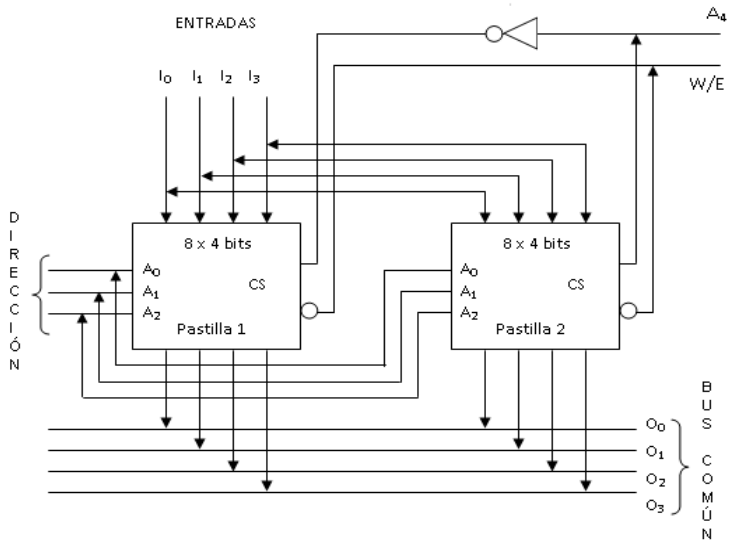
CS: Selección de pastilla.

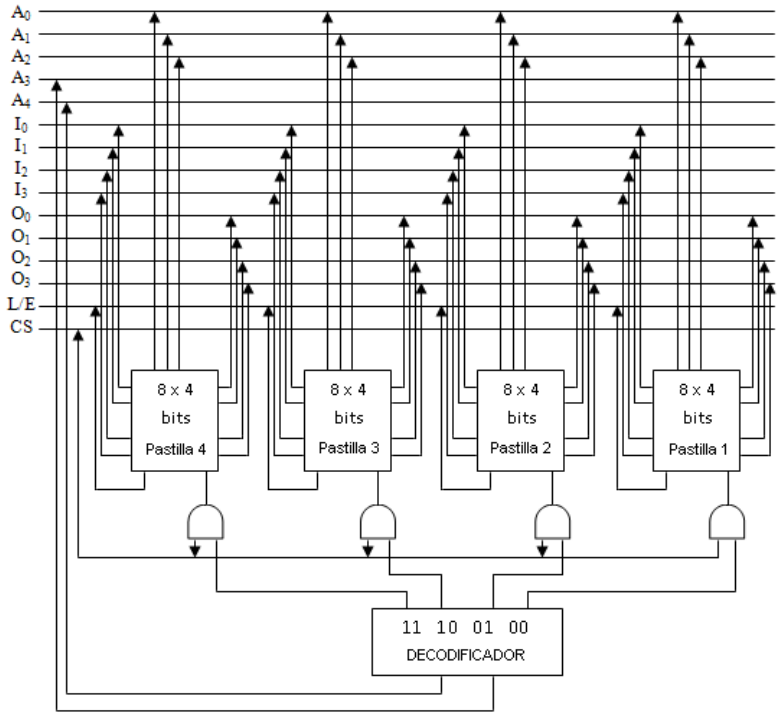
W/E: Habilitación de escritura.

Con dos pastillas de memoria de 8 palabras, 4 bits por palabra, se obtiene una memoria de 8 palabras de 8 bits por palabra.

También pueden obtenerse conexiones que permitan incrementar el número de palabras manteniendo fijo el número de bits por palabra. La siguiente figura muestra la conexión de 2 pastillas de 8 palabras de 4 bits por palabra para obtener una memoria de 16 palabras de 4 bits por palabra. Los 3 bits de dirección se aplican a ambas pastillas, como en el caso anterior, sin embargo, la entrada CS del sistema de memoria ahora es un bit de dirección adicional, que llamamos A_4; cuando $A_4 = 1$, se activa la pastilla 2, y cuando $A_4 = 0$ se activa la pastilla 1, es decir, cuando se activa la entrada CS de una pastilla, se desactiva la entrada de la otra. Los bits de dirección A_0, A_1, A_2, seleccionan la posición de una palabra particular en la pastilla seleccionada. Los bits de entrada de datos y la entrada WE son los mismos para ambas pastillas, hay que tener

en cuenta que sólo en una pastilla se escribe dependiendo del valor de A_4. Ahora una palabra se lee a veces de una pastilla y a veces de otra. La palabra se transmitirá a un bus común, independientemente de la pastilla que la origine. A diferencia de la anterior figura, en esta necesitaríamos agregar un multiplexor para seleccionar la salida de una o de la otra pastilla, mientras que en la otra figura no es necesario un multiplexor ya que la longitud de la palabra es de 8 bits.





CS: Selección de pastilla.

L/E: Habilitación de lectura / escritura.

Con 4 pastillas de memoria de 8 palabras, 4 bits por palabra, se obtiene una memoria de 32 palabras de 4 bits por palabra.

MEMORIAS DE SÓLO LECTURA (ROM)

Una memoria de sólo lectura (Read only memory, ROM), es una memoria de la que podemos leer, pero en la que no se puede escribir. Son memorias de acceso aleatorio, la información almacenada en estas memorias es introducida de forma tal que la información es permanente o semipermanente, como consecuencia tienen la característica de ser **no volátil**; si bien, no necesita energía eléctrica para mantener guardados los datos, pero si para leerlos.

Algunas veces la información almacenada en una ROM se coloca en la memoria en el momento de la construcción y algunas veces se usan dispositivos en los cuales, la información puede ser cambiada.

Básicamente una ROM es un circuito combinacional formado por compuertas AND seguidas de OR, con varias líneas de entradas y salidas, de tal forma que para cada valor de entrada existe un solo valor de salida, o sea, una ROM físicamente es un circuito combinacional que ejecuta una tabla de verdad.

El siguiente diagrama en bloques consiste en n líneas de entrada y m líneas de salida. Cada combinación de bits de las variables de entrada se llama una dirección. Cada combinación de bits que sale por las líneas de salida se llama una palabra. El número de bits por palabra es igual al número de líneas de salida m. Una dirección es esencialmente un número binario que denota uno de los términos mínimos de n variables. El número de direcciones diferentes posibles con n variables de entrada es 2^n.

Una palabra de salida puede ser seleccionada por una dirección única y como hay 2^n direcciones diferentes en una ROM, hay 2^n palabras diferentes que se dice que están acumuladas en la unidad.

Si consideramos una ROM de 32 x 8, esta consiste en 32 palabras de 8 bits cada una. Esto significa que hay 8 líneas de salida y 32 palabras distintas almacenadas. En algunas ocasiones una ROM se especifica por el número total de bits que contiene, el cual será 2^n x m, por ejemplo una ROM de 2048 bits puede organizarse como 512 palabras de 4 bits cada una (2^9 x 4 = 2048 bits)

Estas memorias se construyen generalmente en forma matricial y las compuertas lógicas podrán estar materializadas mediante diodos, transistores MOS o transistores bipolares.

Hay disponibles pastillas de ROM que permiten tanto al usuario como el fabricante almacenar información. Son las PROM (programmable read-only memories). Estas memorias pueden ser escritas una sola vez. Algunas de las técnicas pueden ser, mediante matrices de diodos donde en todas las intersecciones de la matriz encontramos conexiones con enlaces fusibles o diodos conectados en oposición.

Si la memoria ROM se fabrica de tal forma que el contenido de la memoria puede fijarse como lo desee el usuario y posteriormente puede borrarse y escribirse nuevamente, se dice que es una EPROM (ROM borrable y reprogramable – Erasable – Programmable read - only memories).

La forma de borrar las EPROM es mediante la exposición a la luz ultravioleta (U. V.) durante 20 o 30 minutos. Son escritas mediante determinados voltajes aplicados a sus entradas.

Las memorias ROM eléctricamente alterables (EAROM) tienen la característica de que pueden ser borradas y re-escritas por medio de voltajes correctamente aplicados. Estas memorias tienen a ventaja de que se actúa eléctricamente en forma directa sobre el punto de memoria a modificar.

La EPROM al ser expuestas a la luz U. V. se borran completamente en cambio en las EAROM puede seleccionarse cuál es la información que desea borrar y reprogramar mediante dispositivos provistos por las compañías fabricantes de esas memorias. Algunos de estos dispositivos para reprogramar la memoria, son operados desde un teclado, algunos desde cintas, y algunos, desde entradas externas, tales como un microprocesador.

Cuando va a programarse un circuito PROM o EPROM, existe generalmente una habilitación de escritura que debe ponerse a 1 y que hace que las líneas de salida se habiliten para recibir datos, a continuación se fijan las líneas de dirección o la posición en la que se desea escribir. Luego se escribe, aplicando una secuencia de voltajes de gran amplitud a las líneas de salida. Aplicar un borrador o exponer la memoria a la luz ultravioleta, normalmente borra todo su contenido.

En las memorias ROM, la escritura demanda muchísimo más tiempo que su lectura.

Las ROM son ampliamente utilizadas para ejecutar circuitos combinacionales complejos directamente de su tabla de verdad. Son muy útiles para convertir de un código binario a otro (como ser ASCII a EBCDIC o viceversa). También en funciones aritméticas como multiplicadores, para mostrar caracteres en un tubo de rayos catódicos, y en cualquier otra aplicación que requiera un gran número de entradas y salidas.

Se emplean además en el diseño de la unidad de control de los computadores digitales. Como tales, se utilizan para almacenar patrones fijos de bits que representen una secuencia de variables de control necesarias para habilitar las distintas operaciones del sistema. Las unidades de control que usa una ROM para almacenar información de control binario se llaman Unidad de Control Microprogramada.

En una PC la porción de memoria principal que es ROM se denomina ROM BIOS ("Basic Input Output System"). Contiene por un lado programas que se ejecutan cuando se enciende un computador y sirven para:

- Verificar el correcto funcionamiento del hardware y su configuración.
- Traer del disco a memoria principal (o sea escribir en ésta) una copia de programas del sistema operativo del computador (acción conocida como "bootear o arrancar" el sistema).
- Por otro lado, almacena programas que se usan permanentemente para la transferencia de datos entre periféricos y memoria, sea en operaciones de entrada o salida de datos.

También la ROM BIOS contiene tablas, por ejemplo relativas a características de discos.

UNIDAD DE CONTROL
RUTA DE DATOS (ABACUS)

Para analizar el funcionamiento de un computador digital partimos de la siguiente base: un programa es registrado en memoria antes de comenzar su ejecución. Esta memoria se llama Memoria Central o Memoria Principal y entorno suyo se organiza el resto de las diferentes unidades de la máquina. Esta memoria almacena dos clases de información:

- **Las instrucciones del programa** que la máquina deberá ejecutar. La Unidad de Control (Unidad de Instrucciones o Unidad de Gobierno) tratará las instrucciones. **FAMILIA IBO →** Instrucciones con Búsqueda de Operando. Por ejemplo la suma, resta, or, and, xor.

  Mayus → van las entradas de ALU (operaciones, señal de habilitación por ejemplo)

  Minus → van en minúscula (sus, or, etc)

- **Los datos (operandos)** con los cuales efectuará la máquina los tratamientos dictados por las instrucciones. La Unidad Aritmética y Lógica (Unidad de Proceso) tratará los datos, de acuerdo a lo "indicado" por la Unidad de Control (UC).

La UC extrae de la memoria central la nueva instrucción a ejecutar, la analiza y establece las conexiones eléctricas correspondientes dentro de la ALU, extrae de la memoria central los datos implicados por la instrucción, desencadena el tratamiento de dichos datos en la ALU y, eventualmente, almacena el resultado en MC. **La ALU** opera con los datos que recibe siguiendo órdenes de la UC.

Al conjunto UC y ALU se lo denomina Unidad Central de Procesamiento (UCP) o Procesador Central (cuando están integradas en una sola pastilla -chip- se denomina microprocesador).

La UC necesita dos registros para operar:

1. Contador de instrucciones, contador de programa o contador ordinal, que contiene la dirección de la próxima instrucción por ejecutarse. Este registro aumenta su contenido en uno para pasar a la siguiente instrucción.
2. Registro de instrucción, que contiene la instrucción extraída de la MC. Este registro tiene (al menos en Abacus) dos partes:
   a. El código de operación, que indica que tratamiento se debe seguir.
   b. La dirección del operando, dónde se encuentra almacenado.

La UC tiene un órgano denominado Generador de secuencias o **Secuenciador** que, luego de analizar el código de operación, distribuye las órdenes al conjunto de unidades del computador para hacerles ejecutar las distintas fases de la instrucción en curso.

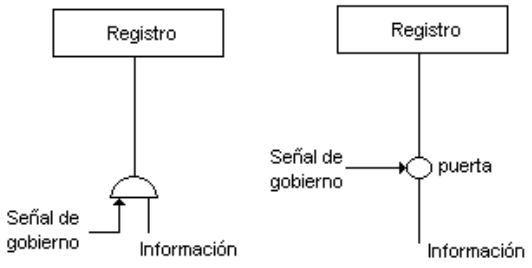
Encaminamiento de las Informaciones

A fin de conocer y comprender el funcionamiento de un computador, vamos a describir la forma en que se ejecutan las transferencias de información en la UCP. Para esto utilizaremos una máquina hipotética que llamaremos Abacus.

Convenciones de Gráfico y Lenguaje

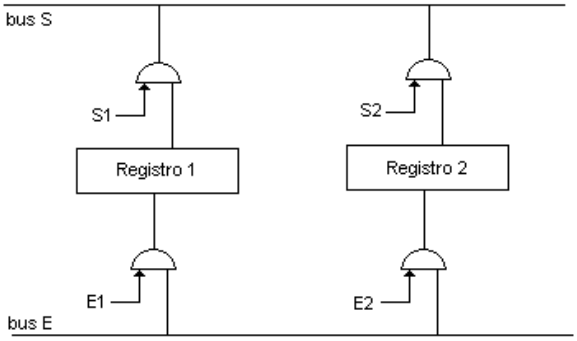
Un registro será esquematizado por un rectángulo. Las entradas y salidas positivas serán simbolizadas por una sola entrada y por una sola salida respectivamente. No se representan las entradas y salidas negativas. Las puertas impulsionales que franquean la entrada a los diferentes

biestables de un registro serán esquematizadas por una sola puerta. Utilizaremos las dos simbolizaciones de la siguiente figura:



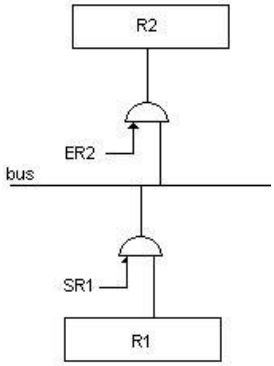
Entrada de información en un registro.

La salida de un registro desembocará generalmente sobre un bus. Dado que varios registros pudieran desembocar sobre el mismo bus, tendrán puertas de salida gobernadas por señales de nivel. Un conjunto de registros podría estar situado entre dos buses como se ilustra en la siguiente figura:



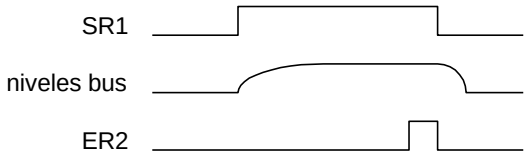
Registros situados entre dos buses

En el caso de una transferencia entre registros, la salida de información de un registro es habilitada por una señal de nivel, una vez estabilizados los niveles en el bus en donde desemboca esta señal, se activa una señal de impulso que habilita la entrada en el registro destino, como se ve en la siguiente figura:

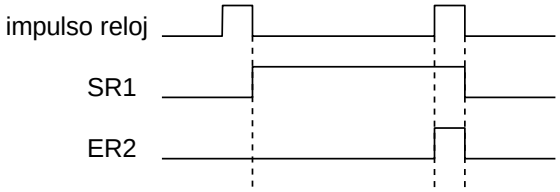


Transferencia de información entre registros

La apertura de la puerta de nivel por la señal SR1 permitirá hacer subir al bus intermediario los niveles de tensión dados por los biestables del registro R1. Una vez estabilizados estos niveles podrá enviarse el impulso ER2, que autorizará la carga en R2 del contenido latente en el bus. Es preciso entonces mantener la señal SR1 durante un intervalo suficiente para que los niveles se estabilicen en el bus intermediario antes de enviar el impulso ER2.



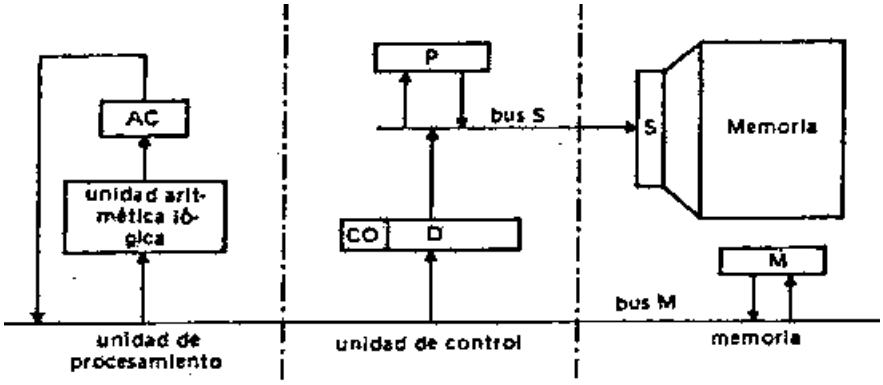
En una máquina, estas señales estarán sincronizadas normalmente con un reloj productor de impulsos periódicos.



Descripción de la Unidad Central de Abacus

Características Generales:

- Instrucción de una sola dirección.
- Transferencias en paralelo.
- Máquina Síncrona (lo que implica que todos los elementos de la máquina están sincronizados con un mismo reloj).
- El reloj producirá un batido cada θ segundos.
- El ciclo de base de la memoria será de 2 θ.
- La longitud de la palabra será de n bits.
- La longitud del Código de Operación será de p bits.
- La longitud de la palabra es suficiente para que los m bits de la parte de dirección permitan direccionar toda la memoria.



Esquema simplificado de Abacus

Abacus es una máquina organizada en torno a dos buses, que son el bus de memoria (bus M) que transfiere las palabras leídas de, o escritas en memoria y el bus de selección (bus S) que transfiere las direcciones.

Resumen de los registros de Abacus:

S:	Registro de Selección de Memoria	*m bits*.
P:	Contador Ordinal o de Programa	*m bits*.
M:	Registro de Intercambio o de Palabra	*n bits.*

I: Registro de instrucción con dos partes:

CO: Código de Operación *p bits*.

D: Dirección *m bits*.

AC: Acumulador *n bits*.

Suma en Abacus

SUM	DIRECCION

Suma el contenido de la célula de memoria designada por la dirección con el contenido del acumulador. Es una operación de tipo 'sum' acompañado de CD (condición de direccionamiento) y la dirección del operando. La suma en Abacus posee 4 faces, en donde en el cronograma de la suma se puede ver como el final se puede encajar con el principio, es como un rodillo.

Fase 1: Búsqueda de la Instrucción.

Suponiendo que la dirección de la instrucción por ejecutar se halle en el contador ordinal, es preciso:

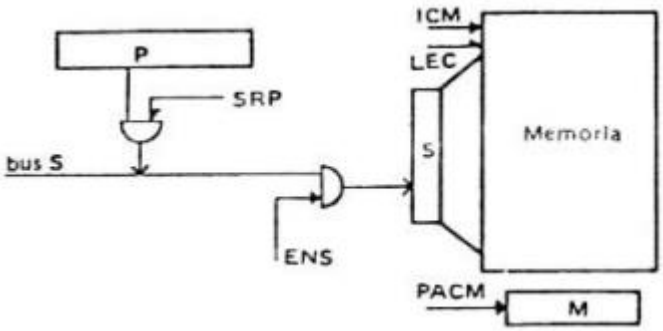
1. Transferir el contenido del Contador Ordinal al Registro de Selección de Memoria (RSM): (P) → S
2. Lanzar un ciclo de memoria, de tal modo que sea transferido el contenido de la célula de memoria direccionada por el registro S al registro de palabra. ((S)) → M.

Desarrollar micro-ordenes (señales de control) para buscar la instrucción. Consiste en mover el contenido de contador de programa hacia registro selección de memoria y lanzar lectura (para leer desde MEMORIA a M la siguiente instrucción, para buscarla!!) Contador de programa guarda la última instrucción incrementada en uno, para buscar la siguiente (está ordenado secuencialmente)

Análisis del Proceso:

(P) → S. Durante el último θ de la instrucción precedente, la señas SRP mantiene abierta la puerta de salida del registro P, por lo que la información de P está latente en el bus S bajo la forma de niveles lógicos. Bastará con validar esta información al primer batido de reloj de la instrucción por medio de la señal de muestreo (de impulso) ENS para introducirla en el registro S.

((S)) → M. Será necesario lanzar un ciclo de memoria mediante la señal impulsional ICM (inicio de ciclo de memoria), indicar que se trata de una lectura, manteniendo por ejemplo la señal LEC durante todo el intervalo del ciclo (y en cualquier caso hasta el inicio de la regeneración) y, si fuera preciso, poner a cero el registro M antes de la lectura de los núcleos (PACM).

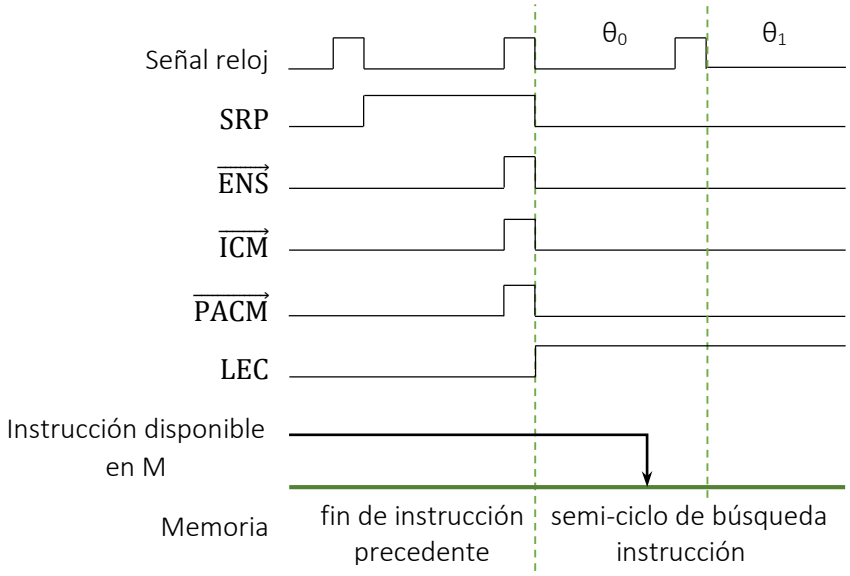


Búsqueda de la instrucción

La fase de búsqueda de la instrucción puede resumirse así:

(P) → S $\quad$ SRP, $\overrightarrow{\text{ENS}}$

((S)) → M $\quad$ $\overrightarrow{\text{ICM}}$, $\overrightarrow{\text{PACM}}$, LEC

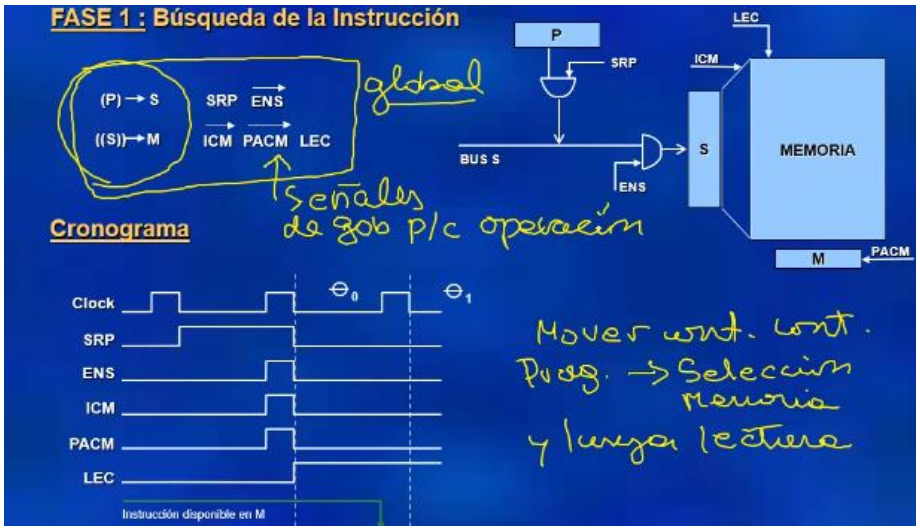


Cronograma de la búsqueda de la instrucción

Para lograr que el contenido de P vaya a S. PRIMERO hay que lanzar una señal de gob (SRP) para que se estabilice sobre el bus. Cuando alcanza el máximo se lanza ENS (señal impulsional), porque en S ya está en contenido de P (La dirección de la sig. instrucción, la cual se va a leer y ejecutar)

Ahora el contenido del contenido de S ((S)) (es decir, ubicación del operando) para a M, para ello se lanza una lectura. Es por ello que se necesita ICM (Inicio ciclo de memoria, señal impulsional. Es como que le dice "ponete disponible bb ;)" preparar la memoria para una lectura o escritura), PACM (Puesta a 0 del registro M. Impulsional. Es necesario antes de que se pase la info porque es por transferencia con previa puesta a 0. ¿Por qué? Pensamos en núcleo magnético, por lo que no se pueden transferir 0; se desmagnetiza la palabra completa.)=> LEC (siempre se necesitan esos dos para la lectura de Abacus).

La sig. instrucción va a decir 'sum dir'.



Fase 2: Búsqueda del Operando.

Debe enviarse la instrucción contenida en el registro M al registro I para proceder a su decodificación: (M) → I, la parte de dirección D de la instrucción, que especifica la dirección del operando ha de ser transferida al registro S: (D) → S, de manera que, tras un semiciclo de lectura, el operando se encuentre en el registro M: ((S)) → M.

Análisis del Proceso:

Búsqueda del operando (hay instrucciones que no requieren búsqueda del operando (salto condicional, almacenamiento; son como de toma de decisión, es un grupo menor). Supone que lo que se leyó ahora tiene que pasar al ámbito de registro de instrucción (de M a D)

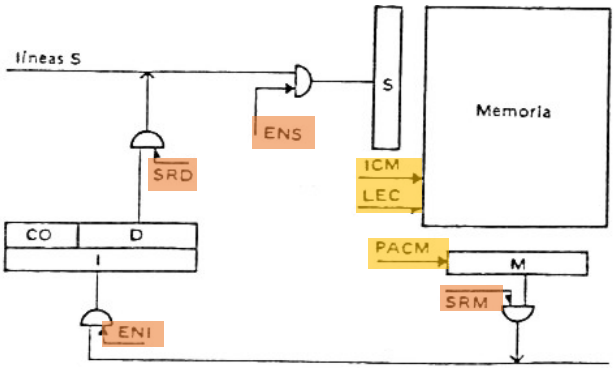
(M) → I. La instrucción se encuentra en el registro M antes del fin del primer θ del ciclo, se abre la puerta de salida del registro M al bus M durante este primer θ, para que, así mantenida la información en el bus M, pueda forzarse su introducción en I gracias a la señal de muestreo ENI, sincronizada con el segundo batido del reloj del ciclo de memoria.

Primero M a I (registro instrucción), para ello primero habilitar salida registro M (SRM) la cual es de nivel. Cuando se llena el bus, lanza un ENI (impulsional) para rescatar la info.

(D) → S. El registro I se divide en dos partes: una parte de control, con el CO e informaciones complementarias, y una parte de dirección. La decodificación de la primera parte indica que esta dirección es la del operando que debe ahora localizarse en la memoria. DECODIFIACION (CO) aquí la AC debe decodificar el código de operación de la instrucción (el tiempo de esto es suuuper corto)

Por consiguiente, es preciso enviar la parte de dirección del registro I al registro S abriendo la puerta de nivel (señal SRD) para mantener la información en el bus S. Al siguiente batido se abrirá la puerta impulsional de entrada en S (señal ENS).

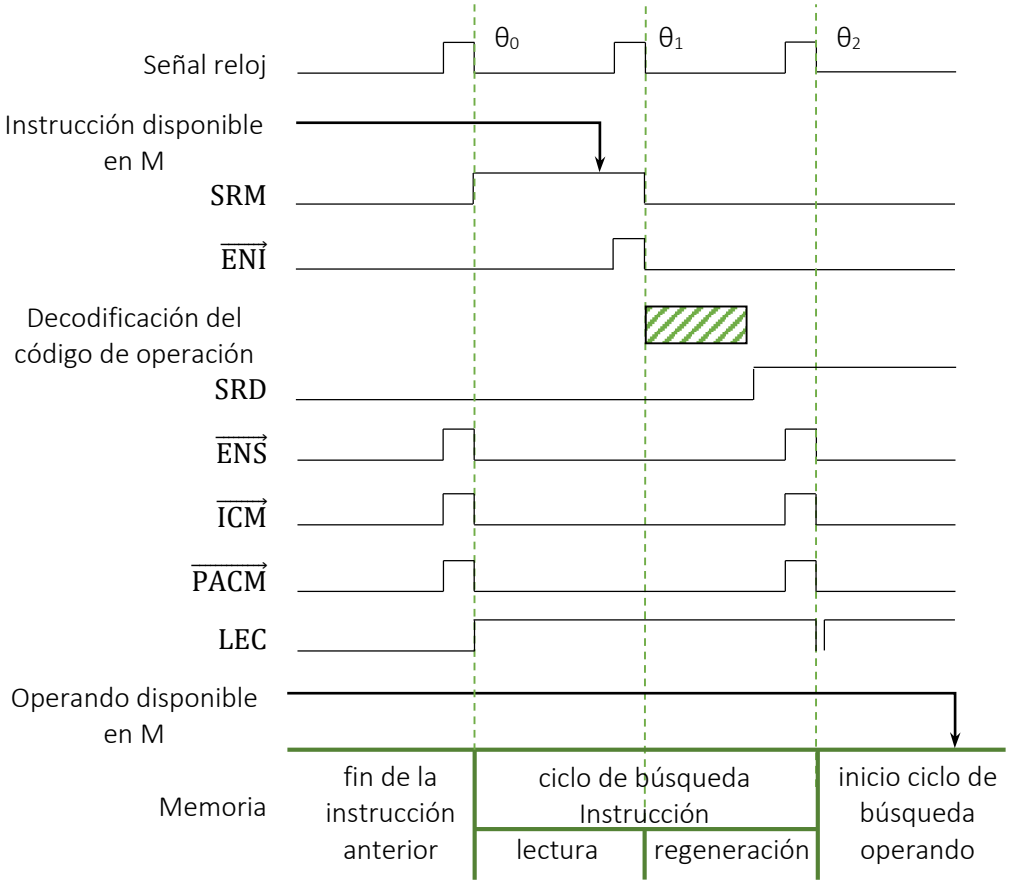
((S)) → M. Se borrará el registro M (señal PACM) simultáneamente con la introducción en S y se lanzará un ciclo de memoria por medio de un impulso ICM, indicando que se trata de una lectura gracias al mantenimiento durante todo el nuevo ciclo del nivel LEC.



Transferencia de la instrucción y búsqueda del operando

La fase de búsqueda del operando puede resumirse así:

(M) → I	$\overrightarrow{\text{SRM}}, \overrightarrow{\text{ENI}}$
(D) → S	$\overrightarrow{\text{SRD}}, \overrightarrow{\text{ENS}}$
((S)) → M	$\overrightarrow{\text{ICM}}, \overrightarrow{\text{PACM}}, \text{LEC}$



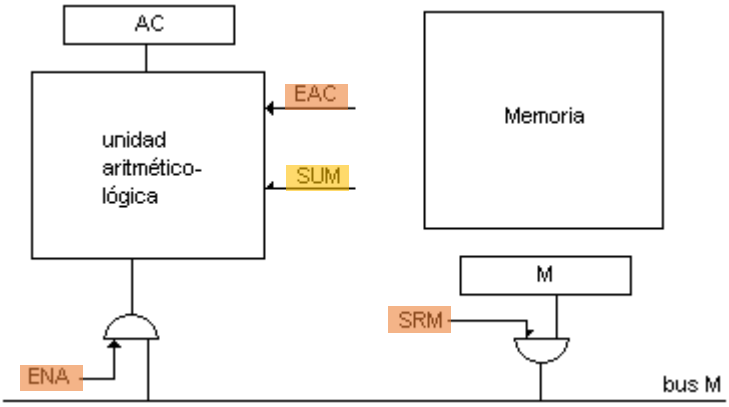
Cronograma de búsqueda del operando

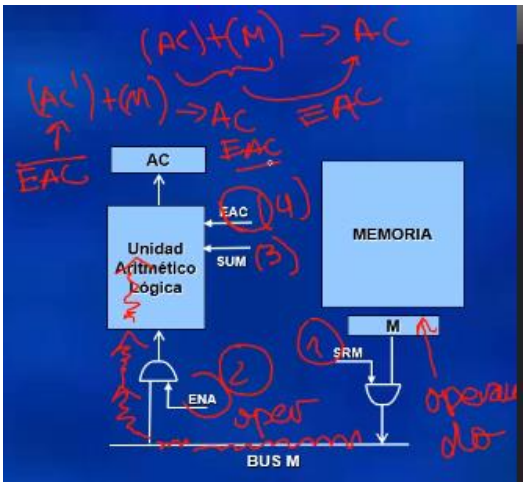
Fase 3: Ejecución de la Suma.

Se trata de sumar el contenido de M con el contenido de AC: (M) + (AC) → AC. Suma entre el contenido de AC y el contenido del contenido de Dir

Los niveles lógicos correspondientes a los registros M y AC durante todo el tiempo de trabajo del sumador, consagrado esencialmente a la propagación del arrastre. Se consigue por las señales SRM, ENA, SUM. Terminada la suma, una señal EAC introduce el resultado en el AC.

SRM, ENA, SUM son todas de dos batidos porque en un solo batido no se puede hacer todo simultáneamente.

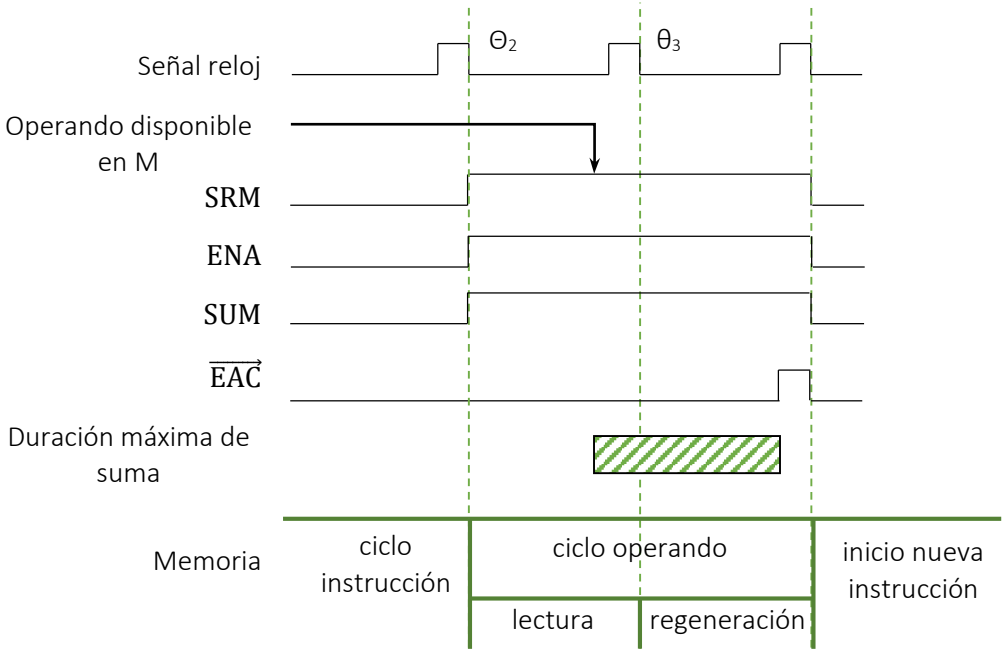




Ejecución de la Suma

La fase de procesamiento del operando puede resumirse así:

(M) + (AC) → AC SRM, ENA, SUM, $\overrightarrow{EAC}$

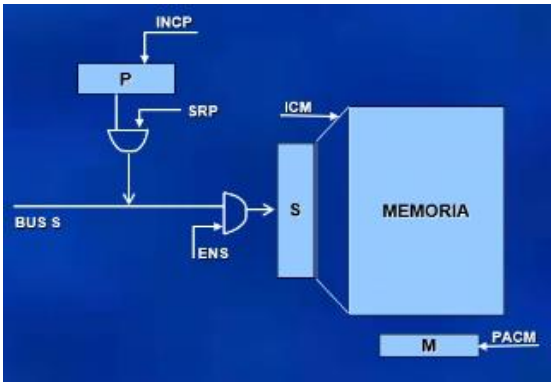


Cronograma de la Ejecución de Suma

Fase 4: Preparación de la próxima instrucción.

Esta fase se dedica esencialmente a incrementar el contador de programa en 1: (P) + 1 → P, enviándole un impulso de cuenta INCP para que al final de la instrucción contenga la dirección de la próxima instrucción. Esta última reanudará nuevamente el proceso: (P) → S, ciclo de búsqueda de la instrucción, etc.

Preparación de la siguiente instrucción. Incrementa Cont de programa y deja su valor en el registro S.

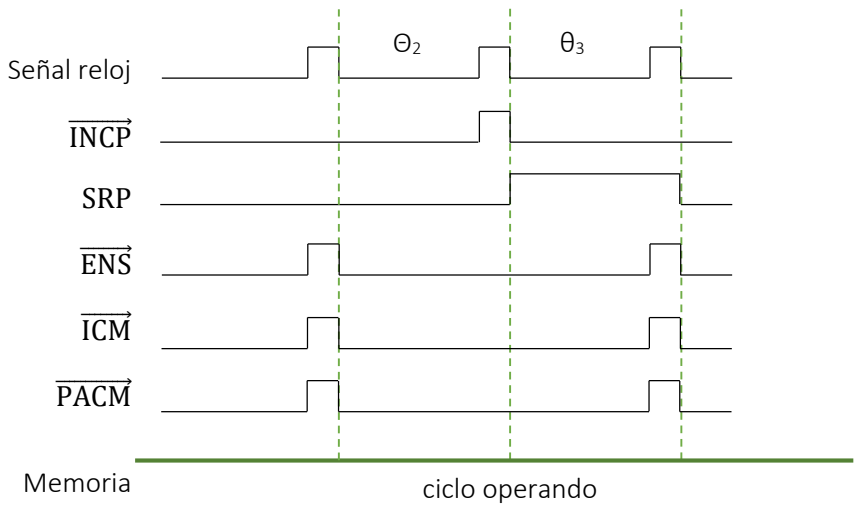


Preparación de la siguiente instrucción

La fase de preparación de la próxima instrucción puede resumirse así:

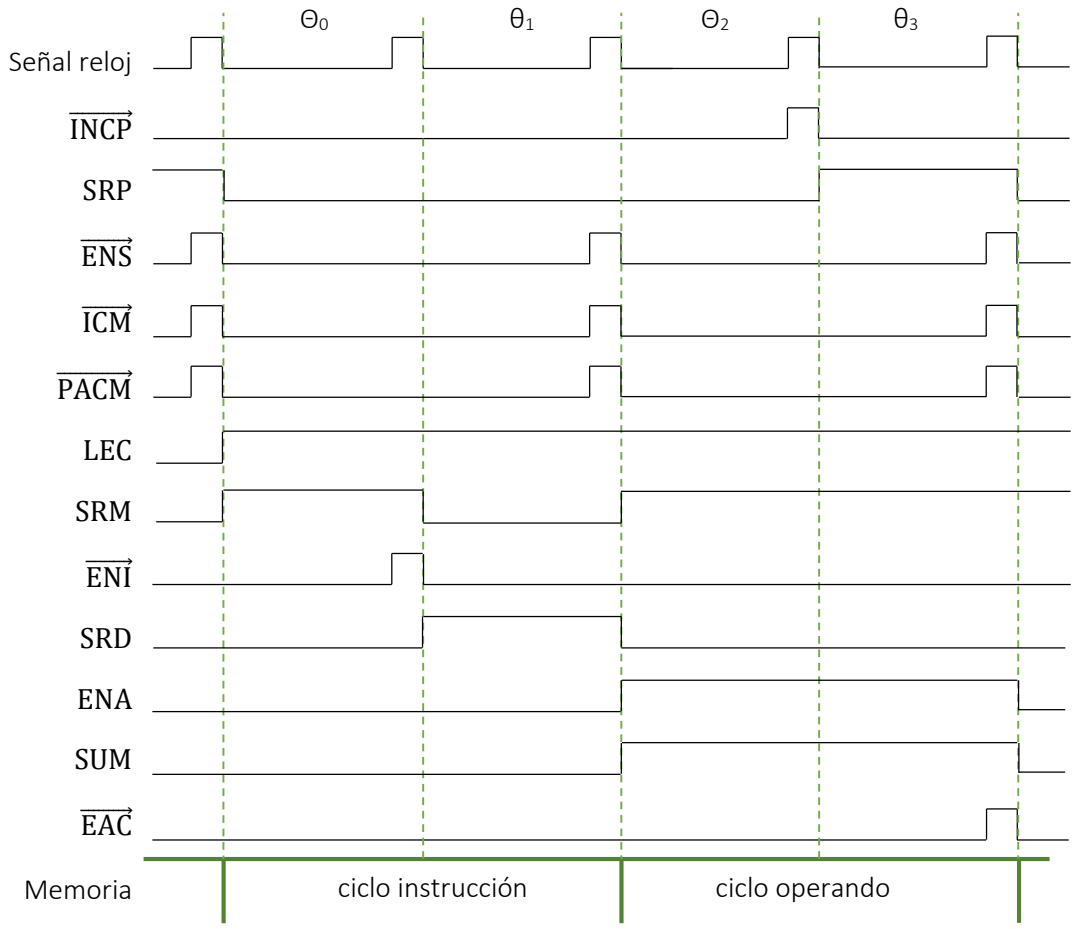
(P) + 1 → P $\overrightarrow{INCP}$

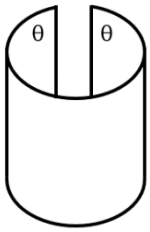
(P) → S SRP, $\overrightarrow{ENS}$



Cronograma de la preparación de la siguiente instrucción

Síntesis de la suma en Abacus:

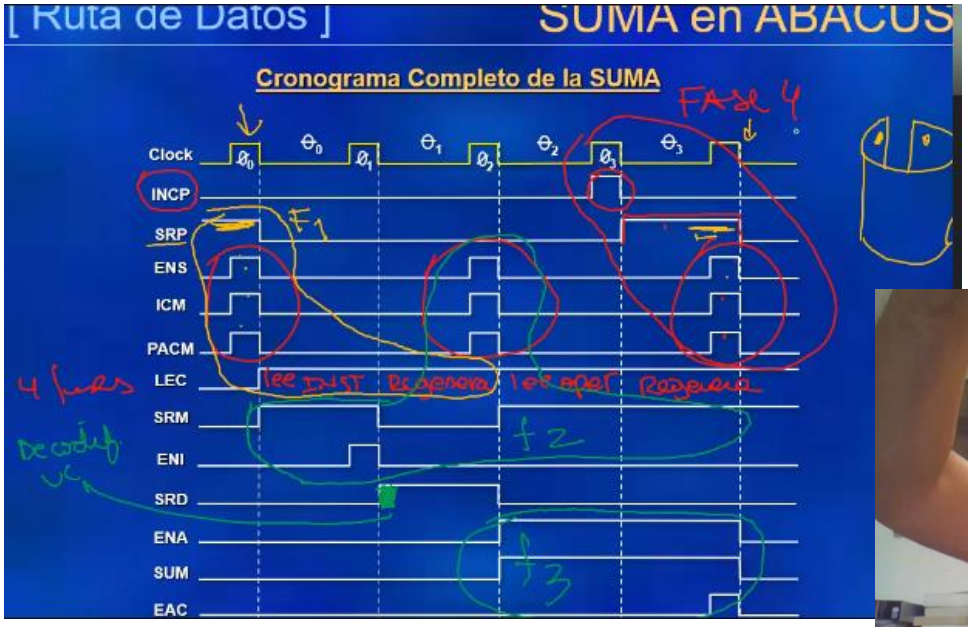




Cronograma de la suma en Abacus

(P) → S	SRP, $\overrightarrow{ENS}$
((S)) → M	$\overrightarrow{ICM}$, $\overrightarrow{PACM}$, LEC
(M) → I	SRM, $\overrightarrow{ENI}$

(D) → S	SRD, $\overrightarrow{\text{ENS}}$
((S)) → M	$\overrightarrow{\text{ICM}}$, $\overrightarrow{\text{PACM}}$, LEC
(M) + (AC) → AC	SRM, ENA, SUM, $\overrightarrow{\text{EAC}}$
(P) + 1 → P	$\overrightarrow{\text{INCP}}$
(P) → S	SRP, $\overrightarrow{\text{ENS}}$



Instrucción de Almacenamiento en Abacus:

ALM	DIRECCION

Almacena el contenido del registro Acumulador en la palabra de memoria designado por la dirección del registro de instrucción. El primer ciclo idéntico al de la suma (búsqueda de la instrucción) mientras que el segundo ciclo se dedica seleccionar la palabra designada por la dirección de la instrucción, para luego escribir en ella le valor del acumulador. En tal sentido el segundo ciclo dedica su tarea estabilizar la información contenida en el acumulador sobre el bus de datos M (señal de gobierno en nivel SAC) para que luego se ha almacenado en la palabra de memoria seleccionado por el campo dirección, a través del registro palabra de memoria M

Mueve el contenido del AC de la ALU a MC según indique la dirección. El primer ciclo es idéntico al de la suma (buscar la instrucción) es por ello que empezamos por tita2 y tita3

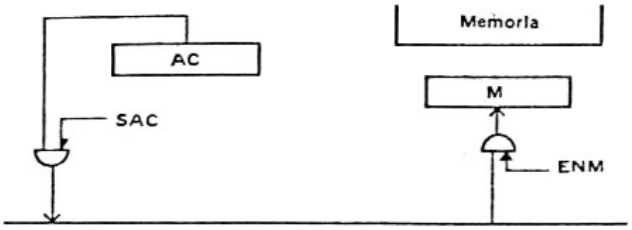
El valor del AC tiene que habilitarse y salir de ese mismo batido al bus M => se lanza la escritura antes para poner a 0 la memoria, es como hace un lugarcito al lugar donde se graba en la memoria. La lectura dura dos batidos y luego la escritura dura dos batidos.

1- la dirección de almacenamiento se posiciona sobre bus S y lanzamos ENS y ICM
2- Se copia (AC) sobre bus M, ingresa a M y luego recién se escribe* (primero PAC y luego escritura)

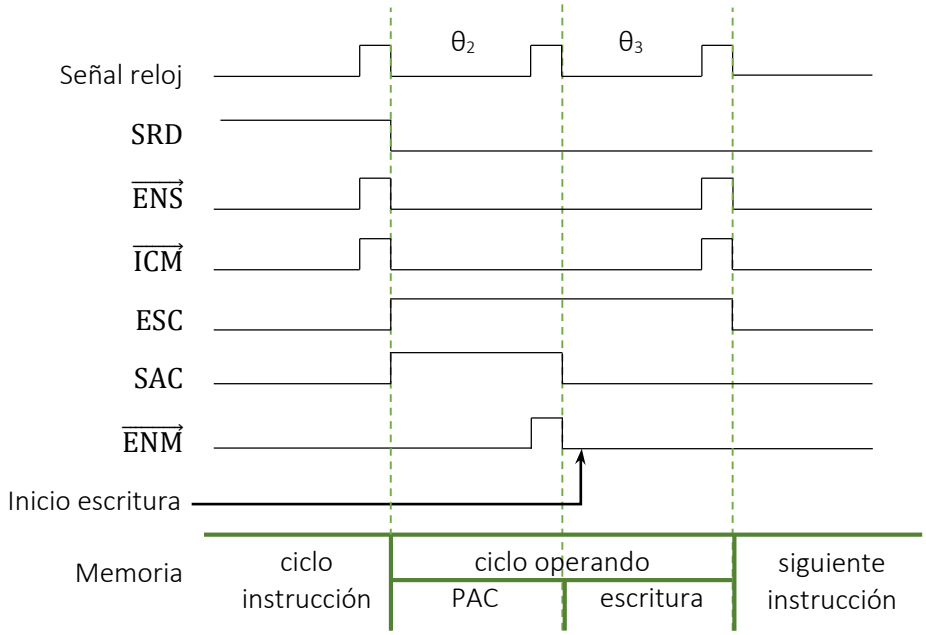
(D) → S SRD, $\overrightarrow{\text{ENS}}$

(M) → (S) $\overrightarrow{\text{ICM}}$, ESC

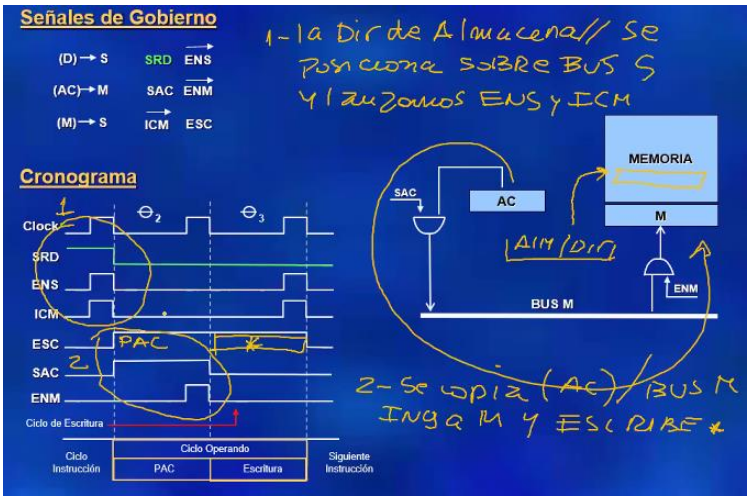
(AC) → M SAC, $\overrightarrow{\text{ENM}}$



Almacenamiento del Acumulador



Cronograma del Almacenamiento



Instrucción de Salto incondicional en Abacus:

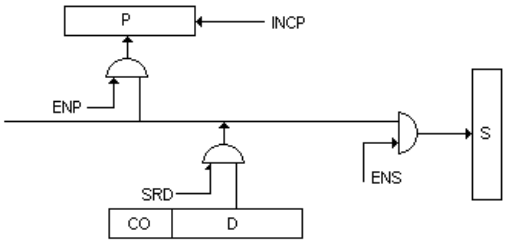
SAI	DIRECCION

Es una estrategia que permite que desde un programa lenguajes de maquina se le permita indicar a través de SAI la dirección a donde deba saltar el programa y efectuar la ruptura de secuencia lineal => se puede lanzar un salto a OTRA parte del programa (es el famoso Go to)

El SAI es una gran necesidad. Consiste en que mueve el contenido de D a S y sobre-escribe el valor de P para que no quede desactualizado. La información de la parte de la Dirección del registro de instrucción, sobre-escribe el valor que lleva el registro Contador de Programa, de manera tal que la siguiente instrucción sea la indicada por la instrucción en curso.

La información de la parte de dirección de la instrucción debe ser transferida simultáneamente a S para buscar la próxima instrucción y a P para actualizar el contador: (D) → S, (D) → P.

Obsérvese que esta instrucción no dura más que un ciclo de memoria y que no hay que incrementar en 1 el contador como ocurre en las otras instrucciones.

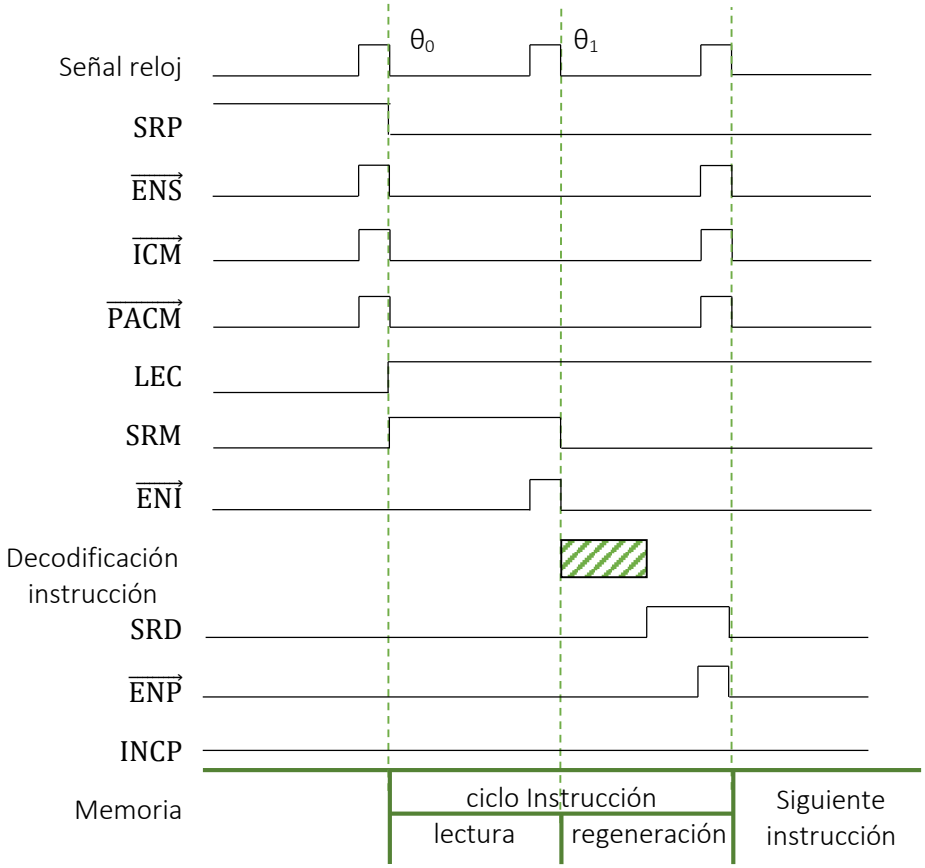


Carga de la dirección de salto

Las primeras tres es la búsqueda de la instrucción ((P) →S, ((S)) →M y (M) →i) y la parte que resta está dedicada a alterar la secuencia del Programa basando el contenido de P en el valor del contenido de D => El contenido de D va a P y a S. No se incrementa P porque ya tiene el valor de SAI

Resumen del desarrollo de la operación de salto incondicional

(P) → S	$\mathrm{SRP}, \overrightarrow{\mathrm{ENS}}$
((S)) → M	$\overrightarrow{\mathrm{ICM}}, \overrightarrow{\mathrm{PACM}}, \mathrm{LEC}$
(M) → I	$\mathrm{SRM}, \overrightarrow{\mathrm{ENI}}$
(D) → P	$\mathrm{SRD}, \overrightarrow{\mathrm{ENS}}$
(D) → S	$\mathrm{SRD}, \overrightarrow{\mathrm{ENP}}, \overrightarrow{\mathrm{ENS}}$



Cronograma del Salto incondicional

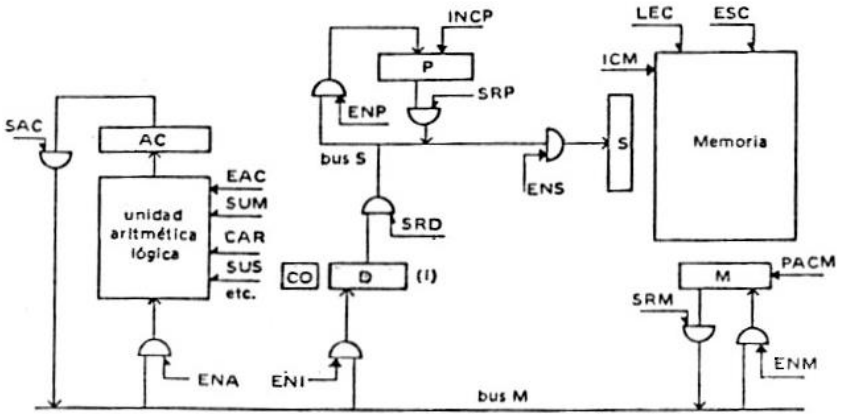
Señales de Gobierno de Abacus

Son todas las señales de impulsos o de nivel empleadas para gobernar las distintas operaciones en el desarrollo de una instrucción. Las clasificaremos en tres categorías

Señales de transferencia:			
ENS	Impulso	(bus S)	→ S
PACM	Impulso	0	→ M
SRM	Nivel	(M)	→ bus M
ENM	Impulso	(bus M)	→ M
ENI	Impulso	(bus M)	→ I
ENA	Nivel	(bus M)	→ UAL
SAC	Nivel	(AC) bus	→ M
ENP	Impulso	(bus S)	→ P
SRD	Nivel	(D) bus	→ S
SRP	Nivel	(P) bus	→ S
INCP	Impulso	(P) + 1	→ P

Señales de gobierno de la memoria:		
ICM	Impulso	Inicio ciclo memoria
LEC	Nivel	Ciclo de lectura
ESC	Nivel	Ciclo de escritura

Señales de gobierno de la unidad aritmético-lógica:		
CAR	Nivel	Carga acumulador
SUM	Nivel	Suma
SUS	Nivel	Sustracción
Etc	. . .	. . .
EAC	Impulso	Introducción del resultado en el AC



Esquema de Abacus

A fin de presentar esta "Unidad" que reúne los elementos anteriormente descriptos, recurrimos a la definición de "Abacus". Máquina hipotética descripta por Meinadier y al concepto de "Ruta de Datos" del mismo autor.

Organización y Componentes de la Ruta de Datos

Ruta de Datos

Conjunto de Órganos para Memorizar, Transferir, o Procesar información proveniente de la MC en forma de instrucciones, direcciones, y operandos. Cada Órgano cuenta con Registros de transferencia que le permiten comunicarse con el resto, a través de los Buses definidos en la Arquitectura. La generación de las señales de gobierno que coordinen el movimiento de instrucciones, direcciones, y operandos, está a cargo de un Órgano Central conocido como Unidad de Control.

- Buses de Interconexión: Para la transferencia de información.
- Registros y memorias: Para su almacenamiento.
- Unidades funcionales: Para su procesamiento.

Las señales de gobierno están distribuidas por toda la ruta autorizando las diferentes operaciones. Se las llama micro-órdenes y son generadas por el secuenciador.

Por una parte haremos la distinción entre unidad de instrucciones y unidad de procesamiento. Esta distinción se basa en la diferencia entre dos tipos de información: Instrucciones y direcciones por un lado, operandos por el otro. Del mismo modo haremos la distinción entre ruta de datos y distribución de micro-órdenes. Esta distinción también se basa en la diferencia entre dos tipos de información: las **informaciones conocidas del programador** (instrucciones, direcciones y operandos) por una parte, **y las informaciones desconocidas del programador**; que son las micro-órdenes. Los puntos de contacto entre ruta de datos y distribuidor de micro-órdenes se sitúan a dos niveles:

1) El registro de instrucción, puesto que las micro-órdenes por distribuir dependen del código de operación;
2) El impacto de las micro-órdenes sobre los diversos componentes de la ruta de datos (puede considerarse a la MC parte de la ruta de datos).

Dimensión de la ruta de datos

Es el número de dígitos transmitidos en PARALELO por la ruta de datos. Esta puede variar a lo largo de una misma ruta. Puede ser fija en lo que refiere al tamaño del registro de Instrucción y de Operando y un número más restringido para las direcciones de los operandos. En Arquitecturas más complejas la dimensión puede ser variables, de uno a más octetos para determinados órganos que requieren procesar información aritmética de simple/doble precisión por ejemplo.

Formato de las Instrucciones

Sumar el contenido de la palabra de memoria designada por la dirección, con el contenido del registro Acumulador. Es la operación típica del grupo de instrucciones aritmético / lógicas correspondientes a la familia de instrucciones IBO (Instrucciones con Búsqueda de Operando). Requieren para su ejecución el desarrollo de cuatro fases.

Registros de la Ruta de Datos y su Interconexión

Plano Funcional:

Es como los registros dan soporte a la Arq. desde el punto de vista de sus funciones.

- **Registros no direccionables**: Poseen una función bien definida y exclusiva. Algunos de estos registros pueden permanecer desconocidos del programador. Se dirá que le son transparentes. Otros son implícitamente diseccionados por el programador, como es el caso del AC de Abacus en cuanto a todas las instrucciones aritméticas y lógicas.

Ejemplo: RI, AC y Multiplicador Cociente, Contador de Programa, R. de Selección del Memoria., Contador de Desplazamientos.

Un registro para una función → son dentro de Abacus el registro AC, el red contador de programa (P) el registro palabra de memoria, el registro contador de desplazamiento => Son registros que ponen la ruta de datos pero no son visibles al programar, son como transparentes; no se pueden alterar.

Varios registros para una función → Registros banalizados porque la gran mayoría de la Arq. puede manejarlos para uno u otra cosa, no están dedicados a una cosa específica.

- **Registros direccionables o usables**: Son los que el programador puede emplear para conservar determinadas informaciones en el transcurso del cálculo.

Le quitan a la MC la tarea de estar dedicada a poder prever la información de los registros direccionales. El programador puede acceder, pero poseen una dirección fija y no están dentro de la memoria central (en un principio) de manera, tal como estén implementados pueden ser rápidamente accesibles dando solo el nombre del registro y que este nombre esté relacionado con el nombre del registro => la programación en mucho más óptima.

El programador puedo usarlos para conservar determinada información en el transcurso => disminuye el cálculo y el costo de la memoria central. Son de tecnología integrada con semiconductores, tiene una alta velocidad con direccionamiento directo; la ganancia es mucho más superior que utilizar la memoria RAM.

Entre los distintos registros direccionables se aprecian tres categorías:

→ *Registros aritméticos*: Para conservar resultados parciales.

→ *Registros de direccionamiento*: Para conservar elementos fijos de direccionamiento de partes de programas o de vectores.

→ *Registros Banalizados:* Un registro direccionable que puede usarse indistintamente como registro aritmético o de direccionamiento.

Plano Estructural:

De qué manera física está implementados.

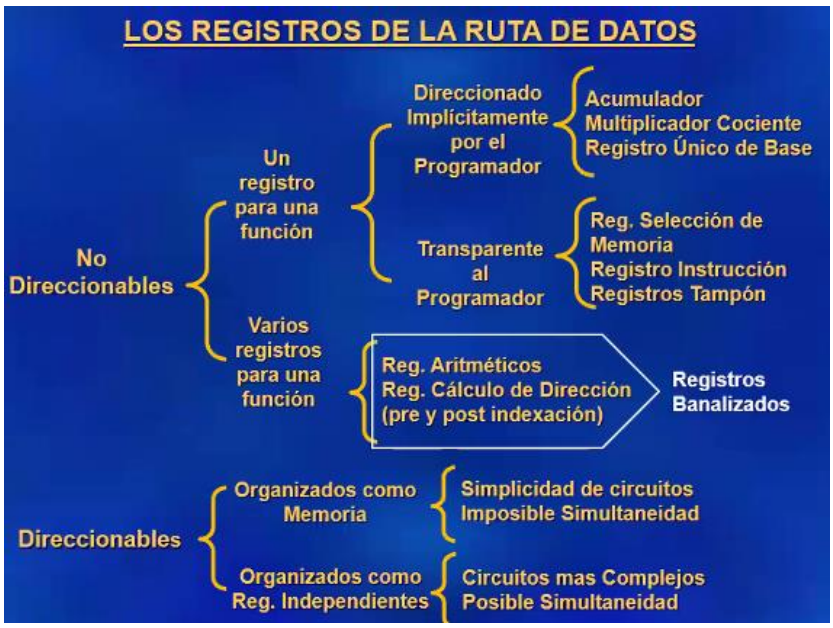
- **Registros independientes:** La dirección de los registros necesita ser decodificada para obtener las señales de apertura o de clausura de las puertas de los registros.

Están físicamente instalados dentro de la Arq. cableados de manera que se puede acceder al registro instrucción => no hay que esperar para interactuar PERO es muy costoso por la cantidad de E/S.

- **Memorias locales:** La dirección del registro le será dada a la memoria sin decodificación previa. En comparación con un conjunto de registros independientes presentan una ventaja económica (menos puntos de entrada y de salida, un solo Circuito Integrado) pero, como contrapartida, no autorizan simultáneamente más que una sola operación de lectura o escritura (es decir, no pueden operar con varios simultáneamente).

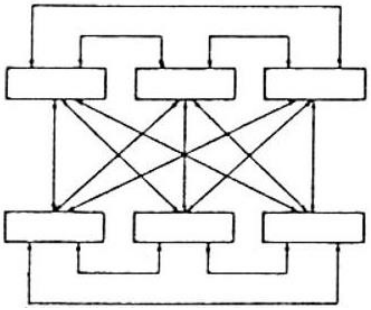
Una pequeña memoria local en un circuito integrado operan como si fuese un registro => no encarece en cantidad de E/S PERO no permite que se pueda acceder de manera simultánea a más de uno.

- **Seudo-Registros:** Son registros direccionables que carecen de existencia física dentro de la ruta de datos. Son simulados en MC o en M. Local.



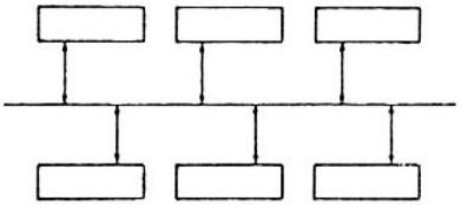
Tipos de Interconexiones

Las interconexiones entre registros pueden ser directas o a través de buses. La interconexión por bus tiene, frente a las conexiones directas, la ventaja de ser barata, pero no permite transferencias simultáneas.



Interconexión Directa de 6 registros 2 a 2

Directa de 6 reg 2 a 2 (es CABLEADA) es costosa pero es versátil y logra paralelismo en Esc Y lec.



Interconexión por bus de 6 registros

Por Bus de 6 reg. Si el bus está ocupado, no permite hacer otra transferencia => NO permite transferencias simultaneas PERO es muy económico.

Secuenciamiento de las instrucciones

Es generado por el **Secuenciador** → es un órgano complejo que sufrió muchos mejoras en el tiempo y cambios de paradigmas. Este se encarga de **generar las micro-ordenes necesarias para ejecutar una instrucción de leguaje de máquina, las cuales a través de la ruta de datos genera las habilitaciones** distribuidas de manera racional y eficiente a lo largo de la ruta de datos de dicha Arquitectura (que son

distribuidas a lo largo de la ruta de datos para activarla y gobernar sus diversos elementos constituyentes). **Es como el cerebro del computador!** El corazón seria aquellos que sincronizan al computador.

Racional→ para no consumir tanto ancho de banda en los buses

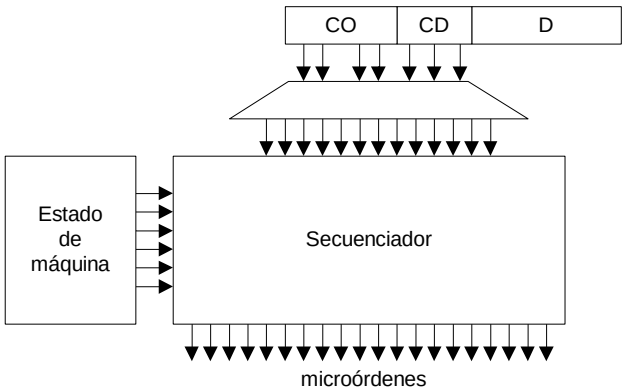
Eficiente→ para no exceder en micro-órdenes para una misma acción, la misma cantidad de excedente consume muchos ciclos extras.

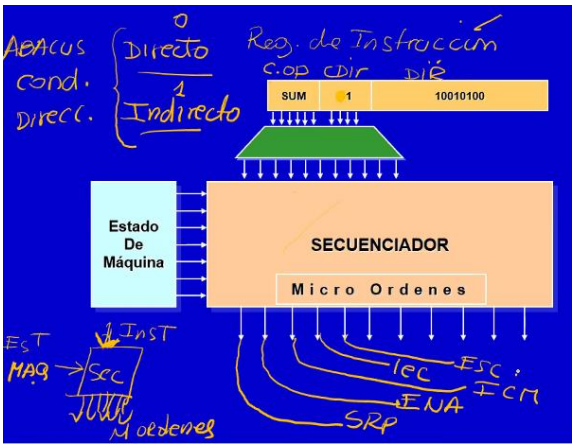
Tanenbaum plantea la estructura de las computadoras con una estructura de capas.

El secuenciador se encuentra entre la Microprogramación y la Lógica Digital, ya que Abacus no tiene la capacidad de Microprogramación, por ello decimos que es Secuenciador en lugar de eso.

Analizaremos el secuenciador, el cual puede ser cableado (para Abacus) o micro-programado. Genera señales impulsionales y de nivel con el objeto de manejar el movimiento de la información. Toma como información la instrucción y genera las señales para que se resuelvan ordenadamente la instrucción mandada. Lee la instrucción y decodifica la instrucción para generar el conjunto de micro-ordenes (señal de gobierno impulsional o de nivel que viaja en la ruta de datos)

Las Informaciones de salida del secuenciador son las micro-órdenes que deben distribuirse por la ruta de datos según cronogramas precisos a causa de los tiempos de respuesta de los órganos gobernados. Las **informaciones de entrada** al secuenciador son suministradas, de una parte, por la instrucción (código de operación, condiciones de direccionamiento y las direcciones de registros), de otra parte, por el estado de la máquina, que agrupa un cierto número de informaciones (indicadores de error, demandas de interrupción, indicadores del estado de la memoria y de las unidades funcionales, etc.)





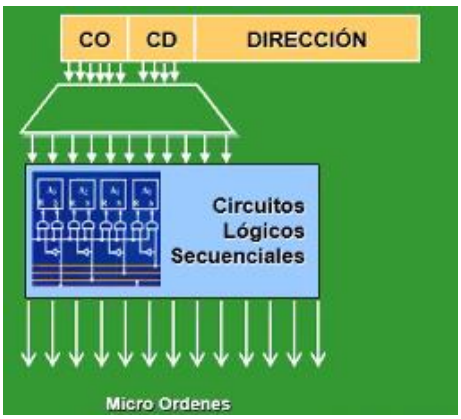
Entorno de un secuenciador

Dado un código de operación valido el secuenciador activa uno de sus circuitos dedicados a generar micro-ordenes según cronograma (de la instrucción en curso)

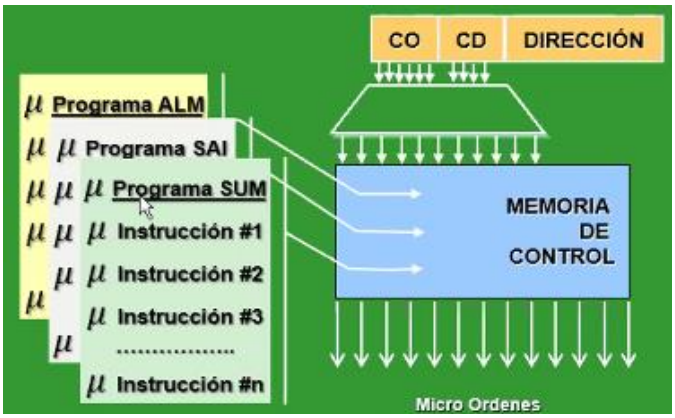
Modelos de Secuenciador:

Ambos modelos tienen por objeto generar el conjunto de micro-órdenes (señales impulsionales y de nivel) y distribuirlas a lo largo de la ruta de datos con el fin de gobernar distintos órganos y componentes y realizar la instrucción en curso, basándose en el conjunto de señales de gobierno diseñadas oportunamente en el cronograma de cada operación

Cableado→ es mucho más rápido que el micro-programado, pero es muy caro y es mucho más grande. Realizados en forma de circuito secuencial electrónico.



Micro-programado→ posee una Memoria de control, son memorias de nivel 0 y se puede re-programar con el fin de mejorar o corregirla; entonces permite que le programador se aleje del hardware. Corresponden a la introducción en la UC de una memoria con un pequeño programa llamado microprograma para cada instrucción.



Secuenciador cableado

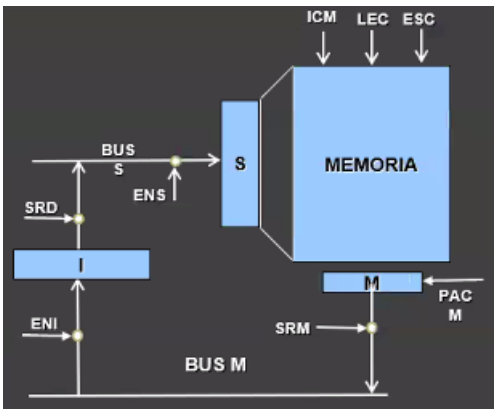
El secuenciamiento de las micro-órdenes en el tiempo puede llevarse a cabo de dos maneras distintas:

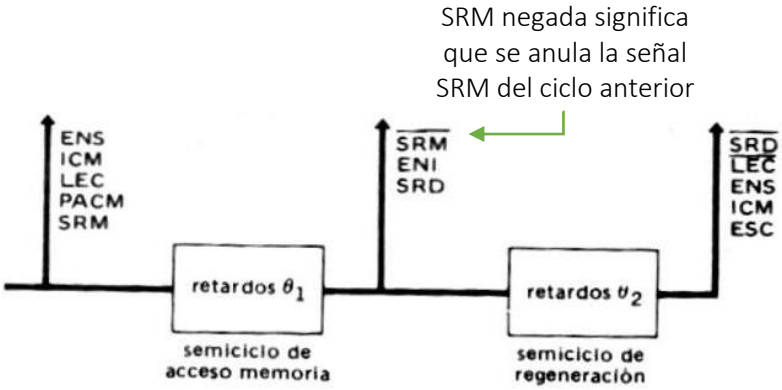
- **Ordenador Síncrono:** el secuenciador conoce los tiempos de respuesta de los diferentes órganos gobernados. Entonces hace intervenir los retardos apropiados entre las diferentes órdenes sucesivas de manera que tengan tiempo de ejecutarse las operaciones. Genera las m-ordenes distanciadas en el tiempo. Son retardos fijos.

Almacenamiento → genera micro-ordenes que distanciadas en el tiempo con retardos distanciados => nos ayuda a realizar la función. Indican que señales de nivel deber permanecer activas y cuales deben anularse.

Principio del Secuenciamiento:

El Objetivo es generar micro-órdenes distanciadas en el tiempo por retardos apropiados. Una cadena de retardos permite, en el plano teórico, realizar esta función. Las señales de nivel permanecen activas hasta su anulación. Por ejemplo, el inicio de la instrucción de almacenamiento en Abacus, suponiendo que la dirección de la instrucción se encuentre sobre el bus S será efectuado por la cadena de retardos, muy esquemáticamente indicada a continuación:





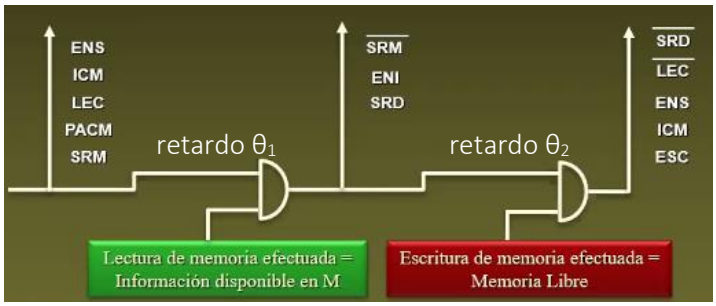
Búsqueda en memoria (solución síncrona)

Ordenador Asíncrono: el secuenciador recibe de los diferentes órganos del calculador señales indicando que han terminado las operaciones indicadas y que se liberan, aquel puede eventualmente comprobar su estado. No lanzará una nueva operación más que después de haber quedado advertido de que las operaciones precedentes han sido ejecutadas completamente y de haberse asegurado que el órgano que debe realizarla está libre.

La memoria asincrónica gobierna cuando se puede pasar de una fase a otra => es decir, cuando puede lazar un conjunto de instrucción o cunado puede anularlos.

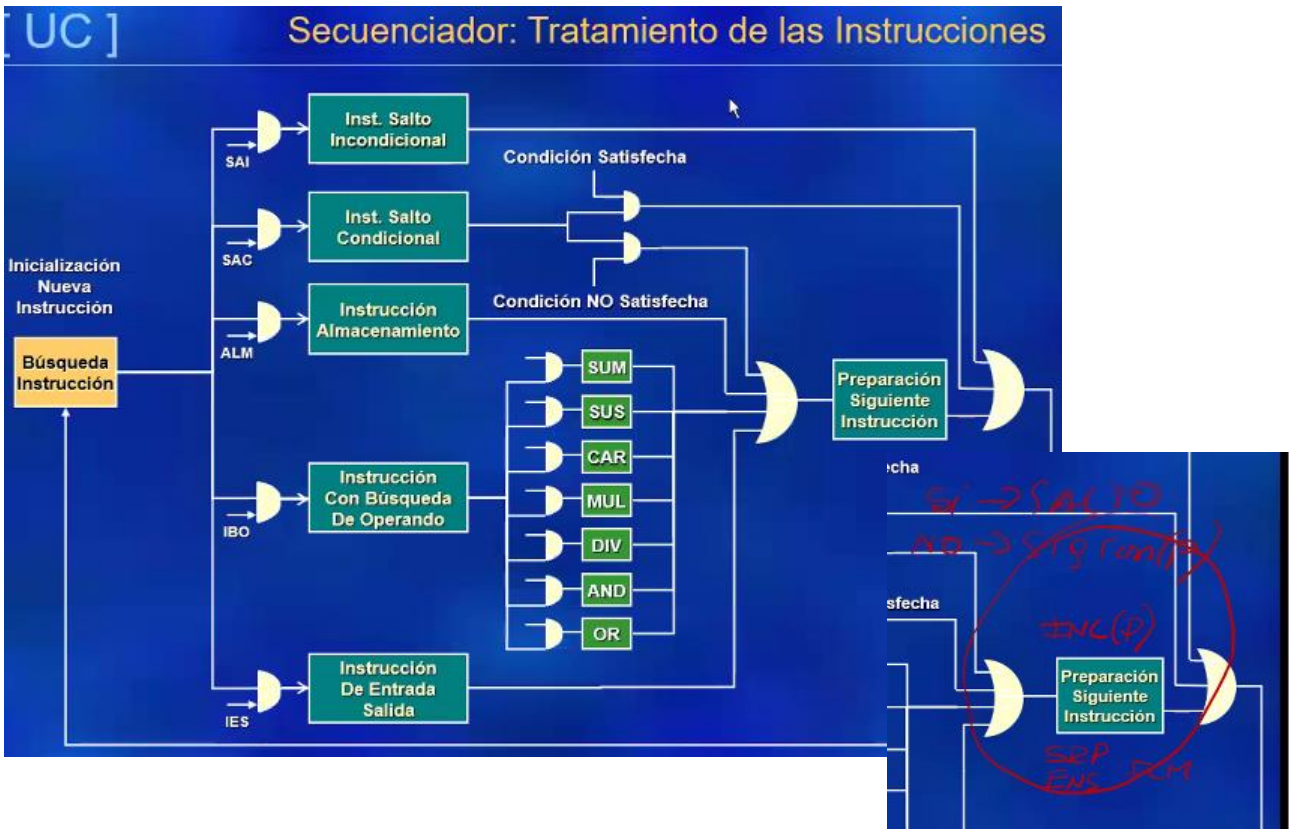
Los tiempos no son fijos, están definidos por el tiempo que le lleva a la memoria RAM el completar un Escritura o lectura.

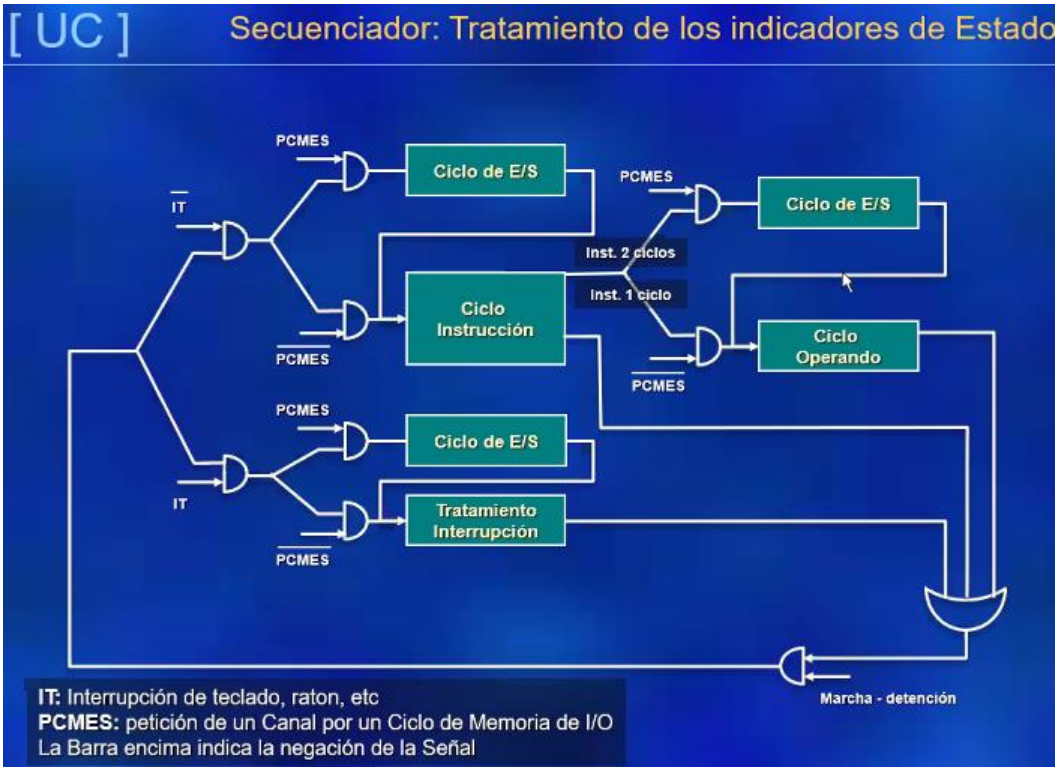
Las compuertas AND de la figura representan simbólicamente que solo se pasan a las siguientes micro-operaciones siempre que el conjunto anterior haya finalizado.

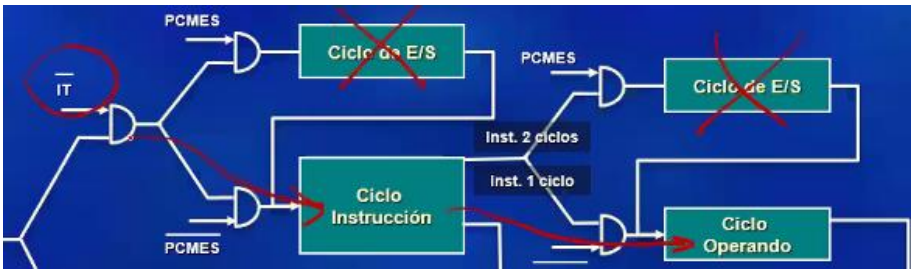


Tratamiento de las instrucciones

Tema de final ahre


[UC]
Secuenciador: Tratamiento de las Instrucciones
Inicialización Nueva Instrucción
Búsqueda Instrucción
SAI
Inst. Salto Incondicional
SAC
Inst. Salto Condicional
ALM
Instrucción Almacenamiento
IBO
Instrucción Con Búsqueda De Operando
IES
Instrucción De Entrada Salida
Condición Satisfecha
Condición NO Satisfecha
SUM
SUS
CAR
MUL
DIV
AND
OR
Preparación Siguiente Instrucción
Preparación Siguiente Instrucción

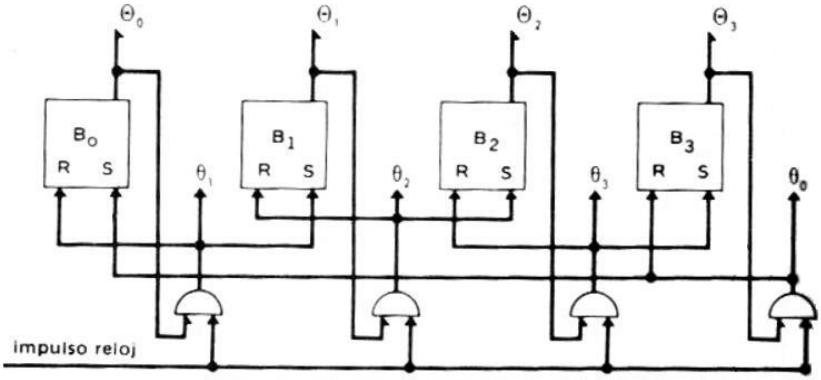

[UC]
Secuenciador: Tratamiento de los indicadores de Estado
PCMES
IT
Ciclo de E/S
Ciclo Instrucción
Inst. 2 ciclos
Inst. 1 ciclo
Ciclo Operando
Tratamiento Interrupción
Marcha - detención
IT: Interrupción de teclado, raton, etc
PCMES: petición de un Canal por un Ciclo de Memoria de I/O
La Barra encima indica la negación de la Señal


PCMES
IT
Inst. 2 ciclos
Inst. 1 ciclo
Ciclo Instrucción
Ciclo Operando

El Distribuidor de Fases

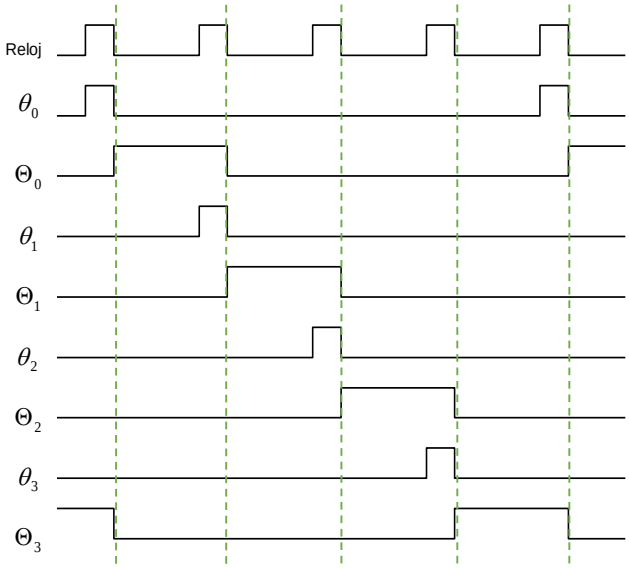
En la mayoría de los casos se sincronizan las distintas micro-órdenes con señales temporales regularmente espaciadas, provenientes de un mismo reloj. A cada impulso de reloj debe ser generado un cierto número de micro-órdenes.

Al ser idénticas todas las señales de reloj, es preciso dar al secuenciador los medios para situarse por referencia al tiempo, por ejemplo, para saber en qué fase del ciclo de la instrucción se encuentra; en ésta la función del distribuidor de fases, que partiendo de los impulsos periódicos del reloj, produce señales de impulso o de nivel acordes a las diferentes fases del ciclo de memoria o de la instrucción.



Distribuidor de fases

En un instante determinado, solamente uno de los biestables B0, B1, B2, B3 está posicionado a 1. El próximo impulso de reloj devolverá este biestable a cero y posicionará el siguiente biestable a 1. Los niveles lógicos Θ0, Θ1, Θ2, Θ3, serán ciertos uno a continuación del otro, pero solo uno en un momento dado. Los impulsos θ0, θ1, θ2, θ3, sincronizados con las señales de reloj, se producirán cada cuatro batidos. Suponiendo que el tiempo de basculamiento de los biestables sea igual a la anchura del impulso de reloj, se obtiene el cronograma de base siguiente:



Cronograma de un distribuidor de fases

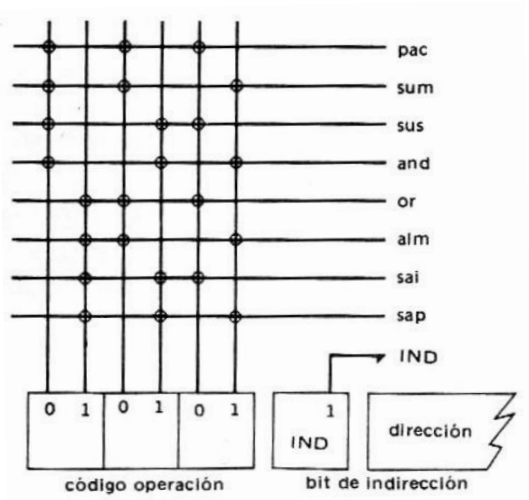
En el caso de Abacus, en el que el ciclo de memoria equivale a dos batidas de reloj, resultará cómodo emplear un distribuidor de dos fases (respectivamente lectura y escritura).

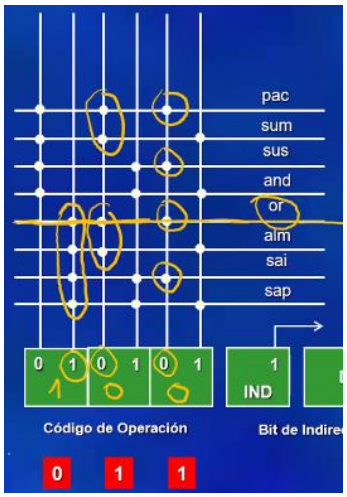
Decodificación de la Instrucción

La instrucción consta de diferentes campos de información de interés para el secuenciador, como son el código de operación, las condiciones de direccionamiento, la dirección de registros o de unidades periféricas, etc. El decodificador de instrucción se encarga de realizar esta decodificación.

Puede utilizarse una matriz de diodos o de transistores. Por ejemplo, para Abacus, dotado de las operaciones indicadas en la siguiente tabla y capaz de direccionamiento indirecto, se necesitaría el decodificador de la siguiente figura:

Nombre Nemotécnico	Código Operación	Efecto de la instrucción
PAC	000	Puesta a cero del acumulador
SUM	001	Suma con el acumulador
SUS	010	Sustracción del acumulador
AND	011	Intersección con el acumulador
OR	100	Reunión con el acumulador
ALM	101	Almacenamiento del acumulador
SAI	110	Salto incondicional
SAP	111	Salto si (AC)>=0





Decodificador del código de operación de Abacus

Se observa en este esquema que la decodificación puede ser considerada prácticamente instantánea y las señales salidas de la matriz permanecen posicionadas todo el tiempo que la instrucción correspondiente se encuentra en el registro de instrucción.

Biestables de Estado

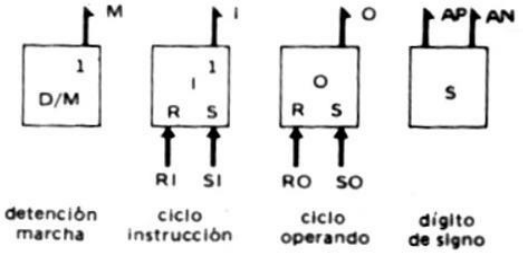
Se define al estado del calculador por el contenido de un cierto número de biestables, llamados biestables de estado. Los hay de dos clases:

Los primeros memorizan lo que hemos denominado el estado de la máquina. Son posicionados por las instrucciones y los resultados de operaciones, o también por acontecimientos exteriores, petición de ciclo de memoria, interrupción, etc.

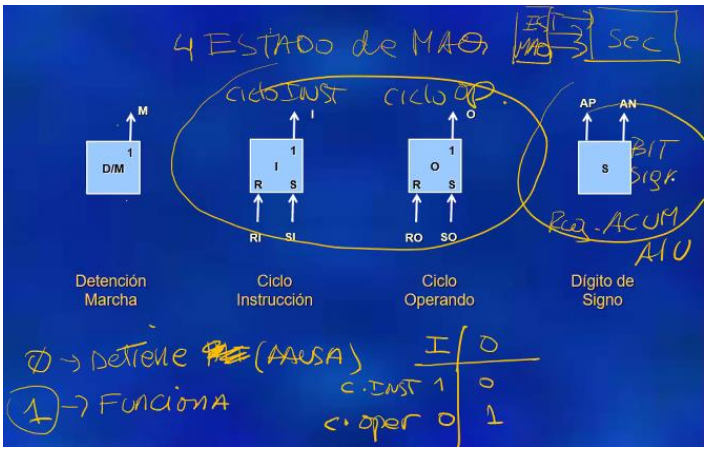
Sus salidas figuran entre las informaciones de entrada al secuenciador. Los segundos constituyen parte integrante del secuenciador.

Es éste quien los posiciona y quien utiliza su salida. Esencialmente sirven de marcas temporales. En cierto modo, los biestables del distribuidor de fases podrían incluirse dentro de esta categoría.

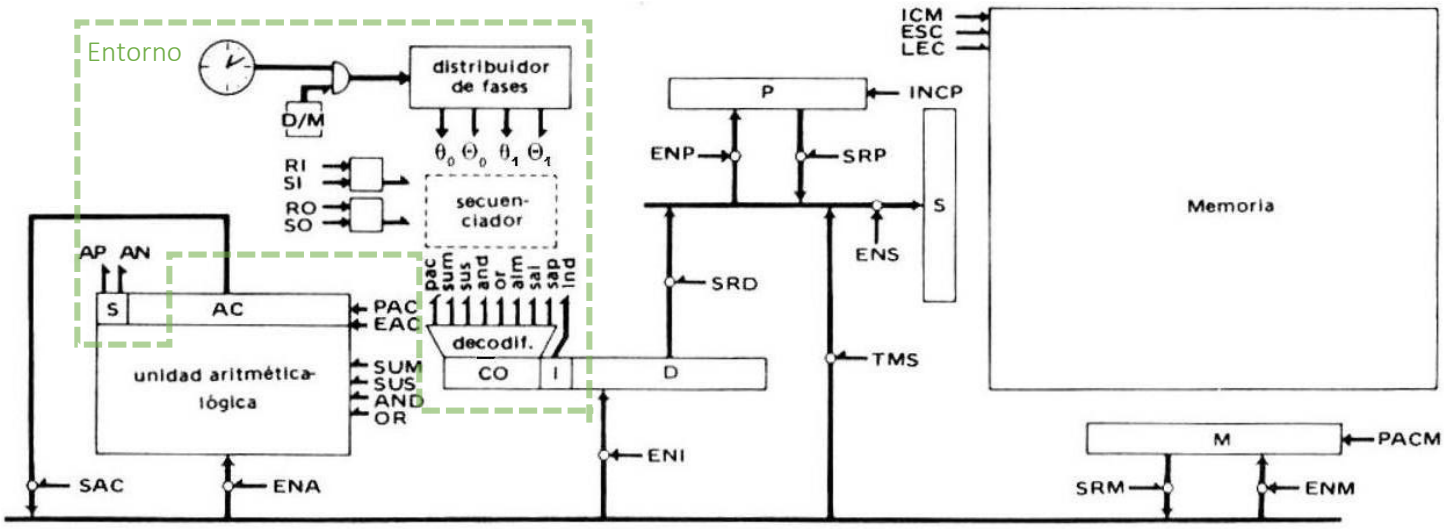
Abacus consta de los cuatro biestables representados en la siguiente figura:



Biestables de estado de Abacus



Esquema General de Abacus



Señales de Control o Microórdenes

Para que la unidad de control realice su tarea debe tener entradas que le permitan determinar el estado del sistema y salidas que le permitan controlar el comportamiento del mismo.

Las entradas al secuenciador serán:

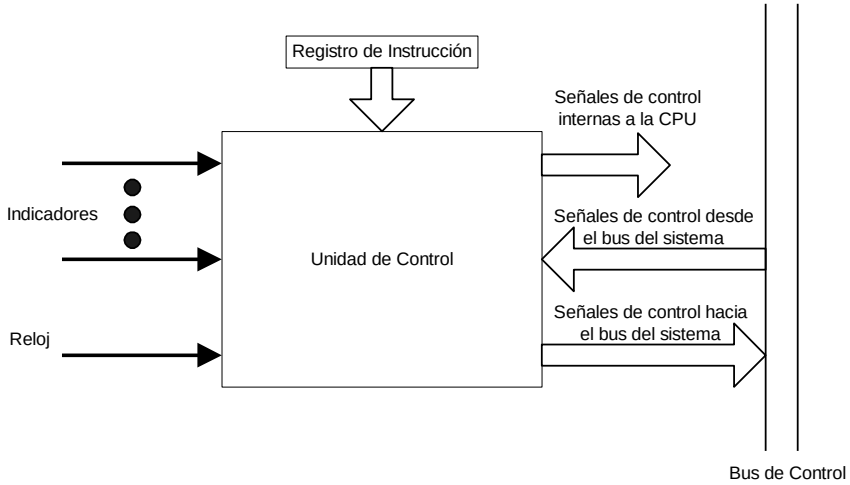
- Las señales del distribuidor de fases.
- Señal de los biestables que indican el subciclo que se está ejecutando (I-subciclo de búsqueda de la instrucción, O-subciclo de búsqueda del operando)

- Señal del biestable S que indica el signo del acumulador
- Código de operación y condiciones de direccionamiento del reg. de instrucción.
- Señales de control de bus de control, incluye interrupciones, de reconocimiento, etc. (no se contempla en el análisis a fin de facilitar el estudio del funcionamiento)

Las salidas del secuenciador serán:

- Señales de control que gobiernan la transferencia de datos, funciones específicas de la ALU, control de memoria y control de los módulos de entrada/salida.

Resumiendo, se podría decir que el objetivo del secuenciador es general las señales de control o micro-órdenes para gobernar el funcionamiento del computador. En la siguiente figura se observa un esquema general de lo descrito anteriormente.



Modelo de la Unidad de Control

Generación de las Señales de Control

Para lograr diseñar los circuitos que generan estas señales se deben seguir dos pasos:

1. Trazar los cronogramas deseados para cada instrucción del conjunto de instrucciones que admite la CPU.
2. Relacionar cada microórden que aparece en los cronogramas con los batidos del reloj del distribuidor de fases y con las distintas informaciones de entrada y de estado del secuenciador. Esto se denomina formular las ecuaciones lógicas de la máquina.

Trazado de Cronogramas

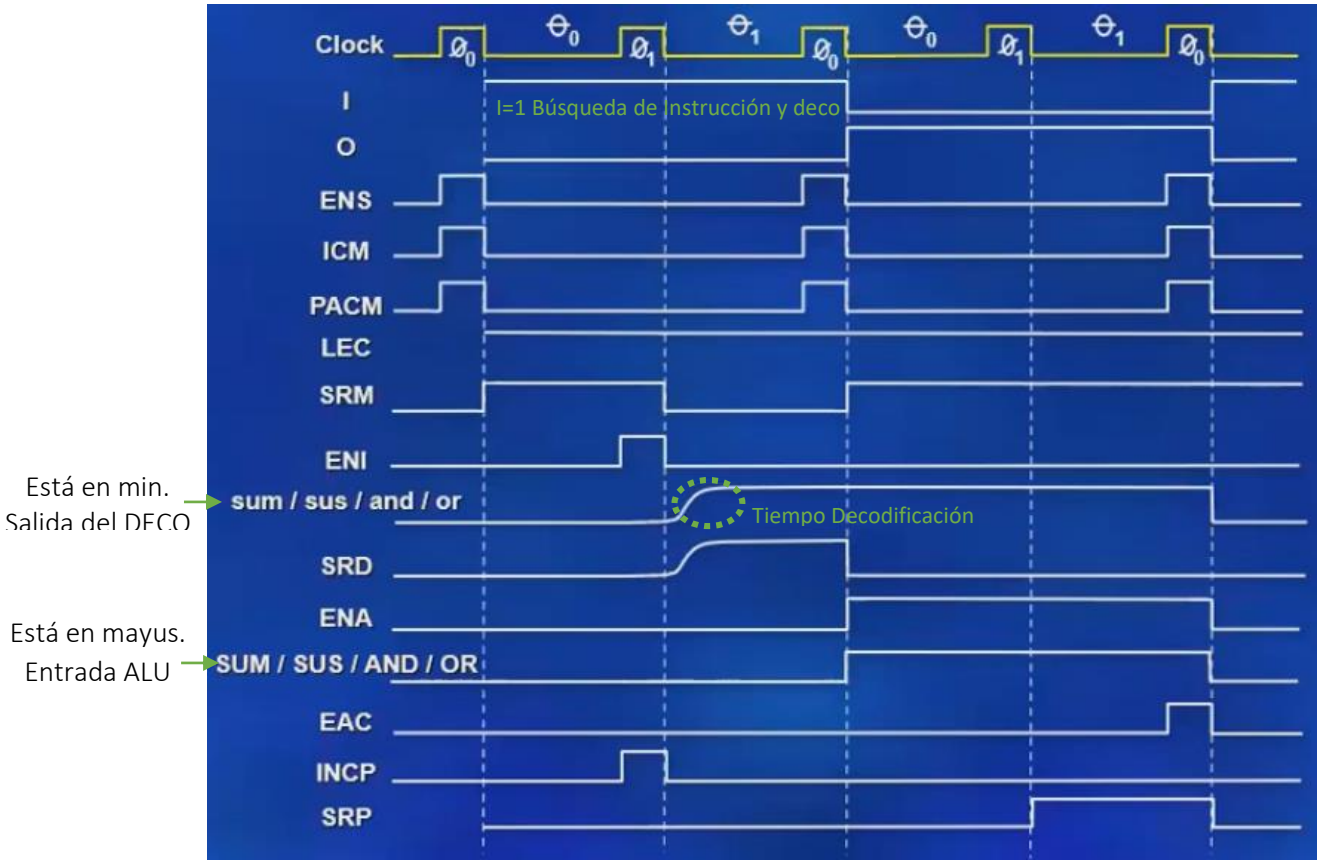
A fin de lograr la realización de los cronogramas se deberá tener en cuenta los tiempos de respuesta de los diferentes operadores implicados en la ejecución de cada micro-operación y de el se deducen las micro-órdenes a generar a cada batido de reloj. Para poder explicar la forma de realizar esta tarea tomaremos el ejemplo de Abacus, para el cual definimos lo siguiente:

Abacus está provisto:

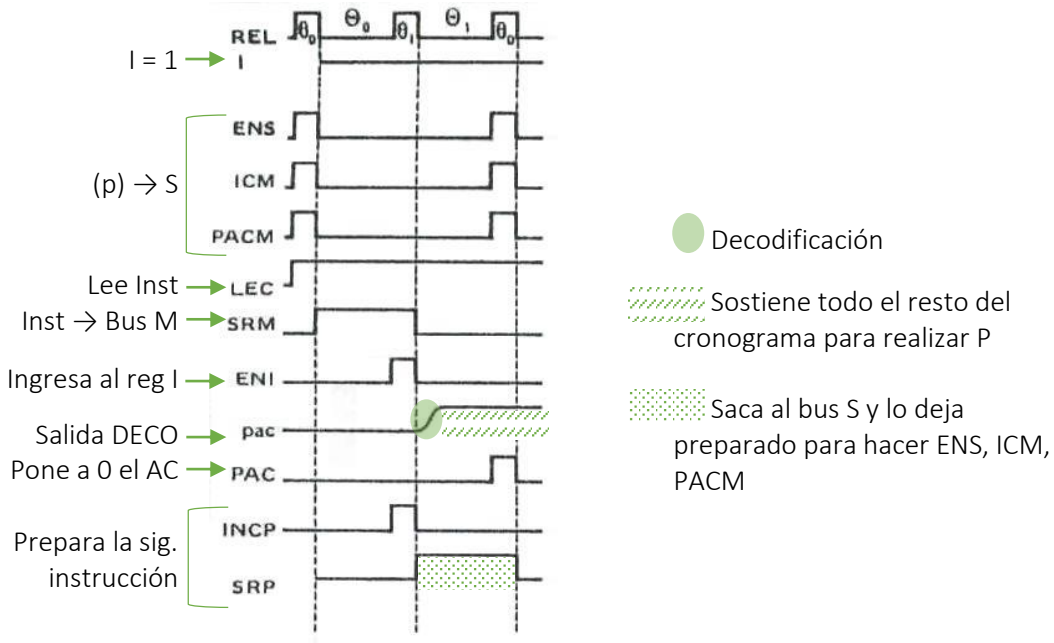
1. De un distribuidor de fases de dos fases
2. De un decodificador de instrucciones descrito anteriormente.

3. De los biestables de estado definidos anteriormente. Se supondrá que el funcionamiento del reloj está condicionado por la señal de detención-marcha, y que la puesta en marcha inicializa el distribuidor de fases.

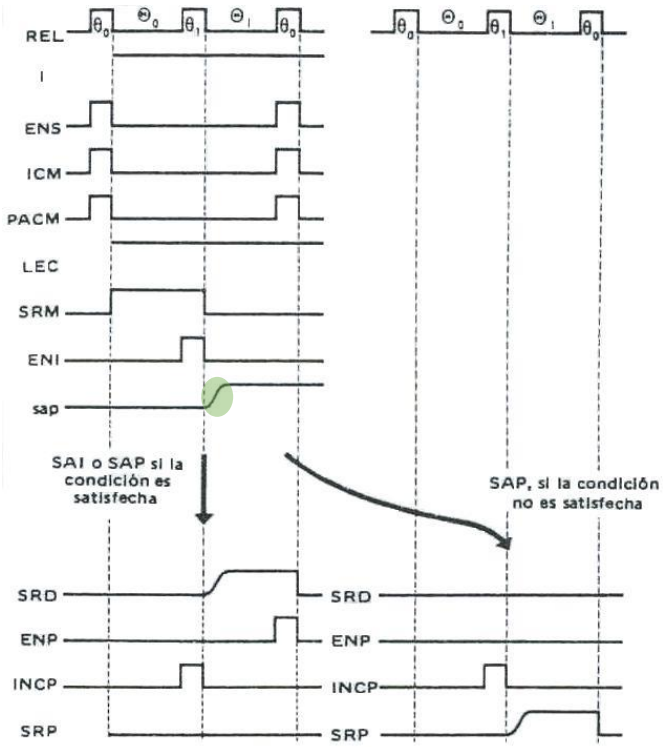
Se observa a continuación los cronogramas de las diferentes instrucciones de Abacus, partiendo del cronograma del secuenciador. Cada cronograma permite definir de forma analítica el comportamiento del secuenciador.



Cronograma sum sus and or

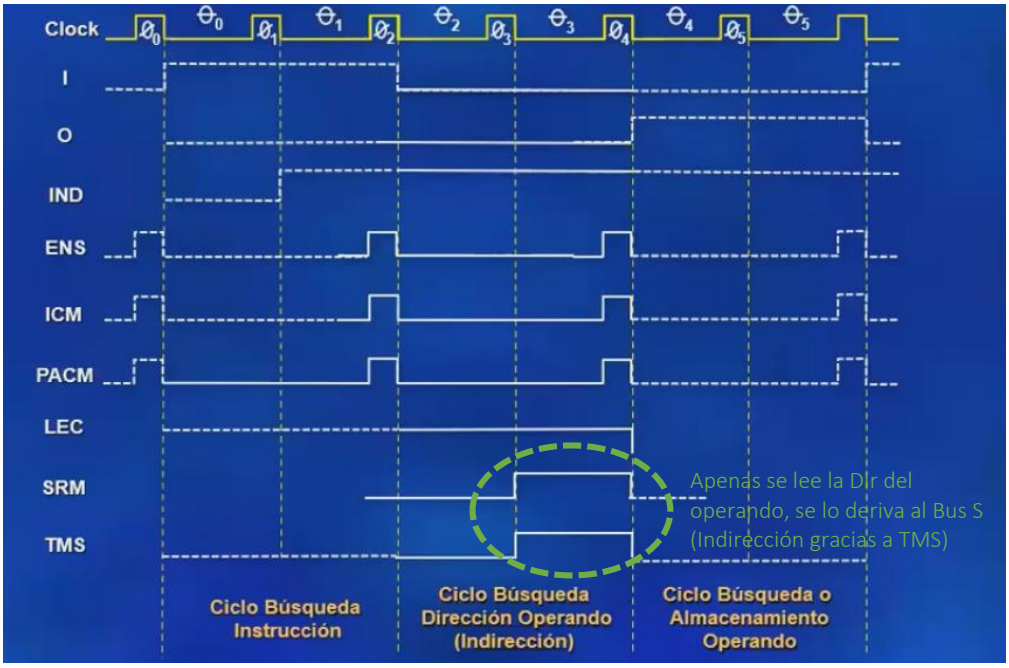


Cronograma de puesta a cero del acumulador



Si signo AC + => Desarrolla (Dir) → P (Condición sastisfecha)
Caso contrario, aborta y sigue con la siguiente instrucción Dir(P)

Cronograma salto incondicional y condicional si signo AC es +



Cronograma Instrucción con ciclo de búsqueda del operando (dir indirecto)

Ecuaciones Lógicas

Un vez realizados todos los cronogramas, el paso siguiente consiste en, micro-órden por micro-órden, buscar todas las veces que debe ser activada y de ellas deducir la expresión lógica en relación con las informaciones de entrada y de estado del secuenciador. **El conjunto de estas expresiones constituye las ecuaciones lógicas del ordenador**.

$$ENS = \theta_0 \qquad ICM = \theta_0 \qquad PACM = \theta_0$$

Deben activarse siempre en θ_0, θ_2 **y** θ_4, pero debido a que Abacus trabaja con un Distribuidor de Fase de dos Fases, se deduce que deben estar activos en el instante θ_0 de todos los cronogramas, tato para la fase de instrucción como para la del operando.

$$ENI = \theta_1 . I \qquad INCP = \theta_1 . I$$

$$\text{Si } I = \text{verdadero} \Rightarrow \theta_1 . I \equiv \theta_3$$

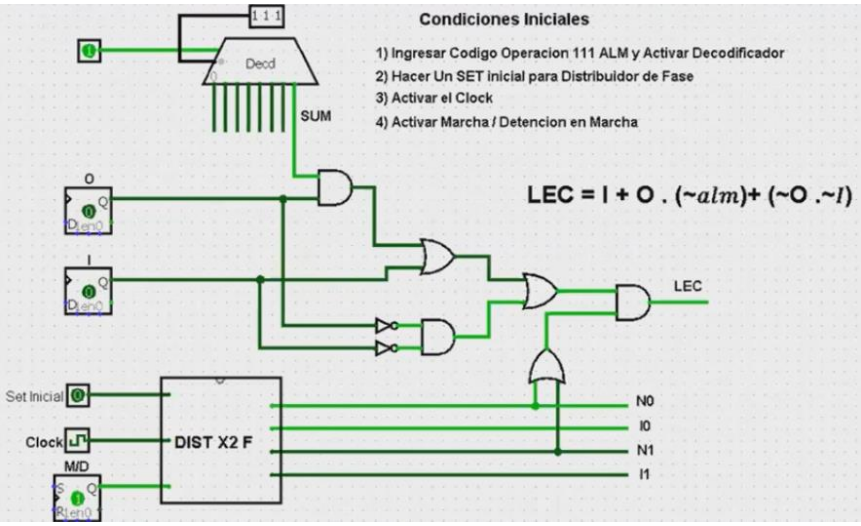
Son señales de tipo impulsional que solo deben generarse en el ciclo de Instrucción, para la etapa θ_1

$$SUM = sum . O \qquad SUS = sus . O \qquad AND = and . O \qquad OR = or . O$$

Son señales de Nivel que deben sostenerse a lo largo de $\theta 0 + \theta_1$ y para cada operación en particular, dentro del ciclo del operando. La expresión $\theta_0 + \theta_1$ se elimina ya que la operación dura las dos fases completas

$$LEC = I + O . \overline{alm} + IND . \theta_1 . (\bar{I} . \bar{O})$$

La señal de lectura es de nivel y debe durar todo el espacio de tiempo entre ciclo de Instrucción y de Operando (menos para almacenamiento) o bien fuera de los ciclos de Operando e Instrucción.



$$SRM = \theta_0 . I + O . \overline{alm} + IND . \theta_1 . (\bar{I} . \bar{O})$$

La salida del Registro M de Nivel debe ser activada:

1. En la búsqueda de instrucción para θ_0, es decir (θ_0 . I)
2. En el ciclo de operando para sum, sus, or, and y durante el tiempo que le lleve a la ALU realizarlo O . ($\theta_0 + \theta_1$) . (sum+sus+or+and). Debido a que ($\theta_0 + \theta_1$) cubre todo el espacio de tiempo que dura O, es inútil expresarlo. A demás a que sum, sus, or y and requieren de un ciclo completo de Operando (con excepción de alm), podemos expresarlo de manera más simple como O . $\overline{alm}$
3. En el caso del Direccionamiento Indirecto, en el ciclo de búsqueda de la dirección y durante todo el ciclo θ_1.
4. Para Direccionamiento Indirecto, es necesario contemplar la habilitación de la señal IND en θ_1 dentro del "Ciclo de Almacenamiento del Operando" (fuera del ciclo de instrucción y el del operando)

Son señales del tipo impulsional que solo deben generarse en el ciclo de la Instrucción, para la etapa θ_1.

SI = θ_0 . O

El propio secuenciador deberá tener la tarea de posicionar a los Biestables I y O, por tal motivo I debe ser puesto a Uno al final del ciclo del operando.

RI = θ_0 . I . (sum + sus + or + and + alm)

I debe ser Puesto a Cero al final del ciclo de aquellas instrucciones que requieren ciclo de operando, es decir, para ciclo de instrucción dentro del espacio θ_0, para la suma, resta, or, and y alm, el biestable I deberá ser reseteado ya que se encuentra en la fase de resolución del Operando.

SO = $\overline{IND}$. θ_0 . I (sum + sus + or + and + alm) + IND . θ_0 . $\bar{I} . \bar{O}$

O debe ser activado ya sea después de un ciclo de instrucción, cuando es con operando y sin direccionamiento indirecto, o bien al final del ciclo de búsqueda de la dirección efectiva, en el caso de direccionamiento Indirecto y por fuera de la instrucción y operando.

RO = θ_0 . O

O debe ser puesto a Cero al final del Ciclo de operando.

A continuación se observan todas las ecuaciones para formar el secuenciador de Abacus:

ENS = θ_0

ICM = θ_0

PACM = θ_0

LEC = $I + O \cdot \overline{alm} + (\bar{I} \cdot \bar{O})$

ESC = $O \cdot alm$

SRM = I . Θ_0 + O . $\overline{alm}$ + IND . Θ_1 . ($\bar{I}$. $\bar{O}$)

TMS = $IND \cdot \bar{I} \cdot \bar{O} \cdot \Theta_1$

ENI = $\theta_1 \cdot I$

SRD = $\Theta_1 \cdot I \cdot (sum + sus + and + or + alm + sap \cdot AP + sai)$

SRP = $\Theta_1 \cdot I \cdot (pac + sap \cdot AN) + \Theta_1 \cdot O$

INCP = $\theta_1 \cdot I$

ENP = $\theta_0 \cdot (sap \cdot AP + sai)$

SUM = $sum \cdot O$

SUS = $sus \cdot O$

AND = $and \cdot O$

OR = $or \cdot O$

ENA = $O \cdot \overline{alm}$

EAC = $O \cdot \overline{alm} \cdot \theta_0$

PAC = $pac \cdot \theta_0$

SAC = $O \cdot \Theta_0 \cdot alm$

ENM = $O \cdot \theta_1 \cdot alm$

SI = $SI = \Theta_0 \cdot O$

RI = $RI = \theta_0 \cdot I \cdot (sum + sus + or + and + alm)$

SO = $SO = \overline{IND} \cdot \theta_0 \cdot I \cdot (sum + sus + or + and + alm) + IND \cdot \theta_0 \cdot \bar{I} \cdot \bar{O}$

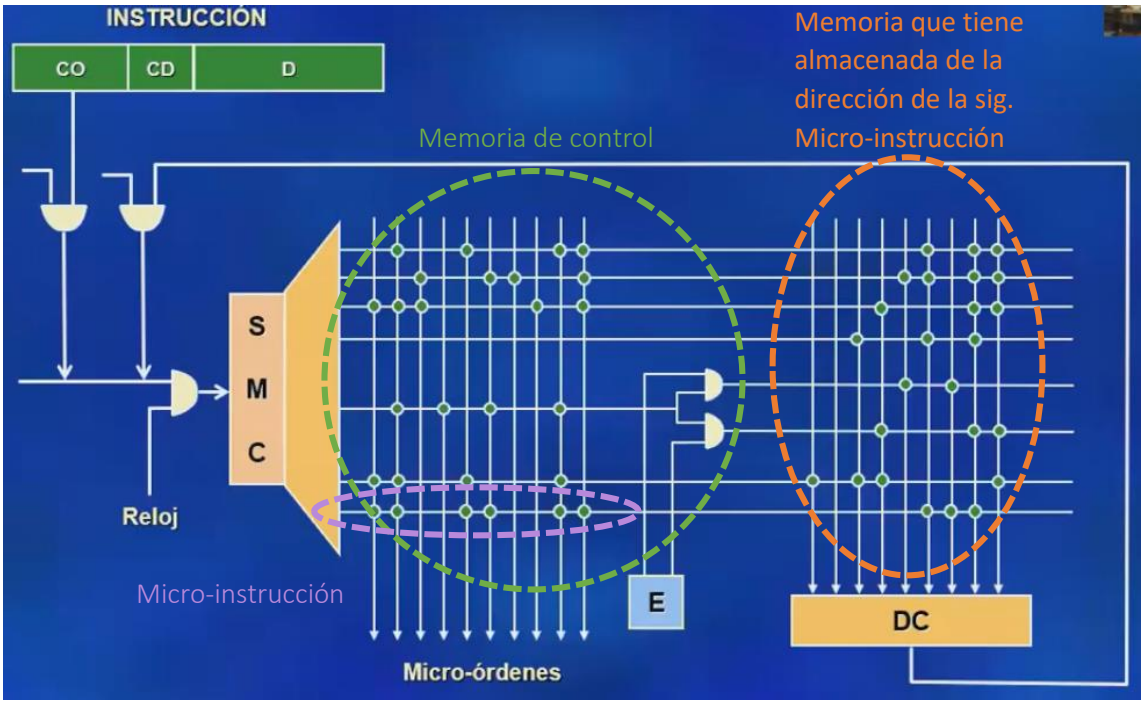
RO = $RO = \theta_0 \cdot O$

El secuenciador de Abacus será la materialización de sus biestables de estado y del circuito combinacional correspondiente a las ecuaciones lógicas. Es de destacar que el ordenador Abacus es una máquina simple con un conjunto de instrucciones reducido para facilitar su estudio.

En una compleja CPU moderna, el número de ecuaciones booleanas necesarias para definir la unidad de control es muy grande. La tarea de implementar un circuito combinacional que satisfaga todas esas ecuaciones llega a ser muy difícil.

El resultado es que, por regla general, se usa una aproximación mucho más sencilla, conocida como microprogramación.

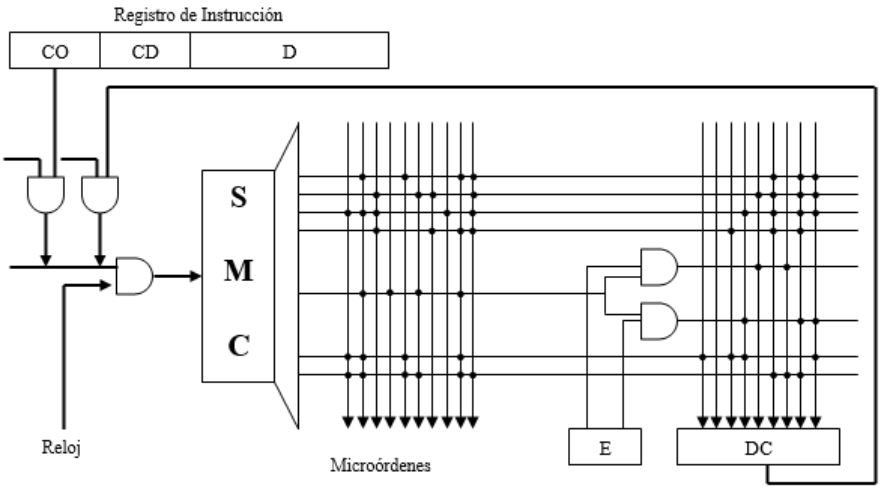
Modelo de Wilkes



Wilkes presenta este modelo innovador donde trata de introducir una mecánica de micro-ordenes de forma micro-programadas. Es un modelo muy flexible el cual nos permite prescindir de ecuaciones lógicas complejas y pasar a una memoria de control

Dada una instrucción en lenguaje de máquina, compuesta como mínimo de cod de operación, el cod de operación está relacionado con la dirección de la primera micro-instrucción (micro-programa). Un micro programa es un conjunto de micro-instrucciones.

El modelo de WILKES contiene la base del funcionamiento de la lógica micro-programada. La representación esquemática es la siguiente:



A la entrada del dispositivo de control tenemos el código de operación de la instrucción a ejecutar, la que se considera como la dirección de la primera micro-instrucción del micro-programa asociado a la instrucción. Esa dirección se envía al registro SMC (Selección de Memoria de Control) que, cuando se decodifica, permite activar el hilo de la palabra horizontal sobre el cual está grabada la micro-instrucción de la siguiente manera:

Los unos están representados por un acoplamiento (marcado con un punto en el esquema) con el correspondiente hilo del bit vertical, los ceros por la ausencia de acoplamiento. A la salida, la micro-instrucción está compuesta por dos partes: una, el código de la micro-instrucción que da directamente las micro-órdenes a distribuir al conjunto de la máquina; y la otra, la dirección de la próxima micro-instrucción (DC o Dirección de Control) que se transfiere, en el siguiente pulso de reloj, al SMC. El micro-programa continúa hasta la última micro-instrucción, la cual cargará en el SMC el nuevo código de Operación.

También se hace referencia en el esquema, a la posibilidad de realizar saltos condicionales, utilizando el biestable E que puede ser puesto a uno bajo determinadas condiciones por las micro-instrucciones.

Microinstrucciones

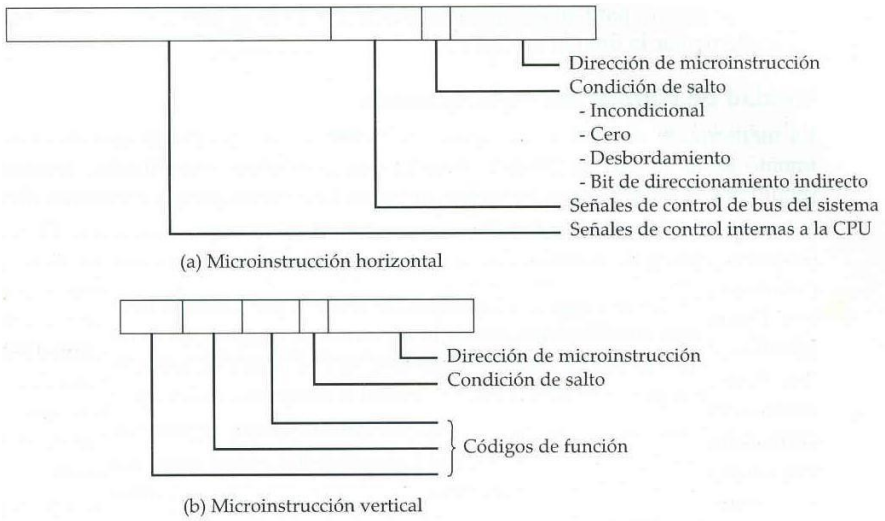
El diseño de un secuenciador cableado para un ordenador complejo, como lo son los actuales, es una tarea muy difícil, además, el diseño es relativamente inflexible. Es difícil cambiar el diseño si se desea añadir una nueva instrucción máquina. **La alternativa actual es un secuenciador micro-programado**.

En realidad es un lenguaje conocido como lenguaje de microprogramación. Cada línea describe un conjunto de micro-operaciones que suceden a la vez y se conoce como microinstrucción. Una secuencia de microinstrucciones se conoce como microprograma, o firmware. Este último término refleja el hecho de que un microprograma está a mitad de camino entre hardware y software. Es más fácil diseñar en firmware que en hardware, pero es más difícil escribir un programa firmware que un programa software.

Para usar el concepto de microprogramación en el diseño de un secuenciador debemos considerar que para cada micro-operación, todo lo que la unidad de control puede hacer es generar un conjunto de señales de control, de modo que, para cualquier micro-operación, cada línea de control procedente de la unidad de control está activa o inactiva. Esta condición puede representarse con un dígito binario para cada línea de control. De este modo, podríamos construir una palabra de control en la que cada bit representara una línea de control. Entonces, cada micro-operación se representaría mediante un patrón diferente de 1s y 0s en la palabra de control.

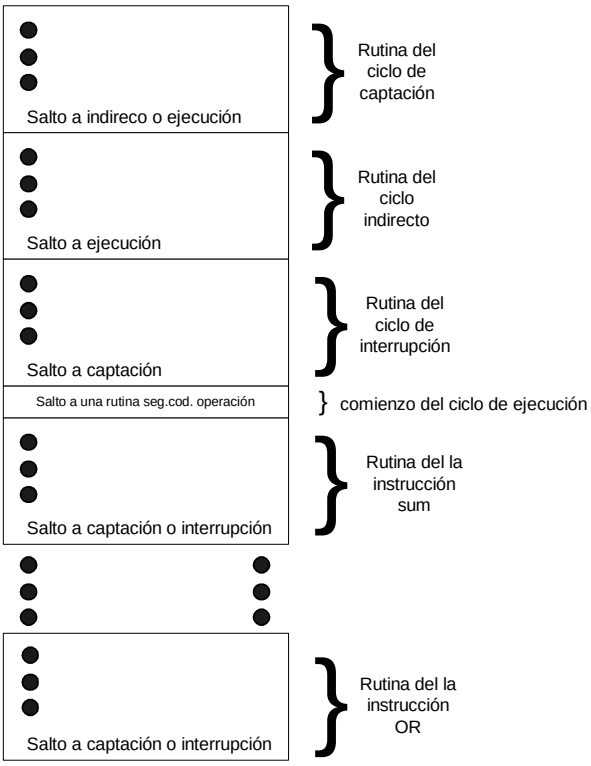
A continuación, hemos de admitir que la secuencia de micro-operaciones no es fija. Algunas veces tenemos un ciclo indirecto; otras no. Por tanto, coloquemos nuestras palabras de control en una memoria, cada palabra en una dirección única. Añadamos ahora un campo de dirección a cada palabra de control, indicando la posición de la siguiente palabra de control a ejecutar si una determinada condición es cierta (por ejemplo, que el bit de direccionamiento indirecto de una instrucción que referencia a la memoria sea 1). Añadamos también algunos bits para especificar la condición.

El resultado se conoce **como micro-instrucción horizontal** y se muestra en la figura siguiente. El formato de la microinstrucción o palabra de control es el siguiente. Hay un bit para cada línea de control interna a la CPU y un bit para cada línea de control del bus del sistema. Hay un campo de condición que indica la condición bajo la cual debe producirse un salto, y hay un campo con la dirección de la microinstrucción a ejecutar cuando el salto se produzca.

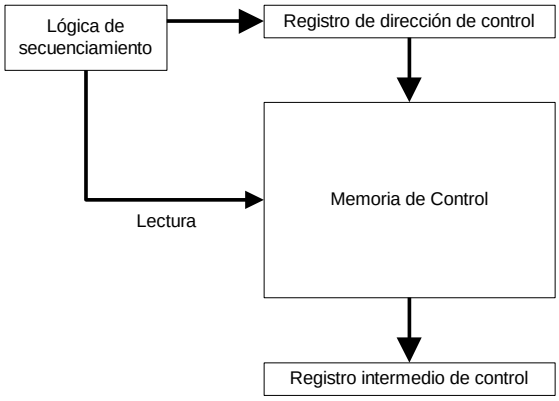


Unidad de Control Microprogramada

La figura siguiente muestra los elementos clave de esta implementación. El conjunto de microinstrucciones se almacena en la memoria de control. El registro de dirección de control contiene la dirección de la siguiente microinstrucción a leer. Cuando se lee una microinstrucción de la memoria de control, se transfiere al registro intermedio de control. La parte izquierda de ese registro se conecta a las líneas de control que salen de la unidad de control. De este modo, leer una microinstrucción de la memoria de control es lo mismo que ejecutar la microinstrucción. El tercer elemento que muestra la figura es una unidad de secuenciamiento que carga el registro de dirección de control y emite una orden de lectura.



Organización de la memoria de control



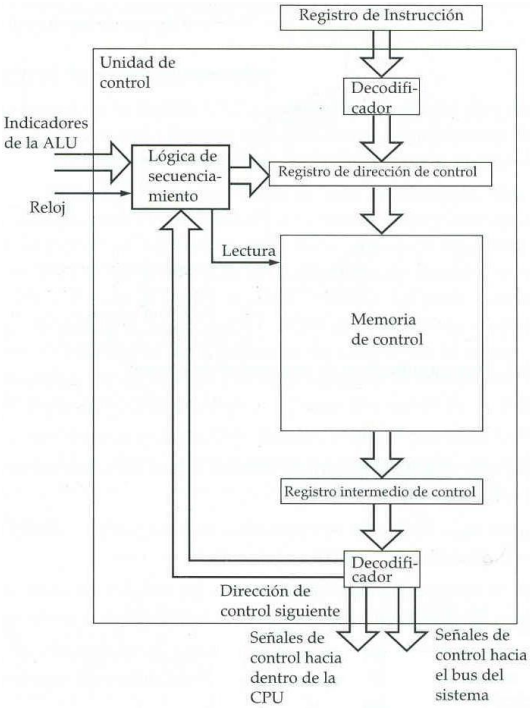
Micro-arquitectura de la Unidad de Control

Examinando la estructura de la figura siguiente vemos la UC tiene las mismas entradas (reg. I, Indicadores de la ALU, reloj) que una UC cableada, y las mismas salidas (señales de control). La UC funciona como sigue:

1. Para ejecutar una instrucción, la unidad lógica de secuenciamiento emite una orden de lectura a la memoria de control.
2. La palabra cuya dirección se especifica en el registro de dirección de control se lee en el registro intermedio de control.
3. El registro intermedio de control genera las señales de control y contiene además la información de dirección siguiente para la unidad lógica de secuenciamiento.
4. La unidad lógica de secuenciamiento carga en el registro de dirección de control una nueva dirección, basada en la información siguiente del registro intermedio de control y en los indicadores de la ALU.

Todo esto sucede durante un pulso de reloj.

Al final de la ejecución de cada microinstrucción, la unidad lógica de secuenciamiento cara una nueva dirección el registro de dirección de control.



Funcionamiento de la unidad de control microprogramada

Dependiendo del valor de los indicadores y del registro intermedio de control, se toma una de las tres siguientes decisiones:

- Captar la microinstrucción siguiente: se suma 1 al registro de dirección de control.
- Saltar a una nueva rutina según indica una microinstrucción de salto: El campo de dirección del registro intermedio de control se carga en el registro de dirección de control.
- Saltar a la rutina de una instrucción máquina: se carga el registro de dirección de control en función del código de operación almacenado en el registro I.

Se observa también en la figura dos módulos rotulados decodificador. El decodificador de arriba traduce el código de operación del registro I en una dirección de memoria de control.

El otro decodificador no se usa con micro-instrucciones horizontales pero sí con microinstrucciones verticales. En una **micro-instrucción horizontal** cada bit del campo de control corresponde a una línea de control. En una **micro-instrucción vertical**, se usa un código para cada acción a realizar, por ejemplo, (P) → S, y el decodificador traduce este código en señales de control individuales.

La ventaja de las micro-instrucciones verticales es que son más compactas (ocupan menos bits) que las microinstrucciones horizontales, a costa de añadir una pequeña lógica y cierto retardo de tiempo.

Secuenciamiento De Micro-instrucciones

Las dos tareas básicas realizadas por una UC micro-programada son:

- **El secuenciamiento de micro-instrucciones**: Obtener la siguiente micro-instrucción de la memoria de control.
- **La ejecución de micro-instrucciones**: Generar las señales de control necesarias para ejecutar la micro-instrucción.

Al diseñar una UC, las dos tareas deben considerarse a la vez, ya que las dos afectan al formato de la micro-instrucción y a la temporización de la UC.

Consideraciones de diseño

Hay dos cuestiones involucradas en el diseño de una técnica de secuenciamiento de microinstrucciones: el tamaño de la microinstrucción y el tiempo de generación de la dirección.

El primer asunto es evidente; minimizar el tamaño de la memoria de control reduce su costo. El segundo asunto es sencillamente un deseo de ejecutar las microinstrucciones tan rápido como sea posible.

Cuando se ejecuta un microprograma, la dirección de la siguiente microinstrucción a ejecutar está en una de estas situaciones:

- Determinada por el registro de instrucción
- Siguiente dirección secuencial
- Salto

La primera situación tiene lugar solo una vez por ciclo de instrucción, justo tras la captación de la instrucción. La segunda situación es la más común en la mayoría de los diseños.

No obstante, el diseño no se puede optimizar solo para los accesos secuenciales. Los saltos, tanto condicionales como incondicionales, son una parte necesaria del microprograma. Además, las secuencias de micro-instrucciones tienden a ser cortas; una de cada tres o cuatro micro-instrucciones podría ser un salto.

Por consiguiente, es importante diseñar técnicas compactas y eficientes en cuanto al tiempo para los saltos a micro-instrucciones.

Técnicas de secuenciamiento

A partir de la micro-instrucción en curso, de los indicadores de condición, y del contenido del registro de instrucción, hay que generar una dirección de la memoria de control para la siguiente micro-instrucción.

Se ha usado una gran variedad de técnicas. Podemos agruparlas en tres categorías, como ilustran las figuras siguientes.

Estas categorías se basan en el formato de la información de dirección de la micro-instrucción:

- Dos campos de dirección
- Un único campo de dirección
- Formato variable

Generación de la dirección

Otro punto de vista es considerar las diversas formas en que la siguiente dirección se puede obtener o calcular.

La siguiente tabla relaciona las diversas técnicas de generación de la dirección. Estas se pueden dividir en técnicas explícitas, en las que la dirección aparece explícitamente en la micro-instrucción, y técnicas implícitas, que requieren lógica adicional para generar la dirección.

Explícitas	Implícitas
Dos Campos	Traducción
Salto incondicional	Adición
Salto condicional	Control residual

Nos hemos ocupado esencialmente de las técnicas explícitas. Con una aproximación de dos campos, hay dos direcciones alternativas disponibles en cada microinstrucción.

Usando un único campo de dirección o un formato variable, se pueden implementar varias instrucciones de salto. Una instrucción de bifurcación condicional depende de los siguientes tipos de información:

- Indicadores de la ALU
- Parte del código de operación o campos de modo de direccionamiento de la instrucción máquina
- Partes de un registro seleccionado, tales como el bit de signo
- Bits de estado dentro de la UC.

También se usan frecuentemente varias técnicas implícitas. Una de ellas, la traducción, se necesita en casi todos los diseños. La parte de código de operación de una instrucción máquina se traduce a una dirección de microinstrucción.

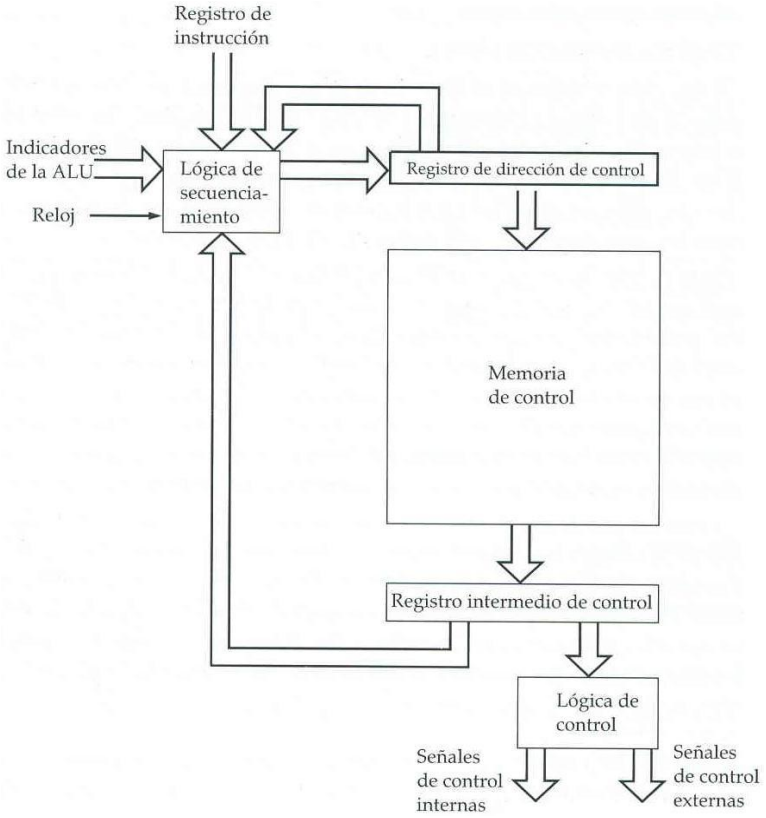
Esto se presenta solo una vez por ciclo de instrucción. Una técnica implícita común combina o suma dos partes de una dirección parar formar la dirección completa.

Ejecución De Microinstrucciones

El **ciclo de micro-instrucción** es el evento básico de una CPU micro-programada. Cada ciclo se compone de dos partes: captación y ejecución. **La parte de captación** depende de la generación de una dirección de micro-instrucción, como se vio en la sección precedente. Esta sección se ocupa de la ejecución de una micro-instrucción.

Genera la dirección de la siguiente micro-instrucción, usando como entradas el registro de instrucción, los indicadores de la ALU, el registro de dirección de control (para incrementarlo), y el registro intermedio de control. Este último puede proporcionar una dirección real, bits de control, o ambos. Este módulo está controlado por un reloj que determina la temporización del ciclo de instrucción.

El módulo de lógica de control genera las señales de control en función de algunos de los bits de la micro-instrucción. Debe quedar claro que el formato y contenido de la micro-instrucción determinará la complejidad del módulo de lógica de control.



Organización de la UC

Una taxonomía de las micro-instrucciones

Las micro-instrucciones se pueden clasificar de varias formas:

- Vertical/horizontal
- Empaquetada/desempaquetada
- Micro-programación “hard”/”soft”
- Codificación directa/indirecta

Todas ellas se refieren al formato de la micro-instrucción.

En la propuesta original de Wilkes, cada bit de una micro-instrucción producía directamente una señal de control o un bit de la dirección siguiente. Hemos visto que **son posibles tanto esquemas de secuenciamiento de direcciones más complejos**, como esquemas que usen menos bits por microinstrucción.

Estos esquemas requieren un módulo de lógica de secuenciamiento más complejo. Existe un tipo de compromiso semejante para la parte de la micro-instrucción que atañe a las señales de control. Se pueden ahorrar bits de la palabra de control codificando la información de control, y decodificándola más tarde para producir las señales de control.

Para hacer esto consideremos que hay un total de K señales de control internas y externas diferentes que tiene que generar la UC. En el esquema de Wilkes, K bits de la microinstrucción se dedicarían a este propósito. Esto permite que se puedan generar las 2K combinaciones posibles de señales de control durante cualquier ciclo de instrucción.

Pero somos capaces de hacerlo mejor si observamos que no todas las combinaciones posibles se usarán. Por ejemplo:

- Dos fuentes no se pueden llevar al mismo destino
- Un registro no puede ser a la vez fuente y destino
- Solo un patrón de señales de control se puede presentar a la ALU cada vez
- Solo un patrón de señales de control se puede presentar al bus de control externo cada vez

De esta manera, para una CPU dada, puede hacerse una lista con todas las posibles combinaciones de señales de control admisibles, obteniendo un número de posibilidades $Q<2^K$, que podrían codificarse con $\log_2 Q$ bits, siendo $(\log_2 Q)<K$.

Esta sería la forma más estricta posible de codificación que preserva todas las combinaciones permisibles de señales de control. En la práctica, este sistema de codificación no se usa, por dos razones:

- Es tan difícil de programar como un esquema decodificado puro (como el de Wilkes).
- Requiere un módulo de lógica de control complejo y, por consiguiente, lento.

En lugar de eso, se adoptan algunos compromisos. Los hay de dos tipos:

1. Se usan más bits de los estrictamente necesarios para codificar las posibles combinaciones.
2. Algunas combinaciones que son físicamente permisibles no se pueden codificar.

El último tipo de compromiso tiene el efecto de reducir el número de bits. El resultado neto, sin embargo, es que se usan más bits que $\log_2 Q$.

Visto lo dicho hasta aquí, podemos ver que el campo de señales de control del formato de micro-instrucción se encuadra dentro de un espectro. En un extremo, hay un bit para cada señal de control, en el otro extremo, se usa un formato muy codificado. La tabla siguiente muestra otras características de una unidad de control micro-programada que también se encuadran dentro de espectros de posibilidades, y que en general depende del espectro del grado de codificación.

Características	
Microinstrucción no codificada	Microinstrucción muy codificada
Muchos bits	Pocos bits
Visión detallada del hardware	Visión global de hardware
Difícil de programar	Fácil de programar
Concurrencia explotada completamente	Concurrencia no explotada completamente
Poca o ninguna lógica de control	Lógica de control compleja
Ejecución rápida	Ejecución lenta
Optimización de las prestaciones	Optimización de la programación

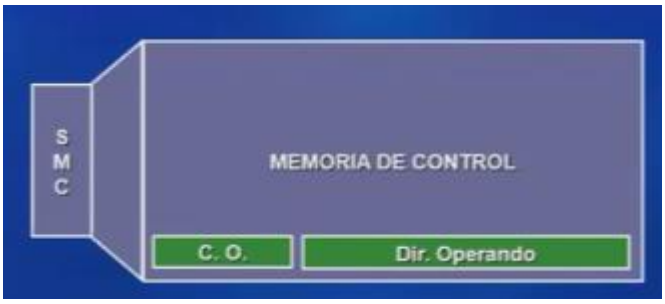
Terminología	
Desempaquetada	Empaquetada
Horizontal	Vertical
"Hard"	"Soft"

Codificación de las Microinstrucciones

Existen dos métodos extremos para pasar de la configuración binaria de la micro-instrucción a las diversas micro-órdenes correspondientes. El primer método consiste en asociar un bit de la microinstrucción a cada micro-órden (mayor longitud de la micro-instrucción y difícil programación, permite ejecutar varias micro-órdenes simultáneas). En el otro extremo, el segundo método consiste en codificar el conjunto de las posibles micro-órdenes con el número mínimo de bits (menor longitud de microinstrucción y mayor decodificación, se necesitaría una micro-instrucción por cada micro-órden).

❀ **Codificación tipo instrucción.** Este tipo de codificación da a la micro-instrucción una estructura semejante a la de una instrucción, con código de operación y dirección de operando (en registro o en memoria local).

La decodificación de las micro-instrucciones para obtener las correspondientes micro-órdenes es relativamente compleja, siendo compensado por la pequeña longitud de la micro-instrucción.

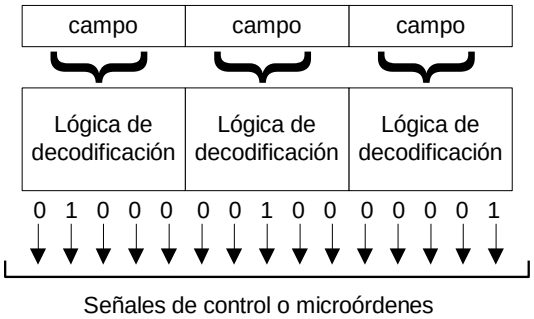


→ Brinda a la Micro-Instrucción un formato semejante al de una instrucción.
→ Consta de código de operación y de dirección del operando.
→ La decodificación de las Micro-Instrucciones para obtener las correspondientes Micro-ordenes es compleja.
→ Se compensa la complejidad con la pequeña longitud de las Micro-Instrucciones.
→ Pone la Micro-Programación al alcance del programador de lenguaje de máquina.

❀ **Codificación por campos.** Se dividen las distintas micro-órdenes en grupos independientes de modo tal que sería imposible que, dentro de cada grupo, se den micro-órdenes simultáneas. Cada grupo, además, corresponde a un determinado tipo de función: apertura de puertas

de diversos registros a un mismo bus, gobierno de una unidad funcional, gobierno de la memoria local, etc.

A modo de ejemplo podemos decir que las micro-órdenes LEC y ESC, de lectura y escritura en memoria, correspondientemente, estarían en el mismo grupo y nunca serían activadas simultáneamente. Cada campo contiene un código que efectúa la decodificación correspondiente y activa una señal de control.



Codificación por campos con una micro-órden activa por campo

→ Cuando la Micro-Instrucción se ejecuta, cada campo se decodifica y genera señales de control

→ Con N Campos se definen N acciones simultáneas.

→ Cada acción se traduce en la activación de una o más señales de control

→ Se debe cuidar que cada señal de control no pueda ser activada por más de un campo (no siempre se puede).

→ Debe ser posible activar cada señal de control al menos por un campo

→ Un campo individual podrá constar de L bits o sea 24 códigos posibles donde cada uno deba codificar un patrón distinto de señales de control

Direccionamiento de las Micro-Instrucciones

Direccionamiento Secuencial

Este modelo es básicamente similar al de Wilkes, ya que en condiciones normales la dirección de la próxima micro-instrucción la calcula el SMC con capacidad de incrementar la última dirección de la micro-instrucción y de manera secuencial. El registro SMC tiene capacidad de cuetna, de manera tal que la primera dirección de la micro-instrucción la da el cod de operación, pero luego se incrementa SMC para apuntar a la siguiente en posición física.

Para el caso de un salto, se especializa un bit de la micro-instrucción como indicador verdadero o falso, dejando el resto de la palabra de control para indicar la dirección de la próxima micro-

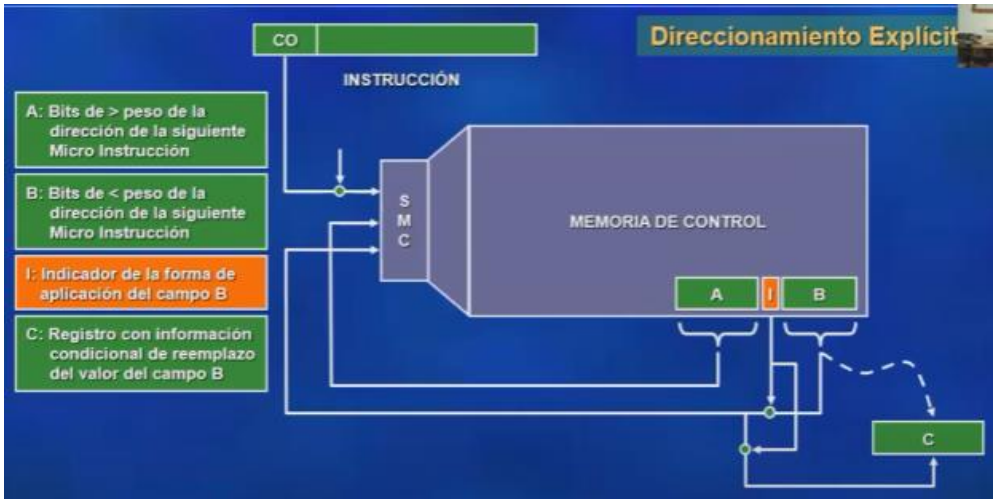
instrucción. Se sacrifica una palabra de la MC para indicar la dirección de salto por condición satisfecha. Por insatisfecha, continúa normalmente, desecha esta función de salto y continúa con la siguiente secuencialmente armada según SMC

Ventaja: simple de programar y no alarga (no encarece) el tamaño de la micro-instrucción.

Desventaja: un salto ocupa una micro-instrucción, por sí solo dependiendo el tiempo. Se lo usa cuando la concepción de la micro-programación de la máquina no requiera muchos altos

Direccionamiento Explicito

Esquema preconizado por Wilkes para evitar cablear un salto condicional. Hay una parte de codificación por campo y la dirección de la siguiente instrucción. Tiene un nivel importante de codificación por campo para reducir el tamaño.



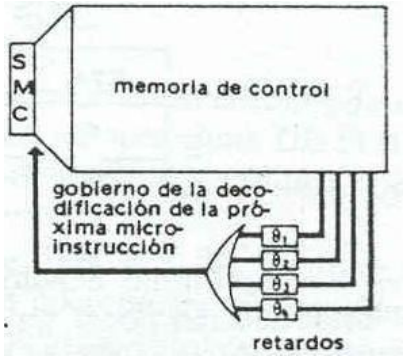
En condiciones normales, la dirección de la siguiente micro-instrucción surge de yuxtaponer las direcciones de los campos A y B. En un salto condicional:

- → Si I=0, el campo B podrá contener los bits de menor peso de la dirección de salto si este fuese conocida
- → Si I=1, B contiene la dirección de un registro C que almacena los bits de menor peso de la dirección de salto

Acompasamiento

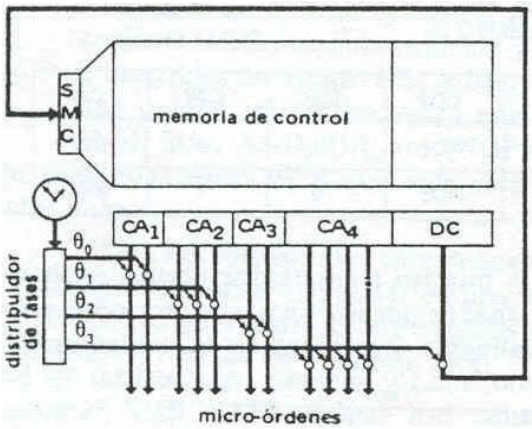
Es con qué ritmo se gobierna la salida de las micro-ordenes

Si se trabaja con CODIFICACIÓN TIPO INSTRUCCIÓN, el único compás existente es el de la demanda de microinstrucciones. Los ritmos son variables por lo que su comportamiento es variable.



Ritmo variable definido por la micro-instrucción

Si se trabaja con CODIFICACIÓN POR CAMPOS, será necesario disponer de un Distribuidor de fases que indique en qué momento emitir las micro-órdenes de cada campo donde los ritmos son fijos. Va gobernando las salidas, tiene tantas fases como cuantos campos tenga.



Micro-programación horizontal y ritmo fijo

Consideraciones acerca de los secuenciadores

Ventajas y desventajas

La ventaja que sobresale en los secuenciadores cableados es la velocidad de ejecución de una micro – instrucción, ya que para ejecutarse solamente depende del tipo de instrucción y el estado de la máquina. No ocurre lo mismo con los secuenciadores micro-programados debido a que al tipo de instrucción y al estado de la máquina se le suma el tiempo de ejecución del microprograma.

Existen, sin embargo dos ventajas de los secuenciadores micro-programados que provocaron que los diseñadores se volcaran a la construcción de los mismos: la facilidad en el diseño y la modularidad en el crecimiento. Diseñar un secuenciador cableado es muy complejo y más aún depurarlo. Agregar una micro – instrucción representa un trabajo muy arduo en los secuenciadores cableados y a veces implica su rediseño.

En la evolución del diseño de computadoras se sopesaron la velocidad y la simpleza, y a partir de ello surgieron dos tecnologías que durante mucho tiempo fueron contrapuestas: CISC y RISC.

Tecnologías CISC y RISC

Evolución histórica

Al principio los programas se realizaban en lenguaje de máquina, esto hacía que el programador tuviese que tener un gran conocimiento del Hardware y dependía de él la mejora o no de la performance en la ejecución de sus programas.

Apareció entonces la idea de hacer que la suma de varias instrucciones sencillas se plasmaran en un microprograma haciendo más fácil la tarea del programador. Esta idea se conoció con el nombre de CISC (Complex Instructions Set Computer).

Funcionalmente la CPU creció con un modulo de decodificación. Después de efectuarse una lectura, la instrucción es decodificada según varios campos y sub-campos. Como primer resultado se obtiene un puntero al microprograma, confinado con el Control Store. Inmediatamente el control se transfiere a éste y las acciones se suceden en el orden del microprograma (utilizando varios ciclos de reloj).

A principios de la década del 80, el mundo aparecía rendido a los pies del CISC, desde las PCs hasta los Mainframes, y a nadie se le ocurría otra cosa que tratar de mejorarlo. Ocupados en esta tarea alguien se dio cuenta que algo andaba mal.

Después de diferentes testeos se comprobó que a excepción de las operaciones de punto flotante, la mayor parte del esfuerzo del procesamiento se concentra en las instrucciones más simples, lo que significa que la mayor parte del tiempo no se utiliza la CPU ya que aquellas se ejecutan por medio de un microprograma.

Como contrapartida a la tecnología CISC surgió entonces la tecnología RISC (Reduced Instructions Set Computer) con las siguientes características:

- Elimina las instrucciones complejas del Instruction Set (a excepción de las operaciones de punto flotante)
- Elimina el micro-código
- Logra ejecutar todas las instrucciones en un ciclo de reloj.
- Jerarquiza el uso del compilador

Características de la Tecnología CISC (Complex instruction set computer)

La denominación CISC se debe a que los procesadores que utilizan esta tecnología pueden ejecutar desde instrucciones simples (como las que ordenan sumar o restar dos números que están almacenados en registros de la CPU y el resultado asignarlo a uno de estos registros), hasta instrucciones muy complejas (como movimiento de cadenas de caracteres de gran longitud y variable, en procesamiento de textos).

Las instrucciones simples, luego de su decodificación, pueden ejecutarse en un pulso de reloj, mientras que las complejas requieren un número de pulsos que dependen de la secuencia de pasos necesarios para su ejecución.

Cada paso se lleva a cabo mediante una combinación binaria (micro-código), que aparece en las líneas de control de la UC con cada pulso de reloj, la cual activa los circuitos que intervienen en ese paso. Las sucesivas combinaciones (micro-códigos) que requiere la ejecución de una instrucción compleja, deben ser provistas por una memoria de control que las almacena y que forma parte de la UC.

Por lo tanto, un procesador CISC necesariamente debe contener una memoria de control con los micro-códigos, para poder ejecutar las instrucciones complejas. Esta es una de las características CISC.

En general; cada operación que ordena una instrucción de un procesador CISC presenta variantes para ser aplicadas a diversas estructuras de datos, desde simples constantes y variables, hasta matrices y otras. Así, una instrucción que ordena sumar tiene muchas variantes (códigos) en función de la estructura de datos sobre la cual opera. Es como si existieran tantas instrucciones que ordenan una misma operación como estructuras de datos típicas se han definido para operar, lo cual se denomina "modos de direccionamiento" de una instrucción.

La existencia de muchos "modos de direccionamiento" para realizar la operación que ordena una instrucción, es otra de las características de complejidad de los procesadores CISC. Esto se manifiesta en que el set de instrucciones de una máquina CISC presente un número elevado de códigos de instrucción.

Esto definitivamente incide sobre el set de instrucciones de una máquina CISC. Un set de instrucciones CISC posee alrededor de 210 instrucciones.

Asimismo, lo anterior exige instrucciones que ocupan distinta cantidad de bytes en memoria.

Un aspecto importante es el solapamiento de micro-órdenes (o “PIPE LINE”) ya que al haber instrucciones que requieren distinta cantidad de pulsos de reloj para su ejecución, podrán generarse errores. De esto se desprende que un procesador CISC no puede utilizar eficazmente su PIPE LINE en la producción de instrucciones.

En la concepción CISC se busca lograr el “salto semántico”, que **es obtener la menor disparidad entre los lenguajes de alto nivel y el lenguaje de máquina**. De existir muchas equivalencias entre sentencias en alto nivel e instrucciones de máquina, el programa traductor entre ambos niveles (compilador) sería más sencillo de fabricar, y los tiempos de compilación disminuirían, que lógicamente son los objetivos de la arquitectura CISC.

Se comprende que esta concepción puede llegar al extremo de fabricar un CISC con instrucciones de máquina que sean equivalentes a sentencias muy usadas en un cierto lenguaje de alto nivel, pero que no se usarán, si se programa en otro lenguaje de alto nivel que no las utiliza.

Consume muchos ciclos de instrucción y es difícil de modificar, porque no es flexible, en donde los tiempos de ejecución son mayores (es más lento)

En otras palabras:

Son instrucciones grandes (muchas instrucciones) y complejas (cada instrucción permite realizar más de una operación, como ser la división, multiplicación, lo cual penaliza al consumo de energía eléctrica y disipación de calor y vida útil). Los ordenadores hogareños si los dejas descargando algo por la noche, quedan calentitos por la mañana.

Todo sistema orientado a ventana tiene un costo, por el manejo de las mismas. Es un programa que corre de backround, donde el procesador trata de dar soporte a todo el manejo gráfico. Pensar en un risk era imposible, es por ello que era necesario un cisc, por su complejidad.

Muchas de las instrucciones disminuyen la brecha semántica entre los lenguajes de alto nivel y la máquina. Muchas de las instrucciones de alto nivel, las tiene implementada en el chip => aumenta el poder. Las instrucciones son complejas por ello; estas son ejecutadas por el assembler, EL COMPILADOR GENERA UN CODIGO ASSEMBLER ups mayus sin querer. Cada máquina tiene su propio lenguaje ensamblador (assembler).

Lograr altas velocidades de ejecución no es fácil => cisc aumenta la superficie de silicio, aumentando las instrucciones.

A la hora de definir una cisc:

Todo es grande, más complejo

Cisc considera que entre más compleja una instrucción le damos un valor agregado a los lenguaje de alto nivel

Incorpora muchos modos de direccionamiento y tan grande como sean posible. Son un montón che. Muchas estrategias le dan un valor agregado y ventajas a los lenguajes de alto nivel

Posee muchos registros de propósito específico. (Una risc se puede manejar con 4 registros nmas, pero a la hora de programar es horrible (seria 10 veces más largo que en una cisc), pero en cisc es un placerrrr)

Control micro-programado. Es la BASE DE TODO WOOOUUUUW.

Maneja el registro instrucción como plantilla.

Características de la Tecnología RISC (Reduced Instruction set computer)

Buscando optimizar la performance de los procesadores, se realizaron estadísticas de las instrucciones de máquina más usadas. Resultó que las instrucciones más simples (que son solo el 20% del repertorio de instrucciones de un procesador CISC) constituían el 80% de los programas típicos ejecutados.

Planificar procesadores con un set de instrucciones simples (operar dos números que están en registros de la CPU, y el resultado asignarlo a uno de esos registros) que presentan muy pocos modos de direccionamiento. Por ser simples, estas instrucciones se ejecutan en un solo pulso de reloj, luego de haber sido decodificadas.

A fin de traducir un lenguaje de alto nivel a este tipo de instrucciones, empleando un mínimo de ellas, se requiere para RISC un programa compilador inteligente, muy elaborado. O sea que es necesario un compilador (software) complejo, como contrapartida de un hardware más simple.

Para mover datos de memoria a registros, y en sentido contrario, fue necesario la existencia de instrucciones que ordenen estos movimientos, que en lenguaje assembler de RISC se llaman LOAD y STORE, respectivamente. Se trata de utilizar estas instrucciones lo menos posible (usando programas compiladores "inteligentes"), pues requieren dos pulsos de reloj para ser ejecutadas, luego que fueron decodificadas.

A diferencia, en un CISC cualquier instrucción tiene la opción de requerir un dato de memoria.

El número total del set de instrucciones de un procesador RISC es reducido (entre 70 y 150 instrucciones, según el modelo).

Debido a que la mayoría de las instrucciones RISC se ejecutan en un pulso de reloj, se obtiene un PIPE LINE eficaz, terminándose de ejecutar en promedio, una instrucción por cada pulso de reloj (en promedio, debido a que las instrucciones LOAD – STORE utilizan dos pulsos). Todas las instrucciones RISC son de formato fijo, siendo que en un CISC ocupan distinta cantidad de bytes. Esto redunda en una mayor sencillez y velocidad de procesamiento.

Por constar de instrucciones cuya fase de ejecución requiere mayormente de un pulso, o a lo sumo dos, no se requiere de una memoria de control que sirva de base para la ejecución del micro-código que debe aparecer a las salidas de la UC con cada pulso de reloj para comandar el procesador.

Esto es, los bits del código de operación de cada instrucción que llega al registro de instrucción RI para ser ejecutados por un procesador RISC, sirven de base para que un circuito de la UC los convierta directamente en la combinación de bits que deben aparecer en las salidas de la UC (micro-código) para que en proximo pulso de reloj se ejecute la instrucción en cuestión.

Por lo tanto, la UC de un RISC no contiene memoria de control (de micro-códigos). Esto, por un lado permite ganar mayor velocidad de procesamiento, pues se evita el acceso a la memoria de control. Por otro lado, beneficia mucho el diseño del chip que contiene un procesador RISC, dado que la superficie que ocupa una memoria de control de un CISC, en un RISC es aprovechada para aumentar el número de registros de uso general de la CPU. Un mayor número de registros permite utilizar menos instrucciones LOAD-STORE, lo cual redunda en menos accesos a memoria principal.

Otra característica de los procesadores RISC es la incorporación de un cache interno para datos y otro para instrucciones, que reducen los accesos a memoria principal. Esta misma idea también fue aprovechada en procesadores de base CISC como el Pentium.

Cada vez que hay que agregar una instrucción a rics, hay que pensar mil veces antes de hacerlo.

A la hora de definir una risk:

Requiere muchas instrucciones de bajo nivel, en comparación a las de alto nivel.

Se necesitan muchas más instrucción para realizar la misma tarea para una instrucción cisc => se discutía en los años ochenta porque la risk ocupaba poco tiempo en los ciclos de computador => mejora el paralelismo gracias a que las instrucciones no monopoliza los ciclos de cpu. Pero para lograr esto debe desechar el concepto de la micro-programación para lograr altas velocidades.

Causa segmentado es cuando el mismo ciclo ya se lanza las micro órdenes para buscar la sig.

Su repertorio es usado por los compiladores optimizadores (analiza el código y ve la posibilidad de reducirlo) porque puede fraccionar, factorizar y re-factorizar este pobre set de instrucción y tiene la posibilidad de mezclar de forma conveniente para que la compilación sea más óptima.

risk necesita tenerla cerca a la ram (en el celular la ram está integrada por ejemplo, son risc modernos!!) al tenerla lejos hay cierta exigencia que la puede hacer más lenta => la unidad cableada es necesaria.

Si vas a diseñar una risk tenes que estudiar todo el universo de las operaciones para ajustar el set de instrucción que va a manejar => es penalizado el sobrante de instrucciones.

Se recarga la ejecución del hardware al compilador => el compilador genera el código mas optimo posible en el mejor tiempo.

Load-store son altamente anaeróbicos.

Existen muy pocos direccionamientos.

Todas las instrucciones tienen longitud fija o de misma longitud.

Comparación entre CISC y RISC

El hecho de que la mayoría de las instrucciones sean de igual complejidad trae aparejado un mejor rendimiento del “pipe-line”. Otro factor que disminuye este rendimiento son la dependencia del resultado de una instrucción para poder ejecutar la siguiente, y un mismo recurso (por ejemplo un registro de la CPU) que es requerido por varias etapas de un “pipe-line”. Estos factores influyen menos en un RISC que en un CISC, resultando así una mayor velocidad de procesamiento.

Por ejemplo, si el resultado de una instrucción es el dato de la siguiente, puesto que la mayoría requiere dos pulsos (fases) para ejecutarse, cuando una instrucción en el pipe-line pasa a la fase de ejecución, las anteriores que entraron al pipe-line ya completaron dicha fase. Solo existe una espera, cuando el dato que opera una instrucción se cargo en un registro desde memoria en la instrucción anterior. Se trata de un pipe-line al cual llegan instrucciones de máquina planificadas por un compilador inteligente.

Los procesadores CISC pierden mucho tiempos en las instrucciones de llamado a subrutina y en las interrupciones, dado los consiguientes accesos la pila de memoria principal que requieren. Las estadísticas indican que en el llamado a procedimientos, el 98% de las veces se utilizan menos de 6 argumentos, y el 92% de las veces menos de 6 variables locales.

Asimismo, que un procedimiento llame a otro, este a un tercero, el tercero a un cuarto, etc., (anidamiento o llamadas sucesivas) sólo se llega a 8 llamadas sucesivas en el 1% de las veces.

Dado que los RISC posee un número elevado de registro estos pueden usarse para el manejo de llamados en lugar de perder tiempo para escribir y leer la pila ubicada en memoria.

Característica	CISC	RISC
Memoria de control	Sí	No
Set de instrucciones	Grande	Reducido
Ciclos de reloj por instrucción	No existe un número definido	Uno o dos
Accesos a MP mediante microinstrucciones	Sí	No, utiliza registros auxiliares. (excepto LOAD – STORE)
PIPE LINE	Ineficiente	Eficiente
Compiladores	Simples	Complejos
Permite modularidad y el crecimiento del set de instrucciones	Sí	No
Formato de Instrucciones	Variable	Fijo
Implementación de paralelismo en la ejecución de las instrucciones	Compleja	Simple
Velocidad de procesamiento	Reducida por los accesos a memoria de control	Optimizada por el paralelismo de ejecución
Predicción de bifurcaciones	No	Sí

Tendencias en la Utilización

Si bien la década de los 80´s y principios de los 90´s estuvieron marcadas por la influencia CISC, debido a que su utilización brindó excelente respuesta a los requerimientos de los usuarios, las mejoras que introdujo la utilización de RISC, en lo referente a velocidad de procesamiento, no fueron descartadas por los diseñadores de microprocesadores.

La muestra más clara de lo referido anteriormente es la fabricación del PENTIUM, un microprocesador básicamente CISC, con características de procesamiento RISC. Este procesador que incluye una memoria de control incorpora también predicción de bifurcación, procesamiento en paralelo, división de la memoria cache en cache de datos y cache de instrucciones. Es, en definitiva, un procesador híbrido, una mezcla de las dos tecnologías.

Esta es la tendencia de los microprocesadores, la creación de híbridos que contengan las ventajas de las dos tecnologías, que durante mucho tiempo parecieron ser contrapuestas y no tener puntos de contacto.

MÁQUINAS DE TURING

Definiciones y ejemplos

Los dos modelos de computación estudiados hasta este punto imponen restricciones considerables a la cantidad de memoria (un AF sólo recuerda su estado actual) o a la forma en que se tiene acceso a ella (un AP sólo puede tener acceso al símbolo de la cima de la pila). Las máquinas en las que se ejecutan estos modelos resultan ser mucho menos poderosas, al menos en principio, que las computadoras reales con las que se está familiarizado.

En este capítulo se estudia una máquina abstracta que propuso el matemático inglés Alan Turing. Aunque todavía parecería muy diferente de las computadoras electrónicas modernas (que no existían cuando Turing formuló el modelo), las diferencias guardan más relación con la eficacia y con la forma de realizar los cálculos que con los tipos de cálculos posibles. El trabajo de Turing y sus contemporáneos sentó gran parte de los fundamentos teóricos de las computadoras modernas.

Turing empezó por considerar una computadora humana (es decir, un ser humano que resuelve de manera algorítmica un problema, con lápiz y papel). Decidió que podría suponerse, sin pérdida alguna de generalidad, que la computadora operaba bajo tres reglas: la primera, que lo único escrito en el papel eran símbolos de algún *conjunto finito fijo*; la segunda, que cada paso dado por la computadora depende sólo del *símbolo que examina actualmente* y de su "*estado mental" en el momento*, y la tercera, que si bien este estado mental podría cambiar como resultado de los símbolos que había visto o los cálculos que había realizado, era posible *sólo un número finito de estados mentales distintos*.

Luego, Turing se dedicó a construir una máquina abstracta que cumpliera esas reglas y pudiera duplicar los que consideraba pasos primitivos que daban las computadoras humanas durante las operaciones de cómputo:

1) Examinar un símbolo específico en el papel.
2) Borrar el símbolo o reemplazarlo con otro.
3) Transferir la atención de una parte del papel a otra.

MT tiene un alfabeto finito de símbolos (en realidad, dos alfabetos, uno de entrada y otro posiblemente mayor para uso durante los cómputos) y un conjunto finito de estados, que corresponde a los posibles "estados mentales" de la computadora humana. En lugar de la hoja de papel, Turing especificó una "cinta" lineal que tiene extremo izquierdo y es potencialmente infinita a la derecha. La cinta está marcada en cuadrados, cada uno de los cuales cabida para un símbolo del alfabeto; si un cuadrado no contiene un símbolo, se afirma que da para un símbolo de espacio en blanco. Por conveniencia, puede considerarse que los cuadrados *están enumerados de izquierda a derecha, a partir de cero*. Si bien esta numeración no es parte del modelo oficial ni resulta necesario hacer referencia a los números para describir el funcionamiento de la máquina. Se piensa que la lectura y escritura se realizan con una cabeza de cinta que en cualquier momento dado está centrada en un cuadrado de la cinta. La versión de máquina de Turing que se usa aquí –la cual es similar a la que propuso Turing, sin ser idéntica a ella– tiene determinado cada movimiento por el estado actual y el símbolo actual de la cinta, además de que el movimiento consiste en tres partes:

1) Reemplazar el símbolo del cuadrado actual con otro, posiblemente distinto.
2) Mover la cabeza de la cinta un cuadrado a la derecha o izquierda (salvo que ya esté centrada en el cuadrado de extrema izquierda, en cuyo caso no puede moverse hacia la izquierda) o dejarla donde está.
3) Pasar del estado actual a otro, posiblemente diferente.

La cinta sirve como dispositivo de entrada (la entrada es simplemente la cadena, que se supone finita, de símbolos que no son el espacio en blanco, en la cinta original), memoria disponible para uso en los cómputos y dispositivo de salida (la salida es la cadena de símbolos que queda en la cinta al final del cómputo). La diferencia más significativa entre esta máquina de Turing y otras más sencillas, estudiadas en capítulos previos, es que en una máquina de Turing el procesamiento de una cadena ya no está restringido a un paso de izquierda a derecha por la entrada. La cabeza de cinta puede moverse en ambas direcciones, además de borrar o modificar cualquier símbolo con el que se tope. *La máquina puede examinar una parte de la entrada, modificarla, dedicar tiempo a ejecutar ciertos cálculos en un área distinta de la cinta, regresar para examinar de nuevo la entrada, repetir cualquiera de esas acciones y, quizás, interrumpir el procesamiento antes de que haya analizado toda la entrada.*

Por razones similares, es posible omitir una tarea que antes realizaban ciertos estados; indicar una aceptación provisional de la cadena leída hasta ese momento. En particular, es posible arreglárselas con dos estados finales o de detención, más allá de los cuales es innecesario que continúe la computadora: un estado h_a, que indica la *aceptación*, y otro h_r para el *rechazo*. Si se pretende simplemente que la máquina acepte o rechace la cadena de entrada, entonces puede moverse al estado de detención apropiado una vez que cuente con información suficiente para tomar una decisión. En el supuesto de que deba llevar a cabo otro cómputo, el estado de aceptación indica que el cómputo terminó normalmente, y el estado h_r puede usarse para indicar una "caída", que resulta de alguna situación anormal en la que la máquina no puede ejecutar su tarea como se esperaba. Sea cual fuere el caso, el computo se detiene si la máquina de Turing llega a cualquiera de los *estados de detención*. Sin embargo -y esto resulta muy importante- también es posible que el cómputo no detenga y la máquina de Turing continúe realizando *movimientos interminables*.

Definición: Una máquina de Turing (MT) es una 5-tupla T=(Q,Σ, Γ,q_0, δ) donde:

Q es el conjunto finito de estados, del cual se supone que no incluye h_a ni h_r, los dos estados de detención (se usan los mismos símbolos para los estados de detención de todas las MT)

Σ y Γ son conjuntos finitos, los alfabetos de entrada y de cinta, respectivamente con Σ ⊆ Γ; Γ se supone que no contiene Δ, el símbolo de espacio en blanco;

q_0 es el estado inicial, es el elemento de Q

δ: Q x (Γ U {Δ}) → (Q U {h_a, h_r}) x (Γ U {Δ}) x {R, L, S} es una función parcial (es decir, posiblemente indefinida en ciertas partes).

En relación con los elementos q ∈ Q, r ∈ Q U {h_a, h_r}, X, Y ∈ Γ U {Δ}, y D ∈ {R, L, S}, se interpreta la fórmula:

$$\delta(q, X) = (r, Y, D)$$

Como indicativa de que cuando T está en el estado q y el símbolo en el cuadrado actual de la cinta es X, la máquina sustituye X con Y en ese cuadrado, cambia el estado a r y mueve la cabeza de cinta un cuadrado a la derecha, lo desplaza un cuadrado a la izquierda (si la cabeza no está en el cuadrado de extrema izquierda) o la deja donde está, según D esté en R o S, respectivamente. Cuando r es h_a o h_r en la fórmula, se dice que T se detiene. Una vez detenida, no puede moverse, puesto que δ no está definida en ningún par (h_a, X) o (h_r, X).

Por último, se permite que la máquina fracase al entrar en el estado de rechazo si intenta mover la cabeza de cinta más allá del extremo izquierdo de la cinta. Esta manera de detenerse de la máquina no se refleja en la función de transición δ. Si la cabeza de cinta está ya en el cuadrado de extrema izquierda; el estado actual y el símbolo de cinta son q y a, respectivamente, y δ (q , a) = (r, b, L), se afirma que la máquina deja la cabeza de cinta donde ésta, sustituye a con b y entra en el estado h_r, no en r.

Esas definiciones y terminología no se han estandarizado por completo. En este análisis, una MT acepta a una cadena si entra tarde o temprano en el estado ha después de empezar con esa entrada. En

ocasiones, la aceptación se define como indicativa de detención (de cualquier tipo), y la única otra forma en que se permite la interrupción del cómputo es mediante un fracaso, porque ya no es posible ningún movimiento. En ambos enfoques, lo significativo es que el observador puede apreciar que la MT detuvo su procesamiento y la razón de que lo haya hecho.

En condiciones normales, una MT empieza con una cadena de entrada x ∈ Σ* cerca del comienzo de la cinta y todos los demás cuadrados de la cinta en blanco. No siempre se insiste en ello, pero siempre se supone que cuando la MT empieza su funcionamiento existe cuando mucho un número finito de símbolos de espacio no blancos en la cinta. Se concluye que ello todavía es válido en cualquier estado del cómputo de la MT. A fin de describir el estado de una MT en, algún ponto, deben especificarse su estado actual, todo el contenido de la cinta (hasta el símbolo situado más ala derecha que no sea espacio en blanco inclusive) y la posición actual de la cabeza. Con ello en mente, se representa la configuración de la MT con el par:

$$(q, x\underline{a}y)$$

Donde q ∈ Q, x e y son cadenas en Γ U {Δ} (una o ambas posiblemente nulas), a es un símbolo en Γ U {Δ} y el subrayado indica la posición de la cabeza. Esta notación se interpreta como indicativa de que la cadena xay aparece en la cinta, que comienza con el cuadrado 0; que la cabeza está en el cuadrado que contiene a, y que todos los cuadrados a la derecha de y están en blanco. En el caso de una cadena no nula w, escribir $(q, x\underline{w})$ o $(q, x\underline{w}y)$ significa que la cabeza está en el primer símbolo de w. Si $(q, x\underline{a}y)$ representa una configuración, entonces será concebible que y termine en uno o más espacios en blanco y también se diría que $(q, x\underline{w}y\Delta)$ representa la misma configuración; pero es usual que cuando se escriba $(q, x\underline{w}y)$ la cadena y sea nula o tenga un último símbolo que no es un espacio en blanco.

Al igual que el caso de los AP, puede trazarse una secuencia de movimientos al mostrar la configuración en cada paso. Se escribe:

$$\left(q, x\underline{w}y\right) \vdash_T (r, z\underline{b}w)$$

Para indicar que T pasa de la configuración de la izquierda a la de la derecha en un solo movimiento, mientras que:

$$\left(q, x\underline{w}y\right) \vdash_T^* (r, z\underline{b}w)$$

Indica que T pasa de la primera configuración a la segunda en cero o más movimientos. Por ejemplo, si T está actualmente en la configuración $\left(q, aab\underline{a}\Delta a\right)$ y δ (q, a) = (r, Δ, L), se escribiría:

$$\left(q, aab\underline{a}\Delta a\right) \vdash_T (r, aa\underline{b}\Delta\Delta a)$$

Las notaciones $\vdash_T$ y $\vdash_T^*$ se abrevian usualmente a $\vdash$ y $\vdash^*$ respectivamente, siempre y cuando no haya ambigüedad.

La entrada se proporciona a una MT mediante la colocación inicial de la cadena de entrada en la cinta, en el cuadrado 1, y la cabeza en el cuadrado 0, que está en blanco. Así pues, la configuración inicial correspondiente a la entrada x es la configuración:

$$\left(q_0, \underline{\Delta}x\right)$$

Ahora es posible definir la manera en que una MT acepta una cadena.

Aceptación por una MT.

Definición: Si T=(Q,Σ, Γ,q_0, δ) es una máquina de Turing y x ∈ Σ*, x es aceptada por T sí, partiendo de una configuración inicial correspondiente a la entrada x, T llega eventualmente a una configuración de aceptación. En otras palabras, x es aceptada si existen y, z ∈ (Γ U {Δ})* y a ∈ Γ U {Δ} tales que:

$$\left(q_0, \underline{\Delta}x\right) \vdash_T^* (h_a, y\underline{a}z)$$

El lenguaje aceptado por T es el conjunto L(T) de cadenas de entrada que acepta T.

Son tres las posibilidades cuando una máquina de Turing procesa una cadena de entrada x. La máquma puede aceptarla, al entrar en el estado h_a rechazarla de manera explícita, si pasa al estado h_r o entrar en un ciclo infinito, de modo que nunca se detiene, sino que continúa en movimiento perpetuo. En cualquiera de los dos primeros casos, el observador aprecia el resultado y sabe si se aceptó la cadena o no. Sin embargo, en el tercer caso, pese a que no se acepta la cadena, el observador no se entera; no se muestra un resultado y queda en suspenso. Por desfavorable que esto parezca, en ocasiones resulta inevitable. En los ejemplos de este capítulo, se construye la máquina de modo que no surja ese problema y cada cadena de entrada se acepte o se rechace de manera explícita.

En muchos ejemplo sencillos, resulta provechoso una vez más trazar diagramas de transición similares a los de los AF, si bien más complejos. El movimiento:

δ(q,X)=(r,Y,D)

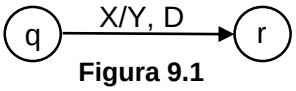


Figura 9.1

(Donde D es R, L o S) se representa como en la figura 9.1

El primer ejemplo debe poner en claro, si todavía no lo está, que esas MT son al menos tan poderosas como los autómatas finitos.

Una MT que acepta {a,b}*{aba}{a,b}* **EJEMPLO 1**

Considere el Lenguaje:

L = $\{a,b\}^*\ \{aba\}\ \{a,b\}^*$ = $\{x \in \{a,b\}^*\ |\ x$ contiene la subcadena $aba\}$

L lenguaje regular, de modo que es posible trazar un AF que reconozca a L, como en la figura 9.2a. No debe sorprender que construir una MT que acepte a L también sea sencillo y, de hecho, sea posible hacerlo de modo que los diagramas de transición se parezcan mucho. La MT se ilustra en la figura 9.2b. Los alfabetos de entrada y cinta son en ambos casos $\{a,b\}$. El estado inicial 0 corresponde en realidad a un estado del AF, ya que la MT no considera ninguna cadena hasta que mueve su cabeza más allá del espacio en blanco inicial.

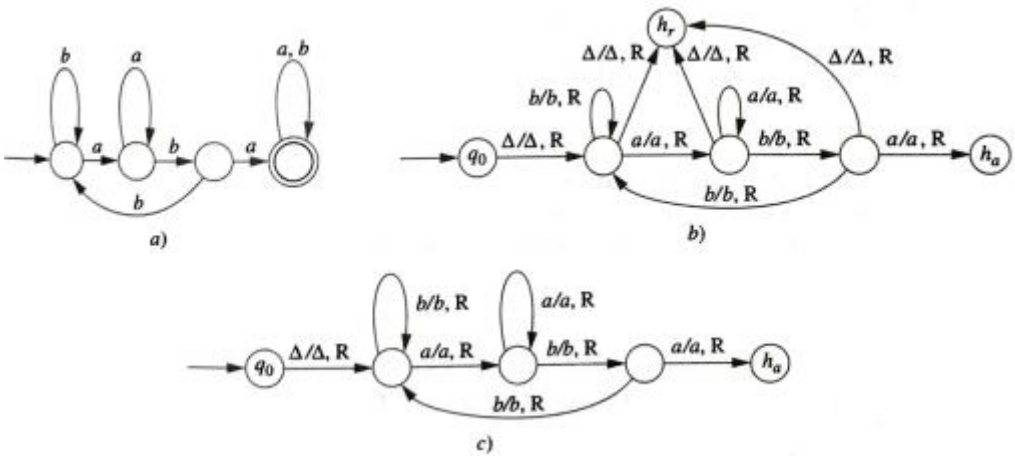


Figura 9.2. Un AF y una MT que aceptan $\{a,b\}^*\ \{aba\}\ \{a,b\}^*$

La figura 9.2b muestra de manera explícita las transiciones que llevan al estado de rechazo h_r, punto en el cual aparece un espacio en blanco (a la derecha de la entrada) antes de la cadena aba. En la figura 9.2c, se ilustra un diagrama simplificado, incluso más similar al diagrama de transición del AF, donde se omiten esas transiciones. Suele ser conveniente simplificar los diagramas de esta manera; siempre que se hace, el diagrama se interpreta como el paso al estado h_r, para cada combinación de estado y símbolo de cinta respecto de la cual no se muestre explícitamente un movimiento. Lo que la MT hace con la cabeza de cinta en un movimiento final de este tipo es, en lo fundamental, arbitrario, ya que ha terminado el cómputo; bien

podría suponerse en general que la cabeza de cinta se mueve a la derecha, como en la figura 9.2b. (Más adelante se analiza la combinación de dos o más MT, de modo que la segunda empieza donde la primera se detiene. En tal caso, lo que la primera máquina hace en su último movimiento no es arbitrario; pero se permite que esa MT compuesta lleve a cabo el cómputo de dos fases sólo si la primera se detiene de manera normal en el estado de aceptación h_a.)

Puesto que se trata de un lenguaje regular, la MT de las figuras 9.2a y 9.2b puede procesar cadenas de entrada de la manera en que se ve forzado a hacerlo un autómata finito, a saber, con el desplazamiento de la cabeza de cinta a la derecha en cada paso, sin que nunca modifique los símbolos de la cinta. Una MT que simule de este modo a un AF puede aceptar cualquier lenguaje regular. Como cabría esperar, este tipo de procesamiento es insuficiente para reconocer un lenguaje no regular.

Observe también que, tan pronto la MT descubre la presencia de aba en la cinta, pasa al estado ha si, con ello, acepta la cadena de entrada completa, pese a que podría no haberla leído en su totalidad. Por supuesto, algunas MT deben leer toda la entrada, incluso si los lenguajes que aceptan son regulares. Por ejemplo, en relación con:

$$L_1 = \{x \in \{a, b\}^* \mid x \text{ contiene la subcadena } aba\}$$

Se muestra un AF y una MT en la figura 9.3. Puesto que la MT mueve la cabeza de cinta a la derecha en cada movimiento, no puede aceptar sin leer el espacio en blanco que está a la derecha del último símbolo de entrada. Al igual que en la figura 9.2c, no se muestran explícitamente las transiciones al estado de rechazo.

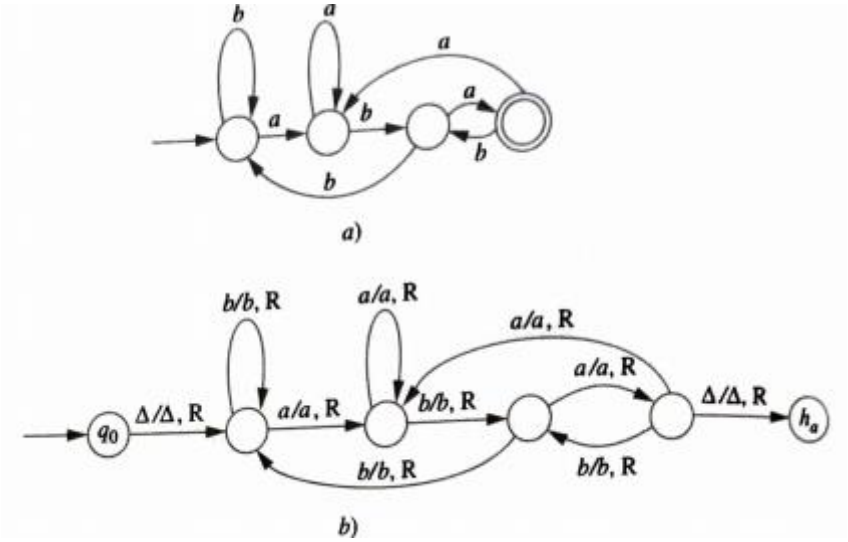


Figura 9.3. Un AF y una MT que aceptan $\{a, b\}^* \{aba\}$

Una MT que acepta a pal. **EJEMPLO 2**

A fin de que aprecie un poco más el poder de las máquinas de Turing a continuación se construye una MT que acepta al lenguaje pal de palíndromos en $\{a, b\}$. Más adeiante, en este mismo capítulo, se introduce la posibilidad de no determinismo en una MT.

Sin embargo, la flexibilidad de las MT permite seleccionar cualquier algoritmo, sin tener que restringirse a una estructura de datos específica, como una pila. Es sencillo formular un enfoque determinista al pensar cómo se verificaría manualmente una cadena larga. Lo primero sería colocar los índices en los extremos. A medida que la vista oscila repetidamente para comparar los dos símbolos de los extremos, los dedos, que son los marcadores, indican a los ojos hasta dónde ir en su desplazamiento gradual hacia el centro. A fin de traducir esto en un algoritmo para para una MT, pueden usarse los cuadrados con espacio en blanco como marcadores en los extremos. Desplazar los marcadores hacia el centro corresponde a borrar

(es decir, cambiar a espacio en blanco) los símbolos recién comprobados. La cabeza de cinta oscila de manera repetida en su comparación del símbolo de un extremo de la cadena remanente sin espacios en blanco con el símbolo del otro extremo. El diagrama de transición se muestra en la figura 9.4. De nuevo, el alfabeto de cinta es $\{a, b\}$, el mismo que el alfabeto de entrada. La máquina toma el trayecto superior cada vez que se topa con una a al comienzo e intenta localizar una a correspondiente al final.

En caso de encontrarse con una b en el estado q_3, de modo que no puede establecer correspondencia con la a del comienzo, entra en el estado de rechazo h_r. (Al igual que en la figura 9.2c, no se muestra esta transición). De manera semejante, ocurre el rechazo del estado q_6 si no puede localizar una b al principio.

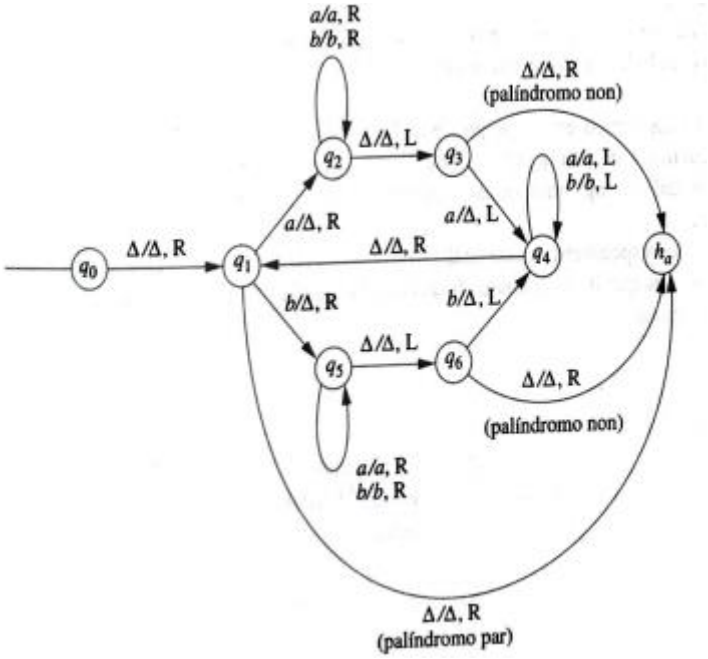


Figura 9.4. Una MT que acepta palíndromos en $\{a, b\}$

Se rastrean los movimientos que hace la máquina para tres tipos diferentes de cadena de entrada, a saber, no palíndromos, palíndromo de longitud par y palíndromo de longitud son:

$$
\begin{array}{llll}
(q_0, \underline{\Delta}abaa) & \vdash (q_1, \Delta\underline{a}baa) & \vdash (q_2, \Delta\Delta\underline{b}aa) & \vdash^* (q_2, \Delta\Delta baa\underline{\Delta}) \\
 & \vdash (q_3, \Delta\Delta ba\underline{a}) & \vdash (q_4, \Delta\Delta b\underline{a}) & \vdash^* (q_4, \Delta\underline{\Delta}ba) \\
 & \vdash (q_1, \Delta\Delta\underline{b}a) & \vdash (q_5, \Delta\Delta\Delta\underline{a}) & \vdash (q_5, \Delta\Delta\Delta a\underline{\Delta}) \\
 & \vdash (q_6, \Delta\Delta\Delta\underline{a}) & \vdash (h_r, \Delta\Delta\Delta a\underline{\Delta}) & \text{(rechazo)}
\end{array}
$$

$$
\begin{array}{llll}
(q_0, \underline{\Delta}aa) & \vdash (q_1, \Delta\underline{a}a) & \vdash (q_2, \Delta\Delta\underline{a}) & \vdash (q_2, \Delta\Delta a\underline{\Delta}) \\
 & \vdash (q_3, \Delta\Delta\underline{a}) & \vdash (q_4, \Delta\underline{\Delta}) & \vdash (q_1, \Delta\Delta a\underline{\Delta}) \\
 & \vdash (h_a, \Delta\Delta\Delta\underline{\Delta}) & \text{(aceptación)} &
\end{array}
$$

$$
\begin{array}{llll}
(q_0, \underline{\Delta}aba) & \vdash (q_1, \Delta\underline{a}ba) & \vdash (q_2, \Delta\Delta\underline{b}a) & \vdash^* (q_2, \Delta\Delta ba\underline{\Delta}) \\
 & \vdash (q_3, \Delta\Delta ba\underline{a}) & \vdash (q_4, \Delta\Delta\underline{b}) & \vdash (q_4, \Delta\underline{\Delta}b) \\
 & \vdash (q_1, \Delta\Delta\underline{b}) & \vdash (q_5, \Delta\Delta\Delta\underline{\Delta}) & \vdash (q_6, \Delta\Delta\underline{\Delta}) \\
 & \vdash (h_a, \Delta\Delta\Delta\underline{\Delta}) & \text{(aceptación)} &
\end{array}
$$

Una MT que acepta a {ss | s ∈ $\{a, b\}$*} **EJEMPLO 3**

En este tercer ejemplo de una máquina de Turing como aceptor de lenguajes, se considera un lenguaje que no es LCL. Sea:

L = {ss | s ∈ $\{a, b\}$*}

La idea subyacente a una MT es separar el procesamiento en dos partes: la primera, encontrar el centro de la cadena y facilitar que la MT distinga a los símbolos de la segunda y la primera mitades; y la segunda, comparar las dos mitades. La primera tarea se logra al trabajar simultáneamente hacia el centro desde ambos extremos, cambiando los símbolos a su versión en mayúsculas conforme avanza. Ello significa que el alfabeto de cinta incluye A y B, además de los símbolos de entrada a y b. Una vez que se llega al centro -lo cual sólo ocurre si la cadena es de longitud par- se cambian los símbolos de la primera mitad a su forma original.

La segunda parte del procesamiento radica en comenzar de nuevo al principio y comparar cada símbolo en minúsculas de la primera mitad contra el símbolo en mayúsculas correspondiente de la segunda mitad. Se lleva un registro del avance al cambiar los símbolos en minúsculas a mayúsculas y borrar los símbolos en mayúsculas correspondientes.

Son dos las formas en que se puede rechazar una cadena de entrada. Si su longitud es non, la MT se da cuenta de ello en la primera fase. En el supuesto de que tenga longitud par y el símbolo de la primera mitad no tenga otro correspondiente en la segunda mitad, la MT rechaza la cadena durante la segunda fase.

La MT correspondiente a este análisis se muestra en la figura 9.5.

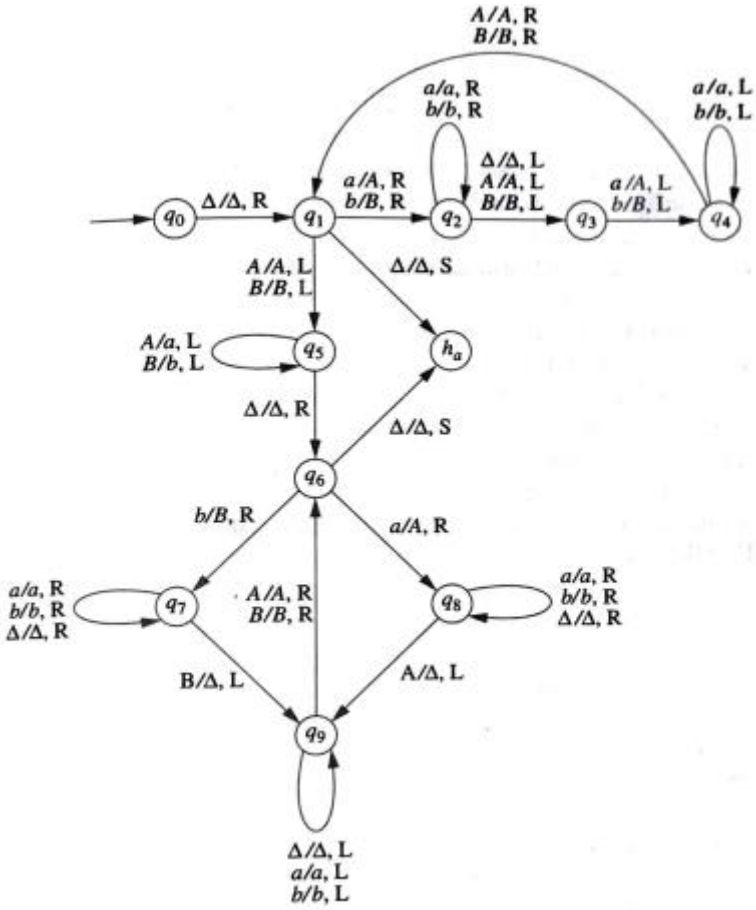


Figura 9.5. Una máquina de Turing que acepta a {ss | s ∈ $\{a, b\}$*}

De nueva cuenta, se rastrean tres cadenas: dos que ilustran ambas formas en las que la MT puede rechazar la cadena y una que está en el lenguaje.

$$
\begin{array}{llll}
(q_0, \underline{\Delta}aba) & \vdash (q_1, \Delta\underline{a}ba) & \vdash (q_2, \Delta\Delta A\underline{b}a) & \vdash^* (q_2, \Delta Aba\underline{\Delta}) \\
 & \vdash (q_3, \Delta Ab\underline{a}) & \vdash (q_4, \Delta A\underline{b}A) & \vdash (q_4, \Delta\underline{A}bA) \\
 & \vdash (q_1, \Delta A\underline{b}a) & \vdash (q_2, \Delta AB\underline{a}) & \vdash (q_3, \Delta A\underline{B}A) \\
 & \vdash (h_r, \Delta AB\underline{A}) & \text{(rechazo)} &
\end{array}
$$

$$
\begin{array}{llll}
(q_0, \underline{\Delta}abaa) & \vdash (q_1, \Delta\underline{a}baa) & \vdash (q_2, \Delta A\underline{b}aa) & \vdash^* (q_2, \Delta Abaa\underline{\Delta}) \\
 & \vdash (q_3, \Delta Abaa\underline{a}) & \vdash (q_4, \Delta Ab\underline{a}A) & \vdash^* (q_4, \Delta\underline{A}baA) \\
 & \vdash (q_1, \Delta A\underline{b}A) & \vdash (q_2, \Delta AB\underline{a}A) & \vdash (q_5, \Delta ABa\underline{A}) \\
 & \vdash (q_3, \Delta AB\underline{a}A) & \vdash (q_4, \Delta A\underline{B}AA) & \vdash (q_1, \Delta AB\underline{A}A) \\
 & \vdash (q_5, \Delta A\underline{B}AA) & \vdash (q_5, \Delta\underline{A}bAA) & \vdash (q_5, \underline{\Delta}abAA)
\end{array}
$$

(Se completa la primera fase)

$$
\begin{array}{lll}
\vdash (q_6, \Delta\underline{a}bAA) & \vdash (q_8, \Delta A\underline{b}AA) & \vdash (q_8, \Delta Ab\underline{A}A) \\
\vdash (q_9, \Delta A\underline{b}\Delta A) & \vdash (q_9, \Delta\underline{A}b\Delta A) & \vdash (q_6, \Delta A\underline{b}\Delta A) \\
\vdash (q_7, \Delta AB\underline{\Delta}A) & \vdash (q_7, \Delta AB\Delta\underline{A}) & \vdash (h_r, \Delta AB\Delta A\underline{\Delta}) \text{ (rechazo)}
\end{array}
$$

$$(q_0, \underline{\Delta}abab) \quad \vdash^* \ldots$$

(Lo mismo que en caso previo, hasta el tercero (movimiento final)

$$
\begin{array}{lll}
\vdash (q_6, \Delta A\underline{b}\Delta B) & \vdash (q_7, \Delta AB\underline{\Delta}B) & \vdash (q_7, \Delta AB\Delta\underline{B}) \\
\vdash (q_9, \Delta AB\underline{\Delta}) & \vdash (q_9, \Delta A\underline{B}) & \vdash (q_6, \Delta AB\underline{\Delta}) \\
\vdash (h_a, \Delta AB\underline{\Delta}) & \text{(aceptación)} &
\end{array}
$$