DIVISION DE EDUCACION CONTINUA

FACULTAD DE INGENIERIA

U.N.A.M

INTRODUCCION A LA COMPUTACION

ELECTRONICA Y PROGRAMACION

José Flores
Ricardo Jerez
Francisco Llamas
Mario Palomar
Alejandro Vega
Benito Zychlinski

Octubre, 1984

TEMARIO

INTRODUCCION A LA COMPUTACION ELECTRONICA Y LA PROGRAMACION*

I. EVOLUCION DE LOS SISTEMAS DE COMPUTO

1. Resumen histórico de la computación
2. Panorama futuro

II. FUNCIONAMIENTO DE UNA COMPUTADORA

1. Sistemas Numéricos
2. Bit, Byte y palabra
3. Códigos de máquina
4. El modelo de Von Neumann

III. EQUIPO (Hardware)

1. Unidad Central de Proceso
2. Unidades de entrada y salida
3. Unidades de memoria auxiliar
4. Sistema de explotación de cómputo

IV. SISTEMAS Y PROGRAMAS (Software)

1. Programación de Sistemas
2. Elementos de Programación
3. Programación Estructurada
4. Lenguajes de programación (BASIC, FORTRAN, COBOL y PASCAL)
5. Análisis comparativo de los lenguajes de programación

V. BANCO DE DATOS

1. Organización de Archivos
2. Bases de DAtos

VI. PROCESAMIENTO DISTRIBUIDO

1. Conceptos Básicos
2. Redes Locales
3. Servicios de Transmisión de datos.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA COMPUTACION ELECTRONICA Y PROGRAMACION

CAPITULO 1
EVOLUCION DE LOS SISTEMAS DE COMPUTO

ING. ALEJANDRO VEGA

OCTUBRE, 1984

INTRODUCCION A LA COMPUTACION

ELECTRONICA Y PROGRAMACION

CAPITULO I

EVOLUCION DE LOS SISTEMAS DE COMPUTO

. Resumen Histórico de la Computación

. Panorama Futuro*

Ing. Alejandro Vega

* Articulos extraidos de la revista "Comunidad Informática", editada por el Instituto Nacional de Estadístistica, Geografía e Informática, numeros 17 y 19.

Octubre, 1984

RESUMEN HISTORICO DE LA COMPUTACION

Orígenes de las Computadoras

El origen de estos dispositivos puede remontarse a la antiguedad, ya que el hombre siempre ha tratado de diseñar mecanismos y dispositivos que le faciliten el trabajo.

Pronto descubrieron nuestros ancestros que los engranes y ruedas podrían usarse para "contar" revoluciones. Se afirma que el griego Hero construyó un primer odómetro. En primer lugar, pueden usarse objetos para contar, piedras por ejemplo. Posiblemente algún pastor primitivo colocaría una pequeña piedra en su bolsa por cada animal de su rebaño que dejara salir en la mañana; en la noche retiraría una de estas piedras, por cada animal que regresara al establo. Y si hubiese quedado una piedra, el pastor sabría que había animales que no habían regresado. Pronto surgió también la idea de usar el "principio de posición" para indicar diferentes cantidades. Poniendo piedras en diferentes lugares podían representarse grupos de uno, de cinco o de diez elementos; esto es el principio del ábaco. En América, los incas usaban ya un sistema para contar llamado quipús. Este era una serie de cuerdas donde la posición de los nudos indicaba la cantidad de cada una de las cosas que se registraban en ellas.

El ábaco representa la primera calculadora mecánica, aunque no lo podemos llamar todavía computadora porque carece de un elemento fundamental, el programa, que no se logrará sino hasta mucho tiempo después.

Con el descubrimiento de los logaritmos hecho por Napier (1550-1617) pronto aparecieron los primeros dispositivos de cálculo analógico: las reglas de cálculo, cuyo funcionamiento está basado en escalas logarítmicas.

Otro ingenio mecánico, que tampoco es una computadora, fue la máquina de calcular inventada por Blaise Pascal (1623-1662). Se trata de una serie de ruedas dentadas en una caja, que entregan resultados de operaciones de suma y resta en forma directa -enseñando un número a través de una ventanita- y que por este simple hecho tiene la ventaja de que evita tener que contar, como en el ábaco; además de que presenta los resultados en forma accesible al ser humano; sin embargo, por problemas mecánicos ésta nunca llegó a trabajar correctamente. Posiblemente la contribución más importante de este siglo al desarrollo de las computadoras modernas, la haya hecho el matemático inglés Charles Babbage (1790-1881), quien estableció los conceptos básicos de una máquina que llamaba diferencial. Estaba diseñada para calcular números, almacenar información y seleccionar diferentes maneras de resolver problemas de acuerdo con el método más eficiente. Preveía ya el empleo de instrucciones operacionales y el de variables. Sin embargo, la ciencia mecánica en esa época no estaba lo suficientemente avanzada como para construir una máquina de la complejidad concebida por este científico. Sus trabajos fueron poco conocidos, al grado de que la mayoría de los investigadores que trabajaron durante la segunda guerra mundial en el desarrollo de computadoras, con frecuencia atacaron problemas que ya habían sido resueltos por este científico inglés.

La aplicación fundamental para la que el gran inventor inglés desarrolló su máquina era obtener tablas de funciones matemáticas usuales (logarit

mos, tabulaciones trigonométricas, etc.) que requerían de mucho esfuerzo manual.

Conceptualmente, el mecanismo era sencillo: evaluar la primera función f1(x1); obtener el nuevo argumento de la serie, x2, y pedir a la máquina que re-calculará la misma función, con el nuevo dato. Claramente, si la máquina "sabe" obtener f1(x1), no será difícil que obtenga f1(x2), y de la misma manera podrá generar toda la serie de valores f1(x1), f1(x2),... f1(xn).

Esta primera computadora "leía" los datos (argumentos) de entrada, por medio de las tarjetas perforadas que había inventado el francés Joseph M. Jacquard, (1752-1834), y que habían dado nacimiento a la industria de los telares mecánicos durante la época conocida como la "revolución industrial".

De esta manera, si se deseaba calcular una segunda función f2 sobre un argumento x1, f2(x1), había que cambiar las especificaciones de f1 por las de f2, lo que, supuestamente, se lograba alterando la disposición de ciertos elementos mecánicos en la sección de control de la máquina.

El americano Herman Hollerith (1860-1929) uso las tarjetas perforadas inventadas por Jacquard para los cálculos censales de los Estados Unidos, inventando un código para los caracteres, que lleva su nombre. En 1944, como resultado de los trabajos de Aiken (1900-1973) apareció el computador Harvard Mark I movido electrónicamente, y cuyas instrucciones y datos se le alimentaban mediante cinta perforada y tenía componentes eléctricos, electrónicos y mecánicos. Fue la primera máquina que presentaba las características de una computadora actual.

Durante la Segunda Guerra Mundial, en los países del "Eje" también se trabajó en el desarrollo de computadoras. "Konrad Zuse" (1910-) construyó el calculador controlado por programa ZUSE-Z3, que al haberse puesto en servicio en 1941 antecedió a la máquina de Aiken.

La primera computadora electrónica denominada ENIAC (Electronic Numerical Integrator and Computer) fue construida en 1946 por J. Eckert (1919-) y J.W. Mauchly (1907-) en la Universidad de Pennsylvania. Esta era la primera máquina totalmente electrónica, capaz ya de multiplicar dos números de diez dígitos en tres milésimas de segundo, (Comparado con los tres segundos que tardaba la máquina Mark 1). Sin embargo, contenía 19000 tubos de vacío, consumía 200 KW y su uso estaba limitado al cálculo de trayectorias balísticas.

El proyecto auspiciado por el Departamento de Defensa de los Estados Unidos, culminó dos años después, cuando se integró a ese equipo el ingeniero y matemático húngaro naturalizado norteamericano. John Von Neumann (1903-1957). Las ideas de Von Neumann resultaron tan fundamentales para el desarrollo de la computadoras modernas, que de hecho a él se le considera como el padre de las computadoras.

Aplicando las ideas del álgebra binaria desarrollada por el inglés George Boole (1815-1864), Von Neumann de la Universidad de Princenton, demostró como emplear lógica y artimética binarias para estructurar programas almacenados. Comprobó que con el mismo lenguaje empleado para codificar un programa se pueden codificar los datos.

La computadora diseñada por este nuevo equipo se llamó EDVAC (Electronic

Discrete Variable Automatic Computer); tenía cerca de cuatro mil bulbos y usaba un tipo de memoria basado en tubos llenos de mercurio por donde circulaban señales eléctricas sujetas a retardos.

La nueva idea fundamental resulta ser muy sencilla: permitir que en la memoria coexistan datos con instrucciones, para que entonces la computadora pueda ser programada de manera "suave", y no por medio de alambres que eléctricamente interconectaban varias secciones de control, como en la ENIAC. Es más, esta idea obliga a un completo reexamen de la "arquitectura" de las computadoras, que recibe desde entonces el nombre de "modelo de Von Neumann". Alrededor de este concepto gira toda la evolución posterior de la industria y la ciencia de la computación.

De 1947 a la fecha las cosas han avanzado muy rápido, más rápido que casi cualquier otro proceso en la historia de la ciencia y la tecnología; a tal grado que hoy en día computadoras mucho más potentes que la ENIAC que ocupan no un sótano completo, sino un circuito de silicio de tamaño tan pequeño que resulta casi invisible a simple vista.

Las Generaciones de Computadoras

Los comienzos de la industria de la computación se caracterizan por un gran desconocimiento de las capacidades y alcances de las computadoras. Así por ejemplo, según un estudio de la época iban a ser necesarias unas veinte computadoras para saturar la capacidad del mercado de los estados Unidos en el campo del Procesamiento de datos...

A partir de estos primeros sistemas, el desarrollo de las computadoras se ha acelerado; una forma conveniente de clasificarlas es en generaciones:

Primera Generación

La primera generación de computadoras estaba caracterizada por el uso de tubos de vacío y por el empleo de diferentes medios para almacenar información. Entre éstos podemos señalar la línea de atraso de mercurio empleada en la máquina UNIVAC-1. Usando desarrollos realizados en la Universidad de Manchester se construyeron los sistemas de almacenamiento electrostático que fueron empleados en los sistemas IBM-701 (International Business Machines). Posteriormente la serie IBM-700 y la serie UNIVAC-1103 usaron sistemas de almacenamiento de tambor magnético, que tenía gran capacidad de almacenamiento, aunque resultaban un tanto lentos. Muchos otros fabricantes fueron adoptando este tipo de memoria. Hacia 1953 se introdujo la memoria de núcleo magnético, desarrollada tanto en los laboratorios de la Radio Corporation of America (RCA), como del Instituto Tecnológico de Massachusetts (MIT), Esta tecnología fue rápidamente adoptada por todos los fabricantes importantes en sistemas de cómputo.

En 1951 aparece la primera computadora comercial, es decir, fabricada con el objetivo de ser vendida en el mercado: la UNIVAC I (Universal Computer) Esta máquina, que disponía de mil palabras de memoria central y podía leer cintas magnéticas, fue usada para procesar los datos del censo de 1950 en los Estados Unidos. Estos son los años de la posguerra, y la nueva invención aún no presagia su gigantesco potencial en la competencia económica internacional, que no llegará sino hasta una década más tarde.

A ésa siguió una máquina desarrollada por una compañía que apenas incursionaba en ese campo: IMB. La IBM 701 (de la que se entregaron 18 unidades entre 1953 y 1956) inaugura la larga serie por venir.

El invento del transistor en 1948, abrió la puerta al desarrollo de una nueva generación de computadoras. Hubo sin embargo, que resolver problemas tecnológicos y de fabricación antes de que este elemento de estado sólido pudiera ser incorporado a los sistemas de cómputo; hasta 1959 comenzaron a aparecer en el mercado las computadoras transistorizadas en cantidades importantes. Todas estas computadoras de segunda generación usaban sistemas de núcleo magnético para almacenamiento de la información, aunque también empleaban discos y cintas magnéticas como dispositivos auxiares de almacenamiento.

A medida que se acercaba la década de los 60's las computadoras iban constantemente evolucionando, reduciéndose de tamaño y aumentando sus capacidades de procesamiento. Al mismo tiempo se iba definiendo cada vez con mayor claridad toda una nueva ciencia: la de comunicarse con las computadoras, y que recibirá el nombre de programación de sistemas.

La segunda generación de computadoras, que se caracteriza por los siguientes aspectos primordiales:

a) Están construidas con circuitos de transistores,

b) Se programan en nuevos lenguajes llamados "de alto nivel",

c) Son de tamaño más reducido, y de costo menor que las anteriores.

En la segunda generación existe mucha competencia y muchas compañías nuevas; cuenta con máquinas bastante avanzadas para su época, como la serie 5000 de Burroughs y la máquina ATLAS, de la Universidad de Manchester.

Entre los primeros modelos se puede mencionar la Philco 212 (esta compañía se retiró del mercado de computadoras en 1964) y la UNIVAC M460. Una empresa recién formada: CDC (Control Data Corporation), produce la CDC 1604, que será seguida por la serie 3000. Estas máquinas comienzan a imponerse en el mercado de las grandes computadoras, como hasta la fecha (con otros nuevos modelos) lo siguen haciendo.

IBM mejora la 709 y produce la 7090 (luego ampliada a la 7094), que gana el mercado durante la primera parte de la segunda generación. UNIVAC continúa con el modelo 1107, mientras que NCR (National Cash Register) comienza a producir máquinas más pequeñas, para proceso de datos de tipo comercial, como la NCR 315.

RCA (Radio Corporation of America) introduce el modelo 501, que tenía un compilador del lenguaje COBOL, para proceso administrativo y comercial. Más tarde introduce el modelo RCA 601.

Tercera Generación

La tercera generación de computadoras está caracterizada, no sólamente por la introducción de circuitos integrados y tecnología de integración a gran escala, sino también por características arquitectónicas importantes como el empleo de bytes de ocho bits para representar caracteres. Con la introducción del sistema IBM 360 en 1964 puede considerarse que se inició esta generación. Este sistema fue sucedido por otro todavía más poderoso el 370. UNIVAC sacó al mercado el sistema 9000 siguiendo los patrones fijados por el sistema 360 de IBM. La CDC introdujo el sistema 6600 en 1964. Esta fue por muchos años la computadora más rápida y poderosa del mercado

y fue sucedida después por el sistema 7600 y finalmente por la serie Cyber.

Otra característica importante de esta tercera generación es la aparición de las llamadas minicomputadoras, representada sobre todo por la serie PDP de Digital Equipment Corporation (DEC). Con el desarrollo de la integración a gran escala, fue posible desarrollar computadoras bastante complejas en uno o dos "chips" semiconductores es entonces cuando aparecen las microcomputadoras.

Como hemos resumido en los párrafos anteriores, a partir de la introducción de la primera computadora y con la llegada de los elementos de estado sólido y después los circuitos integrados, el tamaño de las computadoras se ha reducido drásticamente.

Pero no solamente ha disminuido su tamaño, sino que ha aumentado simultáneamente su capacidad, reduciéndose al mismo tiempo el precio. Por tanto no es de extrañarse que la industria de la computación sea una de las de más rápido crecimiento. Su futuro solamente se ve obscurecido por el aumento en costo que ha sufrido el desarrollo de la programación comparada con la disminución en el costo del equipo.

Cuarta Generación

Durante la década de los 70's, se realizaron constantes innovaciones en la manufactura de circuitos integrados. Se lograron incorporar miles de componentes en un espacio menor a una micra, haciendo la integración de circuitos a gran escala, colocando así a las computadoras en una cuarta

generación, en la que aparece PASCAL, como uno de los lenguajes más poderosos, por ser de aplicación general e incluir los conceptos introductorios de lo que hoy se conoce como programación estructurada.

Micro-computadoras y "computadoras personales"

El avance de la Microelectrónica prosigue a una velocidad impresionante y ya por los años 72-73 surge en el mercado una nueva familia de circuitos integrados de alta densidad, que reciben el nombre de "microprocesadores". Las "microcomputadoras" que se diseñan con base en estos circuitos son extremadamente pequeñas y baratas, por lo que su uso se extiende a mercado de consumo industrial. Actualmente hay microprocesadores en muchos aparatos de uso común, como relojes, televisiones, hornos, juguetes, etc.

Los microprocesadores más usuales actualmente fueron diseñados por dos compañías: el Z-80 de Zilog, y el 6800 de Motorola, aunque el avance en este campo continúa mes con mes. En los últimos años han tenido gran auge los nuevos micro-procesadores, de las familias Z-8000 y 68000.

Las microcomputadoras basadas en estos (y otros) procesadores son de marcas tan diversas como Apple, Canon, Cromemco, Hewlett Packard, IBM, IMS, NEC, Radio Shack y Xerox, entre otras.

Actualmente se habla de las (micro) "Computadoras de uso personal", que son lo suficientemente baratas y accesibles para ser empleadas por pequeñas organizaciones y negocios, donde se encargan de tareas como control de nómina, contabilidad e inventarios. También comienzan a ser de uso

más extendido en aplicaciones "creativas" en computación y como pasatiempo.

Quinta Generación

Actualmente, existen circuitos que tienen alrededor de 260,000 elementos, y se calcula que los adelantos permitirán circuitos mucho más pequeños y cien veces más veloces. Se cree que se ha iniciado una quinta generación con los preparativos para la construcción de una supercomputadora que caracterizará la década de los 80's,

Panorama Futuro

Las nuevas generaciones de estos equipos se caracterizan tanto por avances en los componentes de las computadoras, como pueden ser sus memorias, como en nuevas arquitecturas.

Se preven avances importantes en dispositivos de memoria. Lo más probable es que veamos aplicaciones masivas de las burbujas magnéticas. Dentro de una substancia magnetizada pueden existir burbujas microscópicas varias decenas de miles de ellas.

En realidad son pequeñas partes de sustancias que han sido magnetizadas en una forma diferente al resto; la presencia o ausencia de una burbuja de éstas es empleada para representar un bit de información. Variando rápidamente el campo magnético que rodea a un "chip" las burbujas se mueven alrededor de éste a gran velocidad, existiendo un lugar donde su presencia se detecta. Para leer el bit que una burbuja determinada represen

ta, es necesario esperar que ésta llegue al lugar donde se pueda leer. Esto es más lento que leer información de memoria principal, pero mucho más rápido que emplear discos o cintas. Probablemente se tendrán a mediados de la década de los ochenta,"chips" con 100 MB de capacidad de memoria, aunque hay quien afirme que se alcanzarán hasta 256 MB. Esto, desde luego, es un gran avance si se compara con las burbujas actualmente disponibles que solamente alcanzan entre 64 KB y 1 MB.

El almacenamiento en una burbuja magnética nunca será tan rápido como en "memoria principal". Siempre será más lento, debido al tiempo que tarda la memoria en llegar al punto en donde se detecta; sin embargo, tienen una importante ventaja: si se corta el suministro de energía eléctrica, las burbujas se conservan de manera que la información no se pierde. Desde luego, los discos magnéticos también van a aumentar su capacidad. Posiblemente veamos "disquettes" hasta con 20 MB a fines de la década.

Los dispositivos anteriores representan, respecto a la velocidad de acceso, una situación intermedia entre almacenamiento principal y los discos y cintas. Por tanto,es necesario cubrir este hueco desarrollando dispositivos con velocidad de acceso intermedio. Una de las técnicas que se exploran es el empleo de discos ópticos: éstos tienen pequeños agujeros quemados en su material reflectivo que son detectados con rayos lasser. La presencia o ausencia de un hoyo de éstos representa la presencia de un bit. Estos dispositivos pueden ser muy rápidos, pero una vez que se ha escrito sobre ellos, no pueden ser borrados.

Los dispositivos de almacenamiento mencionados anteriormente no hará desaparecer la memoria de disco. Ya hace más de dos décadas que había perso-

nas que predecían que su potencial había sido agotado. Sin embargo, la densidad de bits en estos sistemas ha aumentado varios cientos de veces en la última década y es posible que siga incrementándose a igual velocidad en la presente. No obstante, la tecnología actual si parece estar llegando a sus límites para un medio de grabado en particular: la cubierta de óxido de hierro, aunque los límites magnéticos teóricos todavía están lejos de ser alcanzados en varios órdenes de magnitud.

Los avances en el desarrollo de circuitos integrados harán seguramente posible que en esta década empiecen a comercializarse grandes computadoras construídas solamente con uno o dos circuitos integrados a gran escala. Estos nuevos circuitos presentan al experto en el diseño de microprocesadores, dos alternativas:

1. Continuar con la tendencia actual en la construcción de circuitos integrados a gran escala diseñando sistemas cada vez más complejos donde el equipo (hardware) realiza funciones que antes realizaba la programación (software).

2. Proceder, como recomienda otro grupo, en la dirección opuesta y construir procesadores más sencillos donde un número mayor de funciones sean realmente realizadas por programación.

Una mayor complejidad permitirá al diseñador usar circuitos integrados cada vez más baratos pero complejos para substituir la cada vez más cara programación. En teoría, estos sistemas tan complejos reducirían el costo del desarrollo de la programación, tendrían un mayor número de funciones integradas al equipo y, por tanto, disminuiría el costo total capita-

lizado de los sistemas.

Sin embargo, hay especialistas tanto en el medio académico, como en los laboratorios de desarrollo que no están de acuerdo y afirman que estas máquinas más complejas ofrecerían poca ganancia tanto en características como en reducción de costo. Ellos proponen al contrario: el empleo de sistemas más simples y por lo tanto más baratos, así como el desarrollo de compiladores más eficientes que optimicen el sistema y simplifiquen el trabajo del programador.

Desafortunadamente, todavía no existen modelos para evaluar el beneficio de una solución respecto a la otra.

Entre las nuevas arquitecturas de computadoras se cuenta ya con procesadores de arreglos cuyo uso se extenderá. Estos procesadores de arreglos son ya de 10 a 100 veces más rápidos que grandes computadoras y velocidades de hasta catorce millones de operaciones de punto flotante por segundo. Gracias a su arquitectura particular, pueden realizar varias funciones diferentes simultáneamente, procesando específicamente vectores, matrices y otros arreglos numéricos. Aplicadas adecuadamente, pueden obtenerse grandes aumentos en la velocidad de procesamiento, por ejemplo, en estudios de flujos de carga se han reportado aumentos de hasta cinco veces en velocidad al emplear procesadores de arreglos en lugar de grandes computadoras.

Como son, en realidad, más que unidades aritméticas memorias de alta velocidad, requieren de una computadora para proveer soporte de entrada/salida y un sistema operativo.

Los circuitos integrados a muy alta escala y las nuevas arquitecturas de las máquinas harán posibles espectaculares avances en la velocidad de las máquinas. Predicciones optimistas estiman que se llegarán a alcanzar velocidades de mil millones de operaciones de punto flotante por segundo (FLOPS) mientras que las máquinas más rápidas actuales alcanzan velocidades entre veinticinco y ciento sesenta millones de FLOPS. Sin embargo, para alcanzar estas predicciones habrá que resolver algunos problemas. No existe todavía un consenso sobre cuál será la mejor arquitectura para estas nuevas máquinas, pero las dos arquitecturas más promisorias parecen ser las máquinas de multiproceso y los sistemas de flujo de datos.

Las primeras son las más flexibles. En ellas un problema es dividido por el programador o compilador de tal manera que el procesador puede realizar los cálculos operando concurrentemente diferentes flujos de datos con diferentes instrucciones.

En los sistemas de flujo de datos, el algoritmo para realizar los cálculos es descrito primero en un lenguaje especial de programación diseñado para aplicaciones de flujos de datos. El programa adquiere con ello la apariencia de una gráfica dirigida que se implanta después directamente en una serie de unidades de equipo (hardware) interconectadas. Cada unidad realiza una sola operación cada vez que los datos llegan a ella.

Otra incógnita es la velocidad máxima que realmente pueden alcanzar estas nuevas máquinas. En cualquiera de estas arquitecturas siempre habrá procesadores que no estén operando debido a conflictos entre memoria y canales de comunicación. La velocidad máxima no será , por lo tanto, la teórica, sino bastante más baja. Las estimaciones sobre su valor todavía

fluctúan grandemente. Para explotar los posibles aumentos en velocidad será necesario:

1. Que todos los procesadores inicien su operación sincrónicamente.

2. Desarrollar algoritmos para organizar la memoria de tal manera que se eviten conflictos.

Cabe preguntar, desde luego, para qué se necesitan tan altas velocidades. Básicamente, para resolver los problemas de simulación en tres dimensiones aplicaciones que requieren de solución de ecuaciones en derivadas parciales con valores en la frontera, como son los de flujos y también los que se presentan en predicción meteorológica, modelado de sistemas complejos y procesamiento de imágenes.

planes de investigación y desarrollo para los sistemas de la quinta generación*

1. ANTECEDENTES

A medida que las técnicas en computación avanzan, la información con computadoras se ha aplicado a diferentes áreas de la producción y se ha convertido en una herramienta indispensable en la sociedad moderna.

Si se desea proporcionar atención a las condiciones y demandas de la sociedad en los años noventa, se requieren funciones de tecnología de información más avanzadas y de más alto nivel; entre éstas se incluyen la utilización de medios más variados, computadoras de fácil uso, mayor productividad en el desarrollo de ***software*** y la aplicación de la tecnología informática en aquellas áreas de tecnología de información donde hasta ahora no se ha aplicado.

Es necesario, para atender a estas necesidades, que la propia filosofía de diseño de la tecnología actual de informática deba estudiarse y evaluarse.

Las computadoras convencionales, diseñadas de acuerdo a la arquitectura de Von Neuman, se construyen con el ***hardware*** más simple debido a que éste era costoso y aparatoso cuando se inventaron las primeras computadoras. La mayoría de las funciones requeridas se implantan mediante el ***software*** para lograr sistemas de procedimiento eficientes. De este modo, las computadoras convencionales están orientadas al proceso numérico; con proceso secuencial de los programas almacenados. Desde el punto de vista económico, se ha buscado mayor rapidez y una capacidad más grande de almacenamiento, dando por resultado los actuales enormes sistemas de macrocomputadoras. Sin embargo, la situación ha evolucionado de la siguiente manera:

1) La tecnología VLSI (integración a muy grande escala) ha reducido substancialmente los costos del ***hardware,*** de manera que éste puede utilizarse en los sistemas de computación en todas las cantidades que sean necesarias.

2) Se requiere de una nueva arquitectura para proceso paralelo, debido a que la velocidad en los dispositivos de proceso secuencial se ha acercado al límite.

3) Deberá usarse proceso paralelo para utilizar con efectividad la producción en masa de los circuitos VLSI.

4) La actual tecnología informática está muy limitada en las funciones básicas para el proceso no numérico de lenguaje, texto, gráficas y patrones, así como para técnicas del campo de inteligencia artificial como inferencia, asociación y aprendizaje.

Es a partir de estas premisas como deberán desarrollarse los Sistemas de Computación de la Quinta Generación (SCQG), que proporcionarán sistemas de proceso de conocimientos.

Los SCQG deberán, por tanto, emplear los últimos resultados de la investigación en tecnología VLSI, así como la relativa para proceso distribuido, ingenierías de ***software*** y del conocimiento, inteligencia artificial y proceso de patrones de información.

Esto nos lleva a concluir que es de gran importancia ejecutar investigaciones y desarrollar los SCQG como una tecnología de información novedosa. Se espera, independientemente de efectuar investigación creativa en este campo, también contribuir a beneficiar a toda la humanidad.

2. TEMAS DE INVESTIGACION Y DESARROLLO

Los Sistemas de Computación de la Quinta Generación están pensados para el proceso de conocimientos con base en las funciones y tecnologías innovadoras de inferencia en un afán por atender las necesidades que se prevén para la década de los años noventa, que incluyen la interacción inteligente entre el hombre y la máquina y la inferencia utilizando bases de conocimientos.

Las funciones requeridas por tal sistema pueden dividirse en cuatro tipos:

1) De solución de problemas e inferencia. Esta función pretende capacitar al sistema para encontrar soluciones a problemas mediante el razonamiento lógico usando conocimientos y datos almacenados en él, así como información proporcionada desde el exterior. Esta capacidad comprende inferencia deductiva, inferencia inductiva, además de la estimación basada en información incompleta, y solución cooperativa de problemas por medio de la complementación mutua de diversos campos del conocimiento.

2) De base de conocimientos. Esta función está dirigida a proporcionar el almacenamiento y recuperación sistemática no sólo de datos, sino también de juicios de congruencia y resultados de pruebas organizadas en un conocimiento.

Además de la acumulación de conocimientos, incluye la representación

* Este trabajo se realizó en el Instituto para la Tecnología de la Computadora de la Nueva Generación, Japón. Traducción del Ing. Alberto Torter Martell.

4. Ahorra horas de clase, las cuales puede dedicar el profesor a mejorar el curso.

5. Hace posible el ahorro de muchas horas de revisión de exámenes, lo que permite al maestro tener tiempo para leer artículos, preparar tareas, atender alumnos, etc.

6. Propicia la individualidad de la evaluación entre los alumnos, evitando la copia.

7. El factor tiempo puede eliminarse de la evaluación si se desea.

8. Hace del profesor un mejor facilitador para el alumno en el proceso de enseñanza-aprendizaje, por no ser él quien lo evalúa.

9. Facilita el examen de un gran número de estudiantes con prontitud en la entrega de resultados y diagnósticos.

Bibliografía

Bloom, B.S., *Taxonomy of Educational Objectives. The Classification of Educational Goals. Handbook I: Cognitive Domain*, David Mckay, Nueva York, 1956.

Krathwohl, D.R., Bloom, B.S. y Masia, B.B., *Taxonomy of Educational Objectives. The Classification of Educational Goals, Handbook II: Afective Domain*, Mckay, Nueva York, 1964.

Green, J.A., *Teacher - Made Tests*, Harper and Row, Nueva York, 1963.

Mosley Donald M., *Teacher - Made Tests*, University of Virginia.

Adkins Wood Dorothy, *Elaboración de Tests. Desarrollo e Interpretación de los Tests de Aprovechamiento*, Editorial Trillas, México, 1976

Milton O., Edgerley J., *"The Testing and Grading of Students"* Change, 1976

Thorndike, Robert L., *Educational Measurement*, American Council on Education, Washington, 1972.

Thorndike, Robert L., Hagen Elizabeth, *Measurement and Evaluation in Psychology and Education*, John Wiley and Sons, Nueva York, 1969

de los mismos enfocada a la solución de problemas, adquisición y actualización de conocimientos y utilización simultánea de fuentes distribuidas de ellos.

3) De interfase inteligente. Esta función está encaminada a proporcionar a las computadoras la capacidad en el manejo del lenguaje hablado, gráficas e imágenes, de tal manera que puedan interactuar con los humanos con flexibilidad y facilidad. Puede decirse que se pretende dar a las computadoras los equivalentes al ojo, boca y oído humanos, empero el objetivo primario es proporcionar a estas máquinas una habilidad lingüística semejante a la del hombre.

4) De programación inteligente. Esta función tiene como fin incrementar la inteligencia de las computadoras, de tal forma que pueda liberar a los humanos del trabajo de programación. En tanto que su objetivo final es lograr la capacidad para convertir automáticamente en programas eficientes los problemas que se le planteen, está enfocada principalmente a lograr sistemas de programación y de verificación modulares y a establecer un lenguaje de descripción de especificaciones.

Para lograr estas cuatro funciones, se requiere desarrollar tecnologías innovadoras en los diferentes campos de arquitectura, ***hardware*** y ***software***. Los principales temas de investigación y desarrollo se enumeran a continuación:

1) Arquitectura ***hardware*** y ***software*** para lograr la función de inferencia. Esto incluirá:

a) Un mecanismo de inferencia basado en la arquitectura de control distribuido que se orienta al proceso paralelo en lugar del proceso secuencial.

b) ***Software*** básico para administrar y ejecutar inferencia paralela.

2) Arquitectura de ***hardware*** y ***software*** para lograr la función de base de conocimientos. Esto consiste en:

a) Mecanismo de base de conocimientos basado en una memoria estructurada, a diferencia de la memoria unidimensional.

b) ***Software*** básico para administrar las bases de conocimientos para una recuperación de alta velocidad y un almacenamiento relacional de los datos de conocimiento.

3) Arquitectura de ***hardware*** ***software*** para lograr la función de interfase inteligente. Esto incluirá:

a) Un mecanismo de interfase inteligente compuesto de un procesador de voz o de señales y otros dispositivos.

b) ***Software*** básico para el proceso de lenguaje natural y comprensión de gráficas e imágenes para asegurar una interacción hombre-máquina flexible.

4) ***Software*** para lograr programación inteligente. Esto constará de:

a) ***Software*** básico para la creación automática de programas óptimos.

Nuevos campos de aplicación

Los sistemas de proceso de conocimientos que se logren con las computadoras de la quinta generación deberán expander ampliamente los campos en los que se aplican, tales como la fabricación, los servicios, la ingeniería y la administración de oficinas y negocios.

Se han seleccionado como modelos para aplicar el ***software*** básico de la

quinta generación, así como para probar y evaluar el sistema básico de ***software:*** el diseño con computadora de circuitos VLSI, la traducción por medio de máquinas y los sistemas de consulta (expertos).

El desarrollo de estos sistemas de aplicación está planteado para las etapas intermedia y final.

3. PLANES PARA INVESTIGACION Y DESARROLLO

3.1 Planes generales

Las metas de la investigación y desarrollo de los Sistemas de Computación de la Quinta Generación son aquellas funciones básicas para el proceso de conocimiento tales como sistemas de solución de problemas e inferencia y sistemas de base de conocimientos que no pueden ser manejados dentro del esquema de los sistemas de computación comerciales.

Hay que acercarse a los sistemas objetivos a través de un largo proceso de prueba y error, produciendo muchas ideas originales en el proceso.

Hasta antes de este proyecto Japón había dedicado pocos esfuerzos a investigación de tecnologías, clave, particularmente en **software** y teorías básicas. El estudio en este campo deberá promoverse, dada su gran influencia en el desarrollo de la tecnología de **hardware,** incluyendo las arquitecturas de computadoras y los circuitos VLSI.

Puesto que este proyecto apunta al desarrollo de la tecnología de computación para los años noventa, es necesario que los planes abarquen una extensión de tecnología básica tan amplia como sea posible. Este proyecto está diseñado para durar 10 años, divididos, como se muestra en la Figura 1, en tres etapas: inicial, intermedia y final.

El hincapié en la investigación y desarrollo en la etapa inicial está en la acumulación, evaluación y reestructuración de los resultados de estudios anteriores en el campo de proceso de conocimiento. Además, se deben seleccionar candidatos para cada tema de investigación y debe desarrollarse la tecnología básica para la etapa intermedia.

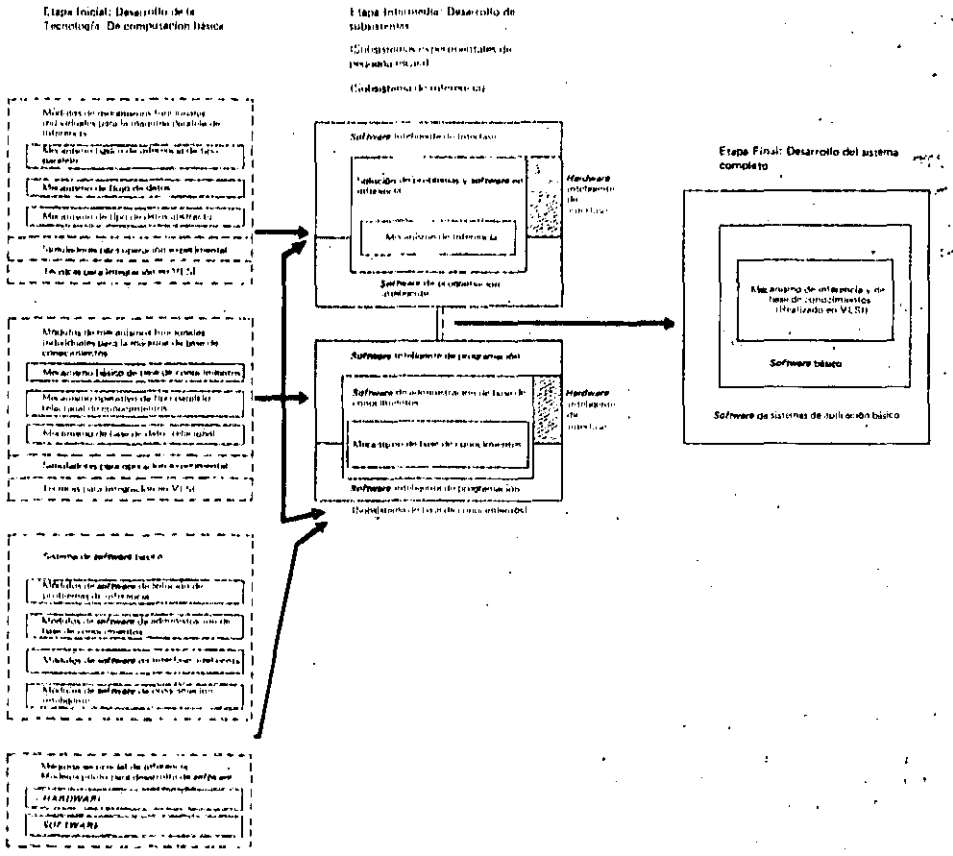
La investigación y desarrollo de la etapa intermedia está enfocada al establecimiento de modelos computacionales como base para el ***hardware*** y el ***software,*** así como algoritmos y arquitectura básica de acuerdo a las evaluaciones de la etapa inicial.

Se construirán subsistemas de escalas mediana y pequeña.

En la etapa final el hincapié se pone en las funciones de ***hardware*** y ***software*** apropiadas, interfases para maximizar esas funciones y arquitectura del sistema total.

Respecto al flujo general de los esfuerzos de investigación y desarrollo, la etapa inicial se ha conceptualizado

Figura 1 Etapas en la Investigación y Desarrollo de la Computadora de la Quinta Generación



de manera que se construyan modelos de ***software*** y ***hardware,*** así como algunos sistemas experimentales configurados con la integración de esos modelos. Estos sistemas incluyen simuladores de ***hardware*** y ***software*** prototipos, para proceso de lenguaje y sistemas experimentales para proceso de lenguaje natural.

La etapa intermedia buscará mejorar y extender los resultados de la etapa inicial e integrarlos en subsistemas de inferencia y de base de conocimientos.

En la primera parte de la etapa final, las configuraciones de los sistemas de ***hardware*** y ***software*** desarrollados durante la etapa intermedia se revisarán y evaluarán. El sistema completo se desarrollará integrando los subsistemas para definir con precisión los objetivos finales.

3.2 Planes de investigación y desarrollo para la etapa inicial

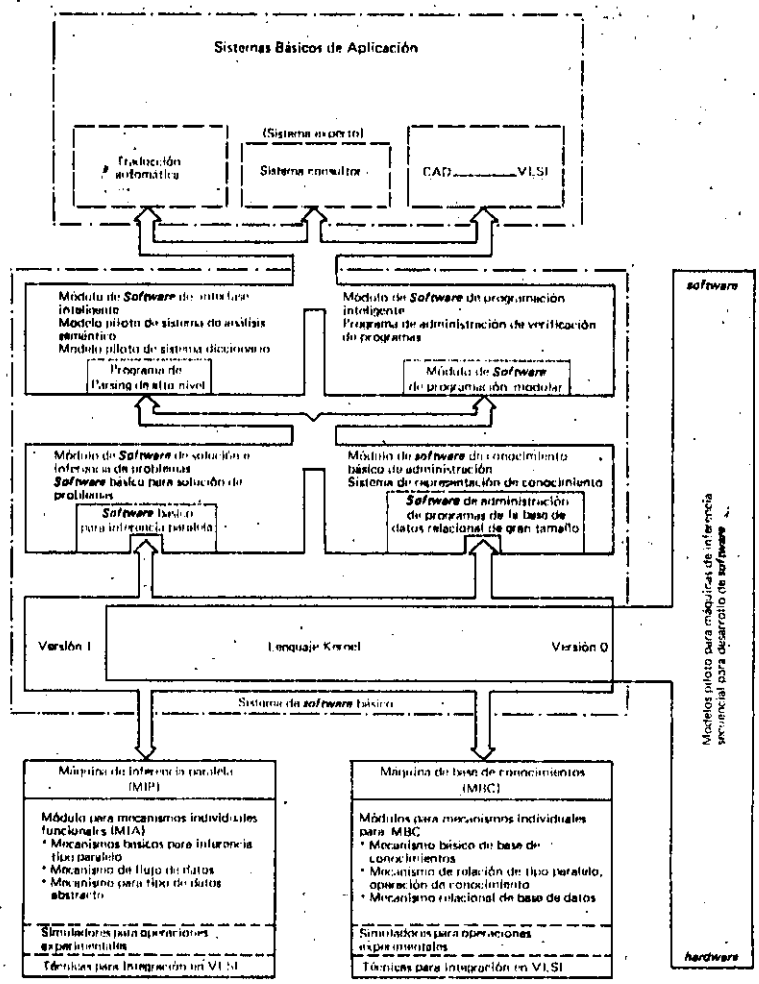
La Tabla 1 describe cada tópico de investigación planeado para la etapa inicial. La Figura 2 muestra una panorámica de la investigación y desarrollo en la etapa inicial indicando las relaciones entre cada tópico de investigación.

La investigación en la etapa inicial del proyecto de los sistemas de computación de la quinta generación se basan en el nuevo lenguaje de programación, la Versión 0 del Lenguaje Kernel, que es una extensión del Prolog. La especificación para la Versión 0 del Lenguaje Kernel se terminó en 1982.

La Versión 0 sirve como el lenguaje de máquina para la máquina secuencial de inferencia, un modelo piloto para el desarrollo de ***software*** y se usa tentativamente para descripción de programas en el desarrollo de ***software***. En tanto que la Versión 0 fue desarrollada para el proceso secuencial, la Versión 1 del lenguaje Kernel se orienta al proceso paralelo. La Versión 1 es un lenguaje de programación lógica apoyado en las esperiencias acumuladas con la Versión 0, con nuevas funciones.

Como se muestra en la Figura 2, la máquina paralela de inferencia es un procesador de alto nivel para ejecutar directamente el lenguaje Kernel, Versión 1. La máquina de base de conocimiento será responsable de la ejecución

Figura 2. Panorama de la Investigación y Desarrollo de la Etapa Inicial



a alta velocidad de las operaciones de conocimiento derivadas del estudio de la representación del conocimiento y de las operaciones de la base de datos relacional.

El ***software*** de la quinta generación incluye dos módulos: uno de solución de problemas y de inferencia para el proceso de problemas y otro de administración y acumulación de conocimientos. Los dos módulos de ***software*** tienen dos niveles jerárquicos. En el nivel inferior se tienen los sistemas de descripción o de soporte a la ejecución para apoyar las funciones del nivel superior. En el nivel superior se tienen el sistema de interfase inteligente, cuyo propósito principal es el proceso del lenguaje natural, y el sistema de programación inteligente, para realizar la programación inteligente, aunque permanece en forma preliminar en la etapa inicial. También puede considerarse que estos dos módulos de ***software*** tienen dos niveles, aun cuando más bien son complementarios y no jerárquicos.

Los sistemas de aplicación elementales en la parte superior de la figura son sistemas medio experimentales, medio prácticos planeados para desarrollarlos en la etapa intermedia de acuerdo con los resultados de la investigación de los sistemas de ***software*** básicos.

Entre éstos, el sistema de consulta (experto) tiene una tecnología bastante bien establecida. Así se ha escogido para probar y evaluar el sistema básico de ***software*** y se planea desarrollar en la etapa inicial una versión preliminar, llamada Sistema Experimental de Base de Conocimientos. Se considera este desarrollo como un tema adicional a la Tabla 1.

TABLA 1

Temas de investigacion y desarrollo para la etapa inicial

tema	descripción
Máquina paralela de inferencia (MAPI)	La máquina paralela de inferencia, junto con la máquina de base de conocimientos, forma el núcleo del ***hardware*** de la computadora de la quinta generación. En la etapa inicial debe hacerse una evaluación y estudio de la configuración del modelo básico de inferencia, compuesto de lo siguiente: (1) Mecanismo básico de inferencia de tipo paralelo para administrar la ejecución paralela de operaciones de inferencia. (2) Mecanismo de flujo de datos para ejecutar operaciones de inferencia y determinar soluciones rápidamente. (3) Mecanismo de tipo de datos abstracto para consolidar operaciones de inferencia detalladas en diversos grupos y controlarlas por grupo.
Módulos para mecanismos funcionales individuales para MAPI	El mecanismo básico de inferencia de tipo paralelo, el mecanismo de flujo de datos y el mecanismo de tipo de datos abstractos consistirá individualmente de submodelos funcionales. Al principio, se construirán prototipos de estos submodelos. Después, se combinarán los prototipos de los submodelos para constuir un módulo prototipo de cada uno de los tres mecanismos funcionales.
Simuladores para la operación experimental	Los simuladores prototipo para la operación experimental se construirán para reproducir las configuraciones de los módulos, usando cantidades y combinaciones diferentes de los submodelos. También serán usados para determinar la configuración óptima de los módulos de los tres mecanismos funcionales y también del módulo básico de inferencia que estará compuesto de estos submódulos.
Técnicas para integración en VLSI	El ***software*** prototipo se desarrollará para evaluar y examinar la convertibilidad a VLSI de los circuitos componentes de cada submódulo diseñado. Se usará en la recopilación de datos y en la evaluación para integración en VLSI.
Máquina de base de conocimientos (MBC)	La máquina de base de conocimientos, junto con la máquina paralela de inferencia forma el núcleo del ***hardware*** de la computadora de la quinta generación. En la etapa inicial se realizará un estudio de evaluación respecto a la configuración del módulo básico de base de conocimientos, compuesto de lo siguiente: 1) Un mecanismo para proporcionar administración general de la ejecución de las operaciones básicas de la base de conocimiento. 2) Un mecanismo de operación de relaciones de tipo paralelo para proporcionar rápida acumulación de conocimientos, recuperación y actualización, conversión de datos, etc. 3) Un mecanismo de base de datos relacional para proporcionar una gran capacidad de acumulación, almacenamiento y administración de conocimientos.
Módulos para mecanismos funcionales individuales para MBC	El mecanismo básico de base de conocimientos de tipo paralelo, el mecanismo de operación de conocimiento y el mecanismo de base de datos relacional consisten, individualmente, de submódulos y se construirán en la etapa inicial. Estos submódulos prototipo se combinarán posteriormente para producir un módulo prototipo para cada uno de los tres mecanismos funcionales.
Simuladores para operación experimental	Los simuladores prototipo para pruebas operativas se construirán para reproducir las configuraciones de los módulos usando cantidades y combinaciones diferentes de los submódulos. También serán usados para determinar la configuración óptima de los módulos para los tres mecanismos funcionales y, asimismo, para el módulo básico de base de conocimiento que constará de estos submódulos.

Técnicas para integración en VLSI

El ***software*** prototipo se desarrollará para evaluación y examen de la convertibilidad a VLSI de los circuitos componentes de cada submódulo diseñado. Será usado para recopilación de datos y evaluación para integración en VLSI.

Sistema básico de *software*

Este sistema forma el núcleo de la computadora de la quinta generación y se compone de los siguientes cuatro módulos de software para el proceso de información de conocimientos.

1.- Módulo de solución de problemas e inferencia.

2.- Módulo de administración de base de conocimientos.

3.- Módulo inteligente de interfase.

4.- Módulo de programación inteligente.

Se desarrollará un lenguaje final extendido que se requiere para la etapa intermedia organizando el conocimiento obtenido mediante el diseño y construcción del sistema de ***software*** básico.

Además, se producirá un sistema de ***software*** prototipo para probar la calidad de las especificaciones y validar su exactitud.

Módulo de *software* de solución de problemas e inferencia

El módulo de ***software*** de solución de problemas e inferencia deductiva, inferencia inductiva, incluyen la proposición de conjeturas que se apoyan en información incompleta e inferencia por complementación mutua de conocimientos.

Se planea el desarrollo de un prototipo de ***software*** básico para inferencia paralela en la primera etapa y usarlo en la rápida ejecución de inferencia deductiva, y de ***software*** básico para solución de problemas y determinación de soluciones eficientes a los problemas.

Módulo de *software* de administración de la base de conocimientos.

Este módulo tiene la capacidad de acumular, usar fuentes distribuidas y adquirir conocimientos. En la etapa inicial se ha planeado el desarrollo de un prototipo de un sistema de representación de conocimientos para poder definir métodos de representación de conocimientos. También se ha planeado un programa de administración de grandes bases de datos de tipo relacional para acumular y administrar un gran volumen de información representada como conocimiento.

Módulo de *software* inteligente de interfase

Este tipo de módulo está pensado para una interacción flexible entre el humano y la computadora.

En la primera etapa se planea el desarrollo de un prototipo para un programa de ***parsing*** de alto nivel y de algoritmos simplificados para la comprensión del lenguaje natural, que es vital para la interacción hombre-máquina. También se desarrollarán tecnologías básicas para el análisis semántico y un modelo piloto de un sistema de soportes de diccionario.

Módulo de *software* inteligente de programación

El módulo de ***software*** inteligente tiene la capacidad de conversión automática de un problema de entrada en un programa de computadora eficiente (a nivel de lenguaje Kernel).

Se planea desarrollar en la primera etapa un sistema de administración de módulos de programa con la capacidad de extracción de módulos y de verificación de las capacidades de un programa con el objeto de establecer programación modular, que es esencial para la programación inteligente, la extracción del programa necesario y la verificación del programa preparado.

Máquinas secuenciales de inferencia

Se desarrollará un modelo piloto de un prototipo de una máquina secuencial de inferencia para el desarrollo eficiente del ***software*** para los sistemas de computación de la quinta generación.

Modelos piloto para el desarrollo de *software*

Este modelo se desarrollará mejorando el lenguaje seleccionado para la inferencia y modificando parcialmente la actual arquitectura de Von Newman.

microelectrónica e implicaciones *

ing. alberto torfer martell**

La industria microelectrónica

La industria microelectrónica aparece en 1947, cuando en los laboratorios Bell, de los Estados Unidos, los investigadores John Bardeen, Walter Brattain y William Shockley desarrollan el transistor.

Diez años más tarde, la compañía norteamericana Fairchild construye el primer transistor plano; posteriormente esta empresa, junto con Texas Instruments, fabrica las primeras pastillas con dos o más transistores en un mismo sustrato de silicio para sistemas de control de proyectiles y para computadoras. A partir de entonces la densidad, esto es, el número de elementos por unidad de superficie, fue duplicándose casi cada año (Ley de Moore). Aparecieron así los circuitos con integración de pequeña, mediana y gran escala *(SSI, MSI, LSI)*, luego lo hicieron los de muy grande y ultragran escala *(VLSI, ULSI)*.

A finales de la década de los sesenta se le pidió a la compañía Intel que desarrollara una familia de circuitos integrados para usarlos en una serie de calculadoras de diferente capacidad. Los ingenieros de la empresa decidieron que, en lugar de crear un circuito integrado diferente para cada modelo de calculadora, sería más sencillo diseñar uno de propósito general, que pudiera usarse en todos los modelos y de esta manera definir las diferentes capacidades en un conjunto de pastillas de memoria *(ROM)*. Así fue como en 1971 nació el microprocesador que integra, en una sola pastilla de silicio, la unidad central de proceso (UCP), la unidad lógica y la unidad aritmética.

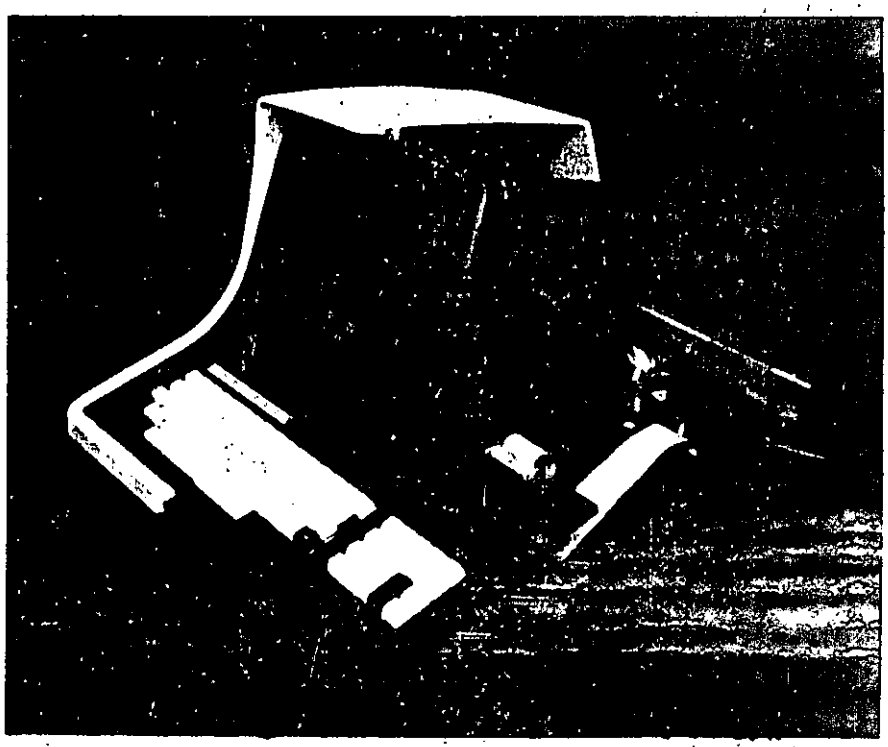
En 1975 la misma compañía fabricó una microcomputadora de 8 *bits* en una sola pastilla; dos años más tarde apareció una memoria de 16 KB igualmente en una pastilla. En la actualidad se tienen micropocesadoras de 32 *bits* con varios miles de transistores (más de 100 mil).

La evolución de la electrónica digital ha incrementado la velocidad, confiabilidad y complejidad de los dispositivos, incluyendo en ellos cada vez más funciones del campo de la electrónica analógica. Es decir, el incremento de digitalización y la capacidad de transformar señales analógicas y digitales, y viceversa, permiten el acoplamiento de actividades que anteriormente, aunque independientes, interactuaban entre sí, lo que es la bases de la transformación de los productos y los procesos productivos.

Los avances en materia de integración de sistemas, diseño *VLSI (Very Large Scale Integrated)* y tecnología de memoria serán la base para los sistemas de gran capacidad del futuro.

Entre los productos elaborados recientemente pueden mencionarse:

- Microcomputadora con memoria *EEPROM*
- Convertidor analógico digital de 10 *bits*
- Memoria *RAM* dinámica de 256 *KB*
- Procesador para reconocimiento de voz



* Tomado de *Mexicon 83, Memoria de la XI Conferencia Internacional de Investigación, Desarrollo y Aplicaciones de la Ingeniería Eléctrica y Electrónica*, México, noviembre de 1983.

** Subdirector de Investigación y Desarrollo, Instituto Nacional de Estadística, Geografía e Informática.

- Pastilla para teléfono, que reemplaza casi todas las partes electrónicas del mismo.
- Controlador de Red Ethernet
- Láseres de luz visible
- Circuito integrado para controlar una terminal gráfica alfanumérica
- Supermicrocomputadora en cinco pastillas.

El estado actual de la tecnología microelectrónica se caracteriza por diversos aspectos, entre los que destacan los siguientes:

- Los circuitos se fabrican en forma integrada en un solo proceso, así que es imposible separar los componentes, aun cuando cada uno realice funciones individuales.

- La integración elimina muchas conexiones y empaca juntos a los circuitos, haciéndolos más confiables y acortando el tiempo de transmisión de las señales eléctricas.

- El proceso permite agrupar cientos o miles de componentes en un sustrato que generalmente es de silicio, aunque puede ser de germanio o de arseniuro de galio, lo que posibilita colocar circuitos complejos en una sola pastilla. Consecuentemente, el precio por cada función o elemento es muy bajo y decrece a medida que el nivel de integración o densidad aumenta y la producción se eleva.

Se han alcanzado estos altos niveles de integración de tres formas diferentes:

- Reduciendo el producto a sus dimensiones mínimas, con lo que resultan elementos con circuitos de mayor densidad.

- Reduciendo la frecuencia de defectos, al mejorar las técnicas de proceso; esto permite la producción de obleas y circuitos de mayor área.

- Mediante innovaciones en las formas de los circuitos, lo que permite una densidad funcional mayor.

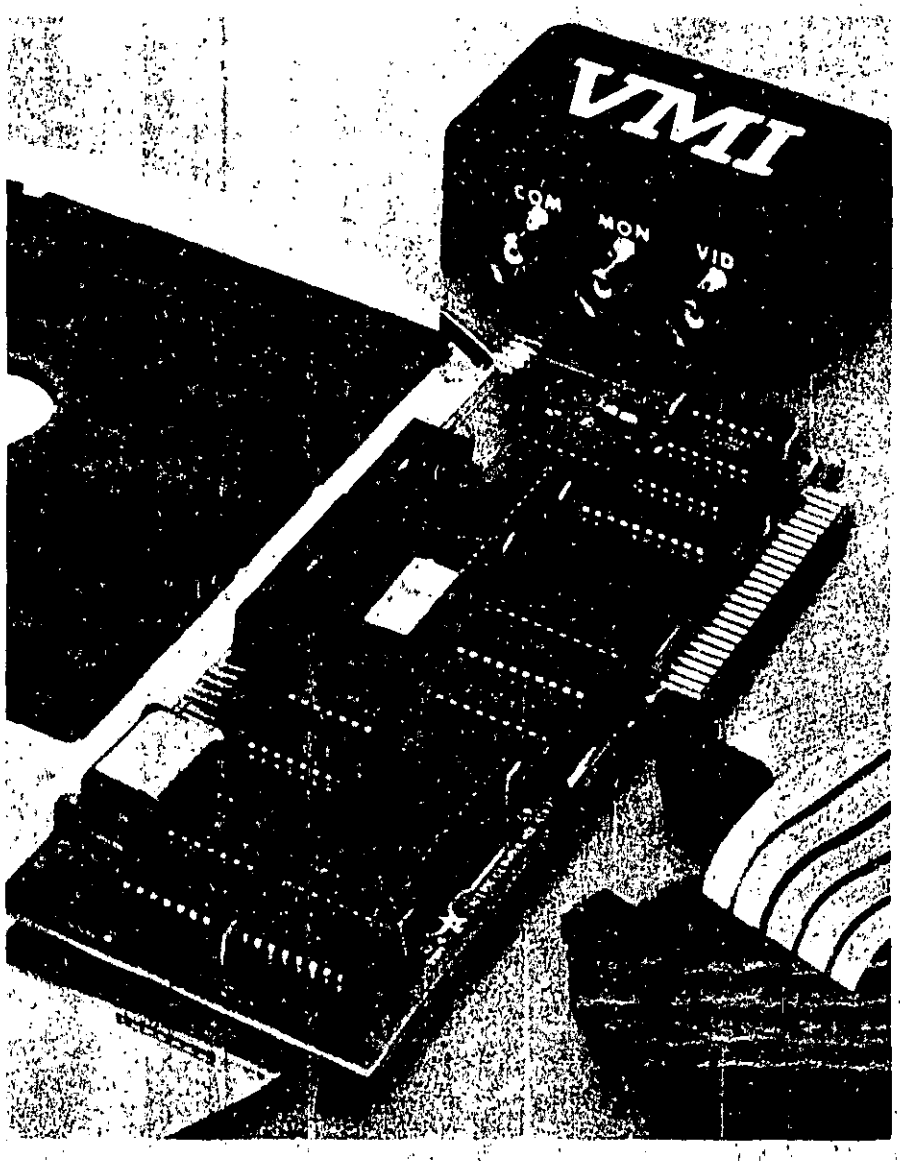
Para aprovechar al máximo la microelectrónica, es necesario integrar con rapidez y bajo costo circuitos que realicen diversas funciones.

Actualmente la mayoría de las tarjetas tienen un pequeño número de circuitos *LSI*, rodeados de un gran número de los de los tipos *SSI* y *MSI*, llamados circuitos de liga, que individualizan la tarjeta para cada aplicación. En tanto que los circuitos *LSI* realizan aproximadamente el 80 por ciento de las funciones, los de los tipos *SSI* y *MSI* ocupan el 80 por ciento del espacio y consumen ese mismo porcentaje de energía.

Resulta, por lo tanto, muy interesante para la industria el desarrollo de circuitos *VLSI* que realicen toda la formación de una tarjeta en una sola pastilla, tanto por ventajas de densidad y desempleo como de costos.

La tecnología de fabricación ha permitido circuitos cada vez más complicados y completos, gracias a la mayor resolución y precisión con que pueden trabajar las máquinas de exposición de mascarillas.

Aun cuando tecnológicamente es posible elaborar circuitos de mayor densidad, en la actualidad el problema principal de la industria se encuentra en el área de diseño y prueba. Según el procedimiento tradicional, es mayor el tiempo que se requiere para diseñar un componente que la vida útil que éste pueda tener. Además, diseñar una pastilla con 100 mil compuertas ló-



gicas cuesta cada una, en promedio, 100 dólares.

Para resolver este problema se han considerado posibilidades que prometen no sólo resolver la crisis, sino modificar también los esquemas productivos. Por un lado, gracias a las nuevas herramientas, los diseños podrán realizarse en semanas, en lugar de años. Por otro, los usuarios se verán involucrados en el diseño de las pastillas, en virtud de que sería muy difícil para el fabricante conocer a fondo todos los posibles campos de aplicación.

Se han logrado algunos avances en materia de especialización de circuitos. Así, se desarrollaron primeramente la memoria *ROM (Read Only Memory,* memoria únicamente de lectura) y las microcomputadoras en una sola pastilla. Al principio eran elementos separados, pero al integrarse con el nuevo procesador, apareció la microcomputadora.

Se han desarrollado varios tipos de *ROM;* entre ellos el *PROM,* el *EPROM* y el E^2 *PROM.* El *PROM* es un circuito que puede programarse fácilmente para realizar tareas sencillas; pero una vez programado no es posible alterarlo. El *EPROM* puede borrarse con luz ultravioleta y el E^2 *PROM* con corriente eléctrica. Aunque son muy versátiles, algunas veces resultan inconvenientes si se requiere una rápida ejecución.

Es por medio de arreglos de compuertas como puede consolidarse la lógica necesaria para una función, por lo que frecuentemente se usan junto a las microcomputadoras. También pueden emplearse para integrar toda una aplicación.

Los arreglos de compuertas, también llamados arreglos lógicos no comprometidos o arreglos lógicos programables *(PLA,* según su nombre en inglés), tienen un numero fijo de compuertas lógicas. Las obleas se preprocesan, excepto en los tres últimos niveles, que son los que especifican la conexión requerida por cada aplicación especializada. Cada vez, con mayor frecuencia, este patrón de conexiones se determina por medio de una computadora siguiendo un proceso llamado colocación y direccionamiento automatizado.

Las celdas estándar son bloques funcionales muy compactos y complejos; por ejemplo, compuertas lógicas, memorias y hasta elementos de proceso, que se obtienen de una biblioteca y se combinan para cumplir con las especificaciones de una determinada aplicación. El acomodo de las celdas, así como sus interconexiones, puede realizarse mediante una computadora de acuerdo con programas llamados ensambladores.

Hoy por hoy los circuitos pueden definirse en función de cada necesidad específica, por medio de los compiladores de silicio. La definición se hace de una manera jerárquica, describiendo primero un plano general de distribución de área y después los bloques lógicos más pequeños hasta llegar a los componentes básicos. Esta información permitirá al compilador duplicar los componentes básicos, si lo necesita, para obtener los subsistemas. También resuelve las conexiones entre celdas y bloques.

Se prevé que los usuarios de la microelectrónica definan sus propios circuitos con este tipo de herramientas.

El diseño de un circuito *VLSI* incluye las siguientes fases:

- Especificación funcional (diagrama de bloque funcional, revisión de registro-trasnferencial).
- Diseño lógico (esquemas lógicos, revisión a nivel compuerta).
- Diseño eléctrico (esquema eléctrico, revisión a nivel circuito, verificación de tiempos).
- Diseño de mascarillas (mascarillas, revisión de reglas de diseño).
- Desarrollo del programa de prueba.
- Comprobación del circuito.

En forma gradual, todas estas funciones se están trasladando a sistemas de diseño mediante computadoras. Eventualmente será posible describir un sistema a un nivel muy elevado de abstracción y obtener en forma automática un circuito *VLSI.*

Dado el costo relativamente bajo de estas fases, se espera que el mercado de usuarios que diseñen sus propios circuitos se incremente de manera significativa en los próximos tres años.

Para el diseño se pueden tener compiladores de silicio o ensambladores. Estos últimos se usan con los sistemas de celdas estándar, donde se seleccionan aquéllas disponibles en una biblioteca, de acuerdo con las necesidades. El compilador de silicio es más ambicioso, ya que pretende partir de una definición de alto nivel para manipular los elementos predefinidos, con el fin de lograr la función especificada.

Los compiladores de silicio se usaron inicialmente en macrocomputadoras, pero una amplia variedad de proveedores de servicios informáticos los están trasladando a estaciones de trabajo con manejo de bases de datos, ya sea por medio de microcomputadoras o con estaciones conectadas a grandes computadoras, donde también se realizan las simulaciones requeridas.

Implicaciones

La revolución microelectrónica afectará no sólo a los países desarrollados, sino también a aquellos en vías de desarrollo.

En resumen, podemos señalar que de acuerdo con sus efectos tiene, entre otras, las siguientes características.

- Mejora y/o sustituye a una amplia variedad de habilidades manuales e intelectuales.
- Puede reemplazar a una amplia variedad de dispositivos de control eléctrico, mecánicos, neumáticos o hidráulicos.
- Por lo tanto, consigue reemplazar a una gran variedad de dispositivos y servicios actuales, o au-

mentar sus capacidades, creando nuevos productos y servicios.

- Es confiable, redituable y económica.

Por todo ello, la microelectrónica constituye un gran potencial que puede ser tanto benéfico como perjudicial. Sobre todo, resultará perjudicial para los países que no controlen esta tecnología, debido a que proporcionará poder económico a aquellos que sí la dominen.

Para analizar estos efectos, consideramos a la microelectrónica desde cuatro puntos de vista diferentes:

1) Como industria, 2) como una herramienta en la nueva era de la información, 3) su aplicación en otros sectores industriales y 4) otras posibles aplicaciones.

Vista como una industria, la microelectrónica consta de tres partes: 1) montaje y empaque, 2) fabricación de circuitos integrados y 3) diseño.

México ha atraído y formado una industria microelectrónica de montaje. Su futuro se ve amenazado por la creciente automatización que la misma microelectrónica ha hecho posible, haciendo regresar a sus países de origen a aquellas industrias que han venido buscando mano de obra barata. Así, este sector tiene un futuro incierto.

En el área de fabricación de circuitos integrados se tiene el problema del bajo mercado nacional, la limitada labor de diseño electrónico digital y los altos costos de investigación y desarrollo. En cuanto al diseño, casi no hay ninguna industria en México que se dedique a éste, por lo que se estima poco probable impulsar una industria derivada de la integración, interfase o instalación de *hardware* inteligente para aplicaciones específicas.

Al analizar el uso de la microelectrónica en la nueva era de la información se encuentra un serio retraso con respecto de países desarrollados. En éstos representan el 55 por ciento de la fuerza de trabajo, en tanto que en las economías subdesarrolladas representa únicamente el 20 por ciento.

La nueva era de la información depende en gran medida de la infraestructura de comunicaciones, que si bien está desarrollada en México, aún no alcanza los niveles de los países altamente industrializados.

El uso de la informática puede contribuir decisivamente a mejorar la productividad y la toma de decisiones en las operaciones de empresas de servicio, en la administración, en las instituciones de salubridad y en las educativas.

Con respecto al efecto de la microelectrónica en otros sectores industriales, puede señalarse que gracias a ella es posible lograr las siguientes aplicaciones:

- Sustitución de lógica alambrada, por ejemplo, en equipo de control eléctrico.
- Sustitución de dispositivos mecánicos, como es la máquina de escribir.
- Transformación de la elaboración de un producto completo; por ejemplo, los relojes.
- Aumento de las funciones de un producto industrial; por ejemplo, instrumentos científicos.
- Creación de nuevos productos, como son los videojuegos.

Así, la microelectrónica será parte de los bienes de capital y de consumo, dificultando aún más la posibilidad de que los países en vías de desarrollo alcancen a los desarrollados, puesto que ahora no sólo se tendrá que dominar el proceso de fabricación del bien de capital, sino también el de microelectrónica para construir los circuitos integrados requeridos por el sistema específico.

Al no dominar la microelectrónica, los bienes de capital y de consumo de los países en vías de desarrollo serán obsoletos y no podrán competir con los del exterior.

Por otro lado, la microelectrónica puede reducir las piezas para determinados productos, haciendo innecesario su montaje en países que únicamente aportan mano de obra barata.

En los procesos productivos, este logro técnico puede emplearse para:

- El movimiento controlado de materiales, componentes y productos.
- El control de variables como temperatura, humedad o presión.
- Los procesos de formado, corte, mezcla y moldeo de materiales.
- El ensamblado de productos y subproductos.
- El control de calidad por medio de la inspección, el análisis o las pruebas.
- La organización del proceso de manufactura, incluyendo diseño, inventarios, envíos, mantenimiento de maquinaria, facturación y asignación de tareas.

De esta forma, el empleo de la microelectrónica en los procesos productivos abaratará el costo de los bienes y servicios producidos mediante los sistemas modernos, incrementando la competencia internacional por los mercados.

La microelectrónica para aplicaciones especiales afectará de las siguientes formas:

- Reducirá los requerimientos energéticos para fuerza motriz y calefacción
- Posibilitará las minifactorías con equipo industrial autocontrolado, autoajustado y autodiagnosticado.
- Facilitará la educación, comunicación y diversión
- Se empleará en la agricultura.
- Se utilizará en máquinas biotecnológicas.

Todo lo anterior requiere de una combinación de capacidades y esfuerzos que ahora son insuficientes en los países en vías de desarrollo.

Conclusiones

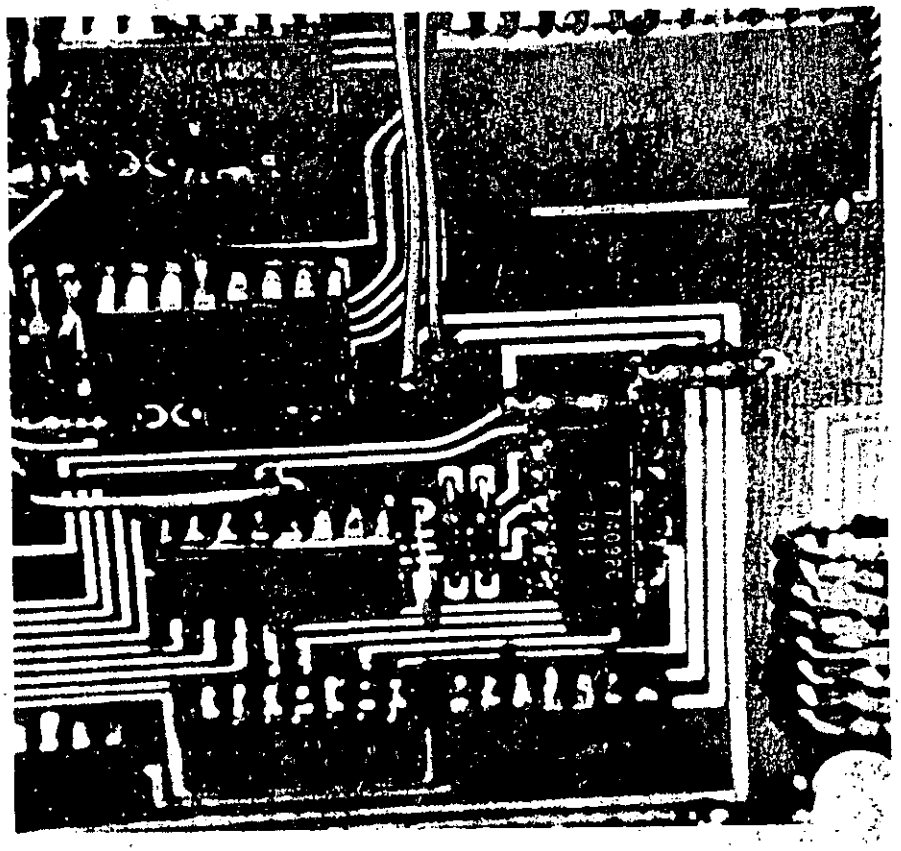
No hay duda de que la microelectrónica representa un reto a los países en vías de desarrollo, pues puede tener las siguientes repercusiones:

- Causar una reducción en el empleo industrial
- Disminuir su capital.
- Dificultar el establecimiento de industrias competitivas a nivel internacional.
- Obstaculizar la creación de una industria microelectrónica autóctona.
- Crear mayor dependencia tecnológica.

Para suavizar los efectos anteriores, es necesario definir una política que mediante diversos mecanismos permita:

- Estar siempre al tanto de los avances mundiales.
- Sensibilizar a las personas responsables de la toma de decisiones.
- Promover y establecer una industria microelectrónica.
- Difundir las aplicaciones de los productos de dicha industria.
- Crear programas educativos en *harware* y *software*.
- Impulsar la investigación y el desarrollo en esta materia.
- Regular la inversión y transferencia de tecnología.

Se menciona que antes de la primera revolución industrial, la diferencia de ingreso por persona entre los países desarrollados y los subdesarrollados era de 2 a 1; luego resultó de 10 a 1. Después de la revolución de las computadoras, ¿cómo será esa proporción? ¿Podemos permitirnos quedar al margen de ella?



Bibliografía

1. Mead y Conway, *Introduction to VLSI Systems*, Addison and Wesley, 1980.
2. Charles P. Lecht, *The Waves of Change*, Mac Graw-Hill, 1977.
3. Adam Osborne, *Running Wild*, Mac Graw-Hill, 1979.
4. John Allison, *Electronic Integrated Circuits*, Mac Graw Hill, 1975.
5. Juan Rada, *Microelectronics: its Impacts and Policy Implications*, UNIDO, 1982.
6. Ernest Braun, *Microelectronics and Government Policies: The Case of a Developed Country*, UNIDO, 1982.
7. *Implications of Microelectronics for Developing Countries*, UNIDO, 1981.
8. Nora y Minc *La informatización de la sociedad*, Fondo de Cultura Económica. 1980.
9. *Scientific American*, vol. 237, núm. 3, septiembre de 1977.
10. "Enginering Work Stations Complete the Network of Design-Automation Tools", *Electronics*, 17 de noviembre de 1982.
11. "Superchips Face Design Challenge", *High Technology*, enéro de 1983.
12. "Silicon Compilers and Foundries will Usher in User-Designed VLSI", *Electronics*, 11 de agosto de 1982.
13. "The Bumpy Road to Submicron Lithography", *High Technology*, marzo de 1983.
14. "VLSI Takes over Many Kinds of Fast Subsystems", *Electronics*, 24 de febrero de 1983.
15. "Automating Chip Layout", *IEEE Spectrum*, junio de 1982.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA COMPUTACION
ELECTRONICA Y PROGRAMACION

FUNCIONAMIENTO DE UNA COMPUTADORA

ING. ALEJANDRO VEGA

OCTUBRE, 1984

INTRODUCCION A LA COMPUTACION

ELECTRONICA Y PROGRAMACION

CAPITULO II

FUNCIONAMIENTO DE UNA COMPUTADORA

- Sistemas Numéricos
- Bit, Byte y Palabra
- Códigos de Máquina
- Modelos Von Neuman*

Ing. Alejandro Vega

* Notas tomadas del libro "Introducción a la Computación y a la Programación Estructurada, Guillermo Levine, Ed. Mc.Graw Hill.

Octubre, 1984

SISTEMAS NUMERICOS

Todos los procesos que se llevan a cabo en la CPU usan el sistema numérico binario. El sistema binario es ideal para las computadoras ya que sólo requiere dos símbolos: 0 y 1. El 0 y el 1 pueden ser fácilmente comparados con positivo y negativo, prendido y apagado, norte y sur, etc.

El sistema numérico decimal hay ciertas características de cualquier sistema numérico que son comunes a todos los sistemas numéricos. Por ejemplo, todos los sistemas numéricos tienen una característica llamada notación posicional. Primero examinemos los números decimales para encontrar esas características comunes.

El número decimal 123 está compuesto de tres elementos decimales el 1, el 2 y el 3. Sin embargo, aunque el 3 es el número más grande, es el más insignificante por la posición que ocupa. La importancia de un elemento, en un sistema numérico, aumenta cuando su posición se mueve a la izquierda. Esto introduce el concepto de posición relativa dentro del número. El 3 representa únicamente su propio valor. El 2, sin embargo, representa no sólo 2, sino 20, y el 1, por su posición, representa realmente 100. El número 123 puede entonces ser escrito así:

$$100 + 20 + 3$$

Se puede ver que la escritura del número 123 es en realidad una versión en taquigrafía de la expresión escrita arriba.

Sin embargo, la importancia de los números conforme vamos de derecha a izquierda se incrementa en multiplos de 10, o sea, 3 realmente representa 3 veces 1, mientras que 2 representa 2 veces 10 y 1 representa 1 vez 100.

Así que podemos representar el número en esta otra forma:

$$1 \times 100 + 2 \times 10 + 3 \times 1$$

Al expresar la importancia de cada posición como una potencia de 10, el número quedaría así:

$$1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$

El número 123, es entonces una versión abreviada que muestra solamente los coeficientes de las potencias de 10.

El 10 es llamado base ó raiz del sistema numérico.

El sistema numérico binario.

Hay evidencia de que los chinos sabían algo acerca del sistema binario hace más de 5,000 años. La primera evidencia directa de su uso se atribuye a Harriot, un matemático inglés del siglo XVI. Sin embargo, no fue hasta después de 1700 que Leibnitz, un matemático alemán, publicó el primer trabajo documentado del sistema binario. El pensó que todas las cantidades podían ser expresadas usando sólo dos dígitos, 0 y 1. Este sistema tiene una base o raíz de 2. Esto significa que la importancia del coeficiente aumenta, de derecha a izquierda, por una potencia de 2, no de 10 como en el sistema decimal. Veamos un número binario y lo que significa:

101

En el sistema decimal diremos ciento uno, sin embargo, cientos es un concepto reservado para el sistema decimal.

No tiene lugar en el sistema binario. Así que se debe leer ese número como uno, cero, uno. En el sistema decimal 101 de cualquier cosa puede hacer una cantidad más o menos grande. Veamos que cantidad está expresada en el sistema binario. Para hacer esto escribiremos el número en forma amplificada tal como lo hicimos con el sistema decimal;

$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Dado que 2^0 es igual a 1 (Cualquier valor elevado a la potencia de 0 es 1) y 2^2 es igual a 4, entonces el valor 101 en el sistema binario es igual a 5 en el sistema decimal ya que 1 x 4 +1 x 1 = 5. como consecuencia podemos ver, que mientras más pequeño sea el valor de la base, mayores son los elementos que necesita el número para expresar la misma cantidad. En nuestro ejemplo se necesitan 3 dígitos binarios para representar un dígito decimal con el mismo valor.

Los valores posicionales en un incremento decimal, de derecha a izquierda, se efectúan en múltiplos de 10. Los primeros seis valores posicionales son:

1
10
100
1000
10000
100000

En el sistema binario, los valores posicionales aumentan en multiplos sucesivos de su base que es 2. Los seis primeros valores posicionales en un número binario son:

1
2
4
8
16
32

Así que en un número decimal un 1 en la tercera columna a la izquierda tiene el valor de cien. En un número binario, un 1 en la tercera columna a la izquierda tendría solo el valor de cuatro.

A continuación se presenta una tabla con los primeros 10 números decimales y su equivalente binario.

DECIMAL	BINARIO
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

Conversión de Binario a Decimal

La conversión de binario a decimal se efectúa empleando potencias de 2. Comenzando con la potencia cero y multiplicando por el dígito que ocupa esa posición, finalmente se suman los productos.

Ejemplo:

Convertir el número binario 10011101 a su valor decimal.

Solución:

$2^0 \times 1 = 1$
$2^1 \times 0 = 0$
$2^2 \times 1 = 4$
$2^3 \times 1 = 8$
$2^4 \times 1 = 16$
$2^5 \times 0 = 0$
$2^6 \times 0 = 0$
$2^7 \times 1 = 128$

157

Conversión de decimal a binario. Para obtener el equivalente de decimal a binario existe una regla muy sencilla. Es la siguiente: Se divide la cantidad decimal entre la raiz 2, el residuo obtenido, que es igual a uno si la cifra es impar o cero si es par representa la posición de orden inferior de la cifra binaria equivalente.

El cociente anterior debe dividirse nuevamente entre la raíz 2. El residuo 1 ó 0 obtenido se coloca a la izquierda del dígito binario anterior, y representa el dígito binario de orden inmediato superior. El cociente obtenido se divide nuevamente entre 2. El proceso se repite hasta reducir el cociente a su mínima expresión.

Ejemplo:

$(8,947)_{10} = (10001011110011)_2$

residuos

8,947	1 ← Posición de orden inferior
4,473	1
2,236	0
1,118	0

559	1
279	1
139	1
69	1
34	0
17	1
8	0
4	0
2	0
1	1

Suma binaria

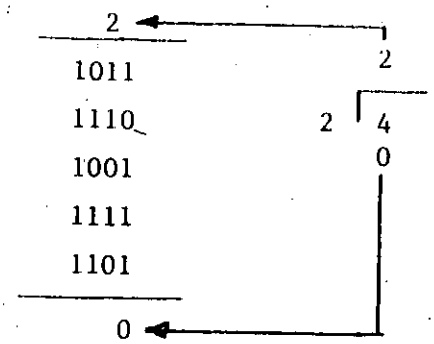
Existen varios métodos para sumar cantidades binarias. Aquí se presenta el más sencillo, ya que guarda similitud con el método usual de sumar cantidades decimales.

En el sistema decimal un dígito no puede excederse del valor 9, ya que es el dígito máximo representable en una posición, por lo tanto si al sumar varios dígitos decimales correspondientes a cierta posición de un grupo de sumandos, el valor resultante ocupase más de una posición, se determina entonces el número de veces que el valor comprende la base 10, que a su vez significa el acarreo que representa el número de decenas, centenas, unidades de millar, etc., que deben añadirse a la posición inmediata a la izquierda.

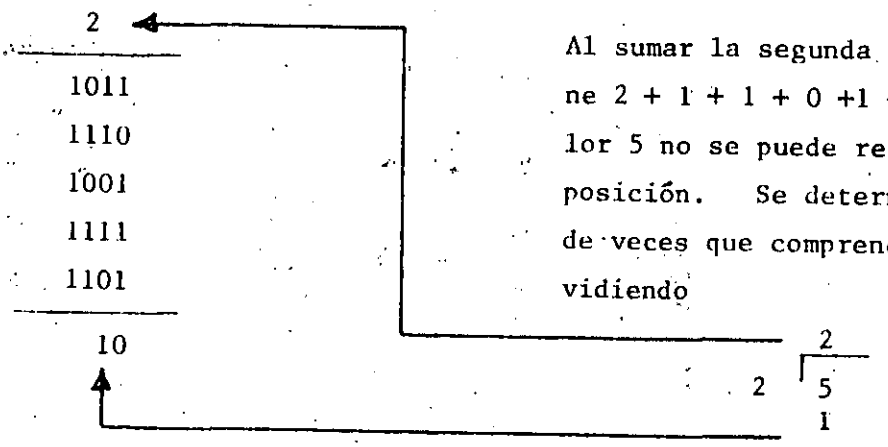
En el sistema binario, un dígito no puede excederse del valor 1, ya que es el número máximo representable en una sola posición si se aplica, entonces, el proceso anterior se tendrá, al sumar, las siguientes cantidades binarias:

```
  1011
  1110
+ 1001
  1111
  1101
```

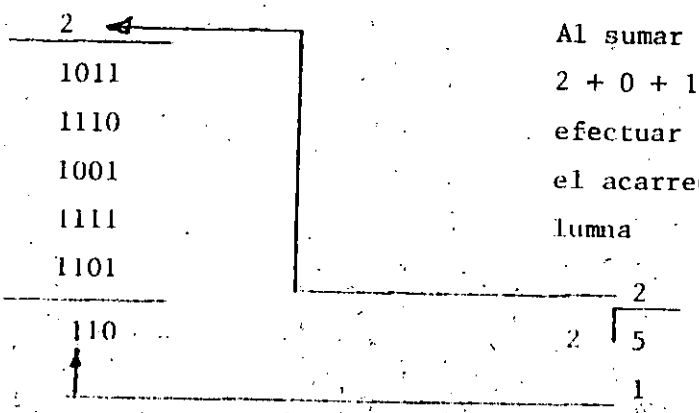
Al sumar la primera posición se tiene: 1 + 0 + 1 + 1 + 1 = 4, el valor 4 es mayor a la base, se determina entonces el número de veces que comprende a la base 2 (binaria) dividiendo:



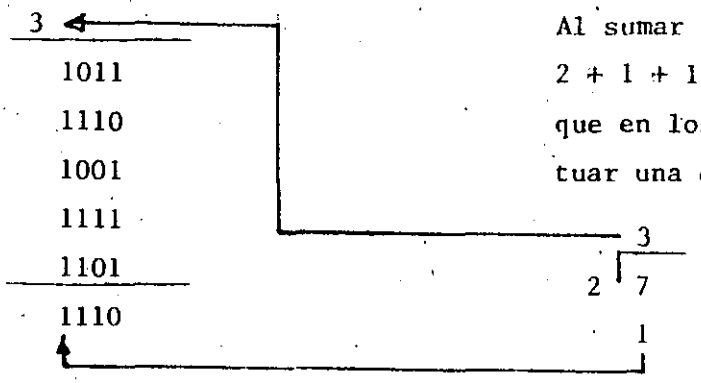
Por lo tanto, se registra el residuo que es el dígito de orden inferior y se acarrea el cociente 2.



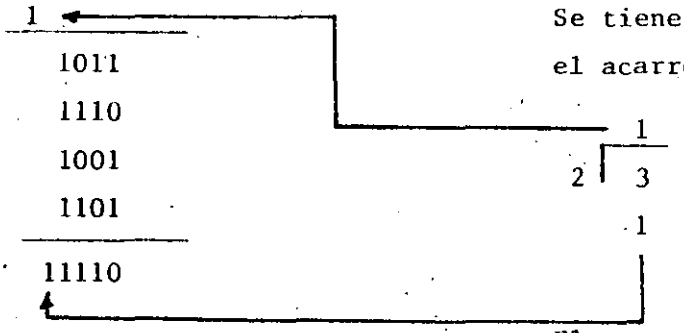
Al sumar la segunda posición se tiene 2 + 1 + 1 + 0 +1 + 0 = 5, el valor 5 no se puede registrar en una posición. Se determina el número de veces que comprende a la base, dividiendo



Al sumar la tercera posición 2 + 0 + 1 + 0 + 1 + 1 = 5 Se debe efectuar una división para obtener el acarreo y el resultado de esa columna



Al sumar la última posición $2 + 1 + 1 + 1 + 1 + 1 = 7$ al igual que en los anteriores se debe efectuar una división..



Se tiene una quinta posición para el acarreo, que también se divide

El acarreo de la sexta posición ya no requiere división por ser menor a la base quedando finalmente:

13222 } Acarreos	
1011	11
1110	14
1001	9
1111	15
1101	13
$(111110)_2$	$(62)_{10}$

Resta Binaria

Como en el caso de la suma, aquí se muestra un método entre muchos que hay para efectuar esta operación aritmética.

1110
0011

Al efectuar la resta de la primera posición de dígitos binarios, se ve que el dígito 0 tiene un valor menor que el dígito 1; por lo tanto, así como en el sistema decimal se pide una unidad al dígito que se encuentra inmediatamente a la izquierda con un valor de diez de acuerdo con la raíz, en este caso se pide un 1 con un valor de 2 binario que añadido al cero se tiene: $2 + 0 = 2$

Ahora si se puede efectuar la resta de esa posición teniendo: dos binario menos uno binario igual a 1 binario.

$$\begin{array}{r} {\scriptstyle 2}\\ 1110\\ -\ 0011\\ \hline 1 \end{array}$$

Pero, para efectuar la disminución de ese valor pedido se añade un 1 a la posición correspondiente del sustraendo, teniendo así:

$$\begin{array}{r} 1110\\ -\ 0011\\ \hline 1 \end{array}$$

$1 + 1 = 2$ Sustraendo actual

Al efectuar la resta de la segunda posición, se ve que el dígito 1 del minuendo tiene un valor menor que el dígito 2 valor actual del sustraendo por lo que se pide una unidad a la posición de la izquierda en la misma forma anterior.

$$\begin{array}{r} {\scriptstyle 2}\\ 1110\\ -\ 0011\\ \hline 11 \end{array}$$

2	+	1	=	3
Pedido		minuendo anterior		minuendo actual

Ahora si se puede efectuar la resta de esa posición, teniendo:

3	-	2	=	1
minuendo actual		sustraendo actual		

Para efectuar nuevamente la disminución del valor pedido se añade un 1 a la posición del sustraendo que es cero:

0 + 1 = 1 Sustraendo actual

Efectuando la resta de esa posición se tiene:

$$\begin{array}{r} 1110 \\ -\ 0011 \\ \hline 011 \end{array} \qquad 1 - 1 = 0$$

En este caso no hubo necesidad de pedir ninguna unidad, quedando realizada finalmente la operación como se indica

	COMPROBACION
$-\ (1110)_2$	$-\ (14)_{10}$
$(0011)_2$	$(\ 3)_{10}$
$(1011)_2$	$(11)_{10}$

El Sistema Numérico Octal.

El sistema numérico octal usa solamente 8 dígitos que van del 0 al 7. A continuación se presenta una tabla con sus equivalentes decimales, octales y binarios.

DECIMAL	OCTAL	BINARIO
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111
8	10	1000
9	11	1001
10	12	1010

Conversión de Octal a Decimal.

La conversión de octal a decimal se efectúa empleando potencias de 8 que es la base, comenzando por la potencia cero y se va incrementando en uno por cada posición que se corre a la izquierda, se multiplica por el dígito que ocupa esa posición y se suman los productos, en una forma semejante a la conversión de números binarios.

Ejemplo:

Encontrar el equivalente decimal del número octal 7526.

$8^0 \times 6 =$ 6
$8^1 \times 2 =$ 16
$8^2 \times 5 =$ 320
$8^3 \times 7 =$ 3584

3926

Conversión de Decimal a Octal.

Para obtener el equivalente de decimal a octal se divide la cantidad decimal entre la raiz 8, el residuo obtenido representa la posición de orden

inferior de la cifra octal equivalente.

El cociente obtenido debe dividirse nuevamente entre la raíz 8. El residuo obtenido se coloca a la izquierda del dígito octal anterior y representa el dígito octal de orden inmediato superior. El cociente obtenido se divide nuevamente entre 8 y el proceso se repite hasta reducir el cociente a su mínima expresión.

Ejemplo:

$(6,741)_{10} = (15125)_8$

6,741	5 ← Posición de orden inferior
842	2
105	1
13	5
1	1

Conversión de Octal a Binario

Para representar cualquier dígito octal se requieren solamente tres dígitos binarios, esto permite efectuar una conversión directa.

Ejemplo:

Convertir el siguiente número octal a su equivalente binario

4	6	3	7
100	110	011	111

Conversión de Binario a Octal.

Para efectuar la conversión de binario a octal se separan los dígitos binarios de 3 en 3 y se convierten a su equivalente en octal.

Ejemplo: Convertir el siguiente número binario a su equivalente en octal.

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

3 5 3 3 5

Sistema Numérico Hexadecimal

El sistema numérico hexadecimal tiene 16 elementos, se toman los dígitos del 0 al 9 y las letras de la A a la F.

A continuación se presenta una tabla con los valores decimales y sus equivalentes en hexadecimal y binario.

DECIMAL	HEXADECIMAL	BINARIO
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Conversión de Hexadecimal a Decimal.

La conversión de hexadecimal a decimal se efectúa empleando potencias de 16, comenzando con la potencia cero y se van incrementando en uno por cada posición que se corre a la izquierda, multiplicando por el dígito que ocupa esa posición, finalmente se suman los productos.

Ejemplo:

$(B\ 9\ A\ C)_{16} = (47,532)_{10}$

$16^0 \times 12 = 12$

$16^1 \times 10 = 160$

$16^2 \times 9 = 2304$

$16^3 \times 11 = 45056$

47532

Conversión de Decimal a Hexadecimal.

Para obtener el equivalente decimal a hexadecimal se divide la cantidad decimal entre la raíz 16, el residuo obtenido representa la posición de orden inferior de la cifra hexadecimal equivalente.

El cociente obtenido debe dividirse nuevamente entre la raiz 16, el residuo obtenido se coloca a la izquierda del dígito hexadecimal anterior y representa el dígito hexadecimal de orden inmediato superior. El cociente obtenido se divide nuevamente entre 16 y el proceso se reqpite hasta reducir el cociente a su mínima expresión.

Ejemplo:

$(9,846)_{10} = (267C)_{16}$

9,852	C
615	7
38	6
2	2

Conversión de Hexadecimal a Binario.

Para representar cualquier dígito hexadecimal se requieren solamente cuatro dígitos binarios, esto permite efectuar una conversión directa.

Ejemplo:

Convertir el siguiente número hexadecimal a su equivalente binario.

A	9	F	D
1010	1001	1111	1101

Conversiones de Binario a Hexadecimal.

Para efectuar la conversión de binario a hexadecimal se separan los dígitos binarios de 4 en 4 y se convierten a su equivalente hexadecimal.

Ejemplo:

Convertir el siguiente número binario a su equivalente hexadecimal.

BIT, BYTE Y PALABRAS

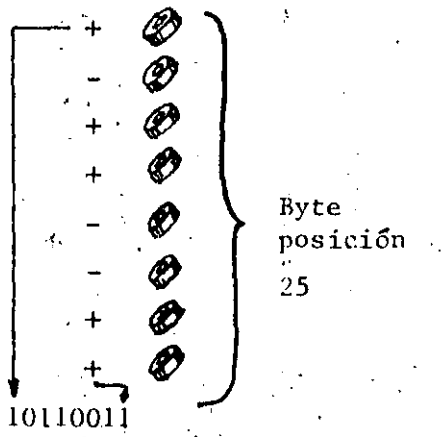
Bit

La memoria consiste de cualquier elemento electrónico o magnético que puede tener solo dos estados: prendido o apagado, positivo o negativo y así por el estilo. Estos estados se crean por medio del paso de la corriente electrónica a través de un material, como en el caso de un núcleo magnético. La dirección de la corriente eléctrica detemina la polaridad o el estado magnético del núcleo. Al invertir la dirección de la corriente se cambia el estado magnético. Por consiguiente, así es como podemos obtener los dos estados. Cada núcleo puede representar ya sea el 0 ó el 1. A los que podemos llamar bits (BInary digiTS) de memoria.

Un Bit puede escribirse: es decir, se puede cambiar de 0 a 1 y de 1 a 0 y también puede ser leido: es decir, que su estado magnético puede ser detectado.

Byte

En una memoria de núcleos, como en otros tipos de almacenamiento, los BITS individuales se "acomodan" uno encima del otro para formar una posición de memoria que sea direccionable, a este grupo de Bits se le conoce como Byte.



Palabra.

Un grupo de BITS forma una palabra. El tamaño de la palabra depende de la computadora y se mide por el número de bits que contiene

	Palabra
IBM	4 Bytes
CDC Serie 170	60 Bits
CDC Serie 3000	24 Bits
Burroughs Serie 6700	48 Bits
Univac Serie 1100	36 Bits

Códigos de Máquina

Se llama código de máquina a la forma en que se representan las diferentes instrucciones y caracteres en la memoria, estos códigos varían de una computadora a otra.

Juego de instrucciones es simplemente un término usado para describir todas las instrucciones que la unidad de aritmética y lógica de una computadora puede ejecutar.

Uno de los factores más sorprendentes acerca de las instrucciones, es su limitada capacidad. Como se puede ver una computadora abarca solamente funciones como operaciones aritméticas, conversiones de un formato de datos a otro, operaciones de comparación, bifurcaciones, desplazamientos, entrada/salida y operaciones lógicas. Es la capacidad humana la que uniendo estas instrucciones en complicadas combinaciones, llamados programas, permite a las computadoras desarrollar trabajos administrativos, calculos científicos, jugar ajedrez, controlar el tráfico, etc.

A continuación se anexan los códigos EBCDIC, SCII, FIELDATA.

OPERATION CODES (HEX)

RR FORMAT INSTRUCTIONS

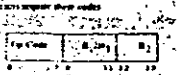
Dec.	Hex.	Mnemonic	Condition Code	(1) Program Exception	Operands	Instruction Code
0	00					
1	01					
2	02					
3	03					
4	04	SPM				
5	05	BALR				
6	06	BCTR				
7	07	BCR				
8	08	SSK				
9	09	ISK				
10	0A	SVC				
11	0B					
12	0C		(if BCDIC=1)			
13	0D		(if BCDIC=1)			
14	0E					
15	0F					
16	10	LPR				
17	11	LNR				
18	12	LTR				
19	13	LCR				
20	14	NR				
21	15	CLR				
22	16	OR				
23	17	XR				
24	18	LR				
25	19	CR				
26	1A	AR				
27	1B	SR				
28	1C	MR				
29	1D	DR				
30	1E	ALR				
31	1F	SLR				
32	20	LPDR				
33	21	LNDR				
34	22	LTDR				
35	23	LCDR				
36	24	HDR				
37	25	LRDR				
38	26	MXR				
39	27	MXDR				
40	28	LDR				
41	29	CDR				
42	2A	ADR				
43	2B	SDR				
44	2C	MDR				
45	2D	DDR				
46	2E	AWR				
47	2F	SWR				
48	30	LPER				
49	31	LNER				
50	32	LTER				
51	33	LCER				
52	34	HER				
53	35	LRER				
54	36	AXR				
55	37	SXR				
56	38	EMER				
57	39	CER				
58	3A	AER				
59	3B	SER				
60	3C	MER				
61	3D	DER				
62	3E	AUR				
63	3F	SUR				

(1) Note: Bit check for BCDIC not shown; bit C not for odd or even parity as needed except for ... points, decimal 64 or 0, for 'none' or decimal 122.

(2) Exceptions to basic instructions.

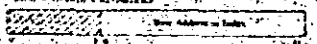
(3) Control bit/field controlling program exceptions shown under mnemonic.

RR Format



(1) R1 - Mask field for all RR instructions except SPM and SVC

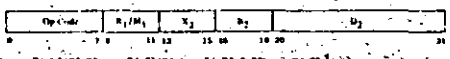
BASE AND DISPLACEMENT REGISTERS



RX FORMAT INSTRUCTIONS

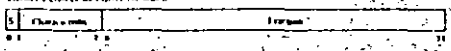
Dec.	Hex.	Mnemonic	Condition Code	(2) Program Exception	Operands	Instruction Code	(1)
64	40	STH		SF			
65	41	LA					
66	42	STC					
67	43	IC					
68	44	EX					
69	45	BAL					
70	46	BCT					
71	47	BC					
72	48	LH					
73	49	CH					
74	4A	AH					
75	4B	SH					
76	4C	MH					
77	4D	BAS					
78	4E	CVD					
79	4F	CVB					
80	50	ST					
81	51						
82	52						
83	53						
84	54	N					
85	55	CL					
86	56	O					
87	57	X					
88	58	L					
89	59	C					
90	5A	A					
91	5B	S					
92	5C	M					
93	5D	D					
94	5E	AL					
95	5F	SL					
96	60	STD					
97	61						
98	62						
99	63						
100	64						
101	65						
102	66						
103	67	MXD					
104	68	LD					
105	69	CD					
106	6A	AD					
107	6B	SD					
108	6C	MD					
109	6D	DD					
110	6E	AW					
111	6F	SW					
112	70	STD					
113	71						
114	72						
115	73						
116	74						
117	75						
118	76						
119	77						
120	78	LE					
121	79	CE					
122	7A	AE					
123	7B	SE					
124	7C	ME					
125	7D	DE					
126	7E	AU					
127	7F	SU					

RX Format

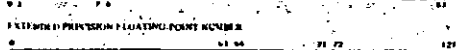


R1, R2 (32, 52) R1, R3 (32) R1, R2 (0, 52) R1, R3 (52)

SHORT FLOATING-POINT NUMBER



LONG FLOATING-POINT NUMBER

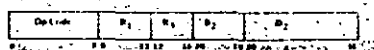


EXTENDED PRECISION FLOATING-POINT NUMBER

RR, SI FORMAT INSTRUCTIONS

Dec.	Hex.	Mnemonic	Condition Code	(2) Program Exception	Operands	Instruction Code	(1)
128	80	SSM					
129	81						
130	82	LPSW					
131	83	(Diagnose)					
132	84	WRD					
133	85	RDD					
134	86	BXH					
135	87	BXLE					
136	88	SRL					
137	89	SLL					
138	8A	SRA					
139	8B	SLA					
140	8C	SRDL					
141	8D	SLDL					
142	8E	SRDA					
143	8F	SLDA					
144	90	STM					
145	91	TM					
146	92	MVI					
147	93	TS					
148	94	NI					
149	95	CLI					
150	96	OI					
151	97	XI					
152	98	LM					
153	99						
154	9A						
155	9B						
156	9C	SIO					
157	9D	TIO					
158	9E	HDV					
159	9F	TCH					
160	A0						
161	A1						
162	A2						
163	A3						
164	A4						
165	A5						
166	A6						
167	A7						
168	A8						
169	A9						
170	AA						
171	AB						
172	AC						
173	AD						
174	AE						
175	AF						
176	B0						
177	B1						
178	B2						
179	B3						
180	B4						
181	B5						
182	B6						
183	B7						
184	B8						
185	B9						
186	BA						
187	BB						
188	BC						
189	BD						
190	BE						
191	BF						

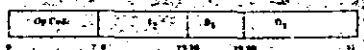
RS Format



R1, R3 (32) PUX, BXH, CM, STM

R1, R3 (52) Shift

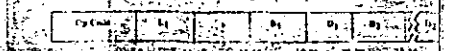
SI Format



SS FORMAT INSTRUCTIONS

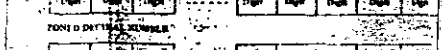
Dec.	Hex.	Mnemonic	Condition Code	(2) Program Exception	Operands	Instruction Code	(1)
192	C0						
193	C1						
194	C2						
195	C3						
196	C4						
197	C5						
198	C6						
199	C7						
200	C8						
201	C9						
202	CA						
203	CB						
204	CC						
205	CD						
206	CE						
207	CF						
208	D0						
209	D1	MVN					
210	D2	MVC					
211	D3	MVZ					
212	D4	NC					
213	D5	CLC					
214	D6	OC					
215	D7	XC					
216	D8						
217	D9						
218	DA						
219	DB						
220	DC	TR					
221	DD	TRT					
222	DE	ED (2)					
223	DF	EDMK (2)					
224	E0						
225	E1						
226	E2						
227	E3						
228	E4						
229	E5						
230	E6						
231	E7						
232	E8						
233	E9						
234	EA						
235	EB						
236	EC						
237	ED						
238	EE	PACK					
239	EF	UNPK					
240	F0						
241	F1						
242	F2						
243	F3						
244	F4						
245	F5						
246	F6						
247	F7						
248	F8	ZAP (3)					
249	F9	CP (3)					
250	FA	AP (3)					
251	FB	SP (3)					
252	FC	MP (3)					
253	FD	DP (3)					
254	FE						
255	FF						

SS Format



D1(L1, B1), D2(L2, B2) = ... MVN, MVC, MVZ, NC, CLC, OC, XC, TR, TRT, ED, EDMK

PACKED DECIMAL NUMBER



ZONED DECIMAL NUMBER

21

Appendix B. Character Codes

Table B-1. Correspondence of ASCII, EBCDIC, and Fieldata, in ASCII Order

ASCII						EBCDIC*			Fieldata			
Octal	Hex	Dec	Character Name	Sym	Card Punches	Hex	Dec	Octal	Sym **	Dec	Octal	Card Punches
0	0	0	NUL		12-0-9-8-1	0	0	0	No conversion			
1	1	1	SOH		12-9-1	1	1	1	No conversion			
2	2	2	STX		12-9-2	2	2	2	No conversion			
3	3	3	ETX		12-9-3	3	3	3	No conversion			
4	4	4	EOT		9-7	37	55	67	No conversion			
5	5	5	ENQ		0-9-8-5	2D	45	55	No conversion			
6	6	6	ACK		0-9-8-6	2E	46	56	No conversion			
7	7	7	BEL		0-9-8-7	2F	47	57	No conversion			
10	8	8	BS		11-9-6	16	22	26	No conversion			
11	9	9	HT		12-9-5	05	5	5	No conversion			
12	A	10	LF		0-9-5	25	37	45	No conversion			
13	B	11	VT		12-9-8-3	0B	11	13	No conversion			
14	C	12	FF		12-9-8-4	0C	12	14	No conversion			
15	D	13	CR		12-9-8-5	0D	13	15	No conversion			
16	E	14	SO		12-9-8-6	0E	14	16	No conversion			
17	F	15	SI		12-9-8-7	0F	15	17	No conversion			
20	10	16	DLE		12-11-9-8-1	10	16	20	No conversion			
21	11	17	DC1		11-9-1	11	17	21	No conversion			
22	12	18	DC2		11-9-2	12	18	22	No conversion			
23	13	19	DC3		11-9-3	13	19	23	No conversion			
24	14	20	DC4		9-8-4	3C	60	74	No conversion			
25	15	21	NAK		9-8-5	3D	61	75	No conversion			
26	16	22	SYN		9-2	32	50	62	No conversion			
27	17	23	ETB		0-9-6	26	38	46	No conversion			
30	18	24	CAN		11-9-8	18	24	30	No conversion			
31	19	25	EM		11-9-8-1	19	25	31	No conversion			
32	1A	26	SUB		9-8-7	3F	63	77	No conversion			
33	1B	27	ESC		0-9-7	27	39	47	No conversion			
34	1C	28	FS		11-9-8-4	1C	28	34	No conversion			
35	1D	29	GS		11-9-8-5	1D	29	35	No conversion			

Table B-1. Correspondence of ASCII, EBCDIC, and Fieldata, in ASCII Order (continued)

22

ASCII						EBCDIC*			Fieldata			
Octal	Hex	Dec	Character Name	Sym	Card Punches	Hex	Dec	Octal	Sym **	Dec	Octal	Card Punches
36	1E	30	RS		11-9-8-6	1E	30	36	No conversion			
37	1F	31	US		11-9-8-7	1F	31	37	No conversion			
40	20	32	Sp,Space			40	64	100		5	5	SP
41	21	33	Exclamation Point	!	12-8-7	4F	79	117		45	55	11-0
42	22	34	Quotation Mark, Dieresis	"	8-7	7F	127	177	n	62	76	0-7-8
43	23	35	Number Sign, Pound Sign	#	8-3	7B	123	173		3	3	12-7-8
44	24	36	Dollar Sign	$	11-8-3	5B	91	133		39	47	11-3-8
45	25	37	Percent Sign	%	0-8-4	6C	108	154		42	52	0-5-8
46	26	38	Ampersand	&	12	50	80	120		38	46	2-9
47	27	39	Apostrophe, Accute Accent	'	8-5	7D	125	175		58	72	4-8
50	28	40	Opening Parenthesis	(	12-8-5	4D	77	115		41	51	0-4-8
51	29	41	Closing Parenthesis	)	11-8-5	5D	93	135		32	40	12-4-8
52	2A	42	Asterisk	*	11-8-4	5C	92	134		40	50	11-4-8
53	2B	43	Plus Sign	+	12-8-6	4E	78	116		34	42	12
54	2C	44	Comma, Cedilla	,	0-8-3	6B	107	153		46	56	0-3-8
55	2D	45	Minus Sign, Hyphen	-	11	60	96	140		33	41	11
56	2E	46	Period, Decimal Point	.	12-8-3	4B	75	113		61	75	12-3-8
57	2F	47	Slash, Virgule, Solidus	/	0-1	61	97	141		60	74	0-1
60	30	48	0,Zero	0	0	F0	240	360		48	60	0
61	31	49	1,one	1	1	F1	241	361		49	61	1
62	32	50	2,two	2	2	F2	242	362		50	62	2
63	33	51	3,three	3	3	F3	243	363		51	63	3
64	34	52	4,four	4	4	F4	244	364		52	64	4
65	35	53	5,five	5	5	F5	245	365		53	65	5
66	36	54	6,six	6	6	F6	246	366		54	66	6
67	37	55	7,seven	7	7	F7	247	367		55	67	7
70	38	56	8,eight	8	8	F8	248	370		56	70	8
71	39	57	9,nine	9	9	F9	249	371		57	71	9
72	3A	58	Colon	:	8-2	7A	122	172		43	53	5-8
73	3B	59	Semicolon	;	11-8-6	5E	94	136		59	73	11-6-8
74	3C	60	Less Than	<	12-8-4	4C	76	114		35	43	12-6-8
75	3D	61	Equal Sign	=	8-6	7E	126	176		36	44	3-8
76	3E	62	Greater than	>	0-8-6	6E	110	156		37	45	6-8
77	3F	63	Question mark	?	0-8-7	6F	111	157		44	54	12-0

Table B-1. Correspondence of ASCII, EBCDIC, and Fieldata, in ASCII Order (continued)

23

ASCII						EBCDIC*			Fieldata			
Octal	Hex	Dec	Character Name	Sym	Card Punches	Hex	Dec	Octal	Sym **	Dec	Octal	Card Punches
100	40	64	Commercial at symbol	@	8-4	7C	124	174		0	0	7-8
101	41	65	Capital A	A	12-1	C1	193	301		6	6	12-1
102	42	66	Capital B	B	12-2	C2	194	302		7	7	12-2
103	43	67	Capital C	C	12-3	C3	195	303		8	10	12-3
104	44	68	Capital D	D	12-4	C4	196	304		9	11	12-4
105	45	69	Capital E	E	12-5	C5	197	305		10	12	12-5
106	46	70	Capital F	F	12-6	C6	198	306		11	13	12-6
107	47	71	Capital G	G	12-7	C7	199	307		12	14	12-7
110	48	72	Capital H	H	12-8	C8	200	310		13	15	12-8
111	49	73	Capital I	I	12-9	C9	201	311		14	16	12-9
112	4A	74	Capital J	J	11-1	D1	209	321		15	17	11-1
113	4B	75	Capital K	K	11-2	D2	210	322		16	20	11-2
114	4C	76	Capital L	L	11-3	D3	211	323		17	21	11-3
115	4D	77	Capital M	M	11-4	D4	212	324		18	22	11-4
116	4E	78	Capital N	N	11-5	D5	213	325		19	23	11-5
117	4F	79	Capital O	O	11-6	D6	214	326		20	24	11-6
120	50	80	Capital P	P	11-7	D7	215	327		21	25	11-7
121	51	81	Capital Q	Q	11-8	D8	216	330		22	26	11-8
122	52	82	Capital R	R	11-9	D9	217	331		23	27	11-9
123	53	83	Capital S	S	0-2	E2	226	342		24	30	0-2
124	54	84	Capital T	T	0-3	E3	227	343		25	31	0-3
125	55	85	Capital U	U	0-4	E4	228	344		26	32	0-4
126	56	86	Capital V	V	0-5	E5	229	345		27	33	0-5
127	57	87	Capital W	W	0-6	E6	230	346		28	34	0-6
130	58	88	Capital X	X	0-7	E7	231	347		29	35	0-7
131	59	89	Capital Y	Y	0-8	E8	232	350		30	36	0-8
132	5A	90	Capital Z	Z	0-9	E9	233	351		31	37	0-9
133	5B	91	Opening Bracket	[	12-8-2	4A	74	112		1	1	12-5-8
134	5C	92	Reverse Slash	\	0-8-2	E0	224	340		47	57	0-6-8
135	5D	93	Closing bracket	]	11-8-2	5A	90	132		2	2	11-5-8
136	5E	94	Circumflex	^	11-8-7	5F	95	137	↑	4	4	11-7-8
137	5F	95	Underline	_	0-8-5	6D	109	155	←	63	77	0-2-8
140	60	96	Grave accent	`	8-1	79	121	171	@	0	0	7-8
141	61	97	Small A	a	12-0-1	81	129	201	A	6	6	12-1
142	62	98	Small B	b	12-0-2	82	130	202	B	7	7	12-2
143	63	99	Small C	c	12-0-3	83	131	203	C	8	10	12-3
144	64	100	Small D	d	12-0-4	84	132	204	D	9	11	12-4
145	65	101	Small E	e	12-0-5	85	133	205	E	10	12	12-5
146	66	102	Small F	f	12-0-6	86	134	206	F	11	13	12-6
147	67	103	Small G	g	12-0-7	87	135	207	G	12	14	12-7
150	68	104	Small H	h	12-0-8	88	136	210	H	13	15	12-8
151	69	105	Small I	i	12-0-9	89	137	211	I	14	16	12-9
152	6A	106	Small J	j	12-11-1	91	145	221	J	15	17	11-1
153	6B	107	Small K	k	12-11-2	92	146	222	K	16	20	11-2
154	6C	108	Small L	l	12-11-3	93	147	223	L	17	21	11-3
155	6D	109	Small M	m	12-11-4	94	148	224	M	18	22	11-4

Table B-1. Correspondence of ASCII, EBCDIC, and Fieldata, in ASCII Order (continued)

24

ASCII						EBCDIC*			Fieldata			
Octal	Hex	Dec	Character Name	Sym	Card Punches	Hex	Dec	Octal	Sym **	Dec	Octal	
156	6E	110	Small N	n	12-11-5	95	149	225	N	19	23	11-5
157	6F	111	Small O	o	12-11-6	96	150	226	O	20	24	11-6
160	70	112	Small P	p	12-11-7	97	151	227	P	21	25	11-7
161	71	113	Small Q	q	12-11-8	98	152	230	Q	22	26	11-8
162	72	114	Small R	r	12-11-9	99	153	231	R	23	27	11-9
163	73	115	Small S	s	11-0-2	A2	162	242	S	24	30	0-2
164	74	116	Small T	t	11-0-3	A3	163	243	T	25	31	0-3
165	75	117	Small U	u	11-0-4	A4	164	244	U	26	32	0-4
166	76	118	Small V	v	11-0-5	A5	165	245	V	27	33	0-5
167	77	119	Small W	w	11-0-6	A6	166	246	W	28	34	0-6
170	78	120	Small X	x	11-0-7	A7	167	247	X	29	35	0-7
171	79	121	Small Y	y	11-0-8	A8	168	250	Y	30	36	0-8
172	7A	122	Small Z	z	11-0-9	A9	169	251	Z	31	37	0-9
173	7B	123	Opening Brace	{	12-0	C0	192	300	?	44	54	12-0
174	7C	124	Verticlal Line	|	12-11	6A	106	152	\	47	57	0-6-8
175	7D	125	Closing Brace	}	11-0	D0	208	320		45	55	11-0
176	7E	126	Overline,Tilde	~	11-0-1	A1	161	241	≤	4	4	11-7-8
177	7F	127	DEL,Delete		12-9-7	07	7	7	≠	63	77	0-2-8
200	80	128			11-0-9-8-1	20	32	40	No conversion			
201	81	129			0-9-1	21	33	41	No conversion			
202	82	130			0-9-2	22	34	42	No conversion			
203	83	131			0-9-3	23	35	43	No conversion			
204	84	132			0-9-4	24	36	44	No conversion			
205	85	133			11-9-5	15	21	25	No conversion			
206	86	134			12-9-6	06	6	6	No conversion			
207	87	135			11-9-7	17	23	27	No conversion			
210	88	136			0-9-8	28	40	50	No conversion			
211	89	137			0-9-8-1	29	41	51	No conversion			
212	8A	138			0-9-8-2	2A	42	52	No conversion			
213	8B	139			0-9-8-3	2B	43	53	No conversion			
214	8C	140			0-9-8-4	2C	44	54	No conversion			
215	8D	141			12-9-8-1	09	9	11	No conversion			
216	8E	142			12-9-8-2	0A	10	12	No conversion			
217	8F	143			11-9-8-3	1B	27	33	No conversion			
220	90	144			12-11-0-9-8-1	30	48	60	No conversion			
221	91	145			9-1	31	49	61	No conversion			
222	92	146			11-9-8-2	1A	26	32	No conversion			
223	93	147			9-3	33	51	63	No conversion			
224	94	148			9-4	34	52	64	No conversion			
225	95	149			9-5	35	53	65	No conversion			
226	96	150			9-6	36	54	66	No conversion			
227	97	151			12-9-8	08	8	10	No conversion			
230	98	152			9-8	38	56	70	No conversion			
231	99	153			9-8-1	39	57	71	No conversion			
232	9A	154			9-8-2	3A	58	72	No conversion			
233	9B	155			9-8-3	3B	59	73	No conversion			
234	9C	156			12-9-4	04	4	4	No conversion			
235	9D	157			11-9-4	14	20	24	No conversion			

25

Table B–1. Correspondence of ASCII, EBCDIC, and Fieldata, in ASCII Order (continued)

ASCII						EBCDIC*			Fieldata			
Octal	Hex	Dec	Character Name	Sym	Card Punches	Hex	Dec	Octal	Sym **	Dec	Octal	Card Punches
236	9E	158			9-8-6	3E	62	76		No conversion		
237	9F	159			11-0-9-1	E1	225	341		No conversion		
240	A0	160			12-0-9-1	41	65	101		No conversion		
241	A1	161			12-0-9-2	42	66	102		No conversion		
242	A2	162			12-0-9-3	43	67	103		No conversion		
243	A3	163			12-0-9-4	44	68	104		No conversion		
244	A4	164			12-0-9-5	45	69	105		No conversion		
245	A5	165			12-0-9-6	46	70	106		No conversion		
246	A6	166			12-0-9-7	47	71	107		No conversion		
247	A7	167			12-0-9-8	48	72	110		No conversion		
250	A8	168			12-8-1	49	73	111		No conversion		
251	A9	169			12-11-9-1	51	81	121		No conversion		
252	AA	170			12-11-9-2	52	82	122		No conversion		
253	AB	171			12-11-9-3	53	83	123		No conversion		
254	AC	172			12-11-9-4	54	84	124		No conversion		
255	AD	173			12-11-9-5	55	85	125		No conversion		
256	AE	174			12-11-9-6	56	86	126		No conversion		
257	AF	175			12-11-9-7	57	87	127		No conversion		
260	B0	176			12-11-9-8	58	88	130		No conversion		
261	B1	177			11-8-1	59	89	131		No conversion		
262	B2	178			11-0-9-2	62	98	142		No conversion		
263	B3	179			11-0-9-3	63	99	143		No conversion		
264	B4	180			11-0-9-4	64	100	144		No conversion		
265	B5	181			11-0-9-5	65	101	145		No conversion		
266	B6	182			11-0-9-6	66	102	146		No conversion		
267	B7	183			11-0-9-7	67	103	147		No conversion		
270	B8	184			11-0-9-8	68	104	150		No conversion		
271	B9	185			0-8-1	69	105	151		No conversion		
272	BA	186			12-11-0	70	112	160		No conversion		
273	BB	187			12-11-0-9-1	71	113	161		No conversion		
274	BC	188			12-11-0-9-2	72	114	162		No conversion		
275	BD	189			12-11-0-9-3	73	115	163		No conversion		
276	BE	190			12-11-0-9-4	74	116	164		No conversion		
277	BF	191			12-11-0-9-5	75	117	165		No conversion		
300	C0	192			12-11-0-9-6	76	118	166		No conversion		
301	C1	193			12-11-0-9-7	77	119	167		No conversion		
302	C2	194			12-11-0-9-8	78	120	170		No conversion		
303	C3	195			12-0-8-1	80	128	200		No conversion		
304	C4	196			12-0-8-2	8A	138	212		No conversion		
305	C5	197			12-0-8-3	8B	139	213		No conversion		
306	C6	198			12-0-8-4	8C	140	214		No conversion		
307	C7	199			12-0-8-5	8D	141	215		No conversion		
310	C8	200			12-0-8-6	8E	142	216		No conversion		
311	C9	201			12-0-8-7	8F	143	217		No conversion		
312	CA	202			12-11-8-1	90	144	220		No conversion		
313	CB	203			12-11-8-2	9A	154	232		No conversion		
314	CC	204			12-11-8-3	9B	155	233		No conversion		
315	CD	205			12-11-8-4	9C	156	234		No conversion		
316	CE	206			12-11-8-5	9D	157	235		No conversion		

26

Table B-1. Correspondence of ASCII, EBCDIC, and Fieldata, in ASCII Order (continued)

ASCII						EBCDIC*			Fieldata			
Octal	Hex	Dec	Character Name	Sym	Card Punches	Hex	Dec	Octal	Sym **	Dec	Octal	Card Punches
317	CF	207			12-11-8-6	9E	158	236		No conversion		
320	D0	208			12-11-8-7	9F	159	237		No conversion		
321	D1	209			11-0-8-1	A0	160	240		No conversion		
322	D2	210			11-0-8-2	AA	170	252		No conversion		
323	D3	211			11-0-8-3	AB	171	253		No conversion		
324	D4	212			11-0-8-4	AC	172	254		No conversion		
325	D5	213			11-0-8-5	AD	173	255		No conversion		
326	D6	214			11-0-8-6	AE	174	256		No conversion		
327	D7	215			11-0-8-7	AF	175	257		No conversion		
330	D8	216			12-11-0-8-1	B0	176	260		No conversion		
331	D9	217			12-11-0-1	B1	177	261		No conversion		
332	DA	218			12-11-0-2	B2	178	262		No conversion		
333	DB	219			12-11-0-3	B3	179	263		No conversion		
334	DC	220			12-11-0-4	B4	180	264		No conversion		
335	DD	221			12-11-0-5	B5	181	265		No conversion		
336	DE	222			12-11-0-6	B6	182	266		No conversion		
337	DF	223			12-11-0-7	B7	183	267		No conversion		
340	E0	224			12-11-0-8	B8	184	270		No conversion		
341	E1	225			12-11-0-9	B9	185	271		No conversion		
342	E2	226			12-11-0-8-2	BA	186	272		No conversion		
343	E3	227			12-11-0-8-3	BB	187	273		No conversion		
344	E4	228			12-11-0-8-4	BC	188	274		No conversion		
345	E5	229			12-11-0-8-5	BD	189	275		No conversion		
346	E6	230			12-11-0-8-6	BE	190	276		No conversion		
347	E7	231			12-11-0-8-7	BF	191	277		No conversion		
350	E8	232			12-0-9-8-2	CA	202	312		No conversion		
351	E9	233			12-0-9-8-3	CB	203	313		No conversion		
352	EA	234			12-0-9-8-4	CC	204	314		No conversion		
353	EB	235			12-0-9-8-5	CD	205	315		No conversion		
354	EC	236			12-0-9-8-6	CE	206	316		No conversion		
355	ED	237			12-0-9-8-7	CF	207	317		No conversion		
356	EE	238			12-11-9-8-2	DA	218	332		No conversion		
357	EF	239			12-11-9-8-3	DB	219	333		No conversion		
360	F0	240			12-11-9-8-4	DC	220	334		No conversion		
361	F1	241			12-11-9-8-5	DD	221	335		No conversion		
362	F2	242			12-11-9-8-6	DE	222	336		No conversion		
363	F3	243			12-11-9-8-7	DF	223	337		No conversion		
364	F4	244			11-0-9-8-2	EA	234	352		No conversion		
365	F5	245			11-0-9-8-3	EB	235	353		No conversion		
366	F6	246			11-0-9-8-4	EC	236	354		No conversion		
367	F7	247			11-0-9-8-5	ED	237	355		No conversion		
370	F8	248			11-0-9-8-6	EE	238	356		No conversion		
371	F9	249			11-0-9-8-7	EF	239	357		No conversion		
372	FA	250			12-11-0-9-8-2	FA	250	372		No conversion		
373	FB	251			12-11-0-9-8-3	FB	251	373		No conversion		
374	FC	252			12-11-0-9-8-4	FC	252	374		No conversion		
375	FD	253			12-11-0-9-8-5	FD	253	375		No conversion		
376	FE	254			12-11-0-9-8-6	FE	254	376		No conversion		
377	FF	255	50 Eight ones		12-11-0-9-8-7	FF	255	377		No conversion		

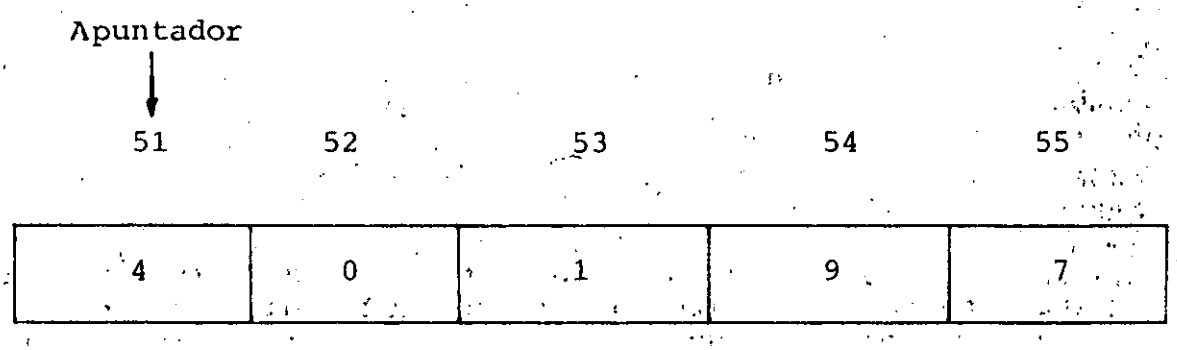
El modelo de Von Neumann

La idea central del modelo de computación propuesto por John Von Neumann es almacenar las instrucciones del programa de una computadora en su propia memoria, logrando así que la máquina siga los pasos definidos por su "programa definido".

Para lograr esto fue necesario resolver el problema de comunicarle a la computadora qué operaciones efectuar sobre los datos previamente almacenados en la memoria.

Abordaremos el problema del almacenamiento de nuestros números recordando que la función de la memoria es la del guardar datos. Para nuestros propósitos, la memoria será un conjunto de celdas (o casillas) con las siguientes características: a) cada celda puede contener un valor numérico y b) cada celda tiene la propiedad de ser "direccionable"; es decir, se puede distinguir una casilla de otra por medio de un número unívoco llamado su dirección. Esto implica que las celdas de la memoria tienen que estar organizadas de modo que faciliten encontrar cualquiera de ellas con un mínimo de esfuerzo. La forma más sencilla de lograrlo es organizándolas en forma de un vector, que no es más que un conjunto numerado (secuencialmente) de celdas.

Podemos referenciar una celda, como dijimos, por medio de su dirección. Usaremos un apuntador para dirigirnos a alguna celda cualquiera. Un arreglo (vector) de memoria se ve como sigue:



Cada casilla tiene una dirección. Por ejemplo, la casilla 51 contiene un 4.

Disponemos ya de una manera de almacenar (y recuperar) valores en la memoria por medio de una dirección unívoca. Definiremos dos operaciones elementales sobre ellas: leer el contenido de una casilla y escribir un valor en una casilla.

Si suponemos que la memoria de una computadora es una especie de almacén, atendido por un empleado que seguirá nuestras órdenes, éstos serán los pasos necesarios para poder efectuar las dos operaciones primitivas:

Para leer:

a) Decidir cuál casilla se va a leer (esto es, proporcionar su dirección).

b) Esperar un tiempo fijo a que el empleado vaya a la memoria y traiga el valor depositado en esa casilla (la casilla no pierde ese valor el ayudante sólo trae una copia del dato y no el dato mismo).

c) Recoger ese dato, y dar por terminada la operación de lectura.

Y para escribir:

a) Proporcionar al ayudante el dato que deseamos sea depositado en una casilla.

b) Proporcionarle la dirección de la casilla sobre la que se desea hacer la escritura del dato.

c) Esperar un tiempo fijo a que el empleado vaya a la memoria y deposite el dato en la casilla designada, para dar por terminada la operación de escritura.

Tenemos ahora que resolver el segundo problema: Cómo almacenar las instrucciones en la memoria. Si queremos hacer uso de lo que ya definimos, tendremos que encontrar una manera de "hacer caber" las instrucciones en las casillas. Esta nos llevará necesariamente al concepto de codificación. En efecto, si en las celdas de memoria sólo caben números, pues entonces habrá que traducir las instrucciones a números, para poderlas almacenar.

Para codificar nuestras instrucciones debemos considerar cuántas (y cuáles) instrucciones hay y qué esquema de codificación vamos a usar.

El primer factor depende fundamentalmente de la capacidad de la unidad de control para hacer operaciones; mientras más compleja -y cara- sea la unidad de control, más instrucciones diferentes podrá efectuar. Después, debemos encontrar un código adecuado para que a cada instrucción definida le corresponda uno y sólo un valor numérico. Para este segundo caso.

Introducción.

Usaremos un "diccionario electrónico" (residente también en la unidad de control), que dirá más o menos lo siguiente.

INSTRUCCION	CODIGO INTERNO
SUMA	57
RESTA	42
.	.
.	.
.	.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA COMPUTACION ELECTRONICA Y PROGRAMACION

EQUIPO

ING. BENITO ZYCHINSKI

OCTUBRE, 1984

INTRODUCCION A LA COMPUTACION

ELECTRONICA Y PROGRAMACION

CAPITULO III

EQUIPO

. Unidad Central de Proceso *

. Unidades E/S *

. Unidad de memoria aux. *

. Sistema de Explotación de Cómputo *

Ing. Benito Zychlinski

* Notas tomadas de la revista Enciclopedia Práctica de la Informática, Nueva Lente/ INCE/ck, números 1, 3 y 15.

Octubre, 1984

INFORMATICA BASICA

LA ARQUITECTURA DE LOS ORDENADORES

DE una forma muy simple podemos decir que un ordenador consta de dos zonas fundamentales: la unidad central de proceso (UCP o CPU, según utilicemos las siglas castellanas o inglesas) que es la encargada de la ejecución de los programas y varias unidades periféricas que permiten al ordenador comunicarse con el exterior, bien sea para capturar datos y mostrar resultados, o bien para almacenar la información.

Unidad Central de Proceso

El auténtico «cerebro» del ordenador es la unidad central de proceso (CPU), en torno a la cual se organizan el resto de los elementos del sistema. En la CPU de los ordenadores convencionales suelen distinguirse tres zonas básicas:

La memoria principal

En ella se almacenan dos tipos de información: el programa o conjunto de instrucciones a ejecutar y los datos que manejarán dichas instrucciones. La memoria está constituida por un conjunto de células capaces de almacenar un dato o una instrucción. Con el fin de que la unidad de control pueda diferenciar a cada una de las células, éstas van numeradas; al número que identifica a una célula se le llama dirección. Una vez determinada la dirección de una célula, se puede leer la información que contiene o escribir una nueva información en su interior. Para poder realizar estas operaciones la memoria dispone de dos registros especiales: el registro de dirección y el registro de intercambio o de datos. Según se vaya a efectuar una operación de «lectura» o de «escritura», se seguirán los siguientes pasos:

— *Lectura:*

1. Almacenar la dirección de la célula en la que se encuentra la información a leer en el registro de dirección.
2. Cargar en el registro de intercambio la información contenida en la célula apuntada por el registro de dirección.
3. Transferir el contenido del registro intercambio al registro de la CPU que corresponda.

— *Escritura:*

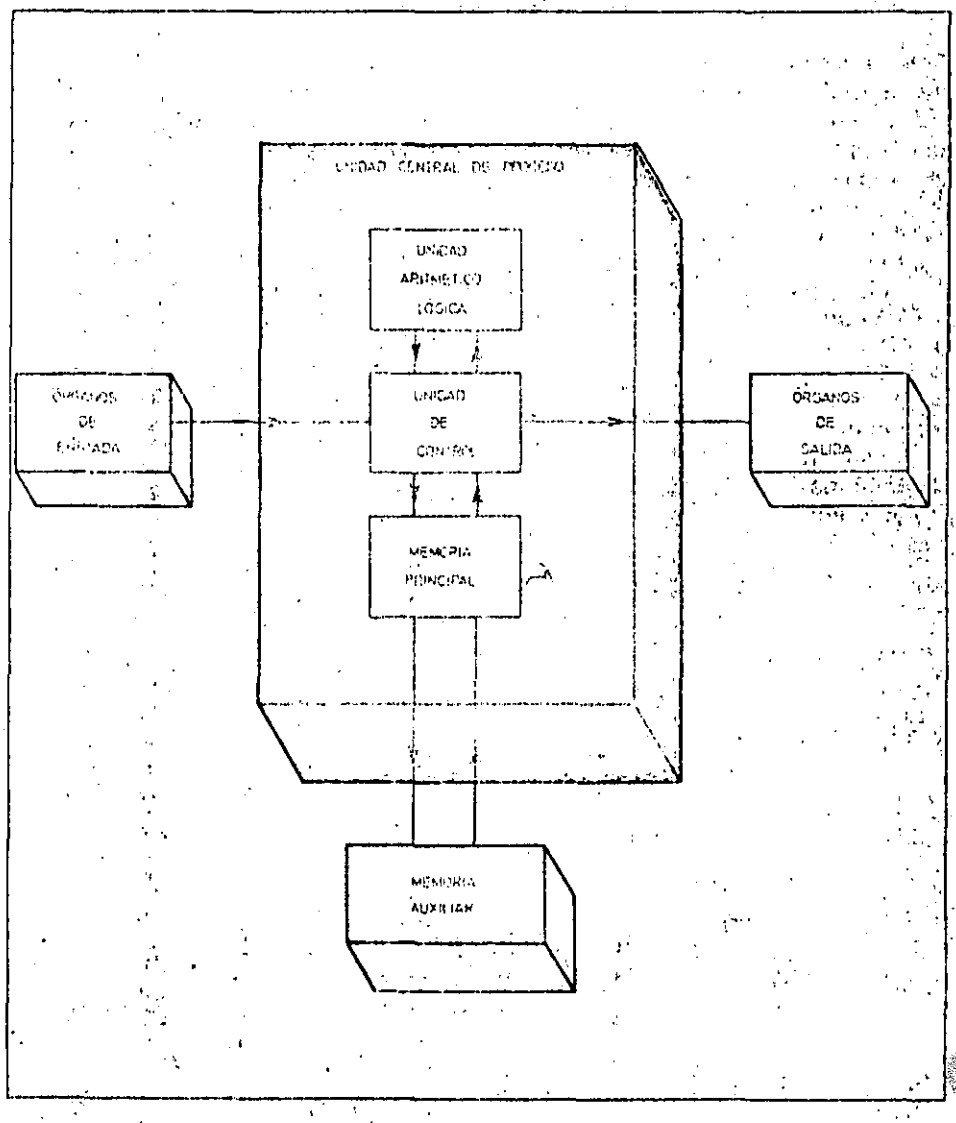
1. Transferir al registro de intercambio la información a escribir.
2. Almacenar la dirección de la célula receptora de la información en el registro de dirección.
3. Cargar el contenido del registro de intercambio en la célula apuntada por el registro de dirección.

Evidentemente, las operaciones de lectura no destruyen la información almacenada en la célula, cosa que, por el contrario, sí ocurre con las operaciones de escritura, ya que la destruyen al sustituirla por una nueva información.

La unidad de control

Esta unidad es la que se ocupa de controlar y coordinar el conjunto de operaciones que hay que realizar para dar el oportuno tratamiento a la información. Su cometido obedece a las indicaciones contenidas en el programa; como resultado de su «interpretación», la unidad de control genera el conjunto de órdenes elementales que revertirán en la ejecución de la tarea solicitada. En líneas generales, su actuación se concreta en los siguientes puntos:

1. Extrae de la memoria principal la instrucción a ejecutar. Para ello dispone de un registro denominado «contador de instrucciones» (o contador de programas), en el que almacena la dirección de la célula que contiene la próxima instrucción a ejecutar y de un



Dentro de la arquitectura de todo ordenador cabe distinguir dos zonas básicas: la unidad central de proceso y el conjunto de órganos periféricos. En la primera de estas zonas es donde residen las unidades que permiten al ordenador realizar su trabajo: el tratamiento de la información.

LA ARQUITECTURA DE LOS ORDENADORES

segundo registro «de instrucción», en el que deposita la instrucción propiamente dicha. Este último registro está dividido en dos zonas: una contiene el código de operación que identifica la operación a ejecutar (suma, resta...) y la segunda contiene la dirección de la célula en la que está almacenado el operando.

2. Una vez conocido el código de la operación, la unidad de control ya sabe qué circuitos de la unidad aritmético-lógica deben intervenir y puede establecer las conexiones eléctricas necesarias a través del secuenciador.

3. A continuación extrae de la memoria principal los datos necesarios para ejecutar la instrucción en proceso, para ello simplemente ordena la lectura de la célula cuya dirección se encuentra en la segunda zona del registro de instrucción.

4. Ordena a la unidad aritmético-lógica que efectúe las oportunas operaciones elementales. El resultado de este tratamiento se deposita en un registro especial de la unidad aritmético-lógica denominado «acumulador».

5. Si la instrucción ha proporcionado nuevos datos, estos son almacenados en la memoria principal.

6. Por último, incrementa en una unidad el contenido del contador de instrucciones, de tal forma que coincida con la dirección de la próxima instrucción a ejecutar. Algunas operaciones, como, por ejemplo, las de bifurcación, se limitan a modificar el contador de instrucciones, de forma que la siguiente instrucción a procesar no sea la que se encuentra inmediatamente a continuación de la que está en curso.

La unidad aritmético-lógica

La misión de la unidad aritmético-lógica es operar los datos que recibe siguiendo las indicaciones dadas por la unidad de control. El motivo por el que a esta unidad se le otorga el nombre de «aritmético-lógica», es que puede realizar tanto operaciones aritméticas

El diálogo entre el ordenador y el mundo exterior —por ejemplo, el usuario del sistema— se practica a través de los órganos periféricos: teclados para entrada de datos, pantallas para la visualización de informaciones entregadas por el ordenador...

como operaciones basadas en la lógica Booleana.

Para que la unidad aritmético-lógica sea capaz de realizar una operación aritmética, por ejemplo una suma, se le deben proporcionar los siguientes datos:

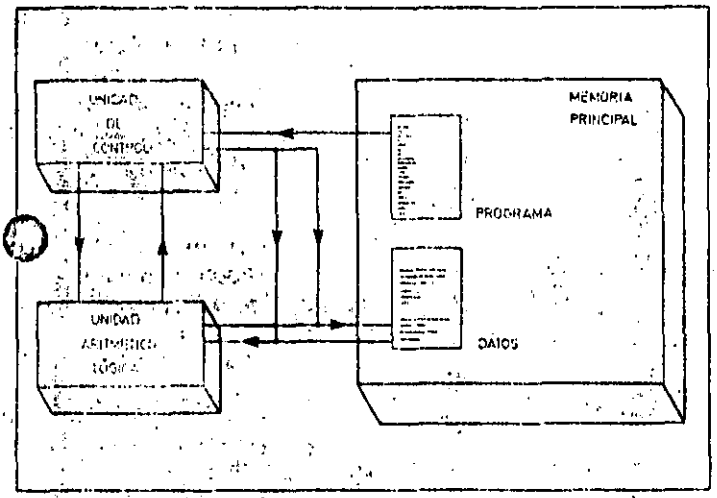
1. Código de operación, que indique la operación a efectuar; en este caso sería el código de la suma.
2. Dirección de la célula en la que se encuentra almacenando el primer sumando.
3. Dirección del segundo sumando.
4. Dirección de la célula en la que se almacenará el resultado.

Cabe observar que en el formato de instrucción que hemos considerado a lo largo de esta descripción general de un ordenador, sólo se dispone de un código de operación y una única dirección de operando (en los ordenadores actuales los formatos de las instrucciones contienen toda la información necesaria). El hecho de que esta instrucción tan condensada se traduzca en un proceso de suma se debe a que, al interpretar su código de operación, la unidad de control genera una secuencia de tres microinstrucciones elementales que afectan al registro especial que hemos denominado «acumulador». En éste es en donde se almacenan los resultados de las sucesivas operaciones. Las tres microinstrucciones elementales que dan lugar a la operación suma —afectando a una sola dirección: el acumulador— son las siguientes:

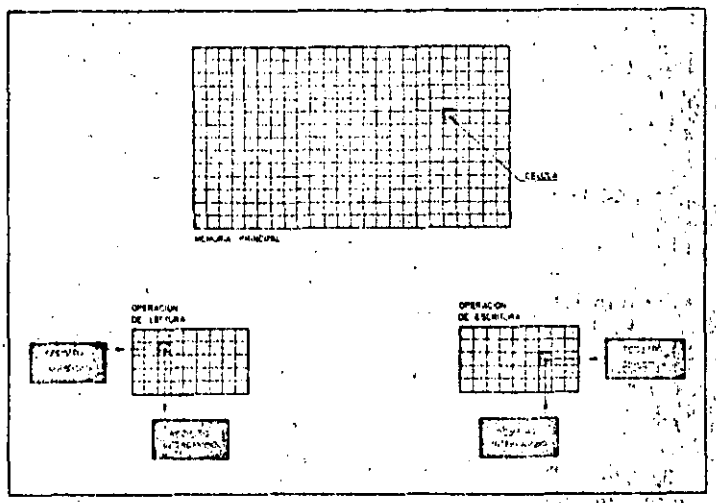
a) Cargar el primer operando en el acumulador.
b) Sumar el segundo operando con el contenido del acumulador.
c) Cargar el contenido del acumulador en la dirección del resultado.

Unidades periféricas

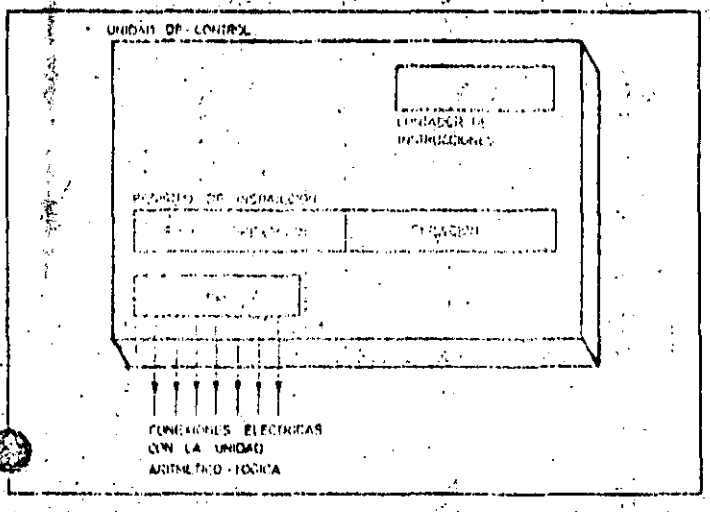
Podemos distinguir dos grandes grupos de unidades periféricas. Las unida-



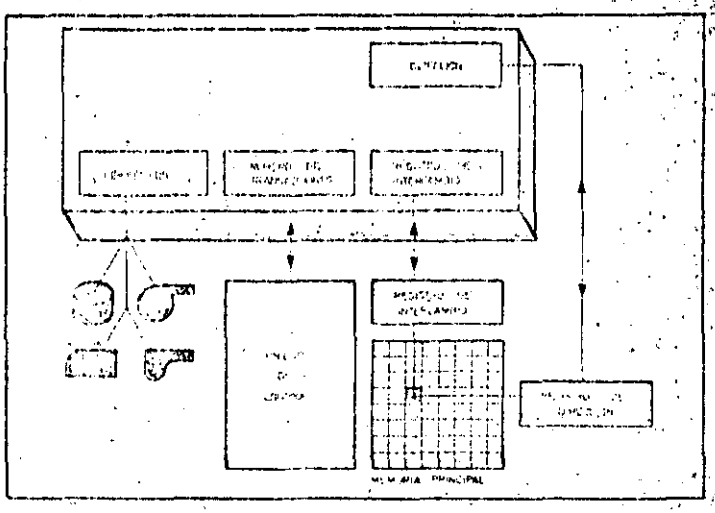
Los tres bloques fundamentales que integran la unidad central de proceso del ordenador controlan, operan y coordinan la actividad del sistema que, en líneas generales, se resume en la lectura e interpretación de un programa almacenado y en su ejecución.



La memoria principal de un sistema ordenador almacena dos tipos de información: programas o conjuntos ordenados de instrucciones y datos. Las operaciones que se realizan sobre esta unidad se reducen a dos: lectura y escritura.



La unidad de control es el auténtico «cerebro» que controla y coordina el funcionamiento del ordenador. A raíz de la interpretación de las instrucciones que integran el programa, esta unidad genera el conjunto de órdenes elementales necesarias para que se realice la tarea solicitada.



La transferencia de informaciones entre el ordenador y los periféricos se realiza a través de determinadas unidades «adaptadoras» denominadas canales. Su capacidad de gestionar y controlar la transferencia de informaciones descarga a la unidad central de este tipo de tareas.

LA ARQUITECTURA DE LOS ORDENADORES

Glosario

¿Cuáles son las dos zonas fundamentales de un ordenador?
La unidad central de proceso (CPU) que se encarga de la ejecución de los programas y del control de las restantes unidades y los dispositivos periféricos.

¿Cuáles son los componentes básicos de la CPU?
La memoria principal, la unidad de control y la unidad aritmético-lógica.

¿Cómo funciona la memoria principal?
Mediante un conjunto de células numeradas y dos registros especiales con los que realiza las transacciones: el registro de dirección que indica el número de la célula afectada y el de intercambio que contiene la información leída o la que hay que escribir en la célula en cuestión.

¿Cuál es el objetivo de la unidad de control?
Controlar la ejecución de las instrucciones del programa; para ello cuenta con dos registros primarios: uno de ellos memoriza el número de la instrucción en curso, mientras que el segundo almacena la instrucción propiamente dicha.

¿Qué tareas realiza la unidad aritmético-lógica?
Tal como su nombre indica, se encarga de ejecutar las operaciones aritméticas y lógicas, almacenando el resultado en un registro llamado acumulador.

¿Qué son las unidades periféricas?
Son dispositivos que se ocupan de facilitar el diálogo entre el ordenador y el mundo exterior o de almacenar grandes volúmenes de información y mantenerla a disposición del ordenador.

¿Qué es un canal?
Es una unidad encargada de realizar las transacciones de información entre la unidad de control y los periféricos. Su utilidad estriba en que descargan a la unidad central del control directo de la entrada y salida de datos.

des de comunicación que permiten el diálogo con el exterior (de entrada o salida) y las memorias auxiliares que sirven para almacenar grandes volúmenes de datos de forma permanente. Como ejemplos de periféricos de comunicación podemos citar el lector de tarjetas perforadas, el teclado, la impresora, la pantalla de operador... y como ejemplo de memorias auxiliares: discos, cintas magnéticas, etc.

La comunicación entre los periféricos y el ordenador se realiza a través de ciertas unidades, denominadas «canales», que se ocupan de gestionar la transferencia de información.

En los ordenadores actuales las transferencias a través de los canales se pueden simultanear con el desarrollo de un programa de cálculo, ya que el canal sólo necesita la unidad periférica implicada en la entrada o salida y la dirección de la célula de la memoria principal en la que se leerá o escribirá la información. El canal mantiene un contador con el número de informaciones a transferir, de forma que le indique el trabajo que tiene pendiente, para ello incrementa una unidad al contador cada vez que le llegue una información para transferir y le restará una unidad cada vez que realice una transferencia; cuando el contador esté a O, el canal advertirá a la unidad de control que ha finalizado la transferencia de información.

El sistema binario

El conjunto de símbolos utilizados en este sistema de numeración se limita a dos (0, 1), en consecuencia, la única forma de representar un número binario es mediante una cadena de dígitos binarios o «BITS»: ceros y unos.

Este sistema de numeración, ideado por Leibnitz en el siglo XVII, constituye el alfabeto interno de los ordenadores electrónicos. La correspondencia en el sistema decimal de cualquier número binario se obtiene aplicando la siguiente expresión:

$$N = d_n 2^n + d_{n-1} 2^{n-1} + \ldots + d_2 2^2 + d_1 2 + d_0$$

¿Cuántos dígitos binarios se necesitan para representar cualquier número que en base decimal sea inferior o igual a «m»?

Si razonamos por inducción, deduciremos que con un dígito binario se puede representar hasta el número natural 1, con dos dígitos binarios hasta $1 \times 2 + 1 = 3$, y con tres dígitos binarios hasta $1 \times 4 + 1 \times 2 + 1 = 7$. Se puede observar que, en general, con «n» dígitos binarios podemos representar hasta el número decimal $2^n - 1$. Luego la respuesta a la cuestión planteada es que son necesarios «n» dígitos binarios, de tal forma que «n» satisfaga las condiciones siguientes:

$$2^n \geq m \text{ y } 2^{n-1} < m$$

Por ejemplo, si m = 8, el número de dígitos binarios «n» será 3, puesto que: $2^3 = 8 \geq 8$ y $2^2 = 4 < 8$.

En el caso de m = 9, resultarán n = 4, dado que $2^4 = 16 \geq 9$ y $2^3 = 8 < 9$.

En definitiva, podemos afirmar que con «n» dígitos binarios (bits) se pueden obtener hasta 2^n cadenas o configuraciones distintas.

Número Binario	Expresión	Número Decimal
0	0	0
1	1	1
10	$1 \times 2 + 0$	2
11	$1 \times 2 + 1$	3
100	$1 \times 4 + 0 \times 2 + 0$	4
101	$1 \times 4 + 0 \times 2 + 1$	5
110	$1 \times 4 + 1 \times 2 + 0$	6
111	$1 \times 4 + 1 \times 2 + 1$	7
1000	$1 \times 8 + 0 \times 4 + 0 \times 2 + 0$	8
1001	$1 \times 8 + 0 \times 4 + 0 \times 2 + 1$	9
1010	$1 \times 8 + 0 \times 4 + 1 \times 2 + 0$	10
1011	$1 \times 8 + 0 \times 4 + 1 \times 2 + 1$	11

LOS DISPOSITIVOS PERIFERICOS

En el terreno de la informática, se denomina periférico a todo dispositivo que permite la comunicación del ordenador con el mundo exterior. Este mundo exterior puede ser la persona humana (el usuario) o bien cualquier tipo de dispositivo que esté controlado por el ordenador o que provea al mismo de información.

Atendiendo a su relación genérica con el elemento de trabajo de los ordenadores, «la información», podemos distinguir tres categorías de periféricos:

- Periféricos de entrada.
- Periféricos de salida.
- Periféricos de almacenamiento.

Periféricos de entrada son aquellos mediante los que se introduce en el ordenador la información que va a ser objeto de tratamiento.

Periféricos de salida son aquéllos a través de los que el ordenador entrega información al mundo exterior (por ejemplo, el resultado de las operaciones realizadas).

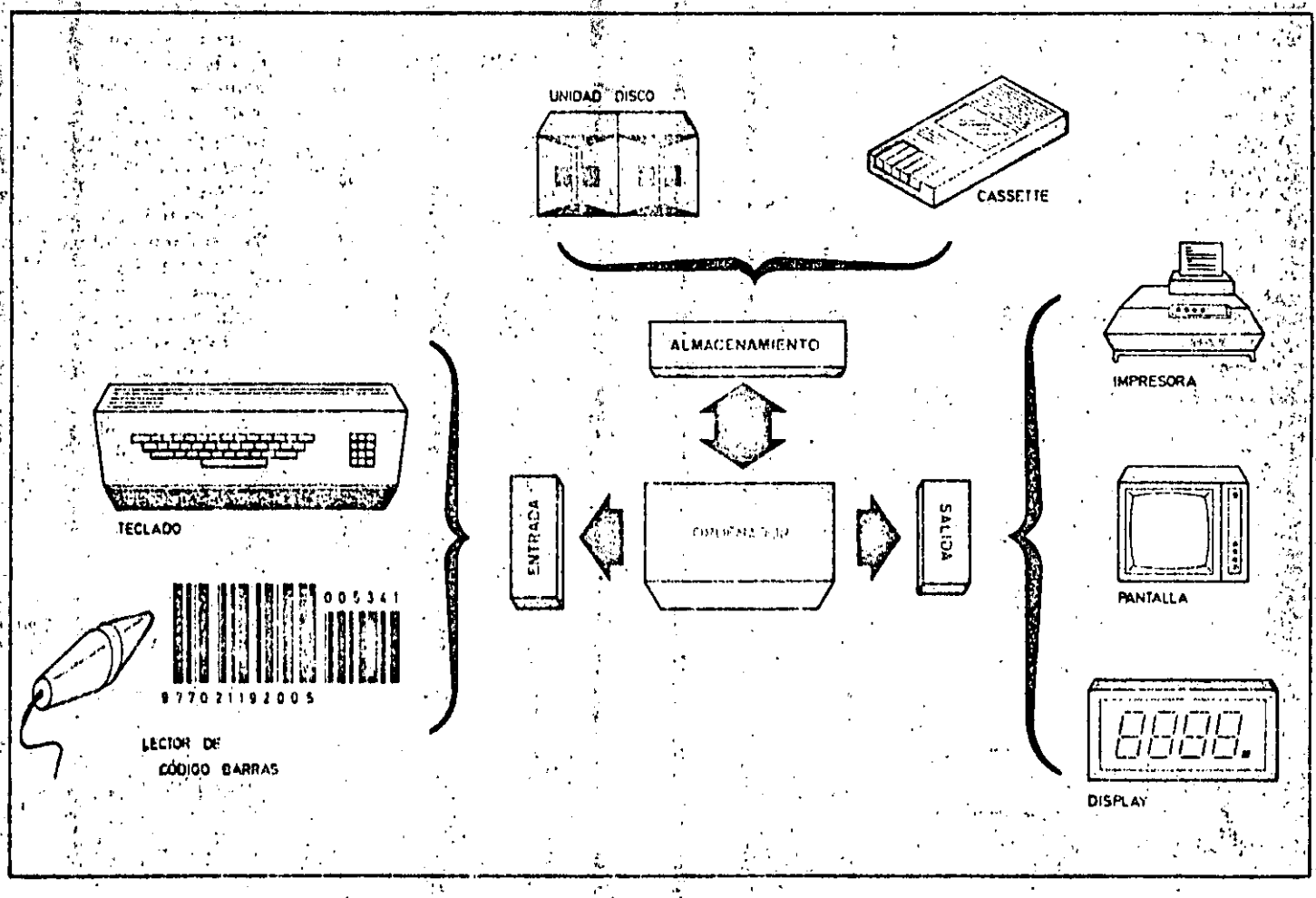
Periféricos o unidades de almacenamiento son aquéllos en los que se apoya el ordenador en su trabajo, utilizándolos como «archivo» de información. El ordenador entrega información a estas unidades, que se ocupan de almacenarla hasta el instante en el que el ordenador la necesite.

Hay periféricos que comparten las características propias de varias de las categorías establecidas. Así, por ejemplo, un terminal está compuesto por un teclado (periférico de entrada) a través del que se suministran datos al ordenador, y una pantalla de rayos catódicos (periférico de salida), por medio de la que el ordenador presenta los resultados al mundo exterior.

Además, hay periféricos que pueden usarse de distinta forma según la ocasión: una unidad de disco flexible puede emplearse como periférico de entrada para suministrar datos al ordenador, como periférico de salida ofreciendo un soporte de la información resultante, o bien, como periférico de almacenamiento en su sentido más estricto.

En resumidas cuentas, el ordenador no es más que una máquina electrónica cuyos circuitos internos operan señales de esta índole. Los periféricos son, pues, los encargados de transformar la información de entrada en señales electrónicas inteligibles por el ordenador, o de «traducir» las señales de salida del ordenador, de forma que pueda entenderlas el usuario o el dispositivo que constituya su «mundo exterior».

Por ejemplo, un teclado transforma la pulsación de una tecla en una señal



Los periféricos o dispositivos que permiten la comunicación del ordenador con el mundo exterior, pueden clasificarse, en tres grandes grupos: periféricos de entrada, de salida o de almacenamiento de información.

LOS DISPOSITIVOS PERIFERICOS

electrónica atendiendo a una determinada codificación. Una impresora convierte la señal electrónica que le llega del ordenador en un carácter que se plasma en un un papel y que puede ser leído.

Tipos de periféricos

La evolución de los sistemas informáticos ha provocado el nacimiento de una gran diversidad de dispositivos periféricos, algunos ni tan siquiera soñados hace una década. De entre ellos los más importantes son:

- Impresoras.
- Terminales.
- Modems.
- Unidades de disco.
- Unidades de cinta magnética.
- Trazadores gráficos o plotters.
- Lectores de código de barras.
- Interfaces industriales.
- Lectores y perforadores de cinta.
- Lectores y perforadores de tarjetas.
- Memorias de burbujas.
- Lectores de tarjetas magnéticas.
- Lápiz óptico (light pen).
- Digitalizadores.
- Displays.
- Lectores de caracteres ortográficos.
- Monitores de rayos catódicos.
- Unidades de síntesis y reconocimiento de la voz.

Impresoras: son periféricos de salida que imprimen en un papel los datos que reciben del ordenador.

Terminales: combinación de periférico de entrada y salida; constan de un teclado para la introducción de datos y de una pantalla para la visualización de resultados.

Modems: periféricos de entrada / salida que permiten la comunicación entre el ordenador y el mundo exterior a través de una línea telefónica.

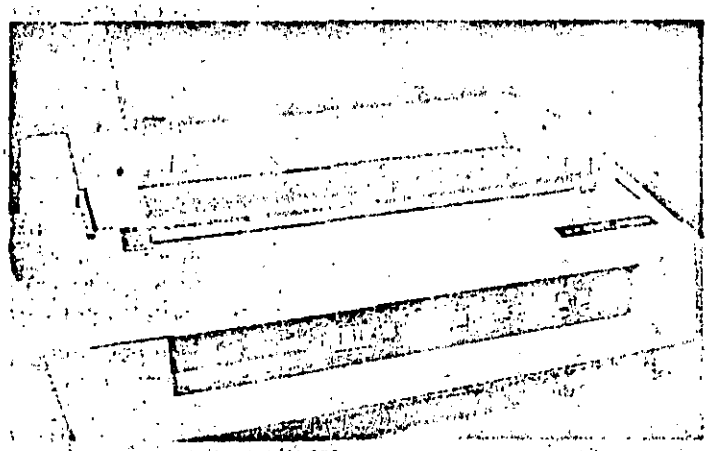
Unidades de disco: normalmente se usan como periféricos de almacenamiento, en los que el soporte que memoriza la información es un disco de tipo flexible o rígido.

Unidades de cinta magnética: pueden ser de tipo cassette. Se emplean como unidades de almacenamiento, que gestionan la lectura o escritura de datos en una cinta magnética.

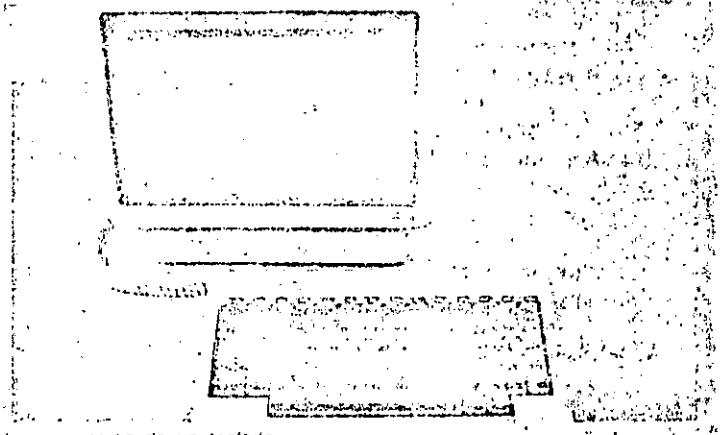
Trazadores gráficos o Plotters: trazan gráficos sobre papel con los datos que les suministra el ordenador.

Lectores de barras: su contenido consiste en la captación de datos a partir de unas barras impresas con determinada codificación.

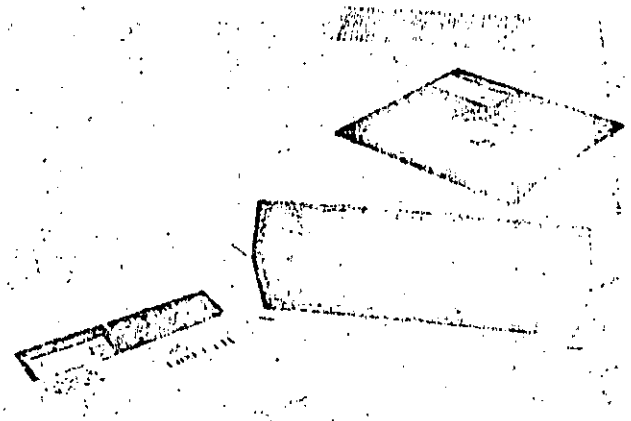
Interfaces industriales: por medio de estos periféricos, el ordenador puede



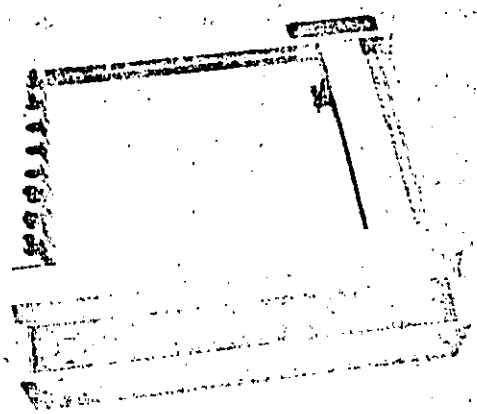
Las impresoras son los periféricos de salida más generalizados. Imprimen sobre papel la información que reciben del ordenador.



La conjunción de un teclado y un monitor o pantalla de vídeo constituye el «terminal»: un periférico doble, de entrada (teclado) y salida (pantalla).



El periférico de almacenamiento de información más común en el campo de los microordenadores es la unidad de disco. Su cometido consiste en grabar o leer información en un disco flexible, normalmente, de 5 1/4 o de 8 pulgadas.



El trazador gráfico o «plotter» es un periférico de salida que traza dibujos sobre papel a partir de la información suministrada por el ordenador.

controlar procesos industriales, tomando lecturas de presiones, temperaturas, etc., y dando órdenes de arranque o parada de motores, apertura o cierre de válvulas, etc.

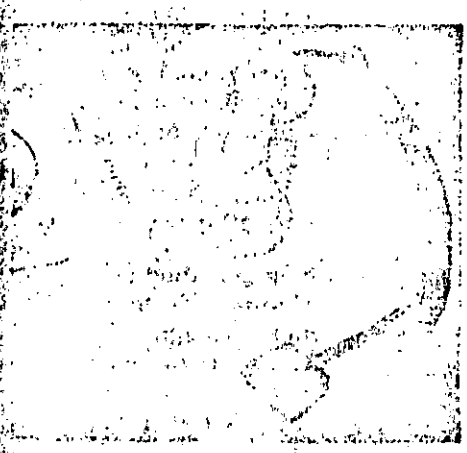
Lectores y perforadores de cinta: leen o perforan una cinta de papel según una determinada codificación.

Lectores y perforadores de tarjetas: leen o perforan tarjetas de cartulina que constituyen un soporte de información.

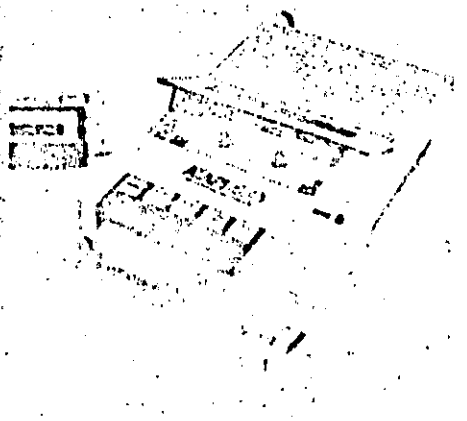
Memoria de burbujas: unidades de almacenamiento de avanzada tecnología.

Lectores de tarjetas magnéticas: leen los caracteres existentes en una banda magnética adherida a una tarjeta (por ejemplo, tarjetas de crédito).

Lápiz óptico (Light pen): permite la introducción de datos aplicando el dispositivo sobre una pantalla de rayos catódicos.



Los modems son periféricos de entrada y salida que permiten al ordenador comunicarse con otros equipos a través de una línea telefónica



Los ordenadores personales han dado paso a un gran número de nuevos dispositivos periféricos. Uno de los más representativos y habituales es el magnetófono de cassette.

Digitalizadores: se utilizan para codificar e introducir en el ordenador datos directamente extraídos de un dibujo o de un plano.

Displays: periféricos de salida a través de los que se visualizan datos.

Lectores de caracteres ortográficos: son capaces de leer caracteres escritos por medios convencionales e introducirlos en el ordenador.

Monitores de rayos catódicos: visualizan la información de salida del ordenador sobre una pantalla semejante a la de los receptores de TV.

Unidades de síntesis y reconocimiento de voz: son capaces de emular la voz humana (a partir de datos suministrados por el ordenador) o reconocerla, trasladándola codificada al interior del sistema al que estén asociados.

Conexión ordenador/periféricos

Un elemento a considerar es la forma en la que se establece la comunicación entre el ordenador y los periféricos; a esta adaptación es lo que suele denominarse «interface».

Dada la diversidad de ordenadores y periféricos, ha sido preciso establecer unas determinadas normas de comunicación que permita, en la medida de lo posible, la compatibilidad entre los distintos periféricos y ordenadores.

La transferencia de datos entre los ordenadores y los dispositivos periféricos suele realizarse —al igual que en nuestro lenguaje convencional— a partir de unidades elementales o «palabras». En este caso, el alfabeto que constituye las palabras es bastante reducido: sólo consta de ceros y unos.

Cada palabra o dato unitario está constituido por un conjunto de señales electrónicas que corresponden, cada una de ellas, a una «letra del alfabeto binario»: 0 ó 1. La comunicación de estos datos puede realizarse, básicamente, según dos métodos:

- Paralelo.
- Serie.

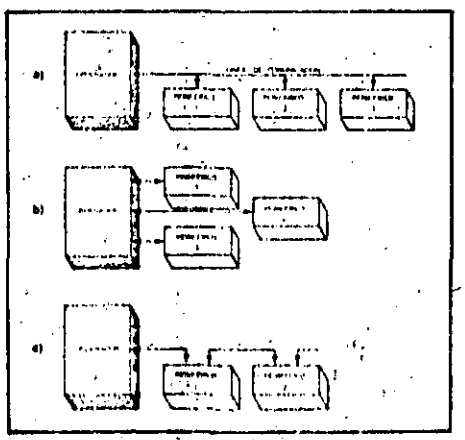
En el primer caso, todas las señales que integran una palabra o dato unitario se transfieren simultáneamente a través de un grupo de líneas paralelas.

Por el contrario, la comunicación es de tipo «serie» cuando las diversas señales se transfieren, una tras otra, sobre una misma línea de conexión.

Las normas de comunicación (o interface) más ampliamente adoptadas por los ordenadores y periféricos son las siguientes:

De tipo paralelo:
- Centronics.

De tipo serie:
- RS-232.
- V-24.
- Bucle de 20 mA.



La conexión de los dispositivos periféricos al ordenador puede realizarse de tres formas básicas: a) línea compartida; b) radial; c) en cadena

Asociación de periféricos al ordenador

La conexión de los dispositivos periféricos al ordenador puede realizarse de diversas formas:

a) *Línea compartida:* todos los periféricos se comunican a través de un solo «bus» o conjunto de líneas.

b) *Radial:* cada periférico se comunica con el ordenador a través de su propio grupo de líneas o bus.

c) *Cadena:* las señales de comunicación se van propagando de un periférico a otro.

Una característica de gran interés reside en la velocidad de trabajo de los dispositivos periféricos. Debido a que el ordenador trabaja a mucha mayor velocidad, la rapidez de operación de un sistema está limitada por la velocidad de trabajo de sus periféricos.

Esta característica, al igual que otras propias de cada tipo de periférico, deben ser evaluadas a la hora de proceder a la elección, ya que de los periféricos depende en gran medida la operatividad del sistema informático.

MEDIOS MAGNETICOS DE ARCHIVO

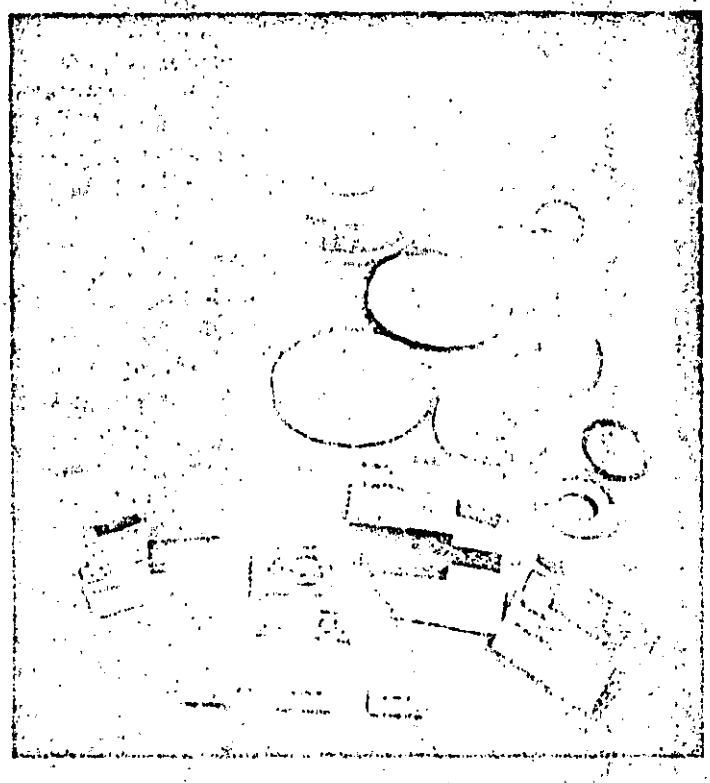
COMO ya se ha visto, los archivos de un ordenador deben residir en un medio que pueda ser leído o grabado por éste. En un principio se utilizaron como medios de almacenamiento de información tanto las tarjetas perforadas y las cintas perforadas de papel, como las cintas magnéticas, discos y tambores magnéticos. En la actualidad se han impuesto los medios magnéticos, debido a que presentan una serie de ventajas sobre los otros dispositivos. Estas ventajas se pueden resumir en las siguientes:

— Los medios magnéticos poseen una velocidad de transferencia de datos que es ideal para el tratamiento más eficaz de la información en archivos. Las tarjetas perforadas y las cintas de papel son demasiado lentas.
— Los medios magnéticos poseen una duración mucho mayor que la de los medios perforados.
— Los medios magnéticos pueden borrarse para reutilizarlos con una nueva información.
— A igualdad de espacio físico, los medios magnéticos almacenan más información que los medios perforados y permiten una longitud de registro prácticamente ilimitada.
— Los medios magnéticos son más baratos por carácter almacenado que los medios perforados, debido a su gran capacidad.

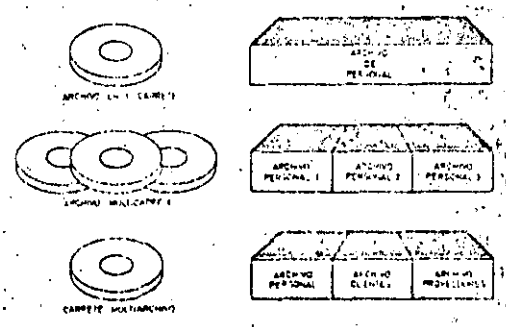
Archivos en cinta

Los registros se almacenan en orden secuencial y se procesan en serie. Es decir, el registro primero se procesa antes que el segundo, el segundo antes que el tercero, etc. Si se quisiera procesar los registros en un orden distinto, el operador tendría que recorrer la cinta completa para ir localizando cada registro, lo que acarrea una notable pérdida de tiempo.
Los registros del archivo en cinta magnética se agrupan en bloques separados por trozos de cinta sin grabar (GAP). La necesidad de esta agrupación de los registros es debida a que para leer la cinta la unidad debe acelerar desde una velocidad cero hasta la velocidad de lectura y, a continuación, disminuir la velocidad cuando se haya completado la lectura. Si se leyera de registro en registro estos tiempos de parada/aceleración serían interminables. Lo mejor es leer o grabar *bloques* para ahorrar tiempo y espacio de cinta. El tamaño de los bloques viene asignado por programa.

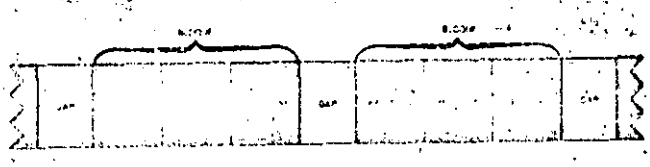
Al comienzo y al final de cada carrete existen varios metros de cinta en blanco, con el objeto de que el operador pueda montar la cinta. Se llama *cabecera* a la parte de cinta en blanco situada al comienzo del carrete y *cola* a la parte de cinta en blanco del final.
Para que el operador sepa cómo colocar correctamente el carrete, existen dos marcas especiales al comienzo y al final de la parte útil de la cinta. La primera, llamada «BOT» (Begining of Tape, principio de cinta) y la segunda, «EOT» (End of Tape, final de cinta).
Con el objeto de que los usuarios de los ficheros puedan identificar los carretes éstos tienen una etiqueta exterior donde aparece el nombre y el número del archivo. Suelen llevar también una



Los soportes magnéticos se han impuesto sobre el papel y las tarjetas perforadas como medios de archivo, debido a su mayor economía —relación precio capacidad— y a la posibilidad de reutilizarlos innumerables veces.



Con los archivos de cinta ocurre lo mismo que con los archivadores clásicos. Pueden almacenarse varios archivos en un solo archivador o, por el contrario, pueden ser necesarios varios archivadores físicos para almacenar un solo archivo.



Los GAP son zonas en blanco de la cinta que sirven para separar los bloques de información. Estos «huecos» permiten a la unidad de cinta alcanzar la velocidad de grabación o lectura adecuada entre cada arranque y parada.

MEDIOS MAGNETICOS DE ARCHIVO

etiqueta grabada magnéticamente con el mismo fin que la anterior.

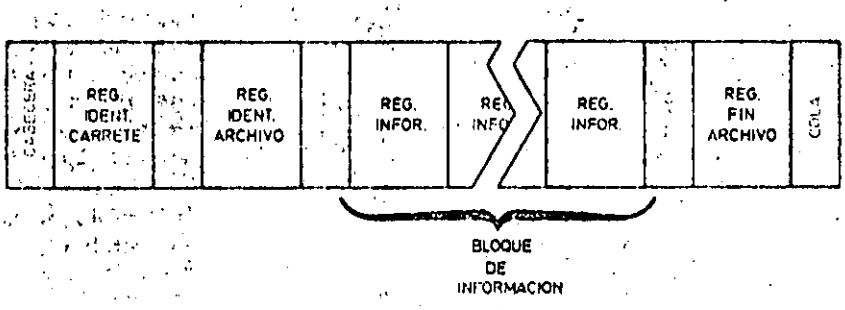
Puede ocurrir que un archivo ocupe un solo carrete o que ocupe, debido a su tamaño, más de un carrete, así como que un carrete almacene varios archivos. Veamos cómo se dispone la información en estos tres casos.

• En el archivo de un solo carrete existen dos bloques de un registro, situados al comienzo de la parte útil y separados uno de otro por un GAP, que indican la identificación del carrete y la del archivo respectivamente. A continuación está el primer bloque de registros de información o datos. A este bloque le seguirán los restantes del archivo. El final del archivo lo indicará un bloque de un solo registro, llamado de control.

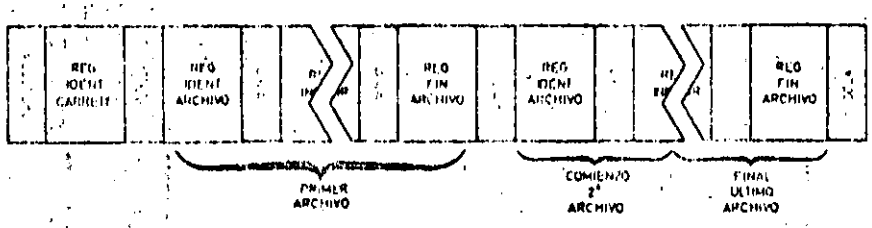
• Cuando un archivo ocupa varios carretes —multicarrete—, la parte del mismo que ocupa cada carrete debe ser indicada en la etiqueta exterior de cada uno de ellos. El final del primer carrete contiene solamente un registro que indica el final de cinta. En el siguiente carrete hay una etiqueta que ocupa dos bloques, el primero que indica el registro de identificación del carrete y el segundo que indica el registro de identificación de datos del archivo, precisando que estamos en el segundo carrete. El último bloque del carrete número 1 y el primero del número 2 deben ser bloques completos, no partes de un bloque.

La parte de un archivo multicarrete contenida en un carrete es a lo que se llama *sección* física del archivo. Si un archivo ocupa dos carretes se dice que contiene dos secciones físicas. El programa que procese este tipo de archivos debe contener las instrucciones necesarias para cambiar automáticamente de una sección a otra. Si se dispone de las unidades necesarias para hacerlo, el programa debe darle tiempo al operador para montar la cinta.

• Si un carrete tiene más de un archivo, se colocan marcas de control y etiquetas que identifiquen cada archivo. Estos carretes reciben el nombre de *multiarchivo*. Para localizar un archivo determinado se debe buscar desde el principio de la cinta el registro de identificación mismo.



Organización de un archivo en un solo carrete de cinta magnética.



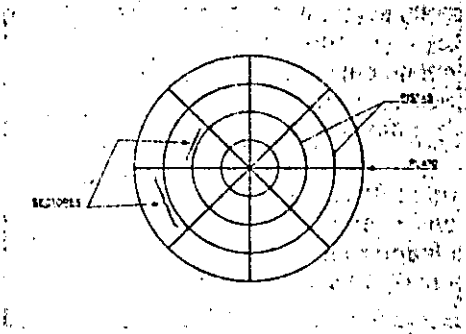
En de la figura puede observarse la organización de un carrete de cinta en la que se encuentran grabados varios archivos (organización multiarchivo).

Archivos en cassettes

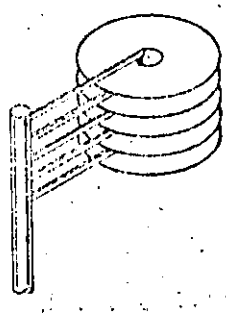
Vamos a terminar el estudio de la cinta magnética hablando de otro soporte magnético, muy utilizado actualmente en microordenadores y ordenadores personales; el cassette de cinta magnética.

Se emplea para almacenar archivos de menor tamaño que los de cinta. Tiene dos pistas para grabar y una etiqueta que identifica la pista a la que se accede.

El área de grabación es más estrecha que la de cinta en carretes y la densidad de grabación también es menor. La organización de los archivos es la misma que en la cinta. Es decir, archivo de un solo carrete, multicarrete y multiarchivos.



Organización de la información en discos magnéticos: pista —zona del disco que recorre el cabezal en una vuelta completa— y sector —divisiones dentro de cada pista.



Disposición en cilindro de una unidad de almacenamiento en disco rígido.

Archivos en disco

Así como la cinta magnética es un medio ideal para grabar los registros de un archivo en orden secuencial, el disco magnético es el medio utilizado con mayor eficacia para leer y grabar registros a los que se quiere acceder directamente. La lectura y grabación en las superficies del disco se realizan por medio de las llamadas cabezas de lectura y grabación, que se sitúan en un brazo.

Un archivo de disco se organiza a partir de varias unidades que se enumeran a continuación:

— Carácter: Contiene 8 bits de información: 1 byte.

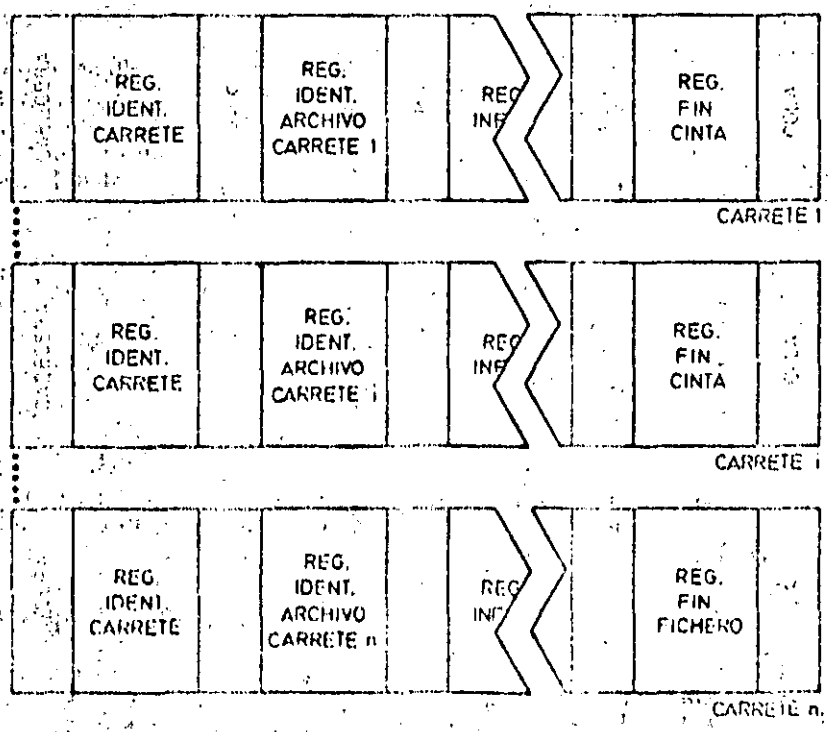
— Sector: Contiene generalmente 512 caracteres.

— Pista: Se llama así a la superficie del disco recorrida por la cabeza durante un giro completo del disco. Cada pista contiene normalmente 8 sectores.

— Disco: Contiene un cierto número de pistas. Existen unidades en las que el disco contiene hasta 1.024 pistas. Cada pista contiene el mismo número de informaciones, por lo que las pistas centrales, de menor longitud que las periféricas, se graban con mayor densidad.

La longitud de los «bloques» del disco viene determinada por el ordenador, y pueden ocupar de 1 a 8 sectores. Un archivo en disco también se puede dividir en secciones que pueden estar formadas por varios sectores contiguos del disco. Un archivo puede ocupar dos o más secciones de un disco o se pueden tener varias secciones de un archivo en discos diferentes. Cuando varios archivos o secciones de archivos se encuentran en un solo disco, es necesario crear el llamado «Directorio de archivos de disco», que contendrá la información referente al nombre del archivo, su situación dentro del disco, fecha de grabación, etc.

El lector puede tener problemas a la hora de distinguir entre un sector y una sección en un archivo de discos. Para evitar confusiones siempre hay que recordar que el sector es una división «física» del disco, mientras que la sección es una división «lógica» del archivo, realizada por el programador. Evidentemente, existe una estrecha relación



Cuando un archivo es muy extenso, son necesarios varios carretes de cinta para su completo almacenamiento. El gráfico muestra la organización de un archivo multicarrete; el último bloque de cada carrete debe ser un bloque completo.

Glosario

¿Se podría grabar un archivo en cinta sin agrupar los registros por bloques?

Sí se podría, pero no sería muy operativo, ya que los tiempos de accesó serían muy grandes al querer acceder a un registro o registros determinados. También la ocupación efectiva de la cinta sería muy baja.

¿Cuándo se emplea, fundamentalmente, la división de archivos por secciones?

Se emplea cuando se tiene que organizar un archivo multicarrete. La información grabada en cada carrete es la que recibe el nombre de sección. El programador, debe preparar el programa que procese este tipo de archivo, para que automáticamente pueda pasar de una cinta a otra cuando se llega al final de la sección. Este procedimiento se conoce con el nombre de «enlace de sección a sección».

¿Qué es un «cartucho de cinta magnética»?

Es un medio de almacenamiento de archivo parecido al cassette de cinta magnética, pero con una capacidad de almacenamiento mucho mayor.

¿A qué se denominan copias de seguridad de archivos?

Si se grabara una determinada información en un solo medio de archivo, podría ocurrir que, debido a cualquier fallo del sistema de proceso, se borrara parte de la grabación, con la consiguiente pérdida de información. Para evitar esto normalmente se hacen dos grabaciones de un mismo archivo. Una de ellas se utiliza para ser procesada y la otra se guarda con fines de seguridad. Esta última, llamada copia de seguridad, se graba en cinta magnética, tanto si la grabación original se hizo en cinta como si se hizo en disco, ya que la cinta es mucho más barata que el disco.

¿Las secciones tienen que corresponder siempre a un carrete?

No. A veces conviene crear en un solo carrete (sección física) varias secciones (secciones lógicas). Por ejemplo, la agrupación de información de un departamento.

MEDIOS MAGNETICOS DE ARCHIVO

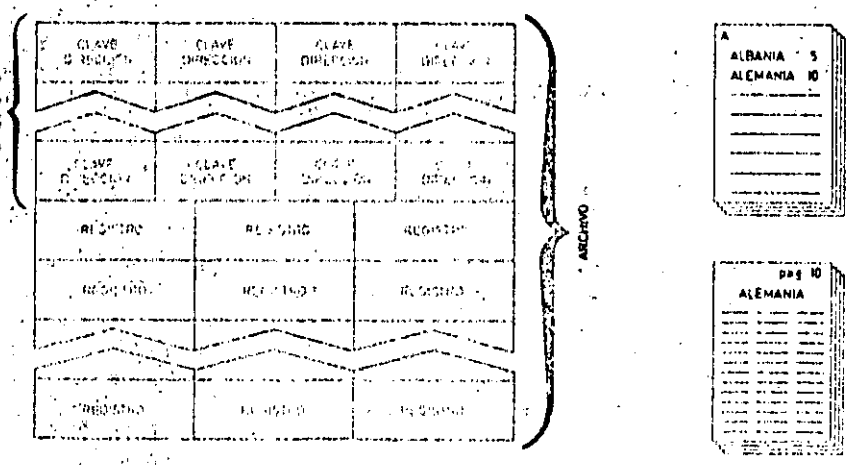
entre las unidades físicas de un archivo (disco, pista, sector) y las unidades lógicas (sección, bloque). Por ejemplo, un bloque está contenido, normalmente, en un sector. Una organización clásica de archivos en disco es el llamado «*cilindro*» que consiste en lo siguiente, si dispone de un paquete de discos con cabeza de lectura y grabación, para cada cara, el archivo se dispone de la siguiente manera: se comienza a grabar el archivo en la primera pista del primer disco. Cuando se graba esta pista, el archivo no continúa en la segunda pista del primer disco, sino en la primera del segundo disco; se completa ésta y se pasa a la primera del tercer disco y así sucesivamente. Con esto se logra que la lectura del archivo se haga rápidamente sin tener que esperar a que existan movimientos de las cabezas, ya que mientras se lee una pista de una cabeza, la siguiente ya está preparada para continuar la lectura. Se asemeja a un cilindro formado por todos los platos y de ahí le viene su nombre.

Archivo en disquette

El disco magnético tiene un «hermano menor», llamado disquette, disco flexible o floppy. Tiene menor capacidad de almacenamiento y se utiliza como elemento de almacenamiento de programas y datos. Lleva una etiqueta exterior para su identificación y su organización interna es similar a la del disco rígido descrito en el epígrafe anterior.

Tambor magnético

Vamos a terminar haciendo una breve descripción del tambor magnético, dispositivo que en la actualidad se utiliza poco debido a que tiene muchos problemas, en comparación con la cinta y el disco, a la hora de intercambiarlo. La información se graba en pistas, cada una de las cuales tiene su propia cabeza. Se utiliza de una manera más eficaz que la cinta cuando se quiere acceder a una parte determinada del archivo y es un medio ideal para almacenar archivos permanentes. Utilizan directorios y los archivos en tambores se pueden dividir en *segmentos* a los que se puede acceder individualmente, por lo que se pueden utilizar como medios de acceso directo.



Con objeto de poder identificar cada archivo y sus correspondientes registros se crea el denominado «directorio del archivo». El directorio guarda un gran paralelismo con el índice alfabético de un libro.

Conceptos básicos

Directorio de archivos

Ya hemos visto cómo en un paquete de discos pueden existir almacenados varios archivos o varias secciones de un archivo. Para poder identificarlo completamente se usa lo que se llama un «directorio de archivo».

El directorio contiene el nombre que se le asigna al archivo, la posición del archivo o de su sección en el disco, la fecha de grabación o actualización y también la fecha en que caduca la información almacenada.

El directorio se crea cuando se va a grabar la información en el disco. Es el sistema operativo del ordenador el que realmente controla el directorio grabándolo y manteniéndolo. Cuando el usuario quiere saber de qué archivos está compuesto un paquete de discos, tiene que acceder al directorio con la ayuda del sistema operativo del ordenador. De lo dicho se deduce inmediatamente que si se modifica un archivo, es decir, se actualiza, esa modificación debe aparecer en el directorio del archivo y es el sistema operativo el encargado de reflejarlo.

Cuando un paquete de discos contiene varios archivos, cada uno de éstos tiene una entrada en el directorio que le es asignada por el sistema operativo cuando se graba. Si el usuario quiere saber si puede disponer de un determinado archivo para procesarlo, tiene que acceder al directorio, con el objeto de cerciorarse de su disponibilidad.

En todos los sistemas existe una rutina de utilidad que permite a cualquier programador imprimir el directorio y estudiar la configuración del paquete de discos.

Para ayudar al programador a localizar el sector que contiene un registro determinado se pueden crear directorios dentro del mismo archivo de datos.

Este tipo de directorios es típico de organizaciones de archivo directo e indexado. El directorio contiene la clave del registro y la dirección y es en realidad un archivo de referencia creado por el sistema operativo o por el mismo programador. A este tipo de directorio se le conoce con el nombre de directorio de índice de claves.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA COMPUTACION ELECTRONICA Y PROGRAMACION

SISTEMAS Y PROGRAMAS

ING. MARIO PALOMAR

OCTUBRE , 1984

SISTEMAS Y PROGRAMAS

. Programación de Sistemas *

. Elementos de Programación *

. Programación Estructurada *

. Lenguajes de Programación *

Ing. Mario Palomar

* Notas tomadas de la enciclopedia Práctica de la Informática, Nueva Lente/INGE/ek. Números 3, 9, 10, 11, 12, 13, 20, 21, 22, 23 y 25.

Octubre, 1984.

SOFTWARE

LOS LENGUAJES INFORMATICOS

PARA que el ordenador pueda llevar a cabo los procesos que desee el usuario es necesario proporcionarle el adecuado conjunto de instrucciones agrupadas y ordenadas en lo que se denomina *programa*.

El procesador irá extrayendo las instrucciones de la memoria central con el fin de proceder a su ejecución. Por razones tecnológicas, la memoria sólo almacena dígitos binarios (bits: ceros o unos); por tanto, las únicas instrucciones que el ordenador es capaz de entender son combinaciones de unos y ceros: instrucciones elaboradas en *código de máquina*.

Las instrucciones en código máquina son difícilmente comprensibles a primera vista, aun cuando en lugar de representarlas en binario se escriban en código hexadecimal. Por ello, la elaboración de un programa se convierte en una tarea dura y, en muchos casos, repleta de errores. Por otra parte, se evidencia la dificultad adicional de que cada ordenador tiene su propio juego de instrucciones elementales.

Inconvenientes del lenguaje máquina

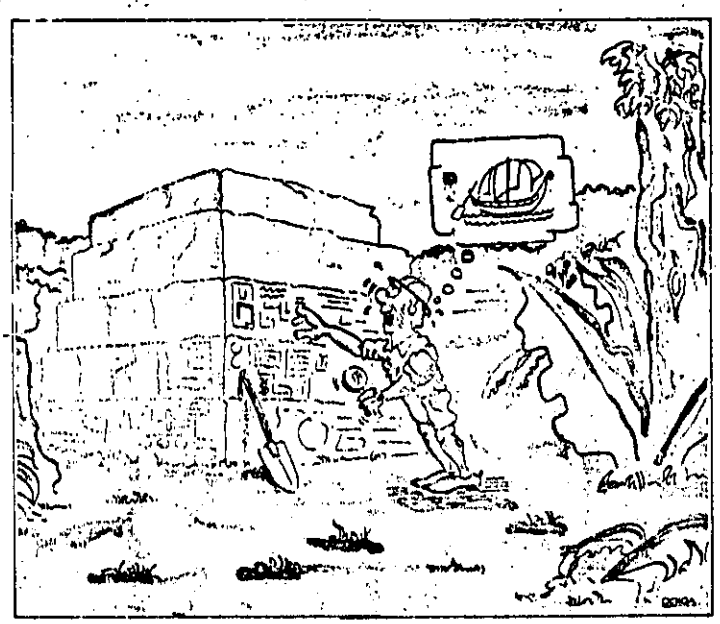
En teoría, dado que el ordenador debe operar con instrucciones que le sean comprensibles, es una condición necesaria que reciba una programación en lenguaje de máquina. No obstante, este tipo de programación presenta tres graves inconvenientes:

a) El programador debe conocer del orden de un centenar de instrucciones elementales, además de asignar a cada instrucción, a cada dato, a cada variable y a cada resultado una dirección real de memoria y recordar, durante la programación, la dirección real asignada. (De alguna forma deberá llevar un plano de la memoria.)

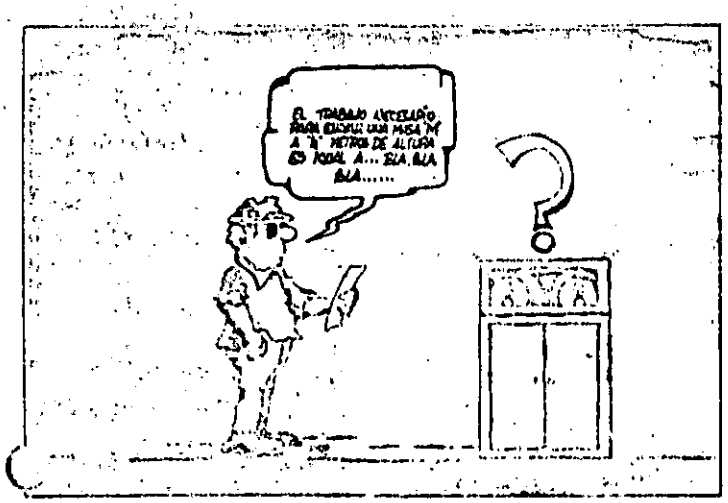
Como quiera que un programa pequeño puede alcanzar fácilmente el centenar de instrucciones, las dificultades aumentan a medida que crece el tamaño del programa.

b) Las instrucciones de nivel máquina sólo ejecutan las operaciones elementales de que es capaz el ordenador que se está utilizando. Por tanto, el programador debe conocer muy a fondo la estructura del ordenador que utiliza y descomponer el programa que tiene que resolver en operaciones elementales que formen parte del repertorio del ordenador.

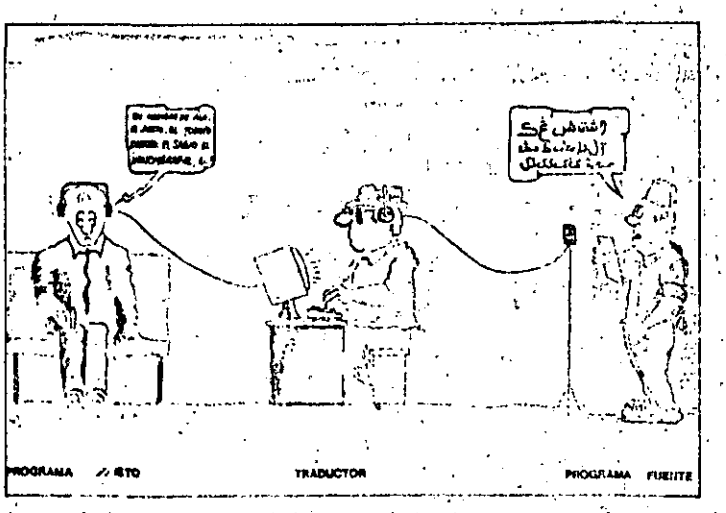
c) Quizá la dificultad más grave es que después del gran esfuerzo realizado para hacer un trabajo en estas condiciones, el resultado —el programa en código de máquina— sólo puede ejecutarse en un tipo de ordena-



Los programas escritos en lenguaje de máquina sólo pueden ser ejecutados por el ordenador que entiende ese lenguaje máquina: Al igual que una inscripción jeroglífica es indescifrable para todo aquel que no sea egiptólogo, un programa escrito en el código de una determinada máquina es ininteligible para las demás.



Los lenguajes de alto nivel se parecen más al idioma que hablan los hombres de negocios, técnicos y científicos que al de la propia máquina.



Los ensambladores y compiladores convierten los «programas fuente» en «programas objeto» descifrables por el ordenador. El proceso de traducción corre a cargo del propio ordenador, auxiliado por los oportunos programas traductores (ensamblador o compilador).

LOS LENGUAJES INFORMATICOS

dor, ya que distintos ordenadores habían en diverso lenguaje máquina.
Para eliminar estos inconvenientes se crearon lenguajes de programación cada vez más alejados del lenguaje de la máquina, pero más próximos al lenguaje humano.
Los diversos niveles de los lenguajes de programación, cada vez más evolucionados, permiten ir eliminando los inconvenientes citados.

Lenguajes próximos a la máquina

Este tipo de lenguaje elimina los problemas de tipo a) al utilizar códigos nemotécnicos en lugar de códigos binarios y direcciones simbólicas de memoria en lugar de direcciones absolutas. Los símbolos de los códigos de operación son fijos para cada lenguaje y las direcciones simbólicas las puede elegir el programador dentro de unas ciertas reglas.

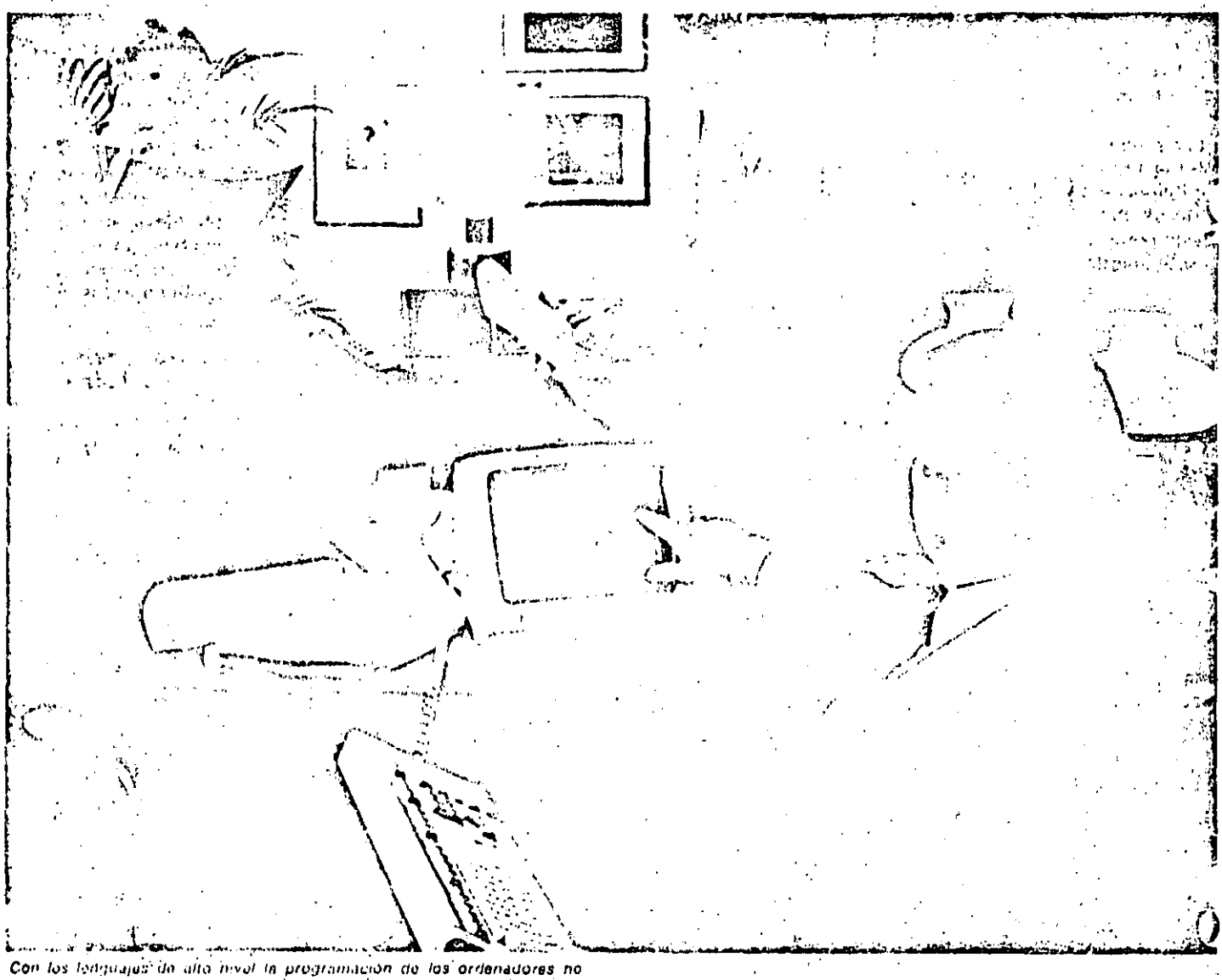
En este tipo de lenguaje, llamado *ensamble*, las instrucciones siguen siendo equivalentes a las instrucciones elementales de máquina, por lo que el programador necesita seguir conociendo a fondo su ordenador.

Un paso posterior incorpora las llamadas *macroinstrucciones*, en las que los códigos de operación ya no coinciden exactamente con los de máquina, es decir, que la descomposición del problema no tiene por qué llegar al nivel más elemental.

Se dice que los lenguajes de ensamble son próximos a la máquina porque siguen la estructura de sus instrucciones y cada tipo de ordenador tiene su propio lenguaje de ensamble. No resuelven el problema de la incompatibilidad entre las distintas máquinas.

Lenguajes próximos al problema

Este nivel de lenguaje resuelve, principalmente, el inconveniente c), ya que



Con los lenguajes de alto nivel la programación de los ordenadores no exige un profundo conocimiento de su estructura interna, con lo que cualquier usuario no especializado en la arquitectura íntima de los ordenadores puede llegar a confeccionar programas plenamente operativos.

al alejarse de la máquina y aproximarse al problema, no se encuentran ligados a ningún ordenador. Estos lenguajes, llamados de alto nivel, pueden ser utilizados en diferentes tipos de ordenadores con las excepciones que comentaremos en su momento.

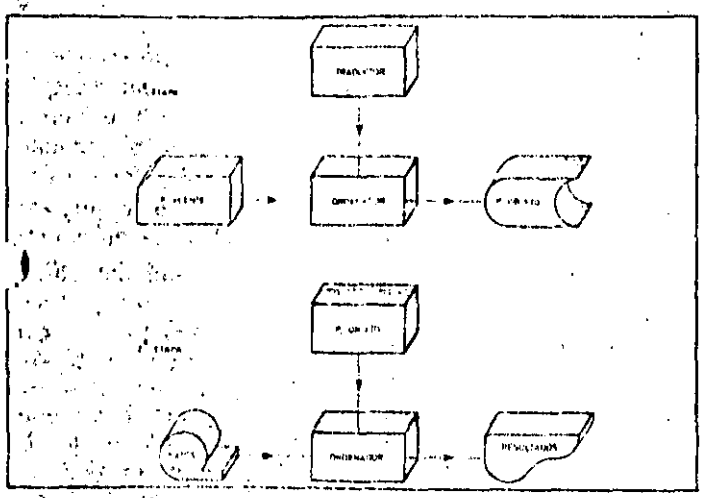
Evidentemente, las instrucciones de los lenguajes de alto nivel son muy distintas de las elementales de la máquina, por lo que, en general, una instrucción de alto nivel realiza el mismo proceso que muchas instrucciones elementales de nivel máquina. El inconveniente b) también es resuelto por los lenguajes de alto nivel, aunque siempre es necesario un mínimo conocimiento de las posibilidades del ordenador que estamos utilizando.

La traducción

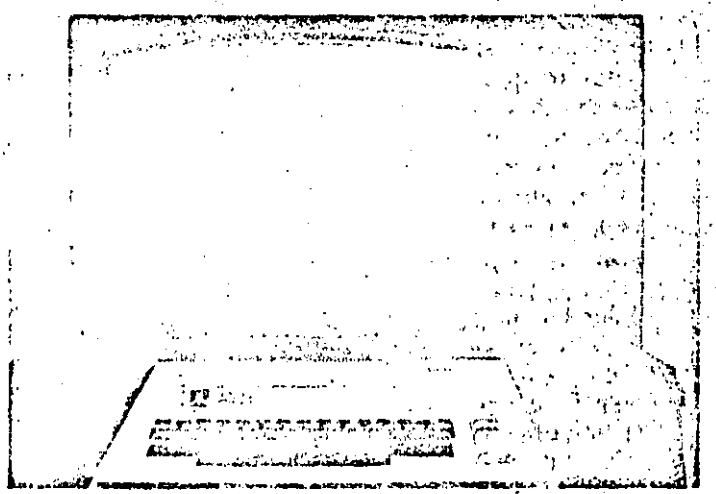
Si la unidad de control sólo procesa instrucciones escritas en su propio lenguaje, a base de unos y ceros..., ¿cómo es posible que pueda trabajar con un programa escrito en un lenguaje tan alejado a su estructura?
Ello es posible gracias al propio concepto de ordenador, ya que un proceso de ordenador implica el que un programa almacenado, utilizando unos datos de entrada, dé lugar a una información resultante en la salida.

¿Qué pasaría si los datos de entrada fueran nuestro programa, escrito en cualquier lenguaje, y el programa almacenado fuera una secuencia completa de instrucciones para su traducción? El resultado sería un programa escrito en lenguaje de máquina.

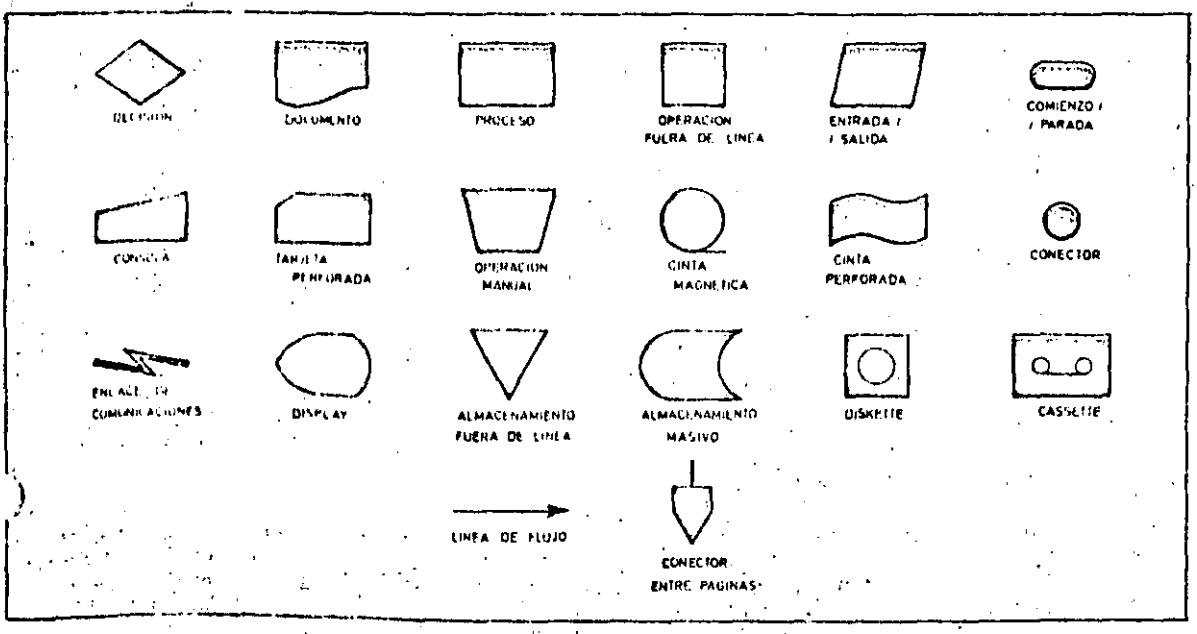
La solución es análoga a la que se aplica en una conferencia internacional. Si un conferenciante habla en una lengua que no entiende el auditorio, se resuelve el problema mediante un *traductor*. En informática se denomina *programa fuente* a un programa escrito en un lenguaje de ensamble o de alto nivel, y *programa objeto*, al escrito en código máquina. Por tanto, un *pro-*



Para ejecutar un programa escrito en lenguaje de alto nivel se requieren dos etapas. En el ejemplo, un «programa fuente» perforado en tarjetas, se convierte en «programa objeto» almacenado en disco. En la segunda etapa, el programa objeto es ejecutado por el ordenador.



El ordenador puede encargarse de traducir los programas escritos en lenguaje de alto nivel a programas en código máquina. Ello lo consigue ejecutando un programa «traductor» que utilice como datos a procesar las instrucciones del programa fuente.



Símbolos normalizados de los diagramas de flujo.

LOS LENGUAJES INFORMATICOS

grama objeto sólo puede ser ejecutado en el ordenador correspondiente.
Un programa fuente podría ser ejecutado en cualquier ordenador si previamente se procede a su traducción, recurriendo al programa ensamblador o compilador correspondiente a la máquina en que queremos trabajar.
Para procesar unos datos con un programa de alto nivel es necesario realizar estos dos pasos:
1. Una vez almacenado el *programa traductor*, cargar como datos el *programa fuente* para obtener como resultado el *programa objeto* en código de máquina.
2. Ejecutar el *programa objeto* (resultante del proceso de traducción anterior), con lo que al alimentar los *datos* del problema se obtendrán los *resultados* buscados.

Compatibilidad de programas

En teoría, todo programa escrito en un lenguaje de alto nivel podría ejecutarse en cualquier ordenador si se dispone del traductor adecuado. En la práctica no siempre es así, ya que tanto los fabricantes de máquinas como los diseñadores de compiladores introducen limitaciones y modificaciones. Debido a ello, para pasar un programa de un ordenador a otro es necesario realizar algunos cambios en el formato de determinadas instrucciones.

Símbolos de los diagramas de flujo

Los símbolos de los diagramas de flujo surgen para mostrar, de una manera gráfica y fácilmente reconocible, los pasos que se siguen en un proceso de ordenador. En realidad, cada usuario de ordenador podría tener sus propios símbolos para representar sus procesos en forma de diagrama de flujo. Esto supondría que sólo él, que conoce sus símbolos, estaría en condiciones de interpretarlos. Para resolver este problema y hacer comprensibles los diagramas a todas las personas, los símbolos se sometieron a una normalización. No vamos a mostrar aquí todos los símbolos de los diagramas de flujo, pero sí hablaremos de los más utilizados.

1. Funciones de proceso

Proceso: Cualquier función de proceso realizada por el ordenador. Por ejemplo, sumar dos cantidades.
Operación manual: Cualquier operación manual realizada «fuera de la línea», pero no por un equipo automático. Ejemplo de un símbolo de este tipo sería el de perforación de tarjetas.
Operación por equipo fuera de línea: Cualquier operación «fuera de línea» que no dependa de la velocidad humana, tal como el efectuado por una unidad de microfilm.

2. Funciones de entrada/salida y archivo

Tarjeta perforada: Los datos de E/S están en tarjetas perforadas.
Cinta magnética: Los datos de E/S se encuentran en cinta magnética.
Cassette: Los datos de E/S se encuentran grabados en una cinta de cassette.
Diskette: Los datos de E/S están en un diskette, también llamado «floppy disk» o disco flexible.
Cinta de papel perforado: Los datos de E/S se encuentran en cinta de papel perforado.
Documento: Normalmente es una salida; aunque puede representar una entrada en los casos de caracteres ópticos (OCR) y magnéticos (MICR).
Almacenamiento masivo: Generalmente disco magnético, aunque también puede indicar tambor magnético u otro medio de archivo.
Visualización (DISPLAY): En general, una pantalla CRT (tubo de rayos catódicos).
Entrada manual: Normalmente un teclado que permite la entrada de datos. También se usa como salida cuando el terminal es de teletipo.

3. Conexiones

Línea de flujo: Une dos símbolos.
Enlace de comunicaciones: Este símbolo indica el medio de transmisión entre elementos remotos de un equipo informático.
Conector entre páginas: Lo mismo que el anterior, pero los dos puntos de unión se encuentran en páginas diferentes.

4. Otros símbolos

Decisión: Para determinar cuál de los varios caminos posibles puede seguirse.
Comienzo o fin: Indica el comienzo o el final de un proceso.

Glosario

¿Qué es una macroinstrucción?

Una macroinstrucción es una instrucción del lenguaje de ensamble que se convierte, tras el proceso de traducción, en las instrucciones de lenguaje de máquina que sean precisas para realizar la tarea ordenada.
Por ejemplo *dividir* es una macroinstrucción que, dependiendo del ordenador que sea, hay que descomponer en otras elementales, tales como: cargar dividendo, restar divisor, contar número de restos posibles, etc.

¿Cuál es el lenguaje absoluto?

Lenguaje absoluto es una denominación otorgada a los lenguajes en código de máquina.

Ensamblador y ensamble, ¿son la misma cosa?

El *ensamblador* es el programa que convierte los programas escritos en *lenguaje de ensamble* (códigos nemotécnicos, direcciones simbólicas), en programas objeto escritos en lenguaje de máquina (series de unos y ceros que la máquina interpreta como sus códigos de operación y direcciones reales).
El ensamblador no es un lenguaje, sino que es un programa traductor. Ahora bien, es frecuente hablar de programar en «ensamblador», es decir, se ha extendido el uso de llamar ensamblador al lenguaje de ensamble.

¿Qué es un compilador?

Un *compilador* es un programa que traduce un programa en lenguaje de alto nivel a código de máquina. Hace el mismo papel que el ensamblador, pero a otro nivel.

¿Es necesario traducir un programa cada vez que se quiere ejecutar?

Si se va a ejecutar siempre en el mismo tipo de ordenador no es necesario. El programa objeto obtenido puede almacenarse en una memoria externa y ser llamado cada vez que sea preciso, ya que es un programa ejecutable.

LOS LENGUAJES MAQUINA

COMO ya sabemos, cualquier ordenador dispone de un conjunto o «repertorio» de instrucciones elementales en «código de máquina», que le indican lo que tiene que hacer.

Cada instrucción debe contener diversos elementos de información, con el fin de que el ordenador la pueda interpretar y, en consecuencia, ejecutar. Una de estas informaciones es lo que llamamos *código de operación*, que indica a la unidad de control cuál es la operación que debe efectuar. El resto de la instrucción (operando) debe indicar el/los dato/s o la dirección de la posición de memoria en que se encuentra el dato o los datos con los que se va a efectuar la operación. Por último, a veces es necesaria una información complementaria (de dispositivo de E/S, *status*, etc.) y que denominaremos *indicador*.

Formatos de las instrucciones

Las instrucciones de las diversas máquinas tienen un formato muy variado, siendo quizá la más compleja la del tipo:

CO	D10	D20	DR	DSI	I

en la que CO es el código de operación; D10, la dirección del primer operando; D20, la dirección del segundo operando; DR, la dirección en la que se debe almacenar el resultado; DSI, la dirección en la que se encuentra la siguiente operación, y, por último, I un indicador que especifica —cuando existe— algo más sobre la instrucción en concreto (periférico, registro especial, etc.).

Este caso sería el de una instrucción de cuatro direcciones, aunque las hay también de tres, siendo lo normal que sean de dos o de una, dependiendo de la arquitectura de la CPU.

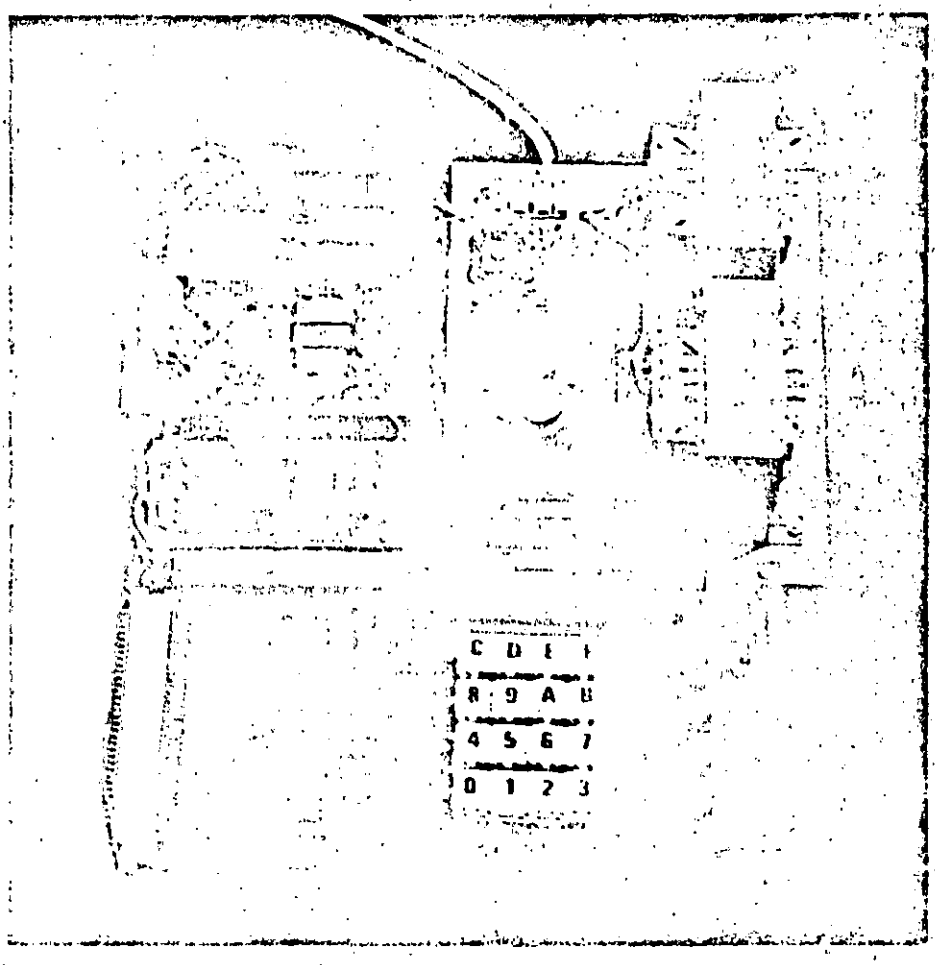
En los microordenadores, las instrucciones de nivel máquina suelen ser de uno, dos o tres bytes. El primer byte contiene el código de operación, mientras que los otros contienen el dato, dirección del dato o el indicador.

¿Cómo opera el procesador?

El procesador (microprocesador en el caso de los sistemas microordenadores) dispone de registros para la ejecución del programa: el contador de programa (en donde se almacena la dirección de la próxima instrucción que se debe ejecutar) y el registro de instrucción (en donde se decodifica la instrucción). El contador de programa indica en qué dirección de la memoria se encuentra almacenada la instrucción. Al leer ésta, el primer byte que contiene el código de operación es trasladado, a través del bus de datos, al registro de instrucción, donde es interpretado por el decodificador; acto seguido, el contador de programa se incrementa en una unidad. Si la instrucción contiene más bytes, el segundo pasa al registro de instrucción y se repite el proceso; y así sucesivamente. Una vez que se han decodificado o interpretado todos los bytes, se ejecuta la instrucción. El proceso se repite para la instrucción siguiente: se traslada el nuevo código de operación al registro de instrucción, desde la dirección indicada por el contador, repitiéndose el proceso hasta que se llega al final del programa.

¿Programar en binario?

Dado que todo el proceso se realiza con bits, es natural que, en principio, el programa deba estar en lenguaje binario. El programador debe escribir en binario tanto el código de operación como los datos y direcciones, tal como era preciso en los primeros ordenadores. El siguiente paso consiste en utilizar el sistema hexadecimal o decimal para escribir las instrucciones. Así, se simplifica la labor del programador. Los códigos de operación y las direcciones en decimal se utilizaron en algunos ordenadores de la segunda generación, aunque hoy día lo usual es



Los microordenadores menos evolucionados suelen programarse en el lenguaje máquina «numérico» propio del microprocesador que constituye su CPU. La representación numérica de la información suele realizarse en el sistema hexadecimal.

LOS LENGUAJES MAQUINA

Conceptos básicos

Introducción a la teoría de los lenguajes

Noam Chomsky inició el estudio de los lenguajes formales al crear un modelo matemático de una gramática en 1956. Hoy día el estudio de las gramáticas formales es uno de los campos más importantes de la informática teórica.

Webster define el lenguaje como «el conjunto de palabras y reglas para combinarlas, que es usado y entendido por una comunidad numerosa». Pero esta definición clásica no es rigurosa matemáticamente. Cuando los seres humanos usan un lenguaje, se puede permitir la existencia de palabras, términos o frases con significado simbólico o ambiguo, ya que la inteligencia y el buen juicio del oyente le permite comprender el sentido exacto que quiere darle el emisor, soslayando las discrepancias que puedan existir entre la forma y el fondo, esto es, entre la sintaxis y la semántica.

Para que una máquina entienda el significado concreto de un mensaje necesita que se le comunique en un lenguaje que esté rigurosamente definido con unas estrictas reglas gramaticales.

La comunicación hombre-máquina se debe realizar a través de un lenguaje escrito que impida la posibilidad de error en la interpretación de los mensajes.

Los conceptos fundamentales de la teoría de lenguajes son los de alfabeto, cadena, lenguaje y gramática.

Se llama *alfabeto* a un conjunto no vacío de símbolos gráficos.

Ejemplos típicos son:

El alfabeto castellano = {ⓐⓑⓒ ... ⓩ}

El alfabeto griego = {ⓐⓑⓒ ... ⓦ}

El alfabeto binario = {⓪①}

Hemos encerrado los elementos del alfabeto en círculos para distinguirlos, ya que ni la coma ni el espacio en blanco indicarían la separación, puesto que éstos pueden ser elementos del lenguaje (el espacio o la coma tienen ese carácter en muchos lenguajes de ordenador). No obstante, lo normal es representar los elementos espaciados o separados por comas. A los alfabetos se les suele llamar también *conjuntos de base* o *vocabularios* y se les representa por los símbolos Σ o V. Hay alfabetos que no son gráficos, como el alfabeto musical, formado por las notas (aunque luego se las represente gráficamente mediante símbolos en un pentagrama), el de los sordomudos o el de los ciegos.

utilizar la representación hexadecimal. Estos lenguajes de máquina reciben el calificativo de *numéricos* para distinguirlos de los lenguajes de máquina *simbólicos*.

Estos últimos sustituyen el código de operación numérico por un código alfabético que es nemotécnico, es decir, que le recuerda al operador lo que tiene que hacer la instrucción. Es más fácil recordar que ADD significa sumar (sobre todo, si se tienen conocimientos elementales de inglés), que mantener en la memoria que para sumar hay que emplear el código hexadecimal 09 (adición en el microprocesador Z80).

Siempre que se programe en el lenguaje propio de la máquina el programador debe llevar el control de las posiciones de memoria en las que almacena los diferentes datos, es decir, necesita utilizar el llamado mapa de direcciones de memoria.

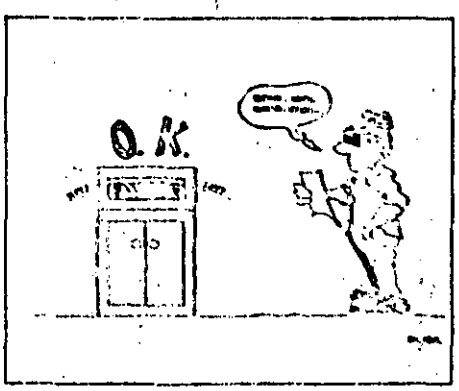
Tipos de instrucciones

Las instrucciones se suelen agrupar en tipos que determinan la naturaleza de la tarea que se ordena a la CPU. Los principales tipos son:

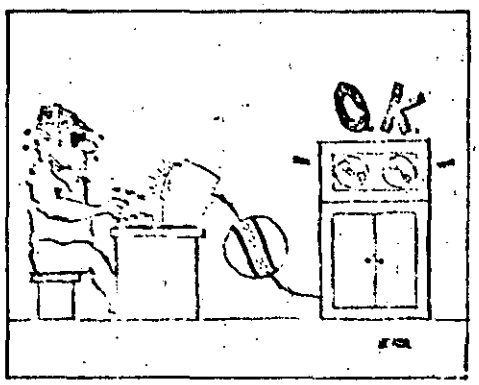
- *Instrucciones de transferencia de datos.* Permiten la lectura o escritura desde o hacia la memoria, y entre registros internos del procesador. Incluyen también la carga y descarga de registros (de memoria a acumulador o viceversa, etc.).
- *Instrucciones de ruptura de secuencia.* Son las instrucciones que realizan los saltos y las bifurcaciones de una parte a otra del programa. Pertenecen a este grupo tanto los saltos incondicionales como los condicionales, las instrucciones de control de bucles, las llamadas a subrutinas, las de retorno al programa principal, etc.

Instrucción de un byte	B7 B6 B5 B4 B3 B2 B1 B0	
Instrucción de dos bytes	B7 B6 B5 B4 B3 B2 B1 B0	D7 D6 D5 D4 D3 D2 D1 D0
Instrucción de tres bytes	B7 B6 B5 B4 B3 B2 B1 B0	D7 D6 D5 D4 D3 D2 D1 D0 D7 D6 D5 D4 D3 D2 D1 D0
	Código operación	Dato o dirección

En general, existen tres formatos de instrucciones a nivel máquina: de uno, dos o tres bytes. El primer byte corresponde al código de operación y los restantes contienen el operando (dato o dirección del dato).



Al programar en lenguaje máquina «numérico» en su nivel más elemental (sistema binario), el programador debe escribir las instrucciones en lenguaje binario: ceros y unos.



Con un leve perfeccionamiento, consistente en la inclusión de un codificador, la tarea de programación en lenguaje máquina puede facilitarse al permitir la escritura de las instrucciones en hexadecimal.

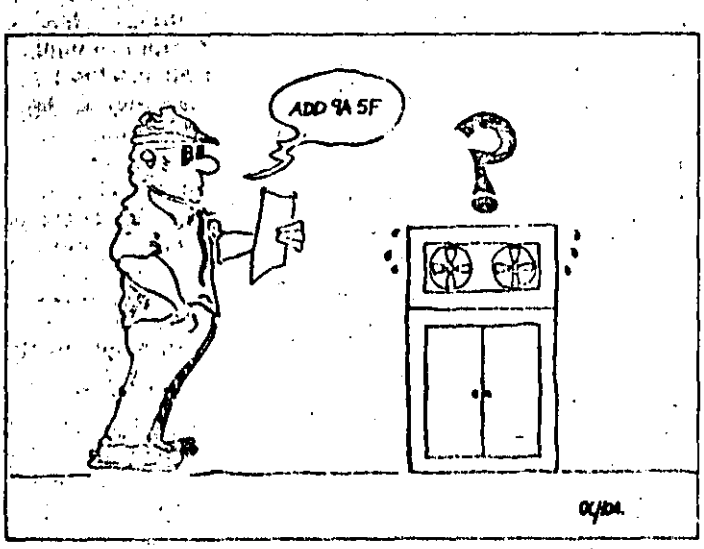
• *Instrucciones de entrada/salida.* Las que relacionan a los periféricos con la memoria.
• *Instrucciones de control.* Las que permiten controlar el programa y el equipo. Comprenden el control del *status* (registro de estado), la no operación, la parada, etc.

Los conjuntos de instrucciones máquina, así como sus códigos nemotécnicos difieren de un equipo a otro, por lo que es necesario que el programador disponga de la tabla correspondiente.

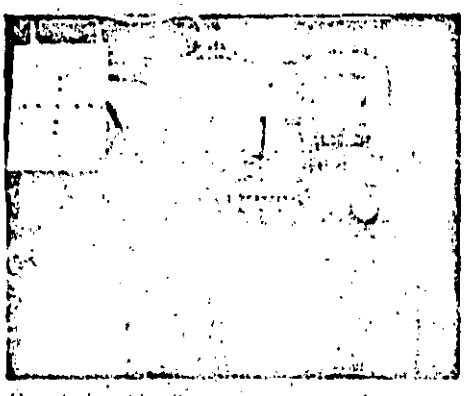
Direccionamientos

Los bytes que siguen al código de operación corresponden al operando: un dato, una dirección o un indicador. El código de operación indica a la unidad de control el tipo de direccionamiento implicado y, por consiguiente, dónde localizar el dato correspondiente. Los tipos de direccionamientos más usuales son:

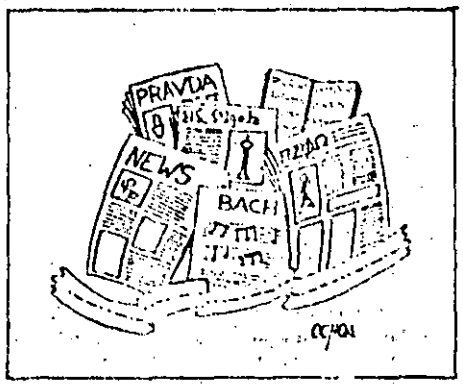
• *Direccionamiento implícito:* Opera entre registros internos. Son instrucciones de un sólo byte. Ejemplo: RTS (60) del microprocesador 6502 que corresponde retorno de subrutina.
• *Direccionamiento inmediato:* Opera directamente con el dato contenido en los bytes de operando. Ejemplo: ADI (C5) del 8080. Suma al acumulador el segundo byte de la instrucción.
• *Direccionamiento directo:* Accede a la posición indicada por los bytes de operando. En esa posición de memoria se encuentra el dato a operar. Ejemplo: SUB del 8085. Resta del acumulador el contenido del registro cuya dirección está en los bytes de dirección.
• *Direccionamiento relativo:* Permite acceder a la posición de memoria, cuya



En los lenguajes de máquina simbólicos, el programador utiliza códigos nemotécnicos en lugar de códigos numéricos. En cualquier caso, sigue siendo necesario llevar un mapa de direcciones reales en las que se van almacenando los datos.



Una de las dificultades inherentes a la programación en lenguaje máquina es que los programas sólo son ejecutables en equipos análogos, basados en la misma unidad central de proceso.



Los innumerables alfabetos que existen sirven para representar de forma simbólica los lenguajes que permiten la comunicación.

Glosario

¿Es necesario saber inglés para programar ordenadores?

No, aunque si es conveniente, ya que todos los códigos nemotécnicos de los lenguajes de programación son recordatorios del verbo inglés que indica la operación a ejecutar. Así, ADD es sumar o SUB es la abreviatura de Substract (restar). Además, los manuales de los fabricantes y mucha de la literatura informática se encuentra en inglés.

¿Por qué se suelen utilizar dos bytes para expresar direcciones de memoria en los microordenadores?

El motivo es que con 16 bits se puede representar en binario hasta el número $2^{16} = 65.536 = 64$ K que coincide con el número máximo de posiciones de memoria directamente direccionables por un microprocesador de 8 bits (con bus de direcciones de 16 bits). Estos microprocesadores constituyen la CPU de la mayor parte de los microordenadores actuales.

¿Cómo se organiza el mapa de direcciones?

El mapa se suele dividir en páginas. Una página es un bloque de direcciones de memoria, usualmente de 4 Kbytes, aunque puede ser de cualquier otra potencia de 2. Así, una memoria de 64 K tendría 16 páginas de 4 Kbytes numeradas en hexadecimal de 0 a F. Cada página tendría las direcciones comprendidas entre el 000 y el FFF. De esta forma la página 0 comprendería las direcciones absolutas desde la 0000 a la 0FFF. La página 1 desde 1000 a 1FFF, y así sucesivamente. La última página, la F, abarcaría desde la dirección F000 a la FFFF.

LOS LENGUAJES MAQUINA

dirección es la indicada por el contador del programa más o menos un número indicado por el segundo byte. Ejemplo: BCC (90) del 6502. Salta a la dirección del contador más el valor del segundo byte, atendiendo a un determinado indicador del registro de estado.

• *Direccionamiento indexado:* La dirección del dato se obtiene sumando el valor del registro indicado al valor de la dirección absoluta incluida en la instrucción. Ejemplo: DEC (D6) del 6502, que resta una unidad al valor que está en la dirección dada por el contenido del registro índice X más el valor del segundo byte de la instrucción.

Al igual que ocurre con los repertorios de instrucciones máquina, los tipos de direccionamiento de las instrucciones difieren de un equipo a otro y son función del diseño de la CPU.

¿Hay ventajas en programar en lenguaje de máquina?

El conocimiento exhaustivo de las instrucciones (códigos y forma de direccionamiento), así como de las direcciones reales de memoria en donde se almacenan los datos, es una de las principales dificultades para programar en lenguajes de máquina. Además, los programas en código de máquina no son ejecutables en otro equipo que no tenga un procesador igual o compatible.

Entonces, ¿es útil conocer el lenguaje máquina de nuestro equipo, aunque éste admita la programación en un lenguaje de alto nivel? La respuesta es sí, por las siguientes razones:

— Si tenemos poca capacidad de memoria podremos recurrir al lenguaje de máquina que es el que menos memoria ocupa.

— Una rutina que se haya de utilizar en múltiples ocasiones, programada en código de máquina no sólo ahorra memoria, sino también tiempo de ejecución.

— Las mejoras o modificaciones en el sistema operativo son más efectivas si se hacen en lenguaje de máquina.

Por último, recordemos que los programas en código de máquina son los que se ejecutan en menos tiempo.

```
                        ;REPEAT ROUTINE
                        ;
                        ;       ORG   $0800
                        ;
                        ;TEMPORARY DATABUFFERS IN PAGE ZERO
                        ;
0800:                   KEY   *     $00DA
0800:                   NOTEL *     $00DC
0800:                   NOTEH *     $00DD
0800:                   LENGTH *    $00DE
                        ;
                        ;INTERVAL TIMER
                        ;
0800:                   CNTA  *     $1AF4  DISABLE TIMER IRQ
0800:                   CNTD  *     $1AF7  DISABLE TIMER IRQ, CLK1KT
0800:                   CNTG  *     $1AFE  ENABLE TIMER IRQ, CLK64T
0800:                   RDFLAG *    $1AD5  B7 IS TIMER FLAG
                        ;
                        ;GOTO MONITOR
                        ;
0800:                   RESET *     $1C1D  NEW I/O DEFINITION
                        ;
                        ;I/O DEFINITION
                        ;
0800:                   PBD   *     $1AB2
0800:                   PBDD  *     $1AB3
                        ;
                        ;IRQ VECTOR
                        ;
0800:                   IRQL  *     $1A7E
0800:                   IRQH  *     $1A7F
                        ;
                        ;
                        ;START OF THE REPEAT PROGRAM
                        ;
0800. 0800 78           REPEAT SEI          DISABLE IRQ LINE
0801: 0801 D8                  CLD
0802: 0802 A9 30               LDAIM IRQRE  SET UP IRQ VECTOR
0804: 0804 8D 7E 1A            STA   IRQL
0807: 0807 A9 1A               LDAIM IRQHE  /256
0809: 0809 8D 7F 1A            STA   IRQH
080C: 080C A9 01               LDAIM $01    PB0 IS OUTPUT
080E: 080E 8D B3 1A            STA   PBDD
0811: 0811 8D B2 1A            STA   PBD    TOGGLE SPEAKER OFF
0814: 0814 85 DD               STAZ  NOTEH  SET NOTE POINTER
0816: 0816 A9 00               LDAIM $00
0818: 0818 85 DC               STAZ  NOTEL  SET NOTE POINTER
081A: 081A 8D F4 1A            STA   CNTA   RESET IRQ LINE, DISABLE TIMER IRQ
081D: 081D 58                  CLI          ENABLE CPU IRQ
                        ;
081E: 081E A9 FF        FETCH  LDAIM $FF    SET TIMER ENABLE TIMER IRQ
0820: 0820 8D FE 1A            STA   CNTG
0823: 0823 A0 00               LDYIM $00    FETCH NOTE
0825: 0825 B1 DC               LDAIY NOTEL
0827: 0827 85 DA               STAZ  KEY
0829: 0829 C8                  INY          FETCH LENGTH
082A: 082A B1 DC               LDAIY NOTEL
082C: 082C 85 DE               STAZ  LENGTH
082E: 082E A4 DA               LDYZ  KEY    LOOKUP CONVERSION
                        ;
0830: 0830 A9 00        TONE   LDAIM $00    TOGGLE SPEAKER ON
0832: 0832 8D B2 1A            STA   PBD
0835: 0835 BE 00 1A            LDXY  DEL    GET FREQUENCY
0838: 0838 20 78 00     TONEA  JSR   EQUALA DELAY 22 MICRO SEC
083B: 083B CA                  DEX
083C: 083C D0 FA               BNE   TONEA  LOOP TIME IS 27 MIKRO SEC*X
083E: 083E A9 01               LDAIM $01    TOGGLE SPEAKER OFF
0840: 0840 8D B2 1A            STA   PBD
0843: 0843 BE 00 1A            LDXY  DEL    GET FREQUENCY AGAIN
0846: 0846 A4 DE        TONEB  LDAZ  LENGTH GET LENGTH
0848: 0848 30 08               BMI   TONEC  TIME OUT?
084A: 084A 20 74 00            JSR   EQUALB EQUALIZE 17 MICRO SEC
084D: 084D CA                  DEX
084E: 084E D0 F6               BNE   TONEB  LOOP TIME IS 27 MICRO SEC*X AGAIN
0850: 0850 F0 D6               BEQ   TONE   RETURN AFTER ONE PERIODE
0852: 0852 A2 04        TONEC  LDXIM $04    LOOP TIME = 4*CNTD*PRESET
0854: 0854 A9 30        TONED  LDAIM $30    PRESET = $30
0856: 0856 8D F7 1A            STA   CNTD   DISABLE TIMER IRQ
                        ;
0859: 0859 2C D5 1A     FULL   BIT   RDFLAG READ FLAG REGISTER, TIME OUT?
085C: 085C 10 F8               BPL   FULL   IS TIMER FLAG STILL ZERO?
085E: 085E CA                  DEX
085F: 085F D0 F3               BNE   TONED  LOOP COUNTER ZERO?
0861: 0861 E6 DC               INCZ  NOTEL  ADJUST NOTE POINTER
0863: 0863 E6 DC               INCZ  NOTEL
0865: 0865 A0 00               LDYIM $00
0867: 0867 B1 DC               LDAIY NOTEL  END OF NOTE BUFFER?
0869: 0869 C9 FF               CMPIM $FF    FOR CHARACTER
086B: 086B D0 D1               BNE   FETCH  IF NOT EOF, CONTINUE
086D: 086D 4C 1D 1C            JMP   RESET  ELSE BACK TO MONITOR
                        ;
```

Listado de un programa confeccionado en lenguaje máquina para el microprocesador 6502. En el mismo aparecen las instrucciones en lenguaje máquina numérico —hexadecimal— (zona izquierda) y en lenguaje simbólico (zona derecha).

SOFTWARE

EL LENGUAJE «ENSAMBLADOR»

PARA evitar el tener que utilizar códigos numéricos y direcciones reales de memoria al programar, se desarrollaron los lenguajes de ensamble, también conocidos como lenguajes simbólicos, lenguajes ensambladores o «assemblers».

Estos lenguajes permiten escribir los programas representando los diferentes elementos de forma simbólica. Estos símbolos son de dos tipos: fijos para los códigos de operación y variables para las direcciones.

Como cada máquina sólo tiene un juego de instrucciones ejecutables por su unidad de control, los lenguajes ensambladores son específicos para cada máquina.

Lenguajes simbólicos simples

Los lenguajes tienen una sintaxis, cuyas leyes dependen de la estructura de la máquina y de las restricciones impuestas a la escritura de las instrucciones.

Los programas fuente aparecen como un conjunto de líneas escritas, denominadas *sentencias*, y que generalmente contienen un código de operación simbólico.

La estructura de una sentencia suele ser:

Etiqueta-Código-Operandos-Comentario

Algunos ensambladores tienen el formato fijo, distribuyéndose las zonas en posiciones prefijadas de la línea de escritura. Otros son de formato libre, bastando con respetar el orden de los elementos y establecer una separación entre ellos.

La *etiqueta* es el identificador de la dirección de la instrucción. Puede ser elegida libremente por el programador. En general es alfabética, aunque a veces puede incluir cifras siempre que el primer caracter sea una letra. No es necesario etiquetar todas las instrucciones, sólo se etiquetarán aquellas a las que haya que referenciar en otro punto del programa.

El campo de código de operación contiene el nombre nemotécnico asignado a la instrucción. Estos códigos son elementos fijos del lenguaje y el programador no puede ni alterarlos ni usarlos como etiquetas ni operandos.

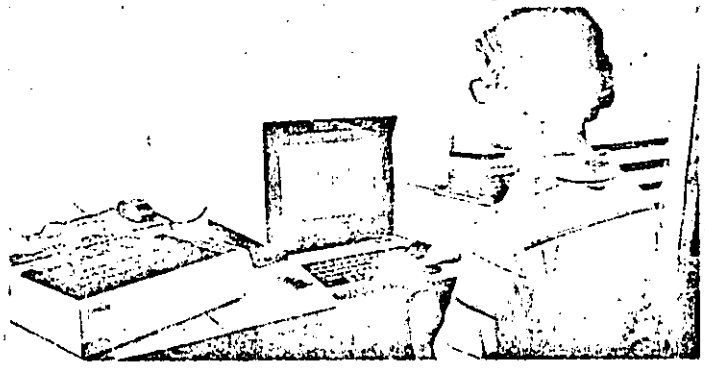
En el campo de operandos se codifican los identificadores de dirección de los operandos. Hay instrucciones que no tienen operando, aunque, por lo general, éste se compone de uno o más valores separados por comas. También pueden darse valores constantes o expresiones aritméticas.

El campo de comentario no es obligatorio y es ignorado durante la fase de traducción. Sirve para facilitar la lectura del programa fuente a cualquier persona.

Las instrucciones corresponden a los tipos ya estudiados en otros temas, aunque los lenguajes de ensamble añaden un nuevo tipo: las pseudoinstrucciones.

Las pseudoinstrucciones

Su nombre viene de que se utilizan para definir datos, codificándose en la misma forma que las instrucciones de máquina, aunque no lo sean. En general, son sentencias que no producen directamente ningún código objeto, sino que ayudan en la definición del resto del programa, complementando la codificación. Suelen conservar los mismos símbolos en los diferentes lenguajes, siendo los principales tipos los siguientes:



Los lenguajes ensambladores ocupan el nivel inmediatamente superior al de los lenguajes máquina, permiten escribir los programas representando los diversos elementos de forma simbólica.

```
02160          RNTY  XI-9                    00702 36 01184 00100
02170          TFLS  ZZ.DT
                                              00714 16 01425 00737
                                              00726 49 01372 00000
                                              00733 00005    01227
                                              00738 00005    01238
02180 RST1     BNR   OPER.DT + 1,7
                                              00744 45 00792 01239
02190          TBTY
                                              00756 34 00000 00108
02200          WNTY  ZZ-9
                                              00768 38 01218 00100
03010          B     REST
                                              00780 49 00888 00000
030200 PER     BTFS  POR.X1
                                              00792 16 01423 00815
                                              00804 49 01392 00000
                                              00811 00005    00960
                                              00816 00005    01193
03030          AM    RST1 + 11,10
                                              00822 11 00755 00010
03040 S''M     BTFS  MAS.DT + 10
                                              00834 16 01423 00857
                                              00846 49 01392 00000
                                              00853 00005    01014
                                              00858 00005    01248
03050          AM    S''M + 28,10
                                              00864 11 00862 00010
03060          B     RST1
                                              00876 49 00744 00000
```

Listado obtenido al final de un proceso de «ensamblado». En el mismo se observan las diversas instrucciones en código máquina generadas por cada macroinstrucción.

EL LENGUAJE «ENSAMBLADOR»

Conceptos básicos

Ensambladores

Los ensambladores son los encargados de convertir los programas fuente, escritos en lenguaje de ensamble, a programas objeto en código de máquina.
Como en todo proceso de traducción, junto con el programa objeto aparecen los listados de errores sintácticos y de correspondencia entre el programa fuente y el objeto.
El trabajo del ensamblador se reduce a una traducción palabra a palabra, cambiando por códigos de operación numéricos y direcciones reales los símbolos del programa. Para ello emplea tablas de traducción de símbolos, localizadas en la memoria, así como la cuenta de la memoria ocupada. Como a veces la definición de los símbolos puede venir detrás de la sentencia en la que aparecen por primera vez, la traducción se realiza repitiendo el proceso dos veces. Cada una de estas veces se denomina *paso*, por lo que este tipo de ensamblador se denomina *de dos pasos*. En el primer paso construye la tabla de símbolos y realiza el análisis morfológico de las sentencias, detectando y escribiendo los errores sintácticos; en el segundo se genera el programa objeto, sustituyendo los símbolos por sus direcciones reales.
También existen *ensambladores de un paso* que realizan el proceso de una sola vez. Estos ensambladores, cuando encuentran una sentencia que contiene un símbolo que todavía no está definido, la retienen en memoria, y según va encontrando definiciones vuelve hacia atrás y completa la traducción. De todas formas, su empleo práctico es bastante limitado.
La traducción de los programas escritos en lenguaje autocodificador o macroensamblador es realizado por los *macroensambladores*, que contienen un ensamblador para traducir el resultado de la expansión de las macros.
Si se codifican todas las macros antes de cualquier llamada, se puede realizar la expansión junto con el primer paso de la traducción, obteniéndose un *macroensamblador de dos pasos*. En caso contrario, es necesario realizar la expansión en una fase anterior al ensamblaje, llamada *preensamblaje*, y dando lugar a los *macroensambladores de tres pasos*.
El último avance en el campo de los ensambladores viene dado por los llamados *macroprocesadores de uso general*, o *mataensambladores*.

De principio y fin de programa (sirven para el control de la traducción).
De definición de constantes (para poder introducir constantes y referirse a ellas por medio de un identificador).
De reserva de zona de memoria (útiles, por ejemplo, para cargar tablas, matrices, etc.).

Ventajas y desventajas de los lenguajes de ensamble simples

Entre las principales ventajas se encuentran la reducción de los errores lógicos (puesto que no se emplean direcciones reales), la fácil eliminación de los errores formales (ya que son detectados en la traducción) y la disminución de los tiempos de programación.
Tienen el inconveniente de que cada ordenador tiene un lenguaje ligado a su estructura y juego de instrucciones, por lo que el programador tendría que conocer diversos lenguajes ensambladores, si quiere trabajar en diversas máquinas.

Lenguajes autocodificadores

La limitación de usar sólo el conjunto

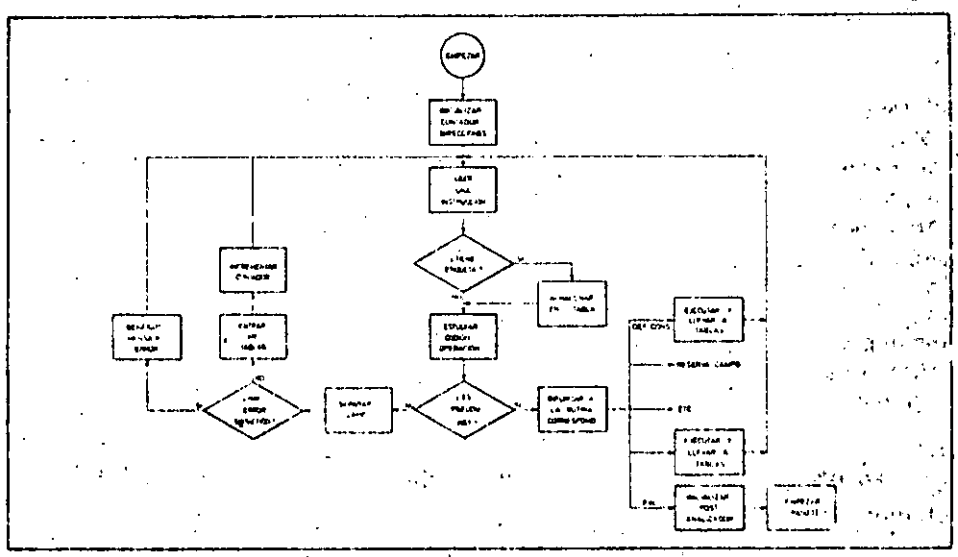


Diagrama de flujo detallado del primer paso de una operación de traducción.

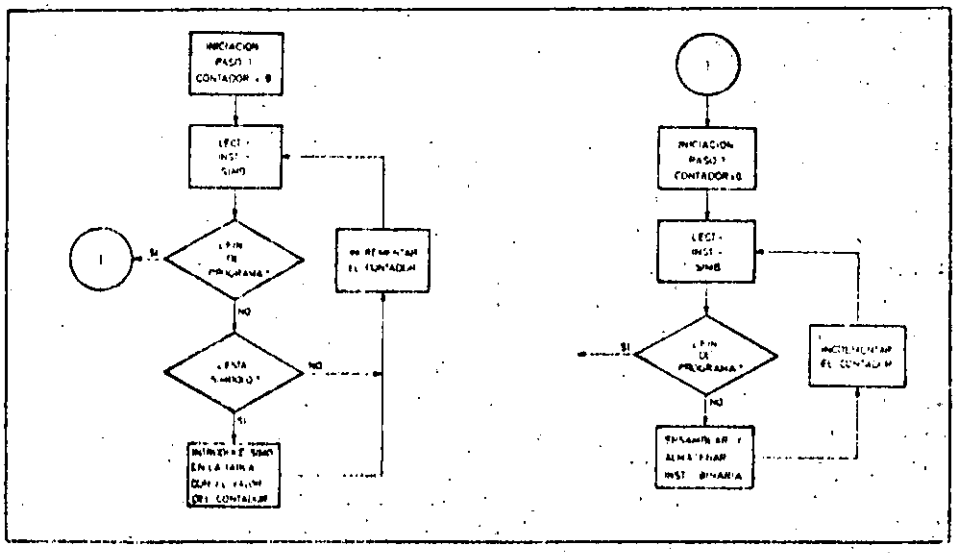


Diagrama de la secuencia de operaciones que realiza un programa ensamblador de dos pasos.

de instrucciones de que dispone una máquina es eliminada por un nuevo nivel de lenguajes: los llamados lenguajes de nivel *autocodificador, macroensamblador*, o *macroprocesador*, que constituyen el primer paso hacia la independencia entre el lenguaje y la máquina, gracias a la introducción de las *macroinstrucciones*.

Una *macroinstrucción* es una instrucción que no se corresponde directamente con una instrucción del lenguaje de máquina, sino que representa operaciones que pueden desglosarse en secuencias más o menos largas de instrucciones máquina. Las macroinstrucciones se componen en general de dos campos: el *campo de operación* y el *código paramétrico*, cumpliendo el primero el mismo papel que el código de operación, mientras que el campo paramétrico contiene los datos con los que se realiza la macroinstrucción. Estos datos pueden ser numéricos o simbólicos. Las macroinstrucciones permiten programar, en una sola instrucción, operaciones tales como la comparación, división, etc.

Existen dos tipos de macroinstrucciones: *del lenguaje* y *del programador*. Las macroinstrucciones del lenguaje son aquellas que son proporcionadas por el constructor, y, al igual que las microinstrucciones, tienen un código prefijado.

Macros del programador

Algunos lenguajes permiten que el programador cree sus propias «macros» mediante el uso de un *lenguaje de definición* de macros. Son muy útiles para introducir subprogramas.

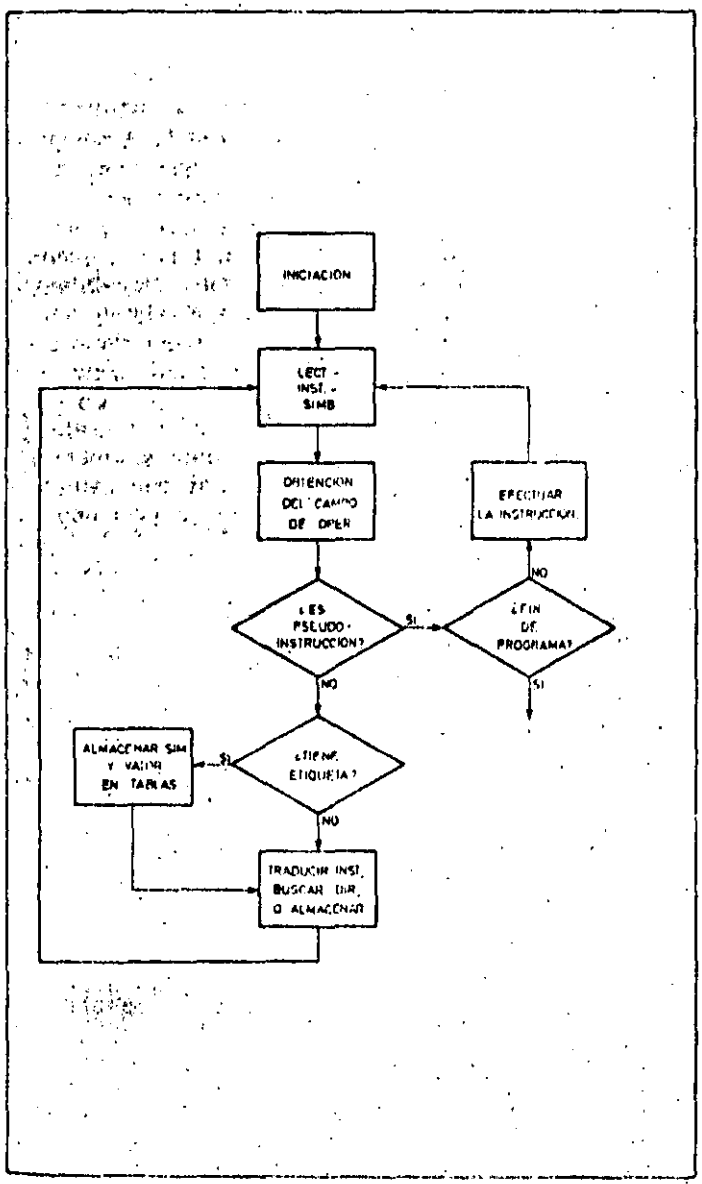
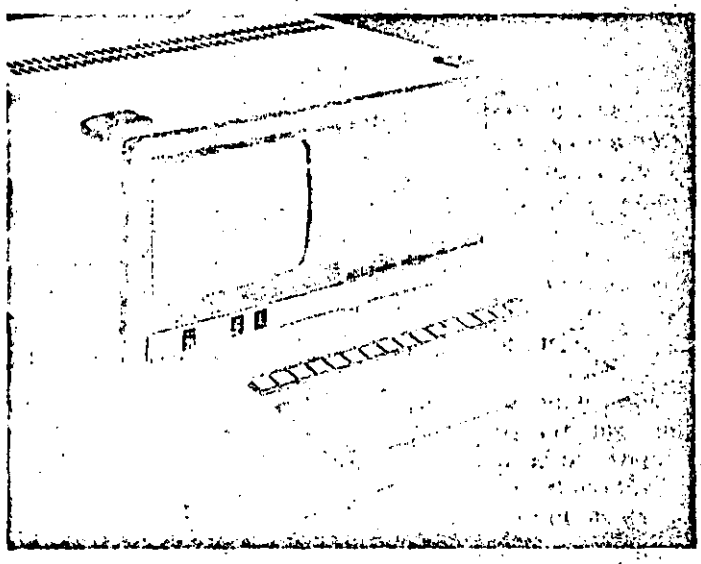
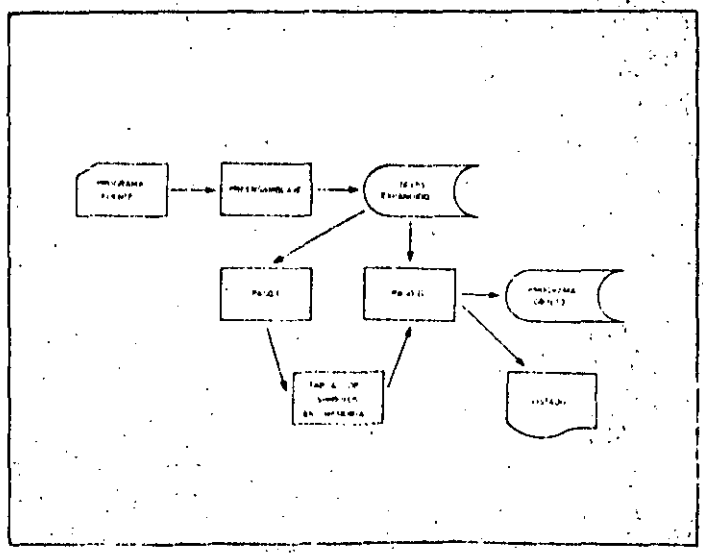


Diagrama de flujo asociado a los procesos de ensamblado «en un paso».



Aun a pesar de que son lenguajes más evolucionados que los de nivel máquina, los lenguajes de ensamble o ensambladores son específicos para cada máquina.



Representación de un proceso de macroensamblaje en tres pasos.

EL LENGUAJE «ENSAMBLADOR»

Para que en la traducción se pueda conocer qué es una macroinstrucción de este tipo, hacen falta dos pseudoinstrucciones: una de principio y otra de fin. Entre ambas se encuentra el cuerpo de la macroinstrucción, constituido por el conjunto de instrucciones que la desarrollan. En primer lugar, se encuentra el *prototipo* de la macro, en el que aparecen el código elegido para la macro y los nombres simbólicos de los parámetros o argumentos.

Los lenguajes de nivel superior permiten la existencia de macroinstrucciones dentro de otras macroinstrucciones o incluso la recursividad de la macroinstrucción, esto es, que una macroinstrucción se llame a sí misma, lo que puede ser útil, por ejemplo, para calcular el factorial de un número.

La principal ventaja de estos lenguajes es no tener que descender al nivel de instrucción elemental de máquina a la hora de programar. Pero de todas formas, al usar también microinstrucciones, siguen siendo lenguajes próximos a la máquina, por lo que los programas escritos en estos lenguajes no se pueden procesar en todos los ordenadores.

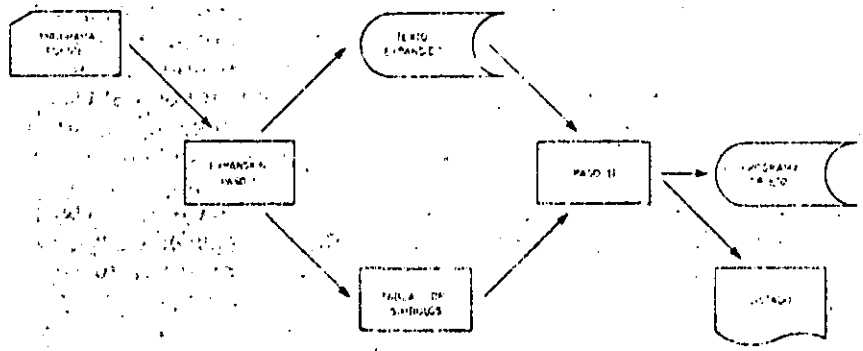


Diagrama correspondiente al desarrollo de un proceso de macroensamblaje en dos pasos.

Etiqueta	Código	Operando	Comentarios
	LD	DT, 0	Poner la variable DT a 0
	LD	DI, 100	Poner la variable DI a 100
SUMT	ADD	DT, DI	Sumar DI a total DT
	SUB	DI, 1	Restar 1 a DI
	JR	NZ, SUMT	Continuar sumando si DI no es cero

El listado de la figura muestra un programa para la suma de los números decimales del cero al cien, confeccionado en lenguaje de ensamble o ensamblador.

Glosario

¿Por qué se llama ensamblador?

Un programador, cuando tiene que codificar un programa muy largo, lo divide en varios subprogramas o rutinas independientes, que escribe, traduce y prueba por separado. Por consiguiente, el traductor debe seguir la pista de todas las referencias cruzadas, es decir, debe estar en condiciones de *ensamblar* todas las partes para dar un resultado único.

¿Es difícil aprender varios lenguajes de ensamble?

La mayoría de los lenguajes de esta clase son muy parecidos entre sí, por lo que cuando se conoce uno de ellos se pueden aprender otros sin una especial dificultad.

¿Qué se entiende por «expansión» de una «macro»?

Se llama expansión de una macroinstrucción al proceso que traduce una macro en la secuencia de instrucciones equivalentes.

¿Qué son los parámetros o argumentos de una macro?

Se llaman argumentos o parámetros de una macro a los operandos de la misma. Reciben este nombre debido a la similitud entre macro y subprograma.

¿Qué es un metaensamblador?

Fue una idea de Fergunson basada en que los diferentes ensambladores tienen muchos puntos comunes. Un metaensamblador podría admitir la descripción de las reglas de ensamblaje para un ordenador particular, dando el resultado como si el trabajo hubiera sido realizado por el ensamblador normal. No están muy desarrollados en la actualidad.

EL LENGUAJE FORTRAN

DESDE la aparición del ordenador electrónico, las universidades y centros de investigación técnica y científica comprendieron la gran ayuda que éste les iba a proporcionar. Pero existía la barrera de los lenguajes de programación. Los lenguajes de máquina y de ensamble estaban bastante lejos de la forma de expresión técnica (módelos matemáticos, expresiones aritméticas, ecuaciones, etc.). De ahí surgió la idea de un lenguaje para la resolución de problemas científicos mediante técnicas de cálculos numéricos. El primero de estos lenguajes fue el SHORT CODE creado en 1949 por el Dr. Mandy para UNIVAC. Unos años más tarde, en 1953, aparece el SPEED-CODING de Backus para IBM.

Treinta años de FORTRAN

El mismo Backus inició el desarrollo del lenguaje FORTRAN en noviembre de 1954, publicando IBM el primer manual en 1956.

El FORTRAN II aparece en junio de 1958, aportando importantes ampliaciones fundamentalmente en el campo de las subrutinas, y el FORTRAN IV lo hace en 1962. A partir de entonces fueron varias las firmas que contribuyeron a su expansión, de tal forma que existen en la actualidad más de cincuenta compiladores.

Para equipos pequeños existen diversas versiones, siendo de destacar el FORTRAN-80 de Microsoft (firma especializada en software para ordenadores personales y micros). Esta contiene 28 tipos de instrucciones diferentes, entre ellas todas las normalizadas por ANSI.

Las principales ventajas del FORTRAN son su capacidad para manejar expresiones matemáticas y su velocidad de cálculo, por lo que es apropiado para las aplicaciones científicas en las que lo importante es el volumen de cálculo y no la entrada/salida.

Trabaja sólo con compilador, realizando la compilación en un solo paso.

Debido a su antigüedad, y a pesar del gran número de revisiones que ha sufrido, hoy día es bastante inferior a otros lenguajes tales como el Pascal o versiones avanzadas del BASIC. De todas formas, los programas escritos en FORTRAN constituyen todavía un elevado porcentaje de las bibliotecas de programas técnicos y científicos.

Las hojas de programación

El FORTRAN utiliza los caracteres alfabéticos de la A a la Z, los dígitos del 0 al 9 y los caracteres especiales siguientes:

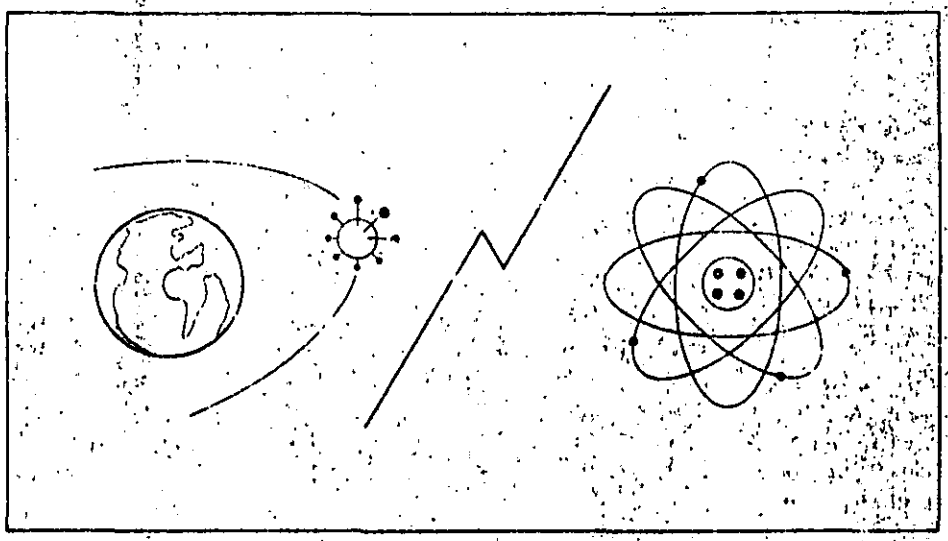
(espacio) = + − * / ( ) , . $ '

Con ellos se escriben los programas en una hoja de diseño especial que admite 80 caracteres por línea.

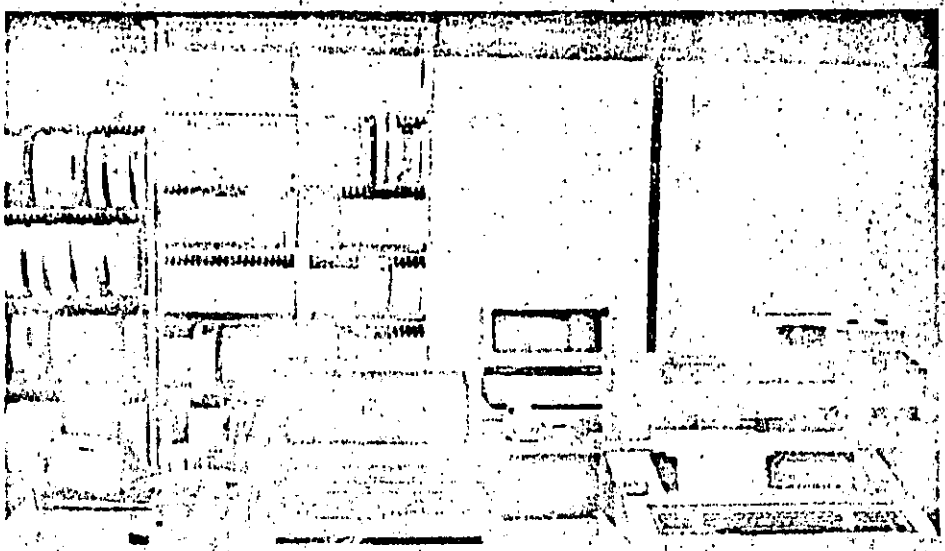
Si en la columna 1 aparece una C, indica que es un comentario que no será traducido por el compilador.

Una cadena de 1 a 5 dígitos, ajustados a la derecha, sirve de etiqueta para referenciar una sentencia. Por consiguiente sólo se etiquetarán aquellas instrucciones a las que tengamos que saltar o identificar.

La sentencia o instrucción se escribe desde la columna 7 a la 72. Algunas versiones permiten continuar la sentencia en otra línea, poniendo en la línea siguiente un carácter distinto.



El lenguaje FORTRAN está orientado especialmente a aplicaciones de tipo científico y técnico, desde la determinación de las órbitas de los satélites espaciales, a la definición de distancias entre elementos atómicos.



El primer manual de FORTRAN fue publicado por IBM en 1956, año en que comenzaron a redactarse numerosas versiones o dialectos, hasta llegar a nuestros días en los que el FORTRAN ya ha sido superado en eficacia por otros lenguajes.

EL LENGUAJE FORTRAN

de 0 o un espacio en la columna 6. Las posiciones 73 a 80 no son tenidas en cuenta por el compilador; se usan para identificar el programa y el número de secuencia dentro del mismo.

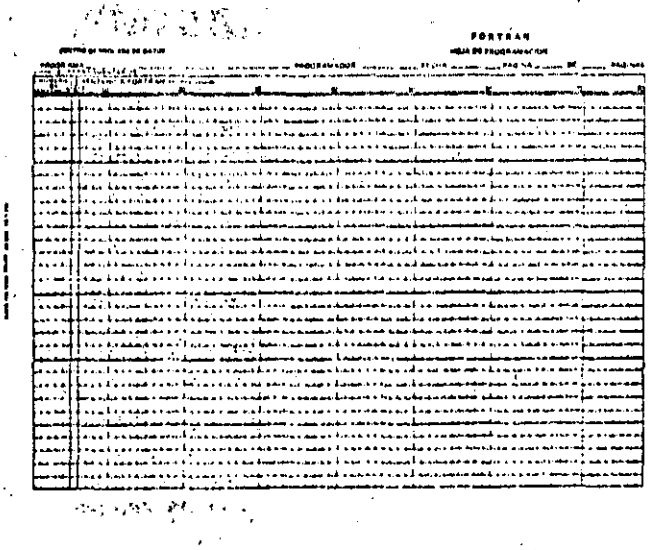
Datos y operaciones

El FORTRAN admite constantes y variables enteras, reales y de doble precisión. En general las de doble precisión, tan necesarias en el cálculo técnico-científico, permiten hasta 16 cifras decimales.

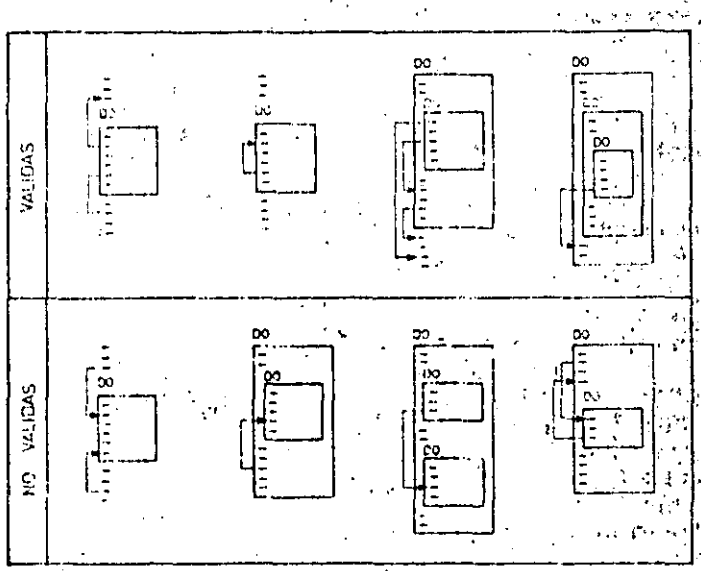
También se admiten constantes Hollerith, es decir, cadenas de caracteres que se escriben entre apóstrofos (por ejemplo, 'NOTA'). Las variables emplean nombres simbólicos que tienen una letra como primer carácter. El resto hasta 8 pueden ser letras o números. Si no se usan sentencias de especificación, el compilador entiende que toda variable cuya primera letra es I, J, K, L, M o N es una variable entera; mientras que las que empiezan por letras desde la A a la H y desde la O a la Z son variables reales.

Las variables matriciales se representan con el nombre seguido por los subíndices colocados entre paréntesis y separados por comas. Así, A(3,I,4) representa un elemento de una matriz tridimensional.

Las operaciones permitidas son: exponenciación (**), multiplicación (*), división (/), suma (+) y sustracción (–). Las normas de prioridad de operaciones y uso de paréntesis, vistos en el BASIC



La figura muestra una hoja de programación típica empleada para la redacción de programas en lenguaje FORTRAN. Dispone de 80 columnas. Las ocho últimas se usan para identificar el programa y el número de secuencia dentro del mismo.



Las instrucciones de tipo «DO» para el control de bucles pueden anidarse, aunque no deben interferirse. El cuadro muestra las estructuras válidas y erróneas de este tipo de instrucción.

```
ASSIGN s TO i
BACKSPACE u
BACKSPACE (alist)
BLOCK DATA [sub]
CALL sub [([e [,e]...])]
CHARACTER [*len[,]] nam [,nam]...
CLOSE (cilist)
COMMON [/[cb]/]nlist[[,]/[cb]/nlist]...
COMPLEX v [,v]...
CONTINUE
DATA nlist/clist/ [[,]nlist/clist/]...
DIMENSION a (d) [,a(d)]...
DO s [,] i = e1,e2[,e3]
DOUBLE PRECISION v [,v]...
ELSE
ELSE IF (e) THEN
END
END IF
ENDFILE u
ENDFILE (alist)
ENTRY en [([d [,d]...])]
EQUIVALENCE (nlist) [,(nlist)]...
EXTERNAL proc [,proc]...
FORMAT fs
fun ([d [,d]...]) = e
[typ] FUNCTION fun ([d [,d]...])
GO TO i [[,]s [,s]...)]
GO TO s
GO TO (s [,s]...)[,] i
IF (e) si
IF (e) s1, s2, s3
IF (e) THEN
IMPLICIT typ (a [,a]...)
  [,typ (a [,a]...)]...
INQUIRE (ilist)
INQUIRE (iulist)
INTEGER v [,v]...
INTRINSIC fun [,fun]...
LOGICAL v [,v]...
OPEN (olist)
PARAMETER (p = e [,p = e]...)
PAUSE [n]
PRINT f [,iolist]
PROGRAM pgm
READ (cilist) [iolist]
READ f [,iolist]
REAL v [,v]...
RETURN [e]
REWIND u
REWIND (alist)
SAVE [a [,a]...]
STOP [n]
SUBROUTINE sub [([d [,d]...])]
v = e
WRITE (cilist) [iolist]
```

Resumen de las sentencias del lenguaje FORTRAN.

son también de aplicación en el FORTRAN.

Existen numerosas funciones que permiten cálculos trigonométricos, logarítmicos, etc. El programador puede crear sus propias funciones.

Instrucciones

Veremos a continuación algunas de las instrucciones más características del FORTRAN.

Instrucciones de asignación. Tiene el formato

variable = expresión

y sirven para dar a una variable el valor obtenido en el cálculo de una expresión aritmética.

Ejemplo: Z(J) =
= (−B−SQRT(B *B−4 *A *C))/(2. *A)
que atribuye al elemento J de la serie Z el valor de una raíz de la ecuación $AX^2 + BX + C = 0$.

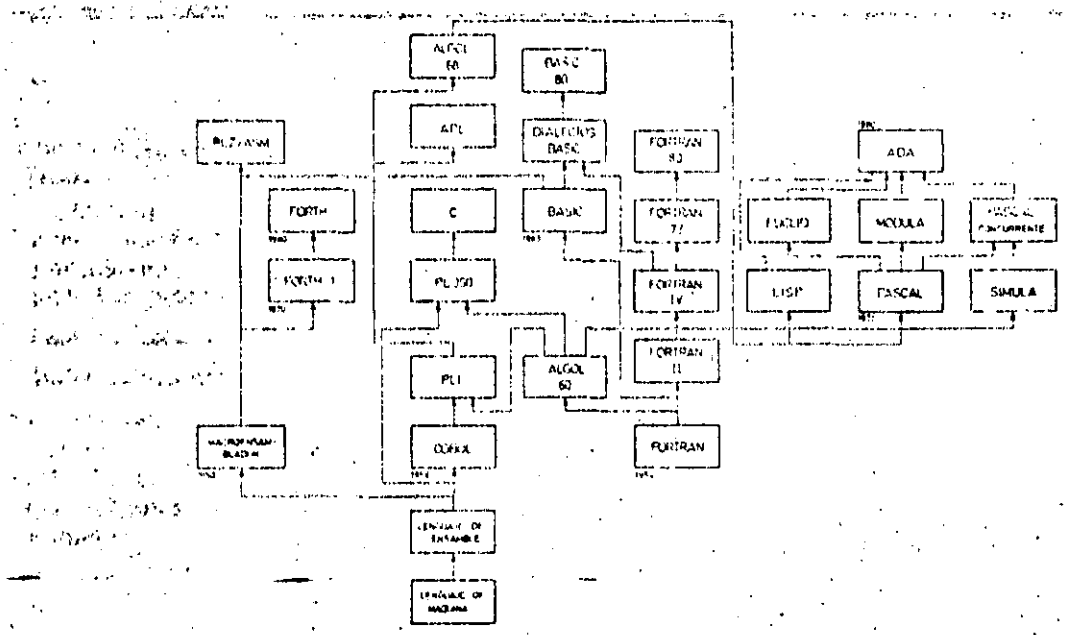
Instrucciones de control. Alteran la secuencia normal.

Salto incondicional: GO TO n (siendo n una etiqueta).

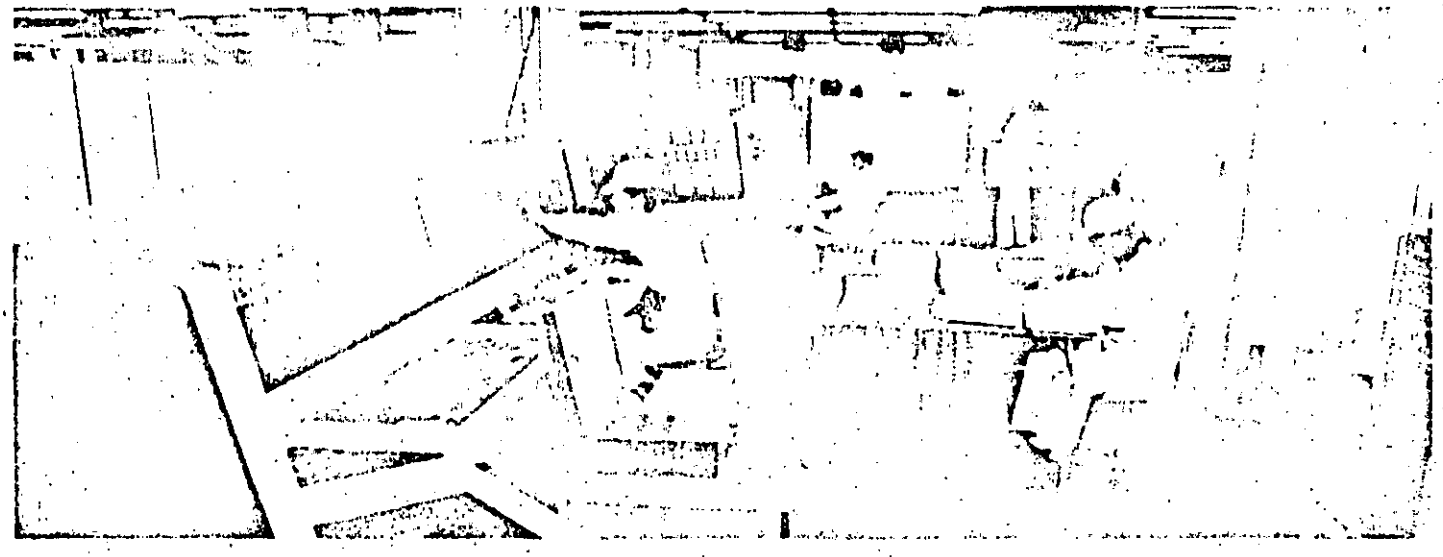
Salto calculado: GO TO (n_1,n_2,...n_n), expresión aritmética (continua en n_1, n_2, etc. según que el valor de la expresión sea 1, 2,...).

Salto condicional: IF(exp) n_1, n_2, n_3 (va a n_1, n_2, ó n_3 según que el valor de la expresión sea negativo, cero o positivo).

Bucle: DO n_1 ind = exp_1, exp_2, exp_3 (repite todas las instrucciones que siguen hasta la n_1 variando el valor del índice desde exp_1 a exp_2 en incrementos de exp_3 en exp_3).



El gráfico muestra la evolución histórica de los lenguajes de programación. Puede observarse cómo el BASIC tomó en un principio conceptos y sentencias del FORTRAN.



Ejemplo de aplicación escrita en FORTRAN. Este programa emplea los caracteres alfabéticos de la A a la Z, los dígitos del 0 al 9, además de otros símbolos especiales.

EL LENGUAJE FORTRAN

Ejemplo: S = 0

```
      DO 7    I = 2,10,2
    7 S = S + A(I)
```

(calcula la suma de los elementos pares de la serie A).
Fin de bucle: CONTINUE (se usa como última instrucción de un DO, en lugar de la que correspondería en caso de transferencia, lo que no está permitido).
Otras: PAUSE, STOP, END.
Instrucciones de declaración. Para reservar zonas de memoria. Dimensionado de matrices: DIMENSION nombre matriz (i_1, i_2 ... i_n). Para compartir la misma posición de memoria: EQUIVALENCE (V_1, V_2 ... V_n).
Para establecer correspondencia entre variables de diferentes unidades del programa = COMMON V_1, V_2 ... V_n

Ejemplo:
Programa principal:
```
DIMENSION a (10)
.
COMMON A, B (50, 10)
```
Subprograma:
```
COMMON D (10),
E (50, 10)
```

Otras: INTEGER, REAL, DATAS
Instrucciones de entrada/salida. Todas las entradas se realizan con la instrucción READ, mientras que las salidas se ejecutan mediante la instrucción WRITE. Ambas llevan asociadas una instrucción FORMAT que define el formato de lectura o escritura.
En resumen, el FORTRAN es un lenguaje muy fácil de aprender para las personas acostumbradas a la notación matemática, aunque hoy día está siendo superado por otros lenguajes más modernos.

Conceptos básicos

Evolución de los lenguajes de programación

El primer lenguaje que se utilizó en la programación fue el lenguaje de *máquina*, codificado en binario o hexadecimal y que sólo era comprensible para la máquina. Debido a su complejidad para el programador se desarrolló el lenguaje de *Ensamblo*, que aunque seguía estando próximo a la máquina, sustituía el código máquina por códigos nemotécnicos y simbólicos. A mediados de los años 50 aparecen los lenguajes *Macroensambladores*, con potentes instrucciones para sustituir a los procesos de codificación nemotécnicos largos e incomodos. Siguiendo los principios de los *Macroensambladores*, a principios de los 70 se desarrolló el FORTH 1 que dio lugar en los 80 al FORTH utilizado en microprocesadores, así como al PLZ/ASM. De forma paralela a los lenguajes *Macroensambladores* se desarrollaron lenguajes que se alejaban de la maquina y se aproximaban mucho más al problema. Los dos lenguajes históricamente mas importantes son el FORTRAN y el COBOL, el primero dedicado al campo científico y el segundo ligado al campo comercial y de gestión.
El FORTRAN fue presentado por IBM en 1954, su desarrollo dio lugar en 1970 al FORTRAN IV y en 1977 al FORTRAN 77.
En 1965 nace un lenguaje derivado del FORTHAN y que las universidades americanas empezaron a utilizar como lenguaje científico, el BASIC, que se ha desarrollado actualmente de tal manera que hoy es el lenguaje tradicional en los microordenadores. Otro lenguaje científico interesante que nació en el año 58 fue el ALGOL, cuya versión ALGOL 60 es la más representativa.
El COBOL se desarrolló en 1959, teniendo versiones mejoradas en los años 74 y 80, llamadas COBOL 74 y COBOL 80.
A la vez que se desarrollaban estos tipos de lenguajes, al principio de los años 60 se comienzan a diseñar lenguajes polivalentes, es decir, lenguajes que sirven para solucionar tanto problemas científicos como de gestión. Nace el PL/1, derivado del COBOL y del ALGOL 60; del PL/1 se deriva el APL, en los años 70, que se utiliza en trabajos interactivos y en la enseñanza asistida por ordenador. No hay que pasar por alto que actualmente el BASIC es también un lenguaje polivalente.
Siguiendo con la idea de llegar a un lenguaje universal, se desarrolló en los años 70 un lenguaje derivado del ALGOL 60 y del ALGOL 68, llamado PASCAL, cuyas versiones (PASCAL UCSD) y dialectos afines son utilizados ampliamente.
En la década actual los lenguajes PASCAL RECURRENTE y LIPS (derivado del ALGOL 68) han conducido al lenguaje ADA, que en principio tiene un amplio futuro. Por ejemplo, el Departamento de Defensa de los EE. UU. ha decidido que a partir de 1985 y con el fin de estandarizar sus aplicaciones, no aceptará ningún trabajo que no esté programado en ADA.

Glosario

¿Qué significa FORTRAN?

La palabra FORTRAN está formada por las sílabas iniciales de FORmula TRANslation, cuyo significado es traducción de fórmula, con lo que indica su utilidad para los problemas técnicos y científicos.

¿Por qué es necesario el cálculo numérico?

El lenguaje FORTRAN no admite más operadores que los aritméticos, mientras que los problemas técnicos precisan trabajar con integrales, derivadas, ecuaciones diferenciales, interpolaciones, ajustes de curvas, etc. El cálculo numérico es una colección de técnicas que permiten sustituir estos cálculos por conjuntos de operaciones aritméticas que dan una solución lo suficientemente aproximada.

¿Son suficientes dieciséis decimales para los problemas científicos?

Muchas veces no, ya que al sustituir los cálculos reales por métodos aproximados de cálculo numérico pueden ser precisas más cifras. El programador debe conocer la teoría de errores y tenerlo en cuenta a la hora de programar.

¿Por qué una línea FORTRAN sólo admite 80 caracteres?

Es una reminiscencia de las tarjetas perforadas, ya que cuando se desarrolló el FORTRAN éstas constituían el medio de entrada más ampliamente utilizado.

EL LENGUAJE COBOL

AUNQUE más lentamente que en el campo científico, los ordenadores entraron en el área de gestión con gran fuerza, sobre todo debido a la aparición de lenguajes de alto nivel orientados específicamente a los negocios. El primer lenguaje de este tipo, históricamente hablando, fue el FLOW-MATIC que en 1955 estableció el concepto de lenguajes de programación basados en palabras del lenguaje natural (en este caso el inglés). Fue creado por el doctor Hopper para UNIVAC. No obstante, el lenguaje de gestión que alcanzó más rápidamente la popularidad fue el COBOL, desarrollado a partir de 1959 por CODASYL. Conocido en un principio por COBOL 60, pretendía ser un lenguaje común a todos los ordenadores. Posteriormente han surgido nuevas versiones, por ejemplo: el COBOL ANSI 74, el COBOL-80 de Microsoft, el CIS-COBOL (que facilita el manejo de pantallas) y el RM-COBOL (para microprocesadores).

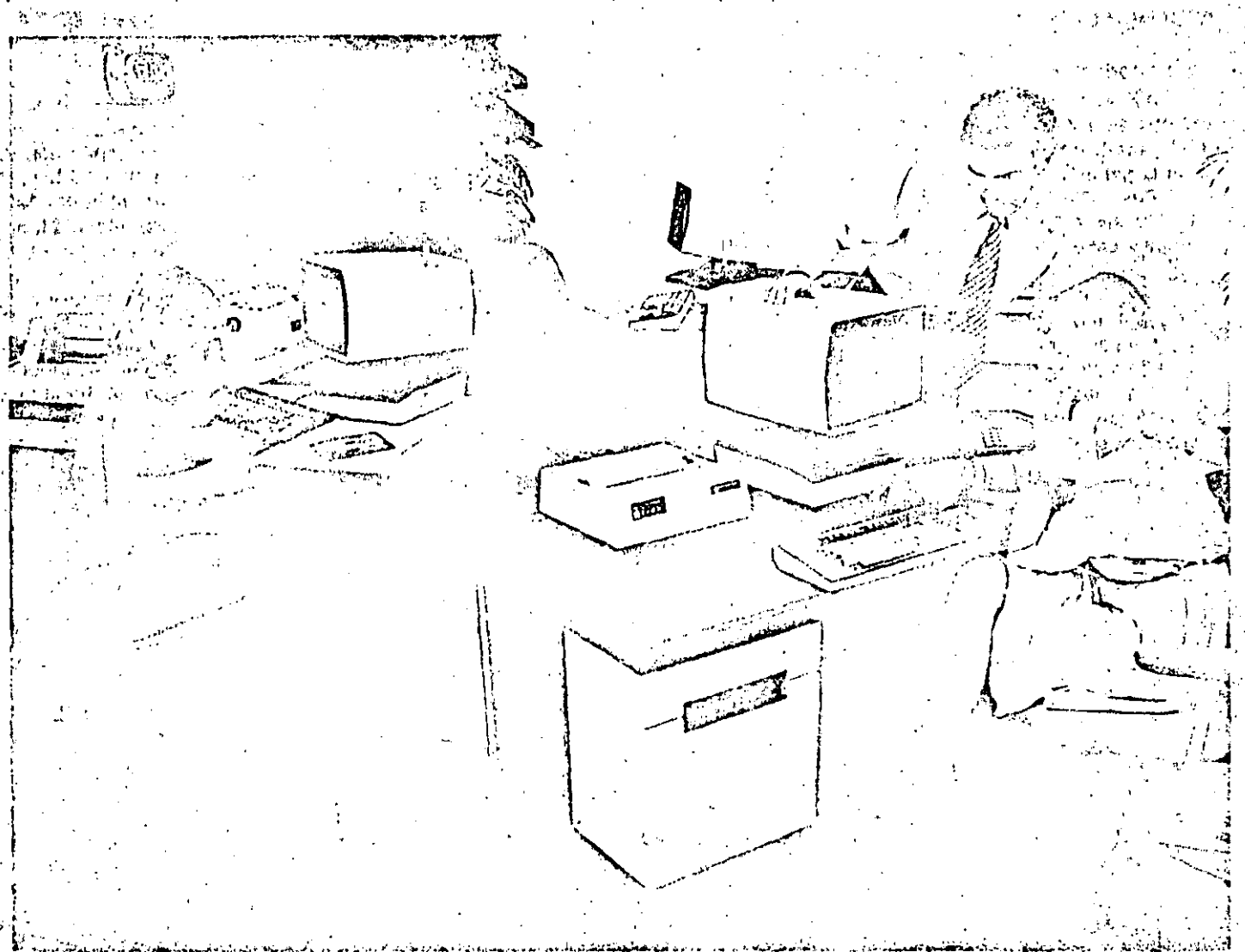
Ventajas e inconvenientes

El lenguaje COBOL tuvo un gran éxito, ya que incluía un concepto básico: el diseño de los datos es independiente de los algoritmos que van a operar con ellos. Este principio permite la definición minuciosa de los elementos a utilizar. Por lo demás, este lenguaje tiene gran capacidad de manejo de campos alfanuméricos, lo que es útil para programar salidas impresas en sistemas de gestión.

A pesar de haber sido superado por otros lenguajes, aún se utiliza ampliamente. Esto se debe a las numerosas aplicaciones desarrolladas a lo largo de veinte años y a la experiencia acumulada por los programadores.

Inicialmente era un lenguaje batch, si bien hoy existen numerosas versiones interactivas que incorporan instrucciones de acceso a pantalla.

Es el lenguaje más estándar de los existentes en la actualidad y los programas escritos en COBOL se pueden implementar fácilmente en distintos ordenadores. Además, su manejo no hace necesario un conocimiento profundo de matemáticas.



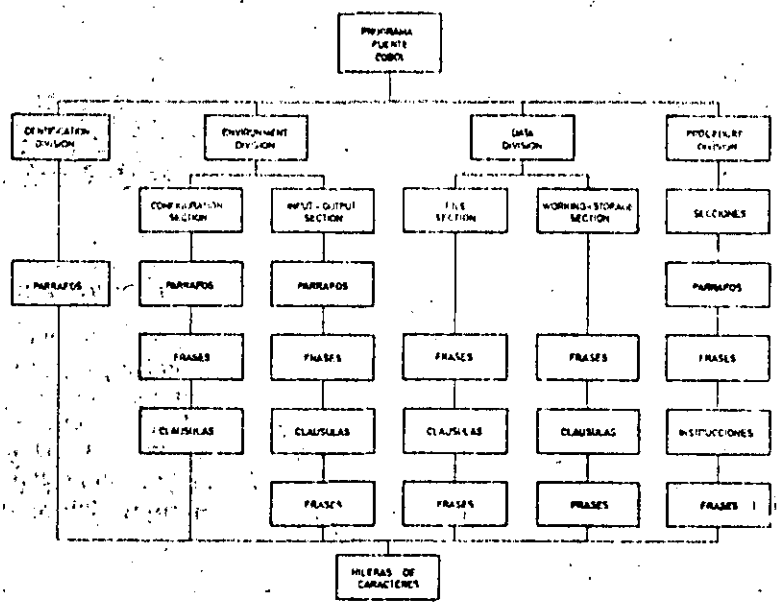
El lenguaje COBOL nació en 1959 orientado específicamente a la gestión administrativa y a los negocios. El objetivo de sus creadores era convertirlo en un lenguaje universal para todos los ordenadores

EL LENGUAJE COBOL

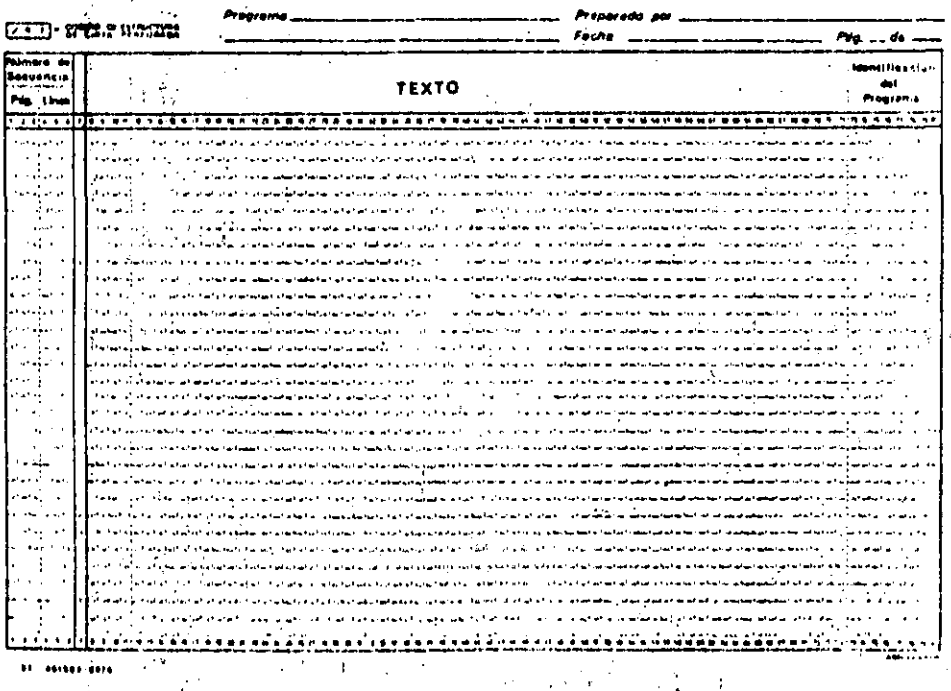
Estructura general

Un programa en lenguaje COBOL se encuentra jerarquizado de la siguiente forma: División, Section, Paragraph, Sentence, Statement, Word y Character. Las divisiones son cuatro, cada una de las cuales informa al compilador de un aspecto del programa. Las divisiones deben escribirse exactamente en el mismo orden en que se reseñan.

— *Identification Division:* Identifica al programa. Además del nombre del mismo, incluye información adicional sobre el autor, fecha, etc.
— *Environment Division:* Adapta el resto del programa a la configuración del sistema y a su sistema operativo. En teoría sería la única división que habría que cambiar si se traslada el programa a otro ordenador.
— *Data Division:* Describe las estructuras de los datos que van a ser procesados.
— *Procedure Division:* Contiene las instrucciones con las que el ordenador procesará los datos; esto es, el programa propiamente dicho.



La misión de las divisiones en las que se estructura el lenguaje COBOL es informar al ordenador de aspectos parciales del programa.



La figura muestra la organización de una hoja de codificación COBOL. Obsérvese la gran similitud que guarda con las hojas de programación en lenguaje FORTRAN.

Tipos de sentencias

Las divisiones se estructuran en secciones, según se indica en el gráfico adjunto. Mientras que los nombres de las secciones de la Environment Division y la Data Division están fijados por el propio lenguaje, en la Procedure Division los elige el programador, seguidos de la palabra «Section».

La Configuration Section describe el ordenador en el que se compila y ejecuta el programa; mientras que la Input-Output Section señala los periféricos utilizados y su asignación a los ficheros.

Las secciones de la Data División son: File Section (para los ficheros), Working Storage Section (para las zonas de maniobra), Constant Section (para las zonas de constantes) y Report Section para la descripción de salidas.

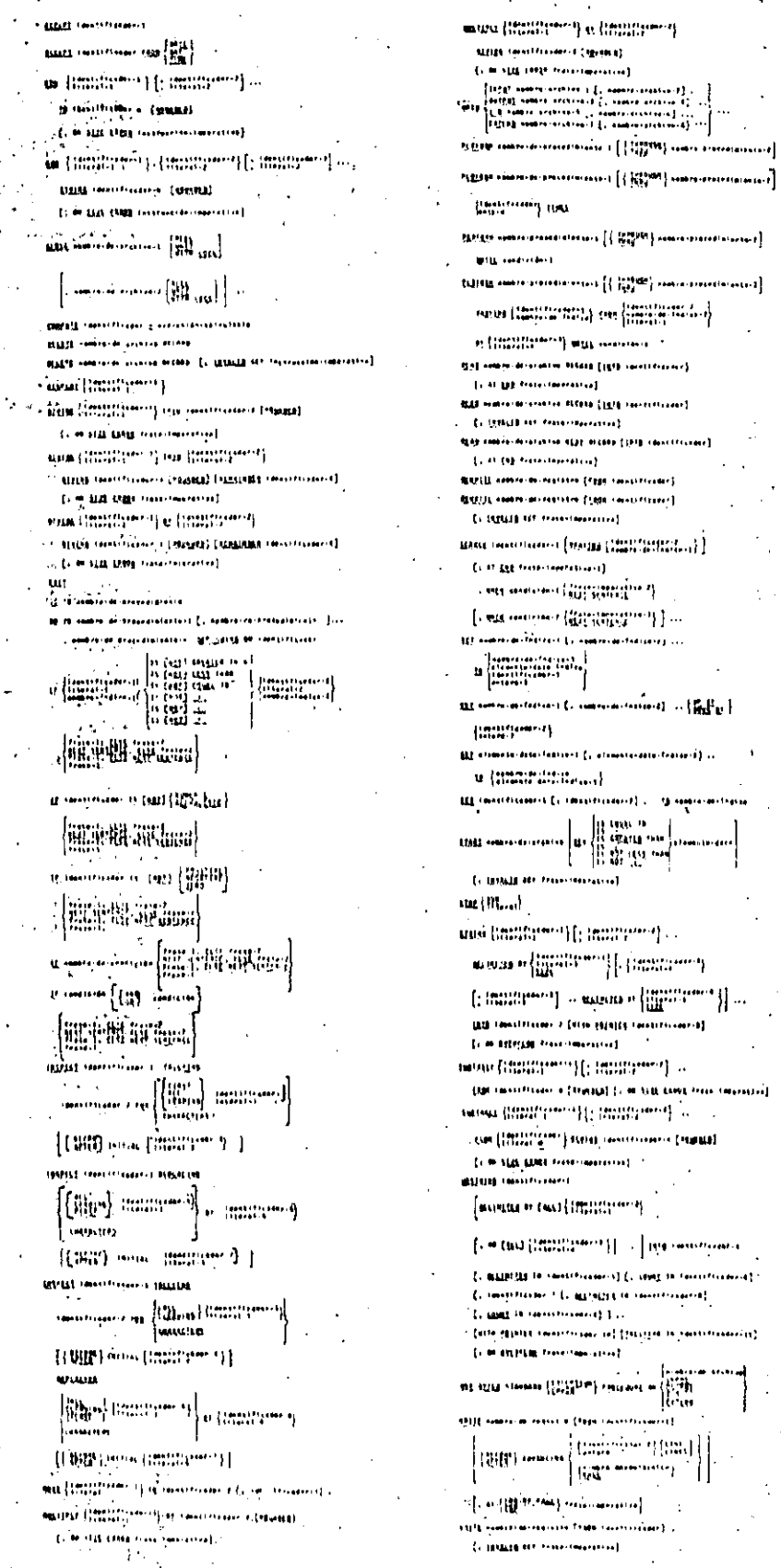
Los párrafos o subdivisiones de las secciones, constan de una o más frases que tienen una función común. Los párrafos de la «Procedure» pueden ser fijados por el programador, mientras que los nombres de los párrafos de las otras instrucciones están fijados por el lenguaje.

Los párrafos de la Environment Division y Data Division están formados por frases (entries) y cláusulas, mientras que los de la Procedure Division lo están por frases e instrucciones.

Una instrucción es una combinación sintácticamente válida de palabras y símbolos, que comienza con un verbo COBOL. Existen tres tipos de instrucciones: del compilador, condicionales e imperativas.

La hoja de codificación

Los programas fuente en lenguaje COBOL se escriben en hojas de codificación normalizadas, que recuerdan a las del FORTRAN. En este caso la línea de continuación se indica en la columna 7, mediante un guión, y la de comentario.



Formato de las instrucciones de la PROCEDURE DIVISION, encargada de contener el programa propiamente dicho.

Glosario

¿Qué significa COBOL?

La palabra COBOL está formada por las iniciales de Comon Business Language. (Lenguaje Común Orientado a los Negocios)

¿Qué es el CODASYL?

CODASYL son las siglas de Conference on Data Systems Languages (Conferencia sobre Lenguajes de Sistemas de Datos). Este comité fué convocado por el Departamento de Defensa de los Estados Unidos y estaba formado, entre otros, por las empresas Burroughs, IBM, Honeywell, RCA, Remington, Sperry Rand, Sylvania, USAF, Navy y el National Bureau of Standards.

¿Qué significa la página y línea en el número de secuencia de la hoja de programación COBOL?

Sólo un número de orden; aunque no tienen que estar consecutivos. El criterio general es numerar las líneas de 10 en 10 para permitir la intercalación de nuevas sentencias. La estructura sigue recordando la época en que se trabajaba con fichas perforadas (80 columnas) y la posibilidad de clasificar las fichas mediante el número de secuencia, utilizando una clasificadora.

¿Qué se entiende por verbo COBOL?

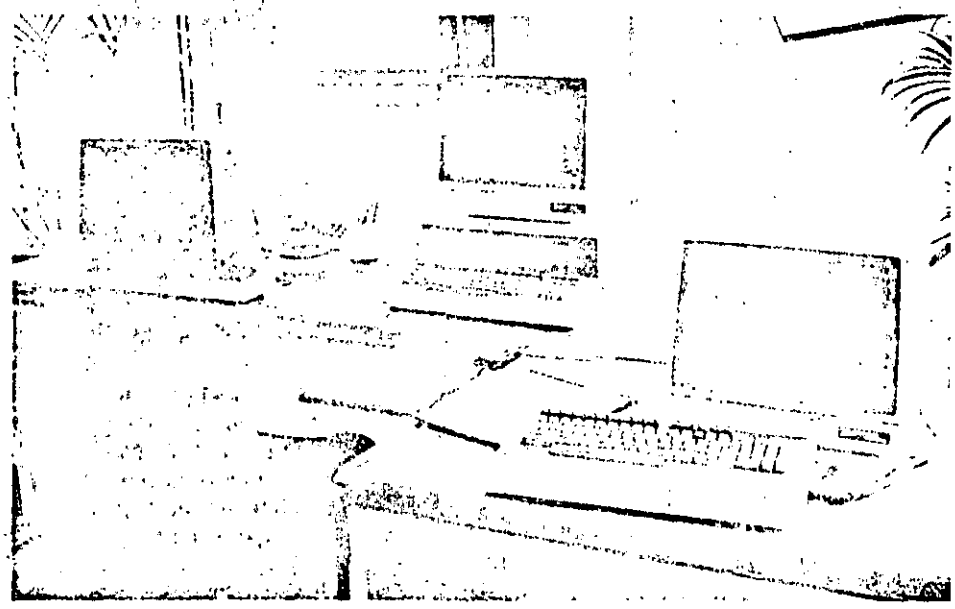
Un verbo o palabra COBOL es una hilera de caracteres que reconoce el compilador y le permiten generar el programa objeto. Los verbos son palabras reservadas del lenguaje.

EL LENGUAJE COBOL

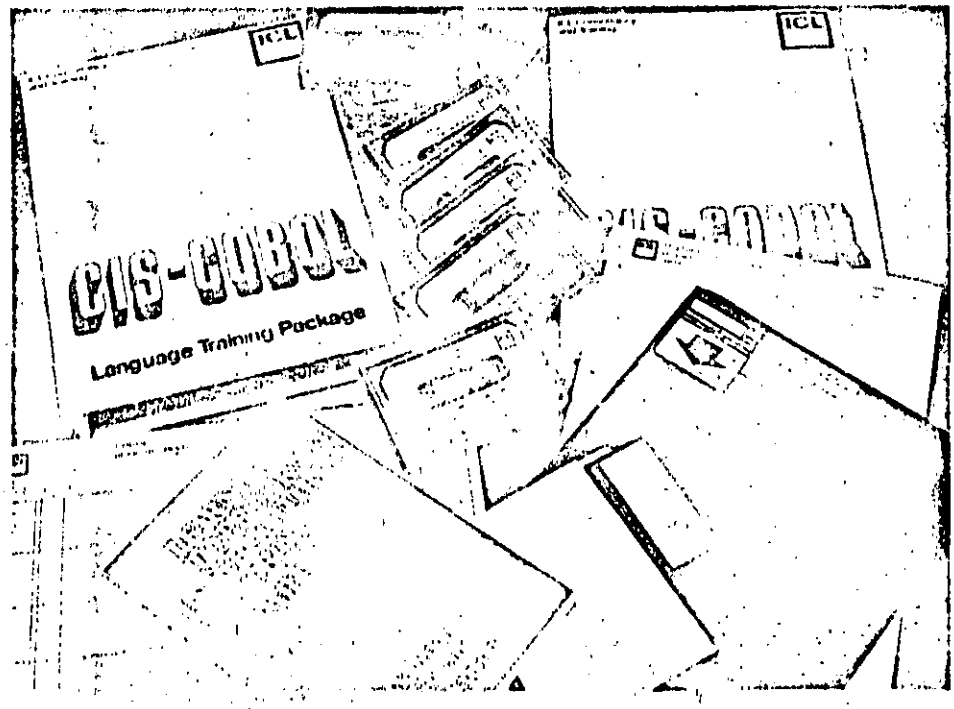
mediante un asterisco en dicha columna 7.

Al codificar un programa en COBOL, o en cualquier otro lenguaje, deben seguirse unas ciertas reglas, cuya misión es disminuir las posibilidades de error. Así, por ejemplo, se debe utilizar lápiz suave y goma de borrar en lugar de bolígrafo; se debe poner un solo carácter en cada posición; escribir ciertos caracteres (tales como el cero y la O, la zeta y el 2, la I o el 1) de forma que se eviten errores; tener cuidado con los puntos, etc.

En los cuadros adjuntos se exponen las instrucciones de la Procedure Division.



En la actualidad existen numerosas versiones o dialectos del COBOL 60 original. Cada uno de ellos supone una mejora o adaptación a necesidades del usuario, pero siempre dentro de la orientación primera: la gestión administrativa y los negocios.



En lenguaje COBOL (Common Business Language) pueden encontrarse miles de aplicaciones orientadas a la gestión administrativa, así como en los más diversos formatos para su entrada en el ordenador: disquette, cassette, etc.

Conceptos básicos

Codificación y control de errores

La codificación consiste en establecer una correspondencia entre la información que queremos representar y su representación, de forma que a cada información le corresponda una y sólo una forma de representación.

Como el ordenador maneja sólo información binaria, toda la información, tanto numérica como alfabética debe representarse mediante cadenas de bits.

Se han utilizado multitud de sistemas de codificación de la información, tanto dentro como fuera del ordenador. Veamos algunas de las razones que han servido de base para construir los códigos más usuales.

• *Número de símbolos a representar*

Al tener que codificar las 26 letras, los 10 dígitos y unos 30 símbolos especiales, se necesitan por lo menos 60 configuraciones binarias, por lo que el número mínimo de elementos del código ha de ser 6 bits ($2^6 = 64$ combinaciones posibles). En general se usan 7 u 8 bits por la razón que se explica seguidamente.

• *Detección y corrección de errores*

La necesidad de que no se produzcan errores en las distintas etapas, obliga a introducir mecanismos que detecten y corrijan los errores de forma automática. Esto se consigue utilizando más bits de los necesarios, los llamados bits de paridad. Se dice que un código es óptimo cuando para representar un símbolo se usa el menor número de posiciones binarias posibles. Cuando se emplean más de los estrictamente necesarios se dice que el código es redundante. Es esta redundancia la que asegura la fiabilidad del código.

• *Rendimiento de un código*

Se define el rendimiento de un código como el cociente entre el número de informaciones codificadas y la cantidad total que podrían representarse con el código utilizado. Se da en tanto por ciento.

Así, por ejemplo, si empleamos 6 bits para representar sólo diez dígitos, el rendimiento será:

$$n = 100 \times \frac{10}{2^6} = \frac{1.000}{64} = 15,6\ \%$$

SISTEMAS OPERATIVOS: EL MONITOR

La evolución de los ordenadores electrónicos ha implicado una gran complicación en la lógica de su funcionamiento. Para conseguir un uso más racional y un mejor aprovechamiento de los ordenadores se han desarrollado una serie de programas que constituyen el software funcional. Los constructores de ordenadores lo suministran bajo diversos nombres, aunque el más general es el de *Sistema Operativo*.

Existen diversas definiciones de Sistema Operativo, aunque nosotros seguiremos la del AUERBACH EDP Report:

«Sistema operativo es una colección ordenada de rutinas y procedimientos que acompañan al ordenador y que normalmente realizan todas o algunas de las siguientes funciones:

- Planificación, carga, inicialización y supervisión de la ejecución de programas.
- Gestión de memoria, unidades de entrada/salida y otros dispositivos.
- Inicialización y control de todas las operaciones de entrada/salida.
- Tratamiento de errores.
- Coordinación de las comunicaciones entre el sistema y el operador.
- Mantenimiento de un registro con las operaciones del sistema.
- Control de las operaciones en los trabajos de multiprogramación, multiproceso y tiempo compartido.

En resumen, el sistema operativo es el conjunto de los programas del sistema que permiten el usuario utilizar el computador de la máquina cómodamente y que optimizan su rendimiento. Una característica fundamental del sistema operativo es que debe incluir un programa *monitor* que controle la ejecución de los demás programas y mantenga el funcionamiento del ordenador, sin intervención del operador más que en caso de necesidad.

Tipos de sistemas operativos

Podemos distinguir cinco tipos principales:

- *Secuencial por lotes*. Permite ejecutar los trabajos uno a uno. Los programas pueden ejecutarse nada más ser introducidos o memorizarse en dispositivos de acceso rápido, ejecutándose secuencialmente más tarde.
- *Multiprogramación*. Permite que varios trabajos se ejecuten simultáneamente. Se consigue mediante el uso de las interrupciones.
- *Tiempo real*. Permite el uso del ordenador por varios usuarios que utilizan terminales remotos y efectúan constantemente operaciones de entrada y salida de datos.
- *Tiempo compartido*. Permite a muchos usuarios utilizar el mismo sistema, que aparentemente sólo está dedicado a cada uno de ellos, ya que cada usuario recibe el control de la CPU durante un determinado intervalo de tiempo.
- *Memoria virtual*. El sistema operativo asume responsabilidades de gestión de la memoria principal.

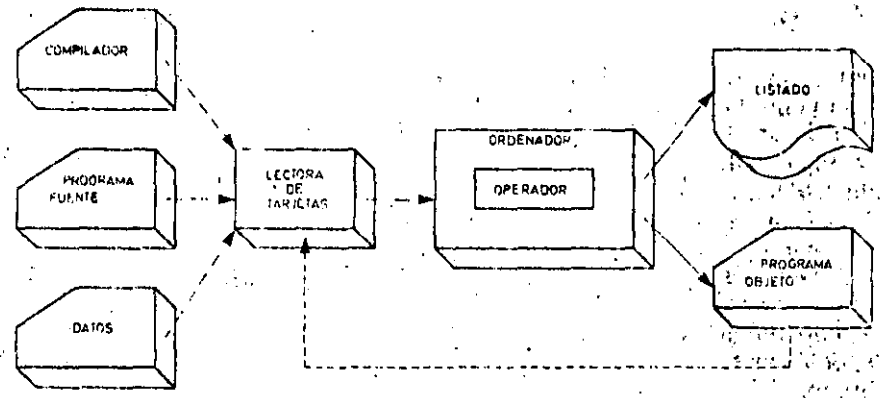
La implementación práctica de los sistemas operativos se hace mediante técnicas de «overlay» (recubrimiento) u «overlapping» (solapamiento).

Componentes de un SO

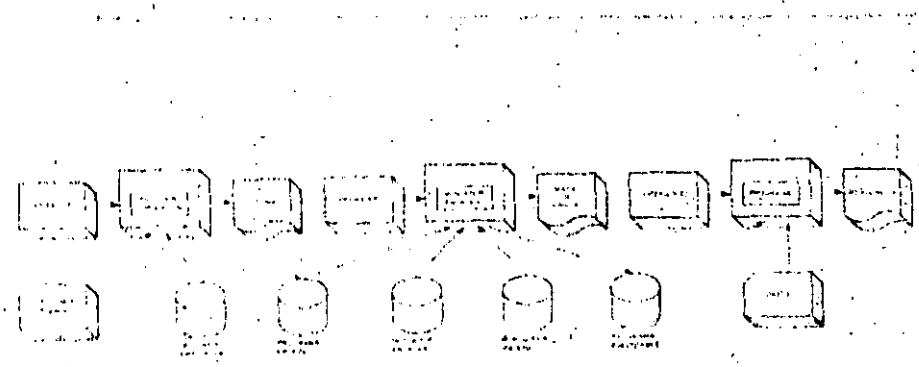
Los sistemas operativos de la tercera generación además del *monitor* o supervisor, encargado de la gestión de trabajos, contienen componentes que gestionan los recursos del sistema, al propio sistema y a los datos.

La gestión de trabajos se encarga de la organización y regulación del flujo de trabajo en el sistema. También permite que los usuarios se comuniquen con el sistema a través de los comandos del operador y las tarjetas de control.

La gestión de recursos del sistema abarca la asignación al programa seleccionado de los recursos necesarios:



Explotación manual de un ordenador con lectora de tarjetas y sin sistema operativo. La compilación, la generación del programa fuente y los procesos de entrada/salida deben ejecutarse manualmente.



Cadena de operaciones necesarias para gestionar un programa en un ordenador que utiliza memorias de discos. Las intervenciones del operador resultan inevitables, aunque vienen facilitadas por el empleo de un sistema operativo.

SISTEMAS OPERATIVOS: EL MONITOR

memoria, tiempo de CPU, operaciones de E/S, etc. Muchas veces estas tareas las realiza el monitor.

Las funciones de gestión de sistema comprenden la generación del sistema, la conservación del sistema y de los programas, y el interface con los compiladores.

La gestión de datos abarca la gestión de fichero, soporte de E/S para acceder y tratar un solo registro del fichero, facilitando las operaciones de búsqueda y aislamiento de una parte específica de un fichero.

El monitor o supervisor

El monitor o supervisor debe estar presente siempre en la memoria. A veces sólo reside en la memoria central una parte de él, la llamada *residente*. El resto es llamado cuando se necesita. El supervisor contiene, en general, todos los subprogramas que realizan las funciones básicas. El sistema supervisa la actividad del programa, rechazando las operaciones no válidas y evitando de esta manera la detención del ordenador por los errores del programa del usuario.

Los elementos del monitor son: control de trabajos, control de E/S, comunicaciones y recuperación del sistema.

El control de los trabajos comprende las funciones que controlan y regulan el uso de los recursos del sistema y, en particular, de la planificación de trabajos, de la asignación de recursos, de carga y de la terminación de programas.

El control de E/S regula las actividades de los dispositivos de E/S. Comprende: la planificación de los recursos de E/S, la transferencia de datos y el soporte de los terminales remotos.

El sistema de comunicaciones se responsabiliza de los intercambios de información entre el S.O. y los usuarios. Cuando un error impide la continuación normal de un trabajo intervienen las rutinas de recuperación, que permiten la reanudación de un trabajo a partir de un determinado punto del proceso.

La planificación de trabajos

La planificación depende del tipo de sistema operativo utilizado, existiendo una gran cantidad de técnicas de planificación. La planificación pretende la utilización más eficiente del sistema y lo normal para lograr este objetivo es que trabajos con muy alta prioridad, y baja utilización de recursos se ejecuten antes que otros con baja prioridad, pero con mejor utilización de recursos. Las técnicas de planificación más comunes son:

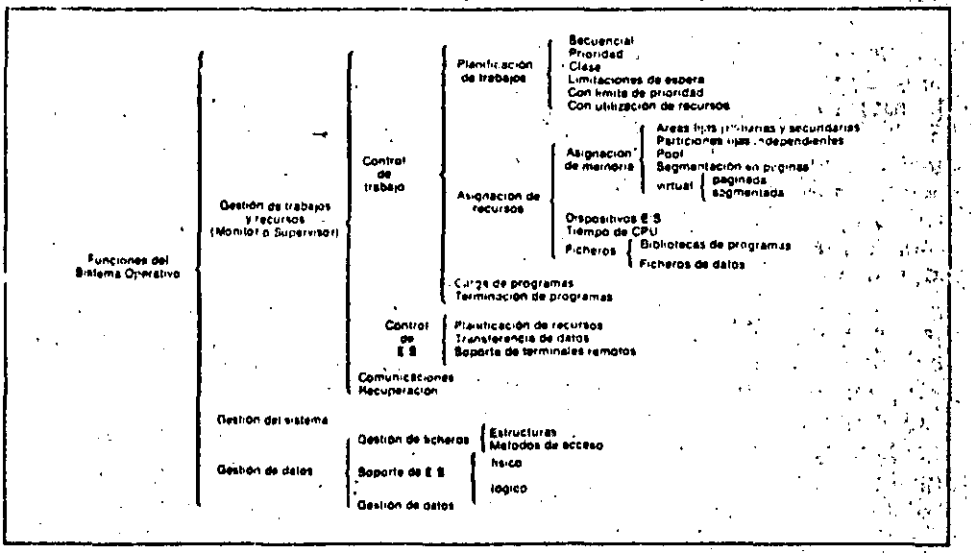
- *Planificación secuencial.* El primer trabajo introducido es el primero en atenderse. Se leen todas las instrucciones y datos de entrada y se almacenan en una memoria de acceso rápido.

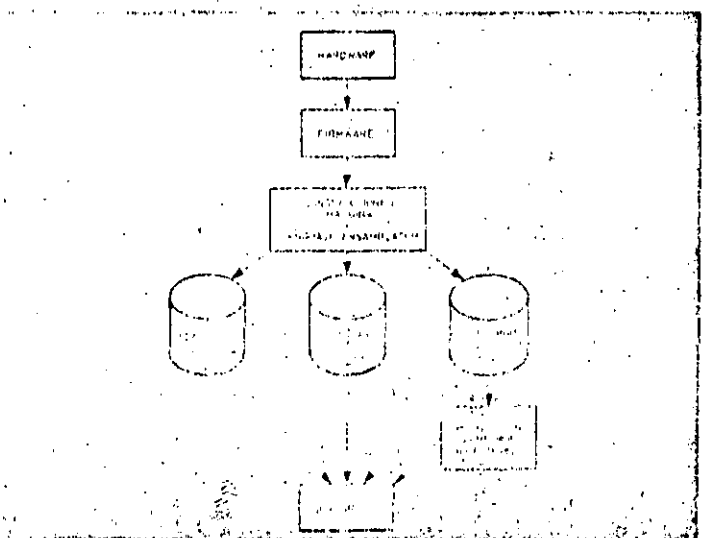
- *Planificación por prioridad.* Esta técnica asigna un código o número a cada programa, que indica el orden en que deben procesarse los trabajos. Los que tienen el mismo número o clase se colocan en colas dentro de la clase. Los trabajos de más alta prioridad son los primeros en ejecutarse.

- *Planificación por clase.* Los trabajos se agrupan en clases y en cada clase se atribuye a cada trabajo una prioridad numérica.

Dentro de cada clase se da preferencia a los trabajos que tienen prioridad numérica más alta. En general se utiliza en sistemas de participaciones de memoria fija: a cada clase corresponde una participación.

- *Planificación con limitación de espera.* Podría suceder, empleando los métodos anteriores, que algunos programas de muy baja prioridad no llegaran a ejecutarse nunca. En este método se asigna un tiempo límite de arranque a cada programa. El sistema operativo





Conjunto de recursos empleados por un ordenador. Entre el usuario y los elementos físicos de la máquina, o hardware, se intercalan todos los procedimientos de microprogramación (o firmware) y de software.

puede comprobar si, con la prioridad normal, el trabajo se va a ejecutar o no. En caso negativo se le asigna una prioridad mayor. Si no puede acabar en ese tiempo, el sistema pide la intervención del operador.

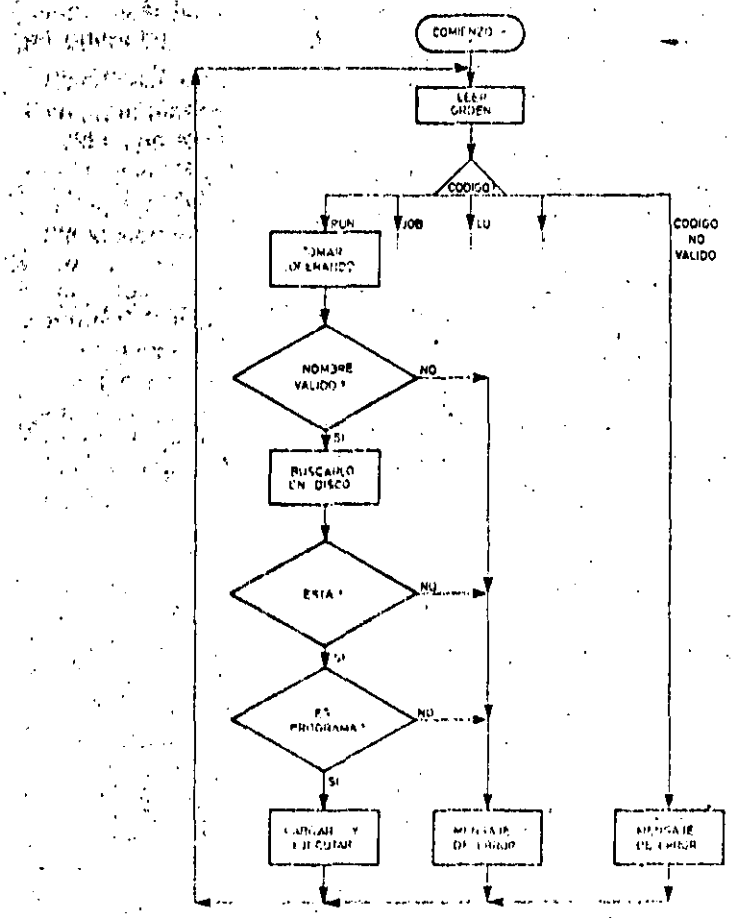
• *Planificación con límite de prioridad.* Si en un número determinado de veces el sistema no ejecuta un programa se le asigna a éste la prioridad más alta, con lo que entra rápidamente en servicio.

• *Planificación con utilización de recursos.* En este método el usuario debe estimar los recursos exigidos por el programa, tales como tiempo de CPU, número de líneas de impresión, unidades de cinta magnética, etc. El sistema asigna una prioridad utilizando un algoritmo que pretende optimizar la utilización de los recursos totales.

Asignación de recursos

El sistema tiene que gestionar la asignación de los recursos del ordenador a los distintos programas mediante rutinas particulares que evitan los conflictos entre los diversos programas. El usuario debe indicar, mediante los parametros del JCL, los recursos que requiere de memoria central, dispositivo de entrada/salida, tiempo de CPU y ficheros.

Muchos sistemas operativos sólo seleccionan un trabajo si todos sus recursos necesarios estan disponibles. Esta técnica implica un gran desaprovechamiento del equipo. Por ello es preferible la técnica de *asignación dinámica de recursos*, que permite ocupar los recursos sólo durante el tiempo en que son utilizados.



Organigrama o diagrama de flujo de un programa monitor. Cualquier comando ejecutado en un ordenador debe ser supervisado a través de un programa de este tipo.

Glosario

¿Qué es el «overlay»?

«Overlay» significa recubrimiento. Este término designa a la técnica empleada cuando la memoria necesaria para el conjunto de datos y de las instrucciones es mayor que la disponible.
La solución consiste en que varios módulos de programa ocupen la misma área en tiempos diferentes, llamando para ello a las rutinas que hagan falta desde las memorias periféricas.

¿Qué es el «overlapping»?

«Overlapping» significa solapamiento. La técnica utiliza la transferencia de datos de una parte a otra de la memoria mientras que la ejecución continúa en otro lado. Es necesario la existencia de procesadores de E/S, de memoria controlada independientemente.

¿Qué es un «Job»?

Un «job» o trabajo es un aplicación global que puede utilizar uno o varios programas de tratamiento, los cuáles, a su vez, pueden llamar a otros programas de tratamiento como compiladores, cargadores, etc.

¿Qué es una tarea?

Tarea es la unidad más pequeña de trabajo que puede acceder a los recursos del sistema. Un trabajo se compone, por tanto, de una o varias tareas.

¿Qué es la generación del sistema?

Es la operación que permite adaptar el sistema operativo a la configuración especifica del hardware.

¿Qué es JCL?

El JCL es el «Job Control Language», es decir, el lenguaje de control del sistema. Consta de un conjunto de órdenes que permiten al usuario comunicarse con el sistema operativo.

¿Tiene inconvenientes la paginación?

Si, ya que el sistema operativo tiene que llevar un mapa de las páginas que están en memoria y en disco. Supone además un fuerte trabajo de «thrashing» es decir, de carga y descarga de páginas entre la memoria central y las periféricas.

SISTEMAS OPERATIVOS: EL MONITOR

Otras funciones del monitor

Existen diversas técnicas en la asignación de los dispositivos de E/S en función del tipo de proceso. Pueden ser asignaciones fijas o asignaciones dinámicas.

La asignación de tiempo de trabajo a los distintos programas es realizada por la rutina «dispatcher».

Los ficheros del sistema son de dos tipos: bibliotecas de programas y ficheros de datos.

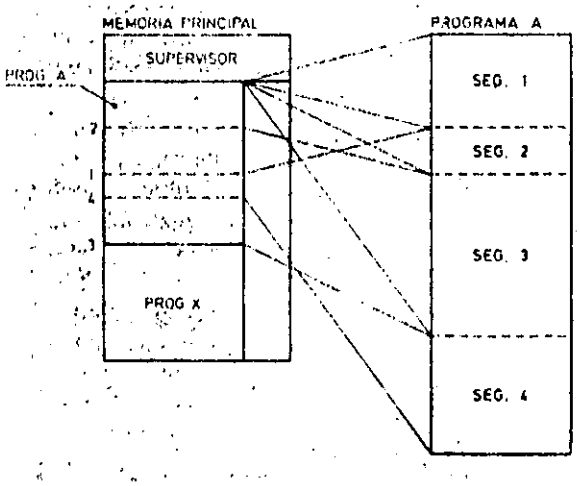
Los programas y sus rutinas pueden ser exclusivos o compatibles. Las estructuras de programa que manejan más frecuentemente los sistemas operativos son la de recubrimiento, la dinámica y la paginada.

Aunque no todos, muchos sistemas operativos realizan diagnósticos de errores, servicios de temporización, servicios de prueba y depuración.

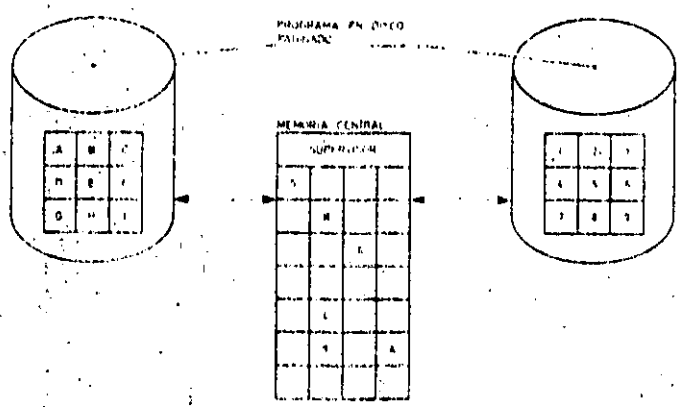
El *diagnóstico de errores* reconoce tanto los errores de hardware como de software y, una vez solventados, reintenta la ejecución. Si el error permanece pide la intervención del operador. Además lleva un archivo con un histórico de los errores que detecta.

Los *servicios de temporización* permiten parar o arrancar de nuevo un programa al cabo de un cierto tiempo y proporcionar fecha y hora a los programas en ejecución.

Las posibilidades de *prueba y depuración de programas* son muy variadas, permitiendo la corrección de los mismos. *Quizá la posibilidad más interesante* viene dada por el *editor*.



Cuando el espacio de memoria disponible es menor que el necesario para almacenar un programa completo se recurre al «overlay»: el programa se divide en segmentos que van entrando, secuencialmente, en la memoria principal.



Todas las páginas y, por tanto, todas las áreas de la memoria virtual son del mismo tamaño. Esta técnica presenta el inconveniente de aumentar los tiempos de ejecución.

Conceptos básicos

Métodos de asignación de memorias

Cuando se trabaja en multiprogramación hay varios métodos para asginar la memoria.

Asignación de particiones fijas e independientes. Las particiones tienen diferentes dimensiones y a cada programa se le asigna a la partición más pequeña que lo pueda contener.

POOL de memoria. A daca programa se le asigna una cantidad de memoria suficiente para contenerle. Cuando termina de ejecutarse el programa, la memoria vuelve al pool.

Segmentación en páginas. Las páginas de tamaño fijo y de dimensiones relativamente pequeñas (de 1/2 a 4 K) se mantienen en un pool. Las instrucciones de los programas tienen que dividirse en páginas. Implica el uso de memorias de masa de acceso rápido. El programa reside en memoria secundaria y en la memoria central sólo se encuentran las páginas que están ejecutándose.

Memoria virtual. Se usan discos de gran velocidad para expandir la memoria principal, que de esta forma parece más grande de lo que es realmente. La memoria virtual puede ser paginada o segmentada.

La memoria *virtual paginada* tiene la ventaja de que el programador no tiene que ocuparse de cómo entra el programa en una determinada partición, aunque el sistema operativo aumenta su trabajo de «thrashing».

La memoria *virtual segmentada* utiliza particiones variables, tanto de memoria real como de memoria virtual, basándose en una división lógica del programa en segmentos. Si éstos no son todos iguales hay una infrautilización de la memoria real, ya que hay que reservar memoria para el segmento mayor.

SISTEMAS OPERATIVOS: GESTION DE DATOS

El sistema operativo se ocupa del movimiento de los datos entre los dispositivos de E/S y la memoria principal. Para ello es necesario que en la memoria principal se preparen áreas de «buffer».

Los programadores de aplicación, sobre todo de los sistemas pequeños, deben reservar parte de la memoria como «buffer» de fichero.

Existen diferentes métodos para la asignación de los «buffer»: buffer único, buffer doble y grupo de buffer.

El método de *buffer único* asigna a cada fichero un área fija de memoria independientemente del hecho de que la aplicación esté utilizando, o no, al fichero.

El método de *buffer doble* es análogo al anterior, pero asigna dos áreas fijas de memoria a cada fichero.

El último método utiliza una zona grande de memoria, en la que hay varias zonas para los buffers. Cuando es necesaria una operación de E/S en un fichero, un «buffer» del grupo es asignado al fichero. Cuando acaba la operación, se libera al «buffer» y se devuelve al grupo. De esta forma se consigue un ahorro de la cantidad de memoria correspondiente a los buffers, siempre y cuando no se trabaje con todos los ficheros simultáneamente.

La transcodificación de datos

El sistema operativo llama a las rutinas de transcodificación de datos cuando son necesarias, haciéndolas trasparentes para los programas de aplicación.

En los sistemas pequeños, muchas veces, la transcodificación tiene que ser resuelta por el programador en su programa de aplicación.

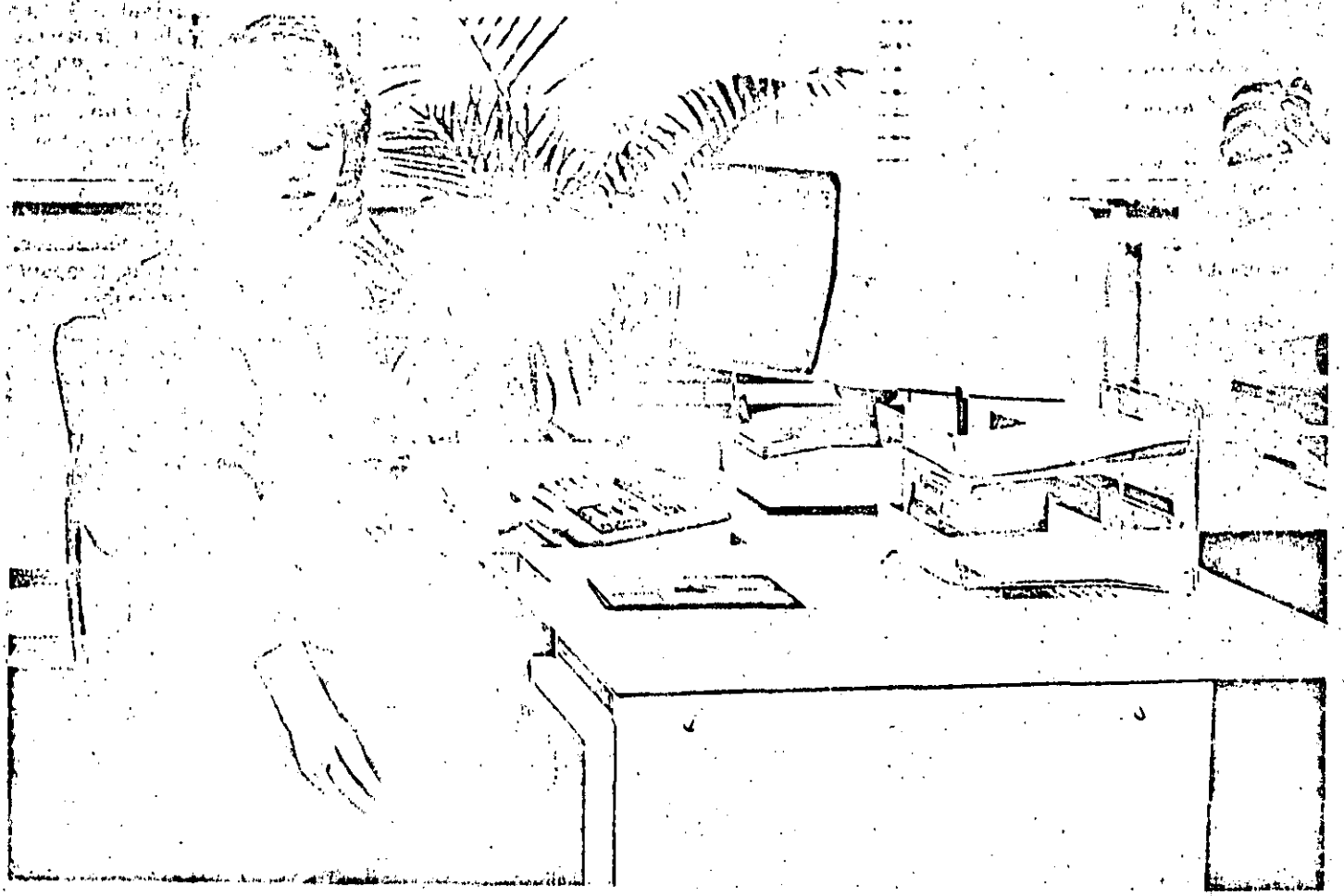
En el caso de entrada de trabajos remotos el sistema operativo gestiona también las comunicaciones entre el ordenador y los terminales remotos, así como las comunicaciones entre los diferentes usuarios.

La gestión de datos comprende la gestión de ficheros, el soporte de E/S y el sistema de gestión de datos.

La gestión de ficheros

Las funciones de gestión de ficheros se orientan al control de dichos ficheros. Aunque los ficheros son un conjunto de registros, el sistema de gestión los administra como entidades independientes.

Los ficheros permanentes se identifican con las *etiquetas*, que pueden ser asignadas, bien por el usuario, o bien por el sistema. La etiqueta de un fichero puede tener diferentes datos, tales como el identificador del fichero, el número de edición, el propietario, la pa-



El sistema operativo simplifica al usuario el empleo de los ordenadores. Consigue, además, racionalizar y optimizar su uso mediante una serie de programas internos entre los que se encuentra la gestión de datos.

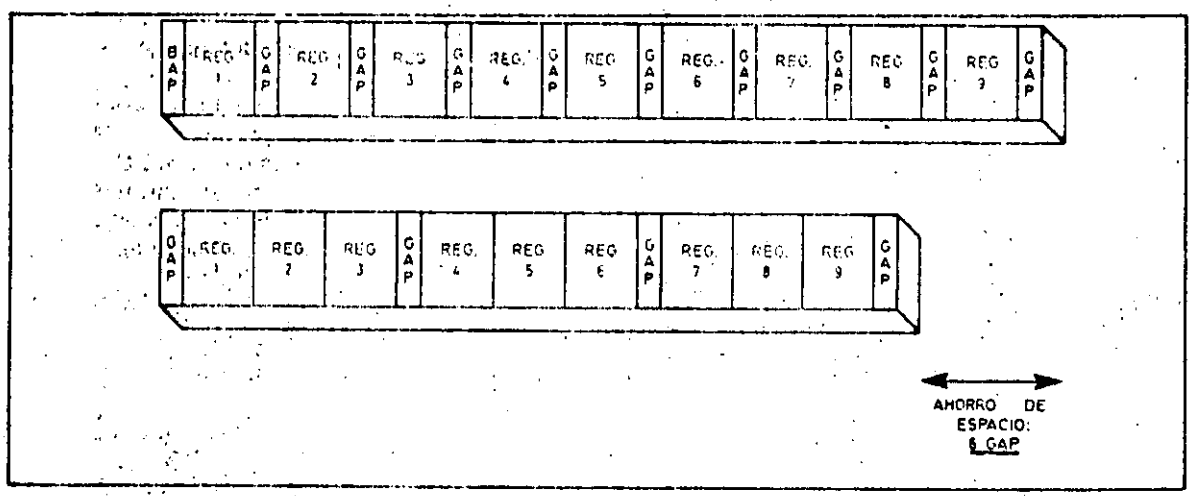
SISTEMAS OPERATIVOS: GESTION DE DATOS

labra de orden para el acceso, etc.
En los sistemas «batch» y de «time sharing» se mantiene normalmente un catálogo, o directorio, con la localización de todos los ficheros conocidos por el sistema. En caso de que el sistema no mantenga un directorio, el sistema compara secuencialmente todas las etiquetas hasta que encuentra el fichero que necesita para el programa en proceso.
Muchos de los sistemas operativos incorporan rutinas de utilidad para facilitar las copias de seguridad, de forma que cualquier daño sufrido por los ficheros se puedan subsanar. Estas funciones de recuperación pueden ser iniciadas por el sistema, automáticamente, o por el operador, a petición del sistema.

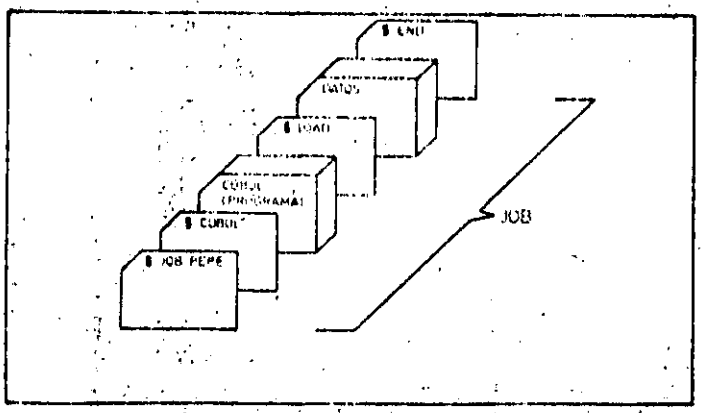
Soporte de E/S

Estas funciones se realizan, tanto a nivel físico como a nivel lógico.

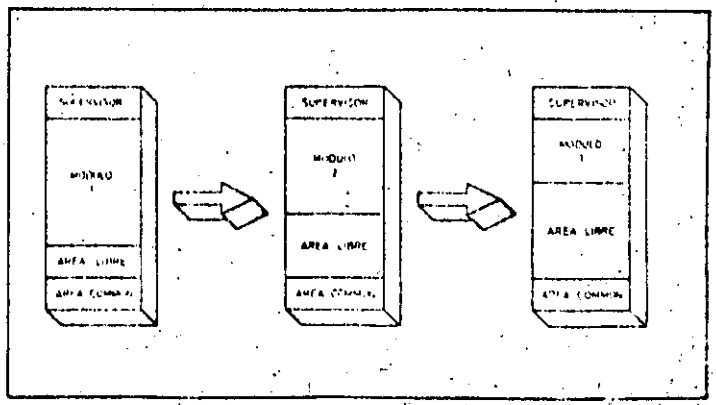
Las rutinas que controlan la E/S física llaman a las operaciones de transmisión de datos, y gestionan, en parte, el acceso de los programas a los datos, teniendo en cuenta los formatos de transmisión.
A un nivel superior, las rutinas de E/S lógicas permiten la manipulación de los datos, con independencia de su estructura física. Estas rutinas constituyen un intermediario entre las operaciones de datos del usuario y la E/S física del sistema.
El soporte de E/S permite a los programas acceder y trabajar con un solo registro en el fichero, con lo que el programador no tiene por qué conocer los problemas de la lectura y escritura de los registros.
Los sistemas operativos gestionan, también, las estructuras de los ficheros y los métodos de acceso a los mismos.
Hay diversas técnicas para asignar las memorias periféricas a los ficheros. En las más sencillas casi toda la gestión, incluida la protección contra destrucciones accidentales, queda en manos de los usuarios; las más complejas asignan, dinámicamente, los espacios necesarios en los discos cuando son solicitadas.
Las principales estructuras de ficheros son la secuencial, la jerárquica, la de índices, la de listas y la de estructura en bucle.
En la estructura secuencial todos los elementos son del mismo rango y están colocados en serie. En la estructura jerárquica el sistema dispone de un esquema de posición que clasifica y memoriza todos los elementos del fichero.
En la estructura de índices el fichero reserva ciertas porciones de memoria para las claves, a fin de localizar la información en el fichero. La característica de la estructura de lista es que cada elemento contiene la dirección del siguiente. En las estructuras en bucle las



El bloqueo de registros en una cinta magnética aumenta la capacidad de almacenamiento de la cinta y disminuye los tiempos totales de acceso a la información.



Ejemplo de un paquete de tarjetas de control que constituye un «job».



Los datos de un programa segmentado se almacenan en un área global («common») lo que permite su empleo en las diversas fases de la ejecución del proceso.

listas son circulares, es decir, el último elemento de cada una de ellas contiene un puntero con la dirección del primer elemento.

Los métodos de acceso, generalmente soportados por los sistemas operativos, son: acceso secuencial, acceso con índice, acceso con claves y acceso aleatorio.

El acceso secuencial puede realizarse con cualquier tipo de memoria auxiliar y es el único que puede emplearse en cintas y casetes.

La búsqueda de un registro en el acceso con índice se hace a través del directorio.

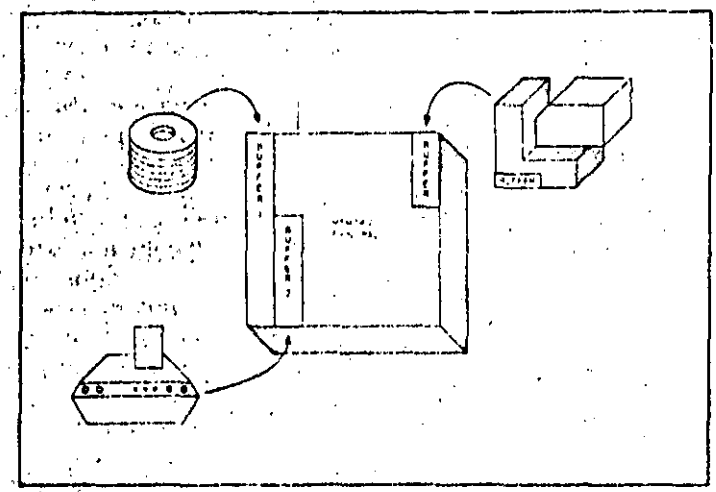
El acceso con claves es muy útil para unidades de memoria que usan instrucciones de búsqueda cableadas en hardware, ya que de esta forma se libera al procesador de las búsquedas en la memoria secundaria.

Para el acceso aleatorio el sistema utiliza generalmente un algoritmo que establece una correspondencia unívoca entre la clave identificadora del registro y la dirección de memoria en el dispositivo, que necesariamente ha de ser de acceso directo.

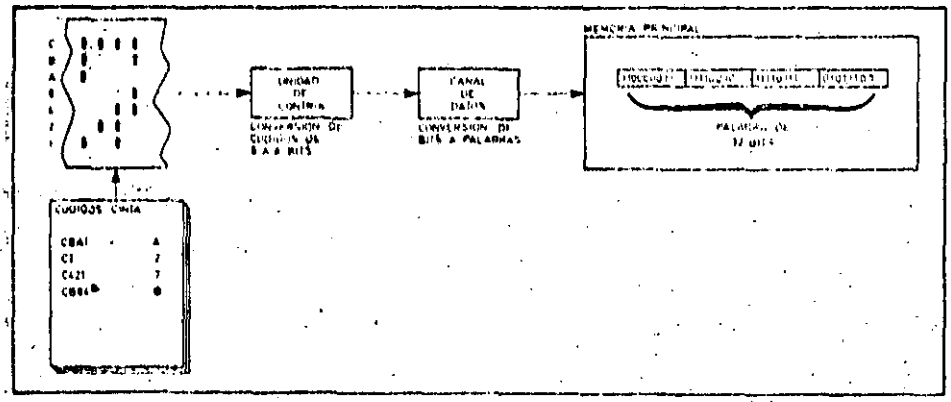
Empaquetado y bloqueo

Con el empaquetado se reúnen en un único bloque físico a varios registros lógicos. El desempaquetado permite aislar un registro del bloque físico de datos. Los registros pueden ser tanto fijos, como variables. Este método exige un buffer de E/S de mayor tamaño que el registro usado. Esta pérdida de memoria por ocupación de «buffer» queda de sobra compensado por la mejor utilización de la memoria periférica.

El mejor aprovechamiento de las memorias externas obliga, también, a bloquear los registros. Hay sistemas operativos que sólo gestionan la E/S física.



Un buffer es un área de almacenamiento intermedio que libera el procesador de trabajo durante la transmisión de datos. A veces los buffers suelen estar localizados en los mismos periféricos.



Cuando el código empleado para almacenar datos en las memorias de masa difiere del código empleado para almacenarlos en la memoria principal, es necesario recurrir a la transcodificación de datos

Glosario

¿Qué es un «buffer»?

Un buffer es un área de memoria que sirve de almacenamiento intermedio entre la memoria central y un periférico. Hay buffers de entrada y buffers de salida.

¿Qué tamaño tiene un «buffer»?

Depende del periférico que la utiliza. Así un buffer de impresión tiene normalmente la capacidad de una línea; un terminal de pantalla requiere un buffer de una página; un buffer de disco requerirá el espacio capaz de contener un sector.

¿Por qué es necesaria la transcodificación?

Porque, en general, la información se almacena en los soportes con unos códigos diferentes a los empleados en la memoria central.

¿En qué se diferencian las estructuras de los ficheros de los métodos de acceso?

Los métodos se refieren al conjunto de rutinas que se usan para introducir en memoria o buscar datos en las memorias de masas. Las estructuras se refieren a los tipos de organización que permiten a los usuarios la clasificación de los datos.

¿Qué es el bloqueo?

Bloqueo es la operación que agrupa varios registros lógicos en un solo registro físico. Así, si, por ejemplo, los registros tienen 50 caracteres y el sector de un disco es de 200, se bloquean los registros de cuatro en cuatro y se aprovecha al máximo la capacidad del disco.
El bloqueo es muy común en archivos en cinta magnética.

¿Qué significan los términos «sort» y «merge»?

Sort significa clasificar, ordenar.
Merge significa mezclar, fusionar.
Las rutinas de sort-merge son las que permiten la clasificación e intercalación de ficheros.

SISTEMAS OPERATIVOS: GESTION DE DATOS

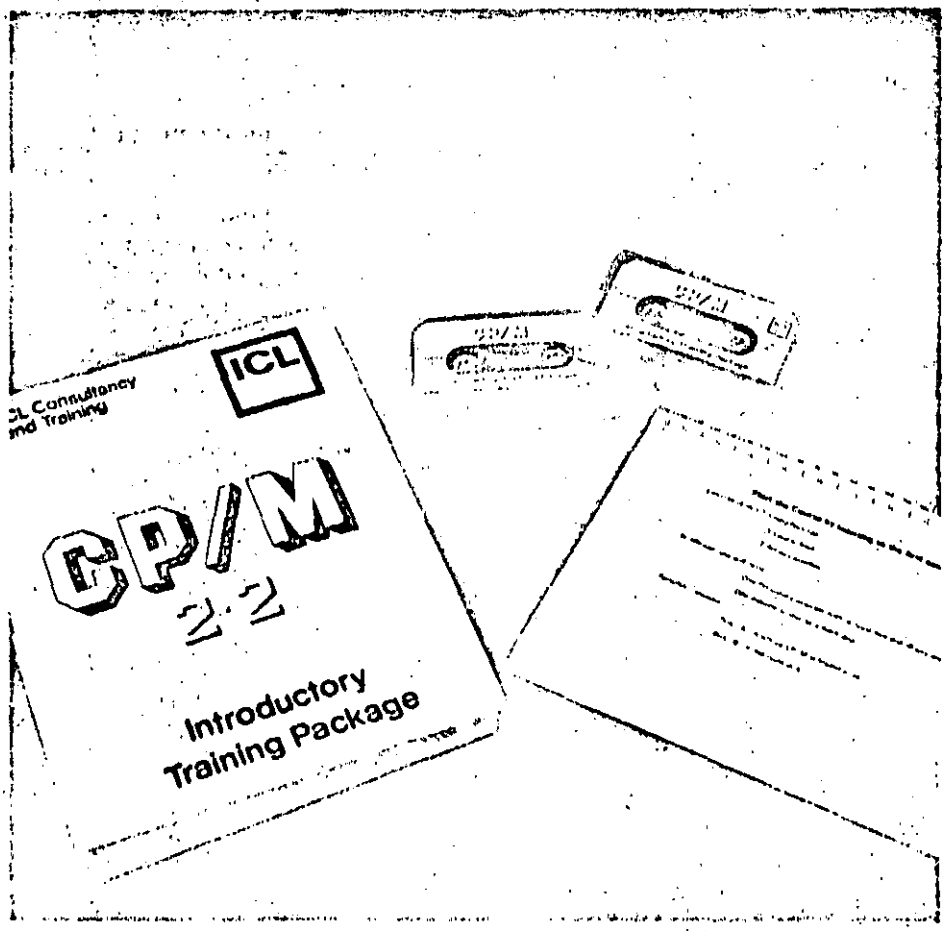
por lo que el usuario tiene que realizar las operaciones de bloqueo y desbloqueo. La gestión de soporte E/S lógico, por los sistemas operativos, permite operar a nivel de registro, sin tener en cuenta la estructura de los bloques físicos.

Funciones de manipulación de datos

Las rutinas de manipulación de datos pueden ser llamadas de diferentes formas: por los programas, a través de una tarjeta de control, o mediante la intervención directa del operador.
Estas rutinas son de dos clases: de representación visual y de soporte de periféricos.
Las rutinas de representación visual proporcionan la visualización de la memoria principal, las tablas de los programas, los directorios y los datos que se encuentran en memorias periféricas. Realizan, también, la conversión de datos a disco o de cinta, a impresora. El soporte de periféricos abarca la conversión de los soportes de memoria, edición de datos, enrollamiento de cintas magnéticas, etc.
Aunque no son exclusivas del sistema operativo, las funciones de clasificación y fusión de ficheros están incorporadas en muchos de ellos, como utilidades. La fusión de ficheros se realiza sólo cuando éstos se encuentran previamente clasificados.

El sistema de gestión de datos

Las funciones del sistema de gestión de datos son realizadas por las rutinas que afectan a las bases de datos. Estas rutinas se añaden a los sistemas operativos.



Los sistemas operativos se ofrecen como opción al usuario de un sistema informático. Suelen almacenarse en una memoria de masa, generalmente disco magnético, y se cargan en la memoria principal en el instante en que se solicite.

Conceptos Básicos

Programa editor

Si durante la compilación o ensamblaje de un programa fuente se detectan errores, es necesario corregir ciertas instrucciones y proceder a un reensamblaje o recompilación. Si el programa fuente se ha introducido en forma de tarjetas es necesario sustituir, añadir o quitar algunas de éstas.
Hoy día es más frecuente introducir los programas directamente en el ordenador, mediante un teclado. El tener que volver a teclear todo el programa fuente como consecuencia de un error puede ser una gran pérdida de tiempo. Para obviar ese inconveniente existen los programas llamados *editor* de textos (text-editor) que aceptan órdenes de modificación, adición o supresión de líneas, mediante códigos parecidos a los del lenguaje de control.
El texto del programa fuente se almacena en un fichero de disco, y en él se realizan las correcciones.
La operación de edición se realiza, tanto en forma interactiva, a través de un terminal de pantalla o un teletipo, o en forma batch, introduciendo las órdenes de edición y los textos corregidos en el flujo de entrada.
La forma más práctica de emplear el programa editor es en modo interactivo. De esta forma se pueden introducir modificaciones en discos que contengan tanto programas fuente, procedimientos compuestos por una secuencia de órdenes de control, o ficheros de datos numéricos o alfanuméricos.
El programa editor evita el uso de un soporte intermedio para la entrada de información al ordenador, a cambio de una entrada más lenta, la manual.

SISTEMAS OPERATIVOS PARA MICROPROCESADORES

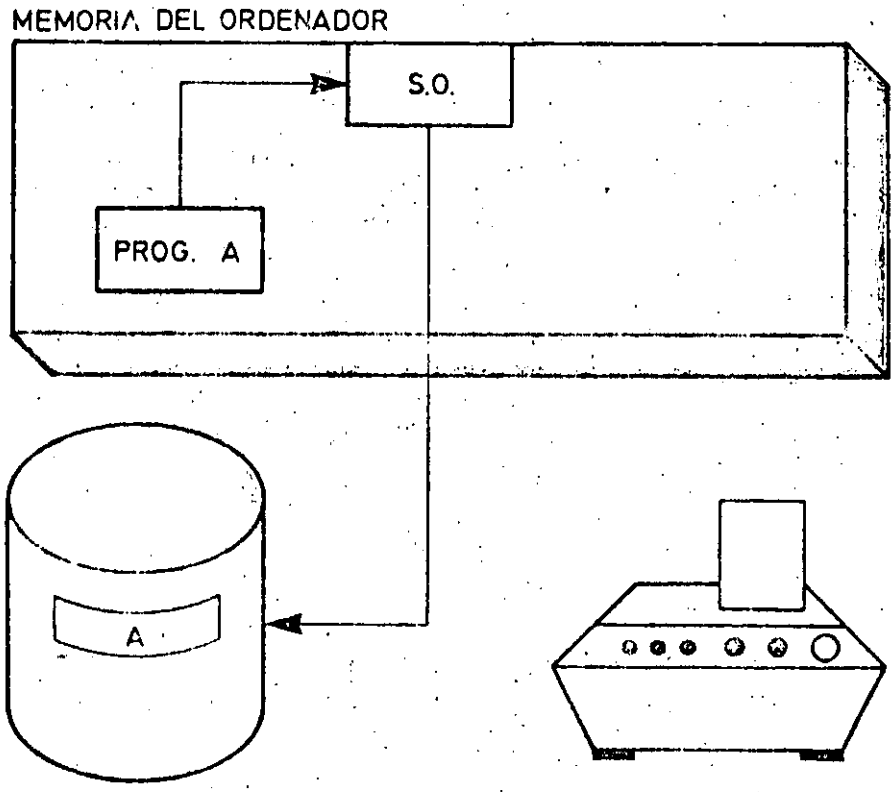
TODOS los ordenadores, salvo los más elementales, utilizan sistemas operativos que liberan al usuario de la programación de rutinas de control y le ayudan en el uso de los soportes magnéticos, listados de ficheros, formateado de soportes, etc.
La pequeña capacidad del microordenador no permite el uso de todas las funciones de los sistemas operativos vistos en los temas anteriores.

Tipos de sistemas operativos para microordenadores

La existencia de microprocesadores de 8 y 16 bits, la posibilidad de ser utilizados por varios usuarios y la cada vez más importante opción de redes de microordenadores, dan lugar a diversos tipos de sistemas operativos.
Así, hay sistemas operativos monousuario para 8 y 16 bits, sistemas operativos multiusuarios para 8 y 16 bits, sistemas operativos concurrentes y sistemas operativos para redes de microordenadores. Aunque algunos microordenadores sólo permiten el uso de un sistema operativo específico, la tendencia actual es que los sistemas operativos sean estándar y puedan, por tanto, utilizarse en diversos equipos.

Sistemas operativos monousuario para 8 bits

Los primeros ordenadores personales de 8 bits, como el Apple y el Commodore Pet, tenían sus propios sistemas operativos. Pero la Compañía Digital Research ha logrado imponer su sistema operativo CP/M como sistema estándar de los microordenadores de 8 bits. Este sistema soporta actualmente muchas más aplicaciones que cualquier otro sistema del mismo tipo. Este liderazgo es tan importante que la mayoría de los microordenadores, incluidos Apple y Commodore, aunque tienen sistemas operativos propios, permiten a sus usuarios el procesar paquetes de aplicaciones en CP/M.
El CP/M tenía inicialmente muchos puntos débiles en cuanto a seguridad, recubrimiento de errores y documentación, pero las utilidades que proporcionaba en el manejo de disquetes le proporcionó rápidamente una fuerte base de aplicaciones. Las últimas versiones del CP/M han corregido muchos de los fallos iniciales y han incluido el manejo de ficheros, protecciones mediante «password», y gestión de mayor número de unidades de discos y de memorias RAM mayores.
Su principal debilidad es haber sido desarrollado para el conjunto de instrucciones de las familias de procesadores 8080 del Intel y Z80 de Zilog, que están perdiendo posiciones frente a los microprocesadores de 16 bits.
Otros sistemas operativos de este grupo son las versiones para 8 bits del UCSP y del TURBODOS, que comentaremos más adelante.



1. El programa A, que debe imprimir 250 páginas, se inicia. Las páginas se van almacenando en el «spool» en disco.

SISTEMAS OPERATIVOS PARA MICROPROCESADORES

Glosario

¿Qué es un sistema concurrente?

Un sistema operativo que permite a un solo usuario el procesamiento simultáneo de varias tareas.

¿Qué es el «password»?

Password es un término inglés que significa «palabra de paso». El password es una clave o «contrasoña» necesaria de conocer para acceder a ciertos programas y ficheros.

¿Qué es el «spool»?

El spool es un archivo en disco en el que se almacena la información de salida durante la ejecución de los programas. De esta forma la impresión se lleva a cabo de forma continuada, una vez que el programa ha finalizado.
El spool mejora el rendimiento de las impresoras y es especialmente útil cuando se tiene una única impresora, se trabaja en multiprogramación, y varios programas requieren salidas impresoras.

¿Qué es el «booting»?

El booting es un programa inicializador. Permite la carga del programa residente del sistema operativo.

¿Qué significa «utilities»?

Es un término inglés que designa a las rutinas de utilidad.

¿Qué es «linkar»?

Es una palabra de la jerga informática derivada del verbo sajón «link», que significa enlazar.
La operación de link permite, por un lado, que las rutinas de utilidad se enlacen adecuadamente con los programas que los necesitan, y permite, por otro lado, el enlace correcto con el siguiente trabajo, una vez acabado el programa anterior.

Sistemas operativos monousuario para 16 bits

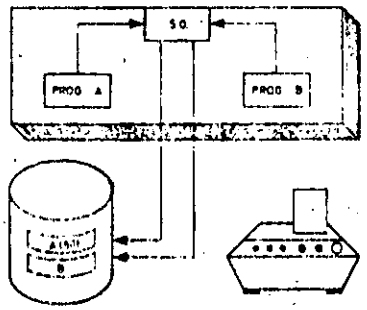
La aparición de los microprocesadores de 16 bits y la incorporación a los ordenadores personales de los discos rígidos han aumentado considerablemente la capacidad de proceso de estos sistemas. Pero el conjunto de instrucciones de los nuevos microprocesadores, basados principalmente en la familia de microprocesadores Intel de 16 bits, es totalmente diferente del conjunto de los antiguos sistemas de 8 bits.
Por ello se ha reescrito el CP/M, del que ha surgido el sistema operativo denominado CP/M-86. Este sistema estaba llamado a ocupar el mismo lugar en el mundo informático que su antecesor. Pero IBM encargó a MicroSoft un sistema operativo para su ordenador personal, el MS-DOS. Como la compatibilidad de software de cualquier equipo con el software de IBM es una característica importante que intentan ofrecer la mayoría de los fabricantes, el sistema operativo MS-DOS recibió un fuerte impulso, colocándose en cabeza de este grupo de los 16 bits.

El MS-DOS es bastante similar, incluso en el nombre de los comandos, al CP/M-86, con el que es además bastante compatible. Su uso es más sencillo y el manejo de disco mucho más rápido que el del CP/M-86. Su principal inconveniente es el número de aplicaciones que soporta actualmente, más reducido que el del sistema operativo de Digital Research.

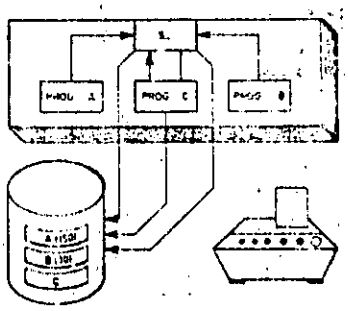
La versión posterior del MS-DOS, el MS-DOS 2-0, es muy parecido a otro producto de MicroSoft, el XENIX. Mantiene un interface de usuario del tipo CP/M, aunque incorpora menús, función de ayuda en un contexto sensitivo, y estructura de ficheros jerarquizada.

Aunque no es realmente un sistema operativo multitarea, incorpora, de hecho, funciones que permiten el correo electrónico, el «spool» de impresión, y comunicaciones con otros sistemas.

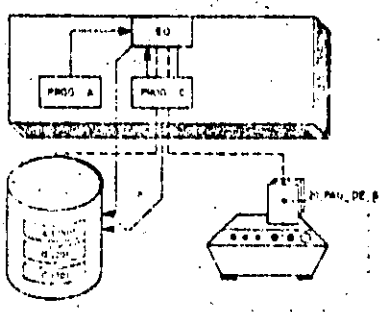
El UCSD es el tercer sistema operativo en cuanto a difusión. Tiene muchas fa-



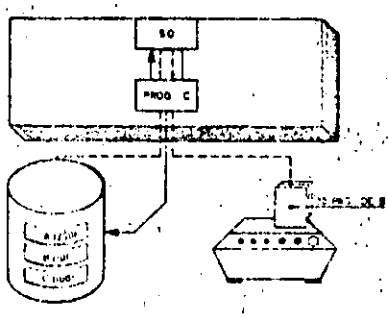
II. Cuando se han almacenado 50 páginas del programa A empieza la ejecución del programa B y se abre un nuevo fichero en el spool para almacenar las 40 páginas que imprimirá este programa.



III. Aún no han terminado de ejecutarse los programas A y B, y empieza la ejecución de un nuevo proceso C. Se abre un nuevo fichero para almacenar sus resultados.



VI. Al cabo de poco tiempo el programa D «aborta», debido a un error. El operador interviene entonces para anular el archivo D del spool. Ordena, además, retener la salida del A y sacar cinco copias correspondientes al programa C.



VII. El programa A finaliza. La impresora está ocupada por el programa B y el C no ha terminado aún de ejecutarse.

cilidades comunes con el MS-DOS y el CPM-86 y es, además, mucho más compatible y trasladable de un equipo a otro. Sus principales inconvenientes son su lentitud y la rigidez de la estructura de comandos, fuertemente jerarquizada.

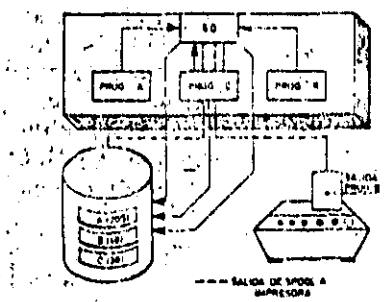
La característica principal del UCSD es el uso de un pseudocódigo a nivel de máquina, que lo independiza de un microprocesador específico. Con un traductor adecuado, cualquier ordenador puede ejecutar los programas en UCSD. La adaptación de programas requiere sólo la elaboración de los traductores. La lentitud viene provocada precisamente por la traducción del código a lenguaje máquina.

Mediante la utilización del lenguaje USCD-Pascal desciende la importancia del pseudocódigo, y se incrementa el papel de la compatibilidad entre el sistema operativo y el lenguaje. También soporta FORTRAN 77, Basic, APL y LISP.

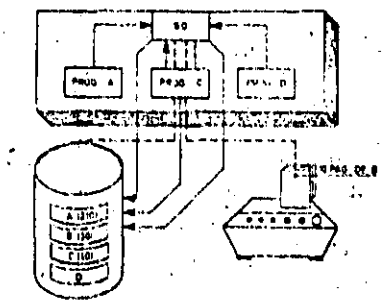
Siguiendo el camino del USCD, la compañía MicroProducts Software ha producido el sistema operativo BOS, para el que existe un lenguaje, el Microcobol, que es un híbrido del PL/1 y el COBOL.

Sistemas operativos multiusuario para 16 bits

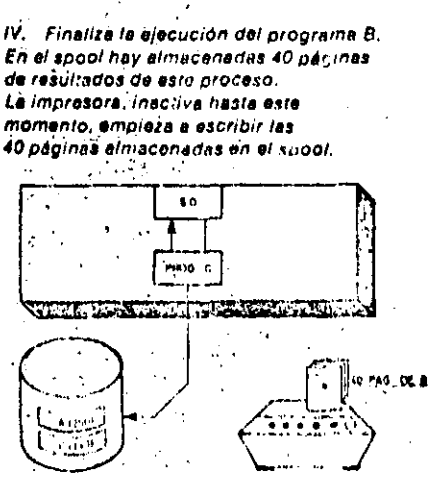
El aumento de capacidad de los ordenadores personales de 16 bits permite el acceso simultáneo a varios procesos, sin tiempos de espera demasiado grandes. Para ello el sistema operativo debe realizar algunas de las funciones típicas de los sistemas operativos de grandes ordenadores, tales como administrar prioridades, encargarse de los protocolos, de los niveles de acceso a los ficheros e interconexión de periféricos. Aunque Digital irrumpió en el área multiusuario con su versión MP/M-86, que es un sistema operativo demasiado básico, el liderazgo lo detentó rápidamente el sistema UNIX, que proporciona un excelente entorno para el



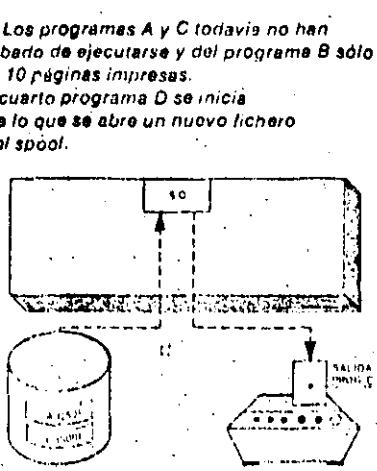
IV. Finaliza la ejecución del programa B. En el spool hay almacenadas 40 páginas de resultados de este proceso. La impresora, inactiva hasta este momento, empieza a escribir las 40 páginas almacenadas en el spool.



V. Los programas A y C todavía no han acabado de ejecutarse y del programa B sólo hay 10 páginas impresas. Un cuarto programa D se inicia para lo que se abre un nuevo fichero en el spool.



VIII. El programa B termina su impresión. Como el programa A está retenido, la impresora queda inactiva.



IX. El programa C termina de ejecutarse y comienza la impresión de sus resultados.

Conceptos básicos

El spool de impresión

Cuando un ordenador trabaja en multiprogramación, varios programas pueden requerir simultáneamente salida impresa. Una solución es disponer de varias impresoras. La otra consiste en retener los programas hasta que el primero que está utilizando la impresora termine su listado. De esta forma la pérdida de tiempo es considerable, ya que los listados suelen ser lentos, comparados con el tiempo de proceso. Este segundo problema lo resuelve el fichero «Spool» de impresión. Los listados no se envían directamente a la impresora; se van almacenando en disco y el sistema operativo los manda a la impresora cuando está libre, o a requerimiento del operador.

De esta forma los programas no quedan nunca retenidos: no es necesario que se acabe la impresión de los resultados de un programa para iniciar el proceso de otro.

Muchos programas pueden tener sus listados almacenados en disco. El sistema operativo los va colocando en una lista de espera, llamada cola del «Spool», según van acabando de ejecutarse los envía a la impresora en el orden de la cola, salvo indicación contraria del operador.

Otra ventaja del «Spool» es que los programas tardan menos tiempo en ejecutarse, porque escribir una línea en disco es bastante más rápido que imprimirla en papel. De todas formas el tiempo de impresión suele ser el de menor ocupación de la máquina. En grandes centros se acostumbra a lanzar el «Spool» en el turno de noche.

Una tercera ventaja del «Spool» es que el operador puede controlarlo directamente: se pueden lanzar tantas copias como se quiera, sin necesidad de repetir el proceso de cálculo; se puede destruir un listado de la cola de espera, sin necesidad de perder páginas impresas; se puede iniciar un listado en la página que se quiera; se puede igualmente controlar la página en que se quiera terminar la impresión, etc.

En resumen, el «Spool» ahorra impresoras y mejora el rendimiento de las instalaciones.

SISTEMAS OPERATIVOS PARA MICROPROCESADORES

desarrollo de programas multiusuario. Las principales facilidades del sistema UNIX son:

- Acceso controlado del usuario.
- Sistema de ficheros jerarquizado.
- Lenguaje de comandos seleccionable por la base de usuarios.
- Alto grado de compatibilidad.

Una de sus mejores características es el control de las funciones de escritura y/o lectura a ficheros mediante «password». El aspecto más importante es su lenguaje de comandos, llamado «shell», tan poderoso que puede ser considerado como un lenguaje de programación. Su potencia viene reforzada por la gran cantidad de rutinas de utilidad suministradas con el sistema. Los comandos del «shell» pueden almacenarse en ficheros, por lo que las tareas («jobs») específicas pueden ejecutarse mediante una sola instrucción. Sus principales debilidades son la falta de retroalimentación interactiva, la pobre consistencia de la sintaxis y de los nombres de los comandos, y el gran número de versiones diferentes existentes.

Dentro de este grupo de sistemas operativos multiusuarios para 16 bits destaca el sistema OASIS. Inicialmente fue desarrollado para el microprocesador Z-80, pero ha sido reescrito en lenguaje «C», por lo que es fácilmente incorporable a diferentes hardwares basados en procesadores de 16 bits. Sus mejores características aparecen en el manejo de ficheros. Estos pueden clasificarse como públicos, privados o de acceso compartido. Los métodos de acceso a los ficheros incluyen las organizaciones directas, secuencial, random y secuencial indexada.

Sistemas operativos para redes de microordenadores

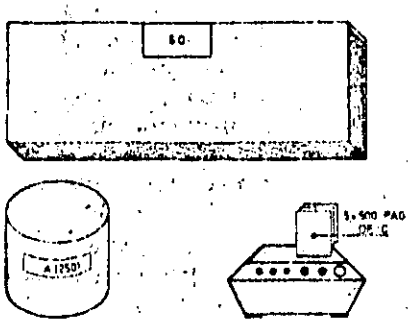
Una alternativa a los microordenadores multiusuario es la interconexión de ordenadores autónomos que permita el acceso compartido a las bases de datos e impresora. El funcionamiento de la red se apoya en un sistema operativo adecuado. El más importante hoy día es el TURBODOS de Software 2000.

En este sistema un procesador maestro se ocupa del manejo de todos los procesos de discos e impresoras, y los microprocesadores esclavos ejecutan los programas de aplicación.

Las principales características del Turbodos son:

- Cada procesador maestro soporta hasta 16 procesadores esclavos.
- Puede gestionar hasta 16 unidades de discos.
- Spool para 16 impresoras con múltiples colas de espera.
- Compatible con el CP/M2.2.
- Sistemas de correo rudimentario.
- Password de seguridad.

Existe una versión monousuario que ofrece un buen rendimiento en el manejo de discos y en el recubrimiento de errores. Su principal defecto es estar escrito en código Z-80. Software 2000 está preparando, sin embargo, una versión para microordenadores de 16 bits.



X. La impresora ha obtenido cinco copias del proceso C. En el spool están almacenados los resultados de salida del programa A, en espera de las instrucciones del operador.

PRINCIPALES SISTEMAS OPERATIVOS PARA MICROPROCESADORES

COMPAÑIA	SISTEMA OPERATIVO	MONOUSUARIO		MULTIUSUARIO	
		8 bits	16 bits	8 bits	16 bits
Digital Research	CP/M	X			
	CP/M-86		X		
	MP/M			X	
	MP/M-86				X
Microsoft	MS-DOS		X		
	XENIX				X
Sof Tech Microsystems	UCSP	X	X		
Phase One	OASIS			X	X
Software 2000	TURBODOS	X		X	
AT & T	UNIX				X
Interactive Systems	UNIX				X
Intel	IRMX-86				X

SISTEMAS OPERATIVOS PARA LOS PRINCIPALES MODELOS DE MICROPROCESADORES

FABRICANTE	MODELO	MICROPROCESADOR	SISTEMAS OPERATIVOS
ALTOS	586	8086	XENIX, MS-DOS, CP/M-86, MP/M-86, OASIS, UNIX
	ACS68000	68000	
DEC	Rainbow	8080/Z80	CP/M-86, MS-DOS
FORTUNE	32:16	68000	XEXIX
IBM	PC	8088	PC-DOS, CP/M-86, MS-DOS UCSD
NCR	DM-V	8088/Z80A	CP/M, CP/M-86, MS-DOS
OLIVETTI	M20	Z8001	PCOS, CP/M-86
PHILIPS	P2500	Z80A	CP/M, UCSD
	P3500	Z80AS	TURBODOS
TANDY	TRS-16	68000	UNIOS, TRS-DOS
VICTORY SYSTEMS	Spirit	80186	CP/M-86, MP/M-86
	Factor	68000	UNIX

AYUDAS AL PROCESO DE PROGRAMAS

Al alejarnos de la máquina, programando con lenguajes de alto nivel, se hace patente la necesidad de contar con programas especializados que faciliten y apoyen la tarea del programador. Estos programas forman parte del software del sistema y suelen ser suministrados por el propio fabricante del ordenador.

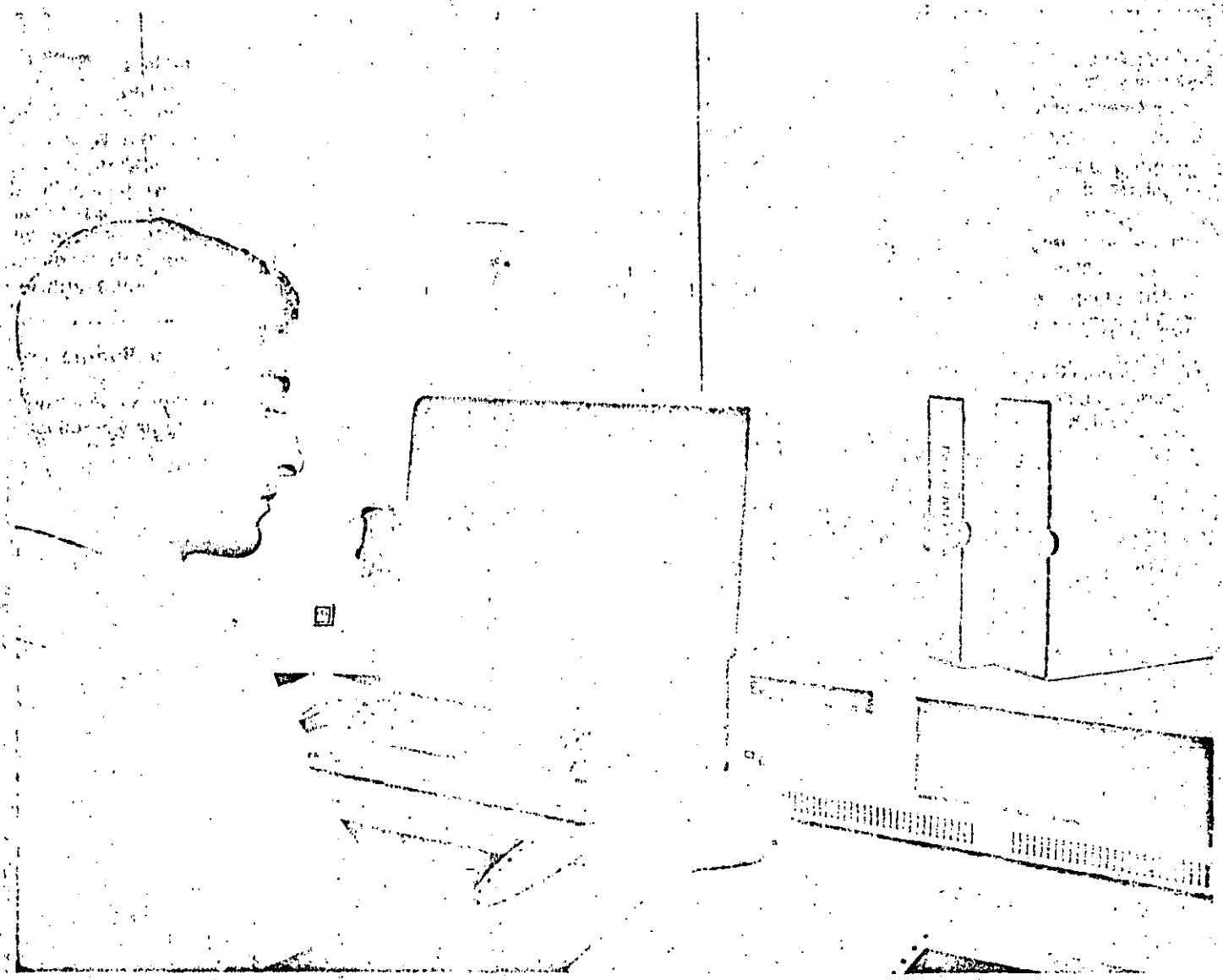
Los compiladores

El incremento del uso de las macroinstrucciones y su constante sofisticación hizo que en los programas confeccionados para distintos equipos se encontraran muchas funciones comunes, tales como leer datos de un fichero en disco o escribir en una cinta magnética. El análisis de estas funciones comunes llevó al desarrollo de los lenguajes de alto nivel.

Para que los programas escritos en estos lenguajes puedan ser ejecutados por el ordenador, es preciso convertirlos previamente en programas objeto, representados en lenguaje máquina. Este proceso de conversión recibe el nombre de compilado o *compilación* del programa fuente.

El *compilador* es el programa auxiliar que controla el proceso de *compilación*, realizando las siguientes funciones:

- Leer las instrucciones del programa fuente, a través de un periférico de entrada.
- Clasificarlas por número de secuencia de las instrucciones.
- Convertir las macro y micro instrucciones a instrucciones en código de máquina.
- Crear la tabla de direcciones de memoria de las referencias (variables, subrutinas, áreas de datos).



Para que la programación en lenguaje de alto nivel sea eficaz es preciso contar con determinados programas auxiliares que trasladen la información de origen a un lenguaje inteligible por el ordenador

AYUDAS AL PROCESO DE PROGRAMAS

- Producir el programa objeto en soporte perforado o magnético.
- Editar un listado, tanto del programa fuente como del objeto.
- Detectar los errores sintácticos del programa.

El proceso de compilación se repite hasta que se obtenga la llamada *compilación limpia*, es decir, una compilación exenta de errores.

Cada lenguaje de alto nivel necesita un compilador para cada tipo de ordenador en que vaya a ser procesado, ya que los respectivos lenguajes de máquina son distintos.

Intérpretes

El principal inconveniente, asociado al empleo de los compiladores, es que para ejecutar el programa es preciso compilarlo previamente, es decir, se trata de un proceso «batch».

Los intérpretes resuelven este problema, ya que traducen, «interpretan» y procesan las instrucciones según se van introduciendo; como resultado de esta ejecución van almacenando datos y visualizando los resultados progresivamente.

Generadores de programas

Existen procesos cuya lógica se puede repetir con frecuencia dentro de los programas, como son la clasificación de registros, la visualización de informes, el formateado de resultados, etc. Para evitar tener que reprogramar constantemente este tipo de procesos, existen los denominados *generadores*. Un *generador* es un programa capaz de construir otros programas, utilizando *parámetros* que dependen de cada caso particular. Por ejemplo: a un programa generador de clasificaciones

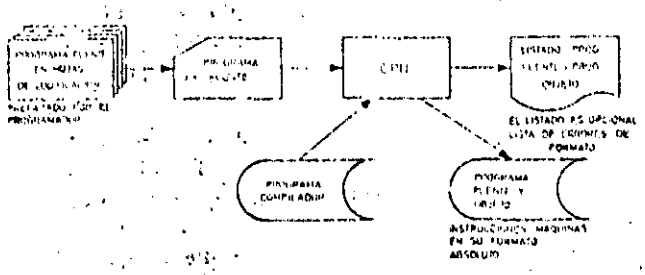
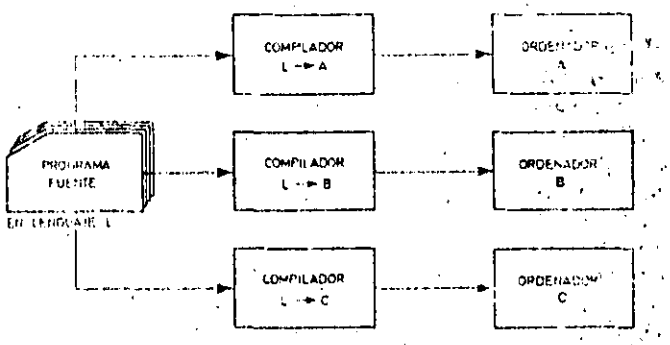
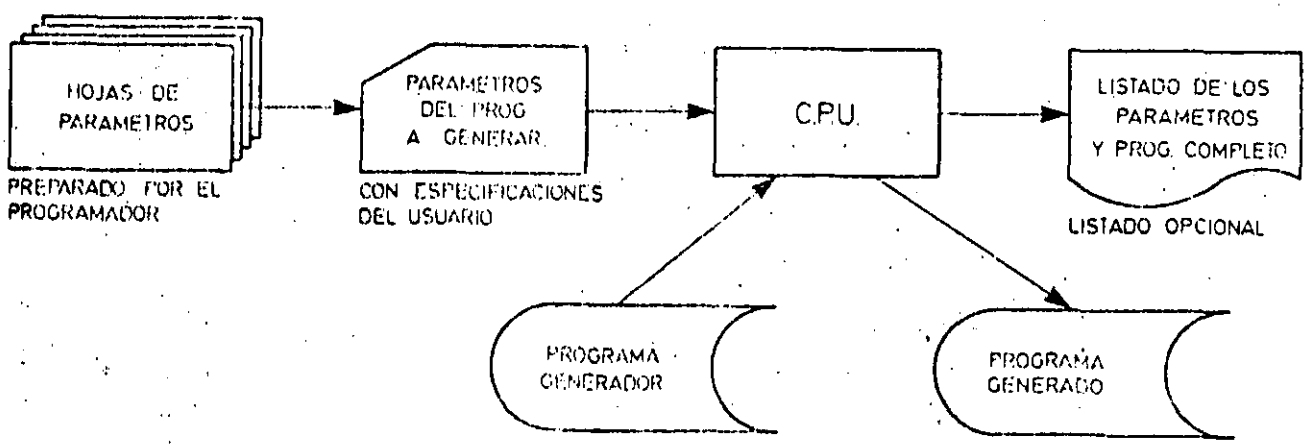


Diagrama representativo de un proceso de compilación.



Para que un programa pueda ser procesado en distintos ordenadores es necesario contar con compiladores especializados en cada una de las máquinas.



El proceso de generación de un programa es semejante al de «compilación»; no obstante la información de entrada no será un programa, sino únicamente los parámetros definitorios del programa a generar.

sólo es necesario darle parámetros, tales como: dónde se encuentran las claves de clasificación dentro del registro y cuál es su longitud, nombre del archivo, tamaño del bloque, longitud del registro, etc.

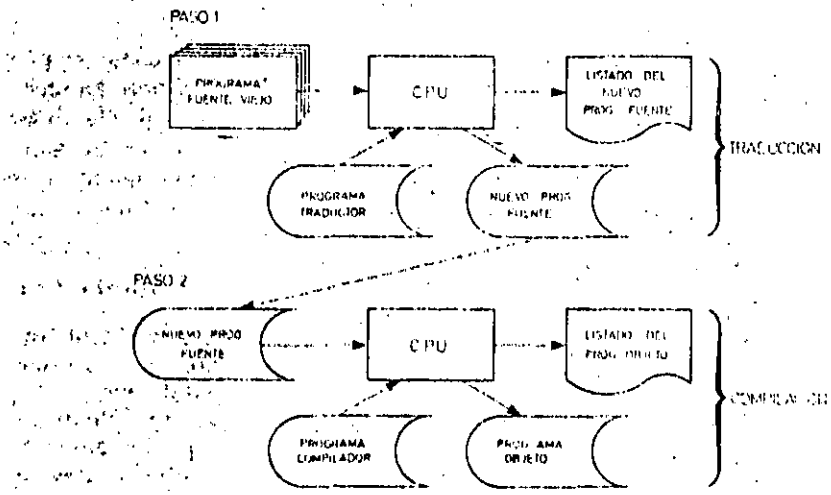
Traductores

Otro inconveniente de los programas escritos en lenguaje de alto nivel es el que supone su traslado a otro ordenador, dentro de cuyo software no existe un compilador para el lenguaje en el que están escritos nuestros programas.

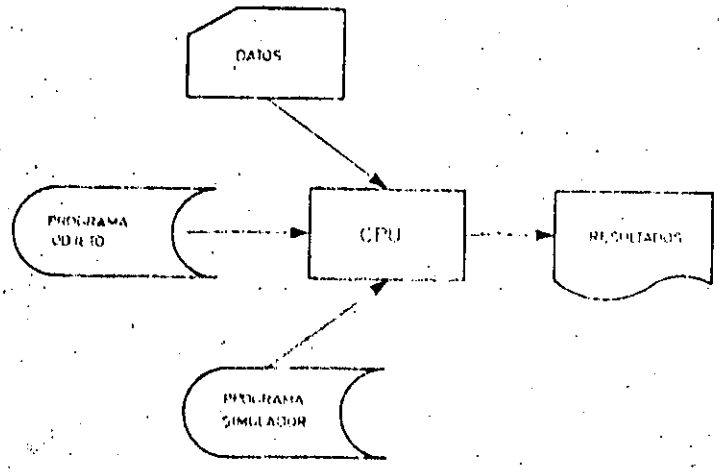
El reprogramar todas las aplicaciones puede ser no sólo muy costoso, sino prácticamente inviable.

Los programas *traductores* convierten las instrucciones fuente de un lenguaje en las equivalentes instrucciones fuente de un segundo lenguaje. Este nuevo programa fuente puede ser compilado.

El uso de los traductores reduce el tiempo necesario y el coste de la puesta en marcha de un nuevo ordenador. No se pueden usar, si durante el mantenimiento de los programas antiguos se han realizado «parches» (patching), a no ser que las modificaciones se hubie-



Los programas traductores convierten un programa fuente en otro programa fuente que, posteriormente, debe ser sometido a un proceso de «compilación».



Para que sea posible utilizar los programas en un nuevo ordenador distinto del original, puede recurrirse a los programas auxiliares denominados «simuladores».

Glosario

¿Por qué se llama compilador?

Porque una de las técnicas empleadas en el análisis lexicográfico y sintáctico es la de pilas.

¿Cuál es la diferencia entre un traductor y un compilador?

El que el traductor (en inglés «translator») convierte un programa fuente escrito en un determinado lenguaje en otro programa fuente en distinto lenguaje. Por su parte, el compilador convierte un programa fuente en un programa objeto en código de máquina.

¿Qué es un «parche»?

Cuando es necesario hacer alguna modificación en un programa compilado se suelen introducir las instrucciones oportunas en el código de máquina correspondiente. Se dice que el programa se ha «parcheado». Por ello hay que tener cuidado, puesto que el programa objeto ya no es la traducción correcta del programa fuente original.

¿El RPG es un lenguaje o un generador?

RPG son las iniciales Report Program Generator y, originalmente, se diseñó para proceso de salida de informes impresos. El programador usaba hojas específicas en las que definía la entrada y la salida. Posteriormente, se expandió hasta convertirse en un verdadero lenguaje de programación, que permite aplicarlo a problemas complejos.

AYUDAS AL PROCESO DE PROGRAMAS

ran introducido también en el programa fuente.

Simuladores

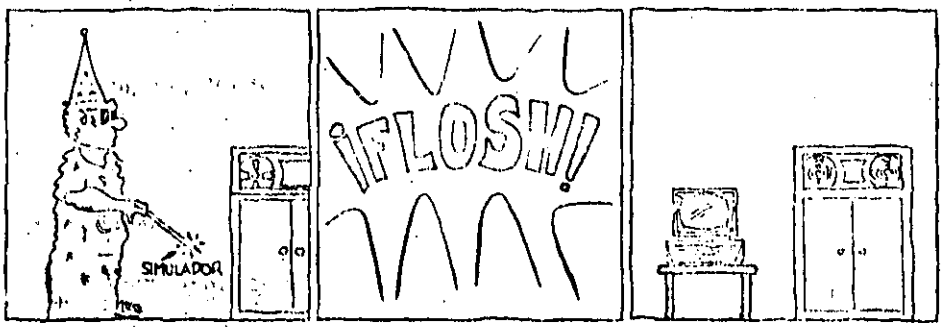
Otra solución para seguir utilizando los mismos programas en un nuevo ordenador, es el uso de programas *simuladores*.

Un programa *simulador* logra que un ordenador actúe como si fuera otro distinto. La recepción, tratamiento y salida de datos aparentemente es igual que con el ordenador simulado.

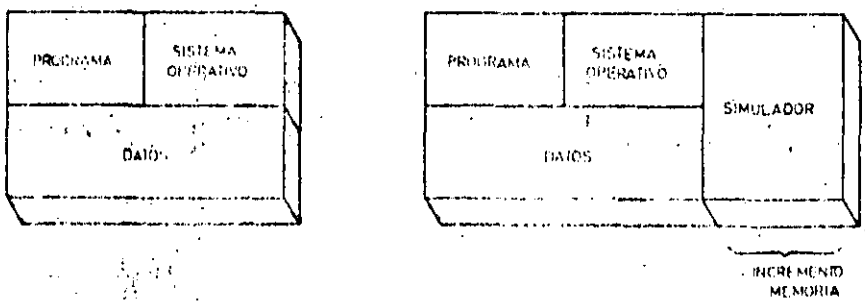
A diferencia del programa traductor (que trabaja con el programa fuente), el simulador opera con el programa objeto.

El principal inconveniente de los programas simuladores es que aumentan mucho el tiempo de proceso y necesitan más memoria, ya que el programa simulador debe permanecer en memoria junto con el programa objeto que va a ser procesado.

De todas formas, estos inconvenientes pueden ser resueltos con soluciones «firmware», que pueden convertir a los ordenadores en «máquinas virtuales».



Un programa simulador es capaz de lograr que un determinado ordenador se convierta virtualmente en otro equipo distinto.



Ocupación de la memoria de un ordenador sin y con la presencia de un programa simulador

Conceptos básicos

Máquina virtual

El concepto de firmware nos sirve como elemento de introducción a la máquina virtual. Recordemos que el firmware se define como un conjunto de microprogramas. Cambiando el firmware de un ordenador, cambian realmente las características del mismo. El firmware hace posible transformar, por ejemplo, un ordenador de gestión en uno de tipo científico. Para realizar esta transformación, lo único que hay que hacer es cargar, en el ordenador de gestión, un firmware que posea características propias de un ordenador científico, como puede ser la de operar en coma decimal flotante, y sin necesidad de incorporar nuevos elementos hardware se consigue que el ordenador opere con un gran rendimiento, tanto en el aspecto comercial como en el científico.

La introducción del firmware oportuno hace que un ordenador compile y ejecute un programa con mayor rapidez, utilice menos memoria interna. Hasta ahora, al adquirir un nuevo ordenador, el usuario se veía obligado a modificar los programas fuente para pasarlos al lenguaje del nuevo ordenador. Un firmware apropiado permite que los programas objeto existentes puedan ser ejecutados sin recompilarlos y el nuevo ordenador actuará como si fuera el antiguo; esto es, opera VIRTUALMENTE de la misma manera que el ordenador antiguo. Tenemos una máquina virtual. Los programas fuente escritos en un lenguaje como el COBOL pueden compilarse y ejecutarse en una máquina virtual COBOL, que actúa como si se hubiera diseñado para cumplir los requisitos del COBOL.

De lo dicho hasta ahora se saca una conclusión importante para el mundo informático. Una máquina puede transformarse en distintas máquinas virtuales a medida que el usuario tenga necesidad de ello, cargando diferentes firmwares. Puede hacer trabajar a su ordenador como una máquina virtual COBOL, o como una máquina virtual FORTRAN, e incluso como una máquina de proceso de comunicaciones conectándole terminales. Todo lo dicho depende fundamentalmente de los distintos firmwares que suministren los fabricantes.

Una ventaja muy importante para el usuario de un ordenador con firmware es que puede ampliar o cambiar su ordenador, convirtiéndolo en un sistema más potente con un coste y un esfuerzo mínimos, ya que no pierde la inversión efectuada en los programas del sistema antiguo.

SOFTWARE

FASES DE UN PROYECTO INFORMATICO

La mecanización mediante ordenador de trabajos complejos, como la contabilidad o la nómina de una empresa, implica la realización de numerosas tareas que comienzan en el momento en que se toma la decisión de iniciar el proyecto y acaban en el instante en que los programas están funcionando en el ordenador.

Estas tareas suelen agruparse en fases o etapas que permiten organizar la secuencia de actividades de todo el personal que interviene en el proyecto.

Casi todos los proyectos informáticos comprenden las siguientes fases de realización:

- Fase de estudio de la viabilidad.
- Fase de análisis del problema.
- Fase de diseño.
- Fase de programación.
- Fase de instalación.

Fase de estudio de la viabilidad

En esta primera fase se estudia si el problema o trabajo, origen del proyecto puede ser adaptado o no al ordenador. Luego se analiza el costo, y se establece el número de recursos (económicos, de tiempo y personal) necesarios para su realización.

Algunos factores dignos de estudio en esta fase son:

- Ventajas o inconvenientes del proyecto.
- Efectos producidos.
- Personal, tiempo y costos implicados.

Todos estos puntos son tratados por el analista del sistema en compañía del usuario que ha solicitado la ejecución del proyecto. Esta fase es típica de cualquier toma de decisión en nuestra actividad normal. Nadie se embarca en ningún proyecto sin un estudio previo de su factibilidad.

Fase de análisis

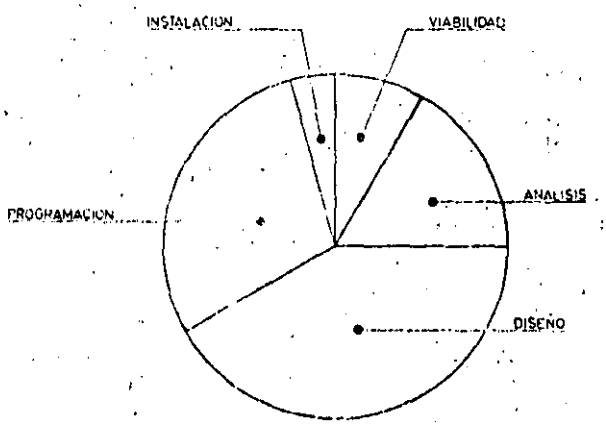
Esta fase comienza cuando se toma la decisión de aceptar el proyecto. Está dirigida por el analista, que estudia, de forma detallada las informaciones y datos que recibe del usuario y establece cuáles se pueden eliminar y cuáles se deben utilizar en el proyecto. El analista utiliza diversas técnicas para la recogida de esta información y su posterior análisis. Una de las técnicas más usuales para la recogida de información es la entrevista con todas las personas que vayan a utilizar el proyecto. Estos usuarios pueden aportar sugerencias y requisitos necesarios para un mejor desarrollo. Otra forma de recoger información son los cuestionarios, la investigación personal y la observación del proceso manual.

Esta fase se realiza también en todas las actividades profesionales. Una vez recogidos los datos se procede a un análisis detallado de los mismos, tanto de forma cuantitativa como cualitativa. Antes de empezar la fase de diseño, el analista y el usuario, o su representante, estudian los resultados obtenidos y se toman decisiones como: continuar con el proyecto, cambiar algunos de los objetivos del mismo, cancelar el proyecto, etc.

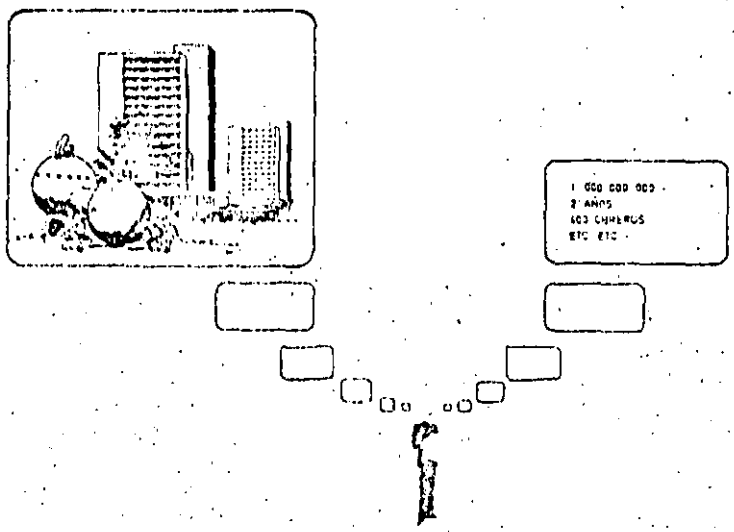
Fase de diseño

En esta fase interviene una nueva persona encargada de continuar el proyecto, es el *diseñador de sistemas*, que se encarga de buscar el tipo de estructuras de los programas o módulos más apropiados para el caso.

La forma ideal de trabajo de un diseñador de sistemas es comenzar el proyecto conociendo los resultados que



La duración de cada una de las fases de elaboración de un proyecto informático son muy desiguales. Como se muestra en la figura, las fases de diseño y de programación son las más largas.



El análisis previo o estudio de la viabilidad es fundamental en la elaboración de un proyecto de cualquier tipo. Un constructor que desee edificar una casa tiene que estudiar primero los costos de realización, el número de trabajadores que serán necesarios, etc.

FASES DE UN PROYECTO INFORMATICO

Glosario

¿Existen otros métodos de organización de un proyecto informático?

Sí, existen otros métodos de organización, entre los que destacan el método organizacional y el funcional. Pero, básicamente, lo importante es que haya una metodología que facilite el trabajo de todos los que intervienen en un proyecto.

¿Qué significado tiene la palabra «DEBUGGING»?

Es un vocablo inglés que se aplica a la depuración de los programas durante la fase de programación.

¿Qué diferencia existen entre el analista del sistema y el diseñador del sistema?

El diseñador del sistema recoge la documentación almacenada por el analista del sistema durante las fases de viabilidad y de análisis para incorporarla al sistema informático que está proyectado. Es el auténtico «especialista» que hace posible que el proyecto se procese por ordenador.

¿Qué es lo que se busca, fundamentalmente, al realizar un proyecto informático?

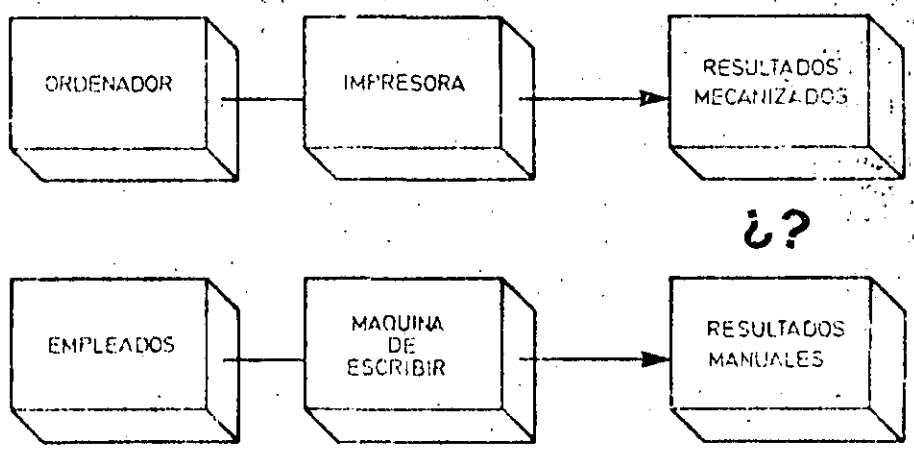
La sustitución de una forma de trabajar manual por un nuevo método mecanizado.
Cuando existe una mecanización previa se busca mejorar el rendimiento del sistema antiguo o la sustitución de un hardware y/o un software anticuado por otro más moderno.

quiere obtener. A continuación establece los procedimientos necesarios para procesar estos resultados y qué datos de entrada tiene que introducir en ese proceso.

Supongamos, como ejemplo, que una empresa necesita automatizar el pago de la Seguridad Social de sus empleados. El analista de sistema sabe que tiene que obtener un listado con el importe que cada empleado tiene que pagar a la Seguridad Social. Para llegar hasta esto necesita una serie de datos correspondientes a cada uno de los empleados. Datos típicos en este ejemplo son, si el empleado es soltero o casado, los hijos que tiene, si trabaja su mujer, etc., ya que, de acuerdo con éstos datos, la cotización a la Seguridad Social es mayor o menor. A partir de ellos busca las fórmulas y cálculos necesarios para producir la información de salida buscada.

Una vez que el analista conoce estos datos tiene que estructurarlos adecuadamente, de manera que se cumpla el siguiente postulado de la informática, «el sistema perfecto es aquel en el que se toman las medidas adecuadas, sobre datos correctos para obtener resultados necesarios en el momento oportuno.

Para lograr este objetivo el diseñador utiliza también el llamado «análisis jerárquico», que consiste en separar el problema en sus partes componentes.



Los datos necesarios para el análisis de un proyecto informático son de dos tipos, cuantitativos, como los datos de los recibos o el volumen de los mismos, y cualitativos. Estos últimos datos son, por ejemplo, el organigrama de funcionamiento interno de una empresa que se quiera automatizar.

En el ejemplo que hemos visto, el reparto de jerarquías tendría dos niveles. En el nivel superior se encuentra la obtención de listados de cotización de la Seguridad Social, y en el segundo, los datos de entrada y proceso. Los datos de entrada se pueden jerarquizar, a su vez, de acuerdo con su importancia. Lo mismo ocurre con el proceso, es decir, se puede dividir el proyecto de forma que aparezcan todas las variables que participan en él.

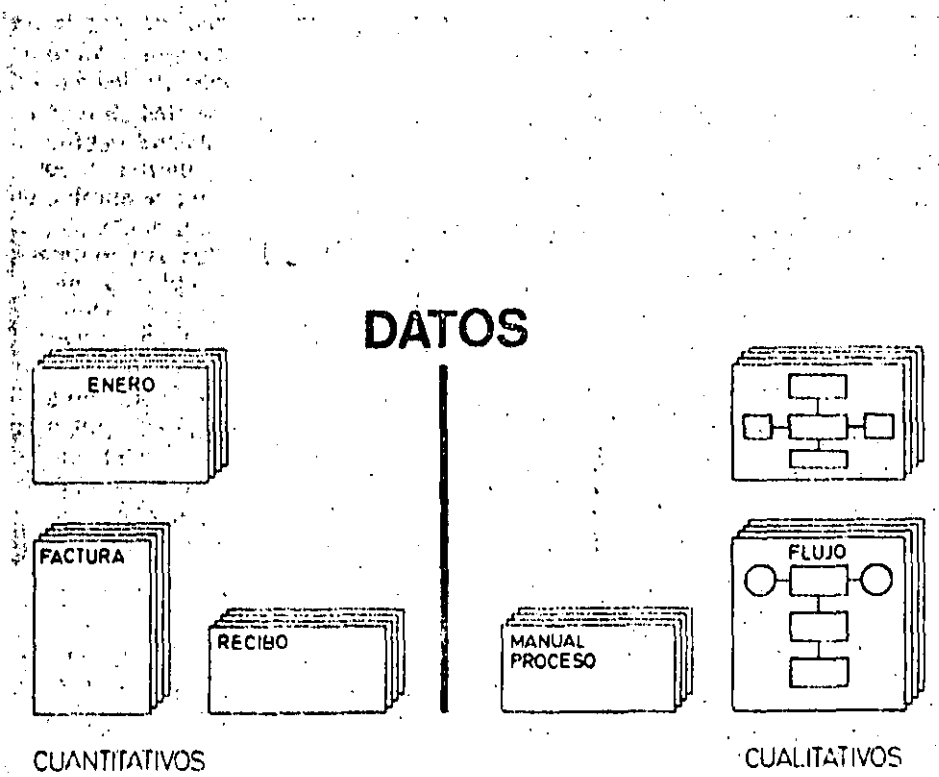
Una vez que se ha establecido la jerarquía de todos los componentes del sistema hay que «juntar» todas las piezas adecuadamente, creando un diagrama de flujo que enlace a todos los componentes del sistema.

Cuando acaba esta fase de diseño hay que realizar, junto con el analista de las dos primeras fases, una comprobación y ver si el sistema cumple con todas las especificaciones necesarias. Esta comprobación puede llevar a conclusiones como continuar con la siguiente fase, cambiar algunas especificaciones, abandonar el proyecto, etc.

Fase de programación

En esta fase interviene el programador que codifica el programa para luego pasarlo a un medio de entrada, como tarjeta perforada, cinta magnética, terminal de entrada, etc.

Una vez compilados los programas, se comienza una etapa de depuración,



Para probar el buen funcionamiento de un sistema informático se lo hace trabajar, simultáneamente y durante algún tiempo con el sistema antiguo, para verificar los resultados obtenidos de ambas formas. A esto se le llama prueba en paralelo.

Conceptos básicos

Documentación y personal de un proyecto informático

En la fase de estudio de la viabilidad intervienen el analista y el usuario del proyecto. El usuario presenta el problema y el analista investiga dicho problema de acuerdo con el usuario. Necesitan crear una documentación donde se reflejen las ventajas o desventajas del proyecto, sus efectos, personal, tiempo y costos implicados, etc.

En la fase de análisis participa, de nuevo, el analista para investigar a fondo el problema y establecer lo que realmente quiere realizar. El analista se documenta utilizando formularios e informes de los sistemas que se van a cambiar, los rendimientos de los mismos, etc. Tanta cantidad de información acarreada en esta fase debe ser organizada adecuadamente. Existen diversos sistemas que simplifican la documentación, y que comprende básicamente cinco tipos de formularios correspondientes a la definición de la salida del ordenador, definición de la entrada de datos, definición de los cálculos del proceso, definición de procesos lógicos y definición de archivos. El analista puede utilizar los diagramas analíticos. Cuando el analista y el usuario llegan a un acuerdo sobre las especificaciones del nuevo sistema, éstas se recogen en el formulario o informe de las especificaciones de requisitos. En esta fase el analista actúa como intermediario entre el usuario y el diseñador.

En la fase de diseño interviene el diseñador de sistemas (que a veces es el propio analista), que estructura de forma adecuada los programas. Su labor comienza con el informe de especificaciones de requisitos. Se ayuda de las organizaciones jerárquicas y de los diagramas de flujo para conseguir sus objetivos. Al final proporciona al programador el llamado cuaderno de carga de programas con todos los datos que se necesitan.

En la fase de programación interviene el programador asesorado por el diseñador del sistema. Codifica y prueba los programas con ayuda del personal que introduce los datos. Las pruebas de consolidación las realiza el programador junto con el analista y el diseñador del sistema.

En esta fase el diseñador y el analista preparan la documentación para los usuarios, donde aparecen los medios de archivo, la documentación de las pruebas realizadas, etc.

FASES DE UN PROYECTO INFORMATICO

que consiste en probar los programas y corregir los posibles errores. Los programas se prueban individualmente y se combinan luego en grupos cada vez más complejos. Estos grupos se prueban también conjuntamente. A este fenómeno de agrupamiento de programas se le conoce como «Consolidación».
Finalmente se prueba el sistema, emulando el funcionamiento real. Si es correcto, se pasa a la fase de instalación. Si no es correcto hay que intentar solucionar los posibles fallos que se detecten.

Fase de instalación

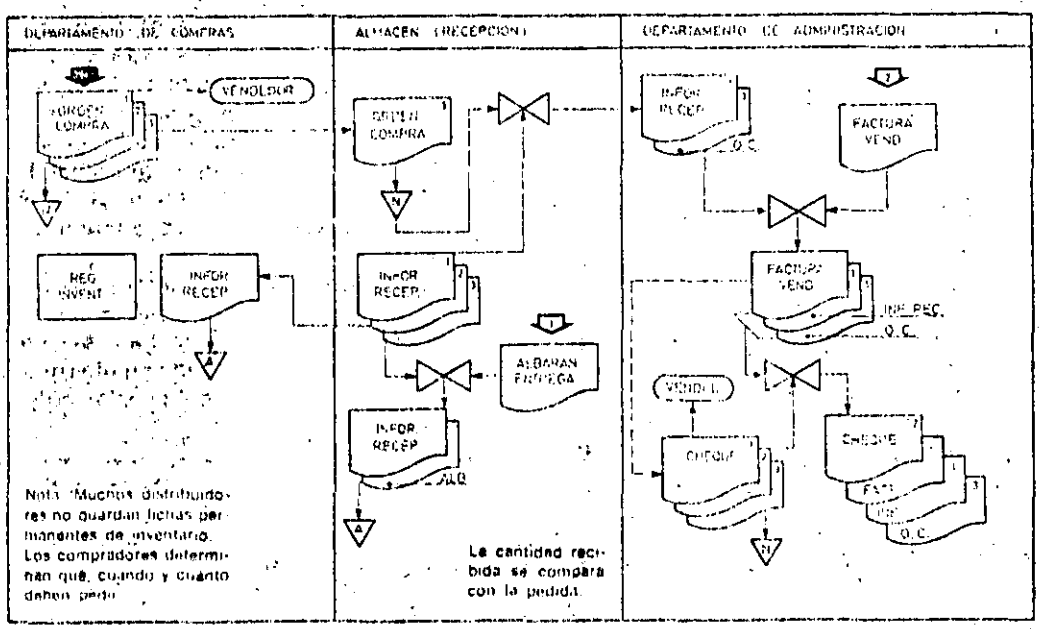
El nuevo sistema se instala para utilizarlo, durante un tiempo, «en paralelo» con el antiguo, lo que permite la comprobación de los resultados obtenidos.

La «prueba en paralelo» es la única que garantiza que todos los casos reales, tanto generales como particulares, son procesables correctamente por el ordenador.

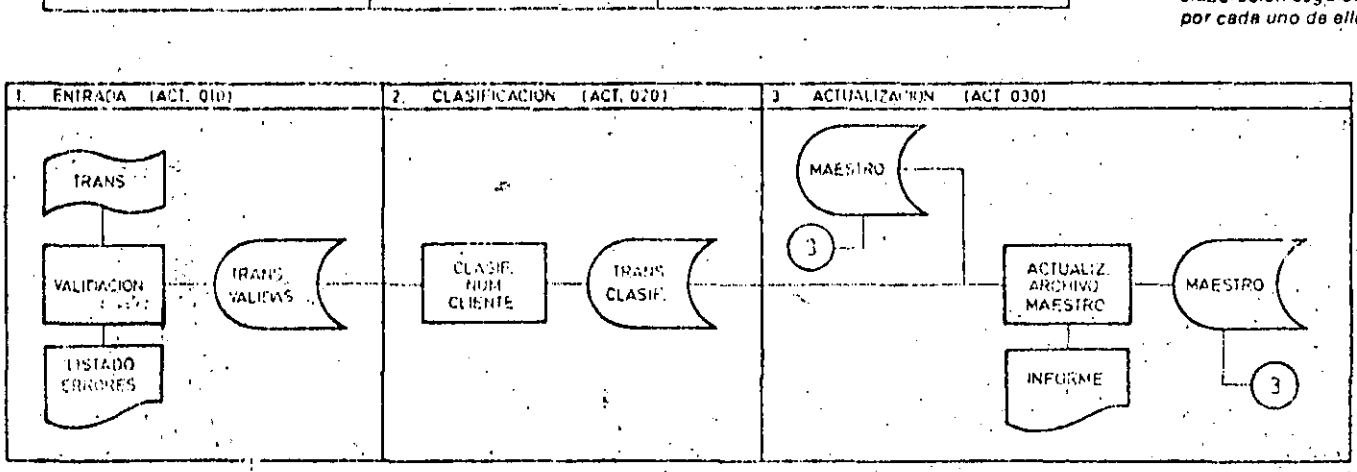
A fin de facilitar a los usuarios completa información de las posibilidades del nuevo sistema, y de la forma en que éste va a afectar a su trabajo cotidiano, la empresa, o grupo que ha realizado el proyecto recicla al personal que va a manejarlo. Organiza para ello cursos que permitan al personal usuario comprender y resolver cualquier duda que se le presente en el manejo del nuevo sistema.

Una vez instalado, el sistema comienza la fase de explotación normal del proyecto. Los usuarios deberán recibir una documentación muy completa sobre el sistema, que incluye manuales de operación, diagramas del sistema y el correspondiente «dossier» de análisis y programación para facilitar correcciones y revisiones.

El proyecto debe ser revisado periódicamente para incorporar, si es necesario, algunos cambios o comprobar que está cumpliendo perfectamente la función encomendada en el momento de su realización.



Un documento importante para el análisis de un proyecto es el diagrama analítico de los documentos, que muestra el proceso de elaboración seguido por cada uno de ellos.



Los diagramas del sistema muestran la forma en que deben emplearse los archivos y las operaciones a realizar por el programa que se está diseñando.

SOFTWARE

PROGRAMACION ESTRUCTURADA

LA experiencia ha demostrado que un programa que presente dificultades para ser modificado está condenado a la «muerte informática». Debe procurarse, por tanto, que los programas sean a la vez flexibles y transportables: flexibles para que se adapten con facilidad a cualquier cambio; transportables de forma que cualquier nuevo proceso pueda utilizar sus subrutinas sin introducir grandes cambios.
Conviene, para ello, emplear técnicas de programación que faciliten el desarrollo de software fácilmente modificable. De esta forma el programador que utilice el software desarrollado anteriormente no tendrá que efectuar dos tareas muy tediosas:

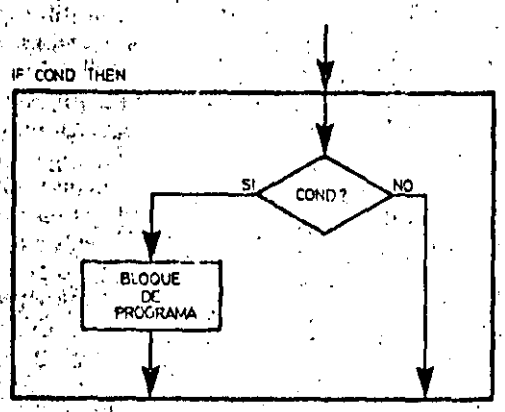
- Escribir partes del programa ya escritas.
- Probar subrutinas ya probadas.

Programación modular, programación estructurada

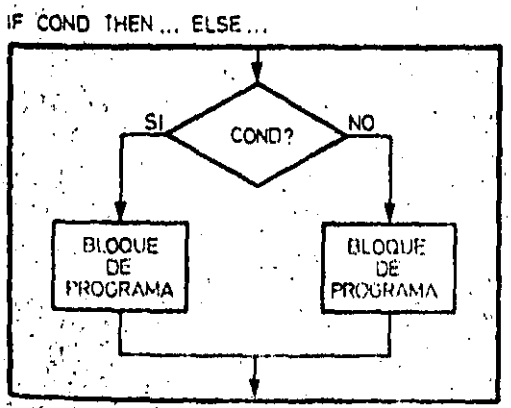
Todas estas consideraciones se acercan a la idea de programación modular: cada problema debe descomponerse en una serie de problemas más pequeños hasta llegar a un nivel en que cada uno de ellos no pueda reducirse más. En ese momento se ha llegado al nivel más bajo del análisis. Es entonces cuando realmente se puede resolver el problema planteado al principio. Cada uno de estos problemas mínimos realiza una sola función: de esta forma un problema de orden superior puede usar, para su resolución, «problemas mínimos», comunes a varios niveles. Una vez demostrada la necesidad de descomponer un problema general en «problemas mínimos», resulta obvio que éstos no son sino los módulos de que consta el programa. Se está, de esta forma, haciendo a la vez programación modular y programación estructurada: el software obtenido es modular, mientras que las técnicas empleadas para desarrollarlo son estructuradas.

Características de un programa estructurado

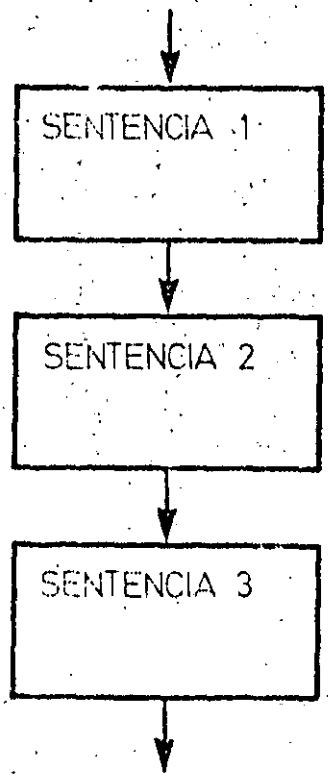
El decarrollo de programas modulares requiere un soporte software adecuado: el grado de modularidad obtenido depende del intérprete o del compilador empleados. En este sentido, resulta muy útil contar con instrucciones flexibles de ejecución de subprogramas o módulos.



Bifurcación utilizada en programación estructurada. Si la condición se cumple se ejecuta el siguiente bloque, si no es así, el programa continúa de forma lineal.



Bifurcación del tipo «Si... Entonces... Sino...». Se cumpliría la condición se ejecuta el bloque de la izquierda, en caso contrario, el de la derecha.



Bloque lineal. La principal característica de este tipo de estructuras es que contiene sólo una entrada y una única salida.

PROGRAMACION ESTRUCTURADA

Para conseguir que los programas sean transportables es necesario programar en base a módulos de pequeño tamaño, cada uno de los cuales debe facilitar toda la documentación posible sobre su funcionamiento. Es preciso que con un simple vistazo al listado de cada módulo cualquier programador comprenda su funcionamiento. Esto facilita, además, cualquier modificación posible.

Por otra parte, los algoritmos de un programa estructurado deben ser muy sencillos. Es preferible utilizar varias instrucciones separadas y visibles que una sola, con muchos niveles de paréntesis, operaciones complicadas, etc. Una sentencia con cinco o seis instrucciones de tratamiento de cadenas, o con varias funciones definidas por el usuario, puede provocar el desconcierto en cualquier programador que intente comprender su funcionamiento. Otro factor muy importante que determina la legibilidad de un módulo es la linealidad de la secuencia de sus instrucciones: en este sentido no resulta aconsejable el uso de sentencias «GOTO», pues cada vez que un programador se encuentra con una de ellas tiene que reconstruir mentalmente el organigrama del programa.

Si los módulos de un programa estructurado están bien construidos, cada uno de ellos ejecutará una sola tarea, y no efectuará ningún tipo de saltos a puntos alejados del programa.

Tipos de sentencias de un programa estructurado

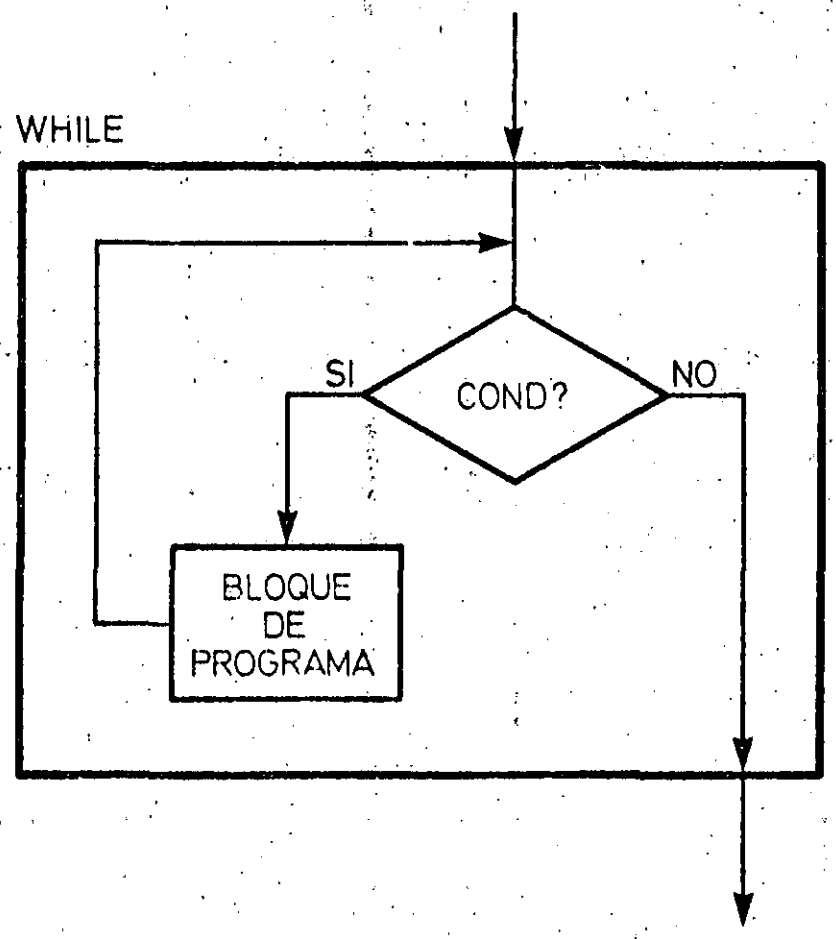
Varios autores han demostrado que cualquier programa estructurado puede construirse por medio de tres tipos básicos de estructuras. Estas son:

- Secuencia lineal.
- Bifurcación.
- Repetición.

Las sentencias lineales son las más comunes en un programa estructurado. Representan una operación o acción ejecutada dentro del programa.

La bifurcación es la operación por la cual el ordenador escoge la acción a ejecutar dentro de un conjunto de posibilidades. Esta elección está determinada por el valor que tomen determinadas variables, calculadas anteriormente por medio de sentencias lineales.

Cuando se repiten varias operaciones hasta que una variable cualquiera tome

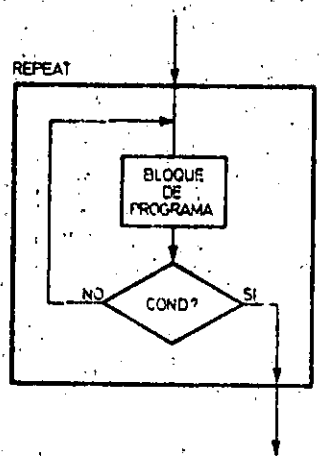


Los bloques de repetición pueden ser de dos tipos. Cuando la comprobación de la variable de condición se efectúa antes de ejecutar el bloque de programa correspondiente, se llama bloque While (Mientras).

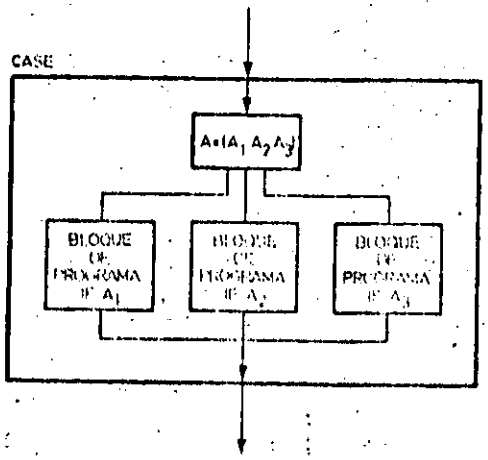
un valor determinado, se está ejecutando una sentencia de repetición.
Aunque con estos tres tipos de sentencias se puede construir cualquier tipo de programa estructurado, no se garantiza su legibilidad. Para asegurarla se crean secuencias lineales independientes. Cuando un bloque de instrucciones se maneja como si fuera una única instrucción se ha creado un procedimiento. Esta, a su vez, puede constar de otros tantos bloques independientes.
Por otra parte, cada bloque de instrucciones utiliza variables que no deben ser siempre las mismas. Las variables de trabajo de un bloque son transferidas desde el bloque superior a través de los «argumentos» de entrada al procedimiento. Los resultados obtenidos pueden, a su vez, emplearse por otros procedimientos.
Las variables implicadas en un procedimiento pueden ser de uso exclusivo de este bloque, o compartirse con el módulo que los llama. Para asegurar la flexibilidad de los módulos cada una de las partes de que se compone debe ser autónoma y contener, por tanto, sus propias variables independientes. Las modificaciones sucesivas no presentarán, de esta forma, problemas, ya que cada variable se usa sólo en un módulo concreto.

Sentencias de bifurcación

Cuando un programa llega a una bifurcación decide, en función del valor que tome determinada variable, el procedimiento que debe ejecutar a continuación. La instrucción más común para



Un bloque de repetición es del tipo Repeat (Repetir), si finaliza por la comprobación de la variable de condición.



Las estructuras de tipo Case proporcionan una gran potencia de programación a los lenguajes estructurados. El programa ejecuta un conjunto de instrucciones diferentes para cada valor de la variable de condición A.

Conceptos básicos

Multiprogramación

Los ordenadores que controlan la ejecución simultánea de varios procesos requieren de una adecuada gestión de los recursos, tanto de software como hardware.
Cuando el ordenador trabaja en multiproceso el problema que debe abordar es el de la asignación de tiempo a cada uno de los procesos en curso. Para ello cada uno de los procesos puede estar en uno de estos tres estados: bloqueado, en espera o en ejecución. Un proceso está bloqueado cuando espera que ocurra algo para poder continuar su ejecución (normalmente que ocurra una entrada o una salida del sistema); un proceso está en espera cuando ya está listo para continuar su ejecución, y sólo precisa que se le dé el control del procesador central. Un proceso en ejecución puede pasar a cualquiera de los otros dos estados, mientras que al estado de ejecución sólo pueden pasar los procesos en espera.
La parte del sistema operativo que gestiona el paso de un estado a otro se llama «scheduler». Asigna para ello una prioridad a cada una de las tareas.
En ocasiones es el propio sistema quien confiere las prioridades, teniendo en cuenta para ello el tiempo que puede durar la ejecución de cada una de las tareas.
Este proceso de asignación de prioridades debe tender a equilibrar los recursos necesarios y disponibles y evitar que muchas tareas estén bloqueadas mientras que la CPU está parada por no tener tareas en espera.

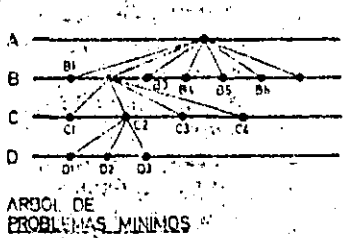
PROGRAMACION ESTRUCTURADA

realizar esta elección es CASE. El comando SELECT, a su vez, selecciona la variable de la que depende el procedimiento a llamar. Para cada uno de los valores —CASE— que tome esta variable se escoge un procedimiento u otro. Si sólo existen dos alternativas de elección posibles se usan sentencias del tipo «IF ... THEN ...» (si ... entonces ...), o del tipo «IF ... THEN ... ELSE ...» (si ... entonces ... sino ...).

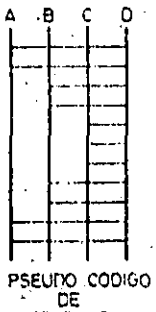
Las instrucciones de repetición pueden ser de varios tipos. En el primer caso el programa comienza por examinar la variable de condición. Según el valor de ésta pasa a ejecutar el procedimiento repetitivo o no. En el segundo tipo la operación a repetir se ejecuta al menos una vez, se cumpla o no la condición impuesta, pues la comprobación de la variable de condición se efectúa inmediatamente después de la operación de repetición.

Utilizando este tipo de estructuras, la labor del programador se reduce a aplicarlos a problemas reales. Puede comenzar su labor traduciendo los algoritmos a pseudocódigo compuesto de módulos, que se descomponen en procedimientos de nivel cada vez más bajo.

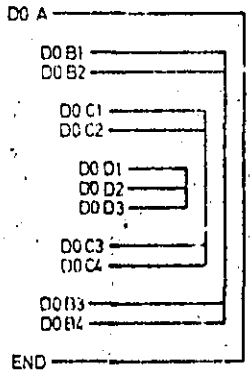
Cada uno de estos niveles permite a un programador, ajeno a la escritura del programa, su lectura, comprensión y posterior modificación.



ARBOL DE PROBLEMAS MINIMOS

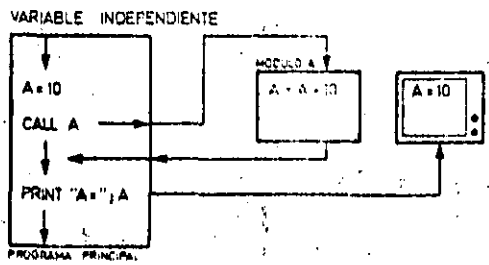


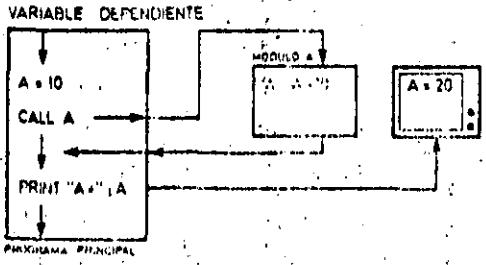
PSEUDO CODIGO DE PROBLEMAS MINIMOS



CODIFICACION DE PROBLEMAS MINIMOS

Cualquier problema general se puede descomponer en un conjunto de problemas mínimos. La figura ilustra, esquemáticamente, la generación, la realización del pseudocódigo correspondiente y la codificación de éstos.





Las variables independientes creadas en el programa principal conservan su valor a lo largo de éste. Las variables dependientes, sin embargo, pueden modificarse por operaciones efectuadas dentro de un módulo cualquiera.

Glosario

¿Es directamente ejecutable el pseudocódigo?

No. El pseudocódigo es un medio de representar la estructura interna de un programa. Tiene la ventaja de acercarse bastante al código final y de ser, además, fácil de leer y escribir.

¿Es suficiente que un programa no contenga instrucciones GOTO para que sea estructurado?

No es suficiente. La carencia de instrucciones GOTO es una característica muy importante de los programas estructurados, pero no es la única condición requerida para que lo sean. Otros factores, como el nivel de secciones independientes de que conste la codificación, son más representativos de este tipo de técnicas.

¿Cómo pueden ejecutarse programas estructurados en una máquina que no contenga la declaración Procedimientos?

Una de las formas posibles es codificar el programa en lenguaje máquina, cambiar las direcciones de principio y final del texto escrito en Basic, por ejemplo, y bifurcar a la posición de memoria reservada para el paso de variables.

¿Pueden programarse estructuralmente sin disponer de instrucciones apropiadas como CASE, LOOP, REPEAT, etc.?

Como todo programa complejo, puede reducirse a tres instrucciones básicas, siempre se puede simular cualquier instrucción de la que se carezca. El producto final será, quizá, poco legible, pero, en todo caso, estructurado.

SISTEMAS Y PROGRAMAS

ALGORITMOS

BASIC

Act. Ricardo Jerez

Octubre, 1984.

DESCRIPCION DE ALGORITMOS

Existen tres formas diferentes de describir un algoritmo:

- Oracional { Lenguaje Natural; Lenguaje Algorítmico }
- Esquemática { Diagrama de Flujo }
- Mixta { Diagrama Warnier }

Oracional: El lenguaje natural requiere el escribir el algoritmo utilizando oraciones del lenguaje común, aunque muchas veces se presente, la ambiguedad, su retórica, el estilo literario, etc.

Por otra parte, el lenguaje algorítmico utiliza oraciones sencillas, relacionadas con figuras de lógica fácilmente reconosibles. Un algoritmo descrito de esta forma debe contener las siguientes convenciones:

- Algoritmo(nombre)	Nombre del algoritmo
- Entrada(parámetros...)	Parámetros de entrada
- Salida(parámetros...)	Parámetros de salida
- Definición	La lista de los tipos de cada una de las variables, arreglos o constantes que necesitemos
Enteras	
Reales	
Alfanuméricos	
-Fin de Definición	
-Inicialización	Inicialización de todas las variables definidas anteriormente
Variable 1 ——— 0	
Constante 1 ——— 1.1416	
Arreglo 1 ——— 0	
-Fin de Inicialización	
- Conjunto de Instrucciones	Son de varios tipos
- Ejecuta nombre	
Entrada Parámetros..	Llamada a subalgoritmo
Salida Parámetros..	

-Si condición entonces Decisión
instrucción
instrucción
.
.
.
.
En caso contrario
instrucciones
.
.
.
-Fin del si

-Mientras condición Haz. Ciclo
instrucción
instrucción
.
.
.
.
- Fin de mientras

- Hasta condición Haz. Ciclo
instrucción
instrucción
instrucción
.
.
.
-Fin del Hasta

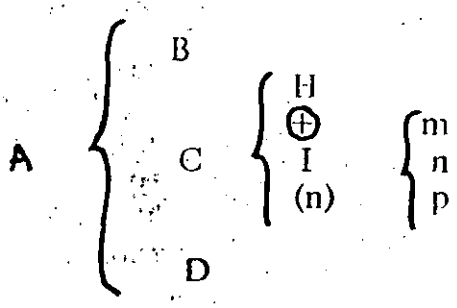
Fin del algoritmo nombre Finaliza el algoritmo

Esquemática: Representa algoritmos en base a dibujos con un significado predeterminado. Sin embargo en algoritmos cortos es clara, pero el crecer ésta los saltos en sus partes se vuelve complicado, el seguirlo y corregirlo es también difícil.

Mixta: Combina las oraciones de un lenguaje natural con símbolos que le permiten esquematizar una jerarquía entre funciones. Los símbolos que se utilizan son:

$\}$ (indica contención)
$\oplus$ (ó exclusivo)
( ) (cuando está debajo de algún elemento indica el número de veces que ésta se repite)

Su manipulación se hace poniendo del lado izquierdo de la llave el nombre de la función principal y a la derecha las funciones que dependen.



Dentro de (se hacen H ó I

La operación I contiene a m, n y p

Dentro de la A se hacen B, C y D.

ESTRUCTURAS BASICAS DE PROCESO

Existen tres estructuras de proceso por medio de las cuales se puede realizar cualquier algoritmo utilizando solamente estas estructuras lógicas de control simple.

1. La secuencia
2. La decisión o alternativa y
3. La repetición

LA SECUENCIA

La secuencia es una estructura de proceso que consiste en enunciar, una serie de instrucciones una después de otra. Una instrucción es: una frase en lenguaje natural, trivialmente traducible a código. Las frases en lenguaje natural son imperativas: idealmente consisten de un verbo activo (calcule, lea, escriba, verifique, etc.) seguido de una cláusula objeto lo más sencilla posible, por ejemplo:

Verifique el crédito del cliente
Calcule el deducible del salario
Encuentre el mínimo de los números en la lista
Asigne a la variable interés el valor de .15
Llame a la subrutina VERINI

LA DECISION

La decisión es una estructura de proceso que permite especificar alternativas en la ejecución de instrucciones dependiendo de una condición. Por facilidad de exposición se divide la decisión en tres tiros: simple, doble y múltiple, la forma clásica de representar la decisión es a través de un diagrama de flujo.

- LA DECISION SIMPLE

Si condición ENTONCES instrucción -1

Si la condición es verdadera la instrucción -1 se ejecuta, en caso contrario la instrucción se ignora y el proceso continúa.

- LA DECISION DOBLE

```
Si condición ENCONCES
    instrucción -1
SINO
    instrucción -2
FIN (SI)
```

Aquí se especifica que si la condición es verdadera, la instrucción-1 de-

berá de realizarse, y si la condición es falsa, entonces la instrucción -2 es la que se ejecuta. Sólo una de las dos instrucciones se ejecuta como resultado de la ejecución de la decisión doble.

- LA DECISION MULTIPLE

Esta estructura de proceso es una generalización de la decisión de la decisión doble, la estructura tiene la siguiente forma:

```
SI        condición-1        ENTONCES
            instrucción -1

SINO SI   condición -2       ENTONCES
            instrucción-2

SINO SI   condición-3        ENTONCES
            instrucción-3

SINO SI   condición-4        ENTONCES
            instrucción-4

   .
   .
   .

SINO
            instrucción-N

FIN(SI)
```

Al igual que en la decisión simple y doble, solamente una de las instrucciones se ejecuta como resultado de la ejecución de la estructura. Por lo tanto, las condiciones deberán ser mutuamente exclusivas. Debido a ésta última propiedad, es común encontrarse como parte del lenguaje de diseño y en algunos lenguajes de computadora la siguiente forma equivalente de la decisión múltiple que se conoce como "CASO":

```
CASO
  condición-1
     instrucción-1
  condición-2
     instrucción-2
  condición-3
     instrucción-3
  .
  .
  .
```

```
SINO
        instrucción-N

FIN (CASO)
```

Esta especificación de la parte de un proceso puede hacerse usando el caso:

```
CASO
        (mes=1, 3, 5, 7, 8, 10, 12)
        asigne al 31 al número de días
        (mes=4, 6, 9, 11)
        asigne 30 al número de días
        (año es bisiesto)
        asigne 29 al número de días

SINO
        asigne 28 al número de días

FIN(CASO)
```

Es también común encontrar la siguiente forma de decisión o alternativa múltiple:

```
CASO    "variable o expresión"
        (Lista de valores-1)
            instrucción-1
        (Lista de valores-2)
            instrucción-2
        (Lista de valores-3)
            instrucción-3
            .
            .
            .
        SINO
            instrucción-N

FIN(CASO)
```

El significado de esta estructura es intuitivo: si la variable o expresión tiene un valor de la lista de valores-1, la instrucción-1 es ejecutada y así sucesivamente. Si la variable o expresión tiene un valor que no está en alguna de las listas de valores la instrucción-N es ejecutada, por ejemplo:

```
CASO  estado-civil
      (CASADO)
```

```
                procese empleado casado
        (SOLTERO)
                procese empleado soltero
        (DIVORCIADO)
                procese empleado divorciado
        (VIUDO)
                procese empleado viudo
        (SEPARADO)
                procese empleado separado

SINO
                reporte error de estado civil

FIN(CASO)
```

Es posible que esta estructura de alternativa múltiple se encuentre en algún lenguaje de computadora de alto nivel.

LA REPETICION

La repetición es una estructura de proceso que permite la especificación de la ejecución iterativa de una serie de instrucciones.

Presentaremos los siguientes tipos de estructuras de repetición:

- La estructura MIENTRAS
- La estructura EJECUTA-HASTA
- La estructura REPITE
- La estructura CICLO

MIENTRAS

Esta estructura para el detallado de procesos tiene la siguiente forma:

```
MIENTRAS condición
    instrucción(es)
FIN (MIENTRAS)
```

El significado de esta estructura es el siguiente: si la condición es verdadera, la instrucción es ejecutada y después de ser ejecutada, la condición vuelve a ser evaluada. Si resulta verdadera la instrucción se vuelve a ejecutar. Este proceso continúa hasta que la condición sea falsa, en cuyo caso, la instrucción no se ejecuta y el proceso ya no se repite.

EJECUTA-HASTA

La estructura del ejecuta-hasta es la siguiente:

```
EJECUTA
        instrucción(es)
HASTA  condición
```

Esta estructura de proceso señala que, al ejecutarse la estructura, la instrucción debe de realizarse. Una vez que la instrucción ha sido ejecutada, la condición es evaluada y si su valor es falso, entonces la instrucción deberá ejecutarse de nuevo. Eso se repetirá hasta que la condición sea verdadera. En este momento, la instrucción no volverá a ejecutar y se dice que la estructura de repetición se ha terminado.

Una diferencia entre la estructura "MIENTRAS" y la estrucura "EJECUTE-HASTA" es que la última ejecuta la instrucción al menos una vez.

REPITE

Esta estructura de proceso es una de las estructuras más conocidas para los conocedores del lenguaje Fortran. Tiene la siguiente forma:

```
REPITE  variable= E-inicio, E-fin-E-incremento
        instrucción
FIN (REPITE)
```

El significado de esta estructura es el siguiente: al empezar la ejecución de la estructura, la variable toma el valor de la expresión E-inicio, si este valor es mayor que el valor de la expresión E-fin, la instrucción no se ejecuta y la estructura se termina. Sin embargo, si el valor de la variable es menor que el valor de E-fin entonces la instrucción se ejecuta. Una vez que la instrucción ha sido ejecutada, la variable se incrementa en un valor igual al de la expresión E-incremento. Nuevamente si el valor de la variable es menor que el valor de E-fin la instrucción se volverá a ejecutar; en caso contrario, la ejecución de la estructura se termina.

La variable deberá ser de tipo entera, al igual que las expresiones E-fin y E-incremento.

CICLO

La forma de esta estructura es la siguiente:

```
CICLO expresión
      instrucción
FIN(CICLO)
```

Con esta estructura se especifica lo siguiente: al inicio de la ejecución del ciclo, se calcula el valor de la expresión que debe de resultar en un número entero. Este número representa las veces que deberá repetirse la ejecución de la instrucción. Posteriormente la ejecución de la estructura se da por terminada. Si el valor de la expresión es cero o negativo, la instrucción no se ejecuta y se termina la estructura.

A continuación se indica como los conceptos de programación estructurada se usan en el detallado de procesos. Al uso directo de estas estructuras en la forma representada en esta fase mediante palabras reservadas e instrucciones en español se le conoce como especificación de procesos en "pseudo-código". Se entiende por pseudo-código un lenguaje estructurado, más no ejecutable.

Las palabras reservadas son las palabras clave que nos indican la estructura del proceso de un programa por ejemplo, SI, ENTONCES, SINO FIN (SI), etc.

Las estructuras básicas de especificación de proceso forman parte del repertorio de instrucciones de algunos lenguajes de computadora. En este caso, la traducción de un diseño en pseudo-Código al lenguaje de computadora es una tarea trivial.

En la presentación de las estructuras de proceso, se han usado el concepto de "instrucción" de una manera genérica para indicar, ya sea una frase en lenguaje natural o una estructura de proceso. Cuando se usa una estructura de proceso dentro de otra estructura de proceso, en la parte o partes donde se indica que debe ir una instrucción, se dice que hay un anidamiento, por ejemplo:

```
MIENTRAS condición
    SI condición-2 ENTONCES
        frase en lenguaje natural-1

    SINO
        CICLO expresión
            frase en lenguaje natural-2
        FIN(CICLO)

    FIN SI
        frase en lenguaje natural-3

FIN(MIENTRAS)
```

Note que el anidamiento de estructuras de proceso se representa usando el sangrado de las estructuras y frases en lenguaje natural. Al terminar el detalle de un proceso en la fase de diseño solo se deberá tener frases en lenguaje natural y estructuras de proceso anidadas.

En la práctica, la programación estructurada es una herramienta que permite escribir programas más claramente, más legibles y por lo tanto, con menos errores; sin embargo, no puede afirmarse que el mero uso de las estructuras de control lleva natural y automáticamente a escribir programas estructurados; una serie de reglas mecánicas no puede ser un sustituto de la claridad del pensamiento.

Desde el punto de vista de la programación estructurada, la mejor documentación de un programa la constituye la claridad de su estructura; además, la única documentación confiable de un programa es el programa mismo, pues sólo leyendo el código puede el programador dar por hecho lo que hace el programa. De ahí el énfasis en la legibilidad del código, base indiscutible de la programación estructurada.

Los algoritmos pueden darse en cualquier disciplina, en el caso que trataremos daremos un algoritmo de una disciplina muy especial. Será de cocina y lo representaremos en cada una de las descripciones mencionadas antes.

Descripción en Lenguaje Natural

ESCOTAFI

Se baten seis claras muy bien, luego se pone una a una seis yemas, se ban incorporando 200 gr. de azúcar, se sigue batiendo hasta que forme cordón se pone zumo de un limón verde, 50 gr. de harina cernida con una cajita de fécula de máiz y dos cucharaditas de royal; ya bien batido, se pone en un molde engrasado, se mete al horno a 250°C, durante media hora.

Descripción en Lenguaje Algorítmico

ALGORITMO ESCOTAFI

ENTRADA (Huevos, azúcar, limón, harina cernida, caja de fécula de maíz, cucharadas de royal)
SALIDA (Escotafi)

INICIALIZACION

Huevos _____ 6
Azúcar _____ 200 gr.
Limón _____ 1
Harina cernida _____ 50 gr.
Caja de fécula de maíz — 1
Cucharada royal _____ 2

FIN DE INICIALIZACION
HASTA que las 6 claras estén bien batidas Haz
Batir

FIN DE HASTA
HASTA que forme cordón y se hayan agregado 6 yemas y 200 gr. de azúcar Haz
Batir

SI hay yemas enconces
Agregar una
SINO
Sigue adelante
FIN SI

FIN DE HASTA

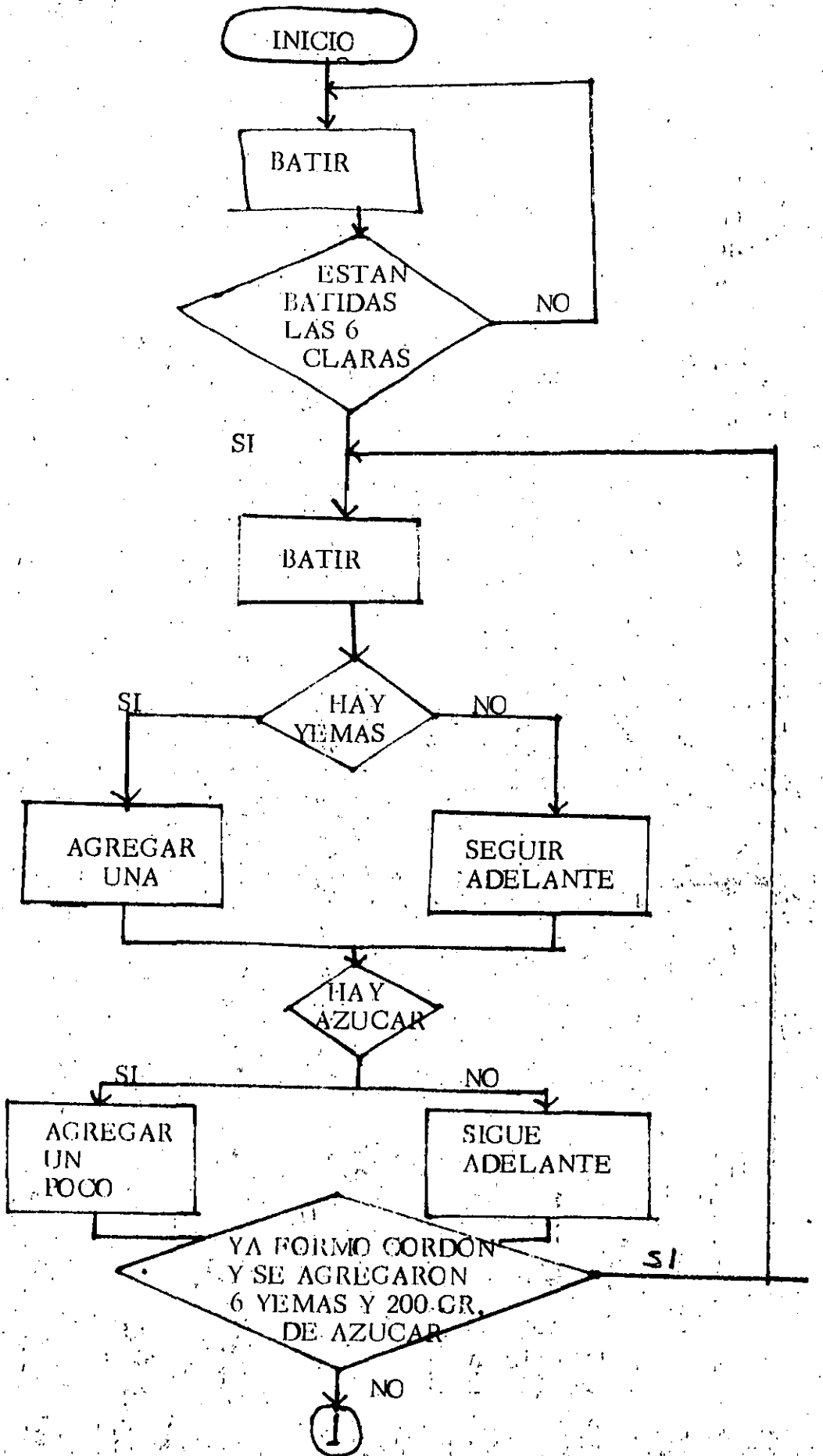
Poner zumo de un limón verde
Poner 50 gr. de harina cernida
Poner una cajita de fécula de maíz
Poner 2 cucharadas de royal
HASTA que esté bien batido Haz
Batir

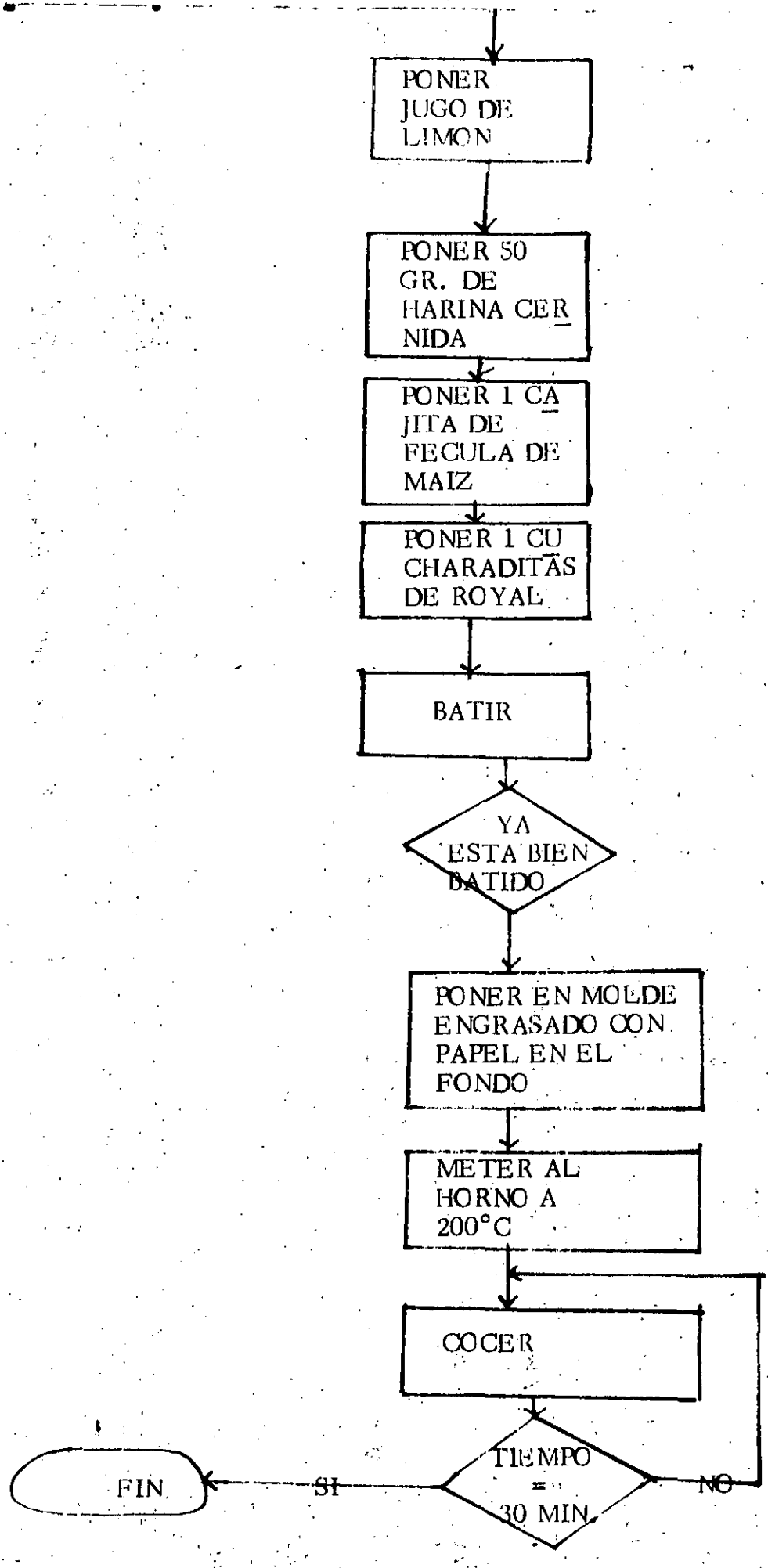
FIN DE HASTA

Poner en molde engrasado con papel en el fondo
Meter al horno a 250° C
HASTA que tiempo = 30 minutos Haz
Cocer

FIN DE HASTA
FIN DE ALGORITMO PARA ESCOTAFI

DESCRIPCION ESQUEMATICA




PONER JUGO DE LIMON
PONER 50 GR. DE HARINA CERNIDA
PONER 1 CAJITA DE FECULA DE MAIZ
PONER 1 CUCHARADITAS DE ROYAL
BATIR
YA ESTA BIEN BATIDO
PONER EN MOLDE ENGRASADO CON PAPEL EN EL FONDO
METER AL HORNO A 200°C
COCER
TIEMPO = 30 MIN.
SI
NO
FIN

DESCRIPCION MIXTA

ESCOTAFI

- **BATIDO 1** (hasta que estén las 6 claras bien batidas)
  - BATIR
- **BATIDO 2** (hasta que forme cordón y se hayan agregado 6 yemas y 200 gr. de azúcar)
  - BATIR
  - Existencia de yemas
    - hay ⊕ → agregar una
    - No hay → sigue adelante
  - Existencia de azúcar
    - hay ⊕ → agregar un poco
    - No hay → sigue adelante
- Condimentación
  - Poner zumo de un limón
  - Poner 50 gr. de harina cernida
  - Poner 2 cucharaditas de royal
- **BATIDO 3** (hasta que esté bien batido)
  - BATIR
- **COCIMIENTO** (hasta tiempo = a 30 minutos)
  - COCER

¿Qué es el lenguaje Basic?

El Basic es un lenguaje de programación que actúa como un intermediario entre el usuario y un computador. Mediante el vocabulario y las reglas Basic, se le ordena al computador lo que se quiere que haga y el computador transforma estas instrucciones en lenguaje de máquina para poderlas ejecutar. Cada lenguaje de programación tiene sus ventajas e inconvenientes. El Basic (Beginners All-purpose Symbolic Instruction Code) es relativamente fácil de usar y se considera adecuado para un gran número de aplicaciones. No obstante, si se espera ejecutar un gran número de sofisticados programas, son más adecuados otros lenguajes de programación, como el Pascal, por ejemplo.

LOS DOS MODOS DE OPERACION EN BASIC

El computador puede operar en dos modos diferentes, en BASIC: el modo directo y el modo indirecto o programado. En el modo directo se obtiene una respuesta inmediata a cada orden o sentencia ; es decir, el computador responde de forma similar a una calculadora. El modo indirecto se emplea cuando se desea almacenar las propias instrucciones BASIC para una ejecución posterior; esto es, cuando se escriben programas para el computador.

Empleo del modo directo.

El modo directo se emplea cuándo se quieren ejecutar cálculos con rapidez o probar la ejecución de sentencias BASIC.

El modo directo tiene dos importantes características. En primer lugar, el BASIC responde de forma inmediata a una orden en modo directo. En segundo lugar, cuando una orden en modo directo se ejecuta, dicha orden no se almacena en la memoria del programa; es decir, una vez que se pulsa ENTER, la orden permanece en la presentación de la pantalla, pero no se almacena como parte del programa.

Para ver la forma de trabajo del modo directo, vamos a calcular el valor de 512/16. En respuesta al mensaje de petición BASIC (OK), escriba lo siguiente, finalizándolo con la tecla ENTER:

```
PRINT 512/16
```

Al pulsar ENTER, el computador responderá con

```
OK
PRINT 512/16
 32
OK
```

Por tanto, no queda guardada en memoria ninguna línea en modo directo, ya que se ha borrado al desaparecer de la pantalla. No obstante, es posible guardar información en la memoria con líneas de modo directo si se especifica lo que denominamos variable. De momento basta con conocer que variable es una palabra o una letra, a su elección, a la que se le asigna un valor determinado para posterior referencia. Como prueba, introducir la siguiente orden en modo directo:

```
PIANO =88
```

Con lo anterior se ha almacenado en memoria el número 88 con el rótulo PIANO. Para comprobarlo, imprimir los valores de la variable con la orden PRINT.

```
OK
PIANO=88
OK
PRINT PIANO
 88
OK
```

BASIC responde buscando los valores actuales de la variable PIANO y presentándolos en la pantalla.

Hasta el momento solamente se han estudiado las órdenes BASIC, en modo directo, de una en una. Pero es también posible introducir varias órdenes en una línea, mientras cada orden esté separada por el signo: (dos puntos). Puede comprobarse con lo siguiente:

```
FOR X =1 TO 10: FOR Y=1 TO X :PRINT"";:NEXT Y:
PRINT "Esta es la " x "a vez" :NEXT X
```

Al pulsar ENTER, el computador responde con:

```
OK
FOR X=1 TO 10:FOR Y=1 TO X:PRINT"";"NEXT Y:
PRINT "Esta es la "X"a vez":NEXT X
Esta es la 2.ª vez
 Esta es la 3.ª vez
  Esta es la 4.ª vez
   Esta es la 5.ª vez
    Esta es la 6.ª vez
     Esta es la 7.ª vez
      Esta es la 8.ª vez
       Esta es la 9.ª vez
        Esta es la 10.ª vez
OK
```

Empleo del modo indirecto

En modo directo, se imprimía y se ejecutaba una sola línea de una o más sentencias BASIC. En modo indirecto, es posible escribir una serie de líneas que pueden quedar almacenadas y ejecutarse posteriormente. Estas series de sentencias BASIC, en varias líneas, se denominan programa. El modo indirecto se emplea, por tanto, para crear programas BASIC.

Existen dos características importantes que distinguen el empleo del BASIC en modo indirecto. En primer lugar, con el modo indirecto puede usarse más de una línea de sentencia para resolver un problema sencillo y dichas líneas pueden quedar conservadas en la memoria. En segundo lugar, el Modo Indirecto se usa, de forma automática, siempre que se inicie una línea BASIC con un número de línea. Al pulsar ENTER al final de cada línea BASIC, dicha línea pasa a formar parte, de forma automática, del programa BASIC que, en este momento, está siendo conservado en la memoria de programa. Si fuera necesario, puede obtenerse en la pantalla cada una de estas líneas, mediante su número correspondiente.

Vamos a escribir un programa corto en modo indirecto. Escribir lo siguiente como respuesta al mensaje de petición BASIC, pulsando ENTER al final de cada línea.

```
OK
10 FOR X=1 TO 10
20 PRINT "Este es un programa util"
30 NEXT X
40 END
```

Al digitar la palabra RUN el computador responderá:

```
RUN
Este es un programa util
Este es un programa util
Este es un programa util
   .
   .
   .
(10 veces)
```

LINEAS DE PROGRAMA

Los programas BASIC giran alrededor del concepto de línea. Existen dos clases de líneas: líneas físicas y líneas lógicas.

Una línea física es la línea, propiamente dicho, en el dispositivo de salida que está empleando. Por ejemplo, cuando se está empleando la pantalla, la línea física es de 80 caracteres de longitud.

La línea lógica, sin embargo, se mide de forma diferente. Puede tener una longitud de hasta 225 caracteres y finaliza con el ENTER. La línea lógica es muy importante, dado que constituye la unidad de información que BASIC procesa en un momento determinado.

Una línea lógica BASIC puede ocupar varias líneas de una presentación (líneas físicas). El lugar en el que se pulsa ENTER, y no el final de una línea en la pantalla, constituye el final de una línea lógica. El BASIC busca el carácter ENTER cada vez que procesa una línea; dicho carácter no ha de ser, necesariamente, el carácter final de cada una de las líneas que se presenten en pantalla, en las sentencias BASIC.

El siguiente ejemplo ilustra cómo una línea lógica puede estar compuesta por más de una línea física.

```
230 PRINT "No solo esta equivocado, sino que no lleva razon, fue su respuesta mordaz al caballero de alarmante apariencia y opinión a imponer"
```

Si trata de introducir una línea BASIC de más de 225 caracteres, se perderán los que excedan de esa cifra, al pulsar la tecla ENTER.

Números de línea

El número que precede a cada línea BASIC se denomina número de línea. Tal como se ha visto, cuando se asigna un número a una línea, dicha línea se conserva y no se ejecuta inmediatamente (modo indirecto). El número de línea actúa como una etiqueta para conservar dicha línea en la memoria, y la línea numerada es ahora parte del programa BASIC en curso.

El número de línea ocupa la primera posición en una línea lógica.

A continuación se exponen ejemplos de líneas con un número de línea válido.

```
1)  1010  PRINT "Este es un número de línea grande"
```

2)
```
10 PRINT "Esta es una lista de valores de X y Y"
15 FOR x=0 TO 70
16 y=x 5
20 PRINT x, y
30 NEXT
40 END
```

Debe notarse que el número de línea da comienzo a la línea BASIC y que la sentencia BASIC va a continuación separada por un espacio, al menos.

NUMERACION DE LAS LINEAS DE PROGRAMA.

Cuando se compone un programa BASIC por primera vez, una buena idea consiste en numerar las líneas de diez en diez. De esta forma, será más fácil insertar otras líneas si fuera necesario. Por ejemplo, puede iniciarse un nuevo programa de la forma siguiente:

```
10 ' Esta es una buena forma para numerar las líneas de un progra
   ma
20 GOTO 500 ' Inicializar sistema
30 GOTO 600 ' Entrada del operador
    .
    .
    .
```

Contenido de una línea BASIC

Cada línea BASIC está compuesta por dos o más sentencias BASIC denominadas órdenes. En la mayor parte de los casos, una sentencia BASIC puede contener instrucciones a ejecutar por el computador, datos para usar con otras setencias, o comentarios que le ayuden a darse una idea sobre lo que el programador ha hecho con un determinado programa.

Sentencias Basic

La sentencia Basic incluye una frase constituida por palabras reservadas BASIC y, si fuera preciso, por un argumento. Por ejemplo, consideremos las siguientes líneas:

```
300 PRINT'Esta parte, entre comillas, es el argumento para la palabra re-
servada PRINT"
310 IF x=4 THEN 450
```

En la línea 300 la palabra reservada es PRINT, que no es otra cosa que la sentencia Basic PRINT, y el argumento es la cadena que le sigue. En la línea 310, las palabras reservadas son IF (Si) y then (entonces), que corresponde

la sentencia Basic, IF-THEN. Los argumentos en esta línea son x=4, denominado expresión, y el número 460, que en este caso, hace referencia a otra línea de programa.

Sentencias Múltiples.

Las líneas de programa Basic pueden contener más de una sentencia. Cada una de ellas ha de separarse del resto por el signo: (dos puntos). Por ejemplo:

```
600 FOR x=1 TO 600: NEXT X
```

En el ejemplo, la primera sentencia es de la clase FOR (para) con los argumentos x, 1 y 600. La segunda sentencia es del tipo NEXT (Siguiente). Cuando se ejecuta esta línea, Basic procesa la línea 600 veces. Esta línea es muy útil para generar retrasos en los programas.

Espacios.

Los espacios en las líneas Basic tiene diferentes efectos, dependiendo de su lugar en la línea. En determinados lugares, se requiere un espacio. Por ejemplo, los números de línea y las palabras reservadas Basic han de separarse por lo menos por un espacio. En general, las palabras reservadas han de separarse por un espacio, como mínimo, de cualquier otro carácter de la misma línea. Una excepción a lo anterior es cuando una palabra reservada va precedida, inmediatamente, por el signo:, es decir, los dos puntos sirven de delimitador. En Basic, los delimitadores separan las partes de una línea.

En la mayor parte de los casos restantes, Basic ignora los demás espacios, excepto si forman parte de una cadena.

Los Elementos del Basic.

Los programas Basic manejan información, a ésta se le conoce con el nombre de datos. Los datos pueden estar constituídos por caracteres o números.

Datos de caracteres. Cadenas

Una cadena es una secuencia de caracteres que comienzan y terminan con comillas (""). Los siguientes datos son ejemplos de cadenas válidas:

```
"La ruta a iluminar no está clara"
"$ 68.48"
" Nombre de cuenta"
```

La cadena puede contener en la mayoría de las versiones Basic, hasta 255 caracteres, con excepción de las comillas ("")

Datos Numéricos. Números

Existen varios tipos de números que el BASIC reconoce. Estos son:

. Números enteros
. Números de coma fija (o punto fijo)
. Números de coma flotante (o punto flotante)

ENTEROS

Los enteros BASIC son aquellos números enteros comprendidos entre -32768 y + 32767. Los números negativos han de ir precedidos por el signo (-). Aunque puede usarse el signo (+) para los números positivos, un número sin signo alguno se considera positivo. Las comas (,) no se permiten en ninguno de estos números. A continuación se exponen unos ejemplos de enteros válidos y no válidos:

4	Válido
-30000	Válido
1777	Válido
12, 840	No válido -no se permiten comas
32840	No válido -demasiado grande
-99999	No válido -demasiado pequeño

NÚMEROS DE COMA FIJA

Los números BASIC de coma fija (o de punto fijo) son números reales cuyo margen puede estar entre ± 99999999999999999 (diecisiete nueves). Debe señalarse que, aunque el BASIC puede operar con números de 17 dígitos, sólo pueden imprimirse 16, como máximo. Más adelante se considerarán las consecuencias de lo anterior.

Los números de coma fija pueden estar constituídos por dígitos a la derecha o a la izquierda del punto decimal y pueden tener una longitud de hasta 17 dígitos. A continuación se muestra un ejemplo de números válidos y no válidos de coma fija.

345. 234234	Válido
-94949494949. 494	Válido
0. 00000000000000001	Válido
0. 000000000000000001	No válido -demasiados dígitos
4, 509	No válido -no se admiten comas

NUMEROS DE COMA FLOTANTE

Los números BASIC de coma flotante son números reales expresados en forma exponencial (como la notación científica). De esta forma un número consta de dos partes: una mantisa seguida de un exponente. El exponente viene precedido por una E. Un número de coma flotante tiene un valor correspondiente a la mantisa multiplicada por diez elevado al exponente tal y como se expone en los siguientes ejemplos:

$$345.44E3 = (345.44) * 10^3$$
$$= (345.44) * 10 * 10 * 10$$
$$= 345,440$$

$$-6.92341E-3 = (-6.92341) * 10^3$$
$$= (-6.92341) * 0.001$$
$$= -0.00692341$$

Los números BASIC de coma flotante van desde 10E-38 a 10E+38 o de -10E-38 a -10E38.
En general, los números de coma flotante se emplean para representar números grandes o muy pequeños, especialmente aquellos números formados por muchos dígitos para ser respresentados en forma de números de coma fija.

Precisión de los números BASIC

La precisión de un número BASIC ha de entenderse como el número de dígitos necesarios para representar exactamente dicho número. Por ejemplo, si conocemos que un determinado número tiene siete dígitos de precisión, solamente los seis primeros dígitos de dicho número son exactos. Cualquier otro dígito se usa, exclusivamente para redondeo.

Existen dos grados de precisión en BASIC: precisión entera, empleada con los números enteros y las precisiones simple y doble, ambas de aplicación a los números de coma fija y variable.

PRECISION ENTERA

Si un número es entero, en donde todos los dígitos empleados para representar dicho número son precisos, mientras dicho número esté dentro del margen válido, previamente establecido, para dichos tipos de número.

Si el resultado de un cálculo en el que se emplean números enteros exactos no es otro entero, el resultado impreso tiene un error de $\pm$ 0.5, si el resultado se redondeó a un entero, o de $\pm$ 1 si el resultado se truncó a un

entero. La diferencia entre redondeo y truncamiento puede verse a continuación:

Número original	Redondeado	Truncado
3.00	3	3
3.01	3	3
3.49	3	3
3.50	4	3
3.99	4	3
-3.50	-4	-3
-3.49	-3	-3

PRECISION SIMPLE

Los números de precisión simple, en BASIC, se representan con siete dígitos, de los cuales los seis primeros son exactos. Un número se considera que posee precisión simple si no es entero y si cualquiera de las siguientes características es cierta:

- El número consta de siete dígitos o menos.
- Va seguido del símbolo admiración (!)
- Es de coma flotante, con una E precediendo al exponente.

Los siguientes números son ejemplos de números de precisión simple en BASIC.

```
847.99
9!
-1234.34E4
```

Para entender el significado de precisión en un cálculo de precisión sencilla, ha de considerarse lo contenido en el ejemplo siguiente:

```
OK
PRINT 234.44/3
 78.1466
OK
```

Dado que se trata de un cálculo de precisión sencilla, sólo son exactos los seis primeros dígitos de la respuesta. No obstante, puede usarse el séptimo dígito para redondear la respuesta a 78.1467. Obsérvese que la verdadera respuesta es 78.1466666...seguido por un número infinito de seises y que si se redondea este número a seis dígitos también se obtiene 78.1467.

DOBLE PRECISION

Los números de doble precisión están almacenados en el interir y constan de diecisiete dígitos. Dieciséis de dichos dígitos quedan impresos (pueden emplearse como salida) y los dieciséis de cualquier resultado impreso pueden considerarse exactos.

Un número puede considerarse como de doble precisión si se da una de las siguientes características:

- El número tiene ocho dígitos o más
- Va seguido del símbolo #
- Es un número de coma flotante, con la letra E precediendo al exponente.

Los siguientes números son ejemplos de números de doble precisión en BASIC.

```
4#
12345678
12345678901234567
-234.90009E3
```

o los referenciados con un nombre, denominado variable.

```
X#=82746.7888
OK
PRINT X#
82746.7888
OK
```

En este caso, a la variable X# se le ha asignado el valor de doble precisión 82746.7888.

Las variables se pueden emplear para representar datos numéricos y datos en caracteres (cadenas). Por ejemplo, en el programa siguiente:

```
10 NUMBER1=3
20 NUMBER2=5
30 COMPUTER$="IBM PC"
40 PRINT "EL "COMPUTER$; "puede escribir variables
   de cadenas."
50 PRINT NUMBER1*NUMBER2
60 END
```

En este programa, se emplean dos variables numéricas, NUMBER1 y NUMBER2, y una variable de cadena, COMPUTER$.

Las variables son muy útiles porque permiten sustituir números o cadenas de caracteres con rótulos (nombres de variables) que se han elegido para representar lo que un número o cadena efectivamente representa. Las variables le permiten también emplear la misma línea de programa con diversos datos. Por ejemplo:

```
2000 PRINT "El número equivalente de millas es"
     4294*.6 "millas."
```

En este caso, cualquier valor para una distancia media en kilómetros se expresa con la constante, 4294. La conversión a millas (multiplicando el número de kilómetros por 0.6) se hace en la sentencia PRINT. Una manera mejor de emplear lo anterior es mediante el uso de variables, como en el caso siguiente:

```
800 KILOM=4294
810 UNIT$="millas"
820 MILES=KILOM*.6
    .
    .
    .
830 PRINT "El número equivalente de" UNIT$; "es" MILES UNIT$
```

En este caso concreto, sustituimos la variable por tres partes de la sentencia PRINT del primer ejemplo. En primer lugar, la expresión matemática 4294*.6 ha sido sustituida por la variable MILES. Esta variable le proporciona al lector del programa una idea mucho más clara del significado de la expresión. En segundo lugar, sustituimos la variable 4294 por la variable KILOM. y, en tercer lugar, sustituimos la palabra millas por la variable UNIT$.

A continuación puede emplearse la sentencia PRINT para presentar el resultado de más de una conversión simplemente cambiando los valores de KILOM y UNIT$.

Nombres de las variables.

Deben tenerse en cuenta varias consideraciones cuando se asignen nombres a las variables.

En primer lugar, un nombre puede contener hasta 40 caracteres y tiene necesariamente que empezar con una letra. Para el resto del nombre sólo puede usarse números, letras y el punto decimal, con la excepción del último carácter, que puede ser un carácter especial que especifique la precisión de la variable, como se verá más adelante.

En segundo lugar, los nombres de las variables no pueden ser palabras reservadas BASIC o palabras reservadas seguidas de los caracteres $, ¡ o #. Como palabras reservadas se incluyen las órdenes BASIC, sentencias, nombres de función y nombres de operador. Los nombres de variable pueden incluír, no obstante, palabras reservadas integradas. Por ejemplo:

SOUND
SOUND¡

Son nombres de variables no válidos, pero

SOUNDVALUE¡

es válido.

En tercer lugar, un nombre de una variable que comienza con FN sólo es válido si dicho nombre fue definido como función definida por el usuario. Dicho tipo de funciones se examinarán más adelante.

Cuarto y más importante, el nombre de la variable, en sí mismo, puede definir el tipo y precisión de la variable. Lo anterior se consigue mediante la inclusión de lo que denominamos variable tipo carácter de declaración ($, ¡, #), al final del nombre de la variable.

NOMBRE DE LAS VARIABLES DE CADENA

El nombre de una variable, seguido del símbolo $, define una variable en cadena. A continuación se exponen unos ejemplos de lo que denominamos nombres válidos de variable de cadena:

E$
ADRESS$
POLITICALAFFILIATION$
ABRACADABRA13$

Nombres de Variables enteras.

Un nombre de variable seguido por % define una variable entera. Por ejemplo:

YEAR%
SPIN%
THISISARIDICULOUSBUTVALIDINTEGERNAME%

Nombre de variables de precisión simple.

Los nombres de variables de precisión simple (sencilla) van seguidos del

símbolo admiración (¡) o no les sigue nada. Por ejemplo, cada uno de los nombres siguientes de variables de precisión sencilla es válido.

KILOMIGHT¡
AMOUNTDUE
NODAYSAFTER9.9.57¡

Nombre de variables de doble precisión

En el nombre de una variable de doble precisión, el último carácter de ser #. Por ejemplo.

PI#
SPEEDOFSOUND#
FREQUENCY#

Cambio de Precisión.

Cuando se asigna un número de una precisión a una variable con una precisión más baja, el número se redondea hasta la precisión iplícita en el número de precisión más bajo. Por ejemplo:

X=45.9039852

A esta sentencia se le asignará el valor 45.90399 para la variable de precisión sencilla X.

Por otra parte, considérese lo siguiente:

T%=32.76724E3

Esta sentencia asignará 32.767 a la variable entera T%.

Finalmente, consideremos lo siguiente:

S%=32.76755E3

Esta sentencia proporcionará a la salida un mensaje de error overflow (desbordamiento), dado que el valor redondeado 32.768 es mayor que lo que puede representarse mediante una variable de entero.

Cuando se trate de combinar constantes y variables de diferentes precisiones, todas ellas reciben el mismo tratamiento que si tuvieran la precisión de la constante o variable de precisión más alta. El resultado de la operación se expresa, también, con esta misma precisión. Por ejemplo:

```
OK
A%=3
OK
PRINT A%/4.44#
.6756756756756757
OK
```

En este caso, la precisión más alta en la sentencia PRINT viene representada por la doble precisión de la constante 4.44#. Como resultado, el BASIC utiliza la variable entero A% como si fuera también un número de doble precisión. El resultado 0.6756756756756757 se presenta como si de un número de doble precisión se tratara. Es preciso recordar que los números de doble precisión se visualizan con 16 dígitos, pero están representados en el computador con 17 dígitos.

Matrices (Arrays)

Una matriz es un conjunto de variables, todas del mismo tipo, con un nombre común. Por ejemplo, supongamos un programa que procesa una lista de nombres de alumnos. Es posible rotular dichos nombres con un conjunto de nombres de variables de cadena singulares, tal como:

```
 10 STUDENTONE$ = "Abigail Aardvark"
 20 STUDENTTWO$ = "Arnie Adams"
                  .
                  .
                  .
190 STUDENTNINETEEN$ = "Zoe Zither"
```

Una forma mejor consiste en colocar una matriz (array) para todos los nombres de los alumnos. Con un matriz, se hace referencia a una variable sencilla (llamada un elemento de la matriz), especificando el nombre de la matriz seguido por un subíndice que corresponde a la variable de interés. En el ejemplo que nos ocupa, puede definirse la matriz STUDENT$(n), en la que n es el subíndice.

```
  5 DIM STUDENT$(19)
 10 STUDENT$(0) = "Abigail Aardvark"
 20 STUDENT$(1) = "Arnie Adams"
                  .
                  .
                  .
190 STUDENT$(18) = "Zoe Zither"
```

La memoria DIM, en este caso, sólo reserva espacio en la memoria para los 19 elementos de la matriz. Es preciso señalar que comenzamos numerando los elementos de la matriz con el 0; normalmente, el elemento más bajo de una matriz es cero.

Las matrices permiten, de forma sistemática, recuperar datos en el progra-

ma. Por ejemplo, veamos una rutina que imprime una lista de los nombres de los alumnos.

```
580 FOR X=0 TO 18
590 PRINT SUTDENTNAME$(X)
600 NEXT
```

Las líneas de programa FOR X=0 TO 18 y NEXT dan origen a que la línea 590 se ejecute un total de 19 veces, con el valor de X aumentando desde 0 hasta 18, por consiguiente.

MATRICES DE MAS DE UNA DIMENSION

Un nombre de matriz puede ir seguido por más de un subíndice, en cuyo caso, cada subíndice representa lo que se conoce como una dimensión de la matriz. Debe señalarse que la matriz STUDENT$(n) tiene de dimensión 1. Esta organización corresponde a una lista de una sola dimensión.

Las matrices de dos dimensiones le permiten organizar información en un formato de fila y columna. Por ejemplo, si se desea almacenar la dirección de cada estudiante, junto con su nombre, puede crearse la matriz de dos dimensiones STUDENT$(n, m). En este caso, la información se almacena de la siguiente manera.

```
STUDENT$(0.0)     STUDENT$(0.1)
STUDENT$(1.0)     STUDENT$(1.1)
      .                 .
      .                 .
      .                 .
STUDENT$(n, 0)    STUDENT$(n, 1)
```

Los nombres de los alumnos aparecen como entrada en la primera columna, mientras que la segunda contiene las direcciones correspondientes.

Se emplea el subíndice n para identificar a un alumno y el subíndice m para seleccionar el nombre de un alumno o su dirección. Por lo anterior, STUDENT$(1, 1) se refiere a la dirección del segundo alumno.

Como un ejemplo de matriz tridimensional, consideraremos una matriz denominada BOOK (p, l, w) utilizada para almacenar texto. En esta matriz, el subíndice p identifica un número de página, l identifica una línea dentro de una página y w identifica una palabra en una línea.

EXPRESION BASICA

Una expresión BASIC puede estar formada por una variable o una constante sencillas o por una combinación de constantes, variables, operado-

res y funciones. Algunos ejemplos de expresiones BASIC son los siguientes:

```
NOTE%
Y+43.2292
AMOUNT*(TAXRATE/100)
```

Los datos de cada expresión se denominan operandos. Los símbolos que definen lo que una expresión hace con sus operandos, se denominan operadores.

Operadores

Los operadores expresan lo que una determinada expresión hace con los datos. Por ejemplo, el operador* es el operador de multiplicación.

Operadores aritméticos

Los operadores aritméticos y las operaciones que ejecutan en una expresión numérica, son los que se indican en la Tabla siguiente.

Los operadores se listan en orden de prioridad (esto es, el orden en que las operaciones se ejecutan cuando una expresión contiene más de un operador). Debe indicarse que los operadores de multiplicación y de división con punto (coma) flotante, y los de suma y resta, tienen el mismo orden de prioridad. Esta jerarquización concuerda con lo que puede encontrarse en el álgebra moderna.

Operador	Operación
^	Potenciación
-	Negación
*	Multiplicación
/	División de coma flotante
\	División entera
MOD	Módulo aritmético
+	Suma
-	Resta

POTENCIACION

Ejemplos de expresiones que contienen el operador ptenciación son los siguientes:

```
234992.234444^.837
CUBESIDE^3
5.234E3^EXPO#
```

En la última expresión, la constante 5.234E3 se eleva hasta el valor EXPO#. Supongamos que el valor EXPO# es igual a 4.2875. Dicha expresión se evaluará, por tanto, de la manera siguiente:

```
5.234E3^EXPO# = 5.234E3^4.2875
              = (5.234*1000)^4.2875
              = 5,234^4.2875
              =8,800,314,000,000,000
              = 8.800314E15   (Este es el valor empleado por el
                               BASIC, a no ser que dicha expre
                               sión esté asignada a un entero o
                               a una variable de precisión sen-
                               cilla.)
```

NEGACION

Por negación se entiende que se toma el valor negativo de un número, siempre que dicho número vaya precedido del signo menos (-). Por ejemplo:

```
-(-395) = 395
-INCOME = -20.89      (Suponiendo que el valor de INCOME fuese 20.89)
-INCOME = 386.29      (Suponiendo que el valor de INCOME fuese de
                       -386.29)
```

MULTIPLICACION

La multiplicación se representa por el operador asterisco (*). Por ejemplo:

```
ITEMA%(X)*WEIGHTA!(X)
AMNTGAGE*PRICEPERLB
```

División de coma flotante

La división de coma flotante se representa por el operador barra (/). El resultado de una división de coma flotante es un número real, representado en el formato de coma flotante si su magnitud lo justifica. La precisión del resultado (el cociente) es la misma que la más alta del dividendo o divisor. Por ejemplo:

```
3#/DIVISOR%
```

nos dará un cociente de doble precisión. Por otra parte,

```
83/PORTNOY
```

nos dará un resultado de precisión sencilla.

DIVISION ENTERA

La división entera se representa mediante una barra invertida(\). En la división entera, el BASIC convierte, en primer lugar, en números enteros el dividendo y el divisor, redondéndolos, si es necesario. Ejecutada la división el cociente se convierte en un número entero mediante redondeo al número inferior. Por ejemplo, las siguientes expresiones ilustran el mecanismo de la división entera:

```
269.9\3=270\3
       = 90
268.9\3=269\3
       =89.6666666...
       =89
```

Mediante el problema siguiente se explica la necesidad de una operación de división entera. Supongamos que queremos distribuír X pájaros entre Y jaulas. Cada jaula ha de contener el mismo número de pájaros y no es posible dividir un pájaro en partes. En este caso, el cociente de una división de coma flotante debe ser transformado en un entero, dado que no puede partirse un pájaro en trozos. Una división entera ejecuta lo anterior de forma automática.

MODULO ARITMETICO

El operador MOD se emplea para obtener el resto entero de una división entera.

La operación de módulo aritmético, que se solicita con el operador MD puede explicarse con el ejemplo siguiente:

```
82 MOD 9=resto entero de (82 9)
        =1
3.43 MOD 8=6
```

En el segundo ejemplo, 6 es el resto redondeado por truncación por defecto de la operación 6 dividido por 8.

Debe indicarse que el operador MOD debe estar separado por, al menos, un espacio del segundo operador; de no ser así, el BASIC lo interpretará

como una variable.

SUMA Y RESTA

Las sumas y restas se representan por los operadores + y -, respectivamente. Cuando el signo - se emplea como operador negativo, es equivalente a la substracción del operando de 0. En otras palabras:

```
- VARIABLE=0-VARIABLE
```

Operadores de relación.

Los operadores de relación pueden actuar con números o cadenas y se emplean para comparar dos operandos del mismo tipo. Los operadores de relación y las operaciones que resuelven se señalan en la Tabla siguiente.

No existe un especial orden de prioridad en los operadores de relación. Se evalúan de izquierda a derecha, en una expresión dada (suponiendo que no existe paréntesis en dicha expresión, tal y como veremos). Los operadores de relación se emplean, por lo general, para comprobar ciertas condiciones. Por ejemplo:

```
          .
          .
          .
230 IF ANSWER>99 THEN 1000
240 PRINT "La respuesta es correcta (OK)."
          .
          .
          .
1000 ANSWER=0
1010 PRINT "La respuesta es demasiado grande y ha sido puesta a
      0."
          .
          .
          .
```

En este caso, la expresión de la línea 230 comprueba el valor de ANSWER (RESPUESTA). Si es menor o igual a 99, se ejecuta la línea 240. Si es mayor que 99, la ejecución continúa en la línea 1000.

Operador	Operación
=	Igualdad
o	Desigualdad
	Menor que
	Mayor que
= o =	Menor que o igual a
= o =	Mayor que o igual a

COMBINACION DE OPERADORES ARITMETICOS Y DE RELACION

Los operadores de relación pueden combinarse con los aritméticos en determinadas expresiones. Por ejemplo:

TESTVALUE<=(69*HEIGHT)

compara el resultado aritmético de la expresión 69*HEIGHT (ALTURA) con el valor actual de TESTVALUE.

RANK+(SCORE>450)+(HANDICAP<>3)

Esta expresión tiene como resultado un valor que es la suma de RANK y del resultado de las expresiones de relación SCORE>450 y HANDICAP<>3. Dicho de otra forma, suponiendo que RANK es igual a 38, el valor de esta expresión dependerá de los respectivos valores de SCORE y HANDICAP, tal como se indica a continuación:

Valores de los operandos	Valores de la expresión
SCORE<=450. HANDICAP=3	38+ 0+ 0=38
SCORE<=450. HANDICAP<>3	38+ 0+ 1=37
SCORE>450, HANDICAP=3	38+ - 1+ 0=37
SCORE>450, HANDICAP<>3	38+ - 1+ 1=36

Debe indicarse que, en este ejemplo, las expresiones entre paréntesis se evalúan en primer lugar. En otras palabras, el orden de prioridad de los operadores depende de cualquiera de los paréntesis de la expresión.

OPERACIONES DE RELACION EN LAS CADENAS

Cuando un operador de relación trabaja en dos cadenas, los caracteres de una de ellas se relacionan con los de la segunda. Esta comparación se hace carácter a carácter, comenzando por el principio de la cadena. Cuan

do se obtienen diferencias en las cadenas, BASIC da un resultado basado en aquellos caracteres. Si se alcanza el final de una cadena antes de hallar una diferencia entre las dos, la cadena más corta se considera menor que la cadena más larga.

Las magnitudes relativas de los caracteres de letras y números son las siguientes:

0, 1, 2, 3, . . . , 7, 8, 9, A, B, C, . . . , X, Y, Z, a, b, c, . . . , x, y, z

El menor ——————————→ El mayor

De hecho, cuando BASIC compara dos caracteres, realmente compara el código ASCII de dichos caracteres.

Los ejemplos, a continuación, explican la forma en que el BASIC realiza las operaciones de relación en las diferentes cadenas:

Expresión	Resultado
"WIDGET" > "WIDGETS"	0
"STAX" < "%TAX"	-1
"Cuenta de espacios" > = "Cuenta de espacios"	0

Operadores lógicos

Los operadores lógicos se emplean para trabajar, con aritmética de Boole, en sus operandos. La aritmética de Boole define un juego de relaciones entre dos operandos, cuando los operandos pueden evaluarse como verdadero o falso. Al igual que las operaciones de relación, las operaciones lógicas dan un resultado de verdadero o falso. Las operaciones lógicas, por tanto, se emplean con frecuencia para tomar decisiones en los programas.

Los operadores lógicos BASIC, listados en su orden de prioridad, se indican en la Tabla 4-3. Las acciones de dichos operadores en una expresión se describe mediante las tablas de verdad de la Figura 1-1

El resultado de una operación lógica se represente en BASIC por -1 para verdadero y 0 para falso. Por ejemplo, la expresión

```
INDEX% >25 AND LIMIT=1.789
```

puede ser sustituida por el valor -1 si INDEX% es mayor que 25 y LIMIT es igual a 1.789. Por otra parte, BASIC sustituirá la anterior expresión por el valor 0 cuando la expresión sea falsa. Debe tenerse en cuenta que esta expresión contiene operadores lógicos y de relación. Cuando se de-

termine su valor, las dos expresiones de relación se evalúan en primer lugar. Sus resultados se usan como operandos para el operador lógico AND (Y).

OPERACIONES LOGICAS CON NUMEROS

Los operandos lógicos pueden también ser números. El BASIC convierte los operandos lógicos numéricos en enteros y comprueba que sus valores están dentro del margen de los enteros BASIC (-32767).

La operación lógica se ejecuta con el valor binario (en base dos).

Tabla 4-3. los operadores lógicos

Operador	Función lógica
NOT	Complejo lógico
AND	Conjunción
OR	Disyunción
XOR	DR (O) exclusiva
IMP	Implicación
EQV	Equivalencia

Operador	Tabla de verdad		
NOT	X		NOT X
	V		F
	F		V
AND	X	Y	X AND Y
	V	V	V
	V	F	F
	F	V	F
	F	F	F
OR	X	Y	X OF Y
	V	V	V
	V	F	V
	F	V	V
	F	F	F
XOR	X	Y	X XOR Y
	V	V	F
	V	F	V
	F	V	V
	F	F	F
	X	Y	X IMP Y
	V	V	V
	V	F	F
	F	V	V
	F	F	V
EQV	X	Y	X EQV Y
	V	V	V
	V	F	F
	F	V	F
	F	F	V

Figura 1-1. Tablas de verdad para los operadores lógicos.

equivalente de los operandos. A continuación, se muestran unos ejemplos de números decimales en forma binaria.

Número decimal	Forma binaria
0	0
1	1
2	10
31	11111

Un número binario consta de "unos" y "ceros" dispuestos en una determinada secuencia, es decir, una serie de valores "verdadero" y "falso".

Los operadores binarios tratan la secuencia de "unos" y "ceros" (o secuencia de bits) de un número binario de forma similar a la empleada por los operadores de relación cuando trabajan con operandos de cadenas. En otras palabras, una operación lógica se hace con cada par de bits en la misma posición de bit de los dos operandos. El resultado de cada operación, por separado, se usa posteriormente para formar el resultado. Por ejemplo:

```
250 AND 28 = 11111010 AND 00011100
           = 00011000
           = 24
```

Concatenación

El operador + se emplea como operador de concatenación con operandos de cadena. Por concatenación se entiende, sencillamente, que las cadenas de cada lado del signo + se unen para formar una cadena más larga. De una expresión de concatenación pueden ser parte más de dos cadenas. Por ejemplo:

```
10 BOILERPLATE1$="Considerando que"
20 BOILERPLATE2$="esta interesado solamente por la paz y la
                  justicia..."
30 PRINT BOILERPLATE1$+"Herbert P. Jones"
         +BOILERPLATE2$
"Considerando que Herbert P. Jones esta interesado solamente por
          la paz y la justicia...
```

FUNCIONES DE BASIC

Una función BASIC se emplea para desarrollar cálculos numéricos, o de cadena, o para controlar u obtener información sobre el hardware. Las funciones suelen operar sobre argumentos, que son expresiones especificadas para cada función. Con determinadas funciones no es necesario el argumento.

Existen tres clases de funciones empleadas por el cálculo: funciones numéricas, funciones de cadena y funciones definidas por el usuario.

Funciones numéricas

Las funciones numéricas nos proporcionan resultados numéricos. El BASIC dispone de varias funciones predefinidas. La siguiente Tabla indica un conjunto de funciones en BASIC que proporcionan resultados numéricos.

FUNCIONES NUMERICAS MATEMATICAS

Muchas de las funciones numéricas empleadas en BASIC son las normalmen

te utilizadas en matemáticas. Por ejemplo:

PRINT SIN(ANGLE)	Visualiza el seno de la variable ANGLE (ANGULO), con el ángulo en radianes.
PRINT ABS(TEST#)	Visualiza el valor absoluto de la variable TEST#.

Tabla de funciones numéricas

Función	Resultado
	Aritmético
ABS(x)	Valor absoluto de x
ATN(x)	Arc. tangente (en radianes) de x
CDBL(x)	Converte x en un número de doble precisión
CINT(x)	Convierte x en un entero por redondeo
COS(x)	Coseno de x en radianes
CSNG(x)	Convierte a x en un número de precisión sencilla
EXP(x)	Eleva e a la potencia x
FIX(x)	Redondea x al entero inferior
INT(x)	El mayor entero menor o igual a x
LOG(x)	Logaritmo natural de x
RND(x)	Número aleatorio
SIN(x)	Seno de x, x en radianes
SQR(x)	Raiz cuadrada de x
TAN(x)	Tangente de x, x en radianes
	Relacionado con las cadenas
ASC(x$)	Código ASCII para el primer carácter en x$
CVI(x$) CVS(x$) CVD(x$)	Convierte x$ en un número entero, sencilla o doble precisión
INSTR(n, x$, y$)	Posiciona la primera ocurrencia de y$ en x$ comenzando en la posición n.
LEN(x$)	Longitud de x$.

FUNCIONES NUMERICAS RELACIONADAS EN CADENA

Existen también determinadas funciones numéricas BASIC que operan en cadenas, pero que dan resultados numéricos. Por ejemplo:

PRINT LEN(FIRSTLINE$)	Imprime la longitud de caracteres de la cadena FISTLINE$.

Funciones de cadena

BASIC tiene varias funciones que proporcionan como resultado, una cadena. En la Tabla siguiente se indican las funciones de cadena BASIC

FUNCIONES DE CADENA QUE OPERAN EN CADENAS

Algunas funciones de cadena operan en las cadenas. Por ejemplo:

PRINT LEFT$(NAME$(Q), 10)	Visualiza los diez caracteres situados más a la izquierda del elemento de matriz de cadena NAME(Q)

Las funciones del tipo LEFT$ se emplean para extraer partes epecificadas de una cadena. Funciones similares a LEFT$ son MID$ y RIGHT$. Otra función muy útil de cadena es STR$. Un ejemplo de su empleo es el siguiente:

PRINT STR$(N)	Muestra en pantalla la representación de la cadena de la variable numérica N por ejemplo, si N es igual a 4923, entonces STR$(N) igualaría la cadena "4293").

FUNCIONES DE CADENA QUE OPERAN CON NUMEROS

Otras funciones de cadena utilizan entradas numéricas y proporcionan resultados en cadena.

PRINT SPACE$(PADDING)	Imprime una cadena de espacios cuya longitud viene especificada por el valor de la variable PADDING.

Tabla de funciones de cadena

Función	Resultado
LEFT$(x$, n)	Los caracteres más a la izquierda de n$
MID$(x$, n, m)	m caracteres, comenzando por x$ en la posición n
RIGHT$(x$, n)	Los n caracteres más a la derecha de x$.
SPACE$(n)	n espacios
STRING$(n, x$)	El primer carácter de x$, repetido n veces.
STRING$(n, m)	Carácter con valor m, en ASCII, repetido n veces
STR$(x)	Convierte x en un valor de cadena
TAB(n)	Tabula a la posición n en una sentencia PRINT o LPRINT
TIME$	Proporciona la hora de sistema
DATE$	Proporciona la fecha del sistema

Funciones definidas por el usuario

Es posible definir las propias funciones del usuario con la sentencia DEF FN, cuyo formato es el siguiente:

DEF FNnombre (argumento,argumento, ...) =definición.

El parámetro "nombre" puede ser cualquier nombre de variable válido que se elija. Una vez que una función haya sido definida, se la puede llamar mediante las letras FN, precedidas por el nombre de dicha función. Por ejemplo, el nombre de la función llamada REVERSE, podría referenciarse de la forma siguiente:

```
300 T=FNREVERSE(X)
```

Los parámetros "argumentos" son los argumentos para la función que es necesario definir explícitamente siempre que se llame la función. La definición es una función que especifica cómo opera la función con dichos argumentos.

Por ejemplo, supongamos que se desea tener una función que calcule el área de un círculo, de un valor de radio dado. Supongamos también que la variable PI# se asignó antes de definir la función. Dicha función, a la que denominaremos FNAREA, puede emplearse de la forma siguiente:

```
20 DEF FNAREA(X)=PI#*X*X
.
.
.
450 SURFAREA=HEIGHT*2*PI#*RAD*2*FNAREA(RAD)
.
.
```

Cuando FNAREA es solicitada. BASIC busca la definición y sustituye la variable "artificial o ficticia" (en este caso, X) por el argumento especificado en la función llamada (en este caso, la variable RAD).

Las funciones definidas por el usuario pueden proporcionar un valor numérico o de cadena. El nombre de una función dada, no obstante, ha de ser un nombre de variable válido para el tipo de variable que proporciona. Por ejemplo, si la función proporciona un resultado en forma de cadena, el nombre de la función ha de terminar con $. En caso contrario, obtendrá como respuesta BASIC la señal de error. (Error de concordancia).

Una función definida por el usuario puede llamarse a sí misma (esto es, puede ser recurrente), mientras que exista un mecanismo que evite el que esto ocurra demasiado número de veces.

SENTENCIAS BASIC

Las sentencias son los elementos básicos de la construcción de un programa. Una sentencia BASIC se compone de una palabra especial (palabra clave BASIC) y, con frecuencia, una expresión o función. Las sentencias se emplean para introducir datos, llevar a cabo cálculos, manipular cadenas y números y proporcionar resultados a la salida. Las sentencias también toman decisiones, basadas en los cálculos anteriores y en los valores de las variables y expresiones. Otra función muy importante de las sentencias BASIC es controlar la ejecución de los programas.

La sentencia BASIC se denomina, con frecuencia, una orden. La diferencia fundamental entre una sentencia y una orden es que aquéllas están constituídas, de forma natural, por instrucciones usadas en modo indirecto. Las órdenes, por el contrario, se emplean en modo directo. No obstante lo anterior, la mayor parte de las sentencias y órdenes pueden usarse en uno u otro contexto.

Sentencias de comentarios

Uno de los más importantes elementos de un programa es lo que se denomina observaciones o comentarios. Las observaciones no se ejecutan en el computador; antes bien, ayudan a antender el desarrollo de las diferentes acciones.

Las observaciones pueden escribirse de dos formas diferentes. La palabra clave REM o el carácter apóstrofe (') han de preceder al texto de la observación. Por ejemplo, las siguientes líneas son equivalentes:

```
340 REM Comienza la clasificación con W=10

340 ' Comienza la clasificación con W=10
```

Las observaciones pueden constituir un apéndice al final de la línea de programa. Por ejemplo:

```
340 W=10 ' Comienza la clasificación con W=10
```

Cuando se incluyan observaciones en el programa es aconsejable ser conciso y completo al hacer la correspondiente anotación.

Sentencias de asignación

Un programa BASIC puede obtener datos de dos sitios diferentes: de fuentes externas, como es el caso de un operador o de un dispositivo periférico, o del propio programa. La fuente considerada en primer lugar que se sitúa bajo la clasificación general de entrada/salida (E/S) se expondrá en las páginas siguientes. Veamos, pues, ahora, cómo es posible tener acceso o especificar los datos dentro de un programa, mediante el uso de lo que denominamos sentencias de asignación.

Sentencia LET

La forma más simple de definir el valor de una variable es mediante la sentencia LET, que asigna un valor numérico, o de cadena, a una variable.

Un ejemplo de la sentencia LET es:

```
20 LET ROTAXO%=300 ' Inicializar ROTAXO%
```

La palabra reservada LET es opcional, por lo que la misma sentencia puede escribirse como:

```
20 ROTAXO%=300 ' Inicializar ROTAXO%
```

Para las sentencias LET numéricas, si la precisión de la variable es diferente que la precisión de la constante, la constante se convierte a la precisión de la variable. Por ejemplo, la sentencia:

```
230 I%=3.5411
```

asignará el valor 4 a I%.

En las sentencias LET pueden usarse expresiones. Por ejemplo:

```
340 U#=T/3+FUDGEFACTOR
350 FLAG=(NOT A AND B) OR (C AND D)
```

Las variables de cadena también pueden definirse con sentencias LET. Por

ejemplo:

```
450 PREAMBLE$="Cuatro puntos hace siete años"
```

Si una variable numérica se asigna a una constante de cadena, o viceversa, se obtiene una respuesta BASIC de "Type Mismatch" (error de concordancia).

SENTENCIAS DATA Y READ

Si un programa emplea variables a cuyos valores se asignan constantes, pueden emplearse las sentencias DATA (DATOS) y READ (LEER). Las sentencias DATA establecen una lista de constantes (numéricas o de cadena) para uso con las sentencias READ. Por ejemplo, las siguientes sentencias definen un listado como el que se presente a continuación.

```
130 DATA 2.1, 3.2, 4.3, 5.4, 6.5, 7.6, 8.7
140 DATA 9.8, 0.9, 1
```

Estas sentencias dan origen a esta lista

```
2.1
3.2
4.3
5.4
6.5
7.6
8.7
9.8
0.9
1
```

Las sentencias READ asignan constantes en sentencias DATA a variables sobre una base de una en una; cuando cada variable en una sentencia READ recibe un valor procedente de la lista DATA, BASIC mueve un puntero, hacia abajo, una posición en dicha lista. Por tanto, la siguiente variable READ que encuentra el BASIC, ya sea en la misma sentencia READ o en otra, recibe el siguiente valor de la lista. Por ejemplo, supongamos que las siguientes sentencias READ se semplean con las dos sentencias DATA anteriores:

```
200 READ T
210 READ Z
```

el tipo de constante en una sentencia DATA y el tipo de variable que se asigna en una sentencia READ han de estar en consonancia. Por ejemplo:

```
40 READ R, R$
    .
    .
    .
400 DATA 594, Ph constante
```

A la sentencia REAd asignará el valor 594 a R y la cadena "Ph constante" a R$

SENTENCIAS DE FLUJO DE PROGRAMA

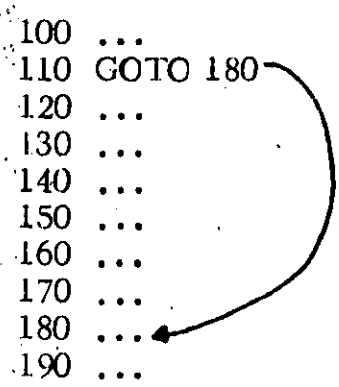
En circunstancias normales, el BASIC ejecuta las líneas de un programa de acuerdo con los respectivos números de línea. Este orden puede cambiarse mediante el uso de lo que se conoce como sentencias de flujo de programa.

Cuando el BASIC encuentra una sentencia de flujo de programa, la ejecución saltará (o se bifurcará) a otra línea, no en secuencia, si se dan ciertas condiciones. Estas condiciones se especifican en las sentencias de flujo del programa. Dichas sentencias se emplean con frecuencia para:

- Ejecutar una parte de un programa muchas veces con diferentes datos
  Ejecutar diferentes partes de un programa, de acuerdo con ciertas condiciones.
- Utilizar el mismo subprograma muchas veces, sin necesidad de escribirlo de nuevo.

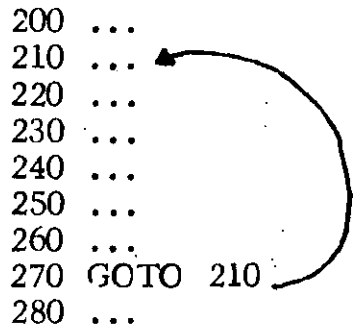
Sentencia GOTO

La sentencia GOTO hace que el programa BASIC se bifurque a la línea del número indicado, cada vez que se ejecute la sentencia; esto es, fuerza una bifurcación incondicional. Por ejemplo:



Debido al GOTO de la línea 110, la ejecución saltará desde la línea 110 a la 180.

La sentencia GOTO también puede ser causa de saltos hacia atrás, como se indica a continuación:



Sentencia ON-GOTO

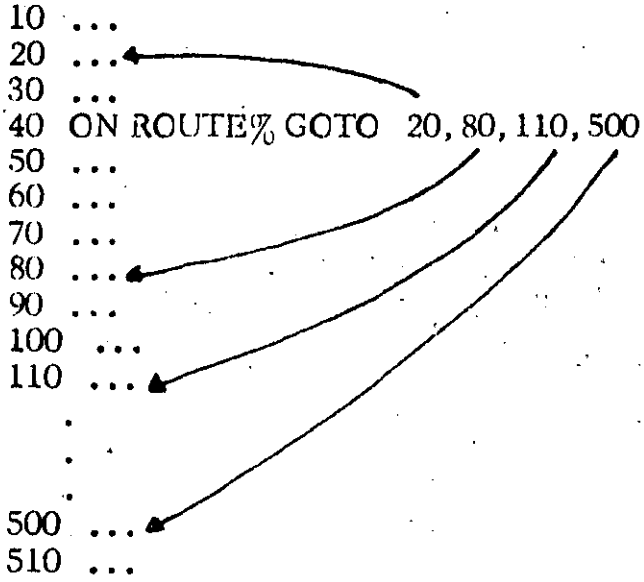
Las sentencias ON-GOTO le permiten hacer una bifurcación a una o varias líneas de acuerdo con el valor de la expresión numérica de la sentencia.

El fomato de la sentencia ON-GOTO es el siguiente:

ON expresión GOTO Línea , Línea ...

El progrma BASIC evalúa la expresión, redondea el resultado a un entero, si fuera necesario, y bifurca hasta la línea cuya posición en la sentencia ON-GOTO concuerda con el valor de la expresión. Por ejemplo, si la expresión es igual a 5, la bifurcación se ejecutará a la línea 5 de la lista. Si el resultado es mayor que el número de líneas o es igual a cero, la ejecución continúa con la siguiente línea del programa. Una señal de error del tipo -- "Illegal Function Call" (llamada a función no correcta) será la respuesta si la expresión es negativa o mayor que 255.

El flujo de programa a partir de una sentencia ON-GOTO se ilustra de la manera siguiente:



Bucles de programa

Un bucle de programa permite usar la misma sección de un programa un gran número de veces, utilizando diferentes datos. Por ejemplo, si se desea cargar una matriz compuesta por 100 elementos con valores comprendidos entre 1 y 100, puede escribirse cien sentencias de asignación.

```
10  COUNT(0)=1
20  COUNT(1)=2
      .
      .
      .
990 COUNT(98)=99
1000 COINT(99)=100
```

Un método mejor consiste en crear un bucle (lazo) usando las sentencias FOR (PARA) y NEXT (SIGUIENTE). Estas sentencias dan origen a que la parte del programa entre FOR y NEXT se ejecuten un cierto número de veces de acuerdo con la variable, cuyo valor se incremente o decremente en cada paso del lazo. Por ejemplo, rellenar la matriz, a la que se hacía mención anteriormente y escribirla en el sistema de presentación, puede intentarse con el siguiente programa:

```
10  DIM COUNT(100)
20  FOR I=0 TO 99
30  COUNT(I)=I+1
40  PRINT "ESTO ES PERMISO" COUNT(I)
50  NEXT
60  END
```

Las líneas 20, 30, 40 y 50 forman el lazo. La sentencia FOR origina que el índice variable I se incremente en una unidad en la sentencia siguiente, cada vez que se pasa el lazo, con I igual a 0 la primera vez. En la próxima pasada, I se incrementa en una unidad. Cuando I es igual a 99, el límite especificado en la sentencia FOR, el lazo se ejecuta por última vez y la ejecución del programa va a la línea 60.

El valor de la variable de índice se compara con el margen dado a la sentencia FOR, una vez que dicho índice se ha incrementado. La forma general de esta sentencia es la siguiente:

```
FOR variable=x TO y STEP z
NEXT variable , variable ...
```

El parámetro opcional STEP z da origen a que la variable de índice (I en

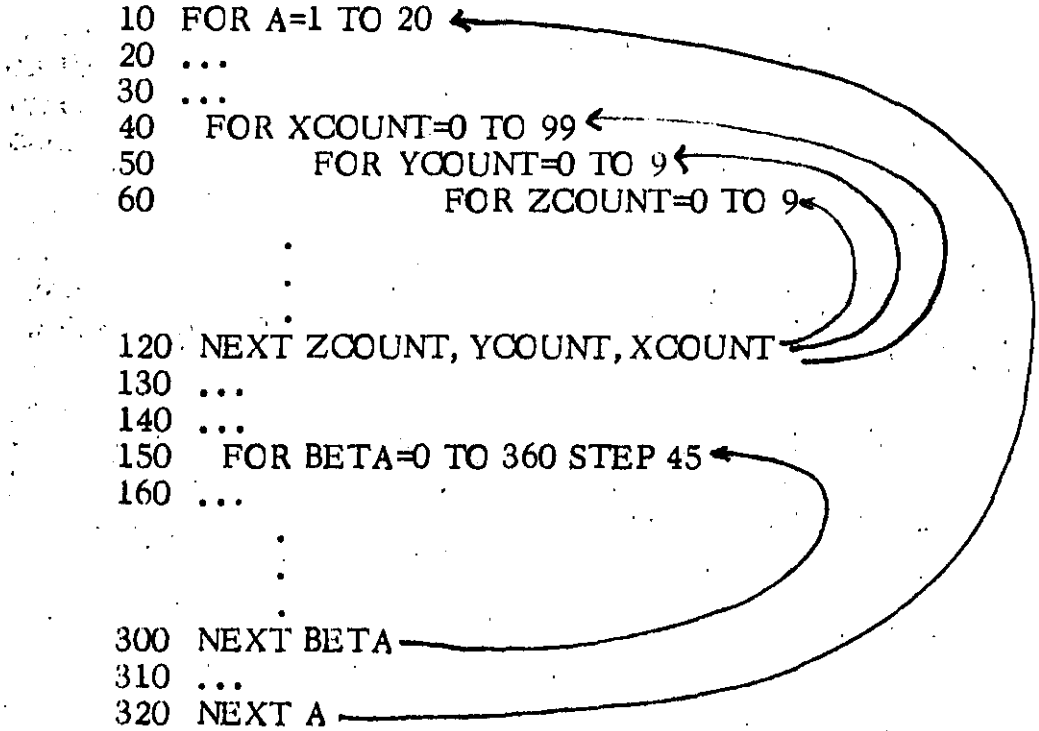
el ejemplo anterior) se incremente en el valor de z. Si z no se incluye, el incremento pasa a 1. El valor de cada paso puede ser positivo o negativo. Si el paso es positivo, el lazo termina cuando el valor del índice es mayor que el valor de y. Si el paso es negativo, el lazo termina cuando el valor del índice es menor que el valor de y.

En el caso de emplear un paso negativo, debe asegurarse que el valor de x en la sentencia FOR no sea menor o igual que el valor de y. En caso contrario, el lazo se ejecutará una sola vez. Del mismo modo, el lazo se ejecutará una vez en el caso de que el paso sea positivo y x sea mayor que y.

Para ver la forma en que operan las sentencias FOR y NEXT, repita el ejemplo anterior, con diferentes valores de paso. Aunque pueden usarse números reales como valores de la sentencia FOR, los números enteros aumentan la velocidad de ejecución.

Lazos anidados encajados

Los lazos creados con las sentencias FOR y NEXT pueden situarse dentro de otros lazos para crear lazos encajados o anidados. Por ejemplo:



Debe notarse que la relación entre varios lazos se ilustra mediante flechas, Por ejemplo, se indica que cada vez que se ejecuta el lazo XCOUNT, el lazo YCOUNT se ejecuta diez veces, mientras que el lazo ZCOUNT se ejecuta cien veces.

Los programas BASIC permiten cierta flexibilidad con las sentencias NEXT en los lazos anidados . Por ejemplo, la línea 120,

```
120 NEXT ZCOUNT, YCOUNT, XCOUNT
```

podría haber sido sustituida por

```
120 NEXT ZCOUNT
121 NEXT YCOUNT
122 NEXT XCOUNT
```

o simplemente por:

```
120 NEXT
121 NEXT
122 NEXT
```

Sentencias GOSUB y RETURN

Si la misma tarea debe ser realizada en distintos puntos del programa, GOSUB es una sentencia efectiva a usar. Cada vez que se encuentra una sentencia GOSUB, BASIC conserva el número de línea de la sentencia. El programa se bifurca entonces hacia la línea especificada en GOSUB.

Cuando se ejecuta una sentencia RETURN, BASIC busca el número de línea conservado y vuelve a la línea siguiente a GOSUB. La línea especificada en la sentencia GOSUB es el principio de lo que se refiere como una subrutina. Una subrutina es llamada desde el programa principal con la sentencia GOSUB. La ejecución de la subrutina finaliza cuando BASIC ejecuta RETURN. Todas las subrutinas deben tener una sentencia RETURN para que BASIC pueda reconocer el final de la subrutina. De otro modo, la ejecución continuará más allá del final de la subrutina.

Para ilustrar el uso de subrutinas, procese el siguiente programa:

```
10 A$="PRIMERO"
20 B$="      "
30 COUNT=1
40 GOSUB 100
50 A$="SEGUNDO"
60 B$="*  *"
70 COUNT=2
80 GOSUB 100
90 END
100 '****ESTO ES LA SUBRUTINA PRINCIPAL****
110 PRINT "ESTE ES LA" A$ "EJECUCION DE LA SUBRUTINA
          PRINCIPAL"
120 GOSUB 160
```

```
130 COUNT=COUNT + 3
140 GOSUB 160
150 PRINT "FINAL DE "A$" EJECUCION"
160 RETURN
170' ****ESTA ES LA SUBRUTINA ANIDADA****
180 BEEP
190 PRINT"    ESTA ES LA SUBRUTINA ANIDAD"B$
200 PRINT "CUENTA=" COUNT
210 RETURN
```

SENTENCIA GOSUB COMPUTADA

La sentencia GOSUB computada, ON-GOSUB, trabaja del mismo modo que la sentencia ON-GOTO. La única diferencia es que la sentencia ON-GOSUB hará que el BASIC llame una subrutina. El ejemplo siguiente ilustra la sentencia ON-GOSUB:

```
200  ON DIRECT GOSUB 1000, 1100, 1200
210  ...
      .
      .
1000 ' SUBROUTINE PARA DIRECT=1
      .
      .
1050  RETURN
1100  'SUBROUTINE PARA DIRECT=2
      .
      .
1150  RETURN
1200  ' SUBROUTINE PARA DIRECT=3
      .
      .
1250  RETURN
```

Sentencia IF-THEN-ELSE

Las sentencias ON-GOTO y ON-GOSUB son ejemplos de ejecución condicional, ya que la acción tomada por estas sentencias es dependiente del valor de una expresión. Otro tipo de sentencia que hace una ejecución condicional es la sentencia IF-THEN-ELSE. La sentencia IF-THEN-ELSE, como hemos visto en ejemplos anteriores, se utiliza para saltar a una línea del programa específica, si se cumple una condición especificada.

Esta sentencia, cuyo formato es:

IF expresión [,] THEN clausula [[,] ELSE clausula]

Observe el espacio extra entre -5 y 90. Observe también que todos los números tienen un espacio añadido automáticamente después de ellos cuando son objeto de salida. Además, los números positivos tienen un espacio añadido delante de ellos y los números negativos tienen un signo añadido.

La separación de expresiones con una coma, por el contrario, hace que la sentencia PRINT tabule su salida. Por ejemplo:

```
PRINT X, Y, Z, "Buffo"
```

se visualizará como

```
43.2        -5          90          Buffo
```

Como puede ver, BASIC divide la línea impresa en varias zonas de impresión.

Sentencia INPUT

La sentencia INPUT puede hacer dos cosas. Primero, acepta datos del teclado y los asigna a las variables en la sentencia INPUT. Segundo, puede hacer salir un mensaje al operador. El formato de la sentencia INPUT es como sigue:

```
INPUT [;] ["digite los datos";]variable [, variable] ..
```

Si el texto de la setencia INPUT se omite sólo da salida a un signo de interrogación y espera la entrada de los datos.

Cada elemento de entrada introducido por el teclado debe corresponder al tipo de variable que se le asigne. Observe, sin embargo, que las entradas de cadenas no necesitan encerrarse entre comillas. Además, los elementos de entrada deben ser separados por comas.

ACCION DE SALIDA A LA PANTALLA

La sentencia PRINT hará simplemente que se imprima un elemento de dato a medida que se escribe. La sentencia PRINT USING formateará primero los datos de acuerdo con las pseudoinstrucciones dentro de la sentencia para permitirle visualizar fácilmente datos en tablas u otras estructuras organizadas.

Para poder inhibir el retorno de carr/avance de línea que se suele añadir a las sentencias PRINT y PRINT USING, necesita finalizar estas sentencias con punto y coma. Para verlo, pruebe con lo siguiente:

```
LOCATE 10, 10:PRINT"ESTA ES LA FILA 10"ESTE ES EL PRINCI
PIO DE LA FILA 11"
```

Ahora, desplazar de nuevo el cursor a la parte superior de la pantalla e introducir.

```
LOCATE 15, 10:PRINT "ESTA ES LA FILA 15";:PRINT " Y ESTA ES
TODAVIA LA FILA 15"
```

La sentencia PRINT USING formatea sus datos de acuerdo con una cadena de máscara que está constituida por caracteres de formateado. Estos caracteres especifican cómo han de visualizarse los datos. El formato para la sentencia PRINT USING es:

```
PRINT USING X$;[lista de expresiones] ;
```

La constante de cadena o variable X$ es la cadena de máscara. Los caracteres de formateado en X$ pueden especificar el número de caracteres impresos en una expresión de cadena, el número de dígitos a la izquierda y a la derecha del lugar decimal y si los datos van precedidos, o no, por un $ para expresiones numéricas. Por ahora, consideraremos como actúan unos pocos de los caracteres de formateado.

Consideremos la sentencia siguiente:

```
PRINT USING"\   \";T$, Y$, U$
```

La cadena de máscara está constituida, en este caso, por dos barras separadas por cuatro espacios. Esta secuencia sólo visualiza los seis primeros caracteres de las cadenas de salida. Por ejemplo, sí:

```
T$="Fee"
Y$="Fi"
U$="Fo"
```

la anterior sentencia PRINT USING producirá lo siguiente:

```
Fee    Fi      Fo
```

Por el contrario, si:

```
T$="Liberte"
Y$="Egalite"
U$="Fraternite"
```

la anterior sentencia PRINT USING daría lugar a:

```
LibertEgalitFrater
```

Para ver cómo PRINT USING puede utilizarse con números, consideremos este problema. Supongamos que necesita obtener la salida de una lista de números de longitudes variables (o precisiones) en tres columnas. Los puntos decimales de los números han de estar todos alineados y los números en cada columna deben visualizarse en formatos diferentes. El siguiente programa indica cómo se hace:

```
10 DATA 123.333, 2, 33.53, 6376, 4, 53.3254, 45, 4, 5.084
20 FOR I=0 TO 8
30 READ A(I)
40 NEXT I
50 FOR I=0 TO 8
60 PRINT USING "####.##";A(I);
70 I=I + 1
80 PRINT USING "        ##";A(I);
90 I = I + 1
100 PRINT USING "  ###.###";A(I)
110 NEXT I
120 END
```

25

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA COMPUTACION Y PROGRAMACION ELECTRONICA

PROCESO DISTRIBUIDO

- CONCEPTOS BASICOS
- REDES LOCALES
- SERVICIOS DE TRANSMISION DE DATOS

ING. FRANCISCO LLAMAS

10 OCTUBRE, 1984

LA RED DE COMPUTADORAS

CONCEPTOS BASICOS

RED.- Una red de computadoras puede ser definida como un grupo interconectado de sistemas de cómputo que se comunican uno con el otro y comparten recursos. Los sistemas de cómputo pueden ser computadoras anfitrionas independientes o microcomputadoras remotas dependientes. Ellos pueden proveer recursos, usar recursos. En una red de computadoras, cada sistema de cómputo puede operar en un modo local bajo su propio sistema operativo, o puede participar en la actividad de la red bajo el control de un supervisor de la red en un nivel más alto.

En una red de computadoras, la transferencia de información entre las computadoras debe de ser automática y es responsabilidad de la subred de comunicaciones . Esta última generalmente consiste de líneas telefónicas y dispositivos de comunicación tales como modems, multiplexores, y procesadores de comunicación. El software ejecutado en los procesadores de comunicación controla la subred, suministrando confiabilidad y rapidez en la transfencia de mensajes entre los integrantes de la red.

En adición a los sistemas de cómputo y a la subred de comunicaciones, una red de computadoras puede incluir una amplia variedad de terminales remotas. Las terminales proveen a los usuarios el acceso a los diferentes sistemas de cómputo en la red.

En breve, una red de computadoras esta compuesta de una o más computadoras, un número de terminales, y una red de comunicaciones que conecta entre si a las computadoras y a las terminales. La red de comunicaciones esta compuesta de facilidades de transmisión o circuitos y nodos. Los nodos pueden consistir desde una cantidad pequeña de lógica de hardware hasta una computadora programable.

CIRUITOS.- El término circuito usualmente se refiere a la ruta lógica o física de un punto a otro punto. Pero en el caso de una red, el medio del circuito generalmente es una línea telefónica.
Los circuitos físicos se establecen por conexiones físicas a la entrada y salida de la ruta en cada punto de conmutación. Los circuitos virtualmente se establecen usando tablas construidas en una computadora y que conmutan para conectar un puerto de entrada a un puerto de salida.

Un circuito es diseñado para llevar información ya sea en forma analógica o digital. Las señales son atenuadas conforme se propagan a través del circuito, y deben ser amplificadas en intervalos regulares para compensar la atenuación que sufren a lo largo de la línea de transmisión.

CANALES.- Un canal es definido como una ruta para la transmisión electrica de datos entre dos computadoras o dos terminales. El propósito de un canal es llevar la información de una localidad a otra.

Un canal incorpora el circuito y los modems (en el caso de que estos últimos sean necesarios)- y posiblemente otro equipo.

Generalmente el canal involucrado, es considerado el elemento básico de las comunicaciones. Importantes parámetros de un canal son :

1. Porcentaje de datos
2. Limitaciones direccionales
3. Características de error
4. Características de retraso

El porcentaje de datos de un canal analógico es usualmente medido en bits por segundo (bps) o bauds. (Baud es el número de elementos de código por segundo, y un elemento de código puede contener uno o más bits). Hay una relación directa entre la capacidad de transmisión de un canal, en bps, y el ancho de banda del canal.

Las limitaciones direccionales de un canal son determinadas por el equipo del canal, tales como modems y líneas de control. Hay tres tipos de canales, de acuerdo a la capacidad direccional, son :

1. SIMPLEX — Permite la transmisión de información en una sola dirección.

2. HALF DUPLEX — Permite la transmisión de informacion en ambas direcciones, pero en una sola direción a la vez.

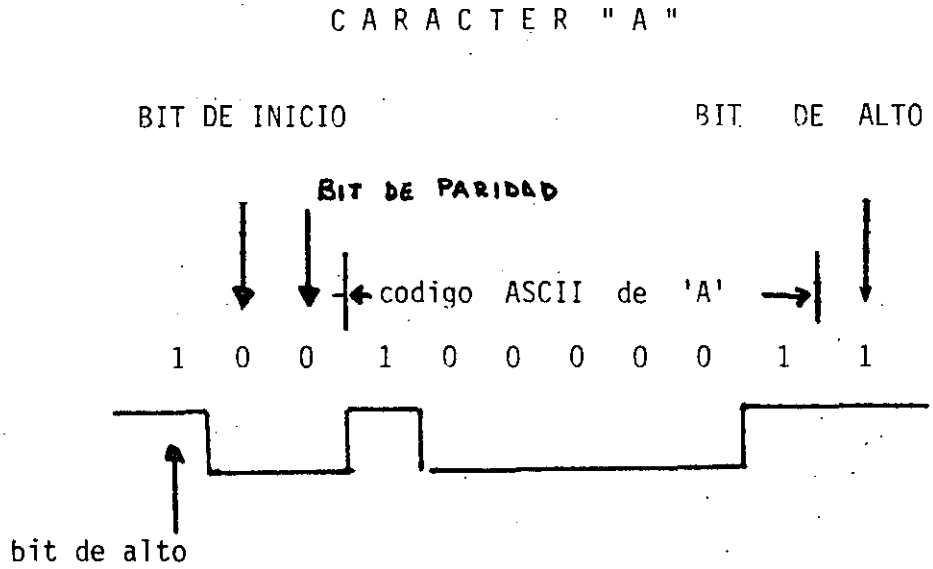
3. FULL DUPLEX — Permite la transmisión simultánea en ambas direcciones.

Modos de Transmisión.- La comunicación de datos involucra el movimiento de una cadena de datos binarios desde un lugar a otro.

Los datos binarios representan informacion: Números, Letras y Símbolos. Cualquier código puede ser usado para representar la información. Con el objeto de que la información transmitida pueda ser recuperada en su destino, debe haber un método de sincronización para los dispositivos receptores y transmisores. Los dispositivos pueden ser sincronizados usando la comunicación de datos en un módo síncrono o asincrono.

Transmisión ASINCRONA. En la transmisión asíncronoa, un bit de inicio es insertado antes del patron de bits que representa cada caracter, y uno o más bits de alto son insertados despues del caracter. Los bits de inicio/alto sincronizan al receptor con el transmisor al inicio de cada caracter.

Los bits de alto son enviados contínuamente entre los caracteres, y la ocurrencia de un bit de inicio indica al receptor que los siguientes ocho -- bits (asumiendo código ASCII) forman el caracter. Al menos un bit de - alto separa dos caracteres consecutivos. En la figura se muestra la letra A en código ASCII con los bits de inicio/alto insertados.



La transmisión asíncrona no posee una buena capacidad de corrección y detección de errores

Los dispositivos asincronos son generalmente mas baratos que los dispositivos sincronos porque necesitan buffers pequeños y menos lógica. Los sistemas interactivos con entrada por teclado (las terminales) utilizan la transmisión asincrona en forma exclusiva para la comunicación con la computadora. La utilización de la línea es muy baja en estos sistemas (típicamente menos del uno por ciento), y un rendimiento tan solo del 72 % en la información transmitida (8 bits utiles de 11 transmitidos).

Transmisión SINCRONA. La transmisión sincrona envia una cuerda de caracteres continua sin los bits de inicio y alto entre los caracteres. La cadena de caracteres es dividida en bloques, y todos los bits en el bloque son transmitidos en intervalos de tiempo iguales.

El receptor debe muestrear la cadena de bits en intervalos precisos exactamente iguales que los intervalos de la señal de reloj del transmisor. Unos osciladores mantienen los relojes de ambos extremos de la línea de comunicación en sincronia. El oscilador en el receptor es ajustado automáticamente para que permanezca en fase con el transmisor. La estación que está transmitiendo introduce al inicio de cada bloque una serie de caracteres de sincronia. Esos caracteres de sincronización identifican el inicio del bloque y sincronizan a los osciladores.

La longitud de los bloques usada en la transmisión sincrona varia desde unos pocos caracteres a varios cientos de caracteres. El bloque siempre empieza con un patron de sincronización y generalmente termina con un patron o caracter de chequeo de errores. El bloque puede ser de longitud fija o de longitud variable. Generalmente hay uno o mas caracteres de sincronia en el bloque, como se muestra en la figura.

caracteres de sincronizacion | informacion | campo de chequeo de errores

Los caracteres de control sincronizan a los dispositivos de transmisión y recepción, e indentifican los campos dentro del bloque. El bloque es ensamblado dentro de un buffer; asi el tamaño máximo del bloque es el tamaño del buffer. Las transmisiones en modo sincrono producen un aumento en la utilización de la línea y un porcentaje bajo de errores, si bien esto incrementa el costo del equipo.

LA DETECCION DE ERRORES

El objetivo de un sistema de comunicación de datos es producir en el receptor la misma cadena de bits que fue enviada por el transmisor. Sin embargo, el ruido en las líneas puede causar que ocurran errores. En caso de error, primeramente es necesario detectar que el error ha ocurrido, y entonces tomar alguna acción para recuperarse del error.

VERTICAL REDUNDANCY CHECK. Muchos metodos de detección de errores son simples pero son lo suficientemente utiles para alcanzar porcentajes de detección de errores razonables, uno de tales metodos es el uso de un bit de pa-

ridad a nivel de cada caracter, suele denominarse detección vertical (Vertical redundancy check VRC). Este metodo agrega un bit (1 o 0) al caracter que ha sido transmitido para que el número de 1's, incluyendo el bit de paridad, sea par (si la paridad par es usada) o impar (si se usa paridad impar).. Por ejemplo, el codigo ASCII para la A es x1000001 donde x es el bit de paridad. El bit de paridad debera ser 0 para una paridad par ---- (01000001) y 1 para paridad impar (11000001).

El receptor cuenta el número de bits en cada caracter que se recibe y detecta un error si hay un numero impar de 1's para una paridad par o un numero par de 1's para una paridad impar. VRC detecta los errores solamente si un numero impar de bits ha sido alterado. Este metodo de detección de errores es el mas utilizado en la transmisión asincrona.

LONGITUD REDUNDANCY CHECK. En las transmisiones sincronas, es mejor usar la redundancia dentro del bloque de caracteres, que dentro de un solo caracter. Un bloque de muchos caracteres sera seguido por uno o mas caracteres de chequeo de error. Longitud redundancy check (LRC) es uno de tales metodos. LRC usa un bit de paridad para cada nivel de bit en los caracteres que son transmitidos en el bloque (llamamos nivel de bit a todos los bits 1s, 2s, 3s etc. de los caracteres que componen el bloque la paridad de todos los niveles de bit forman el caracter de chequeo de error). Al igual que VRC, la paridad par o impar puede ser usada. LRC detecta errores en los cuales un número impar de bits en un mismo nivel fueron cambiados. La combinación de VRC y LRC conduce a una mayor probabilidad de detectar errores.

CYCLIC REDUNDANCY CHECK. Quizas el metodo mas eficiente de detectar errores

es el de detección ciclica (cyclic redundancy check CRC). Los caracteres - de chequeo de errores son generados dividiendo los caracteres que incluye el bloque de datos por un modelo de bits de referencia o un polinomio cociente (generalmente de grado 16). La aritmetica de módulo 2 es usada en el proceso de división, y el residuo es adicionado al bloque como caracteres de chequeo de errores. El modelo de bits de referencia es elegido de modo que haya una alta probabilidad de detectar errores. En el dispositivo receptor, el mensaje recibido es dividido por el mismo polinomio, es posible detectar errores de uno o mas bits.

RECUPERACION DE ERRORES. Una vez que un error ha sido detectado, hay varias opciones para recuperarlo. Muchos sistemas, tales como algunos de proposito general y tiempo compartido, simplemente ignoran los errores. Los sistemas de tiempo compartido generalmente usan transmisiones asincronas con espacios de tiempo relativamente grandes entre caracteres; por lo tanto es improbable que un pico se de en un caracter y cause error. También, los usuarios mismos introducen más errores que los que ocurren en la línea, pero pueden tomar -- acción inmediata para efectuar las correcciones pertinentes.

El metodo mas comun para recuperar errores es la retransmisión del bloque en error. Un mensaje es enviado de regreso al transmisor para que retransmita el bloque.

DISPOSITIVOS DE COMUNICACION

Los nodos en la subred de comunicaciones son dispositivos de comunicación. - Los propósitos de estos dispositivos son :

1. Permitir que la información digital se transmita básicamente en rede analógicas
2. Reducir costos de la red
3. Controlar los canales y la red
4. Proveer acceso a la red
5. Asegurar la integridad de los datos que son transmitidos a través de la red.

Hay tres categorias principales en los dispositivos de comunicación usados en las redes para satisfacer estos propósitos:

- MODES O DATA SETS
- MULTIPLEXORES y
- PROCESADORES DE COMUNICACION

MODEMS.- Una señal digital puede modularse como una señal electrica de alta frecuencia, llamada protadora, de tal forma que la portadora transmite la información digital. La señal digital puede entonces ser recuperada a partir de la portadora modulandola por filtros electrónicos en el receptor.

Modulación es el proceso de modular la portadora, y demodular es el proceso de reucperar la señal original; de aqui el termino modem (Modulación/ Demodulación) para los dispositivos que convierten las señales a la forma analógica y digital. La portadora es una onda senoidal que tiene tres características :

AMPLITUD, FRECUENCIA Y FASE

Esas tres características proveen tres metodos básicos de modulación. La

amplitud modulada (AM) varia la amplitud de la onda senoidal basada en la señal digital. En frecuencia modulada (FM) frecuencias diferentes son usadas para que la portadora represente los estados de 1 o 0 de la señal digital. La tercera característica, la fase, cambia cuando la señal digital cambia de estado(de 0 a 1 , o de 1 a 0) dando lugar a la modulación en fase (PM).

Varias características importantes deberan ser consideradas cuando se evaluan los modems. Hay modems para transmisión asincrona o sincrona. Los modems asincronos son generalmente usados para tasas de transmisión bajas y son capaces de manejar cualquier tasa. Los modems sincronos son usados para tasas de transmisión alta y operan solo a tasas fijas. Algunos modems sincronos pueden operar en dos o tres tasas diferentes, tales como : 2400, 4800, 9600 y 19200 bps las cuales son seleccionadas por un interruptor manual o automático.

Las conexiones electricas de los modems y de los demás dispositivos digitales generalmente se adhieren a la interfaz estandard RS-232C de la Asociación de Industrias Electricas (EIA).

PROCESADORES DE COMUNICACION

Un procesador de comunicaciones (CP) es una computadora digital programada para ejecutar una o más funciones de control o de procesamiento en las comunicaciones de la red. Las funciones típicas ejecutadas por los procesadores incluyen :

1. Procesamiento Front-End, donde el CP provee una interfaz economica y flexible entre la red de comunicaciones y la computadora anfitriona.

2. Concentración remota de los datos, donde los CPs multiplexan datos de varias lineas de velocidad baja en pocas lineas de alta velocidad.

3. Control de terminales, donde el CP controla un grupo de terminales directamente conectadas a el, y multiflexa los datos de todas las terminales a un canal de alta velocidad.

4. Conmutacion de mensajes, donde el CP almacena los mensajes de las diferentes localidades, comprueba los mensajes, y los retransmite a su destino correcto.

5. Control y procesamiento de la red, donde el CP puede ejecutar una combinación de las funciones anteriores, asi como varias funciones de control de la red.

PROCESADOR FRONT-END (FEP)

Procesador Front End (FEP). procesa los datos en orden para liberar a la anfitriona del gasto de tiempo consumido en actividades asociadas con el control y el formateo de los mensajes. El procesador front-end puede ejecutar cualquiera de las siguientes funciones :

1. El FEP ejercita el control de la línea, estableciendo una conexion logica, controlando la transferencia de los datos, y terminando la conexion logica. El control de la linea también incluye la comprobación de errores y procedimientos de recuperación.

2. El FEP recibe mensajes de un bit a un tiempo, va ensamblando esots bits de caracteres y estos a la vez dentro de mensajes completos listos para el procesador anfitrión. También recibe mensajes del anfitrión, añade los bits o caracteres de control necesarios, y transmite el mensaje al dispositivo destino a la velocidad apropiada de la línea.

3. Frecuentemente la computadora anfitriona usa un código diferente al dispositivo remoto. Por ejemplo, la anfitriona puede usar EBCDIC y el dispositivo remoto puede usar ASCII. El FEP ejecuta la conversión entre el código de la anfitriona y el código de transmisión.

4. El FEP provee espacio en el buffer para la información que va a ser ensamblada, ensambla los mensajes que han recibido desde la anfitriona y que van a ser transmitidos al dispositivo remoto. Los mensajes que estan esperando deben ser añadidos a la cola y enviados a su destino correcto de una manera ordenada. El espacio del buffer puede estar asignado ya sea en memoria principal o en disco.

5. El FEP puede procesar algunas transacciones sin interrumpir a la anfitriona.

6. Puesto que el FEP es una computadora programable, esta puede ser programada para ejecutar otras funciones orientadas a una aplicación específica.

PROCEDIMIENTOS DE CONTROL

Una red de computadoras requiere de un conjunto de procedimientos para - controlar el flujo de datos. Los procedimientos bien diseñados nos aseguran eficiencia, y una operación confiable de la red. Tales procedi-- mientos serían:

1. Detectar los distintos errores ya sea recuperando automáti camente el error o notificando al usuario la ocurrencia -- del error. Los errores podrían incluir datos duplicados, modificados o anulados; y la falla del dispositivo o la - liga.

2. Proveer un lenguaje de control efectivo y simple, que permita al usuario accesar la red y usar los recursos disponi bles.

3. Eficacia en el uso de los recursos de la red tales como -- las ligas de comunicación y los procesadores. Esto puede requerir procedimientos mas sofisticados para controlar el flujo de la información y prevenir el congestionamiento.

4. Prevenir que un dispositivo quede colgado (deadlock) si - es razonablemente posible. Si un deadlock ocurre, los pro cedimientos deberan detectarlo y recuperarlo.

Estos procedimientos son re tivamente simples en las redes estrella y a nillo, pero pueden ser muy complejos en redes distribuidas.

Los procedimientos de control pueden ser divididos en cuatro categorias generales :

1. Protocolos de comunicación, los cuales manejan el intercambio de información entre dos entidades comunicativas.

2. Algoritmos para controlar el flujo, los cuales gobiernan la aceptación de los datos dentro del sistema y dentro del procesador de comunicaciones.

3. Estrategias de trayectoria, los cuales manejan las colas de salida en los procesadores de comunicaciones y deciden como y cuando transmitir un mensaje.

4. Control y monitoreo en línea, los cuales monitorean el funcionamiento del sistema, y generan diagnosticos de información.

PROTOCOLOS DE COMUNICACION

Los protocolos de comunicación proveen convenciones estandar que permiten a las entidades de la comunicación entenderse y cooperar unas con las -- otras. Los protocolos aseguran que el intercambio de información sea ordenado y este libre de errores. Un conjunto jerarquico de protocolos simplificaran el diseño, la implementación, y la operación. Los protoco

los en un nivel bajo estableceran y mantendran las conexiones; los protocolos de un nivel alto controlaran la transferencia de mensajes una vez que la conexion ha sido hecha; y los protocolos de un nivel aun mas alto manejaran la transferencia de programas y archivos hechas por los multiples mensajes.

Por ejemplo, un sistema centralizado orientado a terminales usualmente tiene dos niveles de protocolos : Los procedimientos para controlar las líneas administran el circuito físico y posiblemente detectan y corrigen errores, y un protocolo que maneja la información del flujo entre la terminal y la computadora anfitriona o concentrador. Una red distribuida puede tener cinco niveles de protocolos, incluyendo el control de la línea y el control de la transferencia de datos entre dos procesadores que se comunican; entre un CP y su computadora anfitriona asociada o terminales; entre dos computadora anfitrionas y entre dos tareas de usuarios.

Procedimientos de Control de Línea.- Los procedimientos de control de línea ocupan el nivel inferior de los protocolos de comunicación. Estos protocolos controlan el medio de transmisión físico y pueden automáticamente detectar y corregir los errores. Los protocolos incluidos en este nivel son procedimientos avanzados para controlar la comuniación de los datos: IBM's binary sincronos communications (BSC) y syncronous data link control (SDLC), DEC's digital data communications message control (DDCMP), e International Standards Organization High=level data link control ----- (HDLC) son ejemplo de estandares de este tipo de protocolo. En seguida veremos una descripción de las funciones generales que los protocolos realizan en este nivel y varios ejemplos específicos.

Las funciones generales son :

1. CONTROLAR LA TRANSFERENCIA DE DATOS

2. COMPROBAR Y RECUPERAR ERRORES
3. CODIFICAR INFORMACION
4. INFORMACION TRANSPARENTE
5. UTILIZACION DE LA LINEA
6. SINCRONIZACION

CONTROLAR LA TRANSFERENCIA DE DATOS.- Tres elementos controlan la transferencia de datos : formateo, información de control, y procedi---mientos de saludo. En este caso el formateo significa reservar campos - en el bloque de transmisión para información especifica. Adicionalmente al mensaje, un bloque contiene datos de control y datos para comprobar - errores. La figura muestra el formato de un bloque con tres campos para control, mensaje y chequeo de errores.

Encabezado Mensaje Error

La Informacion de Control, generalmente colocada en el encabezado,- puede incluir la dirección, la secuencia del numero de bloque, banderas de control, e información de reconocimiento.

La informacion de la Dirección, identifica el destino o la fuente de los datos, y dirige los datos a su destino correcto. El número de - secuencias en el bloque asegura que los datos lleguen en la secuencia en la cual fueron transmitidos, y que no haya bloques perdidos o duplicados.

Las banderas de control indican la secuencia de los bloques y si el bloque es solo de control o contiene un mensaje. El reconocimiento indica si -- el bloque fue correctamente recibio o no. A estos procedimientos de con-- trol frecuentemente se les llama de saludo o apreton de manos (handshaking).

RECUPERACION Y COMPROBACION DE ERRORES.- Una función importante de un protocolo es asegurar la recepción correcta de los datos. Las tecnicas -- mas comunmente usadas para detectar errores en la transmision sincrona es la comprobación de la paridad (VRC y LRC) y ciclic redundancy check(CRC) El protocolo también debe detectar errores sucesivos. Varias tecnicas diferentes, incluyendo los reconocimientos alternativos y la secuencia de bloques, son usadas para varios protocolos.

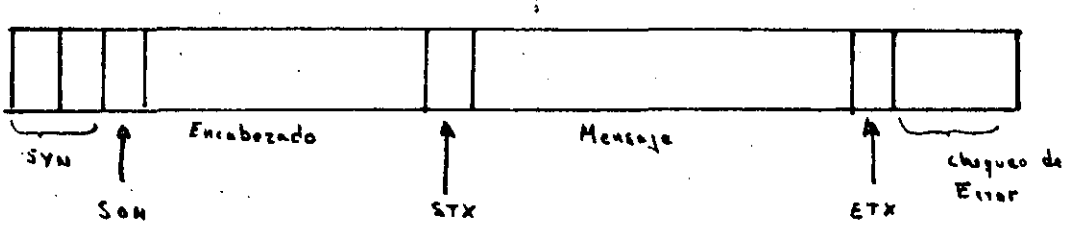
TRANSPARENCIA Y CODIFICACION DE LA INFORMACION.- Frecuentemente es necesario transmitir datos binarios, datos de punto flotante, programas en - codigo objeto, o codigos especializados. Es posible que la secuencia de - bits en esos tipos de datos puedan ser los caracteres de control del protocolo. Por lo anterior el protocolo debe permitir que cualquier caracter - puede ser transmitido en el campo del mensaje.

UTILIZACION DE LA LINEA.- La estructura de un procedimiento para controlar la linea tiene un significado importante en la utilizacion de la -- línea. Algunos de los factores que afectan la utilizacion de la línea es el exceso de control, el manejo de reconocimientos, y el numero de estaciones por línea. Un mensaje transmitido tendra bits de control y frecuentemente bits para detectar errores agregados a el. Los bits de control y detección de errores son considerados como bits de sobrecarga (overhead) ya que constituyen información extra del mensaje en si.

La relación del núemro de bits del propio mensaje al numero total de bits transmitidos es una manera para determinar la utilizacion de la línea. - En las líneas half duplex se debe invertir la direccion cada vez que el - receptor y el transmisor intercambian papeles. El tiempo de inverción de la línea, especialmente cuando los bloques son chicos. La operación en -- full duplex esta necesidad y consecuentemente el tiempo asociada a ella. El factor final que afecta la utilizacion de la linea es el número de dis positivos que comparten la linea; multiples dispositivos pueden compar-- tir una linea con una estructura de multipunto para incremen tar la utilizacion de la línea.

Ejemplos de Protocolos

Binary Syncronous Communications.- El formato para el protocolo Bina ry Sincronous Communications (BSC o BISYNC) de IBM se muestra en la figura .



SYN: caracter de sincronia
SOH: inicio de encabezado
STX: inicio del texto
ETX: fin de texto

BYSYNC usa caracteres de control para delimitar los campos. El encabeza-

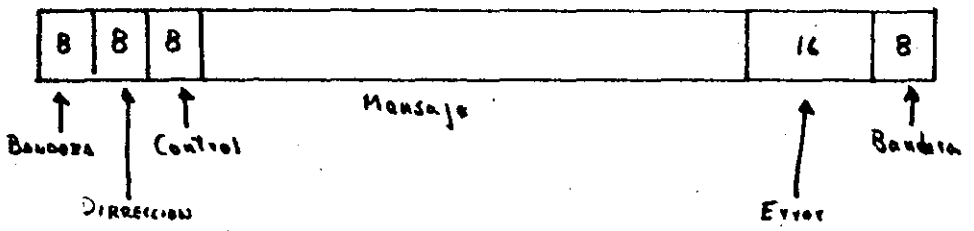
do, es opcional, empieza con un inicio de encabezado (SOH) y debe ser el primer campo en el bloque, cuyo contenido esta definido por el usuario. El campo del texto empieza con inicio de texto (STX) y es finalizado -- con el fin del texto (EOF) o un fin de bloque (ETB). Este campo es de - longitud variable y puede contener información transparente si los delimitadores son precedidos por el caracter de control data link escape --- (DLE).

BISYNC soporta ASCII, EBCDIC y transcode de seis bits para codificar la información. Una combinación de los metodos VRC/LRC para comprobar e--- rrores es usada cuando los datos son codificados en ASCII. Ni EBCDIC ni el transcode de seis bits proveen un bit de paridad disponible, así que CRC es usado para comprobar los erroes en estos codigos. Un divisor con referencia a 16 bits (CRC-16) es usado con EBCDIC para genera un residuo de 16 bits, el cual es transmitido en el campo de detección de errores. El receptor divide el bloque por el mismo modelo de referencia para determinar si hay un error en el bloque. Si ningún error es detectado, el bloque es aceptado y un caracter de reconocimiento (ACK) es enviado al transmisor. Un error causará que el bloque sea descartado y un caracter de reconocimiento negativo (NAK) sera enviado al receptor. -- Varias tentativas para recibir el bloque seran hechas antes de que BISYNC asuma que la linea esta defectuosa. El transcode de seis bits trabaja - de la misma manera, pero la secuencia para comprobar los errores es de una longitud de solo 12 bits (CRC-12)

La transmision en sistema BISYNC esta limitada a half duplex. La línea debe invertir la direccion entre dos mensajes, una vez para el reconocimiento y otra para el bloque de datos. Los reconocimientos son maneja-- dos como bloques de control separados, y cada bloque de datos requiere - un reconocimiento.

BISYNC soporta tanto redes de punto a punto como de multipunto, pero esta limitada a transmision sincrona. La sincronización es alcanzada precedien do al bloque formateado minimamente con dos caracteres de sincronia (SYN).

SYNCHRONOUS DATA LINE CONTROL (SDLC).- El protocolo SDLC de IBM usa solo un caracter de control, llamado caracter bandera. Este caracter enmarca - el mensaje, como se muestra en la figura.



El encabezado tiene generalmente 24 bits de longitud, el campo del texto - es de longitud variable, y el campo final es nominalmente de 24 bits de -- longitud. El encabezado incluye el caracter de control, el campo de direc ción, y el campo de control. La tecnica usada para proveer la transparencia puede incrementar el tamaño de cualquiera de esos campos excepto para las banderas. La bandera tiene un modelo de tamaño fijo (01111110) sin im portar el codigo de datos usado. Si cinco bits '1' consecutivos aparecen en cualquier lugar del bloque excepto en la bandera, un bit '0' sera inser tado despues del quinto bit. Esta tecnica es provista para la transparencia de datos y es llamada bit de relleno. El receptor removera los bits - de relleno para recuperar el bloque en su forma original.

SDLC usa CRC-CCITT para detectar los errores de transmision. Este metodo

usa una tecnica de inversion para calcular la deteccion de errores. Si - en la transmisión un error es detectado, SDLC no responde con un NAK- en cambio envia una respuesta de tres bits en el campo de control indicando el tipo de error. Para ejemplificar diremos que: si B recibe los mensa-- jes 2, 3, y 4 desde A, y el mensaje 4 esta malo. El siguiente mensaje -- que B envia a A contendra 4 en el campo de respuesta. Esto indica a A que B recibio los mensajes 2 y 3 y espera un periodo de tiempo para un re conocimiento antes de enviarlo otra vez. Si un error de secuencia ocurre B, respondera con un NAK y el numero del ultimo mensaje bueno en el campo de respuesta.

SDLC usa las facilidades de transmision eficientemente y puede manejar - transmision half duplex o full duplex. Tiene una baja sobrecarga de ca-- racteres, no necesita mensajes de reconocimiento separados, y soporta tan to redes de punto a punto como de multipunto. No puede ser usada en transmision asincróna o paralela debido al bit de relleno. SDLC no -- esta limitado a los códigos de datos, asi que virtualmente cualquier código puede ser usado. Puesto que el ultimo caracter de control es la ban dera, SDLC sincroniza los bloques con los caracteres bandera.

DIGITAL DATA COMMUNICATIONS MENSAGE PROTOCOL.- El DDCMP usa solo un carac ter de control por mensaje; es el primer caracter en el mensaje. El ca- racter de control distingue entre mensajes de datos, control, y comunica- ciones. El formato de un bloque DDCMP es mostrado en la figura.

CRC: CICLIC REDUNDANCY CHECK.

El encabezado contiene un contador de los bytes que hay en el texto, algunas banderas de control, un campo de respuesta, un mensaje de numero de secuencia, y una direccion. El campo del texto es de longitud variable hasta 16,383 bytes. Cada uno de esos campos es verificado mediante el campo CRC de 16 bits que se anexa.

DDCMP usa los caracteres ASCII de control SOH, ENQ, y DLE para distinguir entre los tipos de mensajes. (Estos caracteres significan datos, control, y comunicación respectivamente mensajes).

El campo del contador especifica el número de bytes en el campo del mensaje y provee la transparencia de la información. El dispositivo receptor determina el inicio del campo del mensaje y acepta el numero de bytes especificando por el contador sin verificar los caracteres de control. Para alcanzar la sincronización se precede al bloque con dos caracteres ASCII llamados SYNC. Si no hay intervalos entre los mensajes, la sincronización no es necesaria despues del primer mensaje.

Ambos modos de transmision half y full duplex son soportados por DDCMP. Para mejorar la utilización de la línea, los mensajes separados ACK no son necesarios al menos que el trafico en la dirección opuesta este ligero. DDCMP también soporta ambas redes de punto a punto y multipunto y, a diferencia de BISYNC o SDLC, puede ser usada en transmisisones sincronas, asincronas y paralelas. DDCMP tiene otra capacidad no encontrada -

en las otras: puede inicializar un sistema que no este operando cargando el software necesario y reinicializar la maquina por medio de la linea de comunicación. Los mensajes de comunicacion son usados para este propósito.

RED PUBLICA DE TRANSMISION DE DATOS

Porqué es una necesidad nacional una red especializada de tranmisión de datos ?.

El aumento en la necesidad de comunicación a velocidades superiores y el resultado del éxito logrado en la conducción de señales de datos a través de grandes distancias ha revelado recientemente la necesidad de formar redes de computadoras que permitan compartir y aprovechar eficientemente su capacidad de procesamiento. En efecto, en México como en muchos otros paises se ha visto la conveniencia de implantar una red dedicada de transmisión de datos que satisfaga la necesidad de procesar información a distancia la cual tiende a incrementarse de manera explosiva ya que el desarrollo de las aplicaciones de la teleinformática constituye un factor de transformación de la organización económica y social y del modo de vida en general.

En consecuencia, con la idea de proporcionar servicios cada vez más confiables y con alto grado de disponibilidad, la propia Dirección General de Telecomunicaciones inició el proyecto para implantar una Red Pública de Transmisión de Datos (RPTD) que empleará la técnica de conmutación de paquetes y que será sin duda la espina dorsal del desarrollo de la teleinformática en el país.

Qué es la conmutación de paquetes y porque esta técnica ?

Es un método nuevo y eficaz de comunicación de datos, que permite que muchas terminales y usuarios de computadoras compartan simultaneamente una red común, logrando con ello una transmisión de datos a bajo costo con alta

confiabilidad. En transmisión de datos se utilizan dos metodos, principales: La conmutación de circuitos y la conmutación de paquetes. Se eligió la conmutación de paquetes porque abate costos por concepto de transporte de la información al aprovechar mejor la infraestructura de telecomunicaciones existente, es muy flexible y favorece la optimización de los recursos informáticos del país. Conviene señalar que la conmutación de circuitos es adecuada cuando se tienen que conectar sistemas de ancho de banda fijo con velocidades homogéneas.

PRINCIPIO

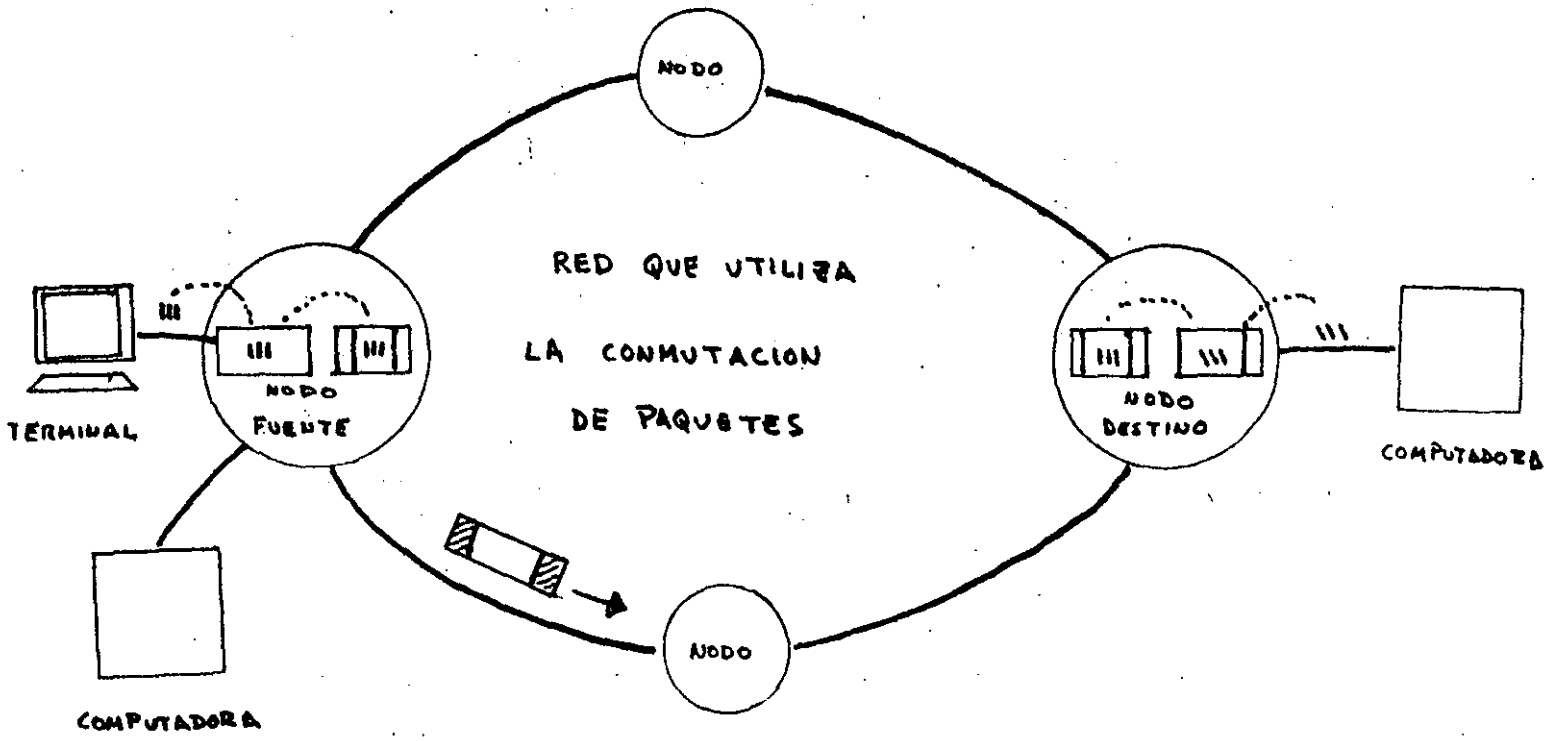
El concepto leigado intimiamente al de conmutación de paquetes es el de guarda-reexpide. Los paquetes en la red parten de un nodo fuente y van a un nodo destino (ver figura); es decir, la secuencia de datos provenientes de una terminal o computadora se envían a un punto de entrada llamado NODO ORIGEN O FUENTE de una red conmutada en paquetes. En el nodo los datos se ensamblan en pequeños seqmentos llamados PAQUETES.

Cada paquete tiene un encabezado con sus direcciones e información de control de la conversación específica a la que corresponde y despues es transmitido a través de la red.

Para esta transmisión no se establece una ruta dedicada, sin embargo, en cada nodo de la red el paquete se conecta al siguiente enlace hacia su destino a través de rutas primarias o secundarias. Una vez en el nodo de destino, el paquete se desensambla y los datos se reciben en la computadora anfitriona en el mismo formato o secuencia en que salieron de la terminal.

Los NODOS son capaces de:

- Enrutar los paquetes hacia su destino
- La fragmentación de mensajes en paquetes
- El manejo de mensajes (agregar encabezados, señales de verificación etc.)
- La detección de errores y de fallas de elementos en la red
- La entrega de mensajes a la anfitriona
- El control de flujo
- La reexpedición de mensajes
- El envío de reconocimiento de entrega y
- El mantenimiento de Estadística de tráfico, entre otras.



RED PUBLICA DE TRANSMISION DE DATOS

El objetivo principal de la Red Pública de Transmisión de Datos es el de DOTAR AL PAIS DE UNA INFRAESTRUCTURA SEGURA, FLEXIBLE, CON UNA ALTA CONFIABILIDAD, GRAN DISPONIBILIDAD Y EXTENSA CAPACIDAD DE CRECIMIENTO que permita mejorar la presentación de los servicios públicos y así fomentar el desarrollo de la teleinformática, disminuyendo costo por conceptos de transmisión, permitiendo su acceso a las empresas pequeñas y medianas que actualmente carecen de ella.

La RPTD entonces:

- Responde al crecimiento rápido de la demanda en teleinformática
- Es adaptable a la gran diversidad de sistemas y aplicaciones
- Presenta grandes garantías de mantenimiento y seguridad
- Coadyuva en la homogeneización de los recursos informáticos actuales y futuros del país.
- Facilita la expansión coherente, armoniosa y eficaz de los recursos informáticos.
- Es de fácil acceso entrante a través de la red teleinformática conmutada, así como de la red telex en entrada.
- Facilita la interconexion de equipos informáticos variados y su evolución hacia la informática distribuida.
- Favorece la descentralización operando en casi todo el territorio y suprime la incidencia de la distancia sobre los costos.
- Ofrece un servicio conforme a normas internacionales.

CARACTERISTICAS DE LA RED

La RPTD será una red pública nacional con una configuración tipo malla que como ya se ha dicho utilizará la técnica de la conmutación de paquetes.

Los planes de implantación de la Red Pública de Transmisión de Datos - se desarrollarán en tres etapas a saber :

- LA PREVIA O EXPERIMENTAL
- LA FASE I
- LA FASE II

La etapa experimental: En esta fase previa, la red consta de 3 conmutadores de paquetes localizados en México, Monterrey y Guadalajara, 2 concentradores ubicados en Hermosillo y Puebla y 7 multiplexores localizados en -- León, Queretaro, Toluca, Cuernavaca, Acapulco, Veracruz y Villahermosa. La capacidad instalada que se pone a disposición de los usuarios puede atender hasta 250 terminales y computadoras. Este servicio se ha puesto en operación experimental con un número de servicios y usuarios restringido puesto que cabe mencionar que la asimilación de una nueva tecnología es paulatina, puesta en operación de este servicio será en el mes de octubre del presente año (figura 1).

Una vez superada esta etapa, se pasará a dar un servicio público amplio incrementándose a 24 el número de puntos de acceso. En esta primera fase el concentrador de Hermosillo pasará a ser un conmutador de paquetes completándose así la red con 4 conmutadores (figura 2). La red en tales condiciones - podrá dar servicio a 950 terminales y computadoras. Según los planes trazados por la Dirección de Telecomunicaciones esta etapa deberá ser operacional en junio de 1981.

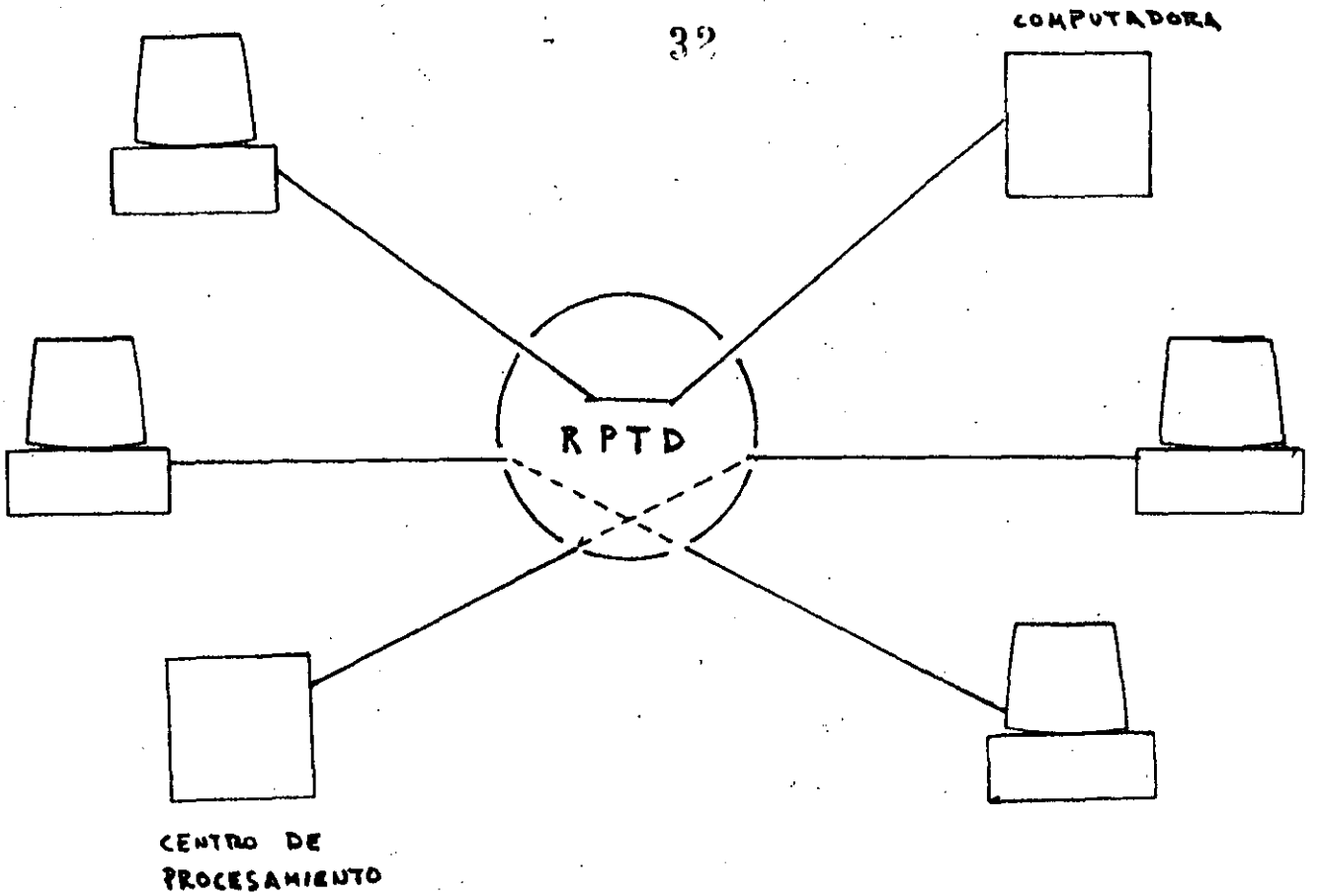
La última etapa contemplada en el proyecto actual, prevista para operar en diciembre de 1982 plantea la necesidad de aumentar en 20 ciudades -- los accesos a la red. En consecuencia, al completarse esta segunda fase se tendrá acceso a la Red Pública de Transmisión de datos en las 44 principales ciudades del país y podrá satisfacer una demanda de hasta 2000 terminales y computadoras . (fig.3)

SERVICIOS QUE PROPORCIONA LA RPTD

Los servicios que ofrecerá la Red Pública de Transmisión de Datos, - son de muy variada naturaleza, en donde básicamente se desea interconectar a 2 ó más usuarios (terminales, computadoras, programas de aplicación, - procesos, etc.) entre si, de tal manera que la red sea transparente y -- permita que procesos o usuarios compatibles y no compatibles puedan comunicarse.

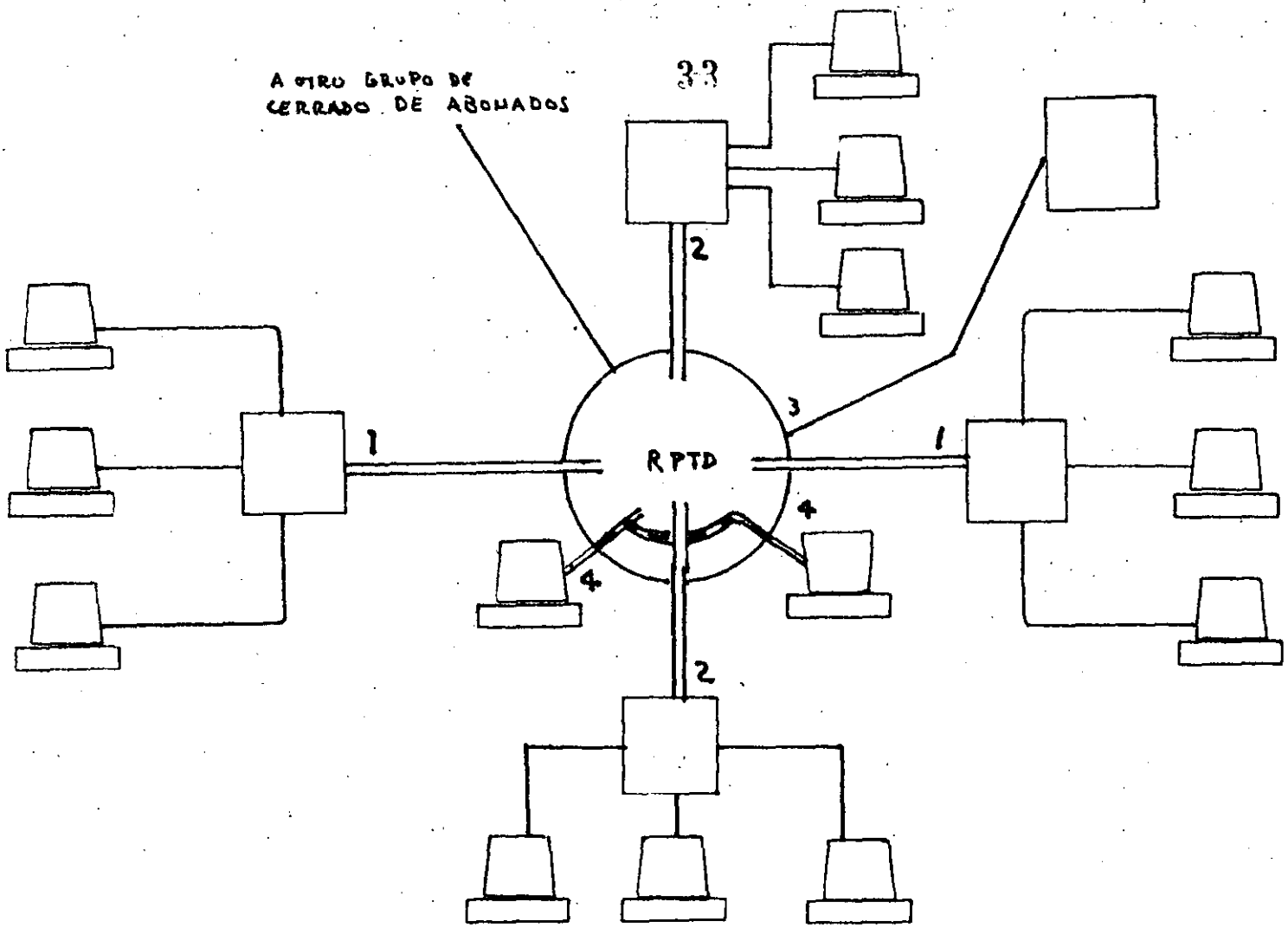
Los servicios mínimos que ofrece la red en su inicio son :

- Circuitos virtuales conmutados (CVC)
- Circuitos virtuales permanentes (CVP
- Grupo cerrado de abonados
- Comunicaciones por cobrar
- Conversión de protocolos a X.25
- Conexión de usuarios asíncronos (X.3, X.28, X.29)
- Conexión de usuarios sincronos (X.25 y otros protocolos)
- Acceso entrante a través de la red telefónica conmutada
- Acceso entrante a través de la red telex.



CIRCUITOS VIRTUALES Y PERMANENTES

Un circuito virtual está caracterizado por el establecimiento a través de la red de un enlace entre 2 canales que aseguren cada uno la conexión de un equipo terminal de datos de la red. Los circuitos virtuales pueden ser conmutados o permanentes. La transmisión se efectúa de la misma manera en ambos casos. El servicio en base a circuitos virtuales permanentes es bastante más simple que los conmutados, ya que se asemeja más a los enlaces especializados y la transmisión puede establecerse en cualquier momento.



GRUPO CERRADO DE ABONADOS

Todos aquellos usuarios que deseen reunirse en grupos y que sistemáticamente rechacen la aceptación de cualquier comunicación que no provenga de alguno de ellos tiene la posibilidad de hacerlo.
La RPTD está preparada para ofrecer el servicio (grupo 1)

El control de acceso de un abonda al grupo, será efectuado de manera automática por la red, la cual debe tener al día las listas de los miembros de distintos grupos.

Un grupo cerrado de abonados puede también conectarse con los abona

dos de libre acceso (2).

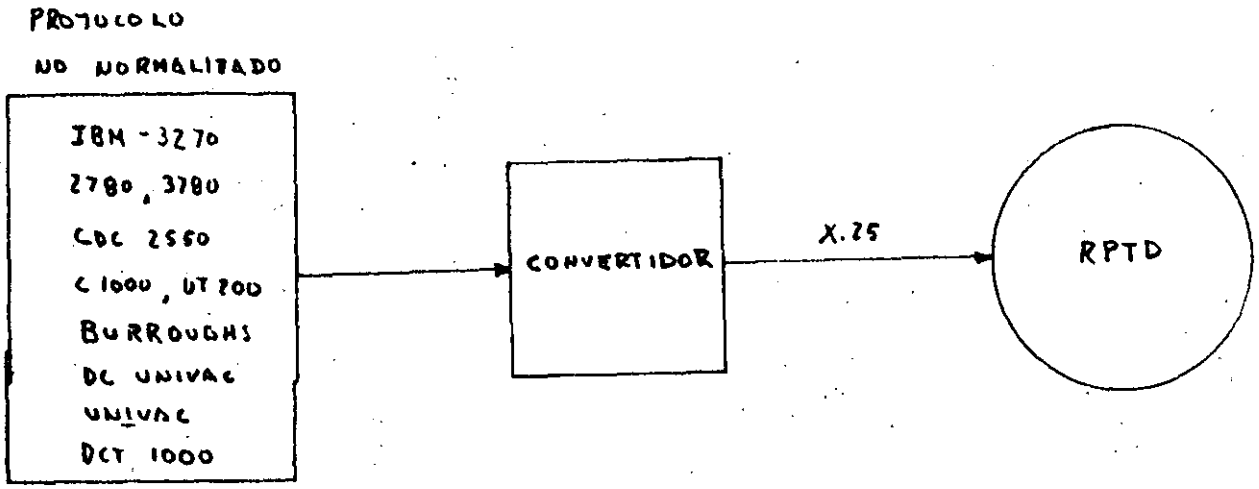
Un usuario podrá solicitar la pertenencia a varios grupos cerrados, existiendo igualmente la posibilidad de poder llamar a los usuarios del grupo de libre acceso (3).

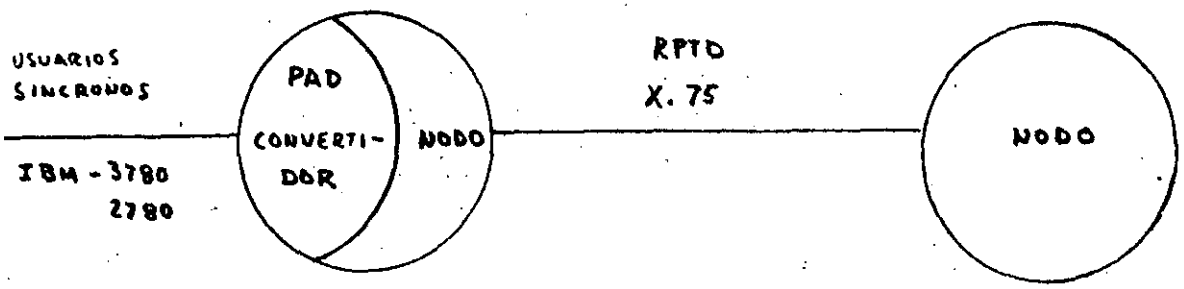
Los usuarios que no deseen pertenecer a un grupo cerrado de abonados, pertenecerán al grupo de libre acceso (4).

Es decir que el servicio de grupo cerrado de abonados otorgado por la RPTD permite la confidencialidad pués tiene totalmente protegido el acceso.

CONVERSION DE PROTOCOLOS

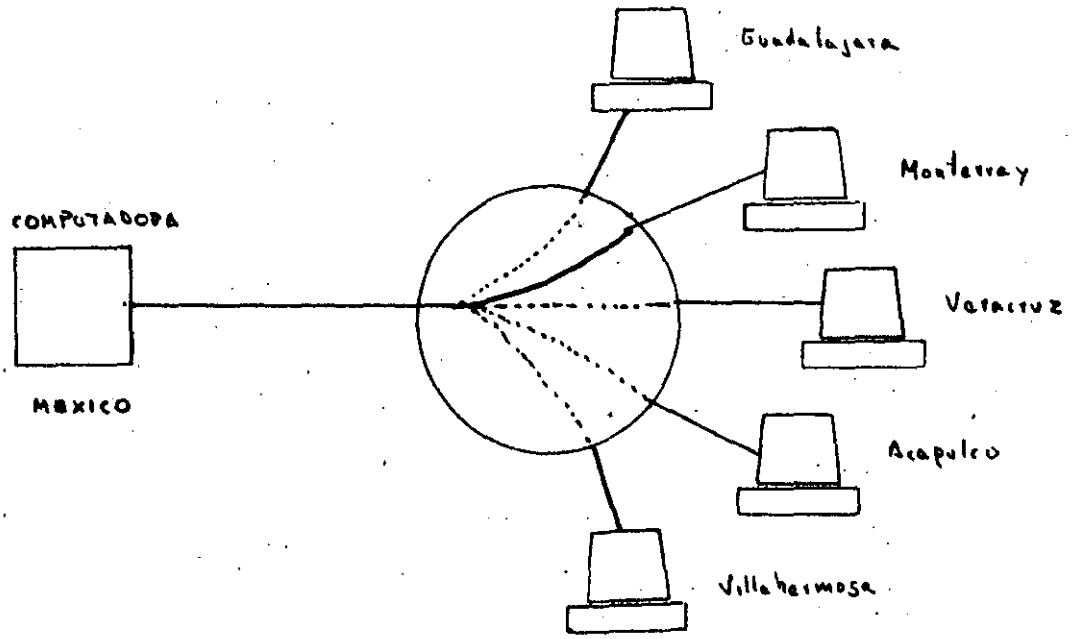
En la Red Pública de Transmisión de Datos, se está previendo la facilidad de transparencia en el acceso. Consecuentemente deberá dar cabida a los principales tipos de terminales y/o computadoras que se encuentren en el mercado nacional. Es decir, se ofrecerá el servicio de conversión de protocolos en los casos en que las terminales tengan un protocolo diferente al protocolo standard .





UTILIZACION DE LA RED

El protocolo normal de acceso a la red (X.25) permite igualmente el multiplexaje de varios circuitos virtuales, ya sea conmutados o permanentes, en el mismo enlace físico, lo que permite por ejemplo a una computadora comunicarse simultáneamente con un número elevado de terminales repartidas en el Territorio Nacional, usando solo una línea de alta velocidad para conectarse a la red.



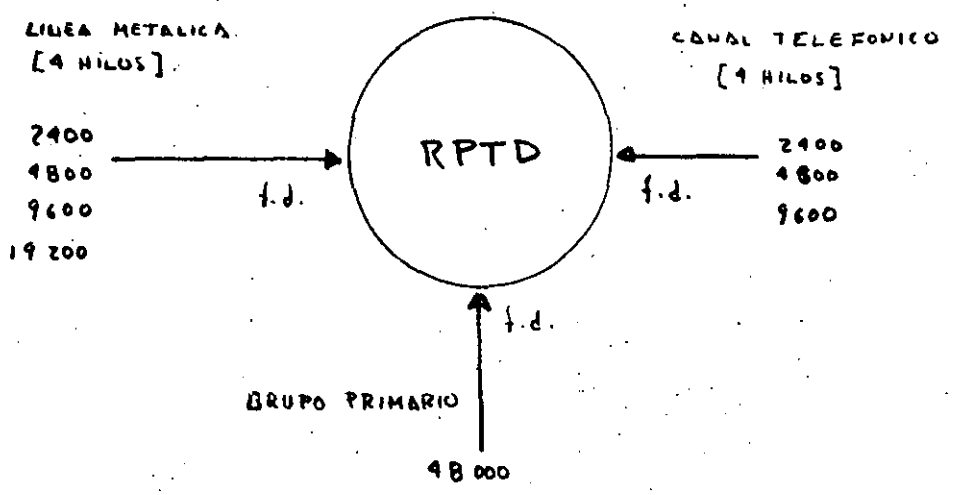
COMUNICACIONES POR COBRAR

Otro de los servicios que los usuarios pueden utilizar, será ---- aquel en que el monto de las llamadas se cargue al abonado solicitado. Este servicio se justifica por el hecho de que en muchos de los casos, diferentes usuarios (terminales) pertenecen a unamisma firma y la tarificación única facilita las tareas administrativas.

TIPO Y MODO DE CONEXION DE LOS USUARIOS

USUARIOS SINCRONOS

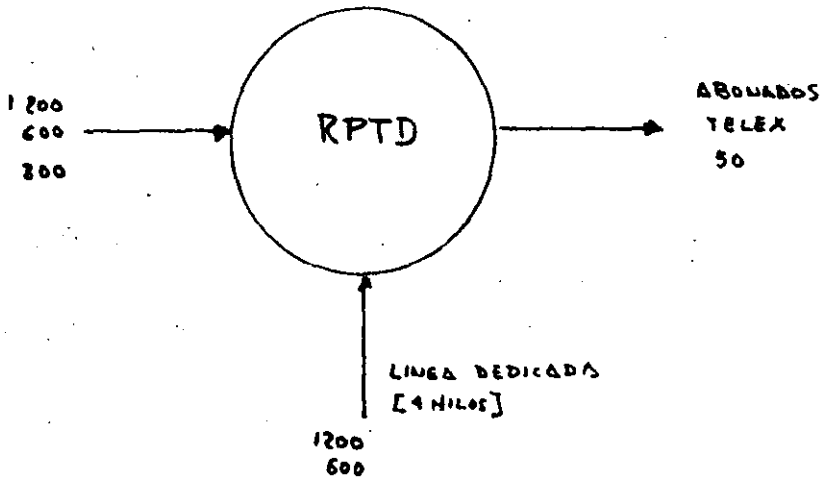
Se enlazarán a la RPTD mediante enlaces dedicados full duplex de 4 hilos a las velocidades de 2400, 4800, 9600, 19200 por canal telefónico - normal y 48000 bps a través de un grupo primario.



USUARIOS ASINCRONOS

Las terminales asíncronas tipo "Start-stop" que utilizan el alfabeto número 5 del CCITT se conectarán a la red con velocidades : 300,600, y -- 1200 bps full duplex a través de la red telefónica conmutada o por líneas dedicadas (2 y 4 hilos) si lo requiere el usuario. La RPTD ofrecerá también acceso a los usuarios de la red telex a una velocidad de 50 bauds.

USUARIOS ASINCRONOS



Es decir la RPTD ofrece la posibilidad de conectarse a la red en diferentes velocidades dependientes de cada aplicación, desde 50-48000 bps por enlace.

PROTOCOLOS

Un protocolo de comunicación de datos es un convenio sobre el significado del Formato y la duración relativa de la información que se intercambia entre dos dispositivos de comunicación. Los protocolos en una red están intimamente ligados a la arquitectura del sistema y a los servicios o funciones que se están proporcionando.

Con el objeto de cumplir con los requisitos de transferencia, flexibilidad y normalización, en la interconexión de equipos informáticos heterogéneos la RPTD está regida bajo los siguientes protocolos:

PROTOCOLO DE ACCESO DE LAS TERMINALES SINCRONAS EN MODO PAQUETE

Estos usuarios se conectan a la red usando el protocolo X.25 con las siguientes características:

1. NIVEL FISICO: proporciona las características mecánicas, electricas, funcionales y de procedimiento, para establecer, mantener y liberar las conexiones físicas entre enlaces de datos. En este nivel se incluyen el tamaño y tipo de clavijas, los niveles de voltaje y las señales de listos hacia y desde cada uno de los dispositivos, entre otras.

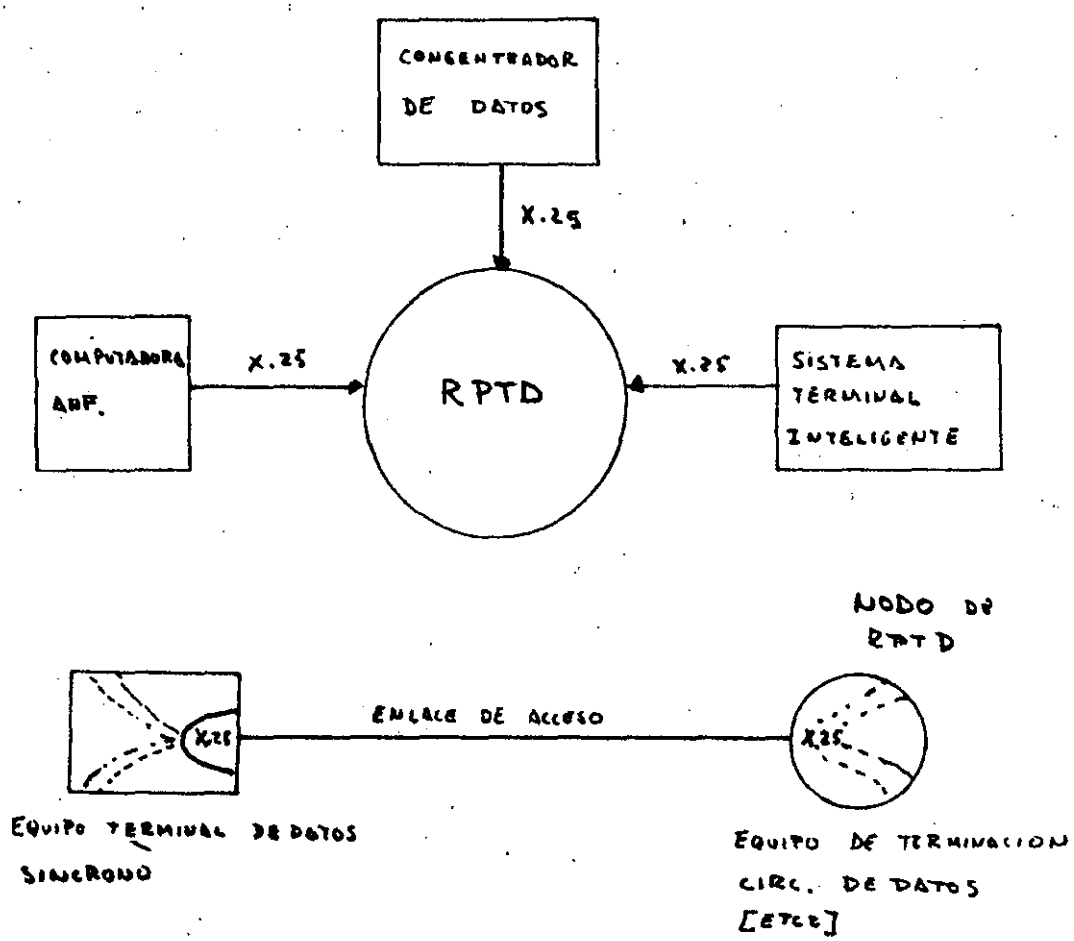
2. NIVEL DE ENLACE: este nivel es responsable del transporte, sin errores de los paquetes a través de cada enlace de la red. En este nivel se proporciona el control para la inicialización, para las tramas, los errores, el flujo de datos, recuperación sobre eventos anormales etc.

   El procedimiento utiliza el principio y terminología del procedimiento de control de alto nivel para enlaces de datos (HDLC High Level Data Link Control Procedure) especificado por la Organización Internacional de Normalización (ISO).

3. NIVEL DE RED: proporciona las funciones y procedimientos para intercambiar servicios de datos entre dos entidades sobre una conexión en la red; es decir se proporcionan las facilidades de llamadas virtuales y circuitos virtuales permanentes. Para permitir las llamadas virtuales y/o los circuitos virtuales permanentes simultáneos, se --

usan canales lógicos . Para las llamadas virtuales, se asigna un número de-grupo de canales lógicos (15) y un número de canal lógico (255) durante la fase de establecimiento de la comunicación. Para los circuitos virtuales permanentes se asigna un número de grupo de canales lógicos (15) y un número de canal lógico (255) por acuerdo con la Secretaría en el momento-de abonarse al servicio.

ES DECIR X.25 CONSTITUYE UNA NORMA PARA LA CONEXION (INTERFAZ) EFICAZ ENTRE CUALQUIER DISPOSITIVO PROGRAMABLE DEL USUARIO (COMPUTADORA ANFITRIONA, CONCENTRADOR DE DATOS O SISTEMA TERMINAL INTELIGENTE) Y UNA RED DE PAQUETES, (ver figura).

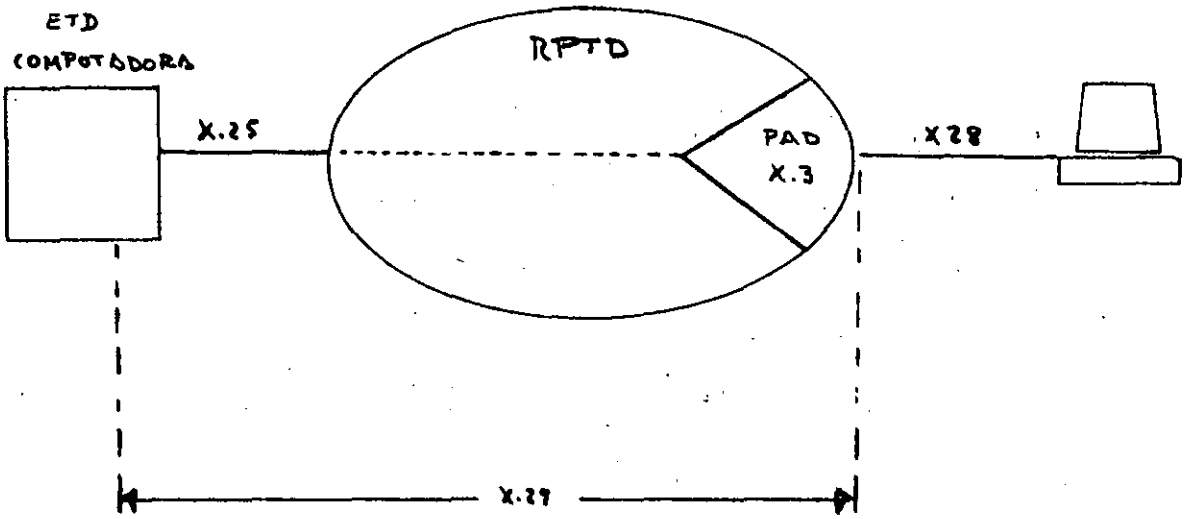


Las recomendaciones X.3, X.28 y X.29 detallan los elementos necesarios para que la red apoye terminales asíncronas (de arranque/parada) no inteligentes.

La recomendación X.3 establece un juego de parámetros que usa el nodo para controlar la terminal a la que dá servicio. Estos parámetros definen características especiales para la terminal.

La recomendación X.28 define la interfaz entre la terminal asíncrona y el nodo. En especial establece el lenguaje de comando que emplea el usuario para fijar los parámetros X.3 y para inicialización, establecimiento control, etc., entre el PAD y el equipo terminal de datos.

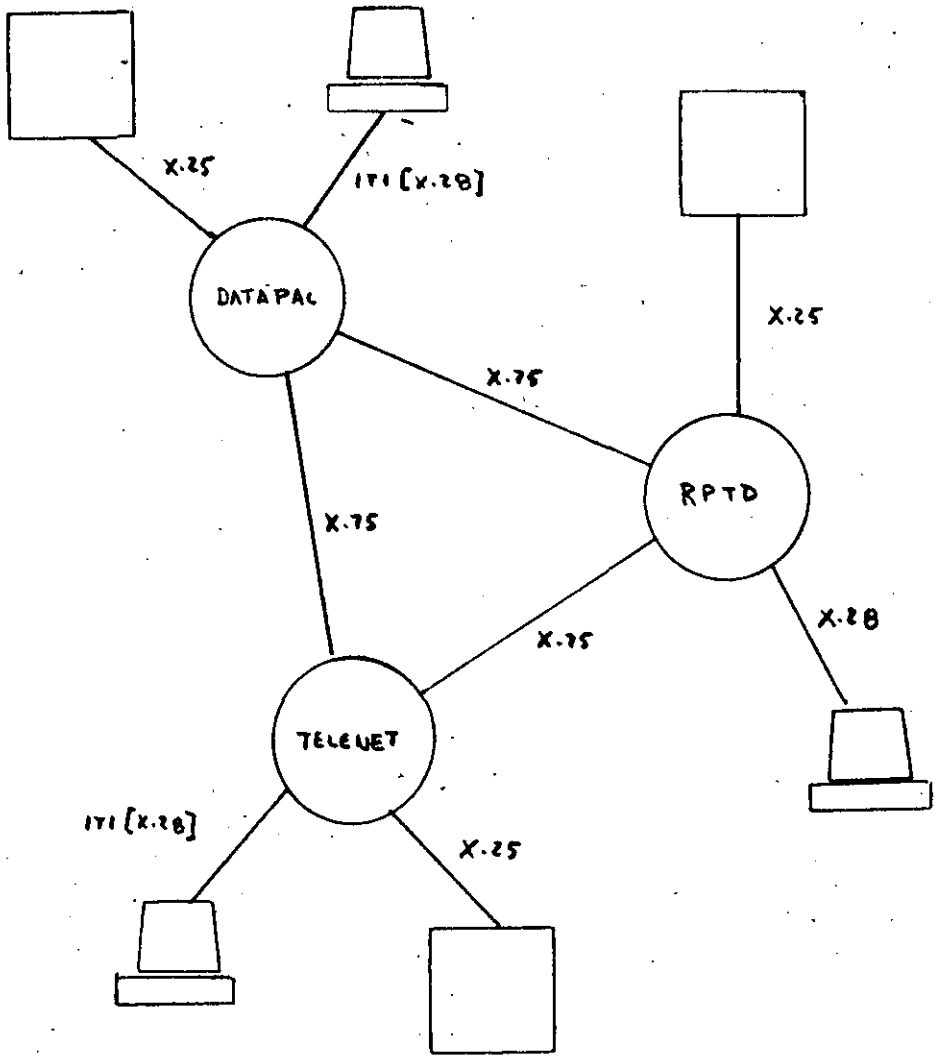
INTERFAZ DE TERMINAL INTERACTIVA



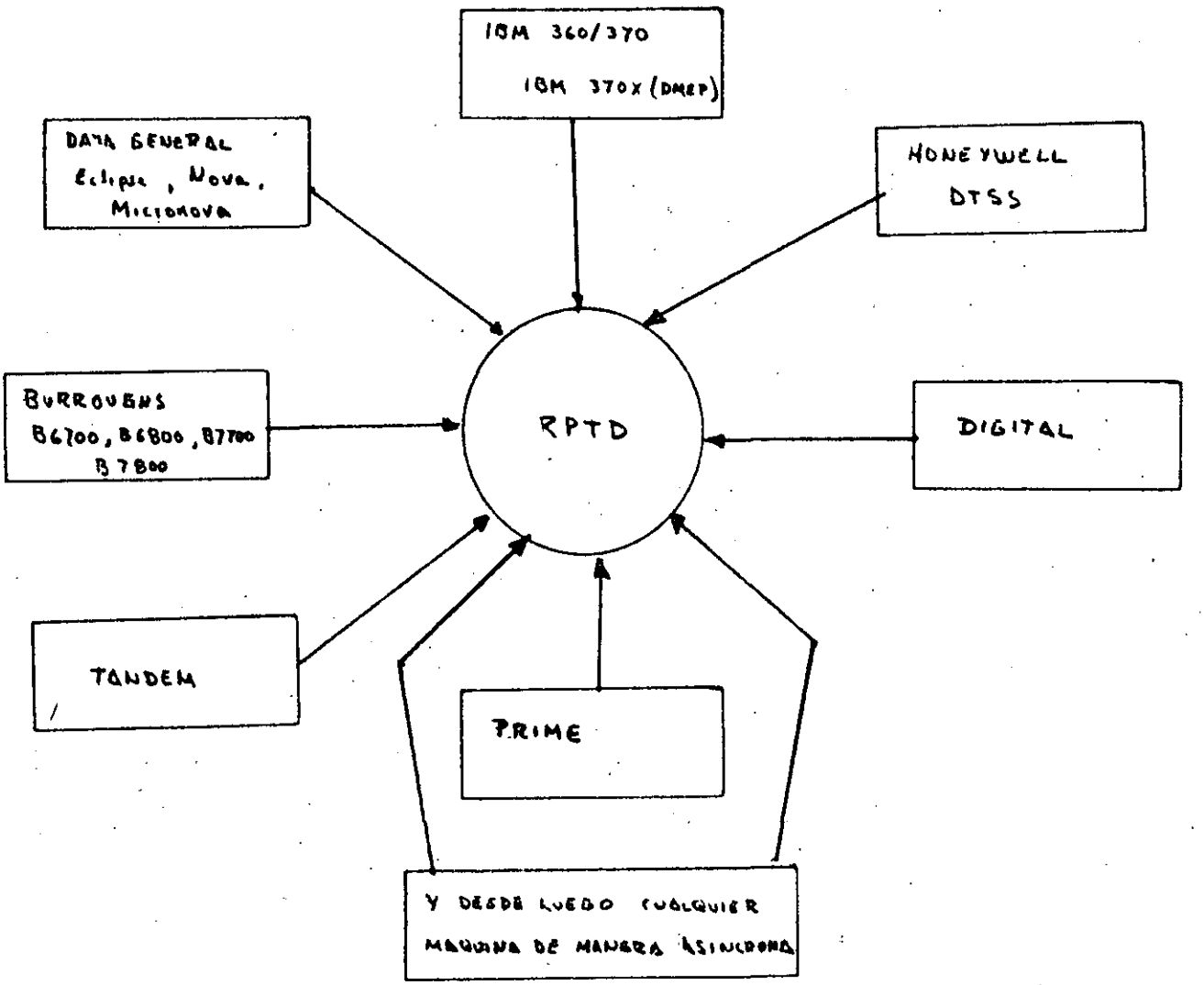
La recomendación X.29 establece los procedimientos para controlar la terminal asíncrona mediante un ETD remoto. Establece un lenguaje de comandos por el cual la anfitriona remota puede cambiar los parámetros de --- X.3

CONEXION CON OTRAS REDES

El protocolo X.75 establece también los procedimientos de interconexión de las redes de datos, éstos junto con los administrativos, de control etc., residentes en una compuerta internacional permiten la interconexión entre redes.



LOS SISTEMAS SE PUEDEN CONECTAR INMEDIATAMENTE A LA RPTD CON X.25 SON:



CONFIABILIDAD Y DISPONIBILIDAD DE LA RPTD

La red de transporte constituida por los nodos, las líneas de comunicación entre estos y los modems de alta velocidad es de tipo malla (Totalmente conectada a través de canales de 64Kbps) lo que garantiza bajos tiempos de respuesta y de alta disponibilidad.

La red externa o puntos de acceso se conectan en forma de estrella a los nodos con líneas respaldadas.

Los equipos de conmutación o nodos son modulares, flexibles y su expandibilidad (pasar de puntos de acceso a nodos) es lo menos sofisticada posible. El equipo es robusto en el sentido de que áún con fallas parciales en sus componentes sigue operando. (i.e. memoria, CPU, interfaces de línea, bus, etc.), esto quiere decir que los nodos están respaldados totalmente.

La red está distribuida en el sentido de inteligencia i.e. las funciones de la red están distribuidas en los nodos y las funciones del centro de control de la red son básicamente de monitoreo, estadísticas y tarificación, etc. Esto hace que en caso de falla total de un nodo o del centro de control la red sigue funcionando.

La RPTD satisfará un amplio rango de aplicaciones y al mismo tiempo ofrecerá un servicio público eficiente y con una amplia gama de servicios puesto que responderá a una disponibilidad del 99.9% y confiabilidad del 99.99% (24 horas diarias los 7 días de la semana respectivamente).

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA COMPUTACION Y PROGRAMACION ELECTRONICA

SISTEMAS Y PROGRAMAS

- CONTINUACION -

C O B O L

ING. FRANCISCO LLAMAS

OCTUBRE, 1984

I N T R O D U C C I O N

PASCAL fue desarrollado entre los años 1968 y 1971 por Nilaus Wirth y un notable equipo de colaboradores en el instituto de Informática del Instituto Federal de Tecnología de Zurich, Suiza.

PASCAL, al igual que algunos otros pocos lenguajes es reconocido como un fenómeno en un mundo donde nacen y "mueren" multitud de lenguajes que -- brillan intensamente un momento para después desaparecer casi por completo, PASCAL, es un lenguaje que como FORTRAN o BASIC comienza a ser aplicado ex-- tensamente, o sea se le comienza a reconocer como "Lingua Franca" o "Lenguaje Popular" de la programación.

Actualmente, son cada vez más y más las instalaciones que cuentan con compiladores de PASCAL y los sistemas basados en microcomputadoras que es-- tan haciendo un uso muy extendido de PASCAL.

La popularidad tan grande, tal vez se deba a que PASCAL fue concebido siempre como un lenguaje de propósito general, pero con especial anten-- ción hacia la enseñanza de la programación de computadoras, poniendo un gran enfásis en la programación estructurada.

No obstante, no es ni su orientación a la educación, ni su facilidad para permitir la programación estructurada, donde reside su más grande calidad, esta reside en la gran capacidad para estructurar los datos; ya que la mejor arma de PASCAL son sus poderosas estructuras de datos.

Siendo tales las condiciones de este lenguaje, resulta idóneo para la introducción de la práctica de los más importantes conceptos para la programación como el uso de las estructuras de datos y la práctica de los conceptos de la programación estructurada.

EL ENCABEZADO EN PASCAL

Comenzaremos nuestro estudio del lenguaje PASCAL considerando la definición que Wirth expone sobre lo que es un programa.

Para Wirth un programa está formado por :

- Una descripción de las ACCIONES que se llevarán a cabo y

- Una descripción de los DATOS que se manipularán con esas acciones

Particularmente, en PASCAL el programa lo podemos dividir en dos grandes partes : ENCABEZADO y el BLOQUE.

El encabezado de un programa en PASCAL es la parte en la cual se le asigna nombre al programa (en algunas implementaciones no es necesaria) y se proporciona también una lista de las variables (o archivos) de entrada /salida del programa.

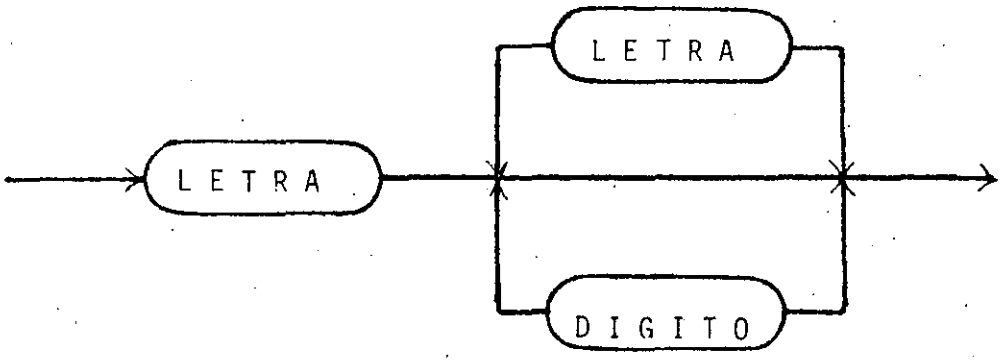
El bloque de un programa en PASCAL, a su vez podemos dividirlo en dos partes; las DECLARACIONES donde se hace la descripción de los datos del programa y las INSTRUCCIONES, mediante las cuales se describen las acciones.

LAS DECLARACIONES Y DEFINICIONES EN PASCAL

En PASCAL existen ciertos OBJETOS que pueden ser declarados o definidos. Estos objetos que pueden ser declarados o definidos estan relacionados con los DATOS; esto es, en la parte de un programa de PASCAL se hace la descripción de los datos.

El hecho de declarar los objetos significa que se asocia un "identificador" con el objeto. En PASCAL un identificador se forma iniciando con una letra y después el número necesario de letras y/o digitos. En la figura 1 se muestra el diagrama que auxilia en la definición de un identificador en PASCAL.

FIGURA 1



IDENTIFICADOR

Para separar entre si a los identificadores, reconoceremos a los separadores sintácticos que son los blancos, el fin de línea y comentarios. Cualquier número de separadores sintácticos pueden existir entre dos símbolos de PASCAL consecutivos, excepto para los casos que se indican a continuación :

No pueden existir separadores sintácticos enmedio de un identificador una constante, o un símbolo reservado de PASCAL.

En el párrafo anterior se hace mención de los símbolos reservados de PASCAL, que con identificadores que no puede utilizar el programador para asociarlos con objetos; estos símbolos o palabras reservadas del lenguaje se presentan en el apéndice A. La utilidad de estos símbolos reside en el hecho de que con su uso hacen posible la difinición de los objetos y acciones de un programa.

Los OBJETOS que se pueden declarar en PASCAL son los siguientes :

- Las etiquetas (que en nuestro caso no utilizaremos).
- Las constantes; con las cuales tenemos medios de hacer corresponder identificadores con constantes numéricas o alfanuméricas --- (cadenas de caracteres).

- Los Tipos; con los cuales existe la capacidad de definir tipos de datos que le convengan al programador; en otras palabras, el programador puede "inventar" sus propios tipos de datos.

- Las Variables; con los cuales el programador puede dar nombre propio a las localidades de memoria que está utilizando para almacenar y organizar sus datos.

En la figura 2 se muestra un diagrama de sintaxis que muestra la forma de definir las etiquetas, constantes, tipos y variables en PASCAL.

Por otro lado; en la definición de todos estos objetos; es necesario, y es regla general en PASCAL, que los objetos que se usan para definir nuevos objetos estén previamente definidos. Esto es posible porque existen objetos que se encuentran ya definidos como las constantes formadas por dígitos o las formadas por caracteres alfanuméricos encerrados entre apóstrofes.

Finalmente diremos que las acciones que se definen en un programa, solo pueden hacer uso de objetos definidos en el mismo programa.

En PASCAL, la forma de hacer la descripción de las ACCIONES es mediante las INSTRUCCIONES, que forman la segunda parte del BLOQUE.

Podemos distinguir dos tipos básicos de instrucciones, las instrucciones simples y las instrucciones estructuradas.

Las instrucciones simples son instrucciones que sólo corresponden a una acción; como asignar un valor a una variable, ejecutar una operación de entrada/salida, etc.

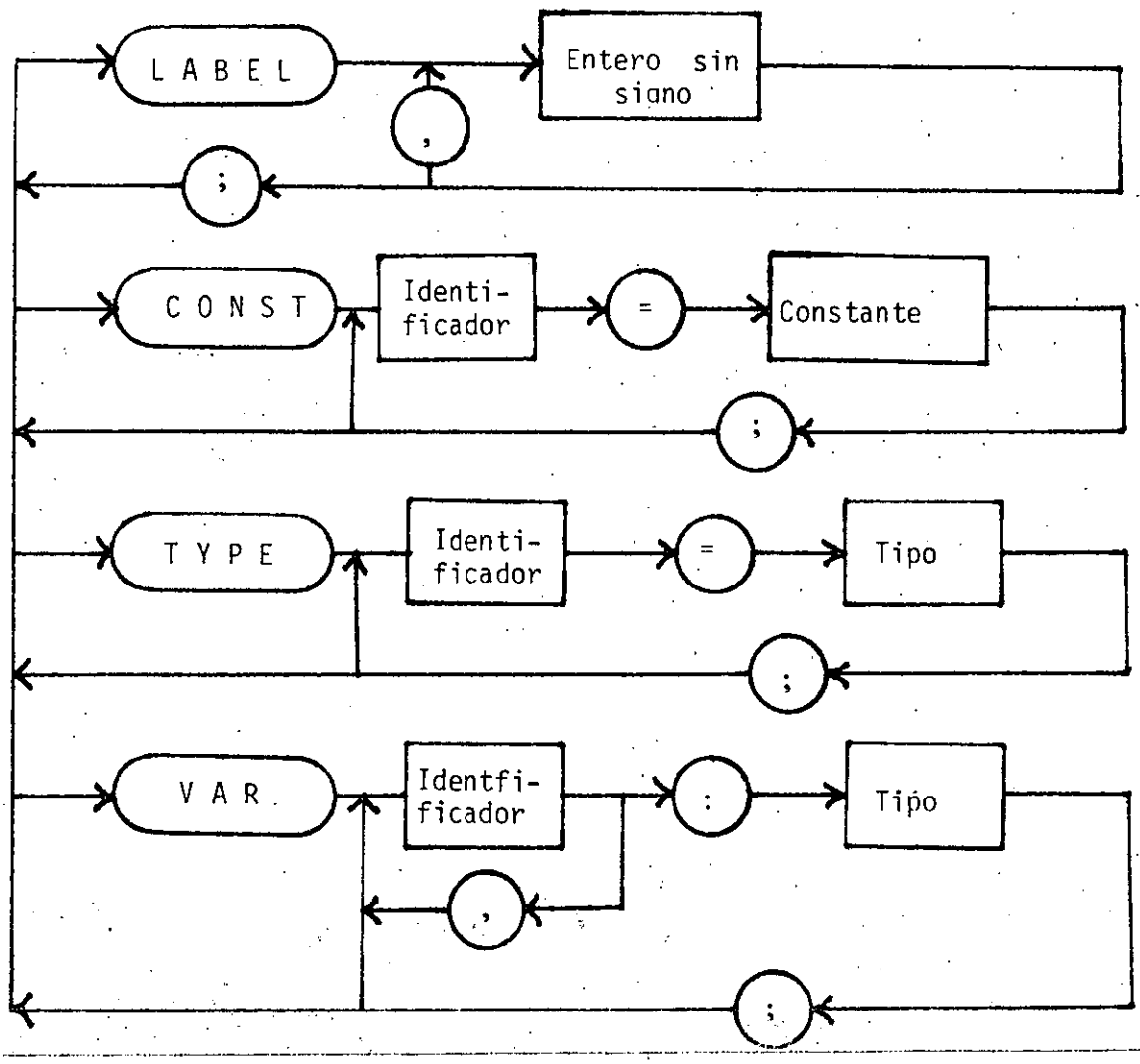
Las instrucciones estructuradas son las que están formadas por las instrucciones que sirven para ejecutar las instrucciones compuestas. Las instrucciones compuestas están formadas por una secuencia de instrucciones que siempre se deben ejecutar con la secuencia en que fueron escritas. Para delimitar las instrucciones que componen una instrucción compuesta se hace uso de los símbolos BEGIN y END el inicio y fin de la secuencia respectivamente.

En el capítulo posterior se estudiarán más profundamente todas las instrucciones, atendiendo a cada uno de los casos particulares de las instrucciones que existen en PASCAL.

TIPO DE DATOS EN PASCAL

Ya se ha mencionado que en un programa de PASCAL es forzoso definir los datos que van a ser objeto de las acciones. La definición de una variable debe incluir una descripción del tipo de variable que se trata y una descripción de su ubicación en la memoria.

FIGURA 2



EL CONCEPTO DE ACCION EN PASCAL

En adelante diremos que una ACCION se entiende como algo que tiene - una duración finita y un efecto bien conocido. Cada acción necesita que - exista el objeto sobre el cual se ejecuta la acción. El efecto de la acción es el cambio en el estado del objeto.

Como un auxiliar para ambas partes, PASCAL brinda la facilidad de definición de CONSTANTES simbólicas. Como se ilustró en la figura 2, es necesario incluir una advertencia al compilador de PASCAL, para indicarle que se comienza la definición de las constantes del bloque. Esta advertencia se hace mediante el símbolo reservado CONST; al cual seguirán las definiciones de constantes que se desee. Como ejemplos de la definición de constantes en -- PASCAL, tenemos :

```
CONST

    blancos='     ';
    cuarenta=40;
    máximo=125.23;
    arroba='  ';
    mínimo=56.24E-12;
    letrero='Este es un letrero o cadena de caracteres
    encerrada entre apóstrofes';
    cantidadoctal=56B;(*este es un comentario para ex-
    plicar que la "B" indica que se trata de una cons-
    tante en código octal (no es una característica -
    general de PASCAL) *)
```

Como vemos el ejemplo; podemos distinguir dos tipos de constantes; - las numéricas y las alfanuméricas o cadenas de caracteres. Las primeras están formadas por dígitos, que dependiendo de la instalación, pueden representarse en más de un código (decimal, octal, binario, etc.). En el ejemplo - podemos apreciar el uso de los COMENTARIOS; que son ignorados por completo por el compilador del lenguaje; y que se inician con la llave ( ) o los símbolos ((*) o (/*) y terminan con la otra llave ( ) o los símbolos (*)) o (*/).

Otra característica importante que notamos es el uso del símbolo punto y coma (;) , que es utilizado como un SEPARADOR, en este caso de definiciones; pero que en general sirve para separar tanto las instrucciones como

las definiciones.

Ahora que ya conocemos algunas caracterisitcas de la definición de - constantes; diremos que las constantes numéricas pueden o no tener punto decimal; y que es válida la notación científica; que se hace mediante el uso del símbolo "E"; que indica que lo que sigue a su derecha es la ponencia -- diez.

Como se indicó anteriormente; en la definición de una variable es -- tan importante, el definir su ubicación en memoria, como el tipo de variable de que se trata.

La afirmación de que es muy importante definir el tipo de una variable, se base en las razones que se enumeran a continuación :

1. Para entender un algoritmo, es esencial conocer el rango de valores de las variables; en otras palabras, el tipo de las variables que estamos utilizando.

2. Resulta una regla general que las unidades de almacenamiento (PALABRAS, BYTES, BITS,etc.) necesitadas para almacenar una variable en memoria, depende del rango de valores que se le defina en el tipo de datos.

3. En general, un programa sólo es válido para un cierto rango de valores de sus objetos. De tal forma que un programa no puede estar bien definido si no se incluye la especifica---- ción del rango de valores y la explicación de los resultados que con ese rango de valores se obtiene.

4. Las operaciones que se realizan con los objetos; en la mayor parte de las ocasiones están intimamente ligadas con la naturaleza de los objetos mismo; así es posible escoger una - división real, división entera, concatenación de caracteres, operaciones del algebra booleana, etc.; y permitir que el compilador de PASCAL verifique que tales operaciones pueden hacerse, según la naturaleza de los tipos de variables que - se definieron.

La forma que acepta un compilador de PASCAL para definir o "inventar" los nuevos tipos de datos, es como se mostró en la figura 2, o sea, que se le indique al compilador que se desea definir uno o más tipos de datos. Esto se hace al incluir el símbolo reservado TYPE; después del cual se pueden definir los tipos que requiera el programa. La forma de hacer la definición de tipo se presenta en los siguientes ejemplos :

```
TYPE

    entero=INTEGER;
    color=(verde,negro,azul,amarillo);
    conjuntodevalidos= SET of char;
    registro=record
        col:color;
        indice:entero;
    END;
    otroarreglo=ARRAY 1..99  of registro;
```

Para PASCAL, existen dos clases de tipos de datos; los datos estructurados y los no-estructurados. De los tipos estructurados existen, los arreglos, los conjuntos, los registros (record's) y los archivos; de los no estructurados existen los definidos por el programador, como el caso del tipo "color" del ejemplo, y los pre-definidos, como son los tipos INTEGER, REAL, BOOLEAN y CHAR. Mención aparte merecen los apuntadores, que son tipos de variables cuyo contenido se usa como la dirección en la cual se encuentra el dato verdadero.

En capítulos subsecuentes se analizarán los tipos no-estructurados definidos por el usuario, los tipos de datos estructurados y los apuntadores.

TIPO DE DATOS NO-ESTRUCTURADOS PRE-DEFINIDOS

A los datos No-Estructurados, también se les conoce como ESCALARES; en los siguientes renglones se explica las características de los ESCALARES pre-definidos en PASCAL; como son los tipos entero, real carácter y booleano.

EL TIPO ENTERO

Si por un momento nos olvidamos de la computadora y el compilador que en ella define a PASCAL, el tipo ENTERO es aquel que reune a todo el conjunto de los número enteros. De la definición anterior, notamos fácilmente que el tipo ENTERO tiene un conjunto infinito de valores; por lo tanto, NO podemos representarlo fielmente en NINGUNA computadora.

Por que sucede lo anteiror?. Porque la memoria de cualquier computadora puede tomar un número finito de estados y por lo tanto no puede representar un conjunto infinito de datos, que sería la forma de representar el tipo ENTERO.En PASCAL, el tipo ENTERO IMPLEMENTADO se llama INTEGER, y según el tipo de computadora y compilador tiene mayor o menor número de elementos.

En el caso de una computadora con palabras de 16 bits, el tipo INTEGER tiene sus fronteras en los número - 32768 y 32767. Esto se debe a que en una computadora con palabra de 16 bits, el bit 15 se usa como bit de signo.

Existe una colección de operadores que se definen para el tipo INTEGER; estos son :

*	MULTIPLICACION
DIV	DIVISION ENTERA
MOD	MODULO (RESIDUO DE LA DIVISION ENTERA)
+	SUMA
-	RESTA

De igual forma se tiene definidas las siguientes funciones; cuyo resultado es del tipo INTEGER.

ABS(X)	VALOR ABSOLUTO (ENTERO)DEL ARGUMENTO "X" (ENTERO).
SQR(X)	CUADRADO (ENTERO) DEL ARGUMENTO "X" (ENTERO)
SUCC(X)	EL SIGUENTE ENTERO DEL ARGUMENTO "X" (ENTERO)
PRED(X)	EL ANTERIOR ENTERO DEL ARGUMENTO"X" (ENTERO).
ROUND(X)	EL ENTERO MAS PROXIMO DEL ARGUMENTO "X" (REAL).
TRUNC(X)	LA PARTE ENTERA DEL ARGUMENTO "X" (REAL).

También, se encuentra definida una constante llamada MAXINT,que representa la frontera superior del tipo INTEGER.

Como última característica del tipo INTEGER; podemos mencionar que todas las operaciones que sean validas en este tipo resultarán con respuestas exactas. Así que el aplicar cualquier operador y función de los antes mencionados; tomando en cuenta las restricciones de rango, se obtendrá una respuesta exacta.

EL TIPO REAL

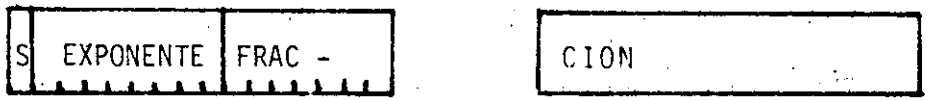
El tipo REAL, como es bien conocido, será el conjunto de todos los números, los racionales y los irracionales. Como en el caso anterior, este es un conjunto con una cantidad infinita de elementos; y por lo tanto no es posible representarlo fielmente en la computadora.

En el caso del tipo REAL, el problema es más grande aún, puesto que a diferencia del tipo INTEGER, que proporciona resultados exactos, siempre y cuando resulten dentro de su rango, en el tipo REAL, no es así; ya que no es posible representar, digamos la fracción igual a un tercio. Como todos sabemos, para los número reales se tiene un contínuo; esto es, que entre dos número tan próximos entre sí como deseemos existe siempre una cantidad infinita de números.

La representación del tipo REAL en PASCAL, depende de la representación de los números reales en la computadora. Una de las formas de representar los reales en computadoras con palabra de 16 bits es la que hace uso de dos palabras; y por medio de dos números enteros representa un intervalo de los reales. En la figura 3, se muestra la representación de los reales para las computadoras de 16 bits, misma que es la que usa el compilador de PASCAL con el cual se han resuelto los ejercicios y problemas de éste documento.

Los valores para la fracción o mantisa quedan dentro del intervalo (0.5,1) y se representan por 24 bits. Como se ve de la figura, solo se tienen 23 bits usados en las palabras que representan los reales. El otro bit (el más significativo de la fracción)se considera siempre en uno y por lo tanto no necesita almacenarse. Esta forma de almacenar la fracción es la llamada forma " Normalizada ".

FIGURA 3



$$X = (1-2*S)*FRACCION*2^{(EXPONENTE - 128)}$$

El exponente puede tomar los valores enteros que se encuentran en el - intervalo (-128,127). Esto es, el bit más significativo del exponente -- se considera como bit de signo. De tal forma que cuando se encuentra en uno su valor se resta de la suma de los demás valores de los bits en 1.

El bit de signo corresponde a positivo cuando tiene el valor cero y a negativo cuando tiene el valor de uno.

Como resultado de esa forma de representación, el tipo REAL implementado, maneja número entre diez elevado a -39 y diez elevado a 38.

En analogía con el tipo INTEGER, existen un cierto número de operadores que se pueden utilizar en combinación con el tipo INTEGER y el tipo REAL. Estos operadores son :

*	MULTIPLICACION
/	DIVISION
+	SUMA
-	RESTA

En este caso hay que tener en mente, que las operaciones que involucren tipos combinados, en realidad implican un mayor procesamiento que las - que involucran sólo operandos del tipo REAL. Esto es debido a que los operandos del tipo INTEGER deben ser convertidos a sus equivalentes en el tipo -- REAL.

En PASCAL existen algunas funciones pre-definidas que pueden tener argumentos del tipo real o integer, y cuyos resultados son del tipo real. Esas funciones son las siguientes:

ABS(X)	VALOR ABSOLUTO
SQR(X)	CUADRADO
SQRT(X)	RAIZ CUADRADA
SIN(X)	SENO TRIGONOMETRICO
COS(X)	COSENO TRIGONOMETRICO
ARCTAN(X)	ARCOTANGENTE TRIGONOMETRICO
LN(X)	LOGARITMO NAUTRAL
EXP(X)	FUNCION EXPONENCIAL

Una ultima aclaración sobre el tipo REAL; aunque se considera como un tipo escalar, no puede ser aplicado en todos los contextos que incluyen los tipos escalares. Así, por ejemplo, no puede servir como índices de arreglos no existe el sucesor de un número real, etc. Se indicarán las excepciones - para el tipo REAL conforme se encuentren los contextos adecuados.

EL TIPO CARACTER

Un caracter es un elemento del conjunto ordenado formado por las le-- tras, los dígitos y algunos símbolos especiales; como los de puntuación,etc. Dependiendo de la computadora y el sistema que en ella se adopte, cada ca--- racter tiene una representación, que hace que se interprete, tanto en los te clados como en las pantalla o impresoras, como una forma o caracter específi co.

Existen varios códigos para representar a los caracteres en una compu tadora; uno de ellos, y tal vez de los mayormente extendidos es el código -- ASCII (American Standar Code for Information Interchange). En este código se hace corresponder un código de 7 bits con cada uno de los 128 caracteres que lo forma. Por medio de estos caracteres se hace posible la comunicación entre el programador y la computadora. Podemos decir que el tipo CARACTER - es aquel que comprende las letras, los digitos y los caracteres especiales.

Para el compilador de PASCAL, el tipo CARACTER se representa por me-- dio del tipo CHAR. Como desgraciadamente, no se puede asegurar que todas - las implementaciones de PASCAL hagan uso del código ASCII para representar - los caracteres; a continuación se definen los elementos "Mínimos" que deben formar parte del tipo CHAR.

1. El tipo CHAR debe incluir el conjunto alfabéticamente ordenado de las letras mayúsculas de la A a la Z, excluyendo las letras "CH" y "LL".

2. Debe incluir también el conjunto numéricamente ordenado de los diez digitos decimales 0..9.

3. Debe incluir finalmente el caracter blanco.

Una constante que pertenezca al tipo CHAR, se denota como un caracter encerrado entre apóstrofes.

Existen, como para los tipos INTEGER y REAL algunas funciones pre-definidas :

CHR(E)	RESULTA CON EL CARACTER QUE OCUPA EL LUGAR "E" DENTRO DEL CONJUNTO ORDENADO DE LOS CARACTE-- RES.
ORD(C)	RESULTA CON UN ENTERO QUE INDICA EL LUGAR QUE OCUPA EL CARACTER "C" EN EL CONJUNTO ORDENADO DE LOS CARACTERES.
SUCC(C)	RESULTA CON EL CARACTER QUE SIGUE AL CARACTER "C".
PRED(C)	RESULTA CON EL CARACTER QUE PRECEDE AL CARACTER "C".

Para el caso particular del código ASCII, en la primer función, los valores permitidos para el argumento "E" son entre 0 y 128, y el resultado

de la función ORD se encuentra en el mismo intervalo. A las funciones ORD y CHR se les denomina genericamente como funciones de transferencias o traducción de caracter a código.

EL TIPO BOOLEANO

El tipo BOOLEANO es el que tiene como elementos a las constantes TRUE (verdadero) y FALSE (falso), que son definidas originalmente para el álgebra de BOOLE (George Boole 1815-1864)

Aunque no necesita ser definido por el programador, podemos considerar que el tipo BOOLEANO implementado o BOOLEAN, se encuentra definido como:

```
TYPE
    BOOLEAN=(false,true);
```

Al igual que los demás tipos escalares pre-definidos de PASCAL, se cuenta con algunos operadores y funciones definidas para el tipo BOOLEANO. Los operadores booleanos de PASCAL son :

OR	SUMA LOGICA
AND	PRODUCTO LOGICO
NOT	NEGACION LOGICA

Las funciones, que tienen resultados del tipo booleano son :

ODD(X)	Resultado VERDADERO si el entero "X" es impara.
EOLN(F)	Fin de línea en el archivo "F" (explicado posteriormente)
EOF(F)	FIN DEL ARCHIVO "F" (explicado posteriormente)

Como antes se mencionó, los operadores relacionales provocan resultados del tipo BOOLEANO. Estos operadores son : (=, , , , =, IN).

Por medio de la aplicación de algunos operadores relacionales a opera

dos del tipo BOOLEANO, se pueden expresar los siguientes conceptos:

IMPLICACION	como	operandobooleano1 <= operandobooleano2
EQUIVALENCIA	como	operandobooleano1 = operandobooleano2
O EXCLUSIVO	como	operandobooleano1 <> operandobooleano2

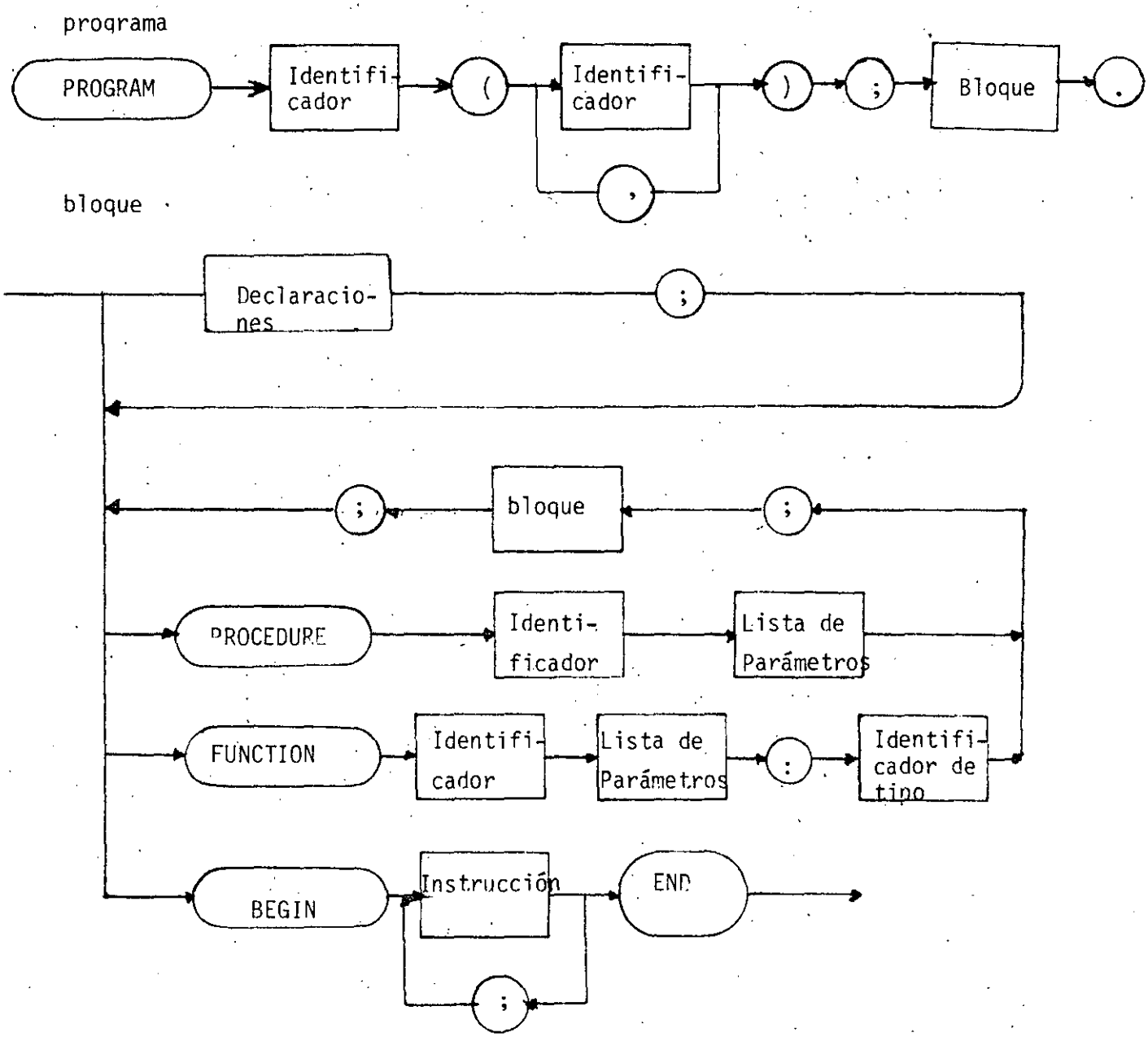
INSTRUCCIONES EN PASCAL

Como antes hemos mencionado, en PASCAL, la descripción de las acciones de un programa se hace por medio de las INSTRUCCIONES. Recordemos que consideramos que existen dos tipos de instrucciones, las no-estructuradas y las estructuradas. En este capítulo se estudian en forma completa ambos tipos de instrucciones.

Como parte de las instrucciones no-estructuradas estudiaremos las instrucciones de asignación y las funciones que hemos considerado como instrucciones de entrada y salida. Dentro de las instrucciones estructuradas serán estudiadas las instrucciones compuestas y aquellas instrucciones con las cuales podempos poner en práctica los conceptos de la Programación Estructurada.

Antes de comenzar a estudiar las instrucciones, en la figura 4 se muestra el diagrama que ilustra la forma en que se escribe un programa de PASCAL. En ese diagrama, se nota que puede escribirse el encabezado; con la palabra PROGRAM indicando el título del programa; después se consideran las declaraciones, que en la figura se ilustra como un conjunto, pero se hace como se muestra en la figura 5; después de las declaraciones se tiene las posibilidad de incluir los símbolos PROCEDURE o FUNCTION, que serviran para indicar la presencia de procedimientos o funciones; que se explicaran más tarde. Después de esos procedimientos y/o funciones deberá aparecer el símbolo BEGIN seguido por las instrucciones del programa y como final el símbolo de END seguido de un punto.

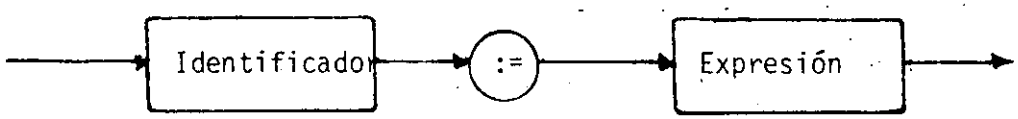
FIGURA 4


programa
PROGRAM
Identifi-cador
(
Identifi-cador
,
)
;
Bloque
.
bloque
Declaracio-nes
;
;
bloque
;
PROCEDURE
Identi-ficador
Lista de Parámetros
FUNCTION
Identifi-cador
Lista de Parámetros
:
Identifi-cador de tipo
BEGIN
Instrucción
;
END

LA ASIGNACION EN PASCAL

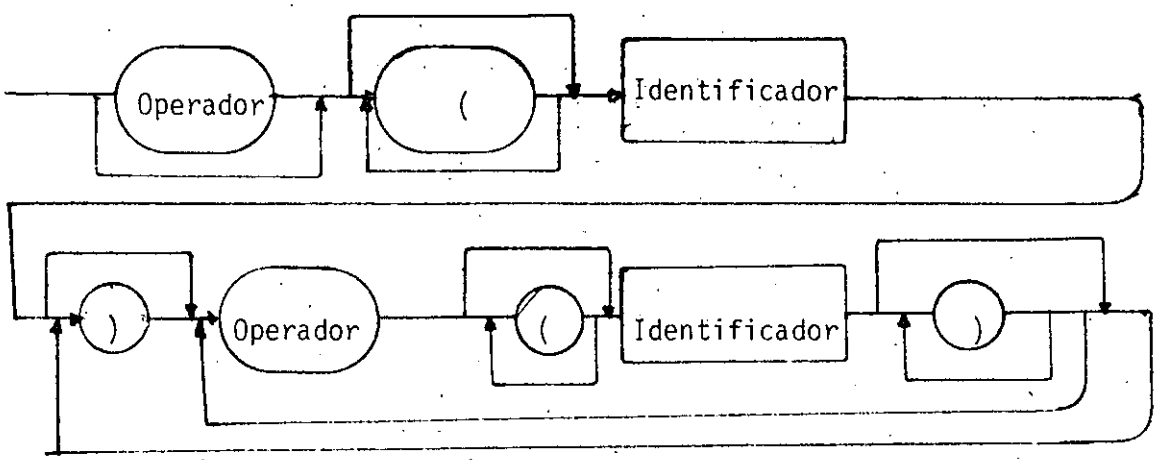
En PASCAL, la acción de la asignación, se realiza como se muestra en la figura 5. Analizando tal figura de izquierda a derecha, observamos que primero se presenta el identificador correspondiente a una variable definida dentro de las declaraciones; a continuación se escribe, el signo de asignación (:=) al cual debe seguir una expresión, que al evaluarse deberá dar un resultado compatible con el tipo de variable a la que se le está realizando la asignación.

FIGURA 5



Según el tipo de la variable, la expresión podrá definirse como se muestra en la figura 6, los operadores de la figura, deberán corresponder al tipo de los objetos que se referencian por medio de los identificadores. Estos identificadores pueden ser de constantes, variables o funciones.

FIGURA 6



Los paréntesis servirán para cambiar el orden de evaluación de las operaciones, según pueda necesiarse en cada caso que defina el

programador. La jerarquía de los operadores es la siguiente:

1o.) PARENTESIS
2o.) FUNCIONES,NOT
3o.) OPERADORES MULTIPLICATIVOS:*/DIV MODAND
4o.) OPERADORES ADITIVOS: + - OR
5o.) OPERADORES RELACIONALES: = <> < > <= >= IN

A continuación se presentan algunos ejemplos de asignación en PASCAL; consideremos, para ello el programa P1 que se muestra a continuación.

```
PROGRAM P1;
CONST
    PI=3.1415926;
VAR
   ENTERA1,
   ENTERA2:INTEGER;
    REAL1,
    REAL2:REAL;
    CAR1,
    BOOL1:BOOLEAN;

(* A CONTINUACION SE HACEN EJEMPLOS DIVERSOS DE ASIGNACIONES *)

BEGIN
    ENTERA1:=4;
    ENTERA2:=ENTERA1*5;
    ENTERA1:=-(43-ENTERA1 DIV 3) + ENTERA2;
    REAL1:=ENTERA1*PI;
    REAL2:=REAL1 - ENTERA2/ENTERA1;
    CAR1:=CHR(ENTERA1);
    CAR2:=CHR(SUCC(TRUNC(REAL1/REAL2));
    BOOL1:=CAR1=CAR2;
    BOOL1:=BOOL1 OR (REAL1 <> REAL2)  AND NOT (BOOL1)
END
```

LA LECTURA Y ESCRIBUTA EN PASCAL

Las funciones que en PASCAL ejecutan las acciones de Lectura y Escritrua son respectivamente READ y WRITE.

La función de lectura tiene dos modalidades; READ y READLN; una de otra difieren en que la primera lee la lista de variables que se le indique sin brincar al inicio de la siguiente línea (en el caso de archivos); -- mientras que la segunda lee la lista de varialbes y brinca al inicio de la siguiente línea.

La función WRITE tiene también dos modalidades; WRITE y WRITELN; la - primera escribe la lista de variables, expresiones y letreros que se le indican sin brincar de renglón. La segunda, además de ejecutar la misma operación que la primera brinca al siguiente renglón.

Los letreros" que se escriben en PASCAL, no son más que constantes - alfanuméricas o cadenas de caracteres encerrados entre apóstrofes.

En PASCAL, además se puede indicar el número de caracteres que se --- desee ocupe la impresión de variables, expresiones o constantes; esto se - hace por medio de la sintaxis que se nota en la última instrucción WRITELN del programa que se muestra a continuación y sirve para ejemplificar las -- instrucciones de entrada y salida en PASCAL.

```
PROGRAM P2;
VAR
   ENTERA:INTEGER;
   FLOTANTE:REAL;
   BOOL: BOOLEAN;
   CAR: CHAR;
( * EJEMPLO DE LECTURA Y ESCRITURA *)

BEGIN
   WRITELN ('DAME UN NUMERO ENTERO');
   READ(ENTERA);
   ENTERA:=SUCC(ENTERA);
   WRITELN(ENTERA);
   WRITELN('DAME UN CARACTER');
   READ(CAR);
   WRITELN(CAR);
   BOOL:=ORD(CARD)  = ENTERA;
   WRITELN(BOOL);
```

```
    WRITELN('DAME UN NUMERO EN PUNTO FLOTANTE');
    READ(FLOTANTE);
    WRITELN(SQRT(FLOTANTE));
    WRITELN('EL PROGRAMA SE TERMINO':14,' ':10;
END.        ENTERA:6, ' ',FLOTANTE:10:10);
```

El resultado de correr este programa es el siguiente:

```
DAME UN NUMERO ENTERO
15
16
DAME UN CARACTER
X
X
FALSE
DAME UN NUMERO EN PUNTO FLOTANTE
1.234570E+14
1.111110E+07
EL PROGRAMA SE              16 123457000000000.0000000000
```

Podemos ver las características de la lectura y escritura de PASCAL; - aunque en el ejemplo se utilizaron como parámetros para la función READ listas de una sôla variable, pueden ser usadas listas con cualquier número de - variables que se necesiten. Es notable el hecho de que la escritura de los enteros se justifica a la derecha (se rellena con blancos a la izquierda -- del digito más significativo del número hasta llenar un campo de 6 lugares); otro hecho interesante es el que se pueden escribir las constantes TRUE o -- FALSE de una expresión booleana.

En el mismo ejemplo notamos las dos formas de escribir una expresión - de punto flotante; la impresión que corresponde a la raiz del número leido se presenta en la misma notación científica con que fué leido, misma que es- la forma normal de presentar una expresión de punto flotante; en el siguiente renglón se la presenta con punto fijo; y con el número de lugares especificados.

Es en el último renglón donde se aprecia la posibilidad de los forma-- tos de PASCAL; en el caso de la cadena (letrero) que se escribe en primer -- término, se preenta truncada porque el campo que se le asignó (14) resulta insuficiente para alojar la totalidad de la cadena; en caso de que sobrasen espacios; la cadena se justifica a la derecha, como es evidente del hecho de

imprimir un espacio en un campo de diez lugares. La variable entera está - presentada dentro de un campo de 6 lugares y por eso se justifica a la dere cha. Por último en el caso de la variable en punto flotante, como el pri- mer campo (10) no es suficiente para alojar el número, el efecto es que se amplía el campo; en caso de que el campo fuese mayor que el lugar requerido para escribir el número se justifica a la derecha; en cambio el campo de de cimales (el segundo 10) siempre permanece fijo, en caso de tener una re-- presentación con mayor número de decimales, el último decimal del campo se redondea.

Con esta revisión de las instrucciones de asignación y de entrada y salida, hemos terminado con la presentación de las instrucciones simples o no-estructuradas en PASCAL.

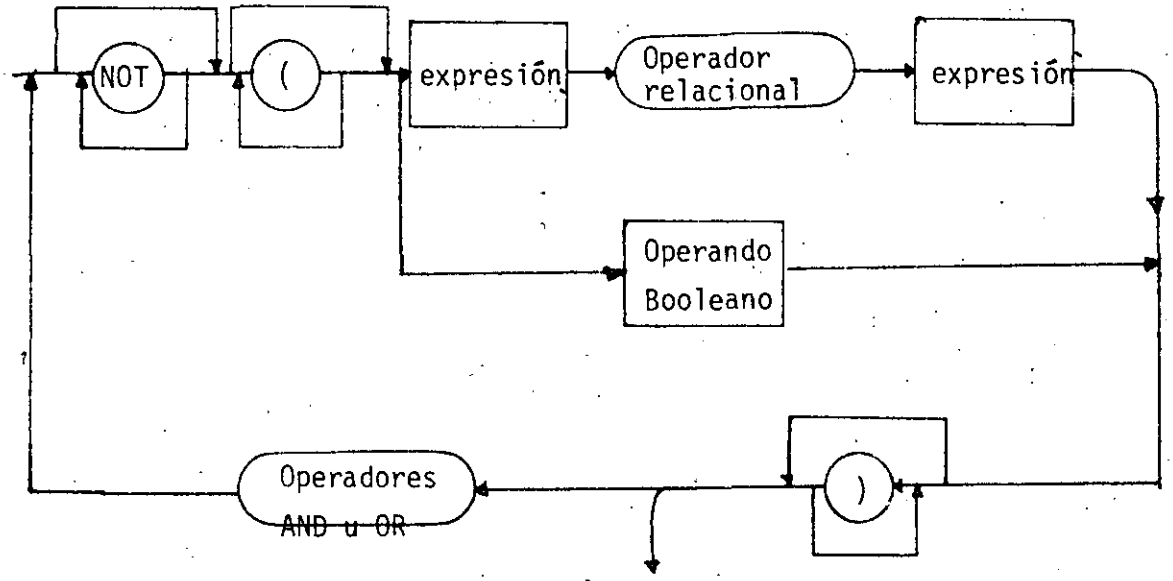
LA INSTRUCCION COMPUESTA EN PASCAL

La instrucción compuesta en PASCAL, es aquella que esta formada - por instrucciones no-estructuradas o estructuradas que deben ejecutarse --- siempre en la secuencia con que fueron escritas. Para delimitar su exten-- sión se hace uso de los símbolos BEGIN (al principio) y END (al final). Un ejemplo de estas instrucciones es el cuerpo mismo de un programa de PAS- CAL; como puede verse el programa anterior; todas las instrucciones se en-- cierran entre los simbolos BEGIN y END.

LAS CONDICIONES EN PASCAL

Las condiciones son preguntas que se pueden responder con respuestas de cierto o falso. En PASCAL estas preguntas se expresan con la sintaxis que se muestra en la figura 7.

FIGURA 7



De la figura observamos que una condición puede constar de una o más expresiones booleadas conectadas por medio de los operadores lógicos :- AND, OR y NOT. Cada una de las expresiones booleanas puede constar de cuando menos un operando booleano (variable o constante), o bien de dos operandos de otro tipo relacionados por medio de un operador relacional. Pero -- siempre al final de la evaluación de una condición el resultado será verdadero o falso.

A continuación se presentan algunos ejemplos de condiciones escri-

tas en PASCAL

```
BOOL;
ENTERA1  = ENTERA2;
ADD(ENTERA1);
CAR IN C'A'. 'B', 'C', 'K' ) ;
CAR  = 'X';
(REAL1  = REAL2) OR (ENTERA1  ENTERA2) AND NOT BOOL;
```

LA INSTRUCCION IF___ THEN___ELSE___.

En el estudio de los conceptos de Programación Estructurada se -- presentó la estructura de DECISION. En PASCAL esta estructura se realiza - por medio de la instruccion IF - THEN - ELSE.

La sintaxis de la instrucción de decisión e PASCAL es la que se - indica a continuación

```
IF condicion THEN
   instruccion1
ELSE
   instruccion2
(*ENDIF*);
```

Esta forna se puede convertir fácilmente a la IF - THEN simplemen te omitiendo tanto el símbolo ELSE como la instrucción que le corresponde. En todo caso, las instrucciones pueden ser estructuradas o no estructuradas y se debe recordar que antes del símbolo ELSE no se debe incluir el separador (;) puesto que el símbolo ELSE no corresponde a ninguna instrucción, -- sino que es también un separador. En caso de tener la necesidad de tener más de una instrucción, se deberá crear un bloque (BEGIN-END), tanto para - el THEN como el ELSE.

El comentario (*ENDIF*) es utilizado únicamente como auxiliar para visualizar el término de la instrucción IF, la cual se termina con el. primer; que encuentre.

En los siguientes ejemplos se muestra el uso de la estructura IF-THEN-ELSE.

```
PROGRAM P3;
VAR
   ENTERA1,ENTERA2:INTEGER;
   REAL1,REAL2:REAL;
   BOOL:BOOLEAN;
BEGIN
   WRITELN('DAME DOS VALORES REALES');
   READ(REAL1,REAL2);
   IF REAL1=REAL2 THEN
      WRITELN('LOS DOS DATOS SON IGUALES')
   ELSE
       WRITELN('LOS DOS DATOS SON DISTINTOS')
   (*ENDIF*);
   WRITELN('DAME DOS VALORES ENTEROS');
   READ(ENTERA1,ENTERA2 )
   IF ENTERA1=ENTERA2 THEN
      IF ENTERA1  0  THEN
         WRITELN('SON MAYORES DE CERO')
      ELSE
         WRITELN('SON MENORES DE CERO')
      (*ENDIF*)
ELSE
      BEGIN
      WRITELN('LOS VALORES ENTEROS SON DISTINTOS');
      IF ENTERA1  ENTRERA2. THEN
         WRITELN(' Y EL PRIMERO ES MAYOR QUE EL SEGUNDO')
      (*ENDIF*)
      END
(*ENDIF*);
IF (REAL1=REAL2)AND(ENTERA1=ENTERA2) THEN
   BOOL:=TRUE
ELSE
   BOOL:=FALSE.
(*ENDIF*)
(*LA INSTRUCCION ANTERIOR ES EQUIVALENTE A:
   BOOL:=(REAL1=REAL2)AND(ENTERA1=ENTERA2)*).
IF BOOL THEN
   IF REAL1  0  THEN
      WRITELN('IGUALDADES Y MAYOR QUE CERO').
   ELSE
```

```
      WRITELN('IGUALDADES Y MAYOR QUE CERO')
   ELSE
      WRITELN('IGUALDADES Y MENOR O IGUAL A CERO')
   (*ENDIF*)
 (*ENDIF*);
END.
```

Una corrida del programa anterior resulta como se ilustra a continuación:

```
DAME DOS VALORES REALES
35.6,124.2E4
LOS DOS DATOS SON DISTINTOS
DAME DOS VALORES ENTEROS
34,56
LOS VALORES ENTEROS SON DISTINTOS
```

Si ponemos atención en la estructura IF que se encuentra controlada por la variable "BOOL"; vemos que se encuentra anidada una estructura -- IF-THEN-ELSE dentro de una estructura IF-THEN; este caso se soluciona -- como se presenta ahí (NO ES RELEVANTE la posición en la que se escriben los símbolos IF, THEN o ELSE, sólo ayudan a la legibilidad). En caso de anidar una estructura IF-THEN dentro de una estructura IF-THEN-ELSE; la sintaxis puede quedar de cualquiera de las dos siguientes formas:

```
IF con1 THEN
   BEGIN
   IF COND2 THEN
      instruccion1
   (*ENDIF.
   END
ELSE
   INSTTUCCION2
(*ENDIF*);
(*ES EQUIVALENTE A LA FORMA*)
IF COND1 THEN
   IF COND2 THEN
      INSTRUCCION1
   (*ENDIF.
ELSE
   INSTRUCCION2
(*ENDIF*);
```

LA INSTRUCCION DE SELECCION EN PASCAL

La versión que PASCAL tiene de la estructura de SELECCION se presenta con la instrucción CASE. Esta instrucción tiene la forma que se muestra a continuación:

```
CASE expresion OF
   etiquetadecase1: instruccion1;
   etiquetadecase2: instruccion2;
      -
      -
      -
   etiquetadecaseN: instruccionN
END
```

La expresión y las etiquetas deben ser del mismo tipo; y siempre deberán ser escalares definidos por el programador, excepto el tipo real. Por lo que conocemos hasta este momento, las expresiones válidas para funcionar como selector del CASE son las expresiones enteras, de caracteres y lógicas. En este último caso, no tiene sentido hacer uso de la instrucción CASE, ya que ese caso se puede resolver fácilmente mediante la instrucción IF-THEN-ELSE.

En el siguiente programa, se presentan ejemplos del uso de la instrucción CASE.

```
PROGRAM P4;
VAR
   ENTERA1:INTEGER;
   CARACTER:CHAR;
(*EL PROGRAMA EJEMPLIFICA LA INSTRUCCION CASE*).
BEGIN
   WRITELN('INDICAME UN NUMERO ENTERO');
   READ(ENTERA1);
   CASE ENTERA1 MOD 10 OF
        0:WRITELN('EL NUMERO QUE TECLEASTE TERMINA EN CERO');
        1,3,5,7,9:WRITELN('EL NUMERO QUE TECLEASTE ES IMPAR').
        2,4,6,8:WRITELN('EL NUMERO QUE TECLEASTE ES PAR Y NO TERMINA EN CERO');
   END;
```

```
   WRITELN('AHORA INDICAME UNA LETRA');
   READ(CARACTER);
   IF NOT (CARACTER IN  'A'..'Z'  ) THEN
      WRITELN('EL CARACTER NO ES UNA LETRA')
   ELSE
       CASE CARACTER OF.
          'A','E','O':WRITELN('LA LETRA ES UNA VOCAL FUERTE');
          'I','U':WRITELN('LA LETRA ES UNA VOCAL DEBIL');
          'R','L':WRITELN('LA LETRA ES UNA CONSONANTE LIQUIDA');
          'B','C','D','E','F','G','H','J','K','M','N','Q','S','T'.
          'V','W','X','Y','Z':WRITELN('LA LETRA ES UNA CONSONANTE
          NO LIQUIDA');
       END
   (*ENDIF*);
END.
```

Al correr este programa, los resultados que se obtienen son:

```
INDICAME UN MUERO ENTERO
136
EL NUMERO QUE TECLEASTE ES PAR Y NO TERMINADO EN CERO
AHORA INDICAME UNA LETRA
R
LA LETRA ES UNA CONSONANTE LIQUIDA
```

LA ITERACION MIENTRAS ___ QUE EN PASCAL

Dentro de las estructuras de programación, se contemplan varios tipos de iteraciones; uno de ellos es la iteración "Mientras-Que". En PASCAL, la forma de indicar esta estructura es por medio de la instrucción -- WHILE. A continuación se presenta la sintaxis de esta instrucción.

```
WHILE condicion DO
   instrucción
(*ENDWHILE*);
```

La condición, como antes se ha mencionado, es una expresión booleana y controla la ejecución de la instrucción, misma que puede ser no-estructurada o estructurada. La instrucción se ejecuta mientras que la condición de control arroje un resultado verdadero.

En el programa P5 se muestran algunos ejemplos del uso de esta instrucción.

```
PROGRAM P5
CONST
   BLANCOS=' ';
VAR
   ENTERA:INTEGER;
   CARACTER:CHAR;
BEGIN
   WRITELN('DAME UN NUMERO MENOR DE 81');
   READ(ENTERA);
   WHILE  ENTERA>0  DO
        BEGIN
        ENTERA:=PRED(ENTERA);
        WRITE('@');
        END
   (*ENDWITILE*);
   CARACTER:=BLANCO;
   WHILE NOT CARACTER IN  'A','E'.'I','O','U'  ) DO
        BEGIN
        WRITELN('DAME AHORA UNA VOCAL');
        READ(CARACTER);
        END
   (*ENDWHILE*);
   WRITELN('LA VOCAL FUE:',CARACTER);
   WRITELN('EL PROGRAMA P5 HA TERMINADO').
END.
```

Al correr este programa, se obtienen los siguientes resultados:

```
DAME UN MUERO MENOR DE 81
15
DAME AHORA UNA VOCAL
1
DAME AHORA UNA VOCAL
E
LA VOCAL FUE: E
EL PROGRAMA P5 HA TERMINADO
```

LA ITERACION REPITE---HASTA QUE --EN PASCAL

La estructura REPITE-HASTA QUE, que es una iteración de las re conocidas por el estilo de programación estructurada, en PASCAL se hace -- por medio de la instrucción REPEAT-UNTIL. La forma de escribir esta ins-- trucción es como se muestra a continuación:

```
REPAT
  instrucción
UNTIL condición
```

Como siempre, la instrucción que se repite puede ser una ins-- trucción no-estructurada o una instrucción estructurada. En el caso especial de una instrucción compuesta, los símbolos delimitadores BEGIN y END de ella, se pueden omitir, ya que los símbolos REPEAT y UNTIL sirven como delimitadores también. No obstante, no resulta incorrecto utilizar ade--- cuadamente los símbolos BEGIN y END en la instrucción REPEAT-UNTIL.

En el programa P6 se muestra el uso de esta instrucción repeti tiva.

```
PROGRAM P6
VAR
   DIGITO,
   CUENTA0,
   CUENTA1,
   CUENTA2,
   CUENTA3,
   CUENTA4,
   CUENTA5,
   CUENTA6,
   CUENTA7,
   CUENTA8,
   CUENTA9:INTEGER;
BEGIN
   CUENTA0:=0;
   CUENTA1:=0;
   CUENTA2:=0;
   CUENTA3:=0;
   CUENTA4:=0;
   CUENTA5:=0;
   CUENTA6:=0;
   CUENTA7:=0;
   CUENTA8:=0;
   CUENTA9:=0;
```

```
WRITELN('ESCRIBA LA LISTA DE DIGITOS;TERMINE');
WRITELN('LA LISTA CON UN NUMERO NEGATIVO');
REPAT
    BEGIN
    REPEAT
       READ(DIGITO)
    UNTIL DIGITO<=9;
    WRITELN('EL DIGITO VALIDO FUE: ; DIGITO:3);
    IF DIGITO  =0  ITEN
       CASE DIGITO OF
           0: CUENTA:=SUCC(CUENTA0);
           1: CUENTA:=SUCC(CUENTA1);
           2: CUENTA2:=SUCC(CUENTA2);
           3: CUENTA3:=SUCC(CUENTA3);
           4: CUENTA4:=SUCC(CUENTA4);
           5: CUENTA5:=SUCC(CUENTA5);
           6: CUENTA6:=SUCC(CUENTA6);
           7: CUENTA7:=SUCC(CUENTA7);
           8: CUENTA8:=SUCC(CUENTA8);
           9: CUENTA9:=SUCC(CUENTA9);
       END;
    (*ENIF*)
    END  (*ESTE "END" NO SE NECESITA* )
UNTIL DIGITO<0:
DIGITO:=0;
REPEAT
    WRITE('LAS VECES QUE SE REPITIO EL DIGITO;
       DIGITO:3,'son':5
    CASE DIGITO OF
       0: WRITELN(CUENTA0);
       1: WRITELN(CUENTA1);
       2: WRITELN(CUENTA2);
       3: WRITELN(CUENTA3);
       4: WRITELN(CUENTA4);
       5: WRITELN(CUENTA5);
       6: WRITELN(CUENTA6);
       7: WRITELN(CUENTA7);
       8: WRITELN(CUENTA8);
       9: WRITELN(CUENTA9);
    END;
    DIGITO: = SUCC(DIGITO).
 UNTIL DIGITO  9;
END.
```

Al correr el programa P6, se pueden obtener los resultados que se muestran a continuación:

```
 ESCRIBA LA LISTA DE DIGITOS; TERMINE
 LA LISTA CON UN NUMERO NEGATIVO
3
EL DIGITO VALIDO FUE:    3
12
1
EL DIGITO VALIDO FUE:    1
111
11
14
45
23
1
EL DIGITO VALIDO FUE:    1
2
EL DIGITO VALIDO FUE:    2
0
EL DIGITO VALIDO FUE:    0
1
EL DIGITO VALIDO FUE:    1
-1
EL DIGITO VALIDO FUE:   -1
LAS VECES QUE SE REPITIO EL DIGITO 0 SON    1
LAS VECES QUE SE REPITIO EL DIGITO 1 SON    3
LAS VECES QUE SE REPITIO EL DIGITO 2 SON    1
LAS VECES QUE SE REPITIO EL DIGITO 3 SON    1
LAS VECES QUE SE REPITIO EL DIGITO 4 SON    0
LAS VECES QUE SE REPITIO EL DIGITO 5 SON    0
LAS VECES QUE SE REPITIO EL DIGITO 6 SON    0
LAS VECES QUE SE REPITIO EL DIGITO 7 SON    0
LAS VECES QUE SE REPITIO EL DIGITO 8 SON    0
LAS VECES QUE SE REPITIO EL DIGITO 9 SON    0
```

LA INSTRUCCION FOR DE PASCAL

La instrucción FOR es utilizada en PASCAL, para ejecutar una instrucción o bloque de instrucción a veces, dependiendo de los límites que sean dados.

La siguiente instrucción FOR de PASCAL es :

```
FOR variable de control:= valor inicial TO(DOWNTO) valor final DO
      instruccion
(*ENDFOR*);
```

El uso del símbolo TO es cuando el valor inicial asignado a la variable de control es menor que el valor final; cuando es el caso contrario, se usa el símbolo DOWNTO. El incremento siempre es de una unidad; positiva cuando se usa TO es negativa cuando se usa DOWNTO.

Como en todas las instrucciones anteriores, la instrucción que se repite puede ser estructurada o no-estructurada. La variable de control puede ser del tipo INTEGER, CHAR, BOOLEAN o no-estructurada definida por el programador.

En el siguiente programa se ejemplifica el uso de la instrucción FOR.

```
PROGRAM P7;
VAR
   FUERTES,
   DEBILES
   CONTADOR:INTEGER;
   VOCAL: CHAR;
   CORRECTO: BOOLEAN;
BEGIN
   FOR CONTADOR:=1 TO 10 DO
       BEGIN
       WRITELN('DAME LA VOCAL NUMERO:;CONTADOR:3,' :');
       REPEAT
           READ(VOCAL);
           CORRECTO:=VOCAL IN  'A','E','I','o','U' ;
           IF CORRECTO THEN
               CASE VOCAL OF
                  'A','E','O':FUERTES:=SUCC(FUERTES);
```

```
                'U','I':DEBILES:=SUCC(DEBILES);
            END
          (*ENDIF*);
        UNTIL CORRECTO
        END
    (*ENDFOR*);
    WRITELN(' EL NUMERO DE VOCALES FUERTES FUE:FUERTES:3);
    WRITELN('EL NUMERO DE VOCALES DEBILES FUE:;DEBILES:3)

END.
```

Al correr este programa, los resultados son:

```
DAME LA VOCAL NUMERO:   1:
C
D
A
DAME LA VOCAL NUMERO:   2:
I
DAME LA VOCAL NUMERO:   3:
E
DAME LA VOCAL NUMERO:   4:
D
E
DAME LA VOCAL NUMERO:   5:
J
I
DAME LA VOCAL NUMERO:   6:
I
DAME LA VOCAL NUMERO:   7:
O
DAME LA VOCAL NUMERO:   8:
U
DAME LA VOCAL NUMERO:   9:
U
DAME LA VOCAL NUMERO:  10:
A
EL NUMERO DE VOCALES  FUERTES FUE:   5
EL NUMERO DE VOCALES  DEBILES FUE:   5
```

LOS PROCEDIMIENTOS Y LAS FUNCIONES EN PASCAL

Antes de entrar de lleno en la explicación de los procedimientos y las funciones, se presentan algunos conceptos comunes de ambas facilidades de PASCAL.

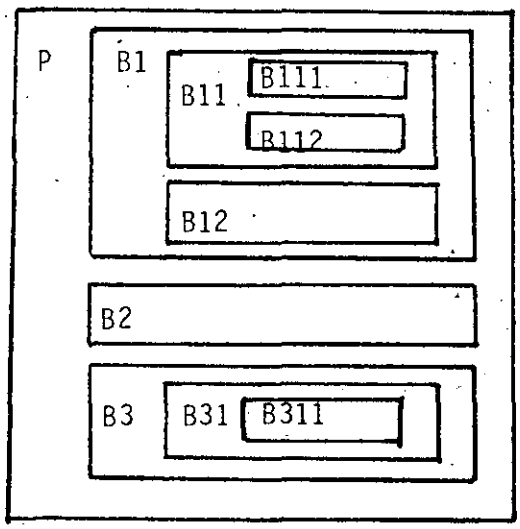
EL CONCEPTO DEL ALCANCE EN PASCAL

Si revisamos el diagrama de sintaxis presentado en la figura, vemos que los procedimientos y las funciones tienen un esquema muy parecido al del programa de PASCAL.

De hecho, un procedimiento o una función, a semejanza de un programa, posee un encabezado y un bloque. Ya sabemos que dentro de un programa se deben DECLARAR todas las constantes, tipos y variables que usa; asimismo sabemos, refiriéndonos de nuevo a la figura 4, que el bloque de un programa se pueden definir los procedimientos y funciones que sean necesarios. Esto es, en un programa escrito en PASCAL se deben definir todos los OBJETOS que usa. Estas definiciones se hacen dentro del bloque. Como los procedimientos y las funciones poseen sus propios bloques, dentro de ellos se pueden definir sus PROPIOS objetos. Estos objetos son llamados OBJETOS LOCALES y tienen validez solo dentro del bloque en el cual se definen. EL ALCANCE de dichos objetos es local al procedimiento o función en cuyo bloque están definidos. En otras palabras, la validez de los identificadores de esos objetos locales está restringida al tiempo de ejecución del procedimiento o función donde están definidos. Finalmente, podemos afirmar que los objetos locales solo existen durante la ejecución del procedimiento o función al cual pertenecen.

En contraste con el alcance de los objetos locales, el alcance de los objetos GLOBALES está definido para la ejecución completa del programa En realidad no es un concepto distinto, ya que los objetos GLOBALES son los que están definidos dentro del bloque del programa, y por lo tanto el tiempo de ejecución para los objetos globales es el del programa mismo.

FIGURA 8



Para distinguir el alcance de los bloques, introduzcamos el concepto de nivel de anidamiento. Este nivel nos indicará la jerarquía del bloque al que nos referíamos. En adelante, entenderemos que un bloque puede ser tanto el programa, un procedimiento o una función. Así, el bloque del programa tiene un nivel de anidamiento cero y es jerárquicamente el bloque de mayor importancia y alcance. El bloque de un procedimiento o función que se encuentre anidado en el programa tendrá un nivel de anidamiento uno, y así sucesivamente.

En un bloque de cualquier nivel de anidamiento pueden existir cualquier número de bloques del siguiente nivel. Estos bloques serán independientes entre sí, y podrán referirse uno al otro, sólo con la restricción de que un bloque (procedimiento o función) que es "llamado" esté --

previamente definido.

Un esquema de anidamiento se muestra en la figura 7. De la figura observamos que el bloque del programa principal (P) se encuentra "Engloba bando" a todos los demás bloques; pero que el bloque B1 solo engloba a los bloques B11,B12,B111 y B112. En tanto que el bloque B2 no tiene bloques anidados, mientras que el bloque B3, tiene anidados a los blques B31 y B311.

También de la figura 7, podemos ver que el programa P puede llamar a los bloques B1, B2 y B3. Como B1, B2 y B3 están en el mismo nivel de anidamiento de un bloque (el del programa) se pueden "llamar" entre sí; en caso de que B1 sea llamado por alguno de los otros dos bloques, se debe utilizar una proposición especial que se emcniona más adelante. Algo que no es posible es que B11 o B12 sean "llamados" por B31 o B2; esto por que no son bloques locales de ellos. O sea, solo el bloque B1 puede activar a loas bloques B11 o B12.

La razón de este comportameinto es debida a que los bloques del nivel dos, pertenecen a diferentes bloques del nivel uno y, por lo tanto, sólo algunos de ellos "existen" simultáneamente. En el caso de la figura, cuando se ejecuta el bloque B1, están presentes los Bloques B11 y B12. Los bloques B111 y B112 son locales de B11 y son inaccesibles a cualquier otro bloque, incluyendo a los de menor nivel de anidamiento.

De la misma figura podemos obtener la tabla 7 en la que se muestra para cada bloque cuales son los objetos a los cuales puede hacer referencia.

BLOQUE	OBJETOS A LOS QUE ALCANZA
P	P
B1	B1, P
B11	B11,B1, P
B111	B111,B11, B1, P
B112	B112, B11, B1, P
B12	B12, B1, P
B2	B2, P
B3	B3, P
B31	B31, B3, P
B311	B311, B31, B3, P

Los objetos de un bloque de cualquier nivel de anidamiento, como antes dijimos, son locales, si alguno de estos identificadores coinciden con el identificador de un objeto de un nivel anterior, sólo es accesible, por el identificador, el objeto definido en el nivel mas alto. Esto se ejemplifica y se entiende facilmente por medio del programa P8.

```
PROGRAM P8;
VAR
    IDENTIFICADOR1,
    IDENTIFICADOR2:INTEGER;
    PROCEDURE BLOQUE1;
    VAR
        IDENTIFICADOR1:INTEGER;
    BEGIN
        IDENTIFICADOR1:=-10;
        IDENTIFICADOR2:=IDENTIFICADOR2-1;
        WRITELN('ESTE LETRERO SE ORIGINA EN EL BLOQUE1, DONDE:');
        WRITELN('IDENTIFICADOR1 VALE: ',IDENTIFICADOR1:4);
        WRITELN('IDENTIFICADOR2 VALE: ',IDENTIFICADOR2:4);
    END;
BEGIN
    IDENTIFICADOR1:=20;
    IDENTIFICADOR2:=3;
    WRITELN('ANTES DE EJECUTAR EL PROCEDIMIENTO LOS VALORES GLOBALES',
            ' SON');
    WRITELN('IDENTIFICADOR1 =',IDENTIFICADOR1:4);
    WRITELN('IDENTIFICADOR2 =',IDENTIFICADOR2:4);
    WRITELN;WRITELN;
    BLOQUE1;
    WRITELN('DESPUES DE LA EJECUCION DEL PROCEDIMIENTO LOS VALORES SON');
    WRITELN('IDENTIFICADOR1 =',IDENTIFICADOR1:4);
    WRITELN('IDENTIFICADOR2 =',IDENTIFICADOR2:4);
    WRITELN;WRITELN;
END.
```

Cuando se corre este programa se obtienen los siguientes resultados:

```
ANTES DE EJECUTAR EL PROCEDIMIENTO LOS VALORES GLOBALES SON:
IDENTIFICADOR1 =  20
IDENTIFICADOR2 =   3

ESTE LETRERO SE ORIGINA EN EL BLOQUE1, DONDE
IDENTIFICADOR1 VALE:  -10
IDENTIFICADOR2 VALE:    2

DESPUES DE LA EJECUCION DEL PROCEDIMIENTO, LOS VALORES SON
IDENTIFICADOR1 =   20
IDENTIFICADOR2 =    2
```

Si analizamos los resultados del programa, notamos que una vez que se ejecuta el procedimiento "BLOQUE1", el valor de "IDENTIFICADOR1" aparentemente ha cambiado; obviamente, no es así, ya que el objeto global que tiene el nombre de IDENTIFICADOR1, conserva el valor asignado al inicio del programa, en cambio con el otro objeto global no sucede así, ya que es un objeto que puede alcanzarse perfectamente desde el procedimiento. Por eso es que notamos que la variable "IDENTIFICADOR2", tiene distinto valor al terminar la ejecución del procedimiento.

LOS PROCEDIMIENTOS EN PASCAL

Los procedimientos son una forma con que PASCAL construye el concepto de la RUTINA, los procedimientos son secuencias de instrucciones a las cuales se les asocia un nombre propio.

El nombre de procedimiento se asigna en el encabezado del mismo; la sintaxis del encabezado de un procedimiento se presenta a continuación:

PROCEDURE nombre (lista de parámetros);

El "nombre", es entonces el identificador con el cual se asociará al procedimiento. La lista de parámetros es una secuencia de identificadores separados por coma para declarar varios nombres de un solo tipo, y por punto y coma cuando se desea declarar más parámetros. La lista de parámetros se puede omitir; no así el separador (;), al cual sigue el bloque del procedimiento.

Como es natural, en el bloque del procedimiento, pueden existir declaraciones de todo tipo de objetos locales; o sea, constantes, variables, tipos, procedimientos y funciones.

El llamado o INVOCACION de un procedimiento se realiza de una manera muy sencilla, esto es, se escribe el nombre del procedimiento seguido por la lista de parámetros (si existen) y el separador (;) si es válido sintácticamente.

En el Programa P8 se vió un ejemplo de la declaración de un procedimiento y su invocación desde el bloque del programa.

LAS FUNCIONES EN PASCAL

A diferiencia de los procedimientos, mismos que pueden ejecutar sus acciones y no reportar ninguna clase de resultado; las funciones, forzozamente deben reportar un resultado.

Es por esto que el encabezado de una función difiere notablemente del encabezado del programa o del procedimiento; a continuación se muestra la definición de la sintaxis necesaria para declarar una función:

FUNCTION nombre (lista de parámetros) : tipo;

El significado del nombre y de la lista de parámetros es análoga al nombre y lista de parámetros del procedimiento, Aparece un elemento nuevo; este es el"tipo". Este indica el tipo de resultado que arroja la función. Este resultado solamente puede ser del tipo no-estructurado o apuntador, el tipo no-estructurado puede pertenecer tanto a los tipos escalares pre-definidos, como a los tipos escalares definidos p or el programador.

Para invocar a una función se le debe hacer parte de una expresión, esto es, debe existir una variable a la cual se le asigne la expresión en la cual parte la función. En el caso de que el tipo de la variable sea booleano; la función se puede usar como una condición; o bien como una parte de una condición.

En el Programa P9 se presentan ejemplos de la declaración y uso de funciones.

```
PROGRAM P9;
VAR
    CARACTER:CHAR;
    SIDIGITO:BOOLEAN;
    (*ESTE PROGRAMA DETECTA DIGITOS DENTRO DE UNA LISTA DE CARACTERES*)
    FUNCTION ESDIGITO:BOOLEAN:
    BEGIN
        IF CARACTER IN ['0'..'9'] THEN
            ESDIGITO:=TRUE;
        ELSE
            ESDIGITO:=FALSE;
        (*ENDIF*)
    END;
    FUNCTION ESCARACTER:BOOLEAN;
    BEGIN
        IF ORD(CARACTER)>=40B THEN
```

```
            ESCARACTER:=TRUE;
        ELSE
            ESCARACTER:=FALSE;
        (*ENDIF*)
    END;
BEGIN
    WRITELN('COMIENZA A DARME LA LISTA DE CARACTERES');
    WRITELN('PARA TERMINAR TECLEA UNA "!"');
    REPEAT
        READ(CARACTER);
        READLN;
        IF CARACTER THEN
            BEGIN
            WRITELN('EL CARACTER ES IMPRIMIBLE');
            SIDIGITO:=ESDIGITO;
            IF SIDIGITO THEN
                WRITELN('SI ES UN DIGITO')
            ELSE
                WRITELN('NO ES DIGITO')
            (*ENDIF*);
            END
        ELSE
            WRITELN('NO ES UN CARACTER IMPRIMIBLE')
        (*ENDIF*);
    UNTIL CARACTER='!'
END.
```

Este programa, no es la solución óptima para el proble,a que resuelve, sin embargo, es tal que permite demostrar facilmente el uso de la funciones. Cabe hacer la aclaración que no todas las -- funciones deban ser del tipo booleano.

Un rasgo distintivo de las funciones es que dentro de su blo que, el identificador de la función debe funcionar como receptor- de un resultado, que es el mismo que será transmitido por su nom- bre; nótesen las asignaciones de las constantes TRUE y FALSE hacia los nombres de las funciones del programa P9.

LOS TIPOS DE PARAMETROS EN PASCAL

Es en este apartado, donde vamos analizar las listas de parámetros. Lo que digamos es válido tanto para las funciones como para los procedimientos.

La lista de parámetros que se presenta en la declaración de un procedimiento o función es llamada la lista de parámetros formales. Estos parámetros son los que sirven para dar "forma" al algoritmo que se realiza en el procedimiento o función.

La lista de parámetros que se representa en la invocación de un procedimiento o función es la lista de parámetros efectivos o actuales. con estos parámetros se llama al procedimiento o función

En la lista de parámetros formales se debe especificar tanto el tipo de parámetros, como su clase. Así, un parámetro formal podra ser del tipo real, integer, boolean, etc. y su clase podrá ser de entrada, entrada y salida o rutinas.

La correspondencia entre los parámetros formales y los parámetros actuales es por posición; así, al primer parámetro formal de la lista le corresponde el primer prámetro de la lista de actuales, al segundo formal, el segundo actual, etc. Como es natural, los tipos de una y otra clase de parámetros deben ser compatibles.

La lista de los parámetros actuales está formada, en general por expresiones separadas por comas. La lista de los parametros formales, pueden estar formada por varias listas, cada una separada por el separador sintáctico (;). Dependiendo de la clase de parámetros que se trate, la lista puede ir precedida por la palabra VAR en el caso de parametros del tipo entrada y salida, por la palabra FUNCTION o la palabra PROCEDURE en el caso de parametros de Rutinas, o sin ningún símbolo en el caso de los parámetros de entrada.

Como se ha visto existen tres clases genéricas de parámetros; en realidad esta división se refiere a la forma en que permiten - el paso de la información. Hay los parámetros de entrada, los de- entraday salida y fanalmente los parametros de rutinas.

A continuación se representa un ejemplo de la declaración de los tres tipos de parametros en Pascal.

```
PROCEDURE EJEMDEPARAMETROS (ENTRADA1,ENTRADA2:INTEGER;ENTRADA3:REAL;
        (*ESTOS PARAMETROS SON DE ENTRADA*)
        VAR ENTSAL1,ENTSAL2:INTEGER; VAR LOG1:BOOLEAN;
        (*ESTOS PARAMETROS SON DE ENTRADA-SALIDA*)
        PROCEDURE FF; FUNCTION XX:CHAR);
        (*ESTOS DEL TIPO DE DIRECCION DE RUTINA*)
```

De la declaración anterior, los parámetros entrada1, entrada2 y entrada3, son parámetros que se comportan como variables locales del bloque y tienen cada llamada valores diferentes, esto es, es - tas variables locales tienen valores iniciales que estan determina dos por el valor de los parámetros actuales en cada una de las eje cuciones del bloque.

Los parámetros de entrada salida tienen acceso a los objetos- que son externos al bloque, por medio de los nombres que tienen en esta lista de parámetros formales. Esto implica que cualquier modi ficación en el objeto determinado por alguno de estos identificado res provoca que el objeto definido por el parámetro actual corres- pondiente también sufra la modificación.

Finalmente, los parametros de rutinas provocan que con el i - dentificador local de procedimiento o función, se invoque al proce dimiento o función que se utilizó como parametro efectivo en el - llamado al bloque. Esto es, con un mismo identificador (el paráme- tro formal) se puede invocar a mas de un bloque (Procedimiento o - Función) ue actúe como parámetro efectivo en cada llamada al blo- que que tiene el parámetro de la clase rutina.

LA RECURSIVIDAD EN PASCAL

Podrá el lector, explicar el concepto de recursividad?. Si no es así entonces podemos decir que la recursión o recursividad se presenta cuando un objeto para definirse necesita referirse así mismo, aunque mas conocida en el mundo de las matemáticas, la recursividad se encuentra aveces en la vida diaria. Tal vez el lector ha visto fotos que se contienen asi misma, o ha visto el efecto de que una camara de televión enfoque un monitor de la señal que ella envia.

No obstante, para los fines de nuestro enfoque generalmente serán mas útiles, para nosotros las aplicaciones de la recursividad en las matemáticas. Es de sobra conocida la definición recursiva de la función factorial de los números naturales y otras. También tienen definiciones recursivas los raboles binarios y otro tipo de estructuras de datos que son recursivas por naturaleza.

Las herramientas con las que en pascal se puede hacer la recursividad son los procemientos y las funciones. La forma de hacerlas es obviamente, el llamado o invocación del procedimiento o función dentro de su bloque mismo.

Si nos referimos a la figura 8 el bloque B1 puede ser llamado recursivamente bien desde las instrucciones de su bloque, o bien desde dentro de los bloques B11, B12, B111 o B112. En estos últimos casos el programador debe tener cuidado para identificar correctamente la recursividad y no perder de vista el hecho de que se puede presentar con invocaciones del nombre del bloque, dentro de sí mismo, aunque esto sea desde un bloque anidado en el.

Los algoritmos recursivos son particularmente poderesos cuando se aplican a resolver problemas que están presentados en términos recursivos. No obstante, el que un problema esté definido recursivamente no garantiza que la solución sea necesariamente la mejor.

TIPOS DE DATOS EN PASCAL

Anteriormente se han introducido los tipos de datos en Pascal; se dijo que existen tipos de datos no-estructurados; pre-definidos por el programador. En este momento estudiaremos los tipos de datos no estructurados definidos por el programador y los tipos de datos estructurados (que finalmente el programador define).

TIPOS DE DATOS NO-ESTRUCTURADOS DEFINIDOS POR EL PROGRAMADOR

En pascal existen dos tipos de datos no-estructurados que pueden definirse; estos son los tipos ESCALAR y SUBRANGO. A continuación se estudian ambos tipos de datos.

EL TIPO ESCALAR

Suponga que está haciendo un programa que levantará una estadística acerca del número de accidentes que se presentan en un cierto lapso, por cada uno de los vehículos utilizados. Los vehículos que participan en la estadística son: motocicleta, automóvil, autobus, ferrocarril y avión. A continuación presentamos un procedimiento que sirve para interrogar al usuario del programa acerca del número de accidentes reportados por cada tipo de vehículo; para cada tipo de vehículo existe una variable entera que sirve como un contador.

```
PROGRAM P10;
VAR
    MOTOACC,
    AUTOACC.
    BUSACC,
    FERROACC,
    AVIONACC:INTEGER;
    PROCEDURE CUENTA;
    VAR
        ACCIDENTES,
        VEHICULO:INTEGER;
```

```
BEGIN
    VEHICULO:=1;
    REPEAT
        WRITE('DAME POR FAVOR EL NUMERO DE ACCIDENTES EN ');
        CASE VEHICULO OF
            1: WRITELN('MOTOCICLETA');
            2: WRITELN('AUTOMOVIL');
            3: WRITELN('AUTOBUS');
            4: WRITELN('FERROCARRIL');
            5: WRITELN('AVION')
        END;
        READLN(ACCIDENTES);
        CASE VEHICULO OF
            1: MOTOACC:=ACCIDENTES;
            2: AUTO&CC:=ACCIDENTES;
            3: BUSACC:=ACCIDENTES;
            4: FERROACC:=ACCIDENTES:
            5: AVIONACC:=ACCIDENTES
        END;
        VEHICULO:=SUCC(VEHICULO)
    UNTIL VEHICULO 5
END;
```

Este procedimiento, resuelve el problema que se indico; sólo que al programador le queda la necesidad de codificar su información y recordar que el "1" se asocia con, la "motocicleta", el "2" con el "automovil", etc. Por medio de una declaración del tipo ESCALAR, digamos, definiendo un tipo "tipovehiculo", se puede evitar la codificación , que si bien en el caso que se presente no tiene la menor dificultad; si le resta claridad al programa. Además consideremos el caso que dentro de un mismo programa existan varias dx esas codificaciones, esto comienza a hacer que la claridad del programa se pierda muy rápidamente y sea más dificil entenderlo.

En el ejemplo que se presenta a continuación se hace la declaración del tipo escalar "tipovehiculo" y se utiliza, de tal forma que la utilidad de este tipo de datos sea más evidente.

```
PROGRAM P1;
VAR
    MOTOACC,
    AUTOACC,
    BUSACC,
    FERROACC,
    AVIONACC:INTEGER;
    PROCEDURE CUENTA;
    TYPE
        TIPOVEHICULO=(MOTOCICLETA,AUTOMOVIL,AUTOBUS,FERROCARRIL,AVION);
    VAR
        ACCIDENTES:INTEGER;
        VEHICULO:TIPOVEHICULO;
    BEGIN
        VEHICULO:=MOTOCICLETA;
        REPEAT
            WRITE('DAME POR FAVOR EL NUMERO DE ACCIDENTES EN ');
            CASE VEHICULO OF
                MOTOCICLETA: WRITELN('MOTOCICLETA');
                AUTOMOVIL: WRITELN('AUTOMOVIL');
                AUTOBUS: WRITELN('AUTOBUS');
                FERROCARRIL: WRITELN('FERROCARRIL');
                AVION: WRITELN('AVION');
            END;
            READLN(ACCIDENTES);
            CASE VEHICULO OF
                MOTOCICLETA: MOTOACC:=ACCIDENTES;
                AUTOMOVIL: AUTOACC:=ACCIDENTES;
                AUTOBUS: BUSACC:=ACCIDENTES;
                FERROCARRIL: FERROACC:=ACCIDENTES;
                AVION: AVIONACC:=ACCIDENTES
            END;
            VEHICULO:=SUCC(VEHICULO)
        UNTIL VEHICULO AVION
    END;
```

Este nuevo procedimiento funciona también como el otro, sólo- que con la ventaja de que la claridad es completa; así, cuando el- vehículo es motocicleta, se ejecutan las acciones correspondientes a la motocicleta; y sin necesidad de conservar la "codificación",- que puede ser fuente de numerosos errores.

Como vimos del último ejemplo, la sintaxis para definir un - tipo ESCALAR es muy simple y se limita a la enumeración de una lista de símbolos que forman los valores permitidos para el tipo que- se define. A continuación presentamos la sintaxis para la defini - ción del tipo ESCALAR.

```
IDENTIFICADOR DEL TIPO =( identificador,
                          identificador,
                          identificador,
                             . .
                             . .
                          identificador)
```

También es posible definir una variable del tipo escal mediante la forma:

```
LISTA DE VARIABLES : (identificador,
                             . . .
                             . . .
                          identificador)
```

La primera forma la usaremos cuando sea necesario definir el- tipo; o sea que dentro del bloque del programa existen varios lugares donde se necesite declarar variables del tipo que se definió.- La segunda puede ser útil cuando solo exista un lugar donde se definen todas la ocurrencias de variables del tipo que se define.

Para los tipos escalares se encuentran definidas las funciones:

ORD, SUCC y PRED

El tipo subrango es el medio por el cual PASCAL permite asignar nombre propio a un subrango de tipo escalar que ya esté definido. Este tipo escalar del cual se tiene un subrango, es llamado el ESCALAR ASOCIADO del subrango.

El tipo subrango, no es más que una indicación de la frontera inferior y de la frontera superior que abarca el subrango dentro del tipo escalar asociado. Los tipos escalares asociados pueden ser INTEGER, CHAR, o escalares definidos por el programadôr.

Las reglas de sintaxis que se siguen para definir un tipo subrango se expresan a continuación:

IDENTIFICADOR DE TIPO = identificador.. identificador;

o mediante la forma:

LISTA DE VARIABLES : identificador .. identificador;

Algunas declaraciones de tipo subrango válidas se presentan a continuación:

```
TYPE
    SINSIGNO=0..65535;
    ELCARAC='A'..'R';
    POCOSVEHICULOS=MOTOCICLETA..AUTOBUS;
VAR
    X,Y,Z:SINSIGNO;
    R:56..90;
    LETRA:ELCARAC;
    TRANSPORTE:POCOSVEHICULOS;
```

Es necesario hacer notar que no se verifica la validez de las asignaciones en variables del tipo subrango, en cuanto a que la expresión que se asigna se encuentre o no dentro del subrango, en su lugar se verifica solamente que la expresión sea de un tipo compatible con el tipo escalar asociado al subrango.

EL ARREGLO EN PASCAL

El arreglo en PASCAL es una colección de elementos del mismo-tipo, podemos decir que el arreglo es una estructura homogénea. El tipo de las componentes del arreglo se conoce como tipo de base - del arreglo.

Las reglas de sintaxis que se deben seguir para la definición de un tipo arreglo son las que siguen:

```
IDENTIFICADOR DE TIPO = ARRAY [tipo de índice,
                               tipo de índice,
                                    . .
                                    . .
                               tipo de índice] OF Tipo de base
```

Y para definir la variables de tipo arreglo se puede seguir - la forma sencilla de tener la lista de variables y despues el tipo. o bien:

```
LISTA DE VARIABLES : ARRAY [tipo de índice
                                  . . .
                            tipo de índice] OF Tipo de base
```

En la definición del arreglo no damos la existencia de los tipos de índice; un tipo de índice puede ser un tipo escalar, o bien de los pre-definidos o de los definidos por el programador. Las excepciones son el tipo REAL y el tipo ENTERO, que no pueden usarse-como índices de un arreglo. Para tener índices enteros, no obstante, se puede utilizar un tipo subrango de enteros adecuadamente definidos.

Una característica de gran importancia en los arreglos de Pascal es la que se refiere al hecho que los límites de los arreglos-son fijos; o sea no existen arreglos dínamicos en Pascal.

Como cada una de las variables que forman el arreglo se puede accesar de la misma forma y con la misma rápidez, se dice que el arreglo es una estructura de acceso aleatorio. La forma de accesar una componente de un arreglo, es indicado el nombre del arreglo seguido por los valores de los índices que identifican a la componente, encerrados entre paréntesis cuadrados. Para indicar cada uno de los índices se pueden hacer uso de expresiones que tengan resultados del mismo tipo de los índices. A continuación se presentan algunos ejemplos de definición de arreglos y de acceso a alguna de sus componentes.

```
TYPE
    CADENA= ARRAY[1 .. 80] OF CHAR;
    PAGINA= ARRAY[1 .. 80,1..25] of CHAR;
    TRANSPORTE = ARRAY [1 . . 100] OF  TIPOVEHICULO;
    TRANSPORTE1= ARRAY [TIPOVEHICULO]OF BOOLEAN;
VAR
    LINEA:CADENA;
    PALABRA: ARRAY [1. .30] OF CHAR;
    SINACCIDENTES:TRANSPORTE1;
    (*LOS ACCESOS A COMPONENTES PUEDEN SER COMO SIGUE: *)
    LINEA[1]:='R';
    LINEA [ORD(PALABRA[I] )-32]:='S';
    PALABRA[1] :='A';
    SINACCIDENTES[MOTOCICLETA]:=TRUE:
    etc.
```

El tipo de base de un arreglo puede ser cualquier tipo de los que se pueden definir en PASCAL; si sucede que el tipo de base de un arreglo es otro arreglo, esto es equivalente a agregarle las dimensiones del último al arreglo original y el tipo de base es ahora, el tipo de base del arreglo que sirvía como tipo de base.

EL REGISTRO (RECORD) EN PASCAL

En el arreglo, como ya sabemos, podemos asociar con un nombre común aun grupo de variables del mismo tipo (el tipo de base); cuando necesitamos asociar a un nombre diversos tipos de variables relacionadas entre sí; podemos hacer uso de la estructura llamada registro de pascal. En similitud con el arreglo, mediante el registro podemos asociar un nombre en común a varias variables; pero en el registro no existe el tipo de base; las variables componentes del registro podrán diferir en su tipo.

Por las caraterísticas anotadas en el parrafo anterior, del registro se dice que es una estructura heterógenea; o sea que sus componentes no son todas del mismo tipo.

Es debido a su heterogeneidad, que el registro, es quizas, según N. Wirth la estructura mas flexible de pascal. Una definición mas formal del registro es la que indica que el registro es una estructura formada por variables componentes llamadas "campos". Estos campos pueden tener tipos disitntos y el acceso a un campo de registro se hacer por medio del nombred el campo. La sintaxis que requiere PASCAL para definir un registro es:

```
IDENTIFICADOR DE TIPO = RECORD
      LISTA DE IDENTIFICADORES DE CAMPO = TIPO;
      LISTA DE IDENTIFICADORES DE CAMPO = TIPO;
            . . .
            . . .
      LISTA DE IDENTIFICADORES DE CAMPO = TIPO;
END;
```

En la definición anterior las listas de identificadores de campo estan formadas por identificadores separados por comas, como en los casos de los tipos que se han visto en este manual, también es posible definir un registro con la sintaxis:

```
LISTA DE VARIABLES : RECORD
     LISTA DE IDENTIFICADORES DE CAMPO : TIPO;
          . . .
     LISTA DE IDENTIFICADORES DE CAMPO : TIPO;
END;
```

Cada tipo de campo en un registro puede ser cualquiera de los que se pueden definir en pascal;e sto es, puede haber campos de -- tipo INTEGER, REAL, BOOLEAN, CHAR, escalares definidos por el usuario, subrangos, arreglos, registros, conjuntos, archivos y apuntadores.

Para tener acceso a un campo de un registro; la sintaxis que se necesita seguir es la siguiente:

NOMBRE DE LA VARIABLE.NOMBRE DEL CAMPO

En el siguiente ejemplo encontramos la forma en que se define un registro; y el acceso a sus diferentes campos. Por supuesto se supone la definición adecuada de todos los objetos que se utilizan

```
VAR
    REGPERSONAL: RECORD
        NOMBRE,DIRECCION:ARRAY[1 . . 32] OF CHAR;
        PUESTO:(DIRECTIVO,SECRETARIAL,ADMINISTRATIVO,OPERARIO,
                CALIFICADO,AYUDANTE,MANUAL);
        SALARIO:REAL;
        PERMANENCIA:1 . . 99;
        FECHA_INGRESO: RECORD
            DIA:1. . 31;
            MES:(ENERO,FEBRERO,MARZO,ABRIL,MAYO,JUNIO,JULIO
                 AGOSTO,SEPTIEMBRE,OCTUBRE,NOVIEMBRE,DICIEMBRE);
            AÑO:0. .99
        END;
    END;
BEGIN
```

```
. . .
. . .
IF REINGRESO THEN
    BEGIN
    REGPERSONAL.FECHAINGRESO.DIA:=DIADEHOY;
    REGPERSONAL.FECHAINGRESO.MES:=MESACTUAL;
    REGPERSONAL.PERMANENCIA:=0;
    REGPERSONAL.SUELDO:=SALARIO;
    END
(*ENDIF*);
END.
```

LA INSTRUCCION WITH

Si ponemos atención en el último ejemplo anterior, vemos que existen varias asignaciones en los campos de la variable "REGPERSONAL"; como el nombre de la variable es bastante grande, resulta un tanto molesto el escribir repetidas veces el nombre de esa variable seguido por distintos campos que forman el registro. Como una facilidad para la escritura de instrucciones en las que participan repetidamente los campos de un mismo registro, existe la instrucción WITH.

Por medio de la instrucción WITH se evita la repetición del nombre de la variable que identifica al registro, y tan sólo se escriben los nombre de ,los campos.

La sintaxis que se define para utilizar la instrucción WITH es:

```
WITH lista de variable REGISTRO  DO
    Bloque de instrucciones
(*ENDWITH*);
```

La lista de varibles del tipo REGISTRO incluirá los nombres de los registros cuyos campos se modifiquen; los nombre deberán ir separados por comas. La instrucción podrá ser compuesta y en ella se podra utilizar los identificadores de campo de los registro como nombre de variables.

En el siguiente ejemplo se muestra como es utilizada la instrucción WITH en el caso de las instrucciones del anterior.

```
WITH REGPERSONAL,FECHAINGRESO DO
    BEGIN
    DIA:=DIADEHOY:
    MES:=MESACTUAL;
    PERMANENCIA:=0;
    SUELDO:=SALARIOS;
    END
(*ENDWITH*);
```

EL CONJUNTO EN PASCAL

El tercer tipo de datos estructurados en PASCAL, es el conjunto en PASCAL, esta formado por el conjunto potencia del tipo de base-del conjunto. El conjunto potencia incluye a todos los subconjun - tos posibles del tipo de base, incluyendo el conjunto vacio.

La sintaxis que debemos seguir para definir un conjunto es:

IDENTIFICADOR DE TIPO = SET OF Tipo de Base

El tipo de base de un conjunto puede ser un tipo escalar definido por el programador, o del tipo subrango. Como el conjunto potencia tiene un enorme número de elementos (2 elevado al número de elementos del tipo de base), las diferentes realizaciones de los - compiladores de pascal, ponen límites usualmente pequeños al número de elemento que puede contener un conjunto. Como la construcción de conjunto no es una operación igual a la que hemos conocido - hasta este momento definiremos los constructores de conjuntos. Estos constructores consisten de la enumeración de los elementos del conjunto, encerrados entre parentesis cuadrados; cada uno de los - elementos se separan de los demás por comas (,); algunos ejemplos-de constructores de conjuntos son:

```
[13]
[R-M, J*K]
['A','E','I','O','U']
```

Para todas la variables del tipo conjunto se encuentran definidas- las operaciones:

\+ UNION
* INTERSECCION
\- DIFERIENCIA

Los operadores relacionales que se pueden aplicar son:

= PRUEBA SOBRE EQUIVALENCIA
<> PRUEBA SOBRE NO EQUIVALENCIA
<= >=PRUEBA SOBRE INCLUSION
IN PRUEBA SOBRE INCLUSION DE UN ELEMENTO

EL ARCHIVO EN PASCAL

La última estructura de datos estática de pascal es el archivo; este tiene necesariamente la conexión con los dispositivos de almacenaje masivo de información, o los perifericos de entrada y salida de la computadora; esto es, una variable archivo de pascal se encontrará vinculada a un archivo que se encuentra en un disco o cinta - magnetica o a una impresora de líneas, una terminal, etc.

En adelante entederemos al archivo como un "archivo secuencial". El archivo especifíca una estructura que consiste de una secuencia de componentes, todas del mismo tipo; en la cual no se tiene fija- la longitud de la secuencia. Es está última característica del archivo la que lo distingue claramente del arreglo. Además, dada su- naturalexa, el archivo es una estructura de acceso secuencial, encontraste con el acceso aleatorio del arreglo. De lo anteriormente expuesto se desprende que en un archivo, sólo se tiene acceso di - recto a una componente a la vez.

La sintaxis para definir un archivo es:

IDENTIFICADOR DE TIPO = FILE OF Tipo de Base

El tipo de base de un archivo puede ser cualquiera de los otros tipos existentes en pascal, osea puede ser entero, punto flotante, carácter, booleano, escal, subrango, arreglo, conjunto, registro,- apuntadores y aún otro archivo.

A continuación se presentan algunas características especiales del lenguaje Pascal cuando se refiere a los rachivos.

EOF Es la variable booleana que indica fin del archivo

EOLN Es la variable Booleana que indica Fin de línea

RESET Regresa el apuntador del archivo al inicio de este.

REWRITE Prepara el archivo para escritura, si existe lo borra.

GET Mueve el apuntador a la siguiente posición del archivo

PUT Agrega el contenido del buffer al archivo.

Según la definicón general, corregida, de PASCAL, los procedimiento READ y WRITE deben estar defnidos para cualquier tipo de archivo, pero em algunas implementaciones de PASCAL, esto no se cumple (casi ninguna). En todo caso las operaciones READ y WRITE tienen las siguientes equivalencias.

```
WRITE(ARCH,X)      ARCH↑:=X ; PUT(ARCH)
READ(ARCH,X)       X:=ARCH↑ ; GET(ARCH)
```

LAS ESTRUCTURAS DINAMICAS DE DATOS EN PASCAL (APUNTADORES)

Las variables que tienen un identificador que asocia a la localidad de memoria que ocupan, son llamadas variables estáticas, atendiendo al hecho que su localización en memoria no cambia durante la ejecución del bloque dentro del cual se encuentra definida. Además- de esta forma convecional de definición de variables, en Pascal seencuentra con la facilidad de generar variables dinámicamente; ta - les variables las conoceremos como variables dinámicas.

Las variables dinámicas no se declaran explícitamente en nin - guna parte de un programa de PASCAL, y por lo tanto no se les puede referenciar directamente por ningún identificador. En lugar de eso-

la generación de variables dinámicas introducen un valor apuntador (que no es otra cosa que la dirección de la variable dinamica a la- que se asocia). La forma en que se definen un tipo de variables dinamicas es por medio de la declaración:

IDENTIFICADOR DE TIPO = ↑identificador de tipo

Para definir variables del tipo "apuntador" a variables dinámicas- se puede utilizar cualquiera de las dos siguientes formas:

LISTA DE VARIABLES : Identificador de tipo "apuntador"

o bien

LISTA DE VARIABLES : Identificador de tipo

Supongamos que se ha definido a la variable "eslabon" como apuntador a una variable dinámica del tipo RRR; para referirse a la va - riable dinámica señalada por el apuntador "eslabon", se necesita - escribir:

```
eslabon↑
```

Así son válidos los ejemplos:

```
VAR
    eslabon,auxiliar:↑INTEGER;
        . . .
        . . .
BEGIN
        . . .
    ESLABON↑:=123;
    AUXILIAR:=NIL: (*NIL ES EL ELEMNTO NULO*)
    AUXILIAR:=ESLABON↑;
    AUXILIAR :=ESLABON↑#AUXILIAR↑;
        . . .
        . . .
END:
```

Para la manipulación de los apuntadores podemos hacer uso de dos procedimientos convecionales, que son

NEW (APUN) Crea una variable "dinámica" y su dirección se la asocia a la variable apun

DISPOSE(APUN) Borra la variable "dinamica" apuntada por el apuntador APUN.

En la definición de PASCAL, se considera que se posee un "colector de basura"; este colector de basura se encarga de detrminar dentro de un área de memoria, cuales son los lugares que aunque contienen información de una variable dinámica, esta ya es sólo basura, y el lugar asignado, se puede rescatar para brindar nuevos lugares a las variables que se crean.

La variable dinámica puede tener cualquier tipo de los que se pueden definir en PASCAL y es por eso que se considera que el apuntador es una herramienta fundamental en la construcción de una cantidad ilimitada de estructuras dinámicas de información.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA COMPUTACION Y PROGRAMACION ELECTRONICA

SISTEMAS Y PROGRAMAS

C O B O L

ING. MARIO PALOMAR A.

OCTUBRE, 1984

EL LENGUAJE COBOL - 1

AUNQUE más lentamente que en el campo científico, los ordenadores entraron en el área de gestión con gran fuerza, sobre todo debido a la aparición de lenguajes de alto nivel orientados específicamente a los negocios. El primer lenguaje de este tipo, históricamente hablando, fue el FLOW-MATIC que en 1955 estableció el concepto de lenguajes de programación basados en palabras del lenguaje natural (en este caso el inglés). Fue creado por el doctor Hopper para UNIVAC. No obstante, el lenguaje de gestión que alcanzó más rápidamente la popularidad fue el COBOL, desarrollado a partir de 1959 por CODASYL. Conocido en un principio por COBOL 60, pretendía ser un lenguaje común a todos los ordenadores. Posteriormente han surgido nuevas versiones, por ejemplo: el COBOL ANSI 74, el COBOL-80 de Microsoft, el CIS-COBOL (que facilita el manejo de pantallas) y el RM-COBOL (para microprocesadores).

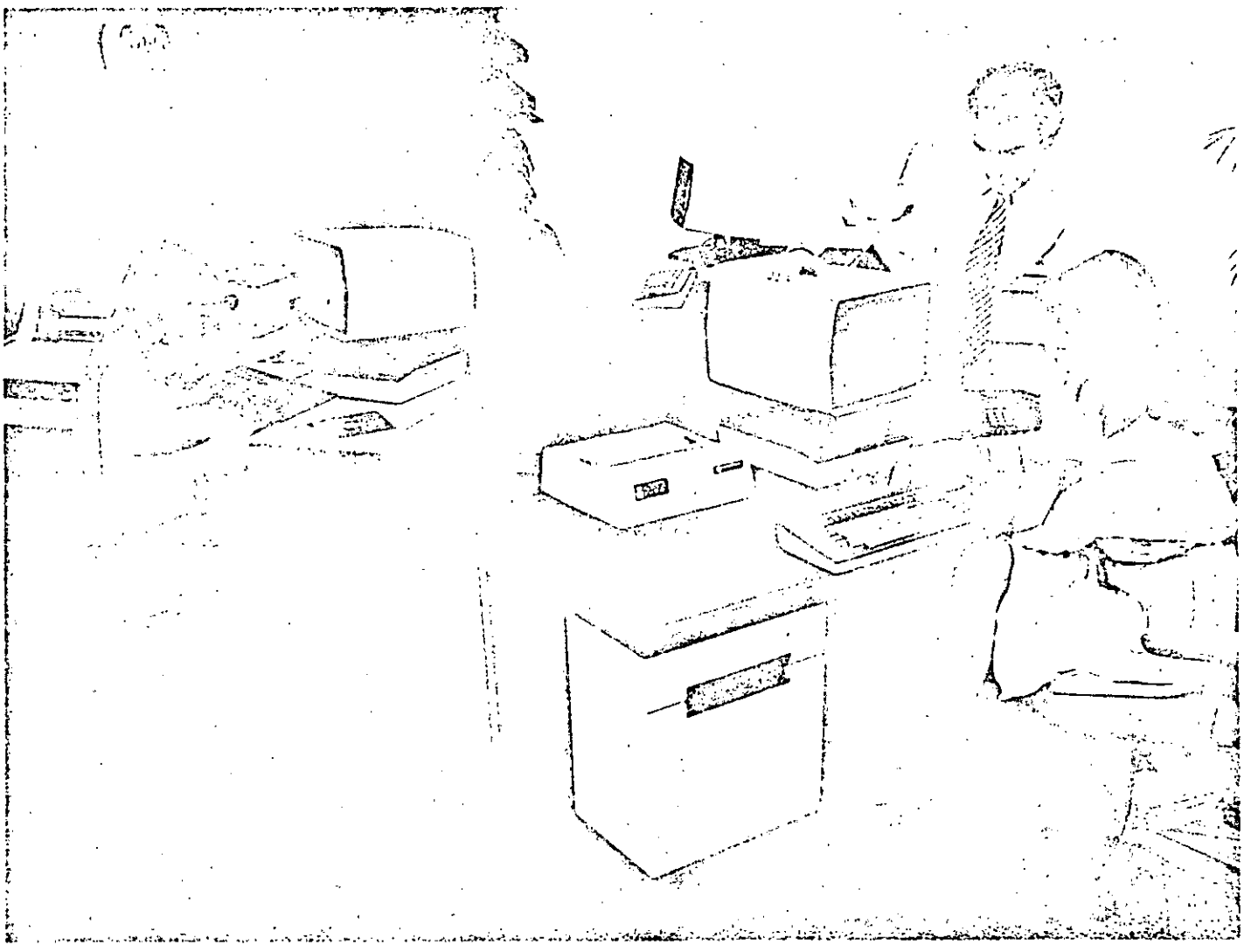
Ventajas e inconvenientes

El lenguaje COBOL tuvo un gran éxito, ya que incluía un concepto básico: el diseño de los datos es independiente de los algoritmos que van a operar con ellos. Este principio permite la definición minuciosa de los elementos a utilizar. Por lo demás, este lenguaje tiene gran capacidad de manejo de campos alfanuméricos, lo que es útil para programar salidas impresas en sistemas de gestión.

A pesar de haber sido superado por otros lenguajes, aún se utiliza ampliamente. Esto se debe a las numerosas aplicaciones desarrolladas a lo largo de veinte años y a la experiencia acumulada por los programadores.

Inicialmente era un lenguaje batch, si bien hoy existen numerosas versiones interactivas que incorporan instrucciones de acceso a pantalla.

Es el lenguaje más estándar de los existentes en la actualidad y los programas escritos en COBOL se pueden implementar fácilmente en distintos ordenadores. Además, su manejo no hace necesario un conocimiento profundo de matemáticas.



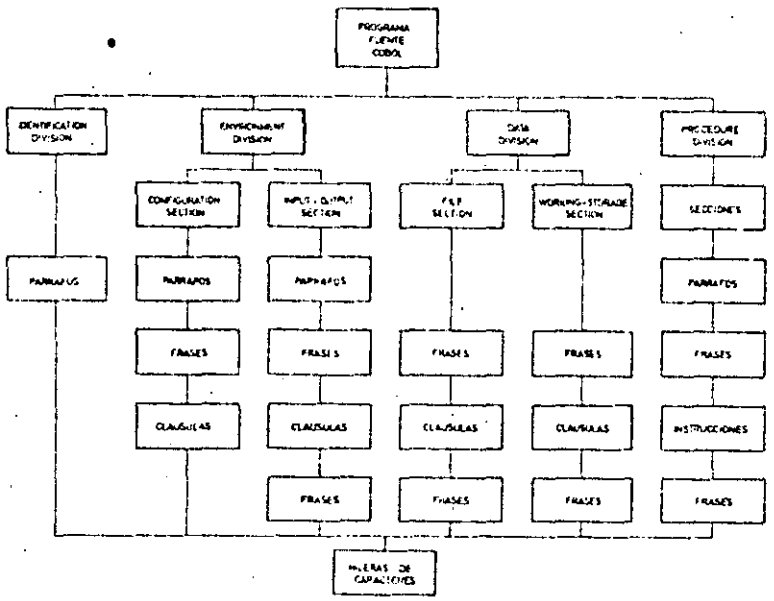
El lenguaje COBOL nació en 1959, orientado específicamente a la gestión administrativa y a los negocios. El objetivo de sus creadores era convertirlo en un lenguaje universal para todos los ordenadores.

EL LENGUAJE COBOL

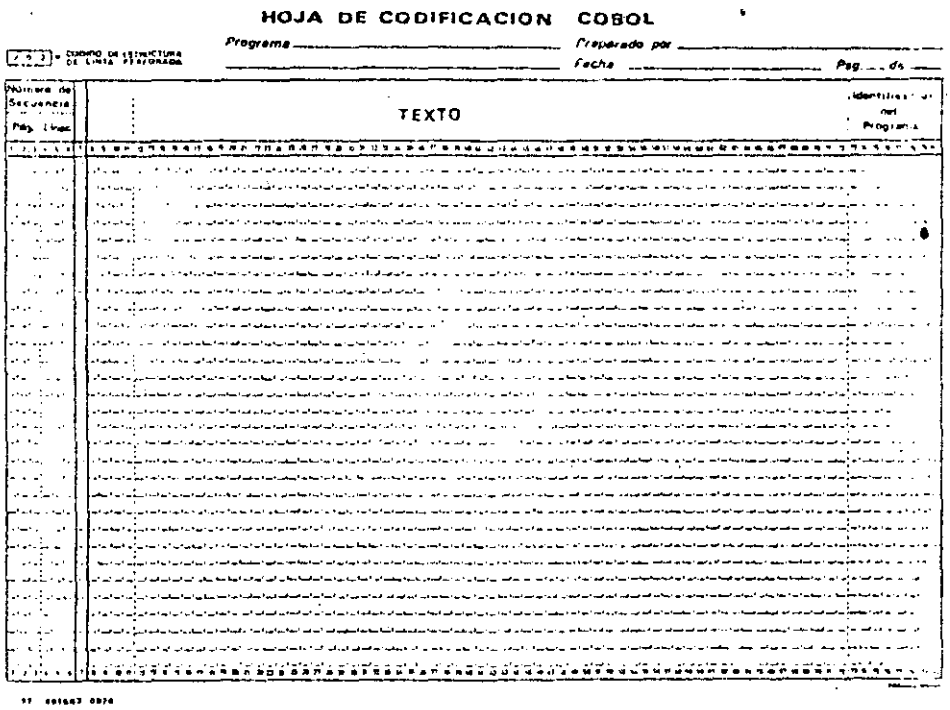
Estructura general

Un programa en lenguaje COBOL se encuentra jerarquizado de la siguiente forma: División, Section, Paragraph, Sentence, Statement, Word y Character. Las divisiones son cuatro, cada una de las cuales informa al compilador de un aspecto del programa. Las divisiones deben escribirse exactamente en el mismo orden en que se reseñan.

— *Identification Division:* Identifica al programa. Además del nombre del mismo, incluye información adicional sobre el autor, fecha, etc.
— *Environment Division:* Adapta el resto del programa a la configuración del sistema y a su sistema operativo. En teoría sería la única división que habría que cambiar si se traslada el programa a otro ordenador.
— *Data Division:* Describe las estructuras de los datos que van a ser procesados.
— *Procedure Division:* Contiene las instrucciones con las que el ordenador procesará los datos; esto es, el programa propiamente dicho.



La misión de las divisiones en las que se estructura el lenguaje COBOL es informar al ordenador de aspectos parciales del programa.



La figura muestra la organización de una hoja de codificación COBOL. Obsérvese la gran similitud que guarda con las hojas de programación en lenguaje FORTRAN.

Tipos de sentencias

Las divisiones se estructuran en secciones, según se indica en el gráfico adjunto. Mientras que los nombres de las secciones de la Environment Division y la Data Division están fijados por el propio lenguaje, en la Procedure Division los elige el programador, seguidos de la palabra «Section».

La Configuration Section describe el ordenador en el que se compila y ejecuta el programa; mientras que la Input-Output Section señala los periféricos utilizados y su asignación a los ficheros.

Las secciones de la Data División son: File Section (para los ficheros), Working Storage Section (para las zonas de maniobra), Constant Section (para las zonas de constantes) y Report Section para la descripción de salidas.

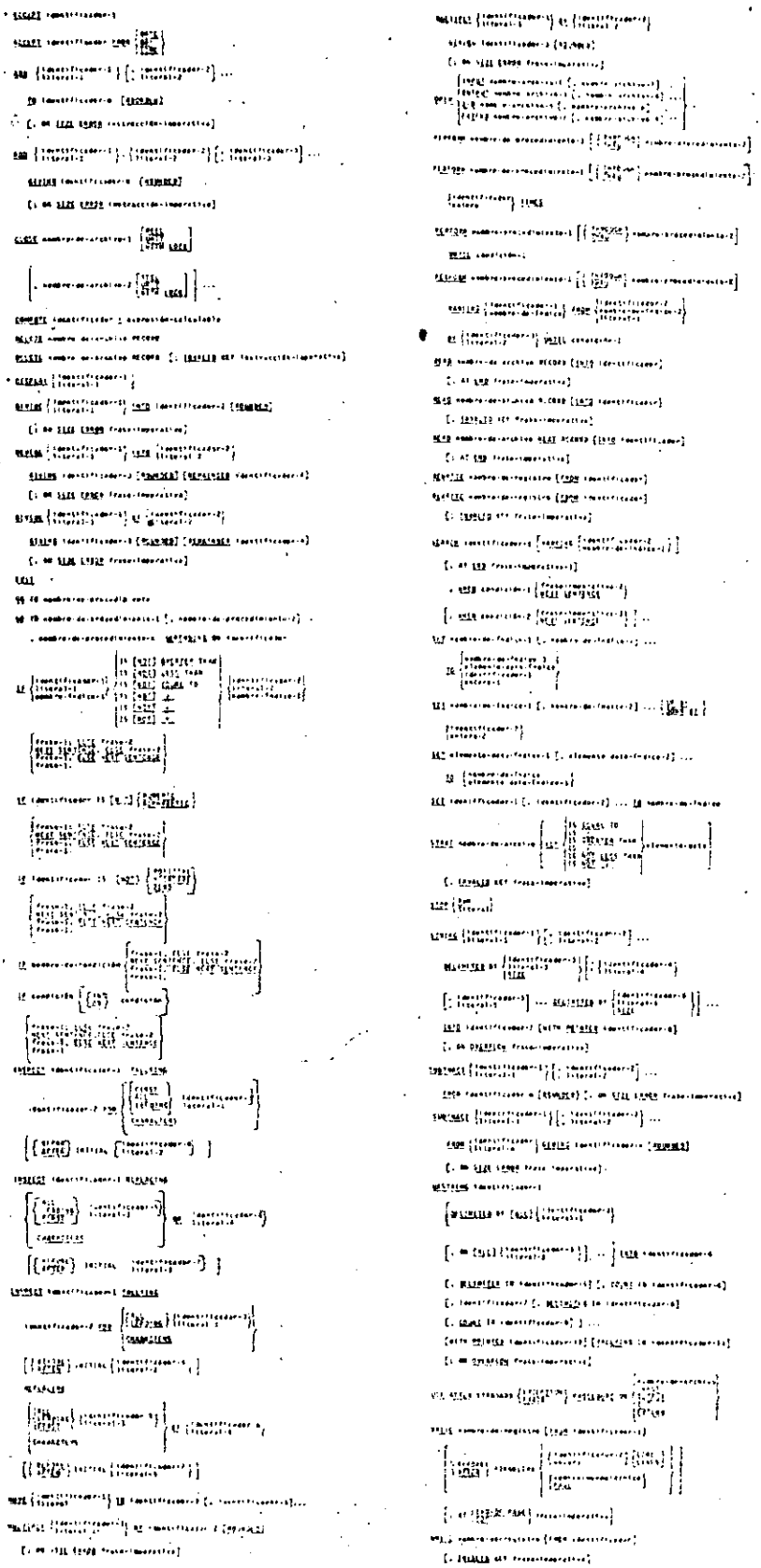
Los párrafos o subdivisiones de las secciones, constan de una o más frases que tienen una función común. Los párrafos de la «Procedure» pueden ser fijados por el programador, mientras que los nombres de los párrafos de las otras instrucciones están fijados por el lenguaje.

Los párrafos de la Environment Division y Data Division están formados por frases (entries) y cláusulas, mientras que los de la Procedure Division lo están por frases e instrucciones.

Una instrucción es una combinación sintácticamente válida de palabras y símbolos, que comienza con un verbo COBOL. Existen tres tipos de instrucciones: del compilador, condicionales e imperativas.

La hoja de codificación

Los programas fuente en lenguaje COBOL se escriben en hojas de codificación normalizadas, que recuerdan a las del FORTRAN. En este caso la línea de continuación se indica en la columna 7, mediante un guión, y la de comentario,



Formato de las instrucciones de la PROCEDURE DIVISION, encargada de contener el programa propiamente dicho.

Glosario

¿Qué significa COBOL?

La palabra COBOL está formada por las iniciales de Comon Business Language (Lenguaje Común Orientado a los Negocios)

¿Qué es el CODASYL?

CODASYL son las siglas de Conference on Data Systems Languages (Conferencia sobre Lenguajes de Sistemas de Datos). Este comite fue convocado por el Departamento de Defensa de los Estados Unidos y estaba formado, entre otros, por las empresas Burroughs, IBM, Honeywell, RCA, Remington, Sperry Rand, Sylvania, USAF, Navy y el National Bureau of Standards.

¿Qué significa la página y línea en el número de secuencia de la hoja de programación COBOL?

Sólo un número de orden, aunque no tienen que estar consecutivos. El criterio general es numerar las líneas de 10 en 10 para permitir la intercalación de nuevas sentencias. La estructura sigue recordando la época en que se trabajaba con fichas perforadas (80 columnas) y la posibilidad de clasificar las fichas mediante el número de secuencia, utilizando una clasificadora.

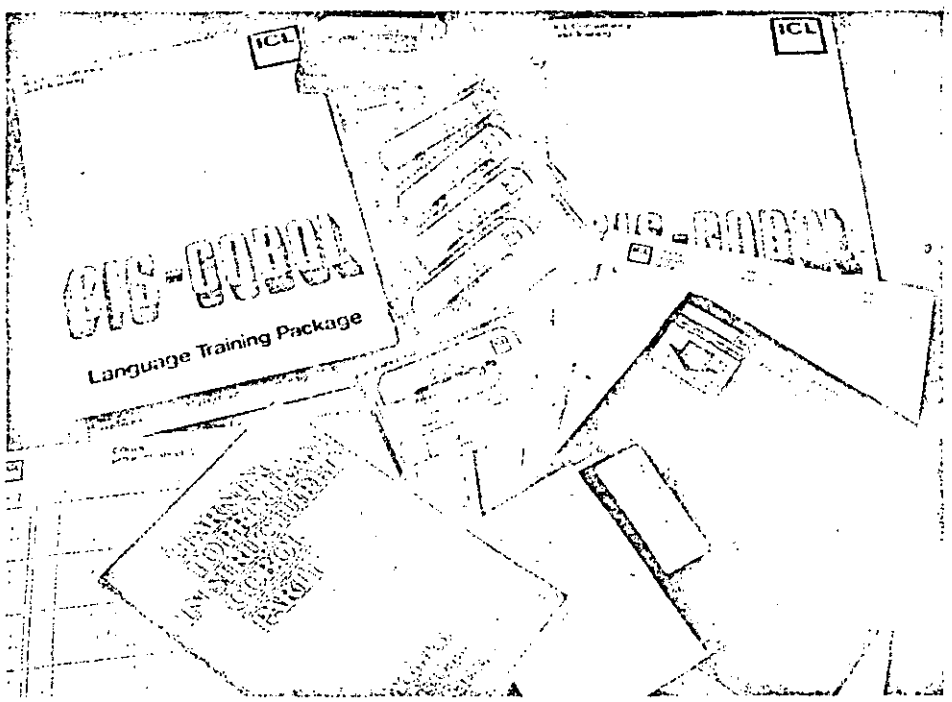
¿Qué se entiende por verbo COBOL?

Un verbo o palabra COBOL es una hilera de caracteres que reconoce el compilador y le permiten generar el programa objeto. Los verbos son palabras reservadas del lenguaje.

EL LENGUAJE COBOL 4

mediante un asterisco en dicha columna 7.
Al codificar un programa en COBOL, o en cualquier otro lenguaje, deben seguirse unas ciertas reglas, cuya misión es disminuir las posibilidades de error. Así, por ejemplo, se debe utilizar lápiz suave y goma de borrar en lugar de bolígrafo; se debe poner un solo carácter en cada posición; escribir ciertos caracteres (tales como el cero y la O, la zeta y el 2, la I o el 1) de forma que se eviten errores; tener cuidado con los puntos, etc.
En los cuadros adjuntos se exponen las instrucciones de la Procedure Division.

En la actualidad existen numerosas versiones o dialectos del COBOL 60 original. Cada uno de ellos supone una mejora o adaptación a necesidades del usuario, pero siempre dentro de la orientación primera: la gestión administrativa y los negocios.



En lenguaje COBOL (Common Business Languaje) pueden encontrarse miles de aplicaciones orientadas a la gestión administrativa, así como en los más diversos formatos para su entrada en el ordenador: disquette, cassette, etc.

Conceptos básicos

Codificación y control de errores

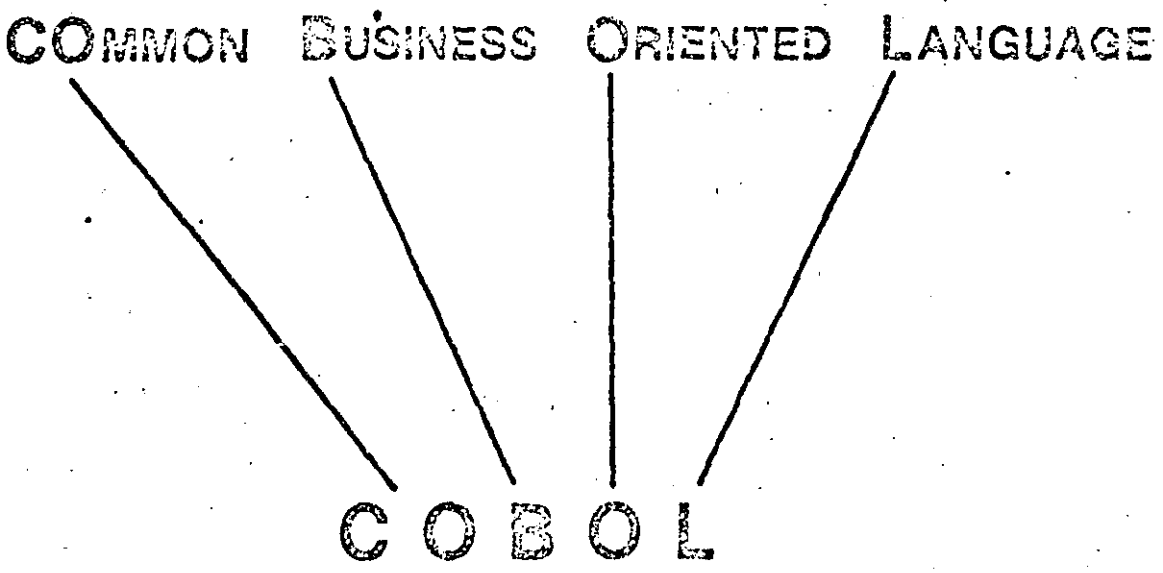
La codificación consiste en establecer una correspondencia entre la información que queremos representar y su representacion, de forma que a cada información le corresponda una y sólo una forma de representación.
Como el ordenador maneja sólo información binaria, toda la información, tanto numérica como alfabética debe representarse mediante cadenas de bits. Se han utilizado multitud de sistemas de codificación de la información, tanto dentro como fuera del ordenador. Veamos algunas de las razones que han servido de base para construir los códigos más usuales.

• *Número de símbolos a representar*
Al tener que codificar las 26 letras, los 10 dígitos y unos 30 símbolos especiales, se necesitan por lo menos 60 configuraciones binarias, por lo que el número mínimo de elementos del código ha de ser 6 bits ($2^6 = 64$ combinaciones posibles). En general se usan 7 u 8 bits por la razón que se explica seguidamente.

• *Detección y corrección de errores*
La necesidad de que no se produzcan errores en las distintas etapas, obliga a introducir mecanismos que detecten y corrijan los errores de forma automática. Esto se consigue utilizando más bits de los necesarios, los llamados bits de paridad. Se dice que un código es óptimo cuando para representar un símbolo se usa el menor número de posiciones binarias posibles. Cuando se emplean más de los estrictamente necesarios se dice que el código es redundante. Es esta redundancia la que asegura la fiabilidad del código.

• *Rendimiento de un código*
Se define el rendimiento de un código como el cociente entre el número de informaciones codificadas y la cantidad total que podrían representarse con el código utilizado. Se da en tanto por ciento.
Así, por ejemplo, si empleamos 6 bits para representar sólo diez dígitos, el rendimiento será:

$$n = 100 \times \frac{10}{2^6} = \frac{1.000}{64} \simeq 15{,}6\ \%$$



V1-1

HANDOUT

COURSE: Basic ANSI COBOL
CODE: H1-1
TITLE: History of COBOL

6

1959 – Conference on Data Systems Language (CODASYL) met to develop a common programming language for business data processing

1960 – Publication of first official report on COBOL

1961 – Release of COBOL – 1961

1963 – Release of COBOL – 1961 extended

- Major additions and modifications included:
  - Addition of sort feature
  - Addition of report writer option
  - Arithmetics to include multiple receiving fields
  - Add and subtract corresponding options

1965 – Release of COBOL – edition 1965

- Included:
  - Options to handle reading, writing, and processing mass storage files
  - Table handling features with indexing and search options

1966 – United States of America Standards Institute, USASI (formerly American Standards Institute, ASI) released standard COBOL based upon COBOL-edition 1965. USA Standards Institute is now called American National Standards Institute (ANSI)

Most recent standard is ANSI-74

HANDOUT

COURSE: Basic ANSI COBOL
CODE: H1 2
TITLE: Benefits of COBOL

7

1. Easy to code because of its similarity to English language

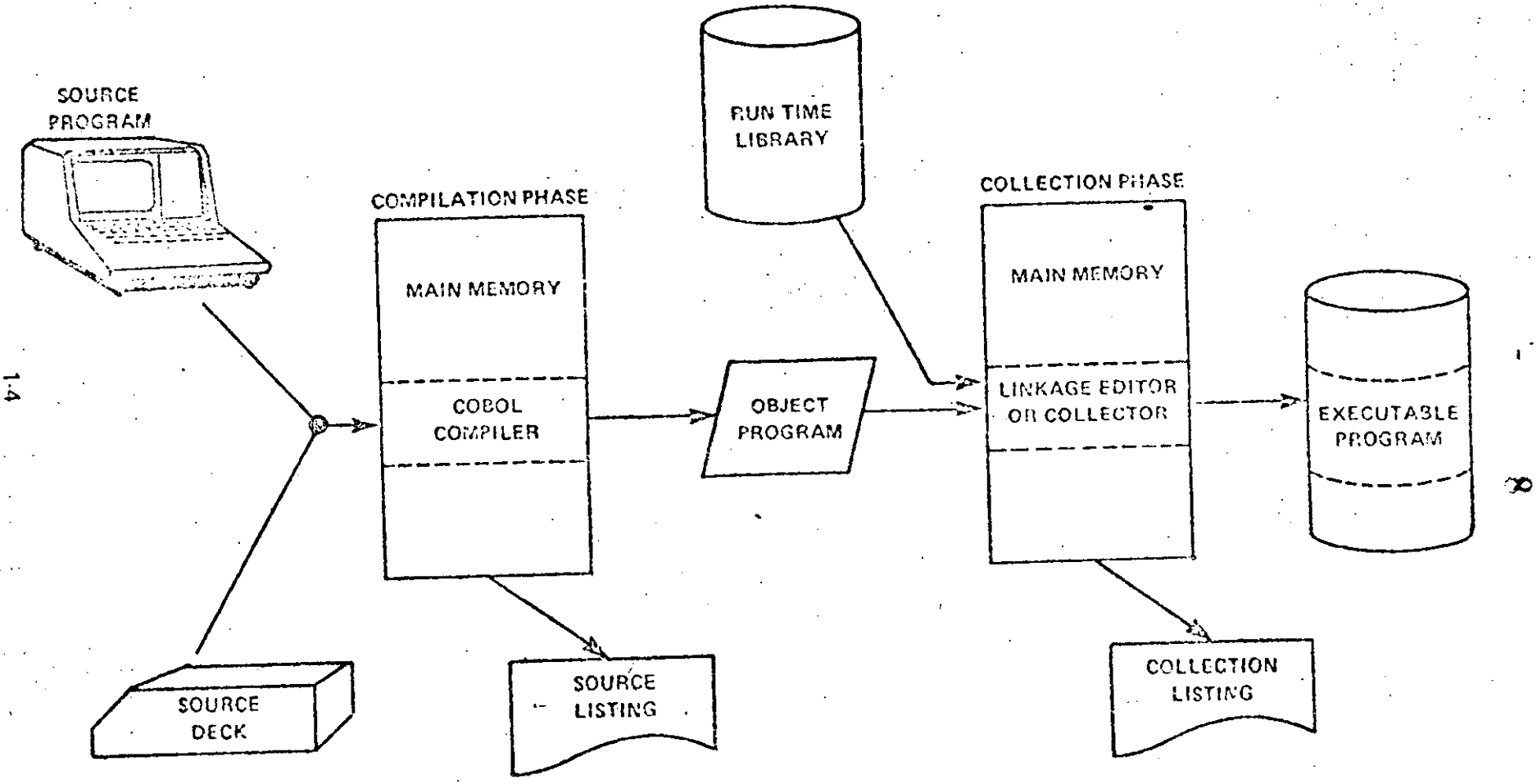
2. Self-documenting

3. Standardized throughout industry

4. Lower costs

5. Reduced training time

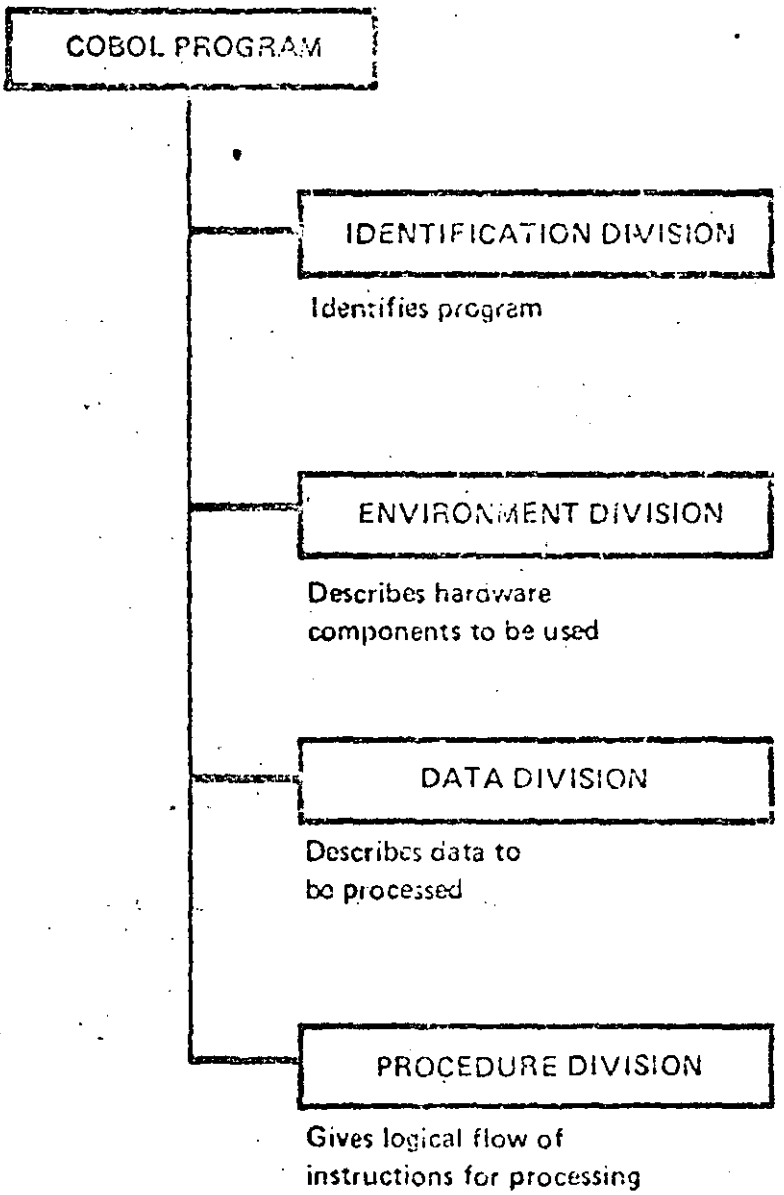
Compilation and Linkage



V1-2

General Structure of a COBOL Program

A COBOL PROGRAM IS DIVIDED INTO FOUR DIVISIONS, EACH OF WHICH IS REQUIRED FOR EVERY PROGRAM WRITTEN. THE FOUR DIVISIONS MUST APPEAR IN A SPECIFIC ORDER.



Logical Breakdown

Program

Division

Section

Paragraphs

Sentences

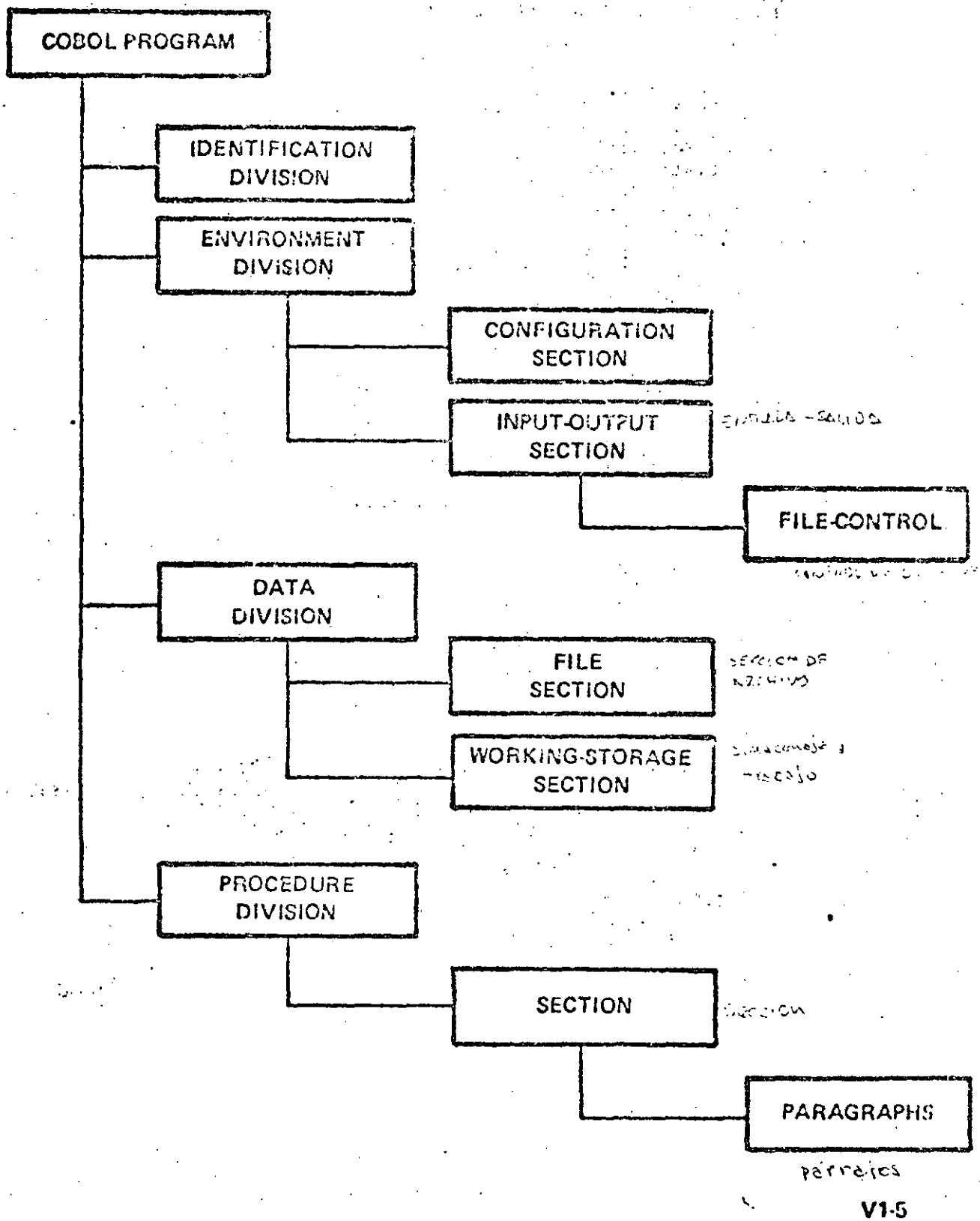
Statements

Words

Characters

V1-4

Four Divisions of a COBOL Program



V1-5

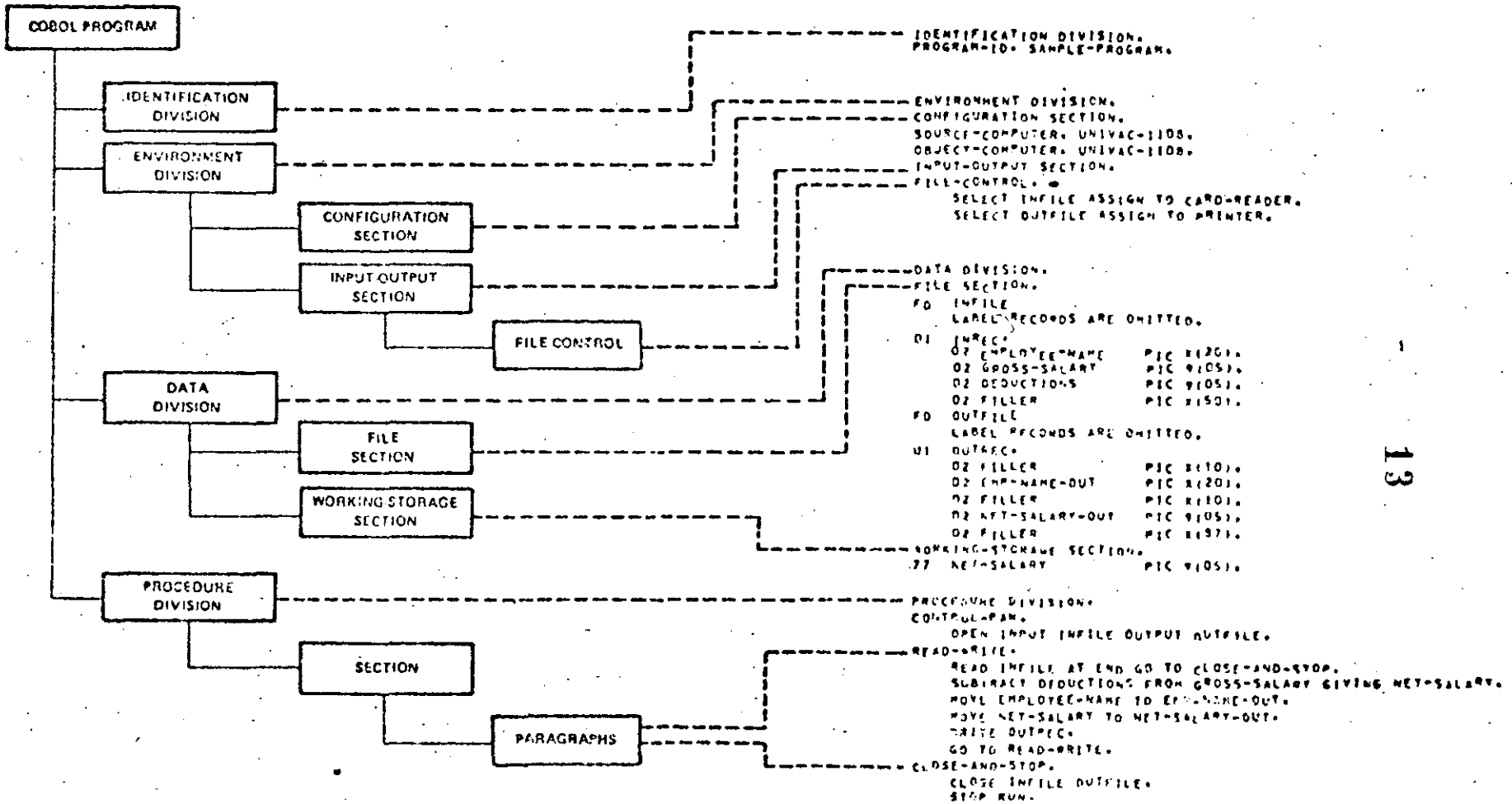
1100 ASCII COBOL SOURCE LISTING

```
 1          IDENTIFICATION DIVISION.
 2          PROGRAM-ID. SAMPLE-PROGRAM.
 3          ENVIRONMENT DIVISION.
 4          CONFIGURATION SECTION.
 5          SOURCE-COMPUTER. UNIVAC-1108.
 6          OBJECT-COMPUTER. UNIVAC-1108.
 7          INPUT-OUTPUT SECTION.
 8          FILE-CONTROL.
 9              SELECT INFILE ASSIGN TO CARD-READER.
10              SELECT OUTFILE ASSIGN TO PRINTER.
11          DATA DIVISION.
12          FILE SECTION.
13          FD  INFILE
14              LABEL RECORDS ARE OMITTED.
15          01  INREC.
16              02 EMPLOYEE-NAME     PIC X(20).
17              02 GROSS-SALARY      PIC 9(05).
18              02 DEDUCTIONS        PIC 9(05).
19              02 FILLER            PIC X(50).
20          FD  OUTFILE
21              LABEL RECORDS ARE OMITTED.
22          01  OUTREC.
23              02 FILLER            PIC X(10).
24              02 EMP-NAME-OUT      PIC X(20).
25              02 FILLER            PIC X(10).
26              02 NET-SALARY-OUT    PIC 9(05).
27              02 FILLER            PIC X(87).
28          WORKING-STORAGE SECTION.
29          77  NET-SALARY           PIC 9(05).
30          PROCEDURE DIVISION.
31          CONTROL-PAR.
32              OPEN INPUT INFILE OUTPUT OUTFILE.
33          READ-WRITE.
34              READ INFILE AT END GO TO CLOSE-AND-STOP.
35              SUBTRACT DEDUCTIONS FROM GROSS-SALARY GIVING NET-SALARY.
36              MOVE EMPLOYEE-NAME TO EMP-NAME-OUT.
37              MOVE NET-SALARY TO NET-SALARY-OUT.
38              WRITE OUTREC.
39              GO TO READ-WRITE.
40          CLOSE-AND-STOP.
41              CLOSE INFILE OUTFILE.
42              STOP RUN.
```

V1-6

Sample COBOL Program Divisions

Four Divisions of a Cobol Program



1. ALPHABETIC CHARACTERS

letters A through Z

2. NUMERIC CHARACTERS

digits 0 through 9

3. SPECIAL CHARACTERS

b	space	,	comma
$	dollar	.	period/decimal
'	quote	/	slash
(	L parenthesis	;	semicolon
)	R parenthesis	<	less than
*	asterisk	=	equal to
+	plus	>	greater than
–	minus/hyphen		

15

Language Elements

THE COBOL LANGUAGE CONSISTS OF THE FOLLOWING SIX LANGUAGE ELEMENTS·

1. RESERVED WORDS
2. PROGRAMMER-SUPPLIED NAMES
3. LITERALS
4. SYMBOLS
5. LEVEL NUMBERS
6. PICTURES

V1-8

- Every COBOL statement must contain at least one reserved word.
- Reserved words have a preassigned meaning to the COBOL compiler.
- They must not be used as programmer-supplied names.
- Examples:
  - READ infile
  - MOVE in-record TO out-record
  - WRITE out-record.
- Read, move, to, and write are reserved words. Syntax rules for read are different than write.
- In COBOL, you READ a file and WRITE a record.

Programmer Supplied Names

- WORDS SUPPLIED BY PROGRAMMER
- USED TO IDENTIFY:
  - DATA (INCLUDING WORK AREAS)
  - PROCEDURE NAMES
  - FILE NAMES
  - RECORD NAMES
  - ELEMENTARY ITEMS

V1-9

1100 ASCII COBOL SOURCE LISTING

```
1         IDENTIFICATION DIVISION.
2         PROGRAM-ID. SAMPLE-PROGRAM.
3         ENVIRONMENT DIVISION.
4         CONFIGURATION SECTION.
5         SOURCE-COMPUTER. UNIVAC-1108.
6         OBJECT-COMPUTER. UNIVAC-1108.
7         INPUT-OUTPUT SECTION.
8         FILE-CONTROL.
9             SELECT INFILE ASSIGN TO CARD-READER.
10            SELECT OUTFILE ASSIGN TO PRINTER.
11        DATA DIVISION.
12        FILE SECTION.
13        FD  INFILE
14            LABEL RECORDS ARE OMITTED.
15        01  INREC.
16            02 EMPLOYEE-NAME      PIC X(20).
17            02 GROSS-SALARY       PIC 9(05).
18            02 DEDUCTIONS         PIC 9(05).
19            02 FILLER             PIC X(50).
20        FD  OUTFILE
21            LABEL RECORDS ARE OMITTED.
22        01  OUTREC.
23            02 FILLER             PIC X(10).
24            02 EMP-NAME-OUT       PIC X(20).
25            02 FILLER             PIC X(10).
26            02 NET-SALARY-OUT     PIC 9(05).
27            02 FILLER             PIC X(87).
28        WORKING-STORAGE SECTION.
29        77  NET-SALARY            PIC 9(05).
30        PROCEDURE DIVISION.
31        CONTROL-PAR.
32            OPEN INPUT INFILE OUTPUT OUTFILE.
33        READ-WRITE.
34            READ INFILE AT END GO TO CLOSE-AND-STOP.
35            SUBTRACT DEDUCTIONS FROM GROSS-SALARY GIVING NET-SALARY.
36            MOVE EMPLOYEE-NAME TO EMP-NAME-OUT.
37            MOVE NET-SALARY TO NET-SALARY-OUT.
38            WRITE OUTREC.
39            GO TO READ-WRITE.
40        CLOSE-AND-STOP.
41            CLOSE INFILE OUTFILE.
42            STOP RUN.
```

DATA NAME

DATA NAME

PROCEDURE NAME

COURSE: Basic ANSI COBOL
CODE: H1-5
TITLE: Programmer-Supplied Name Constraints

- Cannot exceed 30 characters.
- May contain letters (A - Z), digits (0 - 9), and hyphens (–).
- No other symbols or characters may be used.
- Blank spaces within a name are not permitted.
- Reserved words may not be used as programmer-supplied names.
- Data names must contain at least one alphabetic character.
- Procedure names may contain all digits.
- No name may begin or end with a hyphen.

Specify Whether Each Example Is Valid or Invalid

EXAMPLES:

1.	GROSS-PAY	1.	VALIDO ✓
2.	23456	2.	INVALIDO ✓
3.	FILE	3.	INVALIDO ✓
4.	W.W.E.C.	4.	INVALIDO ✓
5.	42-CHARACTER-PAR	5.	VALIDO ✓
6.	-ACCOUNT-PAR	6.	INVALIDO ✓
7.	WEEKLY-PAY-RATE-SCHEDULE	7.	VALIDO ✓
8.	PAY RATE	8.	INVALIDO ✓

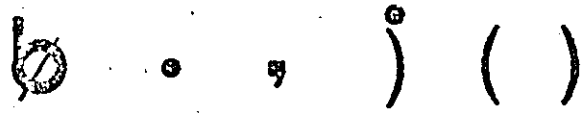
V1-11

COBOL Symbols

ARITHMETIC:

CONDITIONAL:

PUNCTUATION:



V1-12

PUNCTUATION SYMBOLS	PURPOSE	RULES	EXAMPLES
Space	• Delimits words.	• At least one is required between words. • More than one may be used.	GO TO FINAL. GO TO FINAL.
	• Delimits symbolic operators.	• Symbolic operators must be preceded and followed by at least one space.	A + B A < B A = B
Parentheses	• Resolve potential ambiguity.	• No spaces here or here. (.....)	(A + B) Yes (A+B) No
, Comma	• Optional in all cases. • For readability only. • Separates a series of operands or statements.	NO SPACE MAY PRECEDE, BUT ONE MUST FOLLOW	ADD A, B C to D.
; Semicolon	• Optional in all cases. • For readability only. • Separates a series of statements.		MULTIPLY A BY B, ON SIZE ERROR GO TO ERROR-ROUT.
. Period	• Required – After division headings – After section headings – After paragraph headings – At end of data items for ending a paragraph. • Optional – For ending a statement.		DATA DIVISION. FILE SECTION. START-HERE. 02 FILLER PIC X (20).

Examples Of Punctuation Symbols

- Close Sales-File, Account-File
- Close Sales-File,Account-File
- A+B
- A−B
- (A+B)
- (A+B)

V1-13

- A literal is an actual value used in the COBOL program, supplied by the programmer.

- Types of literals:
  - Numeric
  - Non-numeric
  - Figurative constants.

- Examples:
  - Display 'END OF JOB' on printer.
  - Move 0 to Record-Counter.
  - MOVE ZERO to Record-Counter.

25

- May contain 1 to 18 digits.
- A positive (+) or negative (–) and a decimal point (.) may be included with literal.
- If a sign is present, it must appear as the leftmost character with NO space between the sign and the literal.
- If no sign is present, default to positive.
- A decimal point may be used anywhere within the literal except as the right-most character.

Valid And Invalid Numeric Literals

-0.01	V ✓
$30.0	I ✓
32.	I ✓
+55.0	V ✓
12345678901234567	I ✓
33+	I ✓

Indicate Whether Valid or Invalid

V1-14

Non-Numeric Literals

USEFUL FOR:

- DISPLAY MESSAGES ON PRINTER OR CONSOLE.
- PROVIDING MESSAGES OR HEADINGS ON OUTPUT REPORTS.

V1-15

Constraints:

- May contain 1 to 132 characters.
- Any characters from ASCII, FIELDATA, or EBCDIC except quotation marks.
- Must be enclosed in single quotes.
- Two consecutive quotes represent one apostrophe. Count as one character.
- Blank spaces treated as valid characters.
- Quotes not included in size of field.

Valid And Invalid Non-Numeric Literals

Indicate Valid or Invalid

'ABCD' _____ V ✓

'123456' _____ V ✓

'HE CAN"T GO' _____ V ✓

'WON'T WORK' _____ I ✓

'JOB ABORTED' _____ V ✓

'IDENTIFICATION DIVISION' _____ V ✓

'JANUARY 3, 1975' _____ V

NUMERIC			NON-NUMERIC	
Length: 1 – 18 digits			Length: 1 – 132 characters	
Composition: digits 0 – 9 plus + minus – decimal .			Composition: Any characters from the relevant set, ASCII, FIELDATA, or EBCDIC	
RULES	**EXAMPLES**		**RULES**	**EXAMPLES**
1. Decimal Point • There may be only one. • If present may be initial or medial but not final.	1.23 .23 2.3 23.	Yes Yes Yes No	1. Must be enclosed in single quotes.	'ABC' will be output as ABC '12.2' will be output as 12.2
2. Sign • There may be only one. • If none is present, default to positive. • If present, may appear only on the left • No space between sign and literal.	+23. –2.3 23.+ +23.	Yes Yes No No	2. Two consecutive quotes represent one apostrophe.	'HE CAN''T GO' will be output as HE CAN'T GO
3. Length Decimal and/or sign not included in size of field.	+2.3 contains two digits		3. Length • Quotes are not included in size of field. • Two consecutive quotes are counted as one character.	'ABC' is a field of three characters 'CAN'T' is a field of five characters

Figurative constants are reserved words that have a preassigned value. Figurative constants act as literals.

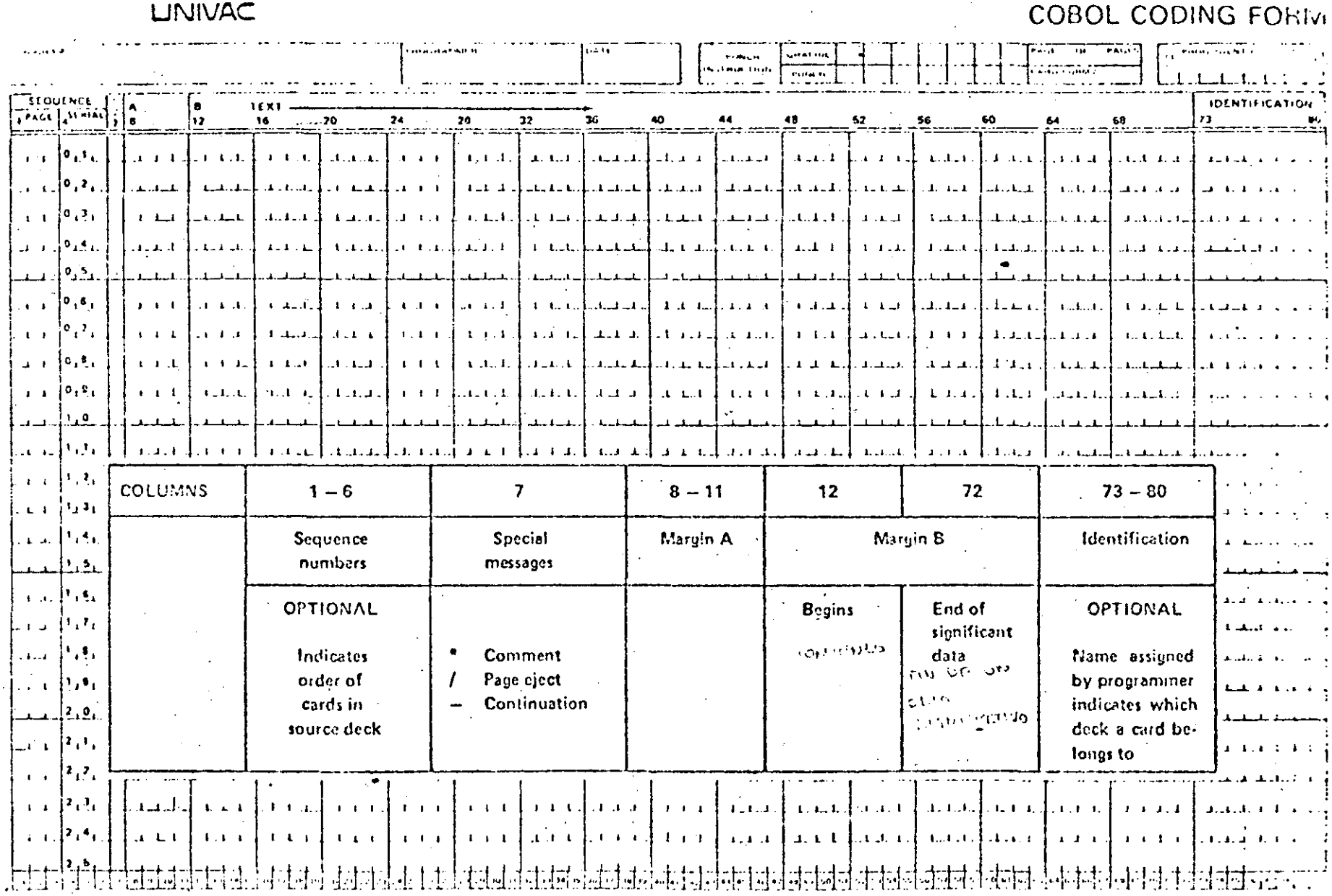
Figurative Constant	Represents
ZERO ZEROS ZEROES	Represents value 0, or one or more of the character 0, depending on context. Fieldata character 0 has octal value 60; ASCII character 0 has octal value 060.
SPACE SPACES	Represents one or more blanks or spaces. Fieldata character blank has octal value 05; ASCII character blank has octal value 040.
HIGH-VALUE HIGH-VALUES	Represents one or more of the Fieldata character with octal value 77; one or more of the ASCII character with octal value 177.
LOW-VALUE LOW-VALUES	Represents one or more of the Fieldata character with octal value 00; or one or more of the ASCII character with octal value 000.
QUOTE QUOTES	Represents one or more of the character. Fieldata character quote has octal value 72; ASCII character quote has octal value 047. Word QUOTE cannot be used in place of a quotation mark in a source program to be bound to a non-numeric literal.
ALL literal	Represents one or more of the string of characters composing the literal. Literal must be either a non-numeric literal or a figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only.
UPPER-BOUND UPPER-BOUNDS	Represents one or more of the characters conventionally used as a high delimiter in processing data. UPPER-BOUND is equivalent to HIGH-VALUE.
LOWER-BOUND LOWER-BOUNDS	Represents one or more of the characters conventionally used as a low delimiter in processing data. LOWER-BOUND is equivalent to LOW-VALUE.

1100 ASCII COBOL SOURCE LISTING

```
1           IDENTIFICATION DIVISION.
2           PROGRAM-ID.  LITERALS.
3           AUTHOR.  GARY ROBEL.
4           REMARKS. THIS PROGRAM GIVES EXAMPLES OF NUMERIC AND NON-NUMERIC
5                    LITERALS. ALSO INCLUDED UNDER LITERALS ARE
6                    FIGURATIVE CONSTANTS.
7           ENVIRONMENT DIVISION.
8           CONFIGURATION SECTION.
9           SOURCE-COMPUTER.    UNIVAC-1108.
10          OBJECT-COMPUTER.    UNIVAC-1108.
11          INPUT-OUTPUT SECTION.
12          FILE-CONTROL.
13              SELECT DUMMY-FILE ASSIGN TO CARD-READER.
14              SELECT OUT-FILE ASSIGN TO PRINTER.
15          DATA DIVISION.
16          FILE SECTION.
17          FD  DUMMY-FILE LABEL RECORDS ARE OMITTED.
18          01  DUMMY-REC                PIC X(80).
19          FD  OUT-FILE   LABEL RECORDS ARE OMITTED.
20          01  OUT-REC                  PIC X(132).
21          WORKING-STORAGE SECTION.
22          01  ASSET-RECORD.
23              02 FILLER                PIC X(20) VALUE ALL '*'.      ----> FIGURATIVE CONSTANT
24              02 FILLER                PIC X(45) VALUE
25                 'THIS IS AN EXAMPLE OF A FIGURATIVE CONSTANT'.      ----> NON-NUMERIC LITERAL
26          01  VALUE-RECORD.
27              02 FILLER                PIC X(20) VALUE ALL '*'.
28              02 GROUP-A               PIC 9(02) VALUE ZERO.
29              02 FILLER                PIC X(05) VALUE SPACES.
30              02 GROUP-B               PIC 9(02) VALUE 10.           ----> NUMERIC LITERAL
31              02 FILLER                PIC X(05) VALUE SPACES.       ----> FIGURATIVE CONSTANTS
32              02 GROUP-C               PIC 9(02) VALUE ZERO.         ----> FIGURATIVE CONSTANTS
33          PROCEDURE DIVISION.
34          ONLY-PAR.
35              OPEN INPUT DUMMY-FILE
36                   OUTPUT OUT-FILE.
37              MOVE 35 TO GROUP-C.
38              WRITE OUT-REC FROM ASSET-RECORD.
39              WRITE OUT-REC FROM VALUE-RECORD.
40              CLOSE DUMMY-FILE, OUT-FILE.
41              STOP RUN.
```

Figurative Constants	Fills a data field with:	For ASCII 36-bit data field in memory would look as follows: (shown in octal)	For EBCDIC 32-bit data field in memory would look as follows: (shown in hexadecimal)
ZERO(S) ZERO(ES)	zeros	in computational mode 000 000 000 000 in character mode 060 060 060 060	00 00 00 00 F0 F0 F0 F0
SPACE(S)	blanks	040 040 040 040	40 40 40 40
QUOTE(S)	quotes	047 047 047 047	7D 7D 7D 7D
HIGH-VALUE(S)	Highest character code in set	177 177 177 177	FF FF FF FF
LOW-VALUE(S)	Lowest character code in set	000 000 000 000	00 00 00 00
All literal	Sequence of any: • Non-numeric literal ALL 'A' ALL 'AB' • Figurative constant ALL SPACES	101 101 101 101 101 102 101 102 040 040 040 040	C1 C1 C1 C1 C1 C2 C1 C2 40 40 40 40

COBOL Coding Form Layout



SEQUENCE NUMBER (1 – 6)

- Written in columns 1 through 6 of the card.
- Use is optional.

CONTINUATION INDICATOR (7)

- A hyphen in Column 7 is used when a numeric or non-numeric literal is broken up and continued on the next line.

PROGRAM STATEMENT (8 – 72)

- Columns 8 through 72 are used for writing the COBOL source statements.
- These columns are grouped into the A-Margin (columns 8 – 11) and B-Margin (columns 12 – 72).

IDENTIFICATION-CODE

- Columns 73 through 80 are reserved for the program identification code.
- This code is assigned by the programmer to identify cards as belonging to a particular program.

COMMENTS

- Comments can be written anywhere in the source program.
- They are identified by writing an asterisk(*) in Column 7.

UNIVAC

COBOL CODING FORM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SAMPLE-CODING.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. UNIVAC-1108.
OBJECT-COMPUTER. UNIVAC-1108.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT OUTFILE ASSIGN TO PRINTER.
DATA DIVISION.
FILE SECTION.
FD  OUTFILE
    LABEL RECORDS ARE OMITTED.
01  OUTREC.
    02 STUFF                    PIC X(12).
    02 FILLER                   PIC X(120).
WORKING-STORAGE SECTION.
77  INDEPENDENT-ITEM            PIC X(12) VALUE 'LOOK AT THIS'.
PROCEDURE DIVISION.
FIRST-PARAGRAPH.
    OPEN OUTPUT OUTFILE.
    MOVE SPACES TO OUTREC, MOVE INDEPENDENT-ITEM TO STUFF.
    WRITE OUTREC.
    CLOSE OUTFILE.
    STOP RUN.
```

What begins in Margin B	Look at lines
Anything listed here must begin at least in column 12 but may begin anywhere thereafter.	
All COBOL Statements	
Beginning with a verb	9, 21, 22, 23, 24, 25
Beginning with a noun	13
Beginning with IF	no example here
In the Data Division	
Filename after FD	12
Record name after 01 — A	14
Identifier after 77 — A	18
All other level numbers:	
02 thru 49 — B	15, 16
66 and 88 — B	no examples here

UNIVAC

COBOL CODING FORM

SEQUENCE (PAGE, SERIAL) | A | B | TEXT | IDENTIFICATION

```
      *  USE OF COLUMN SEVEN *
      *                      *
      *                      *
      *                      *
      *                      *

      * AN ASTERISK IS USED TO:
      *  1. INSERT A COMMENT
      *  2. SKIP A LINE OR TWO
      *  3. MAKE PRETTY PATTERNS

      / A SLASH CAUSES PAGE EJECTION WITHIN THE SOURCE PROGRAM.
      / IF ANYTHING IS WRITTEN DIRECTLY AFTER THE SLASH, IT IS TREATED
      / AS A COMMENT AND WILL APPEAR AT THE TOP OF THE NEW PAGE.

      - A HYPHEN INDICATES CONTINUATION OF AN IMAGE BEYOND COLUMN 72
      a.) REQUIRED FOR A NONNUMERIC LITERAL
          02 HEADER1. PIC. X(64) VALUE 'THIS IS AN EXAMPLE OF WHEN AND
          - 'HOW TO USE THE HYPHEN IN COLUMN 7'.
          02 HEADER2. PIC. X(68) VALUE 'HERE IS ANOTHER EXAMPLE SHOWING T
          - 'HAT THERE MUST BE ANOTHER QUOTE ON THE NEW LINE'.
      b). OPTIONAL FOR WORDS OF CODING
          MOVE SPACES TO OUTREC. MOVE EMPLOYEE-NAME TO EMPLOYEE-OUT. SUBT
          - RACT DEDUCTIONS FROM NET-SALARY.
      *   * THE ABOVE LINE COULD HAVE SIMPLY BEEN WRITTEN AS FOLLOWS:
          MOVE SPACES TO OUTREC. MOVE EMPLOYEE-NAME TO EMPLOYEE-OUT.
          SUBTRACT DEDUCTIONS FROM NET-SALARY.
```

QUIZ I

Instructions:

Mark "T" for true
Mark "F" for false

F ✓ 1. '14' is a numeric literal.

F ✓ 2. A data name must contain imbedded spaces.

T ✓ 3. A numeric literal can be from 1 to 18 digits.

F ✓ 4. All the divisions except the data division contain paragraphs.

F ✓ 5. A program written in COBOL is known as the Object Program.

T ✓ 6. A hyphen cannot be the first character of a programmer-supplied name.

T ✓ 7. Every COBOL statement must contain at least one reserved word.

F ✓ 8. Procedure names cannot begin with a number.

F ✓ 9. Procedure names must contain at least one alphabetic character.

T ✓ 10. Zeros, spaces, high-values, are COBOL words call figurative constants.

T ✓ 11. A period, comma, or semicolon must not be preceded by a space, but must be followed by a space.

F ✓ 12. Blank spaces are permitted within a name.

F ✓ 13. File description is in the input-output section.

T ✓ 14. The environment division describes I/O devices which will be used by the program.

F ✓ 15. A statement is comprised of many COBOL sentences.

QUIZ 1 (Cont'd)

F ✓ 16. Division header can begin at the B-Margin.

T ✓ 17. Figurative constants are reserved words.

F ✓ 18. Conditional symbols are + – * / **.

F ✓ 19. Blank spaces are valid characters in numeric literals.

T ✓ 20. A hyphen (–) in column 7 indicates continuation onto the next line.

CAPITULO 1

REPASO DE INSTRUCCIONES BASICAS

INTRODUCCION
INSTRUCCIONES
MOVE
IR
GO TO
OPEN
WRITE
READ
CLOSE
ADD
SUB
MULTIPLY
DIVIDE

° INTRODUCCION

Se hace necesario que para poder avanzar en la comprensión del lenguaje COBOL, se haya comprendido bien el uso de las instrucciones básicas. Por lo que en este capítulo se presentan estas en su forma básica.

MOVE

Este verbo causa que el contenido de un campo sea copiado a otro campo borrando el contenido del segundo.

El formato general de esta instrucción es:

```
          IDENTIFICADOR-1
MOVE                        TØ IDENTIFICADOR-2  IDENTIFICADOR-3 ..
          LITERAL
```

Ejemplos

```
MOVE CANTIDAD TØ SUMA
MØVE "PEDRO" TØ NØMBRE
MOVE 25 TØ PORCENTAJE
MOVE ZERO TØ TØTAL
MØVE SUELDO TØ TOTAL-DPTO, SUELDO-EMP
```

Esta es la instrucción en su formato sencillo, pero aquí lo importante es saber como va a ser el contenido del campo en que se copia la información. Esto es porque el campo que recibe la información puede ser mayor o menor o igual en longitud y esto hace que el movimiento se efectúe en diferentes formas.

Movimiento numérico

```
                                MOVE CANTIDAD TØ AUXILIAR.

                                AUXILIAR
Si tenemos                       0 3 2 5
Cantidad
 3 2 5                             2 5
                                  3 2 5
```

Aquí se indica que el campo se ajusta a la derecha.

Movimiento Alfanumérico o Alfabético.

```
                                  M A R T I N E Z
M A R T I N E Z                   M A E T E
                                  M A R T I N E Z
```

Aquí se indica que el campo se ajusta a la izquierda

IF

Este verbo hace que se efectúe una pregunta. Y estas se conocen como expresiones condicionales y tienen el siguiente forma

to.

```
                Instrucción-1                  Instrucción-2
IF Condición    hext Sentence  ; ELSE          hext Sentence
```

La condición se clasifica de varias maneras, éstas se explican en el capítulo 5. Si la condición es cierta se ejecuta la -- instrucción-1, si no se cumple se ejecuta la instrucción-2. La clausula NEXT SENTENCE significa que continúe con la si--- guiente proposición.

Ejemplos:

```
1.-                 IF CANTIDAD IS GRATER THAN MINIMØ
      *                CALCULA IMPUESTØ
                    ELSE
                       NEXT SENTENCE.

2.-                 IF CØMPRA IS LESS THAN SIN-DESCUENTØ
                       NEXT SENTENCE
                    ELSE
      *             CALCULA DESCUENTØ.
```

Otra forma de representarlo sería

```
                    IF CØMPRA IS GRATER THAN SIN-DESCUENTØ
                       CALCULA DESCUENTØ.
```

GØ TØ

Esta instrucción transfiere el control al párrafo que indique el GØ TØ, su formato es el siguiente.

```
GØ TØ NØMBRE-DE-PRØCEDIMIENTØ.
```

Ejemplos:

```
LEE-REGISTRØ.

    IF REGISTRØ TS EQUAL TØ DESEADO
       GØ TØ PRØCESARLØ.

    ELSE
       SØ TØ LEE-REGISTRØ.
```

OPEN.

Este verbo hace que un archivo este disponible para utilizarlo, no se puede utilizar un archivo antes de abrirlo, ya que al abrirlo le damos la características de uso, esto es, le indicamos si vamos a leer datos de él o si vamos a dejar datos en él. Existen mas características las cuales se discutirán en el capítulo 5. Su formato es el siguiente:

```
        INPUT
OPEN            NOMBRE-DEL-ARCHIVO
        ØUTPUT
```

Ejemplos:

Si tenemos

ØPEN INPUT DATENT (Datos de entrada)

Estamos indicando implícitamente que este archivo se utilizará para leer.

Si tenemos

ØPEN ØUTPUT REPORTE

Estamos indicando implícitamente que en este archivo va a ser utilizado para escribir (un listado).

READ.

Esta instrucción nos envia un registro de archivo que se espe-

cifique (no importando que este archivo esté es un disco, tarjeta o cinta) a nuestro programa, su formato es el siguiente:

READ nombre-del-archivo AT END instrucción-imperativa

Ejemplos:

```
READ DATEND AT END.
    MØUE "SI" TØ FIN-DE-ARCHIVO.

READ DATEND AT END.
    GØ TØ FIN-PRØGRAMA.
```

WRITE.

Esta instrucción efectúa lo contrario del READ, esto es, enviamos datos de nuestro programa para que sean puestos en un archivo. Su formato es el siguiente:

```
WRITE  Nombre-del-registro FRØM identificador-1
       Before
       After  Advancing identificador-2  o lines
                        entero
```

Esta instrucción es extensa debido a que con ella se pueden efectuar diferentes operaciones, ya sea, que se mande grabar un registro a un archivo que esté en cinta o en disco o se mande escribir un registro (línea) en un reporte.

Ejemplos:

```
WRITE REG-DISCØ.
```

Aquí le indicamos grabar un registro en disco (la palabra disco no quiere decir que éste sea exactamente un disco, ya que esto lo controla el programador).

```
WRITE REG-CINTA.
```

Aquí le indicamos grabar un registro en cinta (la palabra cinta no quiere decir que ésta sea exactamente una cinta, ya que esto lo controla el programador).

WRITE REG-DISCO FRØM REG-CALCULØ

Primero se transfiere la información que está en el campo -- REG-CALCULO al campo REG-DISCO y luego se graba.

WRITE REG-SAL FRØM TITULO-1BEFØRE-ADVANCING

2 LINES

En esta instrucción se manda el contenido del campo TITULO-1 al campo REG-SAL y se escribe, avanzando después dos líneas.

CLOSE.

Esta instrucción sirve para cerrar un archivo; esto es, que ya no se va a usar a menos de que se vuelva a abrir con un -- OPEN. Su formato es:

CLOSE Nombre-de-archivo-1, nombre-de-archivo-2

Ejemplo:

CLOSE DATEND, REPØRTE.

Las instrucciones aritméticas tienen algunas cláusulas en co-- mún, éstas las explicaremos a continuación ya que en los ejemplos no haremos mención de esto.

GIVING Identificador.

Esta cláusula hace que el resultado de la operación sea puesto en el identificador.

Ejemplo:

GIVING RESULTADO.

Esto significa que lo que se obtuvo en la operación sea puesto en el campo llamado RESULTADO.

RØUNDED.

Esta cláusula es usada frecuentemente para redondear resultados donde se opera con cantidades que significan pesos y centavos,-

ya que el resultado de una operación puede dar fracciones que no sean cantidades exactas por ejemplo:
un resultado,podemos tener 30 pesos 47 centavos en este caso, lo mejor sería 30 pesos 50 centavos. Lo que hace esta clausula es sumarle un 5 al resultado y despues quitar un dígito,esto lo haremos con el ejemplo anterior.

Si tenemos.47 centavos + .05= .52 y se quita el 2 queda 50 centavos

Nota: el 5 que se suma depende del campo que se va a redondear

Campo	Ajuste
.0	.5
.00	.005
.000	.005

ØN SIZE ERRØR

Esta cláusula es usada, cuando para detectar posibles errores en cuanto a que se calcule que una cantidad que excede un número de dígitos que teníamos previsto.

Ejemplo:

Si pensamos que el pago a un empleado no excede los $9 999.99 semanales y se llega a dar el caso de que el empleado los rebase,nosotros pondríamos esto de la siguiente forma.

CALCULO ON SIZE ERRØR. Envía Mensaje.

INSTRUCCIONES ARITMETICAS.

ADD.

Esta instrucción causa que dos o más operandos numéricos sean sumados y el resultado sea almacenado, su formato es el siguiente:

```
        Identificador-1   , identificador-1
ADD                                        ... TØ identificador-n
        Literal-1         , literal-2
GIVING identificador-x ROUNDED ... ØN SIZE ERROR instrucción ino-
                                                          perativa.
```

Ejemplos:

ADD CAMBIO, CØMPRA TØ PAGØ

Esto causa que el contenido de cambio y compra se sumen con el valor de pago

ADD CANTIDAD TØ TØTAL GIVING RESULTADO RØUNDED

Esto causa que se sumen los contenidos de cantidad y total y se dejen en el campo resultado y éste sea redondeado.

SUBTRACT.

Esta instrucción es usada para restar un campo o la suma de varios a un campo o a varios, su formato es el siguiente:

```
                Literal-1          Literal-2
SUBTRACT                                        ...FRØM
                Identificador-1    identificador
```

CONCEPTOS IMPORTANTES PARA LA DESCRIPCION DE DATOS.

* INTRODUCCION
* ARCHIVOS
* REGISTROS
* CAMPOS
* CLAUSULAS PARA LA DESCRIPCION DE DATOS
  * NIVELES 66 y 88
  . USAGE
  . REDEFINES
  . JUSTIFIED
  TABLAS (DE 1 y 2 DIMENSIONES)
  DE VALORES CONSTANTES
  VARIABLES.

INTRODUCCION.

Difícilmente en los libros que enseñan Cobol o en los manuales de cobol de los equipos de cómputo, se da una explicación del por que los datos se deben definir de cierta forma (esta se verá en este capítulo) ya que nada mas se concretan a decir con que cláusulas se definen y/o describen. Antes de describir las cláusulas relacionadas con las descripciones de datos, enfocamos esto a definir los conceptos fundamentales para el manejo de los datos.

ARCHIVOS.

Los archivos son un compuesto de datos relacionados. Estos están organizados de tal manera que permitan a la persona que los use de una manera fácil y ágil de manejo.

Si tomamos un ejemplo de un archivo que manejemos en nuestro trabajo como: el archivo de personal, el archivo de proyectos, el archivo de obras etc etc... lo que tenemos, no es más que un conjunto de datos relacionados, y así son los archivos que se ma

nejan desde el punto de visto de cómputo.

REGISTROS.

Los registros es el conjunto de datos que se toma como unidad de un archivo esto es; un archivo siempre contiene un número definido de registros, estos registros debido a sus caracte-- rísticas físicas se pueden clasificar en dos, que son: Regis tros de longitud fija y registros de longitud variable, esto lo ejemplificaremos.

Si tenemos un archivo de personal en el cual se tienen las ca racterísticas de cada uno de los empleados y hay 300 empleados en la empresa diríamos que nuestro archivo de personal consta de 300 empleados,y si tomamos a cada empleado como un registro entonces diríamos que el archivo de personal consta de 300 re- gistros. Si ahondamos un poco más sobre las características - de cada registro podemos tener lo siguiente:

Pensando que por cada registro (empleado) se tienen las si--- guientes características, como son:

La dirección de su casa, el nombre, el sueldo etc.... podría- mos tener un"Registro de Longitud FISA". Pero si vamos más - lejos y pensamos que en cada registro (empleado) tenemos las características de cada hijo como edad, nombre, sexo, etc. -- etc. Habría empleados que no tengan hijos y otros que tengan varios; esto crearía la necesidad, que en cada registro ten-- dría que poner las características de cada hijo y si no tiene, dejar el espacio para que todos los registros mantengan unifor midad. Aquí lo conveniente es poner las características de ca da uno de los hijos si es que existen, en caso contrario, no - poner nada. Esto se podría definir como un "Registro de longi tud Variable".

CAMPOS.

Los campos son las unidades en que se dividen los registros y estos encierran en sí mismos un valor dado para una caracterís

tica.

Si tomamos el archivo de personal que anteriormente describimos tenemos que un registro es el siguiente:

NOMBRE	SUELDO	DPTØ	PUESTO	FECHA DE INGRESO

REGISTRO DE PERSONAL

Para este registro tenemos 5 campos en el que para cada uno tenemos un valor.

. CLAUSULAS PARA DESCRIPCION DE DATØS.

Las cláusulas que nos permiten declarar un archivo y sus características, son las siguientes:

En la ENVIRØNMENT DIVISION.

SELECT OPTIONAL NOMBRE-DE-ARCHIVO

ASSIGN TØ Nombre-de-archivo-externo

Esta cláusula hace que un archivo se ligue con el programa.

La siguiente cláusula es descrita en la Data Division

FD nombre-de-archivo LABEL RECORD IS OMITTED
DATA RECORD IS ó ARE nombre de datos-1...
BLOCK CONTAINS número-entero-1 to número-entero-2

Bajo esta cláusula se describen las características físicas y lógicas de un archivo.

Físicas El tamaño del block.
El tipo de Etiqueta.

Lógicas El o los nombres de los registros.
Identificación de cada campo del registro.
Características de tipo y longitud de cada campo.

Ejemplo:

Suponiendo que tenemos un archivo con datos de refacciones de maquinaria y lo queremos describir.

Esto lo haremos de la siguiente forma:

```
FD ARCHMAQ LABEL RECORD IS ØMITTED
   BLOCK CONTAINS 10 RECORD
   DATA RECORD IS REG-MAQ.

01 REG-MAQ.
   03 NO-DE-PIEZA PIC 9 (6).
   03 TIPO-DE-REFAC PIC 9.
   03 PRECIO PIC 9(4)V99.
   03 DESCRIPCION-REF PIC X(35).
```

52

En este ejemplo se describe primero el archivo, luego el nombre del registro en la cláusula DATA RECORD y por último los campos que conforman el registro.

Al definir cada campo del Archivo y/o cada campo que va a ser utilizado en el programa siempre usamos un número de nivel. Hay números de nivel que definen características especiales éstos los describimos a continuación.

NUMERO DE NIVEL 66

Este número se usa en conjunción con la cláusula RENAMES y sirve para reagrupar campos. Su formato es el siguiente:

```
66 nombre-de-datos-1 Renames nombre-de-datos-2
(THRU nombre-de-datos-3)
```

Ejemplo:

```
01 REG-MAQ.
   03 NØ-DE-PIEZA PIC9(6)
   03 TIPO-DE-REFAC PIC9.
   66 CLAVE-REFAC RENAMES NØ-DE-PIEZA THRU TIPO-
   DE-REFAC.
   03 PESO          PIC 999V99.
   03 PRECIO        PIC9999V99.
   03 DESCRIP-REF PIC X(35).
   03 IMPOR-NAC    PIC 9.
```

NUMERO DE NIVEL 88

Los campos en que se usa este número de nivel son conocidos como Nombres-Condicionales. Este número de nivel nos permite definir valores en un campo, su formato es:

88 nombre-de-datos { VALUE IS / VALUE ARE } literal-1 [THRU literal-2] literal-3 [THRU literal-4]...

Este nombre-de-datos se usa en conjunción con el verbo IF Ejemplo:

```
01 Datos-de-Personal.
   03 Nombre  PIC X(30).
   03 Dirección PIC X(35).
   03 Edad      PIC 99.
      88 Infante values are 1  THRU 11
      88 Adolescente values are 12 THRU15
      88 Joven values are  16  THRU 20
      88 Mayor values are  21  THRU 99.
```

Suponiendo que tenemos un registro de pasajeros que abordan un avión, y tenemos que presentar un reporte de las edades de las personas que visitan determinado lugar del país. Tendríamos que hacer lo siguiente:

```
IF infante
     Suma 1 a Cont-infantes
ELSE
     IF Adolescente
     Suma 1 a Cont-Adolescentes
ELSE
```

```
ELSE
     IF Mayor
     Suma 1 to Cont-Mayores.
```

Esto es más claro que si preguntaramos así:

```
IF Edad<11
   Suma 1 a Cont-Infantes
ELSE
   IF Edad>11 AND Edad<16
   Suma 1 a Cont-Jovenes.
      .
      .
      .
```

CLAUSULA REDEFINES

Teniendo descrito un registro con 10 campos, nosotros podemos volverlo a describir de otra forma y utilizar más de 10 campos, o menos, según nos convénga esto lo hacemos utili-- zando la cláusula REDEFINES.

Ejemplo:

Si tenemos un registro y lo necesitamos de las siguientes formas.

NUMERO DE CLIENTE	NØMBRE-DEL-CLIENTE	CANTIDAD

No. DE VENDEDOR	NOMBRE DEL VENDEDOR	ZONA PORCEN:	CANTIDAD

Una manera más ágil de definir lo anterior sería:

```
01 TABLA-ACCIDENTES.
   03 MES-AC PIC 9(5) ØCCURS 12 TIMES.
```

La cláusula Occurs se usa su conjunción con la palabra TIMES y significan el número de veces que ocurre un campo.

Para poder hacer referencia a un determinado campo que tenga una cláusula Occurs, es necesario decir exactamente a quién. Esto se hace por medio del uso de índices por ejemplo:

Si vamos a sumar 10 accidentes en marzo (para el ejemplo anterior) y sabemos que marzo es el tercer mes lo hacemos así:

```
ADD 10 TØ MES-AC(3)
```

A la variable MES-AC(3) se le conoce como variable suscrita.

CLAUSULA USAGE

Los datos numéricos en la computadora pueden guardarse de dos maneras que son como un dato de caracteres o como un dato numérico, aunque ambos modos utilizan bits su significado es diferente.

Un DATO DE CARACTERES se representa por un valor dado a cada caracter. Cuando se define el conjunto de letras, números y símbolos especiales se puede usar la computadora, a cada uno de éstos se le conoce como caracter y se le asigna un valor por ejemplo:

CARACTER	VALOR
A	1
B	2
C	3

56

2	27
#	28
1	29
*	30
+	31
3	32
.	
.	
.	
1	50
2	53
3	54
.	
.	
.	

Para representar el número 324

El 3=54
2=53 =324=545355
4=55 Se pasa a binario.

UN DATO NUMERICO

Es aquel en que el número es convertido directamente a binario, de aquí podemos decir, que el valor en binario es igual al valor del número, a diferencia de un número representado no un dato de CARACTERES.

En un programa todos los datos numéricos son representados como un dato de caracteres, y cuando se hace un cálculo se tiene que llevar a cabo una conversión, de lo que no nos damos cuenta, esta conversión hace que los cálculos numéricos se lleven más tiempo.

la claúsula USAGE define la representación interna de un campo , ésta puede ser como un DATO NUMERICO (COMPUTATIONAL) o como un dato de ca-

racter (DISPLAY), se usa de la siguiente forma:

```
03 CANTIDAD  PIC 9(5) USAGE IS COMPUTATIONAL
03 ACUMULADO PIC 9(3) USAGE IS COMP
03 RESULTADO PIC 9(6) USAGE IS DISPLAY.
```

TABLAS DE 1 y 2 DIMENSIONES.

Una tabla es una serie de campos con características iguales, descritos conjuntamente de una manera breve y utilizando la cláusula OCCURS.

Una comparación con una realidad, es decir que una tabla es igual a un mueble con cajones iguales en que en los cajones o se tiene algo ya -- guardado o se tiene vacío y se va a llenar.

Una tabla de una dimensión es aquella que tiene únicamente una ocurrencia (es un mueble con un número x de cajones). Una tabla de dos dimensiones es aquella que tiene dos ocurrencias (Es un mueble que tiene cajones con cajoncitos adentro).

Ejemplos TABLAS DE 1 DIMENSIÓN.

```
01 TABLA-DIAS.
   03 FILLER PIC X(9) VALUE "LUNES"
   03 FILLER PIC X(9) VALUE "MARTES"
   03 FILLER PIC X(9) VALUE "MIERCOLES"
   03 FILLER PIC X(9) VALUE "JUEVES"
   03 FILLER PIC X(9) VALUE "VIERNES"
   03 FILLER PIC X(9) VALUE "SABADO"
   03 FILLER PIC X(9) VALUE "DOMINGO"

01 DIAS-SEM REDEFINES TABLA-DIAS.
   03 NOMBRE-DIA PIC X(9) OCCURS 7 TIMES.
```

```
01 TAB-ESTADOS.
   03 RES-ESTADO PIC 9(5) ØCCURS 32 TIMES.
```

TABLAS DE DOS DIMENSIONES.

```
01 TABLA-DE-PARTES.
   03 TIPØS-DE-ARTICULOS ØCCURS 20 TIMES
      05 NØMBRE-DEL-ARTIC PIC X(20) ØCCURS 250 TIMES.

01 TABLA-DESCUENTOS.
   03 FILLER PIC X(20) VALUE "0020 JU06 HE12 HE18 CØ15".
   03 FILLER PIC X(20) VALUE "0025 JU12 HE14 DI18 CØ19".
   03 FILLER PIC X(20) VALUE "0035 JU00 HE08 CØ16 HØ04".

01 TAB-DES REDEFINES TABLA-DESCUENTOS.
   03 ARTICULOS-CON-DESC ØCCURS 3 TIMES.
      05 CLAVE-ART PIC 9(4) ØCCURS  5 TIMES.
```

3.- MANEJO DE ARCHIVOS

60

3.- MANEJO DE ARCHIVOS.

INTRODUCCION.

Todos los equipos de cómputo, tienen una manera de organizar la información en sus archivos, haremos la descripción de las más comunes, éstas se encuentran en casi todos los equipos ya que no son exclusivos de alguna marca de computadora en especial. Los documentos que describe las organizaciones de archivos, casi siempre son libros y algunas revistas, ya que las casas que venden computadores no la proporcionan, porque como decíamos no es exclusiva de alguna en especial.

TIPOS DE ACCESO.

El acceso en el manejo de archivos es de dos tipos, que son secuencial y directo (en los libros o manuales cuando se refieren al acceso directo lo llaman generalmente RANDOMO - - DIRECT).

Los tipos de acceso se refieren a la manera en que vamos -- a obtener de un archivo, esto es la manera de leer un archivo.

ACCESO SECUENCIAL (ACCESS SEQUENTIAL).

Decimos que tenemos en un archivo acceso secuencial, cuando los datos los leemos en la secuencia en que se encuentran - grabados, ésto quiere decir que para obtener un dato tenemos que leer los que están antes de él.

- 61

ACCESO DIRECTO. (ACCESS RANDOM)

Decimos que tenemos un archivo acceso directo, cuando podemos obtener en una sola lectura a un dato que necesitemos, sin haber leído antes a todos lo que se encuentren colocados antes de él.

TIPOS DE ORGANIZACION.

Dentro de la gram mayoría de equipos de cómputo existen organizaciones de archivos comunes como lo son las siguientes

SECUENCIAL.
RELATIVA.
SECUENCIAL INDEXADA

La organización de un archivo se diseña tomando como base la forma en que se quiera recuperar los datos, estas organizaciones mencionadas, tienen características muy especiales y diferentes entre ellas.

ORGANIZACION SECUENCIAL.

Cuando un archivo es organizado secuencialmente sus registros están uno despues de otro, este orden va de acuerdo a la forma en que se escribieron al momento de generar el archivo.

Este tipo de organización se usa cuando no es importante el orden en que se va a leer el archivo.

62

Estos archivos pueden residir en un cinta magnética o en -- mass storaje (DISCO).

ORGANIZACION RELATIVA.

Cuando se define un archivo con organización RELATIVE (Relaliva) los registros estan en una posición relativa a un valor, este valor debe ser un número, el primer registro debe tener un valor de acuerdo al lugar en donde va a residir, - ésto es, si va a estar en el primer lugar el valor numérico - de la posición relativa debe ser uno, si va a estar en el - décimo lugar el valor numérico de la posición relativa debe ser diez.

El valor numérico de la posición relativa es comúnmente conocido como valor de la llave, en lo subsecuente nos referimos a este valor número de la posición relativa como valor de la llave.

Esta llave se define al momento de describir el archivo.

Esta organización es recomendable cuando los valores de las llaves son contiguos, es decir que no haya mucha dispersión entre ellos, un Ejemplo sería organizar un archivo de em -- pleados donde se tiene como llave el número de empleado y - éste es de uno en uno.

Si tenemos un archivo con organización Relative y los valores de las llaves son 10, 20, 50, 80, etc.

La información estará muy dispersa ya que ocupan la posición 10,20,50,80 etc., y se dejarían muchos huecos en el archivo.

Estos archivos para su proceso siempre deben residir en Disco.

ORGANIZACION SECUENCIAL INDEXADA. (INDEXED SEQUENTIAL).

Cuando se define un archivo con organización secuencial indexada, los registro se van almacenando de acuerdo al valor de la llave, todos los registros estarán en orden creciente al valor de la llave; los equipos de cómputo tienen las rutinas necesarias para ir ordenando los registro y para todo lo referente al manejo de estos archivos.

En una organización secuencial indexada, al tiempo de estar generando el archivo, se van almacenando dos tipos de datos, los datos referentes a cada registro, y el valor de la llave que tiene, cada registro en esta organización se le conoce como índice, ya que éste nos indica en dondde se encuentra cada registro.

Estos archivos deben residir en disco para su manejo.

CLAUSULAS PARA LA ESPECIFICACION DE ARCHIVOS.

Para poder manejar un archivo en un programa escrito en COBOL, es necesario especificar todo lo que lo relaciona con el exterior, como lo es el lugar donde reside, el tamaño de

cada registro etc..., y lo que lo relaciona con el manejo que vamos a hacer de él. A continuación describiremos las cláusulas más usuales y su significado.

*** ENVIRONMENT DIVISION. ***

SELECT Nombre-de-archivo ASSIGN TØ nombre-implementado

RESERVE _____ área / áreas

ORGANIZATION IS RELATIVE / INDEXED

ACCESS MODE IS SEQUENTIAL / RANDOM

RECORDO KEY IS _____

En esta cláusula establecemos la relacion del archivo que utilizaremos, ya que el nombre-de-archivo lo utilizamos en el programa y el nombre-implementado, sólo para la relación con las tarjetas de control que dicen qué archivo manejamos y donde ésta.

RESERVE _____ Areas.

Con esta cláusula establecemos el número de áreas que vamos a necesitar.

65

ORGANIZATION IS.

Con esta cláusula establecemos, la organización que se haya decidido para el archivo.

ACCESS MODE

Con esta cláusula establecemos la manera en que vamos a tener acceso al archivo.

RECORD KEY.

En esta cláusula establecemos el campo que nos va a servir como llave.

*** DATA DIVISION ***

FD nombre-de-archivo.

Block Contains _____ Record
Record Contains _____ Characters.
Label Record is _____ Omitted

Este párrafo es comúnmente conocido como "FD" y significa - Descripción del Archivo (File), aquí especificamos lo siguiente; El número de registros que contendrá cada block si es - que bloqueamos los datos. El número de caracteres que contiene cada registro. El tipo de etiqueta del archivo, si - es que va a tener.

66

4.- INSTRUCCIONES.

En este capítulo veremos el uso de las instrucciones más comunes en Cobol aunque nos apoyaremos en un programa de sintaxis para analizar los formatos que debemos respetar.

INSTRUCCIONES DE MANEJO DE ARCHIVOS.

READ

Este verbo nos sirve para leer datos de un archivo, sin importar en donde resida éste. Puede ser un archivo en tarjetas, en disco o en una cinta, su formato es el siguiente -- "READ nombre-de-archivo Invalidkey". El uso de la cláusula Invalidkey depende del modo de acceso que le hagamos al archivo. Esta es usada para cuando se tiene declarado un modo de acceso directo (ACCESS MODE IS RANDOM) pero si el registro a localizar no existe la instrucción ejecuta lo que esté dentro del Invalidkey.

WRITE

Este verbo nos sirve para escribir datos es un archivo, sinimportar en donde resida éste. Puede ser un archivo en disco, en una cinta o en un listado, su formato es el siguiente "WRITE nombre-del-registro FROM___invalid Key".

El uso de la cláusula Invalidkey depende del modo de accesoque se le de al archivo. Esta es usada para cuando se tiene declarada una organización en la que se tenga un modo de -- acceso directo. Esta cláusula se activa cuando se quiere -- grabar un registro y ya existe en el archivo, o cuando los -

67

va a clasificar (SD), declarar los campos que van a ser la base para clasificar, o sea decir en que orden se va a poner la información, a éstos se le conocen como llaves de clasificación, y decir qué archivo se va a utilizar y en que archivo se va a dejar la información ya clasificada..

En el momento de efectuar una clasificación (SORT) se puede procesar los registros antes de ser clasificados o después de ser clasificados.

INTRODUCCION A LA COMPUTACION Y PROGRAMACION ELECTRONICA

BANCO DE DATOS

- ANALISIS COMPARATIVO DE LENGUAJE
- ORGANIZACION DE ARCHIVOS
- BASES DE DATOS

ING. BENITO ZYCHLINSKI

OCTUBRE, 1984

ORGANIZACION DE ARCHIVOS

ING. BENITO ZYCHLINSKI

TIPOS DE ARCHIVOS

I. Maestro

Datos históricos; estos archivos son actualizados periódicamente y usados a lo largo de todo el proyecto.

II. Transacciones

Altas, bajas o cambios a los archivos maestros.

III. Trabajo

Archivos temporales utilizados para almacenar resultados intermedios, tales como Sort, Merge, etc.

IV. Catálogo

En cada volumen de disco existe un directorio de archivos en donde se detalla la localización, el tipo , la fecha de creación, etc.

V. Programa

Los archivos objeto, los cuales se encuentran en módulo ejecutable.

VI. Fuente

Archivos en forma de lenguaje de computación, tales como Cobol, Basic, Fortran, etc.

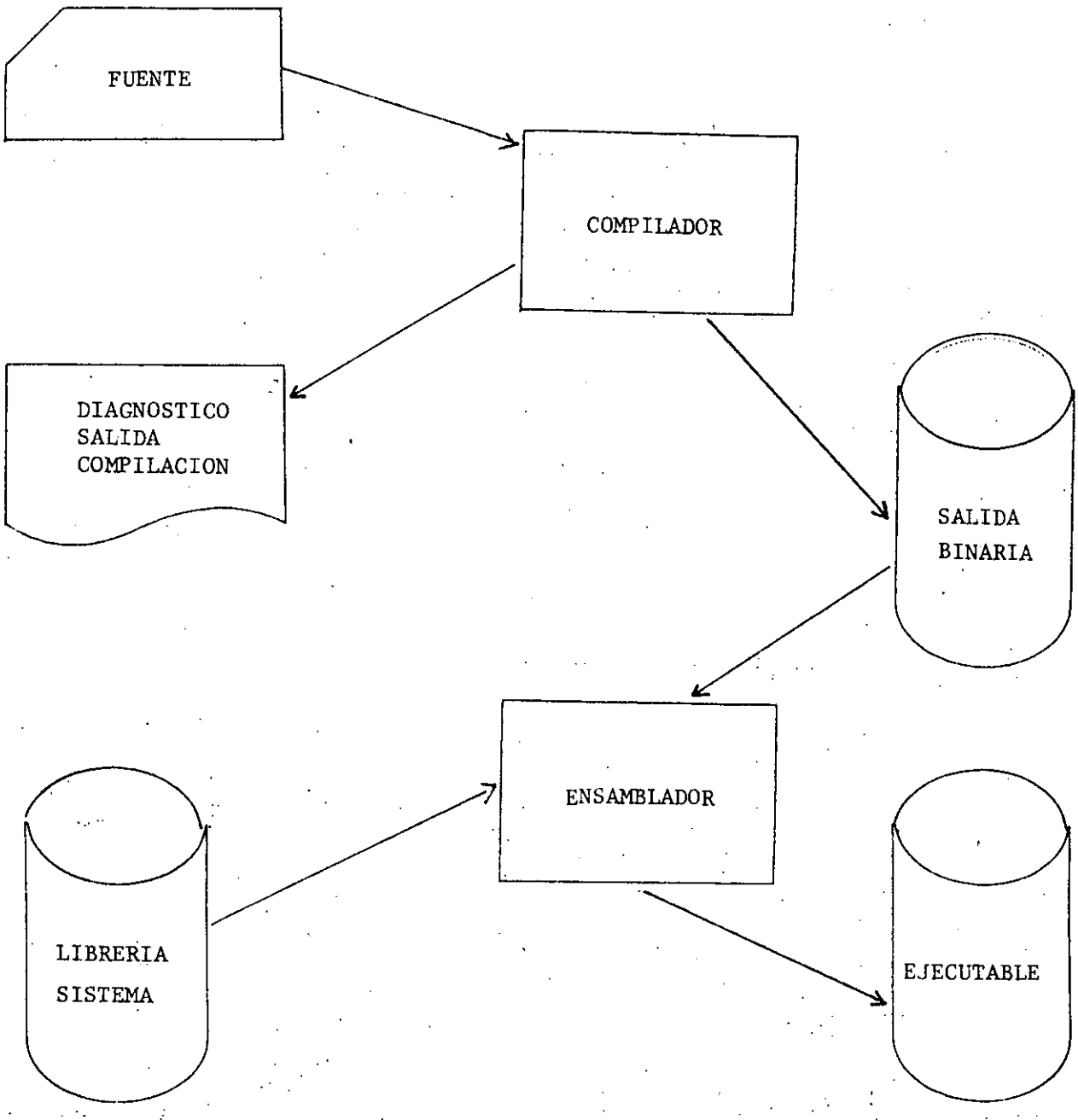
VII. Librería

Subrutinas y programas de utilería, usualmente proporcionados por el fabricante del equipo.

VIII. Spool

Archivos de entrada/salida para periféricos lentos.

Ejemplo:

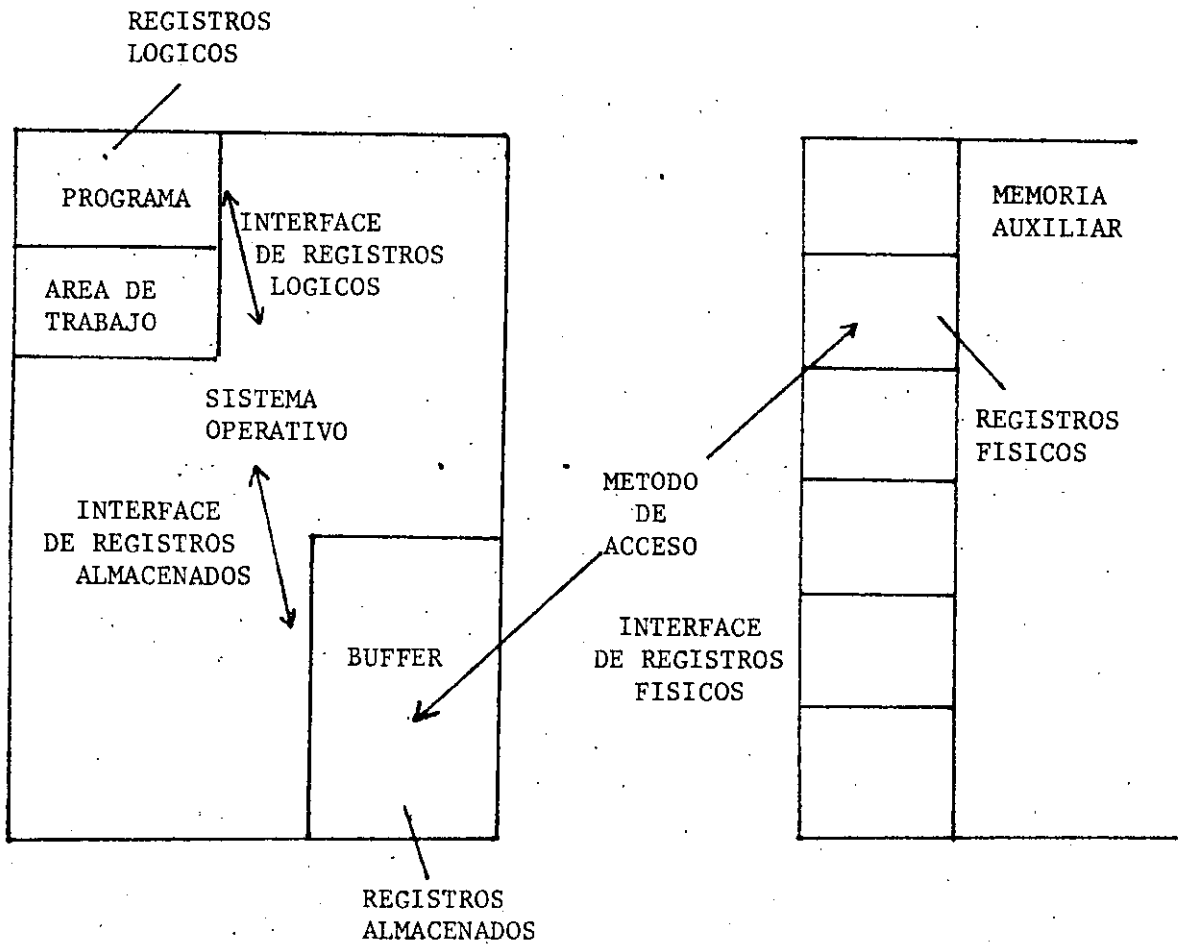


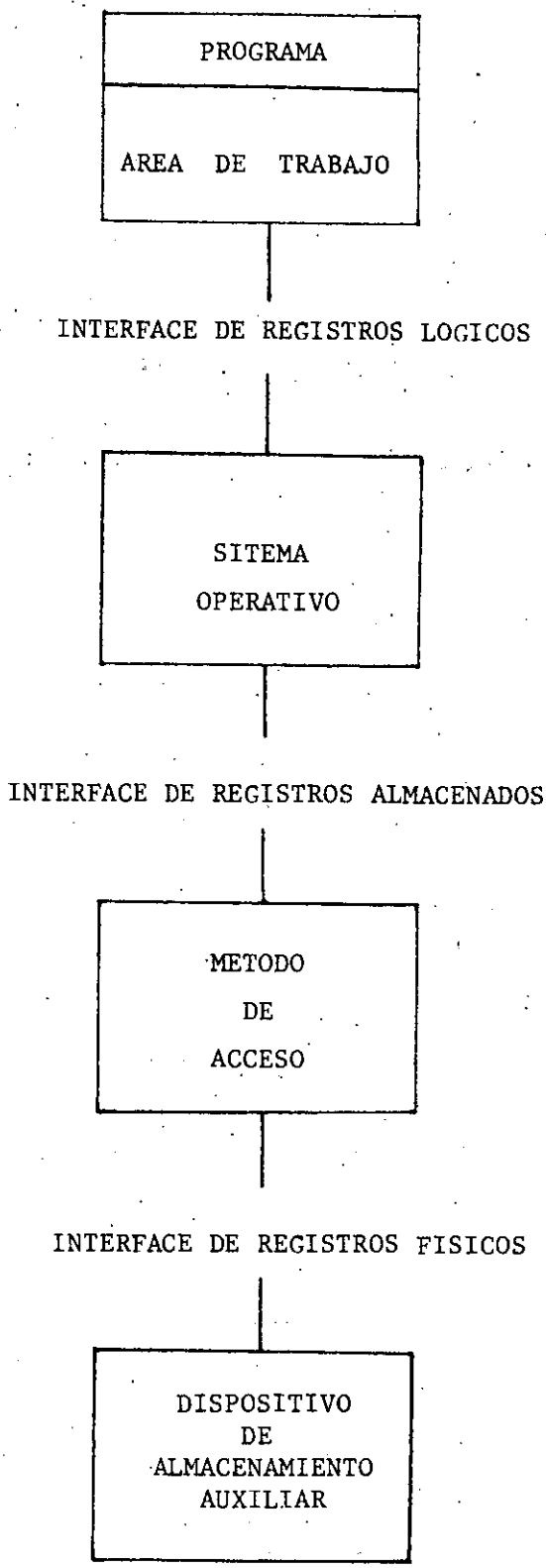
METODO DE ACCESO 4

Es el programa empleado para proveer una interface con el sistema operativo y los dispositivos de almacenamiento auxiliar.

El método de acceso realiza las siguientes funciones:

1. Búsqueda de un registro almacenado en un archivo determinado.
2. Forma de almacenamiento de un registro nuevo
3. Mantenimiento de registros existentes




PROGRAMA
AREA DE TRABAJO
INTERFACE DE REGISTROS LOGICOS
SITEMA OPERATIVO
INTERFACE DE REGISTROS ALMACENADOS
METODO DE ACCESO
INTERFACE DE REGISTROS FISICOS
DISPOSITIVO DE ALMACENAMIENTO AUXILIAR

DISPOSITIVOS DE ALMACENAMIENTO AUXILIAR Y METODOS DE ACCESO

I. Cintas Magnéticas

- SAM (Método de Acceso Secuencial)

II. Discos Magnéticos - Archivos con Regs. de Long. ITE.

- RAM Método de Acc. Aleatorio
- BDAM Método de Acc. Directo

III. Discos Magnéticos - Archivos con Regs. de Long. Var.

1) Indice disperso

- ISAM Método de Acc. Secuencial Indexado
- VSAM Método de Acc. Secuencial con llave

2) Indice calculado

- Hash addressing Org.

3) Indice Denso

- M.T.S. (Michigan Terminal System)

DISPOSITIVOS DE ALMACENAMIENTO AUXILIAR Y SUS CORRESPONDIENTES METODOS DE ACCESO

1. Cintas Magnéticas

- La lectura debe hacerse secuencialmente
- Bastante lento el proceso
- Densidad: 800, 1600, 6250 B.P.I. (BPI=BYTES por pulgada)

Por lo tanto existen dos aspectos que hay que considerar para la mejor utilización de las cintas magnéticas:

A. El Factor de Bloque

Bloque

El almacenamiento de datos en dispositivos magnéticos, requiere que sea hecho en forma óptima, por consiguiente se emplea la técnica de bloqueaje que consiste en reunir a cierto número de registros y almacenarlos como un solo elemento.

Registro Físico

Que es en sí el bloque.

Registro Lógico

Que es uno de los elementos que forman el bloque

Factor de Bloque

Es el número de registros lógicos en cada bloque.

Ejemplo:

DENSIDAD	FACTOR DE BLOQUE	REGISTRO LOGICO	EFICIENCIA
800 BPI	1	80	15%
800 BPI	100	80	94%

B. La Longitud fija o variable de los Registros.

Registros de Longitud Constante:

Mucho desperdicio de espacio si los registros no están completos siempre.

Registros de Longitud Variable:

No hay desperdicio de espacio dado que la longitud del registro siempre es la real.

Método de acceso secuencial (SAM)

- Los registros están ordenados de acuerdo a la llave primaria
- El acceso solo se puede realizar secuencialmente
- Puede ser utilizado en cintas o discos

Ventajas

a) Simple para entender e implementar

b) Las cintas magnéticas son bastantes económicas

c) Eficiente cuando se procesa una buena parte del archivo

d) Util para juntar archivos ordenados bajo la misma llave

Desventajas

a) No hay posibilidad de accesar aleatoriamente

b) Muy ineficiente para accesos individuales

c) Para agregar un registro se necesita copiar todo el archivo

d) Dificultad para agrandar registros existentes

Veredicto

Para cintas magnéticas no hay otra alternativa, pero cuando se usan discos entonces se recomienda para los siguientes casos: Cuando se van a accesar o modificar una proporción elevada de registros, cuando se va a realizar operaciones de intercalación o cuando se cuenta con un mantenimiento de registros en grupo.

2. Discos Magnéticos

Métodos de acceso para discos magnéticos usando registros de longitud constante.

METODO DE ACCESO ALEATORIO (RAM)

- Eliminación de movimiento del brazo
- Almacenamiento vertical por cilindro
- No hay necesidad de almacenar ordenadamente por llave

Ventajas

a) Muy rápido para aplicaciones en línea

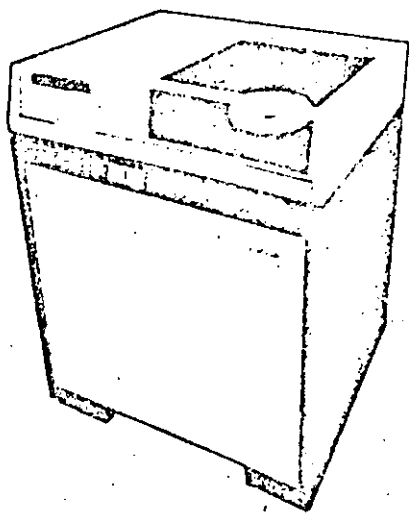
b) Especialmente bueno para accesos a registros individuales

Desventajas

a) Cuando el archivo requiere estar ordenado por llave primaria

b) Cuando se necesita intercalar archivos

UNIDADES DE DISCO MAGNETICO



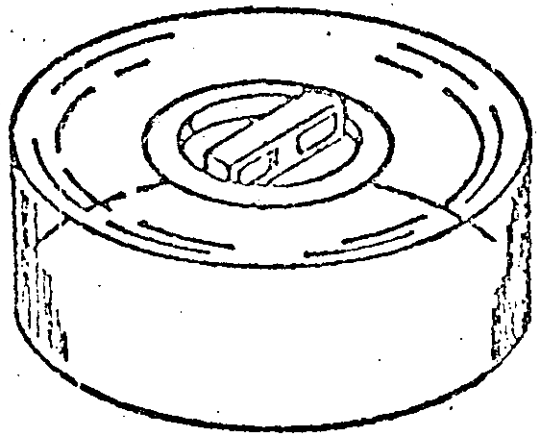
LA UNIDAD DE DISCO MAGNÉTICO ES UN DISPOSITIVO DE E/S VELOCIDAD QUE LEE Y ESCRIBE DATOS EN DISCOS MAGNÉTICOS.

CONTIENE UN IMPULSOR Y PAQUETES DE DISCOS REMOVIBLES E INTERCAMBIABLES QUE SE UTILIZAN PARA ALMACENAMIENTO AUXILIAR.

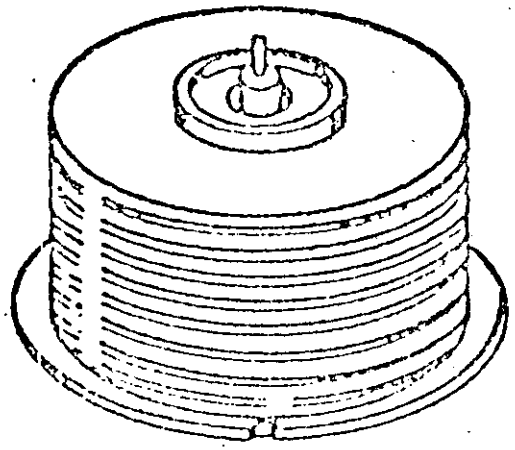
V4-24

PAQUETE DE DISCOS

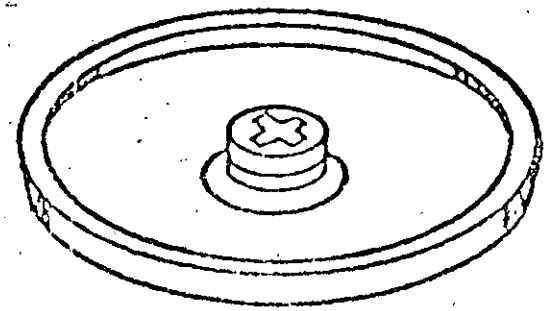
UN PAQUETE DISCO CONSISTE EN UN NÚMERO VARIABLE DE PLATILLOS DE 14 PULGADAS MONTADOS EN UN EJE VERTICAL.



CUBIERTA PROTECTORA Y MANIJA

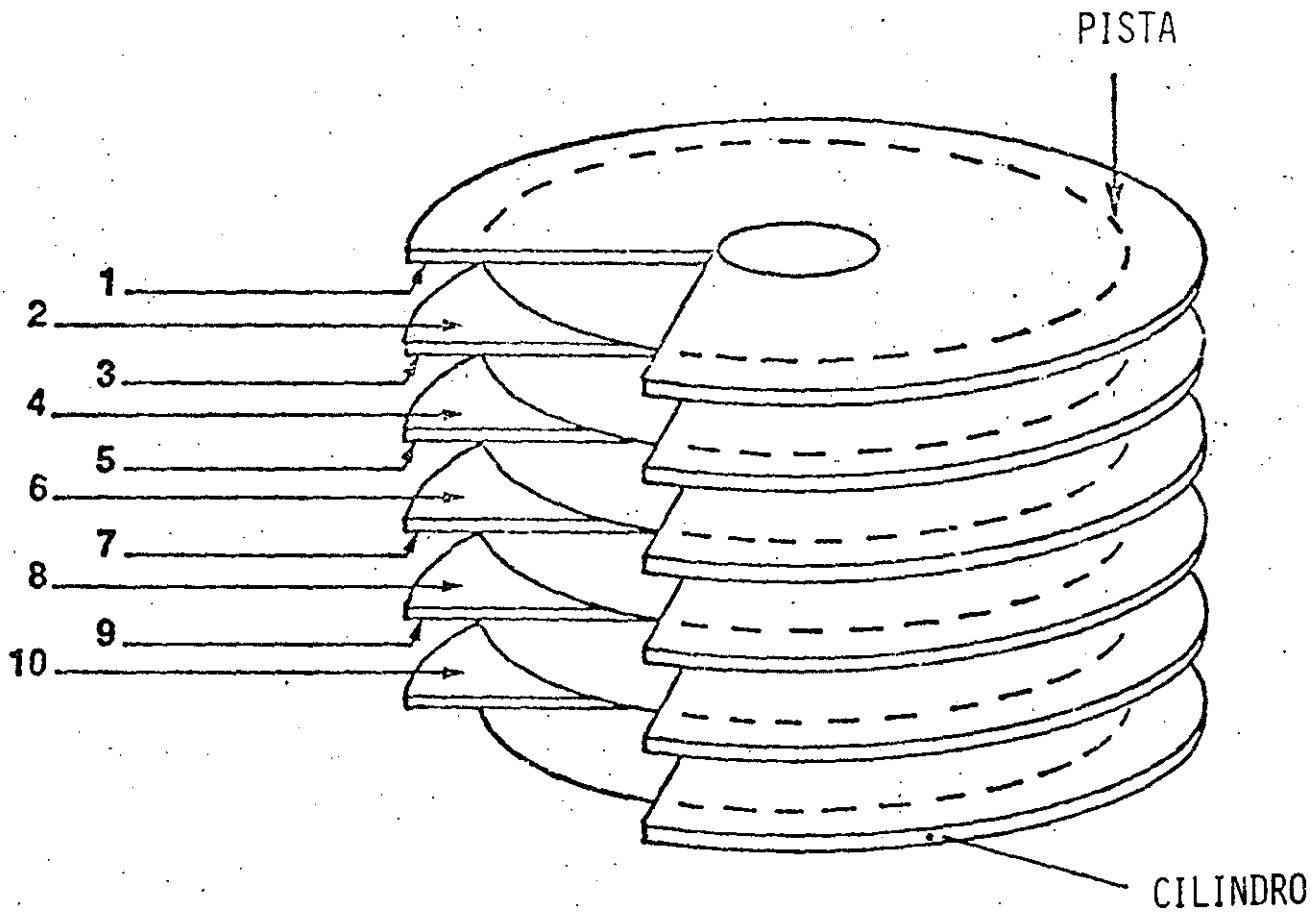


PAQUETE DE DISCOS



CUBIERTA INFERIOR

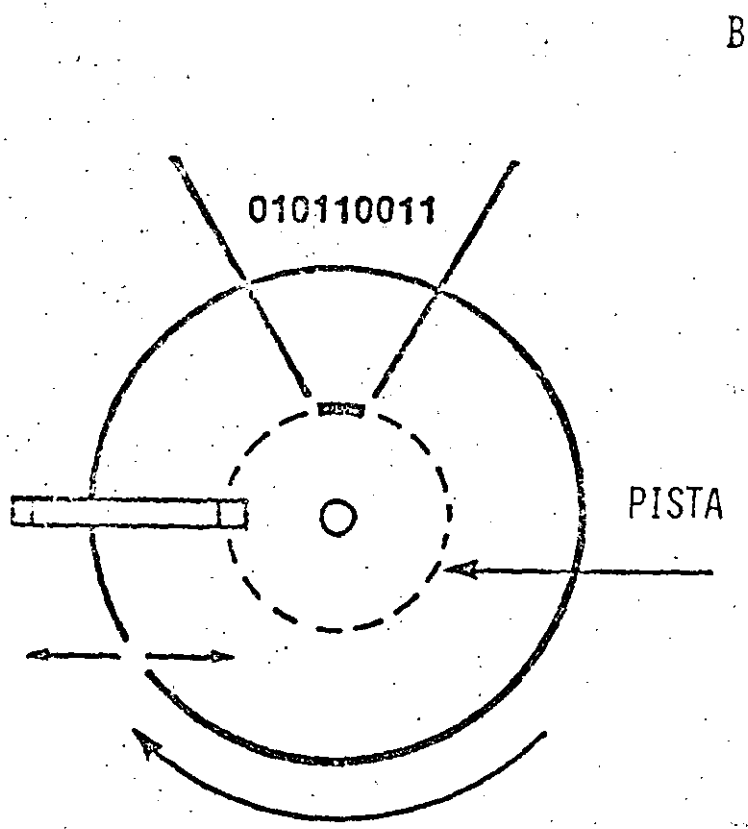
PISTAS PARA GRABACION EN UN PAQUETE DE DISCO

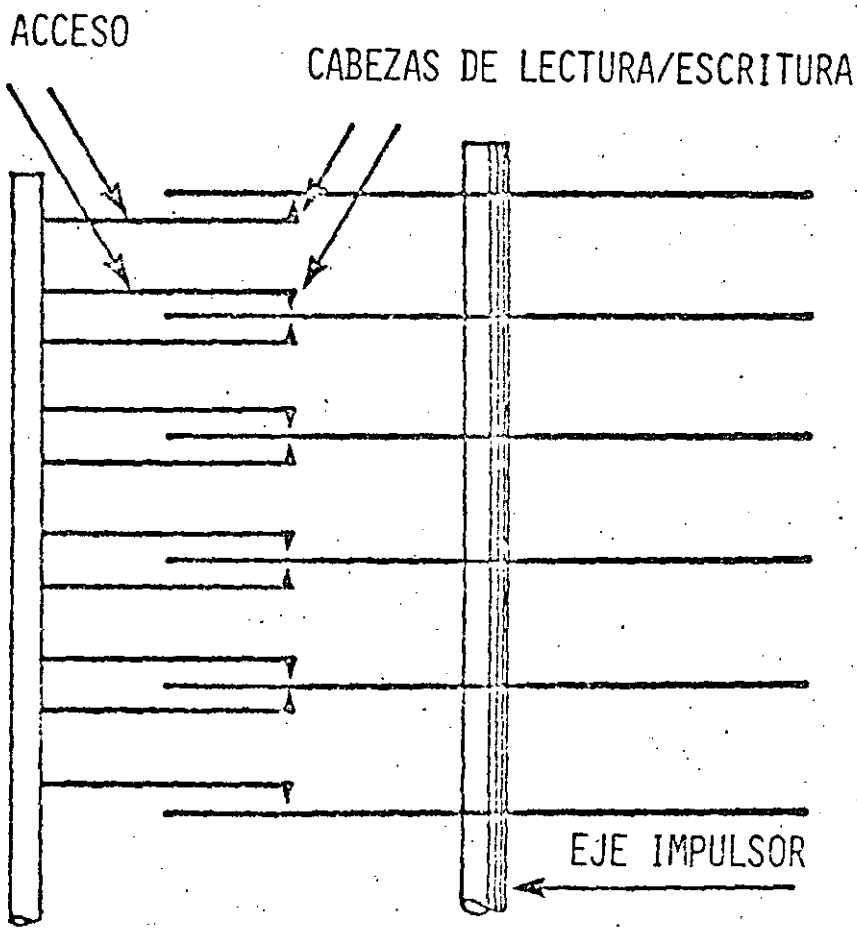


CADA SUPERFICIE PARA GRABACIÓN CONTIENE DE 200 A 800 PISTAS CONCÉNTRICAS DE REGISTRO.

V4-26

GRABACION EN LOS DISCOS





LOS DATOS SE GRABAN A TRAVÉS DE PUNTOS MAGNETIZADOS SOBRE LA SUPERFICIE DEL DISCO, EL CUAL ESTÁ CUBIERTO CON UNA CAPA DE ÓXIDO FERROSO.

V4-27

CAPACIDAD EN DISCOS (EQUIPO IBM)

MODELO	No.BYTES/ PISTA	No.PISTAS/ CILINDRO	No. CILINDRO	CAPACIDAD	TAMAÑO BLOQUE	No.BLOQUE/ PISTA	EFIC. %
3330-1	13030	400	19	100	80	61	37
					3156	4	97
					13030	1	100

RELACION COMPARATIVA DE TIEMPOS DE ACCESO

TIEMPO DE MOV. DE LAS CABEZAS	TIEMPO DE MOVIMIENTO DEL DISCO	TIEMPO TRANSMISION A MEMORIA PRINCIPAL 4000 BYTES =	TIEMPO TRANSMISION ENTRE BUFFER Y AREA DE TRABAJO
10-55 MS	8 MS	6 MS	2 MS
>	>	>	=

METODO DE ACCESO DIRECTO

- La llave primaria es el número de registro y no el valor de algún campo del registro.
- Los registros deben de ser de longitud constante.
- Se calcula la dirección del registro por medio del método de acceso.

Longitud de Registro = 12 bytes

No. EMPLEADO	SALARIO	EDAD
1	4,250	21
2	3,150	36
4	2,780	42
6	4,050	50

- Cada registro tiene una llave única
- La llave única primaria es densa

Ventajas

a) Acceso rápido. No hay accesos a disco extra dado que no hay over flow.

b) Fácil de comprender

c) Procesamiento secuencial es fácil de realizar dado que los registros están ordenados por llave primaria.

Desventaja

a) Mucho desperdicio de espacio. Si los registros son de longitud variable.

b) Inserciones de nuevos registros. Pueden ser facilmente realizados agregando al final del archivo.

c) Si se quiere insertar en el lugar correspondientes se tendrá que copiar todo el archivo nuevamente.

d) La llave primaria es artificial por lo tanto no está relacionada con los datos del usuario.

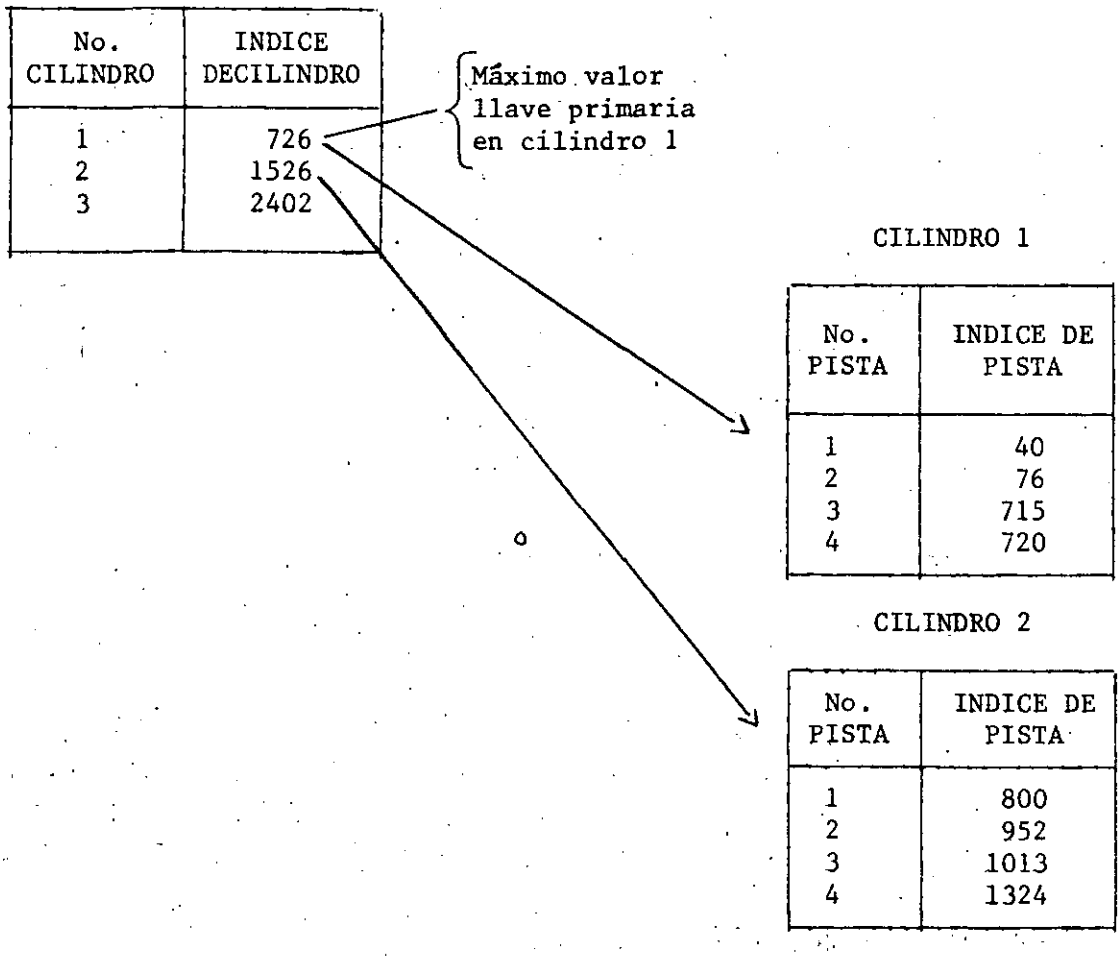
e) Es función del usuario escribir rutinas para localizar cada registro.

Veredicto

Este método es usualmente utilizado para archivos cuyas características no permiten el uso de organizaciones secuenciales o indexadas o para archivos en donde el tiempo para localizar registros individuales debe mantenerse al mínimo.

METODO DE ACCESO SECUENCIAL INDEXADO
(I S A M)

- Inicialmente los registros están ordenados por llave primaria.
- La posición aproximada es calculada primero por número de índice de cilindro y después una búsqueda secuencial dentro del cilindro.
- Estos índices indican el mayor valor de la llave primaria o registro localizada en cada uno de los cilindros y pista.
- Estos índices son dispersos ya que no contienen todos y cada uno de los valores para llave primaria en el archivo principal.



Como Funciona?

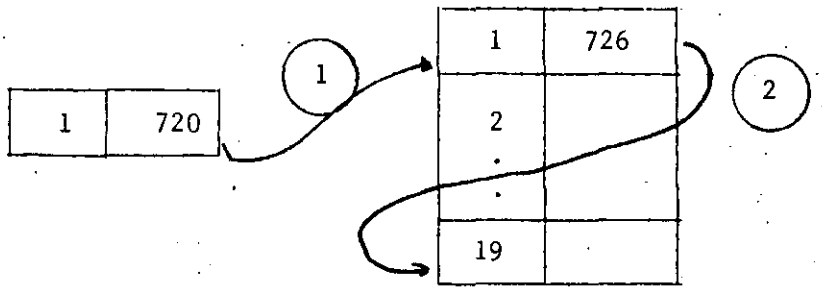
- Después de localizar el índice de cilindro, se realiza una búsqueda secuencial por la pista.

- Las inserciones se puede hacer pero son complicadas.

- Existen áreas de over flow en donde los nuevos registros son insertados, sin embargo esto complica mucho el proceso y lo hace lento.

Ejemplo:

Si queremos insertar el empleado o la llave primaria No. 3, este tiene que ir a una pista de over flow, digamos la 19 del cilindro 1. Entonces la próxima vez que se quiera accesar este registro serán necesarios 2 accesos a disco. Primero a la pista 1 del índice del cilindro y luego al cilindro del 19 por el índice de la pista 1.



- Si existen muchas inserciones hay que reorganizar el archivo secuencialmente.

Ventajas

a) Se tienen ambos métodos. El directo y el secuencial combinados.

b) Registros de longitud variable.

c) Llave primaria relacionada con un dato

Desventajas

a) Lento si hay que hacer muchas inserciones (muchos accesos a disco)

b) La programación es complicada.

METODO DE ACCESO DE MEMORIA VIRTUAL
(V S A M)

- Inicialmente las pistas están llenas hasta un 70%.

- Por lo tanto, las inserciones se realizan en la misma pista o sea cerca de donde se espera estén colocadas.

- Menos accesos de disco

INDICE CALCULADO

CASILLA No.	REGISTRO No.
0	
1	300
2	
3	
4	

CASILLA No.	REG. No.
5	200
6	500
7	
8	
9	100

CASILLA No.	REG. No.
10	400
11	
12	

Ejemplo. Registros 100, 200, 300, 400 y 500.

$$\frac{\text{No. Reg.}}{13} = \text{No. Casilla}$$

- Muchas coliciones

- Pero muy rápido

- Por lo tanto muy útil para aplicaciones en línea con una llave primaria diseprsa y bien controlada.

INDICE DENSO (listas ligadas)

DIRECCION	LINEA No.	EMPL.	SALARIO	EDAD
0	1			
8	2			
16	2.5			

LINEA No.	DIREC.
1	0
2	8
2.5	16

DIRECCION	LINEA No.	EMPL.	SALARIO	EDAD
0	1			
8	2			
16	2.5			
24	2.25			

LINEA No.	DIREC.
1	0
2	8
2.25	24
2.5	16

CONCLUSIONES

Las características inherentes de cualquier archivo que deben ser tomadas en cuenta cuando se escoge un método de organización eficiente son:

1. Volatilidad: Este término se refiere a la adición y eliminación de registros. Un archivo puede considerarse estático si su volatilidad es baja, mientras que es volátil si esta es alta. Por ello es importante considerar estas dos operaciones en los distintos métodos de acceso.

2. Actividad: Es el número de registros por archivo que usualmente se quieren accesar o consultar. Si la actividad es alta entonces el método sec. es de muy poca utilidad.

3. Tamaño: Si un archivo es de tal tamaño que no se puede tener todo en línea, entonces se deberá de tener mucho cuidado al escoger el método de organización. En cambio en un archivo muy pequeño la diferencia entre métodos no es significativa.

4. Crecimiento: Un archivo debe planearse para poder crecer con facilidad.

5. Tipo de índices y/o llaves: Denso o disperso.

I P N

UPIICSA - 22

DEPARTAMENTO DE COMPUTACION

UNIDAD DE INVESTIGACION Y DESARROLLO

INSTITUTO NACIONAL DE
ESTADISTICA, GEOGRAFIA E INFORMATICA
DIRECCION GENERAL DE POLITICA INFORMATICA
DIRECCION DE DESARROLLO INFORMATICO
PROGRAMA INTERNO DE CAPACITACION

SISTEMAS DE BASE DE DATOS

FERNANDO GALINDO SORIA

FEBRERO 1984

SISTEMAS DE BASE DE DATOS

OBJETIVO GENERAL

Al final del evento el participante será capaz de aplicar los conceptos y herramientas de los sistemas de base de datos a - la solución de problemas de información.

CONTENIDO

CAPITULO I. CONCEPTOS GENERALES

OBJETIVO ESPECIFICO

El participante será capaz de abstraer y representar los componentes de una base de datos a partir de su medio ambiente.

CONCEPTOS GENERALES

El ser humano siempre ha tratado de representar y comprender la realidad y en ese intento ha generado múltiples teorías - y desarrollado su creatividad artística, tecnológica y científica. Uno de los últimos modelos representa la realidad - como un sistema dual: materia-información, en el cual para cada elemento físico existe una contrapartida de información (en particular esta concepción de la realidad establece la - base de la informática, en la cual se pretenden desarrollar- y usar sistemas de información, donde cada uno de estos puede ver, en su forma más sencilla, como un conjunto de ele-- mentos de información relacionados entre sí, mediante acciones que transforman esa información).

Partiendo del enfoque dual material-información y generali-- zándolo para englobar en lugar de materia a cualquier cosa - que el ser humano pueda conceptualizar, se puede tratar de - representar la realidad mediante un proceso de abstracción - en el cual se intenta encontrar un modelo excluyendo - - - detalles no relevantes (es decir, realmente no se representa la realidad como un todo, sino que, únicamente se representa una pequeña parcela).

Este modelo parte de la premisa de que la realidad se puede ver compuesta por un conjunto de elementos, conocidos como -

ENTIDADES, donde cada entidad se caracteriza por un conjunto de ATRIBUTOS (características que permiten distinguirlas de otras entidades) y el modelo además incluye la dinámica del sistema, es decir toma en cuenta las acciones que se realizan sobre el sistema.

Las entidades se encuentran a su vez interrelacionadas unas con otras, mediante RELACIONES de tipo SINTACTICO, SEMANTICO o PRAGMATICO, donde las relaciones sintácticas únicamente representan la estructura básica (esqueleto), mediante la semántica se asocia significado a las entidades, atributos y relaciones, y en la pragmática se particulariza el significado a un contexto dado o vista particular de algún observador.

Según el enfoque anterior existen múltiples vistas particulares de la realidad por lo que es necesario establecer criterios y métodos para encontrar VISTAS GLOBALES en las cuales se compaginen diversas vistas particulares.

Por otro lado se tiene que tomar en cuenta que múltiples entidades se pueden caracterizar por un mismo conjunto de atributos, por lo que, en un problema concreto es necesario distinguir entre diferentes entidades, para lo cual, a cada uno de sus atributos se les asocia un VALOR distintivo. Por ejemplo las personas (entidades) se puede caracterizar mediante los atributos nombre, edad, sexo, etc., pero si se quiere ubicar a una persona particular, se tiene que concretizar, por ejemplo: nombre/Juan Pérez, edad/20 años, sexo/masculino, etc., donde Juan Pérez, 20 años y masculino representan DATOS concretos de una persona dada.

Por otro lado, al concretizar, de todas las posibles entidades y atributos que conforman una vista de la realidad, se

elijen sólo los más relevantes (dependiendo de necesidades e intereses) y con ellos se crea un DICCIONARIO DE DATOS, el cual a su vez, sirve de base para obtener un ESQUEMA en el cual se incluyen las entidades, atributos y relaciones de la vista global.

Siguiendo con este proceso a cada vista particular se le asocia lo que se conoce como SUBESQUEMA.

Al conjunto de datos que sirven para representar a cada entidad, atributo o relación de un esquema dado se le conoce como BASE DE DATOS, de donde una base de datos es la concretización de un modelo de información de la realidad y está formada por un conjunto de datos relacionados entre si.

EJERCICIOS

1.- Localice en su medio ambiente al menos 20 entidades.
2.- Agrupe a las entidades en entidades del mismo tipo. Para cada uno de los tipos de entidades mencione 3 atributos.
3.- Construya un sistema relacionando entre si a los tipos de entidades.
4.- Compare la vista que obtuvo de su medio ambiente con otras dos vistas diferentes del mismo medio ambiente.
5.- Integre las vistas en una vista global.
6.- Obtenga el diccionario de datos de la vista global.
7.- Obtenga el esquema de la vista global.
8.- Obtenga el subesquema de su vista particular.
9.- Asocie valores a cada una de las entidades.
10. Contruya la base de datos
    a) Para todas las entidades del mismo tipo
       Construya arreglos como el siguiente:

	ATRIBUTO 1	ATRIBUTO 2	ATRIBUTO m
ENTIDAD 1	VALOR 1, 1	VALOR 1, 2	VALOR 1, m
ENTIDAD 2	VALOR 2, 1	VALOR 2, 2	VALOR 2, m
.	.	.	.
.	.	.	.
.	.	.	.
ENTIDAD n	VALOR n, 1	VALOR n, 2	VALOR n, m

Relacione los tipos de entidades entre si, indicando que - usa para relacionar.

CAPITULO 2 SISTEMAS DE BASE DE DATOS

SISTEMAS DE BASE DE DATOS

OBJETIVO ESPECIFICO

El participante comprenderá la relación existente entre los componentes de un sistema de base de datos y los ubicará - dentro de una organización.

SISTEMAS DE BASE DE DATOS

Encontrar un proceso que permita obtener una Base de Datos a partir de la realidad ha requerido un gran esfuerzo por parte de muchos investigadores y hasta el momento no se ha completado, sin embargo, ya existen algunos propuestos, uno de los cuales se presenta a continuación.

Este proceso consta de 2 etapas divididas en 6 pasos.

Durante la primera etapa se logra obtener a partir de la Realidad un ESQUEMA LOGICO en el cual se representan concretamente las entidades y relaciones de información relevantes para la organización que desarrolla el proceso.

La primera etapa es netamente Informática y no presupone la existencia de computadoras, por lo que, a partir de esta etapa se puede tomar la decisión de utilizar el Esquema Lógico como base para un Sistema de Información MANUAL, SEMI-AUTOMATIZADO o AUTOMATIZADO.

En este documento se partirá de que se desarrollará un sistema de información automatizado, por lo que en la segunda etapa se parte del Esquema Lógico y se obtiene un ESQUEMA FISICO o representación de los datos dentro de la computadora.

Los 6 pasos del proceso son:

* ETAPA 1 INFORMATICA
- ABSTRACCION
- INTEGRACION
- CONCENTRACION
* ETAPA II AUTOMATIZACION
- PROGRAMACION
- REPRESENTACION INTERNA
- REPRESENTACION FISICA

En el paso de ASTRACCION, cada uno de los organismos involucrados obtiene una vista de la realidad, esta vista puede ser de tipo GENERAL cuando trata de abarcar toda la organización o PARTICULAR cuando se concreta a algún área específica.

Durante la INTEGRACION las diferentes vistas se integran para formar la Vista Global. Como siguiente paso se lleva a cabo un proceso en el cual se CONCRETIZA la Vista Global en un Esquema Lógico en el cual cada una de las entidades ocupa una posición concreta dentro del esquema y por su lado las relaciones que aparecen tienen una razón de ser y están completamente descritas.

En este punto se termina la primera etapa y comienza la etapa de Automatización.

En el paso de PROGRAMACION se toma el Esquema Lógico y se adecua a las características del sistema de cómputo en el que será implantada la Base de Datos.

La REPRESENTACION INTERNA, nuevamente el sistema se encarga de asignar localidades físicas reales a cada uno de los componentes de la Base de Datos y de construir una estructura de datos que permita manejar la Base de Datos.

Normalmente el proceso esquematizado anteriormente se lleva a cabo dentro de una organización y debe tender al logro de los objetivos de ésta, usando la Base de Datos como una herramienta para la toma de decisiones al proporcionar información que permite administrar la organización, por lo anterior, cada vez surge con más fuerza la idea de que la información es un recurso y en un momento dado puede llegar a ser el recurso más importante para la organización, es por esto que actualmente se presenta la necesidad de establecer una función (desarrollada por una o más personas) orientada a la ADMINISTRACION DE LA INFORMACION, el responsable de esta función o administrador de la información debe administrar la Base de Datos de la organización, buscar que se oriente al logro de los objetivos de ésta, y coordinar el proceso de desarrollo de la Base de Datos tanto en su etapa informática como en su etapa automatizada.

Resumiendo se puede observar que existen cuatro elementos estrechamente relacionados y formando un SISTEMA DE BASE DE DATOS:

La Administración de la Información, el proceso informático de la Base de Datos, el proceso automatizado de la Base de Datos y la Base de Datos misma.

El Sistema de Base de Datos se encuentra a su vez relacionado con una Organización la cual está inmersa en una partícula de la realidad.

CAPITULO III ADMINISTRACION DE LA INFORMACION

ADMINISTRACION DE LA INFORMACION

OBJETIVO ESPECIFICO

El participante comprenderá la necesidad de administrar los datos, y conocerá las funciones de un administrador de datos y de un administrador de la Base de Datos.

ADMINISTRACION DE LA INFORMACION

INTRODUCCION

Al crear abstracciones de la realidad, uno de los conceptos que más se maneja es el de SISTEMA o conjunto de elementos-interrelacionados entre sí, a un sistema que ejecuta un conjunto de acciones con el fin de lograr un objetivo (sistema teológico) lo llamaremos una ORGANIZACION.

Las organizaciones se encuentran dentro de un MEDIO AMBIENTE, el cual está formado por otros sistemas relacionados con la organización, estas relaciones por lo común representan un - intercambio entre la organización de MATERIA, ENERGIA, INFORMACION o conbinaciones de éstas.

Al llevarse a cabo este intercambio, la organización por - lo común busca:

a) Adecuarse al medio ambiente (adaptación)
b) Mantenerse en el medio ambiente (o nicho ecológico)
c) Transformar el medio ambiente
d) Mantenerse en un medio ambiente que se está trans-- formando.

En este proceso de adaptación-transformación se crea un flujo de información, mediante el cual la organización puede conocer el estado del medio ambiente.

Después de que una organización ha estado cierto tiempo dentro de su medio se llega a una etapa de estabilización en la cual la mayoría de las acciones que se realizan se vuelven repetitivas, o sea que para cierta modificación del medio (detectada por un flujo de información) el organismo siempre responde de la misma forma, creándose dentro del sistema un SUBSISTEMA OPERATIVO.

Sin embargo, dado que las organizaciones tienen un propósito (el cual normalmente no es el de quedarse estáticas) aparte de las acciones operativas, se realizan acciones tendientes a lograr dicho propósito. Estas acciones pueden ser de tipo TACTICO (indican como lograr algo) y ESTRATEGICO (indican los posibles caminos a seguir). Tanto para la táctica como para la estrategia se requiere como componente fundamental la información que indica el estado del medio y como se transformaría el medio si se lleva a cabo una acción determinada.

De lo anterior, se ve que, para que una organización pueda sobrevivir en un medio ambiente y en su caso llegar a cumplir con sus objetivos, se requiere un flujo contínuo de información, constituyendo el sistema nervioso de la organización.

Como ejemplos típicos de organización tenemos a las personas físicas, las entidades sociales y cualquier sociedad (país, estado, etc.).

En cada una de estas organizaciones se pueden ejemplificar los conceptos anteriores:

a) En la persona física existe un conjunto de acciones - (respirar, circulación de la sangre, etc) que se realizan en forma autónoma, sin que realmente ocurra una - toma de decisiones concientes y solamente se regulan - mediante un proceso de información (sistema endocrino, sistema nervioso, etc) que detecta las modificaciones- del medio ambiente (cambio de temperatura, disminución del aire, etc).

Por otro lado, existe una serie de acciones que la persona física realiza en forma consciente y con el fin de lograr su objetivo (estudiar, escapar de un incendio, etc) y las cuales también son reguladas por el flujo de información - (temas de estudio, complejidad del tema; detección del - calor, etc).

Además de lo anterior, la persona física lleva a cabo procesos de toma de decisiones en los cuales marca las rutas que seguirá para lograr sus propósitos vitales, en este - proceso también se nota la importancia de contar o no con información (ya que alguien podría proponerse como propósito estudiar para perforista por desconocer que actualmente casi no se utiliza la perforadora de tarjetas; con información tal vez estudiaría para capturista).

b) Como ejemplos de entidades sociales se tienen: los hospitales, bancos, industrias, comercios, escuelas, oficinas del gobierno, etc., y en cada una de ellas se podría realizar un estudio parecido al del inciso a.

c) En el caso de una sociedad, se tiene como ejemplo a los países, en donde para el funcionamiento del país - también existen acciones operativas, tácticas y estratégica, junto con el nivel político, o toma de decisión - del rumbo del país. Nuevamente un factor clave es la - información.

USO DE LA INFORMACION EN LA ORGANIZACION

Dentro de una organización la información es una herramienta para la toma de decisiones, ya que los datos que se captan, - tanto externos como internos, son procesados para proporcionar una información que a su vez al ser interpretada permite la toma de decisiones y finalmente el llevar a cabo un conjunto de acciones.

Si se observa una organización tipo, se ve que la informa--- ción incide en los niveles estratégico, táctico y operativo, pero, su función es diferente, ya que en el nivel estratégico la mayoría de las actividades tienen que ver con la pla-- neación y por otro lado a nivel operativo casi todas las decisiones son tendientes a controlar.

Por otro lado la importancia que se da a la información varía de una organización a otra, desde el rechazo total a su uso, hasta las organizaciones que basan todas sus acciones en la información.

Otro punto que se debe considerar sobre el uso de la información en la organización es la forma en que se interpreta - ésta. Para lo cual se tiene que considerar que la entidad - que toma las decisiones recibe un conjunto de influencias - incluyendo:

- Relaciones con su medio ambiente
- Información previa
- Estilo cognoscitivo
- Propósitos personales
- Etc.

Las relaciones con el medio ambiente incluyen la posición de la entidad dentro de la organización, la influencia de sus - decisiones, su relación con otras organizaciones, etc.

Dentro de la información previa se tiene por ejemplo: otros datos relacionados, todo el conjunto de vivencias previas - (memoria), decisiones anteriores, etc.

El estilo cognocitivo se refiere al tipo de información que se prefiere, ya sea cualitativa (en base a intuición) o cuantitativa.

Y los propósitos personales, son el conjunto de deseos, me-- tas y objetivos que tiene cada ente en particular.

También se tiene que considerar la forma como se distribuye la información en la organización, ya que, comparándola - con la afluencia de sangre, si falta la sangre, el organismo muere por inanición y si en algún punto afluye más de la debida, puede sobrevenir una embolia, lo anterior implica - la necesidad de establecer un sistema en el cual la información se distribuye selectivamente de acuerdo a las necesidades.

Otro punto que se tiene que considerar respecto a la rela--- ción entre la organización y la información, es el de que, - aunque la organización es un ente dinámico y sufre múltiples cambios durante su vida, existe un núcleo de funciones - - -

básicas que no sufren muchas modificaciones, por lo que, si bien las necesidades de información en general son cambiantes, existe un núcleo de requerimientos de datos que se mantiene estable.

CARACTERISTICAS DE LA INFORMACION

Cuando se ve la información que se maneja en una organización se detecta que ésta tiene múltiples características, y en particular se considerararán las siguientes:

- Fuente
- Selectividad
- Alcance
- Volatilidad
- Marco temporal
- Frecuencia
- Espectativa
- Aleatoriedad

La fuente de una información puede ser interna, externa o mixta, normalmente las áreas operativas utilizan información interna y las áreas estratégicas más externa que interna.

La distribución selectiva de información puede ser de tipo general, o sea aquélla que cubre a la organización (ve al bosque) o de tipo especifico sobre algún componente (una rama).

El alcance tiene que ver con la selectividad y se refiere al grado de detalle o resumen en que se presente la información.

El marco temporal de la información puede ser histórico, actual o prospéctico.

La frecuencia de la información puede ser contínua, periódica o irregular.
La expectativa se refiere a si se anticipo la llegada de esa información o fue inesperada.

Con la aleatoriedad se refiere a si es determinística o probabilística. Dependiendo del tipo de aplicación es el tipo de información que se requiere y por lo tanto la forma de manejo de los datos.

ADMINISTRACION DE LOS DATOS

En los puntos anteriores se vió la importancia de la información como uno de los recursos vitales para un organismo, por lo que actualmente ha surgido la tendencia a crear mecanismos que permitan administrar este recurso.

En primera instancia estos mecanismos tienden a administrar el sistema de información y todos los recursos asociados a él, pero dado que, en el sistema de información de una organización se pueden detectar dos componentes básicos: los datos y las transformaciones o procesos que se llevan a cabo sobre estos, para obtener la información, cada vez es más común la existencia de un área orientada a la ADMINISTRACION DE LOS DATOS, la cual es responsable de establecer las políticas y procedimientos para el manejo de los datos dentro de la organización. El administrador de los datos tiene una función más administrativa que técnica, ya que es el responsable de la Planeación Estratégica y Táctica, y de la organización y control del Sistema de Base de Datos, el responsable directo de administrar la Base de Datos se conoce como

ADMINISTRADOR DE LA BASE DE DATOS y él organiza y controla - la Base de Datos, por lo que tiene una función más técnica - que administrativa.

En general las principales funciones del Administrador de - los Datos son:

- Desarrollo e implantación de políticas para el manejo del S B D
- Desarrollo de los planes estratégicos y tácticos del S B D
- Determinar los requerimientos de información de la - organización y asignar prioridades de desarrollo.
- Administrar la vista global de los datos.
- Encontrar nuevas áreas de aplicación y formas de compartir la Base de Datos.
- Extender el conocimiento del S B D sobre toda la organización.
- Organizar el S B D y obtener los recursos necesarios para su desarrollo.
- Establecer sistemas de control sobre la privacidad, integridad y seguridad de los datos confiados a su - administración.
- Establecer un Sistema de Auditoría Interna del S B D.

Por su parte entre las funciones del Administrador de la - - Base de Datos se encuentran:

- Administrar el uso de la Base de Datos buscando la - seguridad, eficiencia y economía.
- Relación con los usuarios, incluyendo:
  . Proporcionar las vistas que requieren los usuarios.

- Mantener la descripción de los datos
- Coordinar la compartición de los datos.
- Coordinar las facilidades a los usuarios
- Comprender las necesidades de información y relacionarlas con los datos de la Base de Datos.

- Coordinación del área de Desarrollo de la Base de Datos incluyendo:
  - Coordinación de análisis y diseñadores
  - Administrar el Diseño Lógico de la Base de Datos.
  - Decidir el contenido de la Base de Datos.
  - Detectar la organización de datos requerida.
  - Estructurar los esquemas y subesquemas
  - Decidir las estructuras de almacenamiento y las estrategias de acceso.
  - Creación de la Base de Datos
  - Documentación de la Base de Datos
- Coordinación del área de cómputo (en caso de Base de Datos automatizada) de la Base de Datos incluyendo:
  - Coordinación de los programadores
  - Mantenerse al día en tecnología de Base de Datos
  - Obtención del Sistema de Manejo Automatizado de la Base de Datos (construcción compra, adecuación, etc.)
  - Mantener las estructuras de almacenamiento físico de la Base de Datos
  - Coordinar el procesamiento de los datos
  - Documentar el sistema
- Coordinación de la operación de la Base de Datos incluyendo:
  - Definición de métodos y procedimientos
  - Control de cambios a la Base de Datos
  - Responder a los requerimientos cambiantes de los usuarios
  - Buscar el incremento de la eficiencia.

  - Evaluación de la Base de Datos
  - Control del uso de la Base de Datos
  - Medición del rendimiento
  - Reorganizaciones a la Base de Datos
  - Afinación de la Base de Datos
- Coordinación de la seguridad de la Base de Datos incluyendo:
  - Integridad de la Base de Datos (respaldos, procedimientos de recuperación, etc).
  - Privacidad
  - Protección

CAPITULO IV DISEÑO LOGICO DE BASE DE DATOS

DISEÑO LOGICO DE BASE DE DATOS

OBJETIVOS ESPECIFICOS

- El participante comprenderá como se obtienen las vistas de datos particulares y generales de una organización y a - - partir de ellas como se construye la vista global.

- Comprenderá como se construye el esquema lógico de los datos de una organización.

DISEÑO LOGICO DE BASE DE DATOS

El diseño lógico de la Base de Datos es un proceso informático independiente del proceso automatizado y lo mismo se puede utilizar para encontrar la relación de datos a nivel personal, los datos involucrados en la administración de una organización (banco, hospital, empresa, escuela, etc), o los tipos de datos que se manejan en un Archivo General de la Nación y la relación entre estos.

El diseño lógico de Base de Datos parte del universo de una organización y encuentra la estructura de entidades, atributos y relaciones (desde el punto de vista de los datos) que la componen, para lo cual lleva a cabo tres grandes etapas:

- Abstracción
- Integración
- Concretización

En la etapa de abstracción toma el universo de una organización y detecta las diferentes entidades que manejan información, a continuación encuentra las diferentes formas como ven los datos estas entidades y sus requerimientos de información (vistas particulares y generales).

A partir de las vistas particulares y generales, en la etapa de integración se obtiene una vista global en la cual se - -

encuentra representado el universo de los posibles datos y - las diferentes entidades que lo conforman.

Finalmente en la etapa de concretización a partir de la vista global, se toman las entidades, sus relaciones y los atributos asociados y se obtiene un diagrama de entidades a partir del cual se pasa a un modelo relacional y se normaliza - para obtener un esquema normalizado de la Base de Datos.

En el diseño lógico se intenta encontrar un modelo de datos- del universo, excluyendo detalles no relevantes para la organización. Este modelo parte de que el universo está compuesto de entidades y cada entidad se representa mediante un conjunto de atributos.

El modelo debe tomar en cuenta las diferentes vistas de los usuarios del sistema y buscar que la Base de Datos sea - - - COMPLETA (es decir, que para cualquier vista existan todos - los atributos en la Base de Datos).

Dado que diferentes vistas se pueden referir a las mismas entidades y atributos y esto se reflejaría como redundancia o duplicidad de información a menos que se logre disminuir, el PROBLEMA PRINCIPAL DE DISEÑO LOGICO consiste en encontrar un modelo completo que tome en cuenta las diferentes vistas de los usuarios, los tipos de entidades y las relaciones entre éstas, minimizando la redundancia.

OBTENCION DE LA VISTA GLOBAL DE LA BASE DE DATOS

Una de las principales funciones del administrador de los datos es el desarrollo de los planes estratégicos del Sistema - Base de Datos y en particular determinar los requerimientos - de información y las prioridades de desarrollo del Sistema - Base de Datos dentro de la organización.

Para llevar a cabo estas funciones se necesita el desarrollo de los pasos de ABSTRACCION E INTEGRACION de las vistas en - una vista global, en este punto se presenta un proceso orientado a obtener la vista global de la Base de Datos.

Este proceso consta de las siguientes etapas:

- Planeación y organización
  - . Formación del Comité de Base de Datos
  - . Investigación de la situación en la organización
  - . Planeación del desarrollo de la vista global.

- Obtención de las vistas particulares y generales - -- (abstracción)
  - . Definición de la cobertura del Sistema Base de Datos
  - . Detección de las vistas

- Integración de la vista global
  - Construcción del Diccionario de Datos
  - Reducción del universo de Datos
  - Asignación de prioridades para integrar los datos al Sistema Base de Datos.

A continuación se describe cada una de las etapas

PLANEACION Y ORGANIZACION

En esta etapa se estudia la situación de la organización con respecto al Sistema Base de Datos y se propone un plan para detectar la vista global de la organización, en general se desarrollan las siguientes actividades:

1) Formación del Comité de Base de Datos
   En este punto se debe adquirir el compromiso al más alto nivel para apoyar el desarrollo del proyecto. Se establece la magnitud del estudio y los requerimientos de recursos. Finalmente se forma un comité responsable del estudio, formado por un comité directivo integrado por directivos de la organización y responsable de la toma de decisiones y por un comité técnico responsable del desarrollo del estudio.

2) Investigación de la situación en la organización
   Durante esta actividad se estudia la organización con el fin de conocer el marco general en el cual se implantará el Sistema Base de Datos, este estudio incluye:

- Historia general de la organización
- Estructura actual interna
  - Planes
  - Organización
  - Productos o servicios que presta
  - Areas problema
  - Distribución geográfica
  - Estadísticas sobre la organización
  - Etc.

- Consideraciones a futuro
  - Cambios previstos
  - Modificación futura de políticas
  - Etc.

También en este punto se investiga la situación del Sistema Base de Datos actual incluyendo:

- Historia del Sistema Base de Datos
- Situación del Sistema Base de Datos
  - Planes actuales
  - Organización
  - Distribución geográfica
  - Principales problemas
  - Sistemas actuales
  - Recursos actuales
  - Etc.

- Consideraciones a futuro
  - Modificaciones propuestas al Sistema Base de Datos
  - Obtención de recursos
  - Implantación de nuevos sistemas
  - Etc.

- Cotas del Sistema Base de Datos.
  - Cotas internas
    - i) Administrativas
    - ii) Recursos
    - iii) Aceptabilidad
  - Cotas externas
    - i) Economía
    - ii) Gobierno
    - iii) Leyes, etc.

- Relación entre la organización y el Sistema Base de Datos.
  - Tareas relacionadas con el Sistema Base de Datos.
  - Relación áreas funcionales/manejo de datos.
  - Costo general del manejo de datos.

- Estructura actual externa
  - Economía
  - Clientes y proveedores
  - Sistema político
  - Posición de la organización en su medio ambiente.
  - Desarrollo tecnológico
  - Ambito social.

Como última actividad de este punto se detecta, analiza y proponen soluciones generales a los problemas de manejo de datos.

3) Planeación del desarrollo de la Vista Global.
En este punto se establece un plan en el cual se indican cuales serán los pasos para obtener la vista global de la Base de Datos.

- El Plan deberá contener, entre otros, lo siguiente:
  - Propósitos
  - Premisas
  - Marco y acción y restricciones
  - Objetivos
  - Actividades
  - Ruta crítica

Ya elaborado el plan, el siguiente paso consiste en presentarlo a la directiva y obtener su apoyo para ejecutarlo y finalmente sensibilizar a los ejecutivos sobre la necesidad de su participación en el plan.

ETAPA DE ABSTRACCION

En esta etapa se detectan las diferentes vistas de los datos. En el desarrollo de esta actividad se tienen 2 enfoques extremos:

a) Vistas de estructura de información.
Toma en cuenta los atributos que debe tener cada una de las entidades relaciones entre las entidades, etc. y a partir de ahí se encuentran las vistas.

b) Vistas de estructura de procesos.
Detecta las necesidades de los usuarios, incluyendo qué información produce y cuál usa, y obtiene las vistas.

Los métodos de abstracción realmente son una combinación de estos enfoques ya que toman en cuenta tanto las vistas de estructura de procesos como de información.

Las actividades que se proponen para esta etapa son:

- Definición de la cobertura del Sistema Base de Datos.
  - . Investigación preliminar de la organización.
  - . Integración del directorio de usuarios y productores de información.

- Detección de las vistas.
  - . Recopilación de datos en las áreas de la organización.
  - . Detección de la información recibida, necesaria y producida por cada área de la organización.
  - . Obtención de un diccionario de las vistas.

1) Definición de la cobertura del Sistema Base de Datos. Como primer paso se delimita la cobertura del Sistema Base de Datos, detectando quiénes manejan los datos, para lo cual se realiza lo siguiente:

- Investigación preliminar.

  Detección de los puestos oficiales y reales en la organización.
- Integración del directorio.
  - . Detección de usuarios y productores de información.
  - . Integración de directorio preliminar.
  - . Obtención de información sobre los puestos mencionados en el directorio (función, ubicación, teléfono, objetivos, programas, etc.)
  - . Separación por áreas operativas, tácticas y estratégicas.
  - . Integración directorio.

2) Detección de las vistas.

En este punto se encuentra como va el universo de información cada uno de los usuarios y productores a nivel operativo, táctico y estratégico, y se realiza lo siguiente:

- Recopilación de información en áreas tácnicas y estratégicas.

  Se busca detectar el tipo de información necesaria para planear y controlar en la organización.

  . Recopilación de información en áreas estratégicas.

    i) Componentes básicos de la empresa.
    ii) Cambios pronosticados que pueden afectar la organización.
    iii) Datos necesarios para planear y dirigir la organización.
    iv) Datos que producen el área estratégica.
    v) Lista de entidades asociadas al área.
    vi) Principales atributos de cada entidad.

  . Recopilación de información en áreas tácticas.

    i) Detección de la interfase entre áreas.
    ii) Detección de reglas y políticas sobre las operaciones.
    iii) Detectar las necesidades de información para planear y controlar las actividades.
    iv) Datos que producen el área táctica.
    v) Lista de entidades asociadas al área.
    vi) Principales atributos de cada entidad.

- Recopilación de información en áreas operacionales.

Se busca detectar las tareas realizadas y que - entidades y atributos tiene asociados cada tarea, para lo cual:

- Se detectan las actividades, funciones, acciones, decisiones e interrelaciones (diarias, - semanales, mensuales, etc) que se realizan en cada área.
- Se desarrolla un diagrama de flujo (o diagrama Warnier-Orr, etc.) donde se muestra los procedimientos del área.
- Se listan los documentos usados en el área -- (órdenes, catálogos, bitácoras, etc.)
- Se detecta que documentos se usan en cada actividad del área.
- Se detectan los datos.
- Se asocian datos a las actividades.
- Se detectan las entidades asociadas a las actividades.
- Se detectan los atributos de cada entidad.

- Detección de la información recibida, necesaria - y producida por cada área de la organización.
  - Analizar cada función, identificando la información que recibe, de donde procede, frecuencia y volumen, y las entidades y atributos relacionados.
  - Analizar cada función, identificando la información que necesita, de donde procedería, frecuencia y volumen, y las entidades y atributos relacionados.
  - Analizar cada función identificando la información que produce, destino, frecuencia y volumen y las entidades y atributos relacionados.

- Obtención de un Diccionario de las Vistas.
  - Integración del Diccionario de Vistas-Entidades para cada vista (área estratégica, táctica, operativa, usuario o productor de información) se hace una lista de todas las entidades relacionadas.

  - Obtención de un Diccionario entidad-atributo. Para cada entidad se hace una lista de todos los atributos relacionados en el cual se indica para cada atributo lo siguiente:

    I) Nombre del atributo
    II) Tipo de manejo (recibe, necesita, produce)
    III) Con quién está relacionado (quién lo envía, a dónde se manda)
    IV) Descripción del atributo (comentario breve)
    V) Frecuencia de utilización
    VI) Frecuencia de recepción o envío
    VII) Volumen

INTEGRACION DE LA VISTA GLOBAL

Ya que se tiene cada una de las vistas se ve como integrarlas para que reflejen la estructura total de los datos en una -- vista global.

Este proceso consta de las siguientes actividades:

- Obtención de un Diccionario Preliminar de Clases de Entidades.

- Detectar prioridades de integración al Sistema Base de Datos.
- Obtención de las matrices atributos producidos-necesarios y atributos-entidades.
- Obtención de la vista global.

A continuación se presenta como se puede desarrollar cada una de las actividades.

- Obtención de un Diccionario Preliminar de Clases de Entidades.

  Es común que en las diferentes vistas se maneje la misma entidad (con el mismo o diferente nombre) por lo que en este punto, todas las entidades del mismo tipo se integran en una clase de entidad y se crea un diccionario donde a cada clase de entidad se le asocia lo siguiente:

  . Nombre consolidado
  . Sinónimos (nombres equivalentes)
  . Lista de vistas en las que se maneja la entidad.

  También se construye una matriz de clases de entidad contra vistas.

- Detección de prioridades de integración.

  En este punto se ve qué datos son indispensables, cuáles necesarios y cuáles deseables, y se establece una jerarquía en el orden de integración al Sistema Base de Datos, quedando definida la vista global sólo con aquello que realmente se implantará en el sistema.

  Para realizar este punto se proponen varios métodos:

I) Jerarquización Cualitativa.
Para cada clase de entidades se le pide a cada usuario que indique si lo considera indispensable, necesario o deseable.

II Contrastación del sistema actual contra datos que se manejan en la organización.
Permite ver que tanto de los requerimientos de información cubre el sistema actual de Base de Datos con el fin de detectar las principales áreas problema.
Se puede realizar construyendo una matriz de Datos Manejados Actualmente contra Datos necesarios.

III) Detección de Necesidades.
Se suspende el envío de información en la organización durante un período dado (por ejemplo 1 mes) y sólo se proporciona aquélla que se pide explícitamente, registrándose cada petición de información y sacándose las necesidades reales de la organización.

IV) Detección de Información Estable.
Se analiza la organización, viendo cuales son las áreas más estables (normalmente son áreas operativas) y se estudia su flujo de información, obteniendo cual es la información que fluye normalmente en la organización y que por tal motivo puede formar parte del núcleo de datos de la Base de Datos.

V) Análisis del Costo de Manejo de los Datos.

En este punto se analiza el costo de manejo de datos en cada función y por niveles (operativos, táctico, estratégico) dentro de la organización.

Los puntos principales del proceso son:

- Se ve el número de personas por cada función y nivel.
- Sueldo promedio por persona-función-nivel, - persona-función y persona-nivel.
- % tiempo gasta el personal por función, manejando información.
- Se calcula el costo de la información por función. (Costo Informac por función = % tiempo* Sueldo promedio persona-función*# de personas en la función).
- Se calcula el costo de información por nivel (costo información por nivel = % tiempo * Sueldo promedio en el nivel *# peronas en el nivel).
- Se calcula el costo de información por función-nivel.
- Se analizan los costos obtenidos, viendo - - donde incide con mayores costos el manejo de información y si se podrían disminuir, sistematizando el proceso de manejo.

VI) Análisis de atributos producidos-necesarios.

Se construye una matriz de atributos producidos contra necesarios y se estudian los siguientes puntos:

- Que atributos se necesitan y no se necesitan para quitarlos.
- Que atributos se necesitan y no se producen para prevenir problemas y en su caso buscar como obtenerlos.
- Otro punto a analizar es el de la dependencia entre atributos ya que es común que de un atributo se produzca un nuevo atributo - y que de ese nuevo atributo se produzcan a su vez otros atributos, en este caso se - puede simplificar haciendo que del atributo original se produzcan en un sólo paso todos los atributos necesarios.
- Que atributos producen sólo atributos que no se necesitan (islas), para quitarlos.
- Que atributos producen otros atributos que a su vez producen el atributo original (ciclos) para investigar el problema.

El análisis de atributos producidos-necesarios se puede realizar visualmente sobre la matriz o utilizando un conjunto de - herramientas de las Gramáticas Libres de Contexto (teoría de - Lenguajes y Autómatas) que nos dan métodos para detectar los - problemas planteados (detectar islas, ciclos, dependencias, no producción a partir del origen, etc.)

Integración de resultados

A partir de los diferentes métodos presentados anteriormente - se obtienen diferentes propuestas de los datos a integrar, por lo que, en este punto se realiza un concenso, buscando que - queden en el sistema, al menos:

- Los datos, entidades, y atributos indispensables.
- Los datos que se utilizan más.
- Los datos que representan mayor costo (directivos, - toma de decisiones, control, etc.)
- Los datos que menos cambian.

Con todo este universo se crea el Diccionario de Datos del Sistema Base de Datos.

- Obtención de las matrices atributos producidos-necesarios y atributos-entidad.
  A partir del Diccionario de Datos del Sistema Base de Datos.
  - Primero se construye la matriz de atributos producidos-necesarios, donde se ve la relación entre - los atributos del sistema.
  - A cada entidad detectada en el sistema se le asocia el conjunto de atributos relacionados con ella. Puede ser una lista de atributos para cada entidad y/o una matriz de atributos contra entidades.

- Obtención de la vista global.
  Para obtener la vista global, se construye un diagrama de entidades que junto con las matrices anteriores forman la vista global.

CONSTRUCCION DE DIAGRAMA DE ENTIDADES

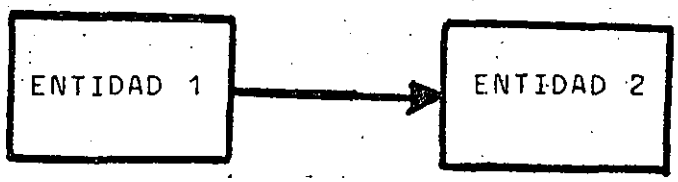
Este proceso parte de la detección de las entidades, luego obtiene las relaciones entre las entidades, para a continuación crear un diagrama donde se representan en forma gráfica las - entidades y sus relaciones.

A continuación se presenta el proceso.

a) Detección de las entidades.
   A partir de la matriz atributos-entidades de la vista global, se obtiene la lista de entidades.

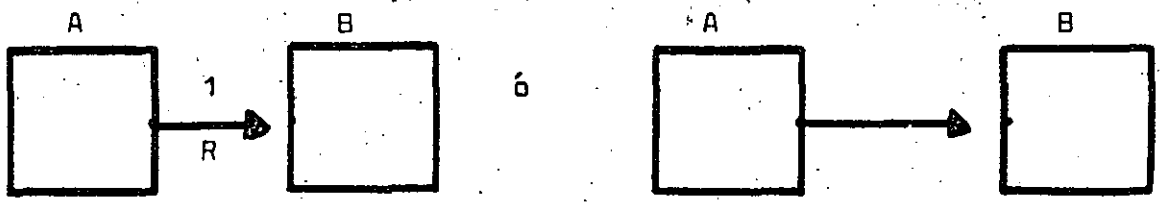
b) Obtención de las relaciones entre entidades.
   Se encuentran las relaciones que existen entre las diferentes entidades para lo cual se observan como están relacionadas en la matriz de datos producidos-necesarios.

c) Construcción del diagrama de entidades.
   En este punto cada entidad se asocia con un rectángulo y cada relación con una flecha etiquetada, y se construye un diagrama donde aparecen interconectados los rectángulos de tal forma que si una relación asocia a dos entidades, en el diagrama aparecen conectadas las relaciones mediante la flecha, como en la siguiente figura:



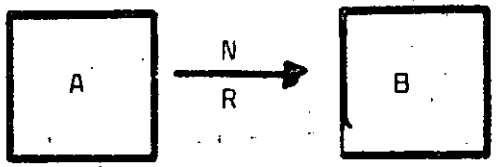
Normalmente las relaciones muestran una cierta jerarquía entre las diferentes entidades, ya sea relaciones verticales (entre padres e hijos) o relaciones horizontales entre entidades del mismo nivel (entre hermanos).

Cuando se relaciona una entidad con otra entidad se dice que es una relación 1:1 y se representa con:



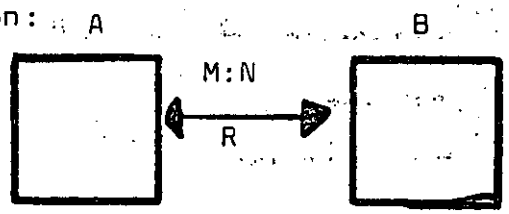
donde A y B son los tipos de entidades y R el nombre de la relación.

Si se relaciona una entidad con un conjunto de otras entidades se dice que es una interrelación 1:N y se representa con:



donde A y B son los tipos de entidades y R es el nombre de la relación.

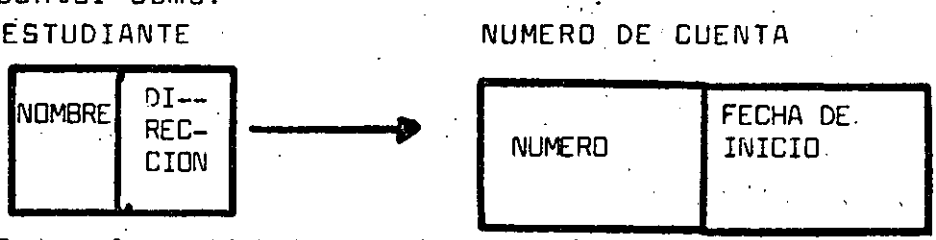
En general si se relaciona un conjunto de entidades con otro conjunto de entidades se tiene una relación M:N y se representa con:



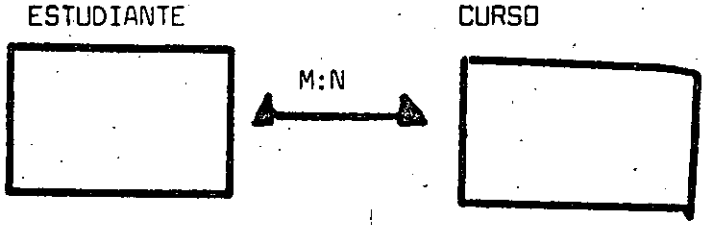
EJEMPLOS:

a) Entre la entidad estudiante y la entidad número de cuenta (boleta) existe una relación 1:1

La entidad estudiante puede constar por ejemplo de los atributos nombre y dirección y el número de cuenta puede constar de número y fecha de inicio, lo que se puede representar como:

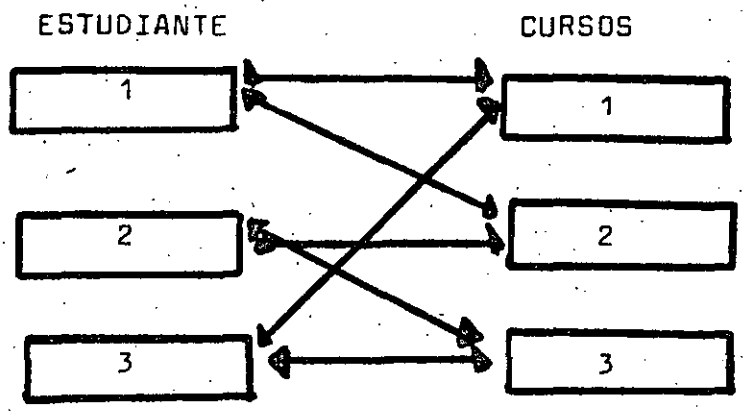


b) Entre la entidad estudiante y la entidad cursos existe una relación N:M



Por ejemplo si: - 68

Estudiante 1 toma cursos 1,2
Estudiante 2 toma cursos 2,3
Estudiante 3 toma cursos 1,3
la relación entre los valores de los atributos sería:



Con lo anterior la vista global queda formada por su universo de vistas particulares y generales (matriz atributos-entidades), la relación entre atributos (matriz atributos producidos-necesarios) y la relación ente entidades (diagrama de entidades).

EJERCICIOS

1.- Describa el medio ambiente de su organización incluyendo dos o tres organizaciones relacionadas.

2.- Indique que información intercambia la organización con cada una de las otras organizaciones.

3.- Mencione algunos elementos que conforman el subsistema estable de la organización.

4.- Para cada una de las siguientes organizaciones, indique cual es el mecanismo que utiliza para captar la información, como procesa internamente los datos y que tipo de acciones toma a partir de ellos.

   a) Un comercio
   b) Una central de emergencias
   c) Un gato
   d) Un país en relación con su comercio exterior

5.- En cada una de las organizaciones siguientes, indique el tipo de información que requiere, para qué la usa y qué genera.

   a) Un área de ventas
   b) Un almacén
   c) Un operador de un torno
   d) Un ratón

6.- Cuáles son las funciones del área de Administración de Datos en su organización.

7.- Describa la estructura funcional de su organización.

8.- Describa 3 tipos de entidades relacionadas con cada área funcional.

9.- Indique 3 atributos de cada tipo de entidad.

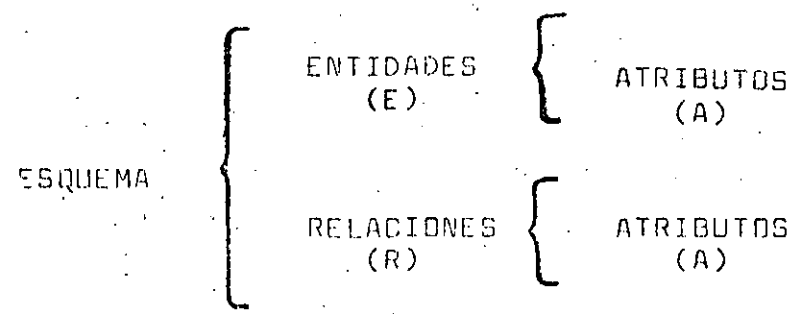
4.2 CONCRETIZACION DEL ESQUEMA LOGICO DE LA BASE DE DATOS

En esta etapa se toma la vista global de la Base de Datos y se construye un esquema lógico en el cual se encuentran representadas las entidades, atributos e interrelaciones relevantes para la organización en una forma completa y con redundancia mínima.

Esta etapa se compone de las fases de:

1.- Construcción del esquema relacional
2.- Normalización del esquema

La vista global de los datos de una organización se puede considerar compuesta por un conjunto de tipos de entidades relacionadas entre sí, los tipos de entidades se caracterizan mediante un conjunto de atributos y si se analiza la estructura se puede observar que las relaciones también se pueden caracterizar por medio de atributos, por lo que, podemos considerar al esquema de una Base de Datos compuesto por tipos de entidades, relaciones y sus atributos.



La idea base del enfoque relacional parte de que la Base de Datos está compuesta por un conjunto de tablas, donde cada tabla representa algún tipo de entidad o alguna relación. Para representar una tabla, se pone su nombre y entre paréntesis la lista de atributos.
NOMBRE - TABLA (atributo 1, ---, atributo n)

En las tablas relacionales (o archivos planos) cada una de sus columnas (o campos) representa un atributo y cada renglón de la tabla (o registro) corresponde a una entidad o relación específica.

Al conjunto de todas las tablas que representan a las entidades y relaciones de la vista global se llama ESQUEMA RELACIONAL.

4.2.1. CONSTRUCCION DEL ESQUEMA RELACIONAL

En este punto, a partir del diagrama de entidades se pasa a un esquema relacional mediante los siguientes pasos:

- Obtención de tabla relacional para cada tipo de entidad.
- Obtención de llaves.
- Obtención del esquema relacional.

A continuación se desarrolla cada paso:

1) Obtención de la tabla relacional para cada tipo de entidad.
   En este punto se asocia a cada tipo de entidad del diagrama de entidades la lista de sus atributos y se representa como una tabla relacional, quedando de la forma:
   Nombre Entidad (atributo 1, atributo 2, ----, atributo n)

2) Obtención de las llaves.

En una Base de Datos normalmente existen muchas entidades de un mismo tipo por lo que se requiere distinguir de alguna forma entre las diferentes entidades, para lo cual se utiliza lo que se conoce como LLAVE, la cual es un atributo (o conjunto de atributos) que permite identificar a la entidad.

Existen 2 clases de llaves:
LLAVES PRIMARIAS son aquéllas que identifican unívocamente un registro.
LLAVES SECUNDARIAS, estas llaves identifican todos aquellos registros que tienen una propiedad dada.

En general mediante la llave primaria se recupera un registro únicamente y mediante la llave secundaria se obtiene un conjunto de registros.

Las llaves primarias deben cumplir con 2 requisitos:

I) Identificación unívoca: Para un valor dado de la llave existe a lo más una entidad.
II) Llave mínima: Si se quita algún atributo de la llave, lo que queda ya no es llave.

Puede existir más de un conjunto de atributos que satisfacen las 2 condiciones anteriores a estos conjuntos se les llama llaves candidatas y de éstas elige una llave principal (o simplemente llave), por ejemplo se puede elegir la que involucre menos atributos. Esta se denota subrayándola en el esquema.

Cuando se construye el esquema se tiene idea de cual o cuales deberían de ser los atributos que forman la llave para cada entidad, sin embargo es común que la llave deseada no cumpla

con los requisitos anteriores, por lo que a continuación se presentan algunas herramientas para encontrar las llaves.

Una forma para localizar las llaves es mediante la detección de las dependencias funcionales.

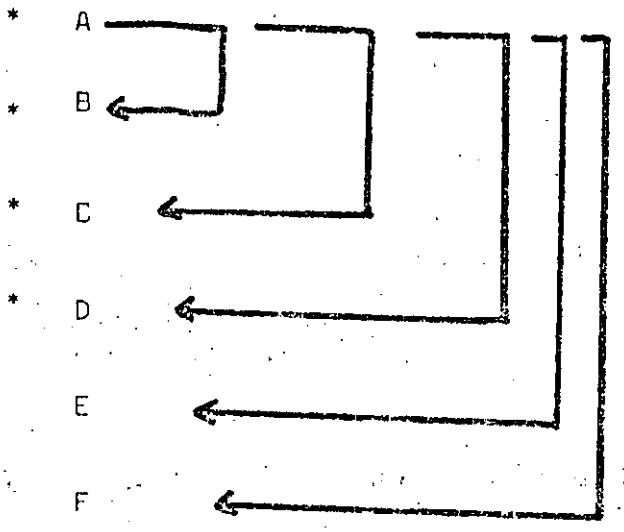
Se dice que un atributo B (o conjunto de atributos B) depende funcionalmente de otro atributo A (o conjunto de atributos A) si a cada valor que toma A corresponde un único valor de B.

Notación
Si B depende funcionalmente deA, se denota como A ⟶ B.
Al atributo A (o conjunto de atributos) se le llama DETERMINANTE.

Ejemplo:
La entidad X tiene los atributos A, B, C, D, E, donde

A ⟶ B
A ⟶ C
A ⟶ D
B ⟶ E
CD ⟶ F (F depende de los atributos C y D)

Estas dependencias se pueden representar también como sigue:



Donde A, B y CD son determinantes.

Si para un tipo de entidad existen los atributos A1, A2 ---, An, D y

D ⟶ A1 (A1 depende de D)
D ⟶ A2 (A2 depende de D)
D ⟶ An (An depende de D)

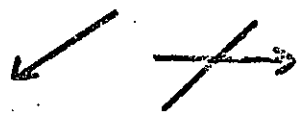
Esto se puede denotar como

D ⟶ A1 A2------ An

Si se tiene un tipo de entidad:
Entidad (L, A1, ------, An) donde
L, A1, ----, An son todos los atributos de ese tipo de entidad.

a) Todos los atributos dependen de L
(L ⟶ A1 A2 ----- An)

b) Todos los atributos dependen del TOTAL de L pero no de una parte (si L es una parte de L entonces


A1 A2 --- An)

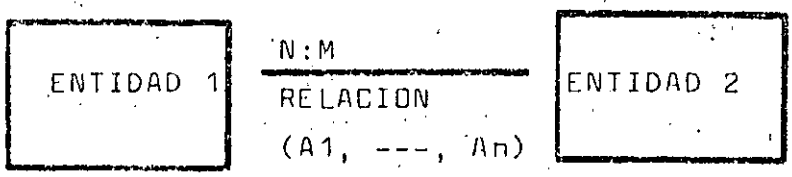
Entonces L es una LLAVE CANDIDATA
Cada uno de los atributos que forman parte de alguna llave candidata se conocen como ATRIBUTOS PRIMOS y los atributos que no corresponden a alguna llave candidata se llaman atributos No- Primos.

Obtención del esquema relacional.

I) Cada relación del diagrama de entidades se sustituye por una tabla donde sus atributos son las llaves de las entidades que se asocian mediante la relación más los atributos propios de la relación.

EJEMPLO

Si se tiene la relación



donde
Entidad 1 (llave 1, atributo 1, ---, atributo n)
Entidad 2 (llave 2, atributo m, ---, atributo 1)

Entonces se obtiene la nueva tabla
Relación (llave 1, llave 2, A, ---, An)

II) El conjunto de todas las tablas (las originales y las que se obtuvieron sustituyendo las relaciones) y sus atributos forman un esquema relacional.

4.2.2. NORMALIZACION DEL ESQUEMA.

Cuando se diseña una Base de Datos existen muchos posibles esquemas, de los cuales algunos son más convenientes que otros.

Los problemas surgen desde el momento que se están representando múltiples entidades y relaciones y entre los atributos se crean dependencias (por ejemplo la dependencia funcional), ocasionando que una modificación (borrado, inserción o actualización) realizada en un punto de la base llegue a afectar algún área que no se quería tocar.

Algunos de los principales problemas son:

a) Inserción anómala.
Este problema surge cuando al insertar el valor de un atributo es necesario insertar los valores de - otros atributos porque en el esquema son dependientes, aunque en la realidad no sea necesario.

b) Eliminación anómala.
En este caso al borrar un dato se borran otros datos dependientes aunque tal vez se necesitarán - después.

c) Actualización anómala.
Si un atributo aparece en forma no necesaria en varios puntos del esquema, al modificar alguno de sus valores, es necesario modificarlo en todos los puntos donde aparezca, con el costo y posibilidad de - error que esto implica el problema anterior y forma parte del problema de minimizar la REDUNDANCIA, que es uno de los objetivos del diseño lógico.

El proceso de NORMALIZACION permite disminuir las anteriores anomalías (aunque no es una panacea y en un momento dado no - resuelve todos los problemas) y busca reducir a su forma más simple el esquema lógico de la Base de Datos.

El proceso de normalización se basa en 2 operaciones básicas sobre las tablas relacionales conocidas como proyección y - unión, con las cuales se puede cambiar la estructura de los esquemas.

A continuación se explican estas operaciones:

a) Mediante la PROYECCION se pueden obtener nuevas tablas a partir de una tabla dada, extrayendo las columnas que se requieren y quitando los renglones repetidos.
EJEMPLO:
A partir de la tabla EMPLEADO (# EMP, NOMBRE, FUNCION, LUGAR) se quieren obtener una tabla que sólo contenga # EMP, NOMBRE y FUNCION Esto se denota como sigue:
FUN-EMP=$\overline{II}$ EMPLEADO (# EMP, NOMBRE, FUNCION)
donde FUN-EMP es el nombre de la nueva tabla, $\overline{II}$ indica la operación de proyección y de la tabla EMPLEADO sólo se tomarán # EMP, NOMBRE y FUNCION.

En el siguiente caso concreto se observa la operación.

EMPLEADO

# EMP	NOMBRE	FUNCION	LUGAR
1	PEDRO	TORNO	LOCAL A
1	PEDRO	CEPILLO	LOCAL B
3	JUAN	TORNO	LOCAL A
4	LUIS	TORNO	LOCAL A
4	LUIS	TORNO	LOCAL B
4	LUIS	CEPILLO	LOCAL B

EXTRACCION DE COLUMNAS

# EMP	NOMBRE	FUNCION
1	PEDRO	TORNO
1	PEDRO	CEPILLO
3	JUAN	TORNO
4	LUIS	TORNO
4	LUIS	TORNO
4	LUIS	CEPILLO

← RENGLONES REPETIDOS

↓ QUITAR REPETICIONES

FUN-EMP

# EMP	NOMBRE	FUNCION
1	PEDRO	TORNO
1	PEDRO	CEPILLO
3	JUAN	TORNO
4	LUIS	TORNO
4	LUIS	CEPILLO

b) Mediante la UNION NATURAL se toman 2 tablas relacionales y a partir de ellas se obtiene una nueva tabla relacional.

Para poder realizar la unión se requiere que existan dos atributos (o conjunto de atributos) que tengan los mismos posibles valores (que tengan el mismo dominio), y que uno de los atributos corresponda a una tabla y el otro a la otra tabla.

Entonces la unión se realiza haciendo que cuando en los dos atributos se tiene el mismo valor se concatenen los dos renglones:

EJEMPLO:

Dadas las tablas:

FUN-EMP (# EMP, NOMBRE, FUNCION) y

EMP-LUG (# EMP, LUGAR) podemos unir por ejemplo mediante el atributo # EMP (se repite en las dos tablas) la instrucción:

EMPLEADO 1 = FUN-EMP*EMP-LUG (# EMP) indica la unión de - - FUN-EMP y EMP-LUG a través del atributo # EMP.

En el siguiente caso concreto se observa la operación:

FUN-EMP

# EMP	NOMBRE	FUNCION
1	PEDRO	TORNO
1	PEDRO	CEPILLO
3	JUAN	TORNO
4	LUIS	TORNO
4	LUIS	CEPILLO

EMP-LUG

# EMP	LUGAR
1	LOCAL A
1	LOCAL B
3	LOCAL A
4	LOCAL A
4	LOCAL B

EMPLEADO 1

# EMP	NOMBRE	FUNCION	LUGAR
1	PEDRO	TORNO	LOCAL A
1	PEDRO	TORNO	LOCAL B
1	PEDRO	CEPILLO	LOCAL A
1	PEDRO	CEPILLO	LOCAL B
3	JUAN	TORNO	LOCAL A
4	LUIS	TORNO	LOCAL A
4	LUIS	TORNO	LOCAL B
4	LUIS	CEPILLO	LOCAL A
4	LUIS	CEPILLO	LOCAL A

Las tablas FUN-EMP y EMP-LUG se obtuvieron mediante proyección sobre la tabla EMPLEADO sin embargo al unirse se obtuvo la tabla EMPLEADO 1 que tiene un conjunto de entidades diferentes a EMPLEADO, o en otras palabras al proyectar sobre una tabla que representa una cierta realidad y unir las proyecciones no necesariamente se conserva la información.

Dado que la proyección es una operación muy usada en la normalización y por otro lado las tablas resultantes de la normalización deben conservar la información original se verá un criterio para decidir si se puede proyectar a partir de una tabla conservando la información.

Una tabla es IRREDUCIBLE si no se puede proyectar en un conjunto de tablas con menos columnas, tales que, esas tablas al unirse formen la tabla original (VETTER 81)

Una tabla irreducible se llama TABLA ELEMENTAL. Si una tabla es elemental quiere decir que al proyectarse en nuevas tablas no necesariamente se conserva la información, por lo que, durante el proceso de normalización siempre se buscará que al proyectar una tabla, las tablas resultantes al unirse den la tabla original.

El proceso de NORMALIZACION es un proceso que consta de 6 pasos en cada uno de los cuales se obtienen las tablas en una nueva forma normal:

- Obtención del esquema en 1. forma normal (1.-FN)
- Obtención del esquema en 2. forma normal (2.-FN)
- Obtención del esquema en 3. forma normal (3.-FN)
- Obtención del esquema en la forma normal de Boyce-Codd (BCFN)
- Obtención del esquema en la 4. forma normal (4.-FN)

- Obtención del esquema en la 5.- Forma normal (5.-FN)

La normalización se aplica a CADA UNA de las tablas de un esquema y cuando TODAS las tablas están en una forma normal entonces el esquema está en esa forma normal.

Una tabla T se NORMALIZA OPTIMAMENTE si se puede proyectar en un conjunto de tablas T1, T2, ---,Tn tales que cada tabla Ti (i=1, ---, n) está normalizada y la unión de T1, T2, ---, Tn es igual a T. También se dice que T1, T2, ---, Tn están en una forma normal óptima.

Si TODAS las tablas de un esquema están en una forma normal óptima entonces el esquema está en forma normal óptima.

A continuación se presenta cada uno de los pasos del proceso de normalización:

1) Obtención del esquema en 1.- Forma normal.
   La normalización en su primera etapa busca que en cada tabla no existan 2 registros con la misma información, que no exista un atributo con un valor constante en todos sus registros y que no existan grupos de repetición (conjuntos de atributos que se repiten para la misma entidad).
   Un esquema que cumple con lo anterior se dice que está en 1.- forma normal (1.-FN) y cada tabla se dice que es una tabla atómica.

   Para pasar un esquema a un esquema en 1 FN se siguen los siguientes pasos:

I) Si se tienen 2 registros con la misma información, se quita uno.

II) Si se tiene un atributo con valor constante se quita y se almacena en forma independiente.

III) Si se tienen grupos de repetición se proyecta la llave del registro y los atributos del grupo de repetición.

2.- Obtención del esquema en 2.- forma normal.

Se dice que un atributo B (o colección de atributos B) es DEPENDIENTE FUNCIONAL COMPLETO de otro atributo A (o colección de atributos A) si B es funcionalmente dependiente del total de A pero no de ninguna subparte de A.

Una tabla está en 2o. FORMA NORMAL si está en 1.-FN y cualquier atributo NO PRIMO es dependiente funcional completo de una llave candidata.

Dada una tabla que está en 1.-FN pero no en 2.-FN se puede pasar a 2.-FN, viendo cual es el conjunto de atributos primos de los cuales depende el atributo problema y se proyectaría por un lado la tabla sin el atributo problema y por el otro una tabla con el atributo problema y los atributos de los cuales depende.

EJEMPLO

Si se tiene la tabla.

TABLA (A, B, C, D, E)

donde AB es la llave candidata, A y B son atributos primos; C, D, E, son atributos no primos.

Y si las dependencias funcionales son:

AB $\longrightarrow$ C

AB $\longrightarrow$ D

A $\longrightarrow$ E

Entonces C y D dependen completamente de la llave candidata AB pero E NO depende completamente de AB (depende sólo de A) por lo tanto TABLA no está en 2.-FN.
Para pasar TABLA a 2.-FN se observa que el atributo problema es E y que depende de A por lo que se obtendrán las nuevas - tablas.
TABLA 1 (A, B, C, D,)
TABLA (A2 (A, E)
Donde AB $\longrightarrow$ C
AB $\longrightarrow$ D

Donde A $\longrightarrow$ E
y existe dependencia funcional completa El proceso se repite para todas las tablas del esquema.

3.- Obtención del esquema en 3.-Forma Normal.
Un atributo C (o conjunto de atributos) depende transitivamente de un atributo A, si:
A $\longrightarrow$ B y B $\longrightarrow$ C y además
B $\neq$ A, B $\not\longrightarrow$ A ó C $\not\longrightarrow$ B para algún B

Una tabla está en 3.-FN, si está en 2.-FN y ningún atributo depende transitivamente de otro.

Si algún atributo depende transitivamente de otro nuevamente se ejecuta una proyección.
Si A $\longrightarrow$ B, B $\longrightarrow$ C entonces se obtienen 2 tablas, una donde estén A y B y la otra con B y C.
EJEMPLO:
Si se tiene la tabla
TABLA (A, B, C, D, E,)
y las relaciones son

A ⟶ B
A ⟶ C
A ⟶ D
B ⟶ E

Entonces se tiene que
A ⟶ B, B ⟶ E y
B ⟶ A, B ⟶ A, C ⟶ B
Por lo que E depende transitivamente de A.
Entonces la TABLA se proyecta en TABLA 1 (A, B, C, D)
donde:
A ⟶ B
A ⟶ C
A ⟶ D
TABLA 2 (A, E)
donde
A ⟶ E

4.- Obtención del esquema en la forma normal de Boyce Codd.
Se dice que una tabla está en la FORMA NORMAL DE BOYCE-CODD si CUALQUIER DETERMINANTE es una llave candidata entonces si el esquema no está en la BCFN entonces quiere decir que existen tablas que tienen determinantes que no son llaves candidatas, entonces cada tabla se normaliza.
Para pasar una tabla a la BCFN se tiene que encontrar el determinante problema (que no es llave) y los atributos que dependen funcionalmente de él y entonces por un lado se proyecta una tabla sin los atributos que dependen del determinante problema y por el otro se proyecta otra tabla con el determinante problema y sus atributos dependientes.

EJEMPLO

Si se tiene la tabla

TABLA (A, B, C, D, E)

con las relaciones

AB $\longrightarrow$ C

AB $\longrightarrow$ D

AB $\longrightarrow$ E

B $\longrightarrow$ E

Entonces AB es una llave, B es un determinante pero no llave, por lo que, se obtienen 2 tablas.

TABLA 1 (A, B, C, D) y TABLA 2 (B, E)

5.- Esquema en 4.-Forma Normal.

Se dice que un atributo A (conjunto de atributos A) - - MULTIDETERMINA a un atributo B (conjunto de atributos B) si para cada valor de A se obtiene un conjunto de posibles valores de B. Esto se representa como A $\longrightarrow\!\!\!\!\!\rightarrow$ B y también se dice que existe una DEPENDENCIA MULTIVALUADA de B en A.

Una tabla relacional está en 4.- forma normal si para cualquier dependencia multivaluada A $\longrightarrow\!\!\!\!\!\rightarrow$ B entonces A es una llave candidata, o sea que todos los atributos dependen (funcionalmente o multivaluadamente) de A.

Si una tabla no está en 4FN quiere decir que dado que A $\longrightarrow\!\!\!\!\!\rightarrow$ B existe algún atributo C tal que A $\not\longrightarrow$ C En este caso se proyecta buscando que desaparezca el - problema.

EJEMPLO

Si se tiene

TABLA (A, B, C) donde

A $\longrightarrow\!\!\!\!\!\rightarrow$ B

B $\longrightarrow\!\!\!\!\!\rightarrow$ C

Entonces se puede proyectar en las tablas
TABLA 1 (A, B)
TABLA 2 (B, C)

6.- Esquema en 5 forma normal.

Recordamos que una tabla se normaliza óptimamente si se puede proyectar en un conjunto de tablas, tales que, al unirse se vuelve a obtener T; se dice que el conjunto de tablas tienen una DEPENDENCIA DE UNION entre sí.

Un esquema está en 5o. forma normal, si cualquier dependencia de unión es implicada por las llaves candidatas de R.

SPP
programación y presupuesto
INSTITUTO NACIONAL DE ESTADISTICA
GEOGRAFIA E INFORMATICA

SISTEMAS MANEJADORES DE BASES DE DATOS

Lic. Vicente López Trueba
Subdirector de Desarrollo de Bases de Datos

COMO componentes del Sistema Nacional de Planeación Democrática cuya consolidación se ha propuesto la presente administración se contempló la necesidad de disponer de una infraestructura de servicios, que apoyen las actividades del proceso de planeación.

Como parte de esta infraestructura los Sistemas Nacionales de Estadística y de Información Geográfica, tienen el objetivo de proporcionar los insumos básicos de información que comprenden el conjunto de datos ordenados, cuantificables o no, que sirven para dar sustento a la toma de decisiones, tanto en el operativo como administrativo de planeación.

Los sistemas tienen el propósito de dar homogeneidad a los procedimientos de captación de datos emitidos por diversas fuentes de información estadística y geográfica, para lo que se ha facultado a la Secretaría de Programación y Presupuesto a través del Instituto Nacional de Estadística, Geografía e Informática, cumplir con estas tareas.

El Instituto para el desarrollo de sus funciones se organiza en cuatro direcciones generales: la Dirección General de Geografía, la Dirección General de Estadística, la Dirección General de Integración y Análisis de la Información y la Dirección General de Política Informática.

Dentro de la Dirección General de Política Informática, se encuentran, atendiendo a los requerimientos de diseño y desarrollo de sistemas para el procesamiento de información del Instituto, la Dirección de Captura y Proceso de la Información y la Dirección de Desarrollo Informático.

La primera como administradora del equipo de cómputo y la Dirección de Desarrollo Informático como responsable del desarrollo de sistemas automatizados y servicios de consulta de bases de datos.

La Dirección de Desarrollo Informático cuenta con cinco subdirecciones:

- Subdirección de Desarrollo de Bases de Datos.
- Subdirección de Desarrollo de Sistemas de encuestas, Registros Administrativos y Sociodemográficos.
- Subdirección de Análisis e Intercambio de

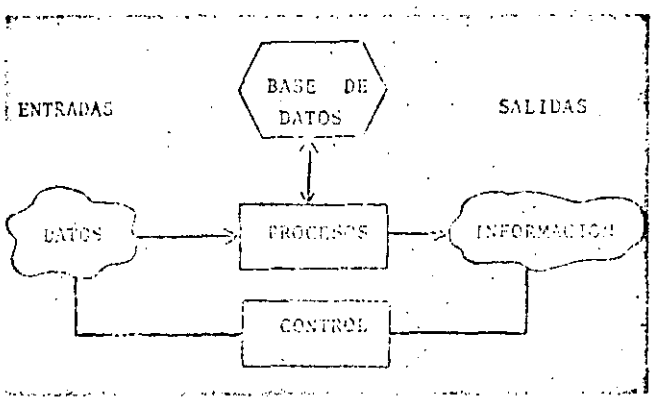
Sistemas.
- Proyecto de Censos Económicos.
- Proyecto de Censos Agropecuarios.

La Subdirección de Desarrollo de Bases de Datos tiene como objetivo:

COORDINAR EL DISEÑO, DESARROLLO, ADMINISTRACION E IMPLEMENTACION DE LOS SISTEMAS MANEJADORES DE BASES DE DATOS, QUE APOYEN CON OPORTUNIDAD Y CONSISTENCIA A LOS SISTEMAS NACIONALES DE ESTADISTICA E INFORMACION GEOGRAFICA.

La consecución de nuestro objetivo nos conduce a identificar las demandas de procesamiento de información representadas principalmente por los sectores:

- *Económico*
- *Industrial*
- *Agropecuario*
- *Salud*
- *Educativo*
- *Comercio exterior*
- *Población*
- *Ocupación*



El Lic. Vicente López Trueba es egresado de UPIICSA en la especialidad de Informática. Ha asistido a muy diversos cursos sobre Logical y ha sido, igualmente, profesor de diferentes materias sobre el mismo tópico.

Su experiencia práctica se centra en el diseño y desarrollo de Sistemas Informáticos.

Actualmente es Subdirector de Desarrollo de Bases de Datos en la Dirección General de Política Informática del INEGI.

Ante la magnitud de la información y su diversidad, nuestros sistemas requieren contar con una estructura tecnológica óptima:

La que permita proveer la interrelación de los fenómenos económicos, demográficos y sociales en su conexión con el espacio territorial.

Consideramos a los sistemas manejadores de bases de datos como uno de los instrumentos fundamentales para lograr el objetivo de proporcionar información con oportunidad y articulación.

CONCEPTOS DE SISTEMAS MANEJADORES DE BASES DE DATOS

Con el objetivo de desarrollar el tema *"Sistemas Manejadores de Bases de Datos"*, me permito hacer algunas definiciones que deseo sirvan como un puente de comunicación en relación a este tipo de sistemas.

Los *Sistemas de Información* actuales contienen como parte importante adicional al modelo general que incluye entradas, procesos, salidas y control, su base de datos.

Las entradas son datos de los hechos del mundo real. Los procesos transforman los datos en información y finalmente el control se establece para mejorar la información mediante la demanda de una calidad adecuada.

El papel de la base de datos en un sistema de información, pudiera considerarse en primera instancia como la memoria que sirve para recordar el pasado, registrar el presente y pronosticar el futuro, por lo tanto la base de datos constituye una de las partes sustantivas que forman un sistema de información.

Ahora bien, cuando un sistema de información se auxilia de un computador, se denomina sistema informático. Este contiene básicamente lo siguiente:

a) Reglas de utilización o procedimientos
b) El computador
c) La programación o software que lo hace funcionar
d) Los archivos organizados que forman la base de datos.

Es importante destacar que para garantizar la organización de la base de datos, es necesario tener un conjunto de programas de cómputo que facilite no solo su organización sino también su manejo y otros beneficios que mencionaré más adelante:

Los programas que manejan las bases de datos, se denominan *Sistemas Manejadores de Bases de Datos*. Este tipo de sistemas tiene una disciplina que permite que un usuario accese por sí mismo mediante un video la información y pueda operarla lógica o matemáticamente.

Los sistemas manejadores de bases de datos proporcionan ayudas en cinco niveles fundamentales, mismos que describiré a continuación.

a) Nivel de usuario o externo.

En el que mediante un lenguaje sencillo, de preferencia carcano al natural o por selección, el usuario puede alimentar la base de datos y obtener los reportes y consultas que proporcionan información.

b) Nivel de definición o conceptual

En donde el sistema "aprende" como organizar la información.

c) Nivel de manipulación o interno

Maneja la información en función de su organización.

d) Nivel físico de almacenamiento de los datos.

En donde se encuentran los archivos magnéticos de tal manera que garanticen:

d.a) Eficiencia en el almacenamiento
d.b) Eficiencia en el acceso
d.c) El respaldo de la información

e) Nivel administrativo

Que garantiza la administración eficaz de los siguientes recursos:

e.a) Los datos que existen en el sistema
e.b) La disponibilidad de recursos de cómputo adecuados para que el sistema funcione.
e.c) Las rutas de acceso eficientes en el nivel físico.
e.d) La aplicación adecuada del sistema.
e.e) Y las autorizaciones para el acceso.

Los beneficios que reportan los sistemas manejadores de bases de datos son:

- Disponibilidad inmediata de la información
- Reducción de la duplicación de datos.
- Optimización del almacenamiento
- Obtención de datos derivados mediante operaciones sencillas
- Control centralizado de la información, con positilidad de distribuirla en medios magnéticos.
- Accesibilidad a la información en base a los niveles de confidencialidad.
- Mecanismos de protección contra daños a la información o acceso indebidos

- Reducción de la actividad en las tareas de programación

Finalmente es importante mencionar que los sistemas manejadores de bases de datos requieren de usuarios que alimenten y exploten la información, de lo contrario sería como tener una mansión y no habitarla.

APLICACIONES ACTUALES

A continuación describiré los sistemas manejadores de bases de datos con que contamos en nuestra Subdirección, en función de sus características y aplicaciones.

SISTEMA RECUPERADOR DE INFORMACION

CARACTERISTICAS:

- Maneja información documental, calificada con palabras clave.
- Proporciona consultas visuales y reportes impresos.
- Facilidad para aceptar diversas bases de datos.

APLICACIONES:

- Base de datos hemerográfica
- Base de datos de sistemas de la APF
- Base de datos de información que produce la S.P.P.
- Sociodemográfica
- De publicaciones informáticas a nivel internacional.

SISTEMA GEOMUNICIPAL DE INFORMACION

CARACTERISTICAS:

- Maneja información resumida de variables sociodemográficas y económicas; en base a la estructura geopolítica de nuestro país.

- Proporciona consultas visuales y reportes impresos.
- Facilita operaciones algebraicas y estadísticas básicas.
- Permite formar regiones.

APLICACIONES:

Integración de variables de:

- Censos
- Encuestas
- Registros administrativos
- Sociodemográficas.

SISTEMA MANEJADOR DE SERIES DE TIEMPO

CARACTERISTICAS:

- Maneja la información formando jerarquías, como una organización.
- Facilita la posibilidad de efectuar cálculos algebráicos y estadísticas básicas.
- Proporciona consultas visuales y reportes impresos.

APLICACIONES:

- Base de datos de cuentas nacionales
  Cuenta con un esquema que contiene:
- 75 ramas de actividades (sectores)
- 16 secciones que describen cuales son los principales conceptos, y
- Una serie de elementos que incluye información de las series 1970 a 1984.

SISTEMA MANEJADOR DE CUADROS ESTADISTICOS

CARACTERISTICAS:

- Manejo de matrices de datos con información estadística.
- Reportes impresos por selección de cuadro
- Formación de subconjuntos de datos relacionados por renglones y columnas.
- Edición por Compugraphic (máquina de foto-composición).

APLICACIONES:

- Anuarios estadísticos
- Boletines.

ACTIVIDADES DE LA SUBDIRECCION DE DESARROLLO DE BASES DE DATOS

La Administración moderna incluye hoy entre sus fases el concepto de innovación, sencillez y racionalización. Esto implica en el ambiente informático un reto interesante para posibilitar el aprovechamiento de nuestras capacidades y avanzar al lugar donde deseamos estar y por lo tanto aprender a ser.

Para lograr lo anterior es necesario reconocer qué problemas tenemos e idear un modelo que cubra la cantidad necesaria y suficiente de instrumentos que den solucion a las demandas existentes.

Dicho modelo lo realizamos mediante las siguientes actividades:

a) Actualizar los sistemas.
b) Adquirir un sistema manejador para bases de datos estadísticos.
c) Desarrollar sistemas manejadores de bases de datos.

A continuación describiré las tareas de cada punto:

a) Actualizar los sistemas

Esto significa aceptar que los usuarios de los sistemas existentes se encuentran satisfechos de los servicios que proporcionan en cuanto al contenido que guardan.

Este planteamiento nos conduce a renovar los sistemas, mejorando sus lenguajes de consulta haciéndolos amigables a los usuarios.

b) Adquirir un sistema manejador de bases de datos.

Buscando alternativas de solución para el manejo masivo del Sistema de Estadística Nacional, encontramos que en Canadá, desarrollan un sistema denominado RAPID, el cual contempla las siguientes características:

- Eficiencia en el manejo de millones de registros.

En "Statistics Canada" se reporta que RAPID se uso para el manejo de 8 millones de registros de vivienda y 23 millones de habitantes, (cifra menor a la que requerimos).

- Eficiencia en almacenamiento (por la comprensión de datos.)
- Eficiencia en accesos (debido a que los datos se encuentran en archivos separados).
- Lenguaje conversacional de tipo relacional; para operaciones con la información.

Estas características hacen a éste sistema RAPID, un posible instrumento para manejar la estadística de nuestro país mediante muy grandes bases de datos, razón por la cual hemos establecido los contactos para obtener la documentación y los sistemas correspondientes.

c) Desarrollo de sistemas manejadores de bases de datos.

Mencionaremos que en opinión de muchos expertos, el problema de los bancos de datos reside en mantenerlos alimentados.

En verdad que la alimentación no es cosa sencilla. Imaginen ustedes que a través del tiempo se ha procesado tal cantidad de información proveniente de censos y encuestas factibles de ser alimentadas en bases de datos, que llegar a formular su coordinación y estructuración sería una larga tarea con personal y presupuestos suficientes.

Es por esto que dentro de nuestras actividades estamos desarrollando un sistema manejador de bases de datos que, a partir de la información actual, tenga como objetivo manejar:

1. Formatos diferentes, residentes en un diccionario de datos.

2. Mediante lógica preconstruida generar programas que obtengan:

2.1 Subconjuntos de información
2.2 Selección de información
2.3 Integración de variables
2.4 Mezclas con otros archivos
2.5 Manejo de diversos formatos.

3. Relacionar información con paquetes de análisis estadístico (SPPS, TSP).

4. Relacionar información con paquetes de generación de reportes.

Con éste sistema estaremos en posibilidad de contar con información no solo en publicaciones, sino de proporcionar cintas, microfichas, cassets y discos flexibles a los demandantes, respetando en todos los casos los principios de confidencialidad estadística.

En la actualidad contamos con un generador de programas que facilita esta tarea, y que de hecho ha sido de gran utilidad para la generación de sistemas censales y de encuestas.

Es importante destacar que en el desarrollo de dichos sitemas, hemos implementado metodologías de software y logical de cuarta generación, que implican fundamentalmente el manejo de una ingeniería basada en técnicas estructuradas modulares.

El generador de programas nos ha facilitado la integración de variables de sistemas censales, información que a su vez puede servir de entrada a alguno de los sistemas manejadores de bases de datos existentes.

De esta manera procederemos a realizar una alimentación automática de información final de encuestas y censos que se vayan produciendo o bien de las que ya estén terminadas.

PROBLEMAS DE INTEGRACION DE BASES DE DATOS

El éxito de las bases de datos, no se logra tan solo con los sistemas manejadores de bases de datos, sino que se requiere combatir dos problemas básicos en cuanto a las organizaciones.

a) Síndrome de la organización geométrica.

b) Síndrome de la confidencialidad.

El primer punto se refiere a la dificultad de alcanzar decisiones para integrar las bases de datos.

Este problema se debe a la cantidad geométrica de relaciones de comunicación que se presentan entre los grupos de individuos.

El segundo punto se refiere a la confidencialidad, en donde las áreas se muestran excépticas acerca de la información que proporcionan al sistema, dudando de la forma en que será utilizada y mostrando resistencia a que sus datos sean conocidos por otros, aduciendo confidencialidad.

METODOLOGIAS

Para reducir el efecto de los problemas antes mencionados, nuestra subdirección ha introducido dos metodologías que facilitan la integración y consulta de las bases de datos.

METODOLOGIA DE INTEGRACION PARA BASES DE DATOS

- Requerimentos de necesidades de las partes usuarias.
- Diseño lógico
- Propuesta de solución con los instrumentos disponibles.
- Implementación de la solución.
- Evaluación del funcionamiento de la base de datos.

Esto significa lo siguiente:

REQUERIMIENTO DE NECESIDADES pretende dar una formalidad al proceso de integración, este consiste en un documento donde se indican los principales productos que se desean obtener, así como la frecuencia de utilización, el impacto económico y por lo tanto los recursos humanos y de equipo (hardware) que son requeridos.

EL DISEÑO LOGICO, muestra conceptualmente cómo se realizaría la identificación de entidades de datos, atributos, relaciones, los procesos a efectuar, así como los responsables y los compromisos de disponibilidad del sistema.

LA PROPUESTA DE SOLUCION, implica que el usuario apruebe el proceso de implantación, facilitando los recursos necesarios para la alimentación del sistema y su adecuado uso, mediante el conocimiento de las reglas que impone el sistema de bases de datos.

IMPLEMENTACION DE LA SOLUCION, requiere de la elaboración del conjunto de acciones que permitan disponer de la base de datos para consulta, esto implica que el usuario ponga a disposición de esta operación los recursos preparados y adecuados para lograr esta fase.

EVALUACION DEL FUNCIONAMIENTO, proporciona un control de calidad para garantizar que el sistema realiza las operaciones con los recursos y tiempos correctos.

La metodología de integración ayuda a planear y controlar el uso adecuado de los recursos con los que cuenta la Subdirección de Desarrollo de Bases de Datos y facilita la comunicación con la Dirección de Proceso y Captura de la Información, para poder calendarizar los compromisos con el usuario, de acuerdo a los recursos de cómputo de que disponen.

METODOLOGIA DE CONSULTA DE BASES DE DATOS

- Requerimientos de consulta
- Capacitación
- Control de consultas
- Evaluaciones de consultas

LA METODOLOGIA DE CONSULTA DE BASES DE DATOS, tiene una relación con aquéllos usuarios que desean tener acceso a la información existente en bases de datos, el apoyo más importante que proporciona la Subdirección de Desarrollo de Bases de Datos, con-

siste en proporcionar la capacitación para el uso correcto de los sistemas.

El mérito más importante de las metodologías consiste en coordinar, las partes involucradas en la composición de un sistema automatizado de información, siendo los usuarios de la información, productores y personal técnico.

CONCLUSIONES

EL AVANCE TECNOLOGICO DE LA INFORMATICA PERMITE QUE LA INFORMACION SEA DISTRIBUIDA A LA MAYORIA DE LOS SECTORES QUE PLANEAN EL DESARROLLO DE LA VIDA NACIONAL.

Consideramos nuestra obligación como mexicanos insertarnos en ese avance con independencia y oportunidad.

Para ello requerimos adquirir el conocimiento y la experiencia que nos permita desarrollar la tecnología necesaria para desarrollarnos como nación.

Los conceptos anteriores, hacen que nuestra responsabilidad por mejorar y fomentar la tecnología informática mexicana aumente.

Esto desde nuestra perspectiva nos obliga a desarrollar instrumentos que permitan atender las demandas de procesamiento de la información, haciendo que los usuarios sean parte activa de ese proceso.

Finalmente quisiera decir que no es posible ignorar que el desarrollo de la tecnología tiene un costo.

Considero que regirnos únicamente por el menor costo sin evaluar los parámetros de costo-beneficio estratégico puede llevarnos, inevitablemente a la pérdida de nuestra identidad nacional.

informática

30

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA COMPUTACION ELECTRONICA Y PROGRAMACION

ORIGENES DE LAS COMPUTADORAS

ING. BENITO ZYCHLINSKI

OCTUBRE, 1984

ORIGENES DE LAS COMPUTADORAS

SIGLO VIII A.C.		ABACO
SIGLO XVI	NAPIER	LOGARITMOS
SIGLO XVII	BLAS PASCAL	MAQUINA DE CALCULAR
SIGLO XVIII	JOSEPH M. JACQUARD	TARJETA PERFORADA
SIGLO XIX	CHARLES BABBAGE	MAQUINA DIFERENCIAL
1900	HERMAN HOLLERITH	TARJETAS PERFORADAS
1941	KONRAD ZUSE	ZUSE-23
1944	AIKEN	COMPUTADOR HARVARD MARK I
1946	J. ECKERT-J. MAUCHLY	ENIAC (UNIVERSIDAD DE PENNSYLVANIA)
1947	J. VON NEUMANN	EDVAC

PRIMERA GENERACION

CARACTERISTICAS:

TUBOS DE VACIO

MEMORIA DE MERCURIO

ALMACENAMIENTO: ELECTROSTATICO

TAMBOR MAGNETICO

1951 UNIVAC I

1953 MEMORIA DE NUCLEO MAGNETICO

1953 IBM 701

SEGUNDA GENERACION

CARACTERISTICAS:

- TRANSISTORES
- MEMORIA DE NUCLEO MAGNETICO
- DISCOS
- CINTAS
- LENGUAJES DE ALTO NIVEL
- TAMAÑO MAS REDUCIDO

BURROUGHS SERIE 5000

PHILCO 212

UNIVAC M460, 1107

C D C 1604 Y 3000

I B M 709, 7090, 7094

N C R 315

R C A 501, 601

TERCERA GENERACION

CARACTERISTICAS:

CIRCUITOS INTEGRADOS

BYTES DE OCHO BITS

MINICOMPUTADORAS

1964 IBM 360, 370

UNIVAC 9000

C D C 6600, 7600, CYBER

D E C P D P

CUARTA GENERACION

CARACTERISTICAS:

INTEGRACION DE CIRCUITOS A GRAN ESCALA

MICRO-COMPUTADORAS

COMPUTADORAS PERSONALES

PASCAL

QUINTA GENERACION

CARACTERISTICAS:

CIRCUITOS MAS PEQUEÑOS Y VELOCES

VLS Y ULS

PANORAMA FUTURO:

AVANCES EN LOS COMPONENTES DE LAS COMPUTADORAS:

MEMORIAS (BURBUJAS MAGNETICAS)

NUEVAS ARQUITECTURAS

DISKETTES CON 20 MB

PROCESADORES DE ARREGLOS 10 A 100 VECES MAS RAPIDAS

SE ALCANZARAN LOS 100 MIL MILLONES DE OPERACIONES DE PUNTO FLOTANTE

ARQUITECTURA:

MULTIPROCESO

FLUJO DE DATOS

SISTEMAS NUMERICOS

1 2 3

100 + 20 + 3

$1 \times 100 + 2 \times 10 + 3 \times 1$

$1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$

SISTEMA NUMERICO BINARIO

1 0 1

$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$1 \times 4 + 0 \times 2 + 1 \times 1 = 5$$

DECIMAL	BINARIO
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

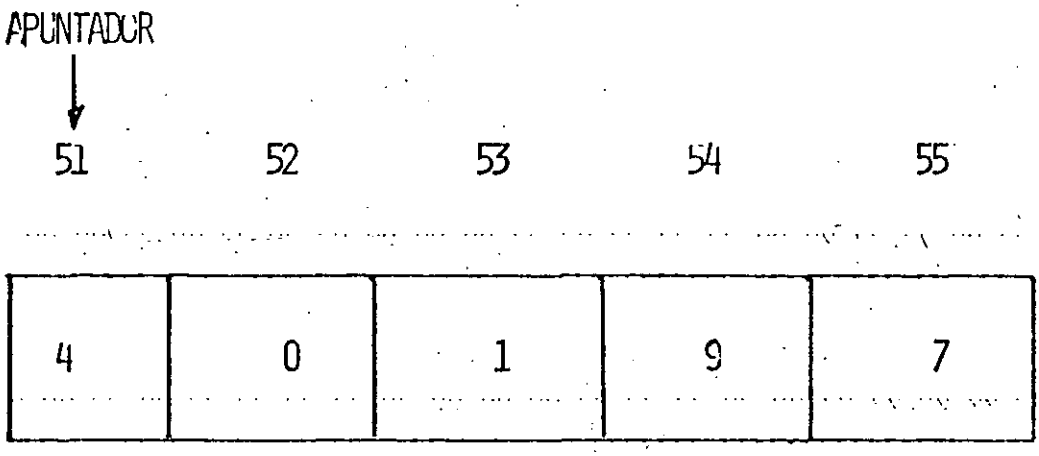
DECIMAL	OCTAL	BINARIO
0	0	000
1	1	001
2	2	001
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111

DECIMAL	HEXADECIMAL	BINARIO
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

MEMORIA

CONJUNTO DE CELDAS.

- CADA CELDA PUEDE CONTENER UN VALOR NUMERICO.
- CADA CELDA TIENE LA PROPIEDAD DE SER DIRECCIONABLE



PARA LEER:

- DECIDIR QUE CASILLA SE VA A LEER
- ESPERAR UN TIEMPO FIJO PARA TENER UNA COPIA DEL DATO
- RECOGER ESE DATO Y DAR POR TERMINADA LA OPERACION DE LECTURA.

PARA ESCRIBIR:

- PROPORCIONAR EL DATO QUE DESEAMOS SEA DEPOSITADO EN UNA CASILLA
- PROPORCIONAR LA DIRECCION DE LA CASILLA EN LA QUE SE VA A ESCRIBIR EL DATO
- ESPERAR UN TIEMPO FIJO PARA QUE DATO SE ESCRIBA EN LA CASILLA DETERMINADA Y DAR POR TERMINADA LA OPERACION DE ESCRITURA

EJEMPLO:
EN UNA CALCULADORA
SUMAR 5+7 Y OBTENER
EL RESULTADO.

1. PRESIONAR LA TECLA "5", LA CALCULADORA GUARDA ESTE NUMERO EN ALGUNA MEMORIA TEMPORAL, HASTA QUE DECIDAMOS QUE VA A HACER CON EL.
2. PRESIONAR LA TECLA "+" CON LO CUAL LA CALCULADORA TRASLADA EL "5" A UN ACUMULADOR INTERNO Y ESTA LISTA PARA RECIBIR EL SEGUNDO OPERANDO.
3. PRESIONAR LA TECLA "7" CON LO QUE HACE LA SUMA DE MANERA INTERNA EN EL ACUMULADOR Y MANTIENE EL RESULTADO OCULTO
4. PRESIONAR LA TECLA "=" ESTO AVISA A LA CALCULADORA QUE TERMINO CON LA SERIE DE OPERACIONES Y PARA QUE LIBERE EL RESULTADO

PROGRAMA

CONJUNTO EXPLICITO DE PASOS A SEGUIR PARA LOGRAR UN FIN DETERMINADO.

EN NUESTRO CASO

1. OBSERVA EL PRIMER NUMERO
2. LLEVALO AL ACUMULADOR PARA SUMARLO CON EL NUMERO QUE SIGUE.
3. EFECTUA LA SUMA, USANDO ESTE SEGUNDO NUMERO QUE AHORA OBSERVAS.
4. MUESTRAME EL RESULTADO

PRIMER PROGRAMA

SUMAR 5+7

1. NECESITAMOS TRES CASILLAS, DOS PARA LOS DATOS Y UNA PARA EL RESULTADO.
2. DEFINIR CON DETALLE LAS OPERACIONES A EFECTUAR Y SU ORDEN.
3. INTRODUCIR TODOS LOS DATOS E INSTRUCCIONES EN LA MEMORIA.

INSTRUCCION	CODIGO INTERNO	LONGITUD DE LA INSTRUCCION
CARGA AC	21	2
GUARDA AC	96	2
SUMA	57	2
RESTA	42	2

PROGRAMA PARA SUMAR 5+7

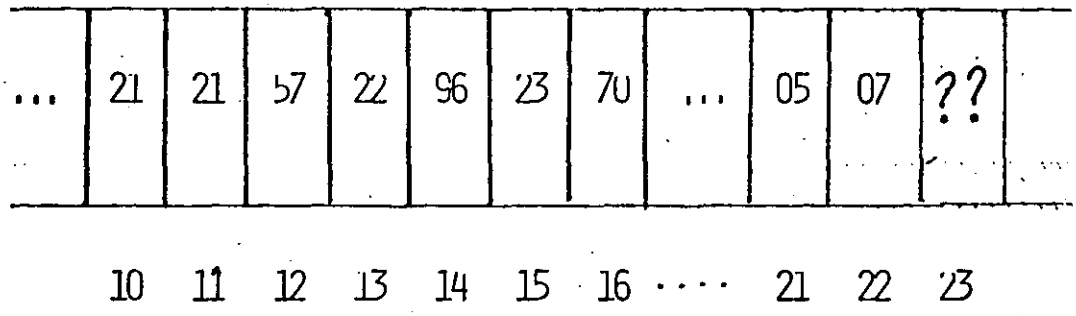
INSTRUCCION	DIRECCION	CODIGO	COMENTARIOS
CARGA AC	21	2121	COLOCAMOS EL PRIMER NUMERO EN EL ACUMULADOR
SUMA	22	5722	EFECTUAMOS LA SUMA
GUARDA AC	23	9623	DEJAMOS EL RESULTADO EN LA CASILLA 23
ALTO		70	

PROGRAMA OBJETO

212157229623 70

NUESTRO PROGRAMA OBJETO, YA

CARGADO EN MEMORIA, SE VERA ASI:



QUE SE VERA

HARDWARE

UNIDAD CENTRAL DE PROCESO (C P U)

MEMORIA PRINCIPAL

UNIDAD DE CONTROL

UNIDAD ARITMETICO LOGICA

REGISTROS

PERIFERICOS

ENTRADA/SALIDA (E/S)

ALMACENAMIENTO

PROCESADOR DE (E/S)

CANALES

CONTROLADORES

QUE SE PODRA CONTESTAR

CUALES SON LOS DOS COMPONENTES BASICOS DE UNA COMPUTADORA?

CUALES SON LOS COMPONENTES DE LA C P U?

QUE ES LA C P U?

QUE SON LAS UNIDADES PERIFERICAS?

DONDE DEBE RESIDIR UN PROGRAMA PARA SU EJECUCION?

QUE DIFERENCIA EXISTE ENTRE MEMORIA PRINCIPAL Y MASIVA?

QUE TIPOS DE ENTRADA, SALIDA Y ALMACENAMIENTO CONOCE?

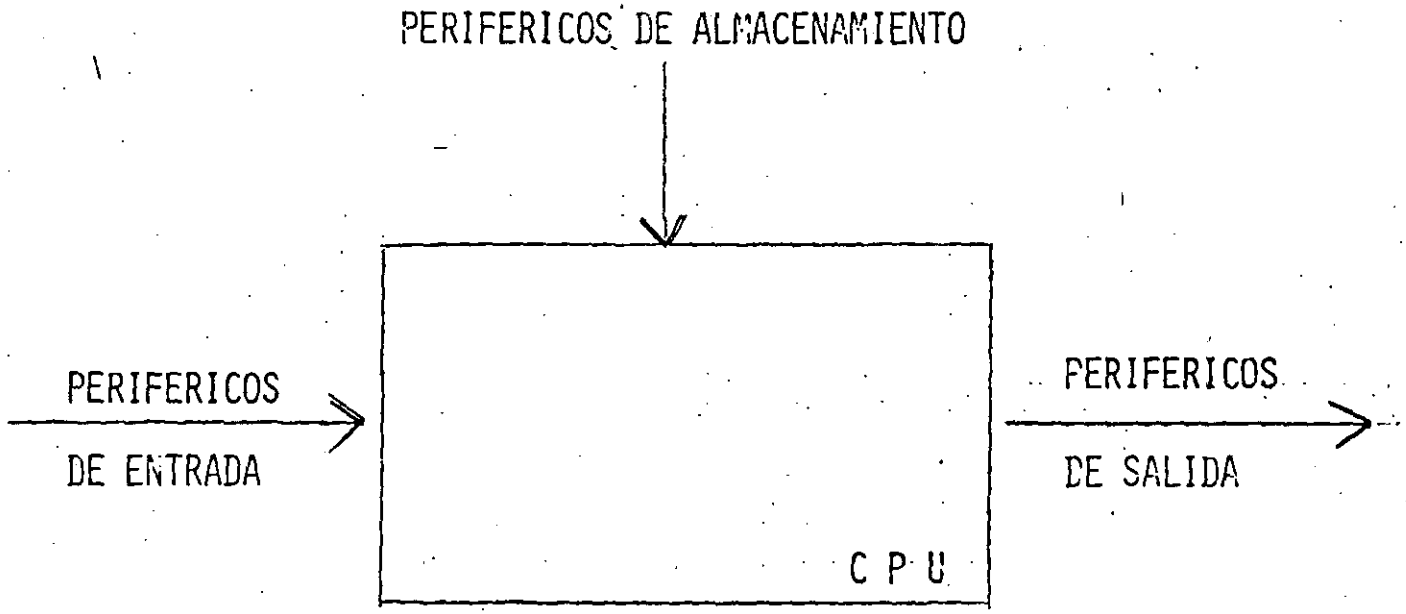
QUE CARACTERISTICAS CONSIDERA IMPORTANTES EN UNA IMPRESORA?

HARDWARE:

SON TODOS AQUELLOS COMPONENTES MECANICOS, ELECTROMECANICOS, Y ELECTRONICOS QUE FORMAN LA ESTRUCTURA FISICA DE LA COMPUTADORA, CUYO OBJETIVO ES EFECTUAR FISICAMENTE LOS PROCESOS DE CAPTACION DE INFORMACION, OPERACIONES LOGICO-ARITMETICAS, ALMACENAMIENTO DE LA INFORMACION Y OBTENCION DE RESULTADOS:

LA ESTRUCTURA FISICA SE DIVIDE EN DOS PARTES PRINCIPALES:

COMPONENTES BASICOS DE LA COMPUTADORA

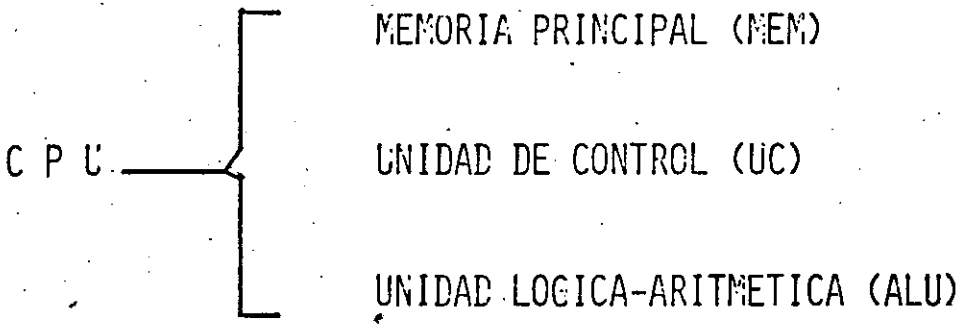


UNIDAD CENTRAL DE PROCESO (C P U):

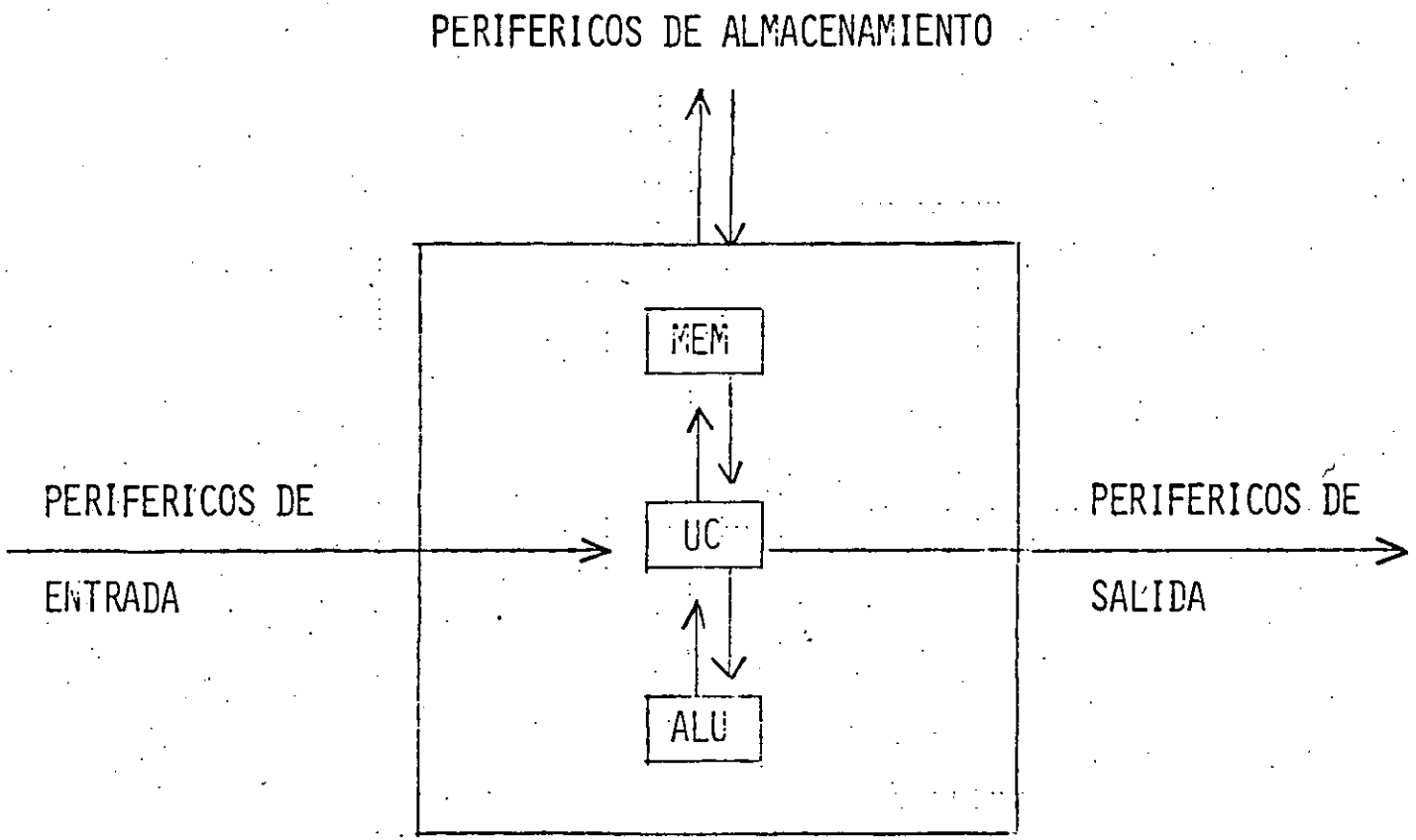
ES EL DISPOSITIVO QUE SE ENCARGA DE EJECUTAR UN CONJUNTO DE INSTRUCCIONES (PROGRAMA) PARA UN FIN ESPECIFICO.

ES UN DISPOSITIVO QUE CONTROLA Y COORDINA TODAS LAS ACTIVIDADES QUE LLEVA A CABO UNA COMPUTADORA. EN EL SE LLEVAN A CABO LAS OPERACIONES DE INTERPRETACION DEL PROGRAMA Y EL TRATAMIENTO LOGICO-ARITMETICO DE LOS DATOS.

SE COMPONE PRINCIPALMENTE DE TRES COMPONENTES:



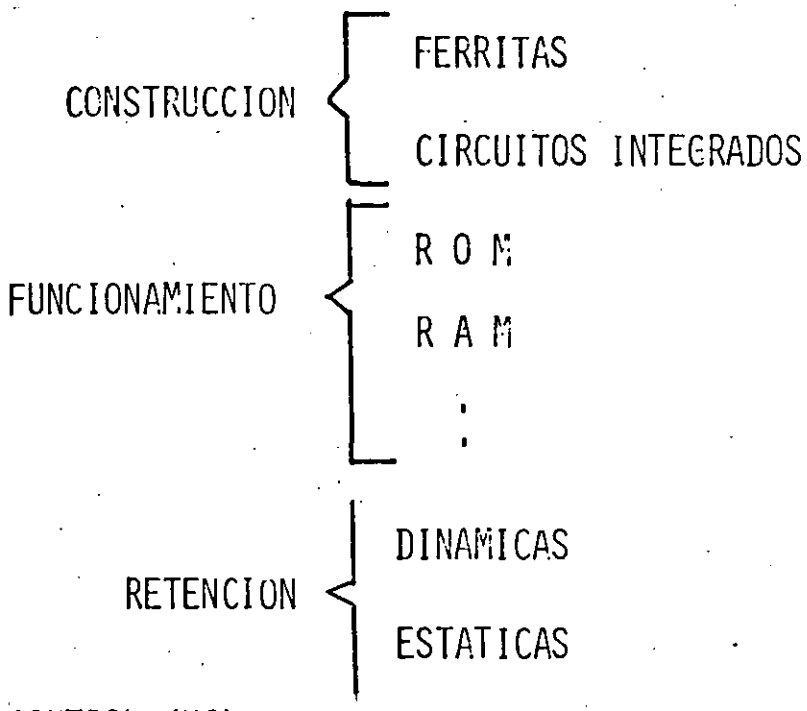
C P U


PERIFERICOS DE ALMACENAMIENTO
MEM
PERIFERICOS DE
ENTRADA
UC
PERIFERICOS DE
SALIDA
ALU

MEMORIA (MEM):

ES UN DISPOSITIVO CAPAZ DE ALMACENAR INFORMACION BINARIA.

SE PUEDEN DIVIDIR DE DIFERENTE MANERA:



UNIDAD DE CONTROL (UC):

ES UN DISPOSITIVO QUE SE ENCARGA DE CONTROLAR Y COORDINAR EL CONJUNTO DE OPERACIONES QUE HAY QUE REALIZAR PARA DAR EL OPORTUNO TRATAMIENTO DE LA INFORMACION.

UNIDAD LOGICA-ARITMETICA (ALU)

SE ENCARGA DE LA EJECUCION DE LAS OPERACIONES LOGICAS Y ARITMETICAS DE ACUERDO A LAS INSTRUCCIONES DE LA UNIDAD DE CONTROL

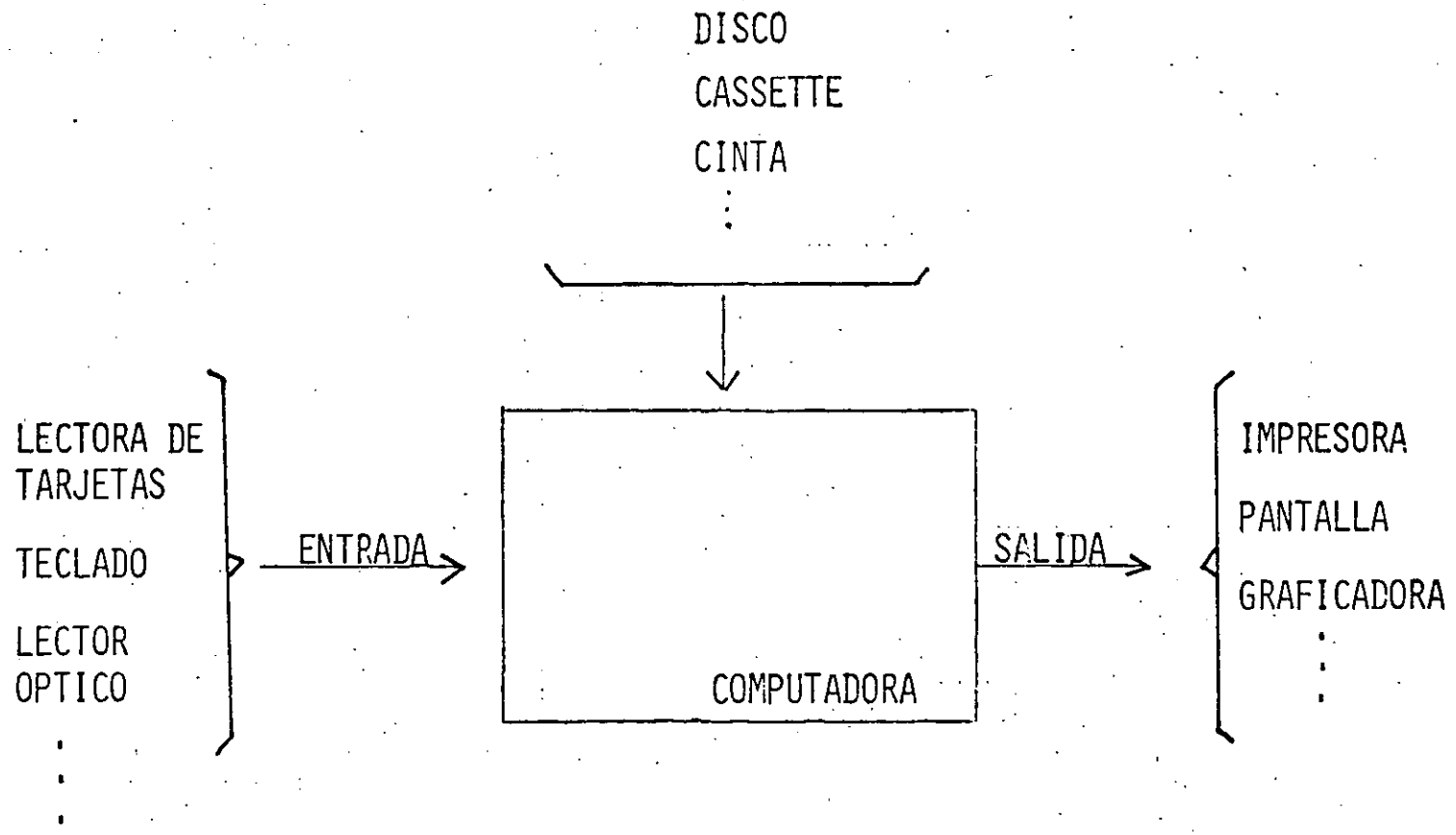
PERIFERICOS:

SON DISPOSITIVOS QUE SE ENCARGAN DE ENLAZAR A LA COMPUTADORA CON EL MEDIO AMBIENTE A TRAVES DE FUNCIONES DE ALIMENTACION Y ENTREGA DE RESULTADOS, O DE ALMACENAR GRANDES VOLUMENES DE INFORMACION MANTENIENDOLA A DISPOSICION DE LA COMPUTADORA.

SE DIVIDE EN TRES CATEGORIAS

- PERIFERICOS
  - ENTRADA
  - SALIDA
  - ALMACENAMIENTO

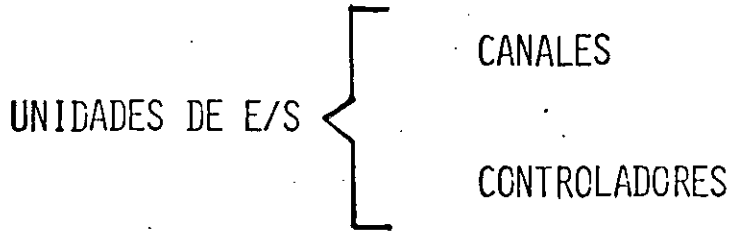
PERIFERICOS



UNIDADES DE E/S:

ADAPTAN LA INFORMACION PROCEDENTE DEL EXTERIOR PARA PODER SER INTERPRETADA POR LA COMPUTADORA, ASI COMO ADAPTAR LA INFORMACION SUMINISTRADA POR LA COMPUTADORA PARA QUE PUEDA SER TRATADA POR LOS PERIFERICOS.

EXISTEN DOS TIPOS DE UNIDADES DE E/S:



PROCESADOR DE COMUNICACION DE DATOS (DCP):

ES UN DISPOSITIVO QUE PERMITE TENER COMUNICACION DIRECTA ENTRE LAS COMPUTADORAS Y LAS TERMINALES REMOTAS.

PERIFERICOS

IMPRESORAS

LECTORAS DE TARJETAS

LECTORAS OPTICAS

TERMINALES

UNIDADES DE DISCO

PLOTTERS (GRAFICADORES)

DIGITALIZADORES

LECTORAS DE TARJETAS MAGNETICAS

MONITORES DE RAYOS CATODICOS

INTERFACES INDUSTRIALES

UNIDADES DE CINTA

IMPRESORAS:

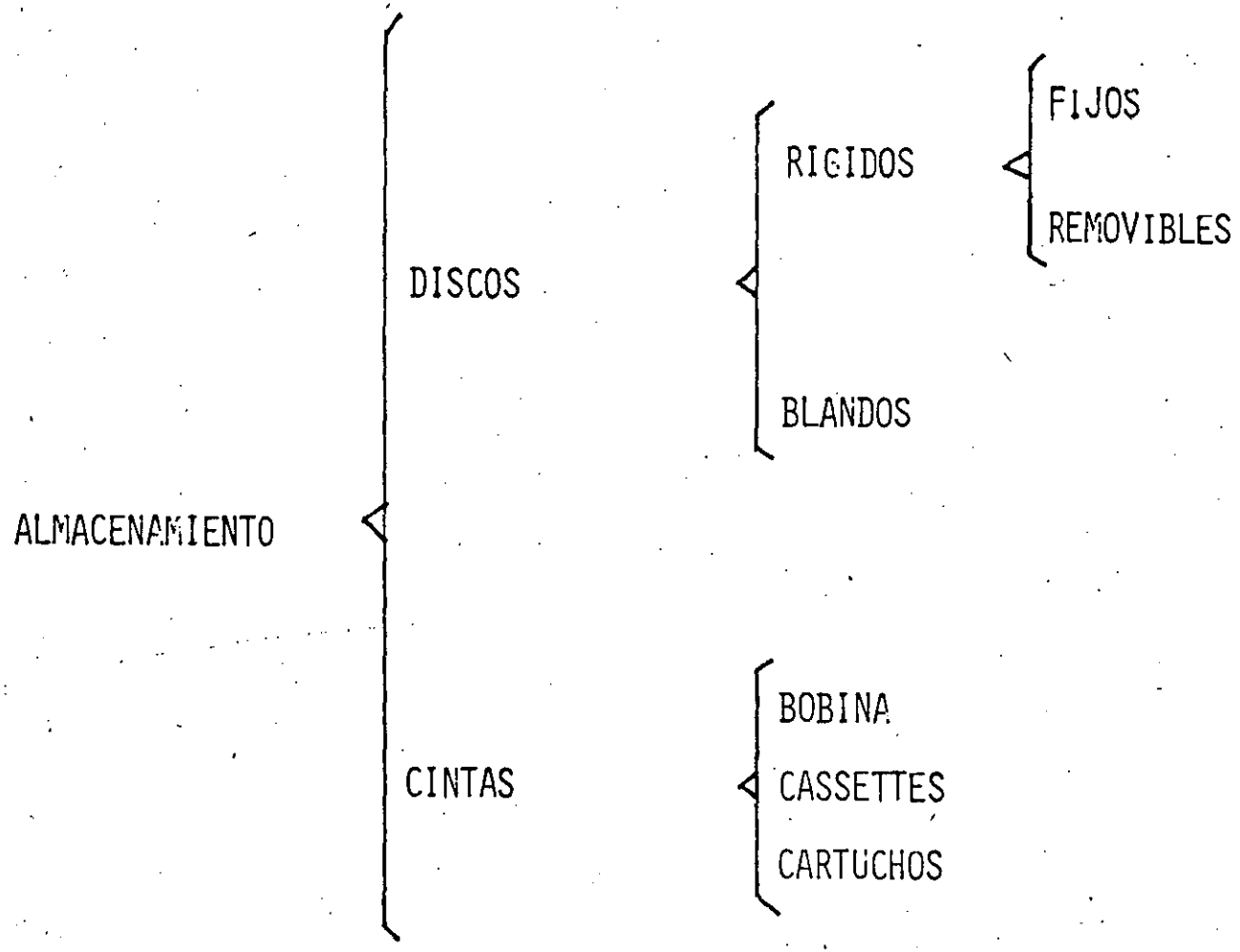
SON DISPOSITIVOS ELECTROMECANICOS QUE SIRVEN PARA IMPRIMIR LA INFORMACION ENVIADA POR LA COMPUTADORA.

EXISTEN DIFERENTES TIPOS DE ACUERDO A SU MECANISMO DE IMPRESION

- TIPOS DE IMPRESORAS
  - MARGARITA
  - MATRIZ DE PUNTOS
  - LINEAS
  - BANDA
  - BOLA
  - CILINDRO
  - LASER

- CARACTERISTICAS TECNICAS
  - ANCHO DE PAPEL
  - DENSIDAD DE CARACTERES POR LINEA
  - DENSIDAD DE LINEAS
  - FORMA DE ALIMENTACION DEL PAPEL
  - VELOCIDAD DE TRANSMISION DE CARACTERES
  - TIPO DE INTERFACE
  - TIPOS DE LETRA
  - ESPACIADO PROPORCIONAL
  - POSIBILIDAD DE SUBRAYADO
  - NUMERO MAXIMO DE COPIAS
  - CAPACIDAD GRAFICA

PERIFERICOS DE ALMACENAMIENTO



UNIDADES DE CINTA

CARACTERISTICAS DE LAS UNIDADES DE CINTA

- ANCHURA DE LA CINTA
- NUMERO DE PISTAS
- CAPACIDAD DE ALMACENAMIENTO
- DENSIDAD DE GRABACION
- CODIGO DE GRABACION
- VELOCIDAD DE LA CINTA

QUE SE VERA

SOFTWARE

CONCEPTOS ELEMENTALES

HISTORIA DE LOS LENGUAJES

PROGRAMACION DE SISTEMAS

ENSAMBLADOR

COMPILADOR

:

:

:

SISTEMA OPERATIVO

PROGRAMACION DEL USUARIO

METODOS DE PROCESO DE DATOS

ELEMENTOS DE PROGRAMACION

ADMINISTRACION DE PROYECTOS EN INFORMATICA

QUE SE PODRA CONTESTAR

QUE ES SOFTWARE?

QUE ES UN PROGRAMA?

CUAL FUE EL PRIMER LENGUAJE DE PROGRAMACION?

QUE LENGUAJE APLICARIA PARA CALCULOS MATEMATICOS?

QUE DIFERENCIA EXISTE ENTRE COMPILADOR E INTERPRETE?

QUE PROGRAMA SUPERVISA A LA COMPUTADORA?

MENCIONE TRES APLICACIONES DEL USUARIO

QUE ES PROCESAMIENTO DISTRIBUIDO?

CUALES SON LAS ETAPAS EN UN PROYECTO DE INFORMATICA?

DEFINICIONES

DATO:

ES CUALQUIER ELEMENTO QUE SIRVE DE PUNTO DE PARTIDA PARA UNA DECISION, CALCULO O MEDIDA.

INFORMACION:

ES EL RESULTADO DE UN PROCESO DE DATOS.

INSTRUCCIONES:

SON LAS QUE GOBIERNAN LA TRANSFERENCIA DE INFORMACION DENTRO DE LA MAQUINA, ADEMAS DE ESPECIFICAR LAS OPERACIONES LOGICAS Y ARITMETICAS QUE SE EFECTUARAN.

PROGRAMA:

ES UN CONJUNTO DE INSTRUCCIONES PERFECTAMENTE LEGIBLES POR LA COMPUTADORA, ORDENADAS SECUENCIALMENTE PARA REALIZAR UN DETERMINADO TRABAJO O PARA SOLUCIONAR UN PROBLEMA.

SOFTWARE:

SON TODOS AQUELLOS PROGRAMAS QUE ESTAN ESCRITOS EN UN LENGUAJE APROPIADO A LA ESTRUCTURA FISICA DE LA COMPUTADORA Y CON LOS CUALES ES POSIBLE UTILIZARLA.

EL SOFTWARE PUEDE SER DIVIDIDO EN DOS GRANDES RAMAS:

- SOFTWARE
  - SISTEMA
    - ENSAMBLADORES
    - COMPILADORES
    - INTERPRETES
    - RUTINAS DE UTILERIA
    - PAQUETES DE BIBLIOTECA
    - INTRINSECOS
    - SISTEMA OPERATIVO
  - USUARIO
    - CONTABILIDAD
    - GESTION ODONTOLOGICA
    - CONTROL ESCOLAR
    - INVENTARIOS
    - METODOS NUMERICOS
    - SIMULACION
    - :

SISTEMA OPERATIVO (S.O.)

PROGRAMA QUE SE ENCARGA DE LA SUPERVISION DEL CONTROL INTERNO DE LA COMPUTADORA.

ALGUNAS FUNCIONES DEL SISTEMA OPERATIVO

- MANEJO DE MEMORIA
  (DISTRIBUCIÓN, RESERVACIÓN)
- CONTROL DE PROGRAMAS
  (SEGURIDAD, DIRECTORIOS)
- MANEJO DE PERIFERICOS
  (INTERRUPCIONES, COLAS)
- FACILIDADES
  (ESTADÍSTICAS, INSTRINSECOS)

LENGUAJE DE MAQUINA:

ES EL PRIMER LENGUAJE DE PROGRAMACION, COMPUESTO DE UN CONJUNTO DE SIMBOLOS BINARIOS QUE SERAN INTERPRETADOS POR LA COMPUTADORA.

LENGUAJE ENSAMBLADOR:

ESTA COMPUESTO DE MNEMONICOS Y DIRECCIONES SIMBOLICAS, PARA LA REPRESENTACION DE UNA INSTRUCCION.

LENGUAJE MACROENSAMBLADOR:

TIENEN UN CONJUNTO DE MACROINSTRUCCIONES, DONDE CADA UNA DE ELLAS EJECUTAN UNA SERIE DE INSTRUCCIONES DE ENSAMBLADOR.

LENGUAJE DE ALTO NIVEL:

ES UN LENGUAJE QUE TIENE POCA DEPENDENCIA DE LA MAQUINA, MAS PODEROSO, TENIENDO UN CONJUNTO DE INSTRUCCIONES DE OPERACION, MENOS BASICAS QUE EL LENGUAJE ENSAMBLADOR.

LOS LENGUAJES DE ALTO NIVEL PUEDEN SER CLASIFICADOS DE LA SIGUIENTE FORMA:

CLASIFICACION DE LENGUAJES DE ALTO NIVEL

- LENGUAJES CIENTIFICOS
- LENGUAJES DE GESTION
- LENGUAJES POLIVALENTES
- LENGUAJES PARA PROCESO DE LISTAS
- LENGUAJES ESPECIALES

LENGUAJES DE QUINTA GENERACION:

LENGUAJE QUE PERMITE GENERAR PROGRAMAS EN LENGUAJE DE ALTO NIVEL.

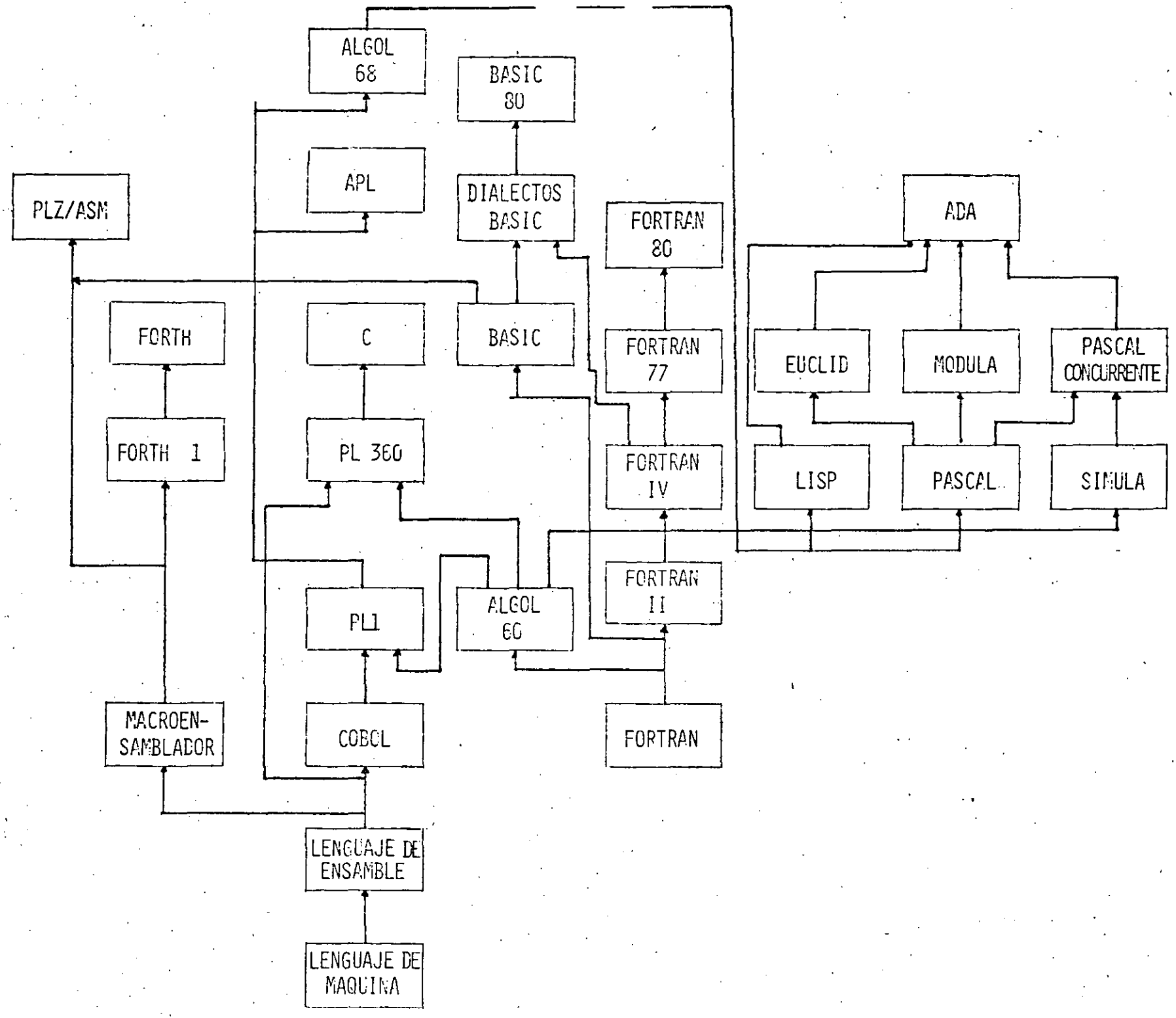
HISTORIA DE LOS LENGUAJES

COMO EN EL CASO DEL HARDWARE EL SOFTWARE TAMBIEN HA TENIDO UN GRAN DESARROLLO, TRATANDO DE INCLINARSE HACIA EL MEJOR ENTENDIMIENTO POR EL SER HUMANO.

A CONTINUACION SE DA UNA RESEÑA DE LA APARICION DE LOS LENGUAJES:

DESARROLLO DE LOS LENGUAJES

- LENGUAJE DE MAQUINA
- LENGUAJE ENSAMBLADOR
- LENGUAJE MACROENSAMBLADOR
- LENGUAJE DE ALTO NIVEL
- LENGUAJES DE QUINTA GENERACION


ALGOL 68
BASIC 80
APL
DIALECTOS BASIC
FORTRAN 80
PLZ/ASM
ADA
FORTH
C
BASIC
FORTRAN 77
EUCLID
MODULA
PASCAL CONCURRENTE
FORTH 1
PL 360
FORTRAN IV
LISP
PASCAL
SIMULA
PL1
ALGOL 60
FORTRAN II
MACROEN-SAMBLADOR
COBOL
FORTRAN
LENGUAJE DE ENSAMBLE
LENGUAJE DE MAQUINA

COMPARACION DE LENGUAJES

LENGUAJE	DEPENDENCIA DEL EQUIPO	DIFICULTAD	MANTENIBILIDAD	DOCUMENTACION	TIEMPO DE EJECUCION	REQUERIMIENTO DE MEMORIA
DE MAQUINA	A	A	B	B	A	M
ENSAMBLADOR	A	M	M	A	A	M
ALTO NIVEL	B	B	A	A	M	A
5A. GENERACION	B	B	A	A	M	A

A = ALTA

M = MEDIA

B = BAJA

METODOS DE PROCESO DE DATOS

SE TIENEN DIFERENTES FORMAS PARA EL PROCESAMIENTO DE DATOS EN FUNCION DE LA CAPACIDAD DE LA COMPUTADORA Y DEL SISTEMA OPERATIVO ENTRE LOS QUE SE ENCUENTRAN:

METODOS PRINCIPALES DE PROCESO DE DATOS

- EN LOTES (BATCH)
- INTERACTIVO
- TIEMPO COMPARTIDO
- PROCESO DISTRIBUIDO
- PROCESO DESCENTRALIZADO

ELEMENTOS DE PROGRAMACION

LOS ELEMENTOS DE PROGRAMACION SON AQUELLOS QUE SUSTENTAN LOS PROGRAMAS:

SE TIENEN TRES TIPOS BASICOS DE PROGRAMAS:

- TIPOS DE PROGRAMAS
  - LINEALES
  - CICLICOS
  - CONDICIONALES

DE UNA MANERA MAS FORMAL UN PROGRAMA CONSISTE DE LOS ELEMENTOS BASICOS DEL LENGUAJE Y SUS INSTRUCCIONES, DIVIDIENDOSE DE LA SIGUIENTE FORMA:

- ELEMENTOS BASICOS DEL LENGUAJE
  - CONJUNTO DE CARACTERES
  - NOMBRES SIMBOLICOS
  - CONSTANTES
  - VARIABLES
  - PALABRAS RESERVADAS

- TIPOS DE INSTRUCCIONES
  - INICIO/FIN
  - CALCULO ARTIMETICO
  - CALCULO LOGICO
  - TRANSFERENCIA DE CONTROL
  - ENTRADA/SALIDA
  - DEFINICION
  - TRANSFERENCIA DE DATOS

ADMINISTRACION DE PROYECTOS EN INFORMATICA

PARA EL ADECUADO DESARROLLO DE UN PROYECTO INFORMATICO SE DEBEN TOMAR EN CUENTA LAS SIGUIENTES FASES Y TECNICAS UTILIZADAS:

FASE	TECNICA(S)	PRODUCTO GENERADO
PLANEACION	ANALISIS DE COSTO/BENEFICIO COSTOS	REPORTE PRELIMINAR
ANALISIS	ANALISIS ESTRUCTURADO	ESPECIFICACION DE REQUERIMIENTOS
DISEÑO	DISEÑO MODULAR PSEUDOCODIGO	DIAGRAMAS DE ESTRUCTURA DISEÑO DETALLADO
CONSTRUCCION	PROGRAMACION ESTRUCTURADA	CODIGO FUENTE
PRUEBAS	CAJA NEGRA CAJA BLANCA	PLAN DE PRUEBAS
LIBERACION	CAPACITACION CONVERSION MANTENIMIENTO	MANUAL DE USUARIO SISTEMA FUNCIONANDO

DESCRIPCION ALGORITMICA

- ORACIONAL
  - LENGUAJE NATURAL
  - LENGUAJE ALGORITMICO
- ESQUEMATICA
  - DIAGRAMA DE FLUJO
- MIXTA
  - DIAGRAMA WARNIER

ORACIONAL

- ALGORITMO (NOMBRE)
- ENTRADA (PARAMETRO, ...)
- SALIDA (PARAMETRO, ...)
- DEFINICION DE VARIABLES
- INICIALIZACION DE VARIABLES
- CONJUNTO DE INSTRUCCIONES

SI CONDICION ENTONCES INSTRUCCION

MIENTRAS CONDICION HAZ

INSTRUCCION(ES)

FIN DE MIENTRAS

HASTA CONDICION HAZ

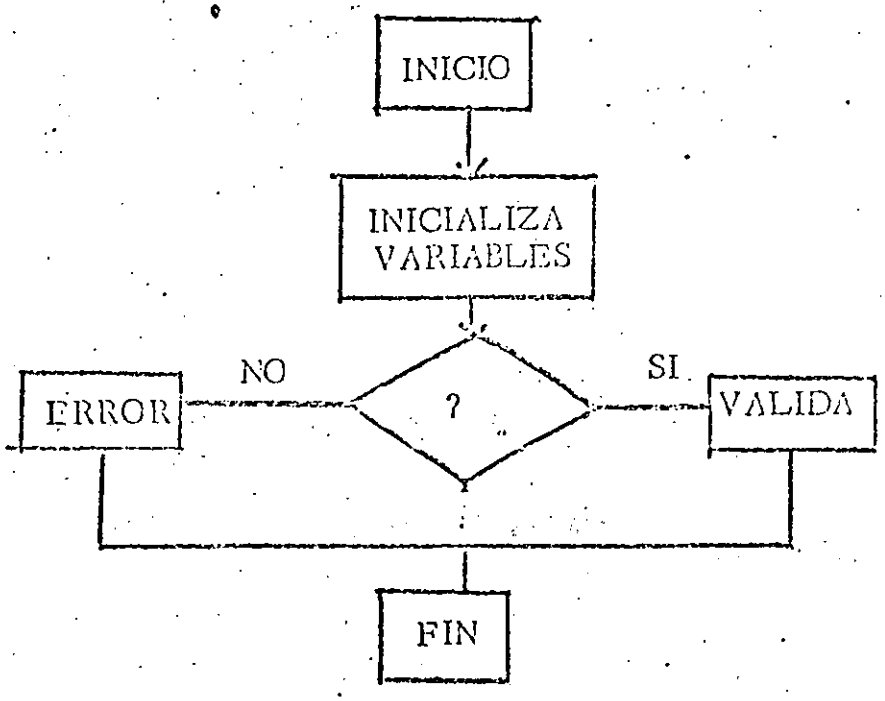
INSTRUCCION(ES)

FIN DEL HASTA

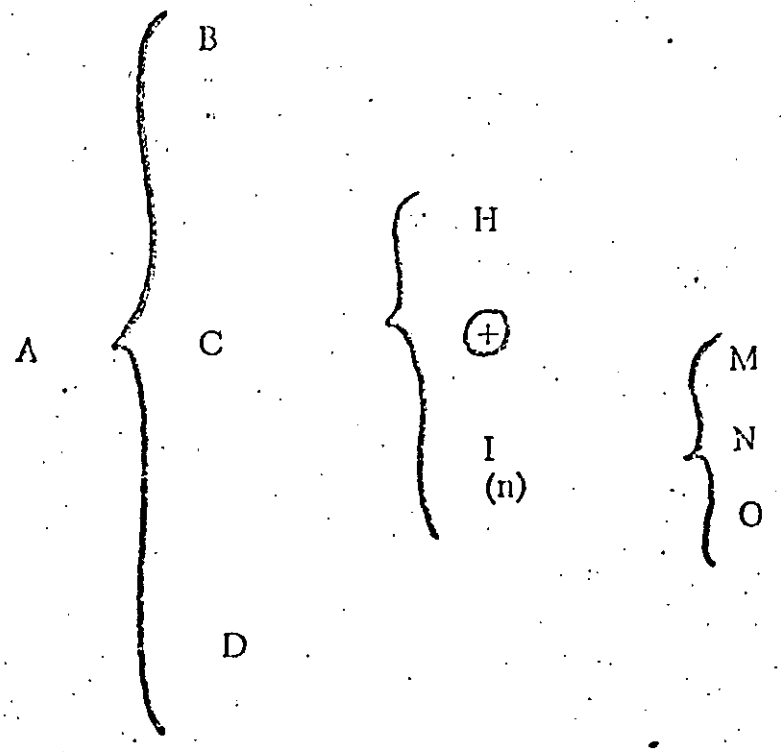
FIN DEL SI

FIN DEL ALGORITMO (NOMBRE)

ESQUEMATICA



MIXTA



ESTRUCTURAS BASICAS

- SECUENCIA

- DECISION O ALTERNATIVA
  - SENCILLO
  - DOBLE
  - MULTIPLE

- REPETICION

SECUENCIA

VERIFICA CLIENTE

CALCULA SUELDO

ASIGNA INTERÉS

REPORTA ERROR

.
.
.
.
.

DECISION SIMPLE

SI CONDICION ENTONCES INSTRUCCION

DECISION DOBLE

SI CONDICION ENTONCES

INSTRUCCION-1

SINO

INSTRUCCION-2

FIN SI

DECISION MULTIPLE

SI CONDICION-1 ENTONCES

INSTRUCCION-1

SINO SI CONDICION-2 ENTONCES

INSTRUCCION-2

SINO SI CONDICION-3 ENTONCES

INSTRUCCION-3

.

.

.

SINO

INSTRUCCION-N

FIN SI

C A S O

```
CASO
    CONDICION-1
        INSTRUCCION-1
    CONDICION-2
        INSTRUCCION-2
    CONDICION-3
        INSTRUCCION-3
        .
        .
        .
    SINO
        INSTRUCCION-N
FIN CASO
```

CASO

"VARIABLE O EXPRESION"

(LISTA DE VALORES-1)

INSTRUCCION-1

(LISTA DE VALORES-2)

INSTRUCCION-2

(LISTA DE VALORES-3)

INSTRUCCION-3

.
.
.

SINO

INSTRUCCION-N

FIN CASO

EJEMPLOS

CASO

(MES = 1, 3, 5, 7, 8, 10 , 12)

ASIGNA 31 AL NUMERO DE DIAS

(MES = 4, 6, 9, 11)

ASIGNA 30 AL NUMERO DE DIAS

(AÑO BISIESTO)

ASIGNA 29 AL NUMERO DE DIAS

FIN CASO

```
CASO  ESTADO CIVIL
      (CASADO)
         PROCESA EMPLEADO CASADO
      (SOLTERO)
         PROCESA EMPLEADO SOLTERO
      (DIVORCIADO)
         PROCESA EMPLEADO DIVORCIADO
      (VIUDO)
         PROCESA EMPLEADO VIUDO
      (SEPARADO)
         PROCESA EMPLEADO SEPARADO
      SINO
         REPORTE ERROR ESTADO CIVIL
FIN CASO
```

L A R E P E T I C I O N

- MIENTRAS
- EJECUTA - HASTA
- REPITE
- CICLO

MIENTRAS

MIENTRAS CONDICION HAZ

INSTRUCCION-1

INSTRUCCION-2

INSTRUCCION-3

FIN MIENTRAS

EJECUTA - HASTA

EJECUTA

INSTRUCCION-1

INSTRUCCION-2

INSTRUCCION-3

.
.
.

HASTA CONDICION

REPITE

REPITE VARIABLE = INICIO, FIN, INCREMENTO

INSTRUCCION-1

INSTRUCCION-2

INSTRUCCION-3

.
.
.

FIN REPITE

C I C L O

CICLO EXPRESION

INSTRUCCION-1

INSTRUCCION-2

INSTRUCCION-3

.
.
.

FIN CICLO

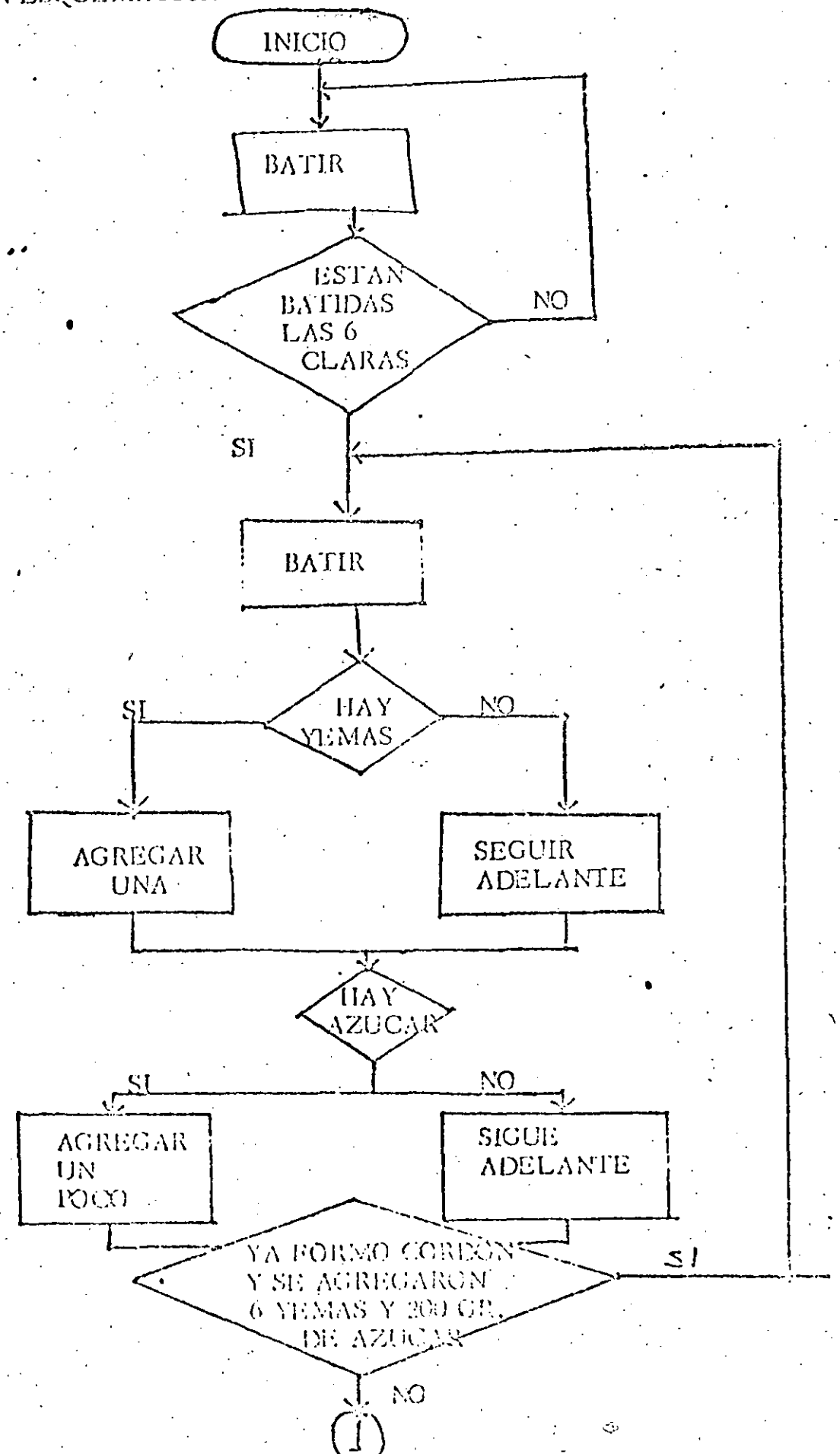
ESTRUCTURAS ANIDADAS

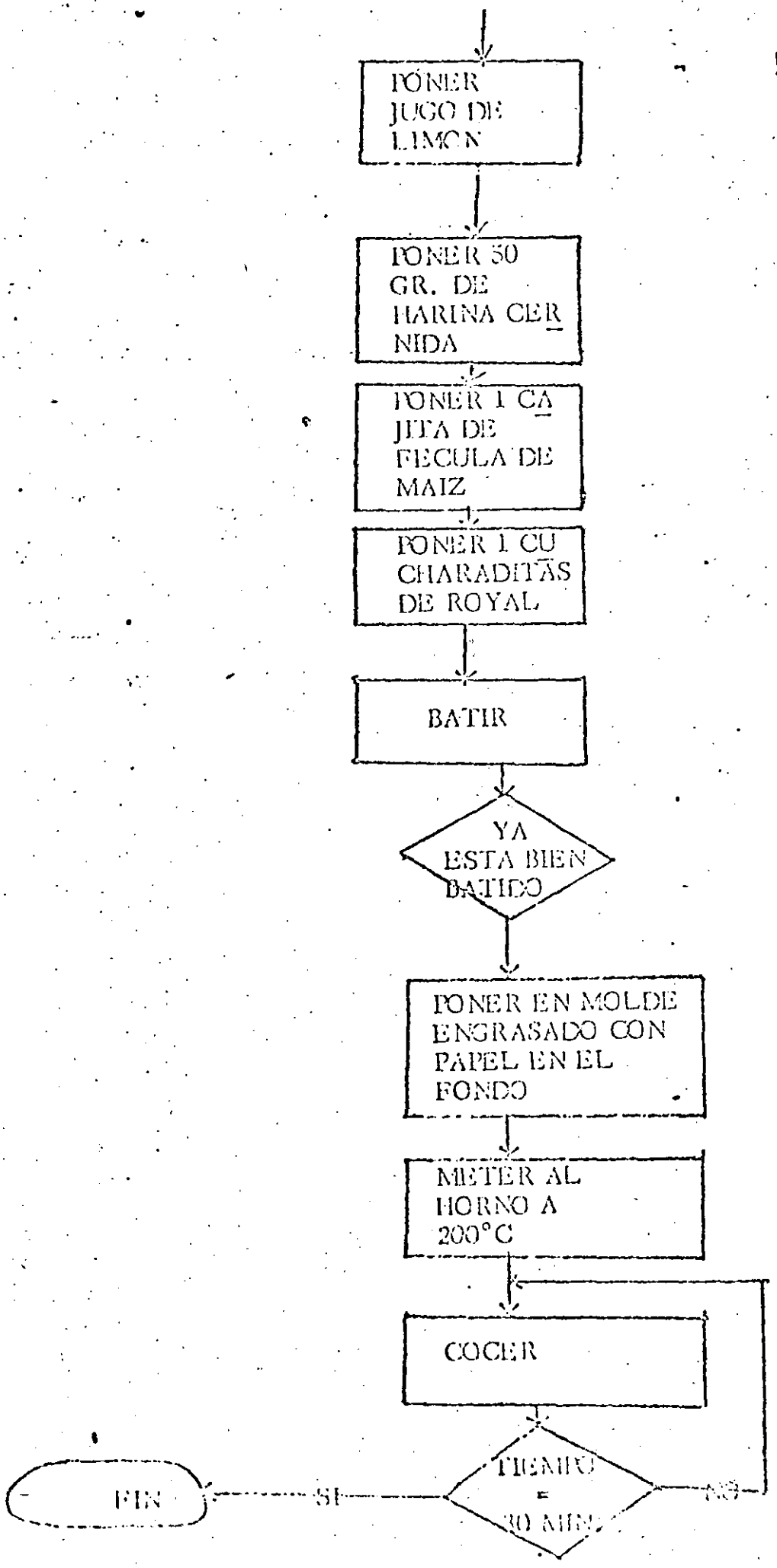
```
MIENTRAS  CONDICION  HAZ
    SI  CONDICION-1  ENTONCES
        INSTRUCCION(ES) - 1
    SINO
        CICLO  EXPRESION
            INSTRUCION(ES) - 2
        FIN CICLO
    FIN SI
    INSTRUCCION(ES) - 3
FIN MIENTRAS
```

DESCRIPCION EN LENGUAJE NATURAL.

SE BATEN SEIS CLARAS DE HUEVO MUY BIEN, LUEGO SE PONEN - UNA A UNA SEIS YEMAS, SE VAN INCORPORANDO 200 GR. DE AZUCAR, SE SIGUE BATIENDO HASTA QUE FORME UN CORDON; SE PONE ZUMO DE UN LIMON, 50 GR. DE HARINA CERNIDA CON UNA CAJITA DE FECULA DE MAIZ Y DOS CUCHARADITAS DE ROYAL; YA BIEN BATIDO, - SE PONE EN UN MOLDE BIEN ENGRASADO, SE METE AL HORNO A - 250°C , DURANTE MEDIA HORA.

DESCRIPCION ESQUEMATICA




PONER JUGO DE LIMON
PONER 50 GR. DE HARINA CERNIDA
PONER 1 CAJITA DE FECULA DE MAIZ
PONER 1 CUCHARADITAS DE ROYAL
BATIR
YA ESTA BIEN BATIDO
PONER EN MOLDE ENGRASADO CON PAPEL EN EL FONDO
METER AL HORNO A 200°C
COCER
TIEMPO = 30 MIN
SI
NO
FIN

Descripción en Lenguaje Algorítmico

ALGORITMO ESCOTAFI

ENTRADA (Huevos, azúcar, limón, harina cernida, caja de fécula de maíz, cucharadas de royal)
SALIDA (Escotafi)

INICIALIZACION

Huevos _____ 6
Azúcar _____ 200 gr.
Limón _____ 1
Harina cernida _____ 50 gr.
Caja de fécula de maíz — 1
Cucharadas royal _____ 2

FIN DE INICIALIZACION
HASTA que las 6 claras estén bien batidas Haz
Batir

FIN DE HASTA
HASTA que forme cordón y se hayan agregado 6 yemas y 200 gr. de azúcar Haz
Batir

SI hay yemas encimas
Agregar una
SINO
Sigue adelante
FIN SI

FIN DE HASTA

Poner zumo de un limón verde
Poner 50 gr. de harina cernida
Poner una cajita de fécula de maíz
Poner 2 cucharadas de royal
HASTA que esté bien batido Haz
Batir

FIN DE HASTA

Poner en molde previamente con papel en el fondo
Meter en horno a 180°
HASTA que pinchando con un palillo salga limpio
Cocer

FIN DE HASTA

FIN DE ALGORITMO ESCOTAFI

DESCRIPCION MIXTA

- ESCOTAFI
  - BATIDO 1 (hasta que estén las 6 claras bien batidas)
    - BATIR
  - BATIDO2 (hasta que forme cordón y se hayan agregado 6 yemas y 200 gr. de azúcar
    - BATIR
    - Existencia de yemas
      - hay ⊕ → agregar una
      - No hay → sigue adelante
    - Existencia de azúcar
      - hay ⊕ → agregar un poco
      - No hay → sigue adelante
  - Condimentación
    - Poner zumo de un limón
    - Poner 50 gr. de harina cernida
    - Poner 2 cucharaditas de royal
  - BATIDO 3 (hasta que esté bien batido)
    - BATIR
  - COCIMIENTO (hasta tiempo = a 30 minutos)
    - COCER

HISTORIA DEL LENGUAJE FORTAN

CON EL ALVENIMIENTO DE LAS COMPUTADORAS, SE DESCUBRIO LA GRAN AYUDA QUE PROPORCIONARIA PARA LA SOLUCION DE PROBLEMAS EN LA INVESTIGACION TECNICA Y CIENTIFICA AL IGUAL QUE EN EL CAMPO DE LA INGENIERIA.

PERO EXISTIA EL PROBLEMA DE LOS LENGUAJES DE PROGRAMACION DISPONIBLES: LENGUAJES DE MAQUINA Y ENSAMBLADORES.

DE AHI SURGE LA NECESIDAD DE UN LENGUAJE PARA LA SOLUCION DE PROBLEMAS CIENTIFICOS Y TECNICOS, QUE MANEJA UNA NOTACION PARECIDA A LA ARITMETICA.

ESTE LENGUAJE SE LLAMO FORTRAN (FORmula TRANslation). LA PRIMERA VERSION LA PUBLICO EN 1954, J. BACKUS PARA IBM, POSTERIORMENTE EN 1958: FORTRAN II Y PARA 1962: FORTRAN IV.

A PARTIR DE ENTONCES EXISTEN ALREDEDOR DE 50 VERSIONES O COMPILADORES FORTRAN. LA MAS AVANZADA ES FORTRAN 77

A PESAR DE SUS CONSTANTES REVISIONES EXISTEN LENGUAJES MAS PODEROSOS COMO PASCAL O VERSIONES AVANZADAS DE BASIC.

POR SUS CARACTERISTICAS FORTRAN ES UN LENGUAJE QUE SE PUEDE ENTENDER Y APRENDER FACILMENTE, ES TRANSPORTABLE E INDEPENDIENTE DE LA MAQUINA.

ELEMENTOS DEL LENGUAJE FORTRAN

- ELEMENTOS BASICOS
- CONSTANTES Y VARIABLES
- EXPRESIONES ARITMETICAS
- PROPOSICIONES O INSTRUCCIONES FORTRAN
  - DE ESPECIFICACION
  - DE CONTROL
  - DE ENTRADA Y SALIDA
  - SUBPROGRAMA

ELEMENTOS BASICOS

. CONJUNTO DE CARACTERES

SE COMPONE DE LOS SIGUIENTES ELEMENTOS:

A) LETRAS

A..... Z

A..... z

B) DIGITOS

0..... 9

C) CARACTERES ESPECIALES

Y = + - (/) , . $ " ' : !

CON ESTOS COMPONENTES SE CONSTRUYEN LOS SIMBOLOS, EXPRESIONES Y ENUNCIADOS QUE UTILIZA EL LENGUAJE FORTRAN, PARA CONSTRUIR UN PROGRAMA FUENTE.

CONSTANTES Y VARIABLES

CONSTANTES:

. ENTERAS O DE PUNTO FIJO. SE COMPONEN DE UN CIERTO NUMERO DE DIGITOS SIN PUNTO DECIMAL. (EL SIGNO + ES OPCIONAL)

EJEMPLO:

VALIDOS: 549 - 41 +1 - 3CC 2

NO VALIDOS: 5.0 7 9 3 4 5 9 4 5 3

. REALES O DE PUNTO FLOTANTE. SE COMPONEN DE VARIOS DIGITOS CON PUNTO DECIMAL YA SEA EL PRINCIPIO, FINAL O ENTRE DOS DIGITOS CUALESQUIERA. SI APARECE UN PUNTO EN CUALQUIER CONSTANTE SE CONSIDERA DE PUNTO FLOTANTE.

EJEMPLO:

VALIDAS: 450.3 - 0.00049 + 12.45 - 0.7

NO VALIDAS: 530 1 2 4 5 9 4 3 1.3 2

TAMBIEN SE PERMITE LA FORMA EXPONENCIAL:

1.328E02	-	132.8
1.328E2	-	132.8
-4.724E-03	-	-.004724
+7.61E3	-	7610.
-6432E-3	-	-6.432

VARIABLES

. ENTERAS. SE REPRESENTAN POR COMBINACIONES DE 1 A 8 LETRAS Y/O DIGITOS; DONDE SIEMPRE EL PRIMER CARACTER DEBE SER UNA DE LAS LETRAS: I, J, K, L, M o N.

SUS VALORES SON SIEMPRE ENTEROS.

EJEMPLO:

VALIDAS: NUM KIL MET LIB JUE 129

NO VALIDAS: CONT PI ZAC 123

: REALES. SE REPRESENTAN POR COMBINACIONES DE 1 A 8 LETRAS Y/O DIGITOS DONDE SIEMPRE EL PRIMER CARACTER DEBE SER UNA DE LAS LETRAS DE LA A, A LA H, Y DE LA O, A LA Z.
SUS VALORES SON SIEMPRE REALES.

EJEMPLO:

VALIDAS: ALFA CAR TIP SOL BIT C29

NO VALIDAS: N29 3.9 SUBTOTAL 01

EXPRESIONES ARITMETICAS

. OPERADORES ARITMETICOS

LAS OPERACIONES ARITMETICAS Y LOS SIMBOLOS QUE SE UTILIZAN EN FORTRAN SON:

		JERARQUIA(*)
ADICION	+	4
SUBSTRACCION	-	
MULTIPLICACION	*	
DIVISION	/	3
EXPONENCIACION	**	2
AGRUPAMIENTO	()	1

. EXPRESIONES ARITMETICAS

EN FORTRAN:	COMUNES:
A**2 + B**2	$A^2 - B^2$
B**2 - 4 * A * C	$B^2 - 4\ AC$
(A-B)/2	$1/2 \times (A-B)$

REGLAS PARA FORMAR EXPRESIONES ARITMETICAS

1. LAS CONSTANTES Y VARIABLES DEBEN DE ESTAR EN EL MISMO MODO, AUNQUE SE PERMITE EL MODO MIXTO.

2. CUALQUIER EXPRESION SE PUEDE ENCERRAR O AGRUPAR ENTRE PARENTESIS.

3. NO SE PERMITEN DOS OPERADORES ARITMETICOS EN SECUENCIA

4. NO PUEDEN SUPONERSE SIGNOS DE OPERACION

5. LA EXPRESION:
   A**B**C. ES VALIDA Y SE EVALUA COMO: A**(B**C) CORRESPONDIENDO: A^{B^C}

. ASIGNACION

EL SIGNO = EN FORTRAN SE INTERPRETA COMO: EL VALOR QUE APARECE DEL LADO DERECHO ES ASIGNADO AL DEL IZQUIERDO.

```
A = B  Z
A = A - 1
Y = SQRT (X)
```

EL LADO IZQUIERDO SIEMPRE ES UNA VARIABLE, EL DERECHO PUEDE SER UNA CONSTANTE, VARIABLE O EXPRESION ARITMETICA.

PROPOSICIONES FORTRAN

LAS PROPOSICIONES SE CLASIFICAN EN EJECUTABLES Y NO EJECUTABLES:

A) NO EJECUTABLES

- BLOCK DATA
- COMMON
- DATA
- DIMENSION
- ENTRY
- EQUIVALENCE
- EXTERNAL
- FORMAT
- FUNCTION
- IMPLICIT
- INTRINSINC
- PARAMETER
- PROGRAM
- SAVE
- SUBROUTINE

B) EJECUTABLES

- ASSIGN
- BACKSPACE
- CALL
- CLOSE
- CONTINUE
- DECODE
- DEFINE FILE
- DO ELSE
- ELSE IF
- ENCODE
- END
- ENDFILE
- GO TO
- IF
- INQUIRE
- OPEN
- PAUSE
- PRINT
- READ
- RETURN
- REWIND
- STOP
- (WHILE)
- WRITE

PROPOSICIONES DE ENTRADA SALIDA

COMO SU NOMBRE LO INDICA SON INSTRUCCIONES PARA INTRODUCIR O SACAR INFORMACION DE LA COMPUTADORA.

. ENTRADA O LECTURA

READ (U, N) LISTA

. SALIDA O ESCRITURA

WRITE (U, N) LISTA

DONDE:

U Y N SON ENTEROS SIN SIGNO
U - ES EL TIPO DE PERIFERICO DE ENTRADA
N - ES EL NUMERO O ETIQUETA DEL POSTULADO RORMAT
LISTA - LISTA DE NOMBRES DE VARIABLE

. POSTULADO FORMAT

DESCRIBE EL FORMATO QUE TIENEN LAS VARIABLES A IMPRIMIR

N FORMAT (..., ..., ...)
N NUMERO ASOCIADO A UN READ O WRITE
DENTRO DEL PARENTESIS SE DESCRIBE EL FORMATO PARA CADA TIPO DE VARIABLE

. ENTERAS

N I W

. REALES

N F W.D
N E W.D

. CARACTERES ALFANUMERICOS

N A X

. PARA LETREROS

'ENTRE APOSTROFOS' O N H CARACTERES HOLLERIT

EJEMPLOS

```
      READ (2, 100) L, K, I, NOMBRE
100   FORMAT (2 I 6, I 4, 3 1 A 1)
      WRITE (6, 101) L, K, I, NOMBRE
101   FORMAT ( /, 10X, 'VALORES', 3I6, 31A1)
      READ (2,200) CANT, TOTAL, PI, SALD
200   FORMAT (F6.3, F8.2, F6.4, E9.4)
      WRITE (6,200) CANT, TOTAL, PI, SALD
```

L = 113000
K = 54321
I = +500

CANT = 32.787
TOTAL= 43591.40
PI = 3.1416
SALD = 4983E + 04

NOMBRE: JUAN JOSE PEREZ ARRECLA

PROPOSICIONES DE CONTROL

. DE SALTO

GO TO N

GO TO (N_1, N_2, N_3, ..., N_N), EXPRESION ARITMETICA

IF (ESP) N_1, N_2, N_3 (ARITMETICO)

IF (L) S (LOGICO)

DONDE LAS EXPRESIONES LOGICAS SON:

.LT.
.GT.
.LE.
.GE.
.EQ.
.NE.

Y LOS OPERADORES DE RELACION:

.OR.
.AND.
.NOT.

. DE REPETICION, CICLO, LAZO O BUCLE

DO N I = K_1, K_2, K_3

_____

_____

_____ INSTRUCCIONES

_____

_____

N CONTINUE O EXPRESION ARITMETICA

REPITE TODAS LAS INSTRUCCIONES HASTA N VARIANDO EL VALOR DE I DESDE K_1 A K_2 CON INCREMENTOS DE K_3 EN K_3.

EJEMPLO

```
      ISUM = 0
      DO 10 I = 1,100
1     ISUM = ISUM+I
      STOP
```

INSTRUCCIONES DE DELCARACION O ASIGNACION

- PARA RESERVAR ZONAS DE MEMORIA

  DIMENSION $A(N_1)$, $B(N_2)$, $MAT(N_3, N_4)$

  DONDE A, B Y MAT SON ARREGLOS Y MATRIZ RESPECTIVAMENTE N_1, N_2, N_3, Y N_4 SON SUBINDICES.

- PARA AREAS COMUNES DE ALMACENAMIENTO

  DIMENSION $A(N_1)$, $B(N_2)$, $MAT(N_3, N_4)$

  COMMON A, B, MAT

SUBPROGRAMAS

UN SUBPROGRAMA ES UN CONJUNTO DE INSTRUCCIONES ESCRITAS EN FORTRAN O LENGUAJE DE MAQUINA, QUE REALIZA UNA OPERACION ESPECIFICA COMO PARTE DE UN PROGRAMA PRINCIPAL

TIPOS DE SUBPROGRAMAS:

- FUNCIONES PROPIAS DEL COMPILADOR FORTRAN
- FUNCIONES DEL USUARIO
- SUBPROGRAMA FUNCTION
- SUBPROGRAMA SUBROUTINE

```
      READ (2, 100) N

      INTEGER N, T(100)

      READ (5, 100) N

      DO 10 L = 1, N

         READ(5,100) T (L)

   10 CONTINUE

      I = 1

   70 CONTINUE

      J = J - 1

   80 CONTINUE

      IF (T(I) ≤ T(J)) GO TO 120

         K = T(I)
         T(I) = T(J)
         T(J) = K

  120 CONTINUE

      IF (J = N) GO TO 150

          J = J + 1

      GOTO 70

      DO 30 L = 1 - 0 N
          WRITE (6,100) T(L)
   30 CONTINUE
      END
```

CENIET	PROGRAMA	PROGRAMADOR	PAGINA	FECHA
DISEÑO DE DATOS				

'C' PARA COMENTARIOS FORTRAN HOJA DE PROGRAMACION

NUMERO O ETIQUETA DE REF. | CON | ——— POSTULADO O INSTRUCCION FORTRAN | IDENTIFICACION

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

HISTORIA DE COBOL

AUNQUE MAS LENTAMENTE QUE EN EL CAMPO CIENTIFICO, LOS ORDENADORES ENTRARON EN EL AREA DE GESTION CON GRAN FUERZA, SOBRE TODO DEBIDO A LA APARICION DE LENGUAJES DE ALTO NIVEL ORIENTADOS ESPECIFICAMENTE A LOS NEGOCIOS. EL PRIMER LENGUAJE DE ESTE TIPO HISTORICAMENTE HABLANDO, FUE EL FLOW MATIC QUE EN 1955 ESTABLECIO EL CONCEPTO DE LENGUAJES DE PROGRAMACION BASADOS EN PALABRAS DE LENGUAJE NATURAL EN ESTE CASO EL INGLES . FUE CREADO POR EL DOCTOR HOPPER PARA UNIVAC. NO OBSTANTE, EL LENGUAJE DE GESTION QUE ALCANZO MAS RAPIDAMENTE LA POPULARIDAD FUE COBOL, DESARROLLADO A PARTIR DE 1959 POR CODASYL. CONOCIDO EN UN PRINCIPIO POR COBOL 60, PRETENDIA SER UN LENGUAJE COMUN A TODOS LOS ORDENADORES: POSTERIORMENTE HAN SURGIDO NUEVAS VERSIONES, POR EJEMPLO EL COBOL ANSI 79, EL COBOL-80 DE MICROSOFT, EL CISCOBOL QUE FACILITA EL MANEJO DE PANTALLAS Y EL RM-COBOL PARA MICROPROCESADORES .

VENTAJAS

1.- COMUN EN LA MAYORIA DE LAS COMPUTADORAS.

2.- POR ESTAR COMERCIALMENTE ORIENTADO, ES UN LENGUAJE CUYA ESTRUCTURA ES PARECIDA A LA DEL IDIOMA INGLES, POR LO TANTO, SUS CONSTRUCCIONES NO SE CODIFICAN CON CODIGOS COMPLEJOS, LO QUE PERMITE ENTRENAR PROGRAMADORES FACILMENTE.

3.- SUS PROGRAMAS SON AUTODOCUMENTADOS Y CUALQUIER PERSONA ENTRENADA EN COBOL PUEDE ENTENDERLOS.

CONSTANTES

LA FINALIDAD DE CREAR UN CAMPO CONSTANTE ES LA DE ALMACENAR VALORES QUE NO CAMBIEN DURANTE EL PROCESAMIENTO DE LA INFORMACION Y QUE NO DEPENDAN DE LA ENTRADA DE LA MISMA.

- CONSTANTES FIGURATIVAS
- CONSTANTES DEL USUARIO

VARIABLES

TIENEN POR OBJETO HABILITAR EL PROCESAMIENTO DE LA INFORMACION PARA CAMBIAR EL CONTENIDO DE LOS CAMPOS DE ACUERDO A LAS NECESIDADES DE PROGRAMACION. POR LO TANTO UNA VARIABLE ES AQUELLA CUYO CONTENIDO CAMBIA DENTRO DEL PROGRAMA

¿Cómo escribimos un programa en COBOL?

Existe una hoja de codificación diseñada especialmente para el lenguaje COBOL, su formato está basado en 80 posiciones de información distribuidas de la siguiente manera:

COLUMNAS:	SIGNIFICADO:
1 - 3	Número de página. Cada una contiene 25 renglones para escribir la secuencia del programa.
4 - 6	Número del renglón. Este se inicia en 10 y se asciende en rangos de 10 para poder intercalar posteriormente nuevas órdenes.
7	Columna de indicaciones al compilador:
	- (Guión) Continuación de títulos que no caben en un sólo renglón.
	* Dá ordenes al compilador de la clase de información contenida, por ejemplo, los renglones de comentarios que sólo son útiles a la documentación del programa.
	/ Brinco espaciado de renglones en la impresora del programa fuente.
8	Margen A. Los nombres de las divisiones, secciones y párrafos así como los niveles 01 y 77 deben de empezar en este margen.

12 Margen B. Los enunciados del programa, y las declaraciones empiezan en la columna 12 terminando el campo para redactar la información en la columna 72.

73 - 80 Identificación del programa. El nombre que se le asignó dentro de un proceso en particular.

dejando un blanco o más al final del renglón hasta la columna 72.

6. Cada renglón que se escribe en la hoja de codificación es perforado en una tarjeta.

¿Qué es un párrafo?

Un párrafo está construido de una o más oraciones que forman un conjunto lógico dentro del programa. Debe empezar siempre con un nombre el cual es llamado nombre de párrafo o etiqueta.

REGLAS GENERALES DE LOS PÁRRAFOS:

1. Un párrafo consta de una o más oraciones precedidas por un nombre de párrafo.

2. Los nombres de párrafo se codifican al margen A, o sea columna 8, terminan con un punto seguido de un espacio en blanco como mínimo.

3. Cada nuevo párrafo se inicia con otro nombre de párrafo o etiqueta y puede haber tantos párrafos como se necesite en la lógica del programa.

CAPITULO III

CONSTRUCCION DE UN PROGRAMA EN COBOL

REGLAS GENERALES DE LAS ORACIONES.

1. Una oración en COBOL está formada por operadores, literales, nombres de datos, palabras reservadas y verbos.

2. Las oraciones se crean de uno o más de los elementos anteriores, terminándolas con un punto, seguido de un espacio en blanco.

3. Una coma o un punto y coma, son usados como separadores entre los elementos de una oración, si son utilizados estos separadores, pueden o no estar precedidos por espacios en blanco; pero necesariamente deben estar seguidos de un espacio en blanco.

4. Las oraciones se codifican al margen B, o sea en la columna 12.

5. Cada oración debe estar escrita en un renglón; sin embargo, si una oración va a continuar en el siguiente renglón, debe ser dividida entre elementos completos

4. Cada oración que pertenece al párrafo debe empezar como mínimo en la columna 12, o sea, en el margen B, pudiendo estar escrita en el mismo renglón donde se escribió el nombre de párrafo, o bien, se puede optar por empezar a escribir las oraciones en el renglón siguiente.

TIPOS DE INSTRUCCIONES

Generalmente se dividen en dos grandes grupos que son:

1. EXPRESIONES.

1.1. Expresiones aritméticas. Están compuestas por operadores aritméticos combinados con nombres de datos y literales aritméticas, y su expresión es reducida a un valor numérico simple.

Ejemplo:

CALCULO - SUELDO * .08

Reglas para las expresiones aritméticas.

a) Los paréntesis redondos son utilizados para especificar el orden de los cálculos, si éstos aparecen, inician los cálculos a ejecutar.

b) Se calculan primero las exponenciaciones.

c) A continuación se ejecutan las multiplicaciones y divisiones que haya.

d) Y por último, se relizan las sumas y res-- tas.

e) Los operadores de relación definen la escritura de la condición.

Ejemplos:

```
X1= +B**2*SQRT((B**2-4*A*C)/2*A)
X2= -B**2*SQRT((B**2-4*A*C)/2*A)
NETO= SUELDO+HORASEXTRAS-ISR-IMSS-FALTAS
```

1.2. Expresiones condicionales. Se obtiene de ellas un valor verdadero o falso, que puede contener una o más condicio-- nes en su expresión, las cuales

pueden ser o no verdaderas. Así las expresiones condicionales pueden ser simples o compuestas según el número de condiciones que prueben. Se forman de la combinación del verbo IF con los operadores lógicos AND, OR y NOT y nombres de datos, nombres de condición, nombres de registro y de archivos.

Reglas para expresiones condicionales simples:

a) Están compuestas de una sola expresión.

b) Utilizan únicamente el operador lógico NOT, cuando es necesario para cumplir su función.

Reglas para expresiones condicionales compuestas:

a) Se combinan dos o más condiciones con un operador lógico OR o el operador AND.

b) El operador lógico AND se utiliza cuando se desea que las dos condiciones se cumplan.

c) El operador lógico OR se utiliza para elegir que una de las dos condiciones se cumplan.

d) Los paréntesis redondos se utilizan para agrupar las expresiones.

e) La evaluación de las condiciones se lleva a cabo bajo las siguientes reglas:

De izquierda a derecha para cada una de las condiciones que aparecen.

El operador AND es evaluado primero.

A continuación se evalúa el operador OR.

6) Los caracteres de relación definen la estructura de la condición.

Ejemplos:

Expresión simple:

IF CLAVE IS NOT EQUAL "X".....

IF A IS EQUAL TO B.....

Expresiones compuestas:

IF CLAVE IS EQUAL TO X AND RFC IS EQUAL TO RFC-WS

IF SUELDO IS LESS 1000 OR SUELDO IS EQUAL ZERO

PROPOSICIONES.

Una proposición consta de un verbo y sus operandos, los cuales hemos definido como sujeto emisor y receptor. Las hay de dos clases:

2.1. Las proposiciones imperativas son aquellas que no tienen opción a escoger.

Ejemplo:

```
ADD 1 TO CONTADOR
GO TO LEE
```

2.2. Las proposiciones condicionales permiten una opción a escoger de dos elecciones posibles. Se forman de la combinación del verbo IF con los caracteres de relación y los operandos.

DIVISIONES DE UN PROGRAMA.

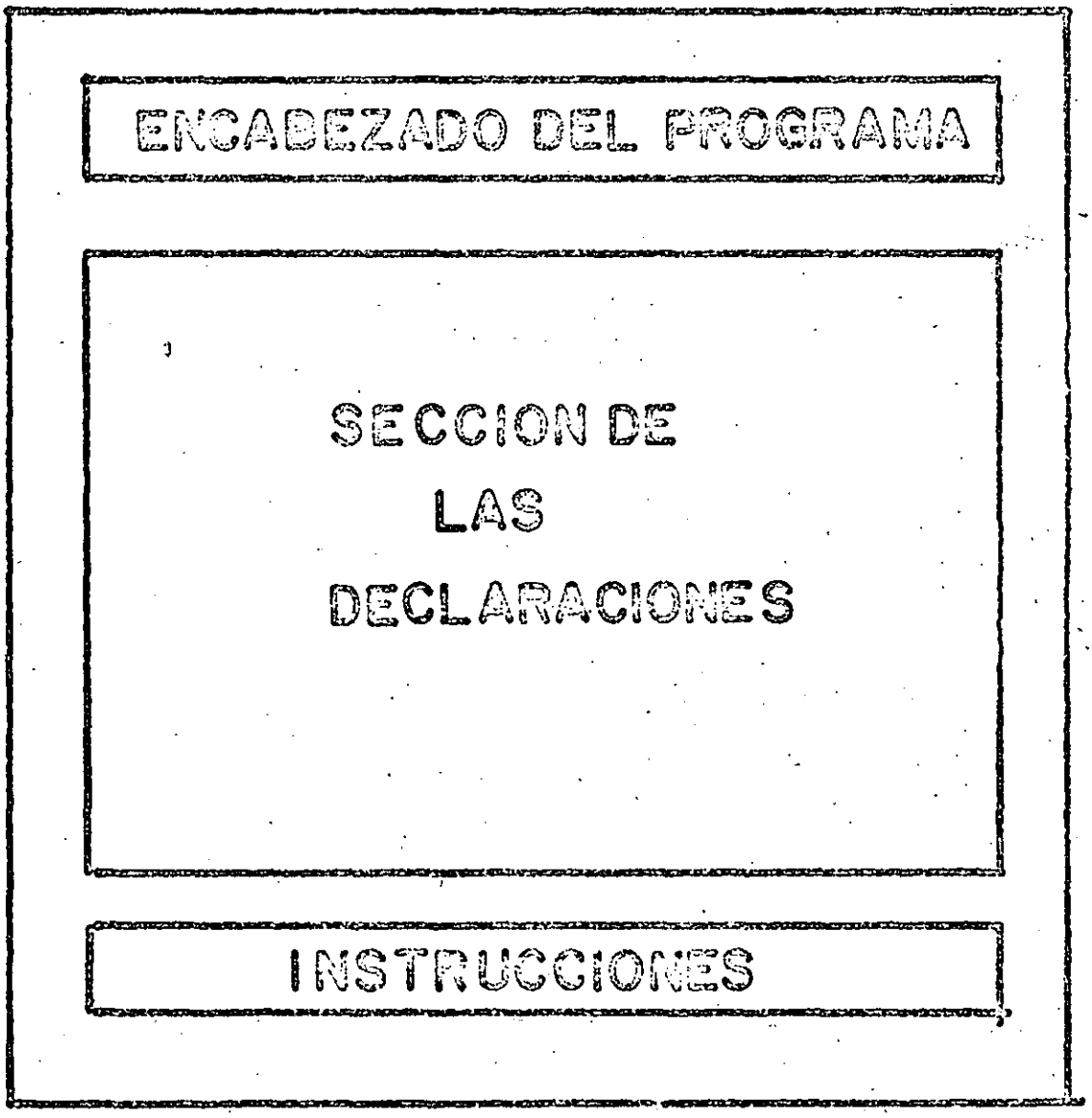
La estructura mínima de todo programa en COBOL, consta de cuatro grandes divisiones:

IDENTIFICATION DIVISION.	División de Identificación.
ENVIRONMENT DIVISION.	División del medio ambiente en que el programa se va a desarrollar.
DATA DIVISION.	División de Descripción y almacenamiento de datos.
PROCEDURE DIVISION.	División de Procedimientos.

Las cuales deben aparecer estrictamente en el orden en que se definieron.

P A S C A L

ESTRUCTURA DE UN PROGRAMA EN PASCAL



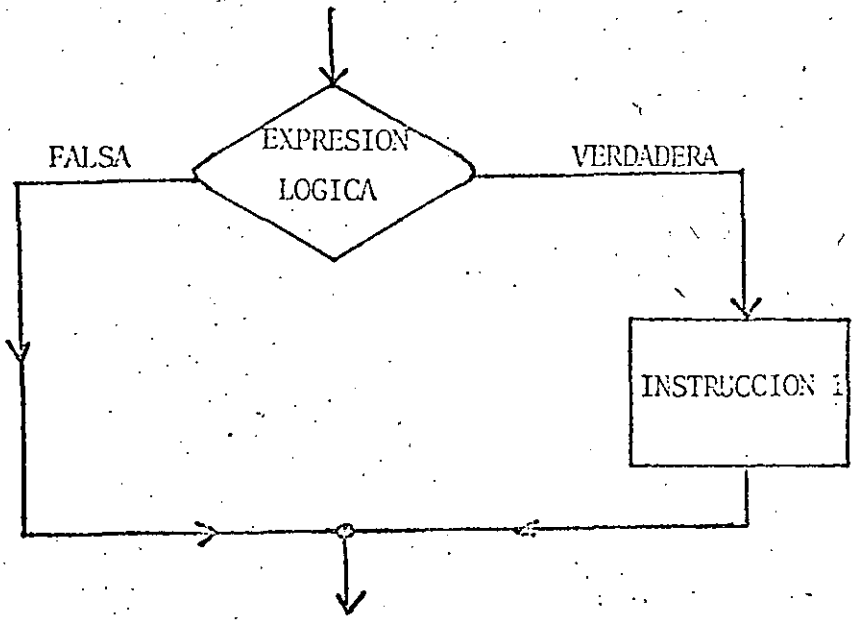
```
PROGRAM nombre(INPUT, OUTPUT);
CONST
     PI = 3.14159;
     LETRERO = 'hola';

TYPE
     COLOR = (AZUL, ROJO, BLANCO);

VAR
     INDICE : INTEGER;
     AREA :  REAL ;

BEGIN                          (*instrucciones*)
     INDICE := 107 ;
     AREA := 32.85;
     WRITE (LETRERO)
END.
```

TIPOS PREDEFINIDOS

- INTEGER
- BOOLEAN
- CHAR
- REAL

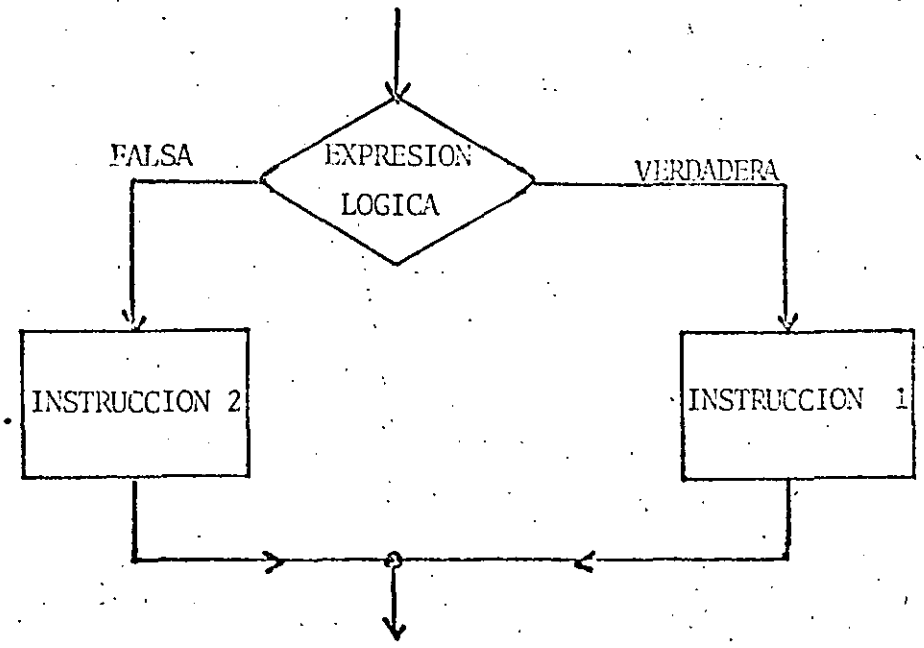
I N S T R U C C I O N IF

1. IF expresión lógica THEN
   instrucción 1
   (* ENDIF*);



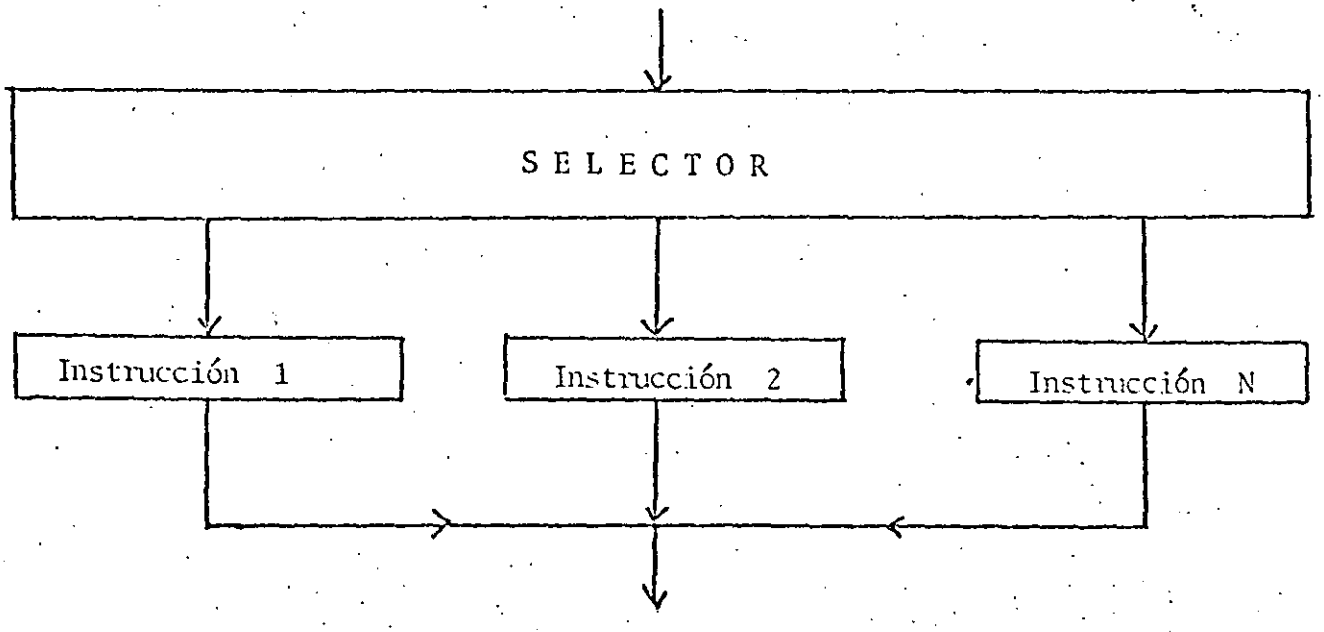
2. IF expresión lógica THEN
   insturcción 1
   ELSE
   instrucción 2

   (*ENDIF*);



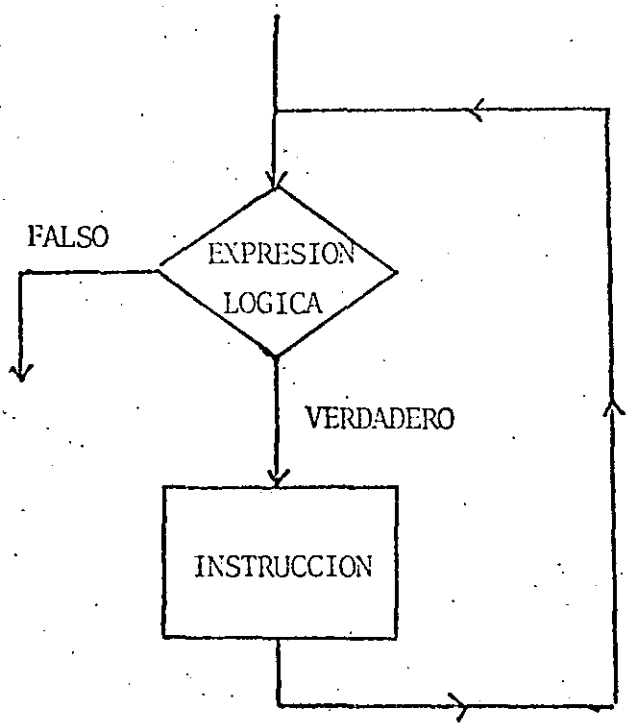
I N S T R U C C I O N CASE

```
CASE      selector    OF

      lista 1   :     instrucción 1;
      lista 2   :     instrucción 2;
        .
        .
        .
      lista N   :     instrucción N;

END;
```



I N S T R U C C I O N W H I L E

```
WHILE    expresión lógica    DO
         instrucción
(*END WHILE*);
```



I N S T R U C C I O N REPEAT

REPEAT

Instrucción 1;

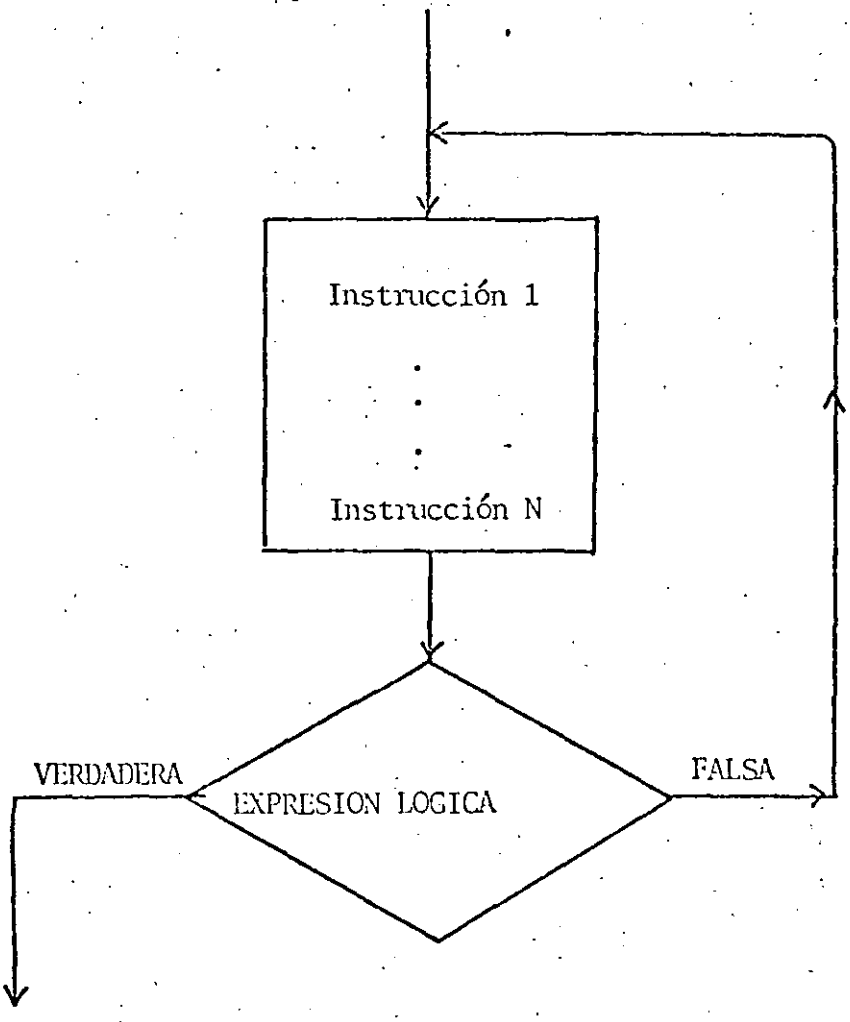
.

.

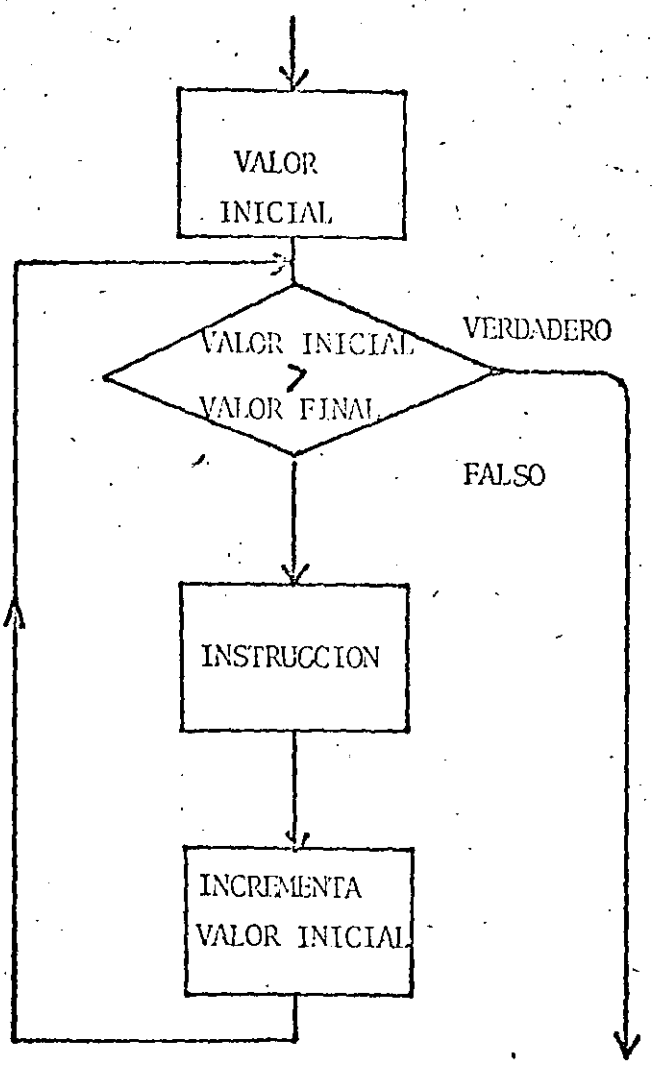
.

Instrucción N

UNTIL expresión lógica ;



I N S T R U C C I O N FOR



FOR VAR:= valor inicial TO valor final DO

Instrucción

(*ENDFOR*);

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA PROGRAMACION Y COMPUTACION
ELECTRONICA

FUNCIONES DEFINIDAS POR EL USUARIO

3 OCTUBRE, 1984.

12.2.- FUNCIONES DEFINIDAS POR EL USUARIO ①

El lenguaje FORTRAN proporciona una manera sencilla por medio de la cual un programador puede definir la función que él desee por medio de lo que se llama proposición de definición de función, siempre y cuando el cálculo que realice la función pueda escribirse como una expresión aritmética.

Para escribir una proposición de definición de función, deberá escribirse, primero, la forma en que el programador utilizará la función poniendo como argumentos el nombre de cualquier variable, después el signo igual (=), y después la expresión aritmética que el programador desee que ejecute la función cada vez que sea llamada, colocando la variable del argumento en el lugar adecuado dentro de la expresión aritmética.

Por ejemplo, suponga que desea definir una función que obtenga la tangente de un ángulo dividiendo el seno entre el coseno:

```
TAN (X) = SIN (X) / COS (X)
```

Una vez definida la función, el programador podrá utiliaarla como si fuera una función proporcionada por el compilador. P.E.

```
VAR = TAN (ANG)
XYZ = SQRT (TAN(A1/A2) ** 2 - 1)
```

Para definir correctamente una función, hay que tomar en cuenta lo siguiente:

- La proposición de definición de función deberá colocarse después de las declaraciones de variables, si hay, y antes de la primera proposición ejecutable del programa.

- Todos los argumentos (nombres de variables) que se utilicen a la izquierda del signo igual deberán utilizarse en la expresión aritmética que va a la derecha del signo igual. Sin embargo, sí se permite que en la expresión aritmética se utilice una variable que no sea argumento, en cuyo caso se referirá a la misma variable que se utilice en el programa. ②

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA

EXPRESION MATEMATICA

COMPLEMENTO

M. EN C. RICARDO CIRIA MERCE

OCTUBRE, 1984.

```
C
C     EJEMPLO 6.1          EXPRESION ARITMETICA
C
      PRINT *,'TECLEE EL NUMERO DE ALUMNOS DEL GRUPO '
      READ *,NUMERO
      PROGPO = 0.0
C
C     LA VARIABLE "INDICE" ES EL CONTADOR DE ALUMNOS PROCESADOS
C
      INDICE = 1
   10     PRINT *,'TECLEE LAS CINCO CALIFICACIONES DEL ALUMNO ',INDICE
          READ *,CALIF1,CALIF2,CALIF3,CALIF4,CALIF5
          PROALU = (CALIF1+CALIF2+CALIF3+CALIF4+CALIF5)/5.0
          PRINT *,'EL ALUMNO ',INDICE,' TIENE ',PROALU,' DE PROMEDIO'
          PROGPO = PROGPO + PROALU
          INDICE = INDICE + 1
C
C         EL SIGUIENTE "IF" PREGUNTA SI LA VARIABLE "INDICE" ANTES
C         INCREMENTADA, ES MENOR O IGUAL A LA VARIABLE "NUMERO" PARA
C         REGRESAR EL CONTROL A LA PROPOSICION CON LA ETIQUETA 10, O
C         CONTINUAR CON LA SIGUIENTE PROPOSICION (ETIQUETA NUMERO 20)
C
          IF ( INDICE-NUMERO ) 10,10,20
   20 PROGPO = PROGPO/NUMERO
      PRINT *,'EL PROMEDIO GENERAL DEL GRUPO ES: ',PROGPO
      STOP
      END
```

```
C      EJEMPLO 6.2          ARREGLOS
C
C      EL VECTOR 'CALIF' CONTENDRA LAS CALIFICACIONES DE CADA ALUMNO.
C      EL NUMERO DE CALIFICACIONES POR ALUMNO ES CONTROLADO POR
C      LA VARIABLE 'NUMCAL'; PERMITIENDOSE COMO MAXIMO
C      EL NUMERO DE ELEMENTOS EN EL VECTOR 'CALIF', O SEA, 10
C
       DIMENSION CALIF(10)
       PRINT *,'TECLEE EL NUMERO DE ALUMNOS DEL GRUPO '
       READ *,NUMALU
       PRINT *,'TECLEE EL NUMERO DE CALIFICACIONES POR ALUMNO '
       READ *,NUMCAL
       PROGPO = 0.0
C
C      LA VARIABLE 'INDALU' ES EL CONTADOR DE ALUMNOS
C
       INDALU = 1
   10      PRINT *,'TECLEE LAS ',NUMCAL,
     -             'CALIFICACIONES DEL ALUMNO ',INDALU
           PROALU = 0.0
C
C          LA VARIABLE 'INDCAL' ES EL CONTADOR DE
C          CALIFICACIONES POR CADA ALUMNO
C
           INDCAL = 1
   20          READ *,CALIF(INDCAL)
               PROALU = PROALU+CALIF(INDCAL)
               INDCAL = INDCAL+1
C
C              EL SIGUIENTE 'IF' CONTROLA LA LECTURA DE
C              LAS CALIFICACIONES DE CADA ALUMNO
C
               IF ( INDCAL-NUMCAL ) 20,20,30
   30      PROALU = PROALU/NUMCAL
           PRINT *,'EL ALUMNO ',INDALU,' TIENE ',PROALU,' DE PROMEDIO'
           PROGPO = PROGPO + PROALU
           INDALU = INDALU+1
C
C          EL SIGUIENTE 'IF' CONTROLA EL PROCESO DE CADA ALUMNO
C
           IF ( INDALU-NUMALU ) 10,10,40
   40  PROGPO = PROGPO/NUMALU
       PRINT *,'EL PROMEDIO GENERAL DEL GRUPO ES: ',PROGPO
       STOP
       END
```

```
C     EJEMPLO 6.3 -
C
C     EL VECTOR 'NOMBRE' CONTENDRA EL NOMBRE DE CADA ALUMNO.
C     SE GUARDARA UN CARACTER EN CADA ELEMENTO DEL VECTOR,
C     POR LO QUE SOLO PODRAN GUARDARSE NOMBRES DE 32 O MENOS CARACTERES
C
      DIMENSION CALIF(10),NOMBRE(32)
      WRITE(6,10)
   10 FORMAT(' TECLEE EL NUMERO DE ALUMNOS DEL GRUPO '(I2)')
      READ(5,20)NUMALU
   20 FORMAT(I2)
      WRITE(6,30)
   30 FORMAT(' TECLEE EL NUMERO DE CALIFICACIONES POR ALUMNO (I1)')
      READ(5,40)NUMCAL
   40 FORMAT(I1)
      PROGPO = 0.0
C
C     LA VARIABLE 'INDALU' ES EL CONTADOR DE ALUMNOS
C
      INDALU = 1
   50     WRITE(6,60)INDALU
   60     FORMAT(' TECLEE EL NOMBRE DEL ALUMNO NUMERO ',I2,'  (A32)')
          READ(5,70)NOMBRE
   70     FORMAT(32A1)
          WRITE(6,80)NUMCAL,NOMBRE
   80     FORMAT(' TECLEE LAS ',I1,' CALIFICACIONES DE: ',32A1,
     -           '  (F3.0)')
          PROALU = 0.0
C
C         LA VARIABLE 'INDCAL' ES EL CONTADOR DE
C         CALIFICACIONES POR CADA ALUMNO
C
          INDCAL = 1
   90         READ(5,100)CALIF(INDCAL)
  100         FORMAT(F3.0)
              PROALU = PROALU+CALIF(INDCAL)
              INDCAL = INDCAL+1
C
C             EL SIGUIENTE 'IF' CONTROLA LA LECTURA DE
C             LAS CALIFICACIONES DE CADA ALUMNO
C
              IF ( INDCAL-NUMCAL ) 90,90,110
  110     PROALU = PROALU/NUMCAL
          WRITE(6,120)INDALU,NOMBRE,PROALU
  120     FORMAT(1X,I2,'.- ',32A1,5X,F5.2)
          PROGPO = PROGPO + PROALU
          INDALU = INDALU+1
C
C         EL SIGUIENTE 'IF' CONTROLA EL PROCESO DE CADA ALUMNO
C
          IF ( INDALU-NUMALU ) 50,50,130
  130 PROGPO = PROGPO/NUMALU
      WRITE(6,140)PROGPO
  140 FORMAT(//,' PROMEDIO GENERAL DEL GRUPO: ',F5.2)
      STOP
      END
```

```
C     EJEMPLO 6.4          ITERACIONES CON LA PROPOSICION 'DO'
C
C     TODAS LAS ITERACIONES SE REALIZAN CON LA PROPOSICION 'DO'
C     INCLUYENDO UN 'DO' IMPLICITO PARA LA LECTURA DE LAS CALIFICACIONES
C
      DIMENSION CALIF(10),NOMBRE(32)
      WRITE(6,10)
   10 FORMAT(' TECLEE EL NUMERO DE ALUMNOS DEL GRUPO (I2)')
      READ(5,20)NUMALU
   20 FORMAT(I2)
      WRITE(6,30)
   30 FORMAT(' TECLEE EL NUMERO DE CALIFICACIONES POR ALUMNO (I1)')
      READ(5,40)NUMCAL
   40 FORMAT(I1)
      PROGPO = 0.0
C
C     EL SIGUIENTE 'DO' CONTROLA EL PROCESO DE CADA ALUMNO
C
      DO 110 INDALU=1,NUMALU
         WRITE(6,50)INDALU
   50    FORMAT(' TECLEE EL NOMBRE DEL ALUMNO NUMERO ',I2,'  (A32)')
         READ(5,60)NOMBRE
   60    FORMAT(32A1)
         WRITE(6,70)NUMCAL,NOMBRE,NUMCAL
   70    FORMAT(' TECLEE LAS ',I1,' CALIFICACIONES DE: ',32A1,
     -        '  (',I1,'F3.0)')
C
C        EL SIGUIENTE 'READ' LEE DE UNA SOLA VEZ TODAS LAS
C        CALIFICACIONES POR MEDIO DE UN 'DO' IMPLICITO DE LECTURA,
C        POR LO QUE LOS DATOS DEBERAN ESTAR TODOS EN LA MISMA LINEA
C
         READ(5,80) ( CALIF(INDCAL),INDCAL=1,NUMCAL )
   80    FORMAT(9F3.0)
         PROALU = 0.0
C
C        EL SIGUIENTE 'DO' ACUMULA TODAS LA CALIFICACIONES
C        EN LA VARIABLE 'PROALU'
C
         DO 90 INDCAL=1,NUMCAL
            PROALU = PROALU+CALIF(INDCAL)
   90    CONTINUE
         PROALU = PROALU/NUMCAL
         WRITE(6,100)INDALU,NOMBRE,PROALU
  100    FORMAT(1X,I2,'.- ',32A1,5X,F5.2)
         PROGPO = PROGPO + PROALU
  110 CONTINUE
      PROGPO = PROGPO/NUMALU
      WRITE(6,120)PROGPO
  120 FORMAT(//,' PROMEDIO GENERAL DEL GRUPO: ',F5.2)
      STOP
      END
```

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA

IF ARITMETICO

COMPLEMENTO

ING. HERIBERTO OLGUÍN ROMO

OCTUBRE, 1984

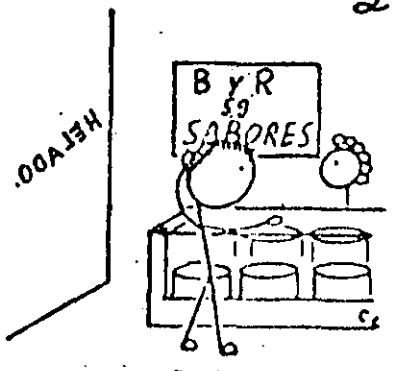
OBJETIVO: Construir y usar la proposición IF aritmética. 1

La proposición IF aritmética permite tomar tres caminos alternativos, según una decisión. A continuación se da la forma general de la proposición IF aritmética.

FORMA GENERAL	
1 2 3 4 5 6 7 8 9 ...	
nnnnn	IF(e) m_1,m_2,m_3

COMPONENTES	
nnnnn	es el número de la proposición
IF	es la palabra clave
e	es el argumento de la proposición IF y puede ser cualquier expresión aritmética válida
m_1	es el número de la proposición a la cual se efectuará la transferencia si e<0
m_2	es el número de la proposición a la cual se efectuará la transferencia si e=0
m_3	es el número de la proposición a la cual se efectuará la transferencia si e>0

EJEMPLOS	
1 2 3 4 5 6 7 8 9 ...	
	IF(A-1.0)5,10,20
	IF(A**2-B**2)5,5,10



Decisión

La e es cualquier expresión aritmética válida. Una expresión aritmética es una secuencia de constantes numéricas y/o de variables conectadas por los operadores aritméticos, como, por ejemplo X o (X + 2.*Y − 4). La expresión aritmética tiene un valor numérico; este valor puede ser menor que cero, exactamente igual a cero o mayor que cero. Esta condición determina la proposición que se ha de ejecutar después de la IF. Se hace la transferencia a la proposición m_1 si e < 0, a la proposición m_2 si e = 0, y a la proposición m_3 si e > 0. Esto se muestra en el diagrama de flujo de la fig.

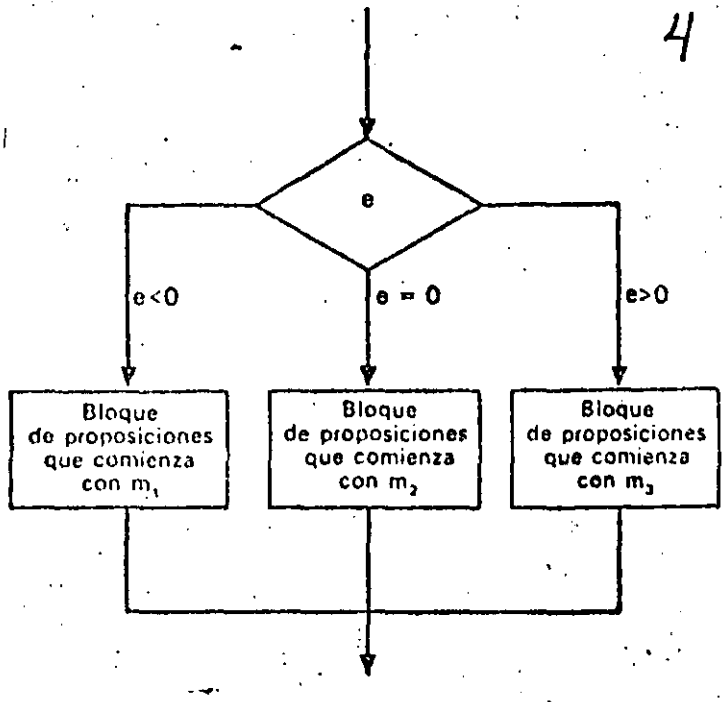


Diagrama de flujo para la proposición IF aritmética.

A continuación se da un ejemplo de la proposición IF aritmética.

JEMPLO 1

Se desea escribir un programa para extraer las raíces de la ecuación cuadrada $ax^2 + bx + c$ utilizando la fórmula

$$R_1, R_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Las siguientes son las proposiciones necesarias para calcular el radicando $(b^2 - 4ac)$ y transferir el control al segmento de programa identificado por proposición 10 si $(b^2 - 4ac) < 0$, a la proposición 20 si $(b^2 - 4ac) > 0$ a la proposición 15 si $(b^2 - 4ac) = 0$.

```
      IF(B**2-4.*A*C)10,15,20
   10 ...
   15 ...
   20 ...
```

La proposición que sigue a una IF aritmética siempre debe tener un número de proposición.

Para «saltar» los segmentos de programa no deseados se utiliza la proposición GO TO, como se muestra en la próxima solución más completa de la ecuación cuadrática.

JEMPLO 2

```
      ...
      DISC=B*B-4.*A*C
      IF(DISC)10,15,20
   10 WRITE(3,90)
   90 FØRMAT('NØ HAY RAICES REALES')
      GØ TØ 25
   15 X1=-B/(2.*A)
      X2=X1
      GØ TØ 22
   20 X1=(-B+SQRT(DISC))/(2.*A)
      X2=(-B-SQRT(DISC))/(2.*A)
   22 WRITE(3,92)X1,X2
   92 FØRMAT('bLAS RAICES REALES SØN', 2F10.1)
   25 CØNTINUE
```

EJEMPLO 3

Escribir proposiciones que sumen los enteros positivos de uno a 100.

```
      I=1
      ISUM=0
    8 ISUM=ISUM+1
      I=I+1
      IF(I-100)8,8,6
    6 CØNTINUE
```

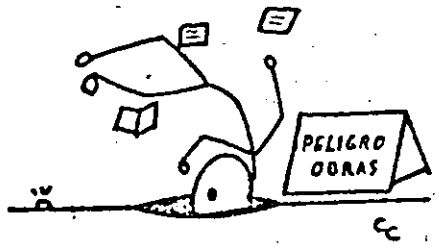
Escribir un ciclo propio es especialmente ventajoso cuando no hay ningún «conteo» asociado naturalmente al ciclo y cuando los valores iniciales, terminales y de incremento no son enteros.

El ejemplo 4 ilustra tal situación.

EJEMPLO 4

Escribir proposiciones que resuelvan $y = \text{sen}x + \cos x$ para valores de x de -10 a 10 radianes en incrementos de 0.1.

```
      X=-10.1
    5 X=X+.1
      IF(X-10.)6,6,10
    6 Y=SIN(X) + CØS(X)
      WRITE(3,90)X,Y
      GØ TØ 5
   10 CONTINUE
   90 FØRMAT('b',2E13.5)
      ...
```



Caida

el ejemplo 5, el valor de la expresión debe ser exactamente cero para a trasferencia de la proposición 20.

EMPLO 5

Este programa hace que el computador entre en un ciclo sin fin, ya que el

```
      A=0
    5 A=A+.1
      Y=A**3
      WRITE(3,10)A,Y
   10 FØRMAT('0',2F7.1)
      IF(A-1.0)5,20,5
   20 CØNTINUE
```

*TP**

Inspeccionar todos los ciclos iterativos para prevenir los ciclos iterativos sin fin.

- **¿tiene el ciclo iterativo una condición de salida?**
- **cuidado con los «iguales» como una condición con los reales. Pruebe en condiciones «no iguales» o use los enteros como prueba.**

* Tácticas preventivas.

or de (A-1.0) jamás será exactamente cero. Situaciones como esta pueden tarse si no se basa nunca la prueba en «iguales», sino en «menos que o ales» o «mayor que o iguales» al utilizar números reales.
La proposición If en el ejemplo anterior tendrá la forma

```
      IF(A-1.0)5,20,20
```

que A siempre excederá a 1.0 en algún punto,

Otro método para evitar el ciclo sin fin es usar aritmética entera o «exacta», no se muestra en el ejemplo 6.

EMPLO 6

```
      I=0
    5 I=I+1
      A=SIN(FLØAT(I)/3.)
      Y=A**3
      WRITE(3,10)A,Y
   10 FØRMAT('0',2E12.5)
      IF(I-10)5,20,5
   20 CØNTINUE
```

¡No entrar a menos que haya salida!

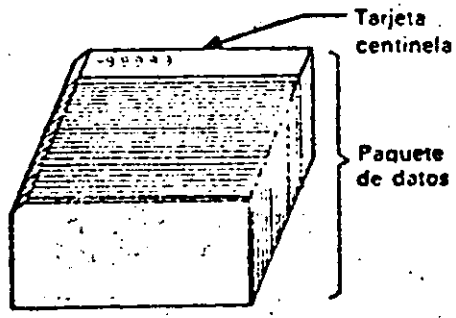
El ejemplo siguiente muestra cómo crear su propio ciclo para la unidad de procesamiento (ver al ejemplo de la sección 10.3) sin usar la opción END=. Un procedimiento para manejar la condición de fin de datos es usar como la última de la serie de tarjetas de datos una *tarjeta centinela*. Esta tarjeta contiene algún valor, tal como 9999, en un campo no usado o un valor fuera de los límites de los valores posibles para uno de los datos leídos. Inmediatamente después de haberse leído el valor del dato, se prueba el campo que contiene el indicador para el fin de datos. La proposición IF aritmética provee un medio para hacer esto.

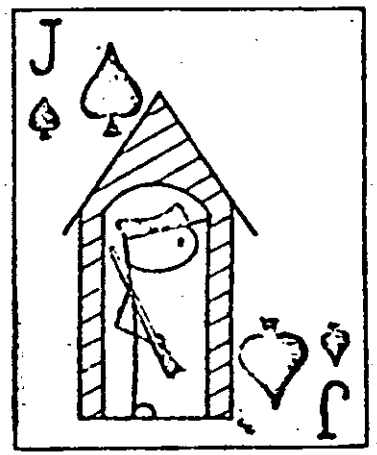
EJEMPLO 7

Entrar en la memoria todas las tarjetas de identidad de los alumnos e imprimir la información que contienen. El primer campo en cada tarjeta contiene el número de identidad del alumno, un entero entre 100000 y 999999. Puesto que cualquier número negativo es inválido, seleccionamos en forma arbitraria −99999 y lo perforamos en el campo correspondiente al número de identidad del alumno. (Se podría usar cualquier número negativo o simplemente una tarjeta en blanco.) Esta tarjeta centinela se ubica después de la última tarjeta válida de datos.

El programa lee cada tarjeta, comprueba si el campo de números de identidad del alumno contiene un número negativo y transfiere a STOP cuando lo encuentra. Esto se muestra a continuación.

```
      READ(1,90)ID,CLASS,GRPT
   90 FØRMAT(I6,4X,A2,4X,F10.1)
      IF(ID)99,99,10
   10 ...procesar la tarjeta
      GØ TØ 5
   99 STØP
```

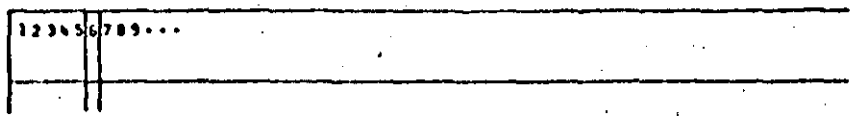




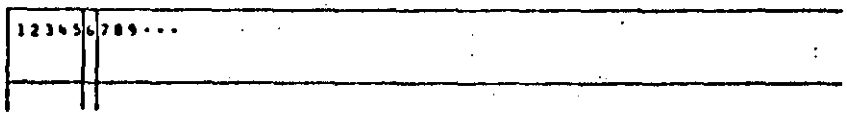
Tarjeta centinela

EJERCICIOS

1. Escribir una proposición IF aritmética que transfiera el control a la proposición 10 si A es menor que 25, a la proposición 20 si A es igual a 25 y a la proposición 30 si A es mayor que 25.



2. Escribir una proposición IF aritmética que transfiera el control a la proposición 10, 20 ó 30, dependiendo de si IMO < 0, IMO = 0 o IMO > 0, respectivamente.



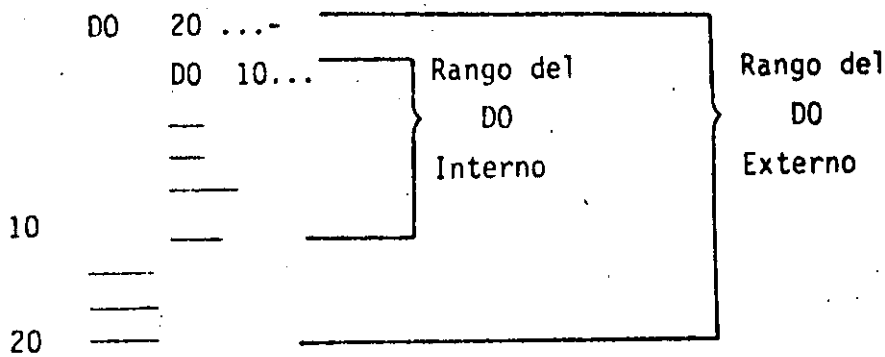
INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA

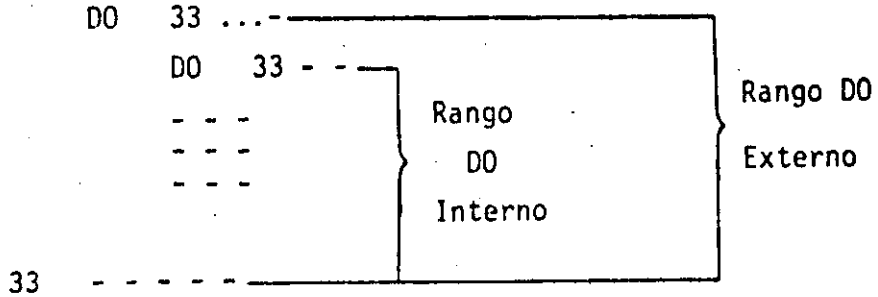
A N I D A M I E N T O

OCTUBRE, 1984.

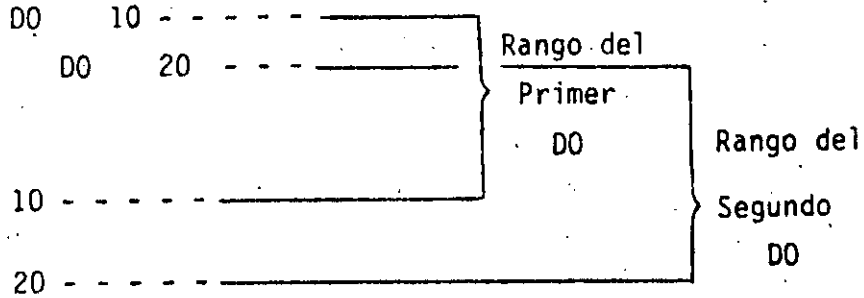
10.2.- ANIDAMIENTO

Hay ocasiones en las que es necesario utilizar una proporción DO dentro del rango de otra proposición DO, a lo cual se le llama - anidamiento . Al utilizar un anidamiento de proposiciones DO hay que tener cuidado con lo siguiente: el final de la proposición DO interna debe estar antes, o cuando mucho, en la misma proposición que en la que termina la proposición DO externa, es decir, el rango de la proposición DO interna (ó anidada) debe estar completamente contenido dentro del rango del DO externo; veamos unos ejemplos:





Lo siguiente es incorrecto:



No hay límites en el nivel de anidamiento, es decir, en el número de DO que se pueden anidar dentro de otro: sin embargo, si hay muchos - DO anidados el programa puede parecer confuso, por lo que se recomien da usar siempre la proposición CONTINUE que se explica en seguida.

```
C
C     EJEMPLO 6.1          EXPRESION ARITMETICA
C
      PRINT *,'TECLEE EL NUMERO DE ALUMNOS DEL GRUPO '
      READ *,NUMERO
      PROGPO = 0.0
C
C     LA VARIABLE "INDICE" ES EL CONTADOR DE ALUMNOS PROCESADOS
C
      INDICE = 1
   10     PRINT *,'TECLEE LAS CINCO CALIFICACIONES DEL ALUMNO ',INDICE
          READ *,CALIF1,CALIF2,CALIF3,CALIF4,CALIF5
          PROALU = (CALIF1+CALIF2+CALIF3+CALIF4+CALIF5)/5.0
          PRINT *,'EL ALUMNO ',INDICE,' TIENE ',PROALU,' DE PROMEDIO'
          PROGPO = PROGPO + PROALU
          INDICE = INDICE + 1
C
C         EL SIGUIENTE "IF" PREGUNTA SI LA VARIABLE "INDICE" ANTES
C         INCREMENTADA, ES MENOR O IGUAL A LA VARIABLE "NUMERO" PARA
C         REGRESAR EL CONTROL A LA PROPOSICION CON LA ETIQUETA 10, O
C         CONTINUAR CON LA SIGUIENTE PROPOSICION (ETIQUETA NUMERO 20)
C
          IF ( INDICE-NUMERO ) 10,10,20
   20 PROGPO = PROGPO/NUMERO
      PRINT *,'EL PROMEDIO GENERAL DEL GRUPO ES: ',PROGPO
      STOP
      END
```

```
C       EJEMPLO 6.2             ARREGLOS
C
C       EL VECTOR 'CALIF' CONTENDRA LAS CALIFICACIONES DE CADA ALUMNO;
C       EL NUMERO DE CALIFICACIONES POR ALUMNO ES CONTROLADO POR
C       LA VARIABLE 'NUMCAL', PERMITIENDOSE COMO MAXIMO
C       EL NUMERO DE ELEMENTOS EN EL VECTOR 'CALIF', O SEA, 10
C
        DIMENSION CALIF(10)
        PRINT *,'TECLEE EL NUMERO DE ALUMNOS DEL GRUPO '
        READ *,NUMALU
        PRINT *,'TECLEE EL NUMERO DE CALIFICACIONES POR ALUMNO '
        READ *,NUMCAL
        PROGPO = 0.0
C
C       LA VARIABLE 'INDALU' ES EL CONTADOR DE ALUMNOS
C
        INDALU = 1
   10   PRINT *,'TECLEE LAS ',NUMCAL,
     -          'CALIFICACIONES DEL ALUMNO ',INDALU
        PROALU = 0.0
C
C       LA VARIABLE 'INDCAL' ES EL CONTADOR DE
C       CALIFICACIONES POR CADA ALUMNO
C
        INDCAL = 1
   20       READ *,CALIF(INDCAL)
            PROALU = PROALU+CALIF(INDCAL)
            INDCAL = INDCAL+1
C
C           EL SIGUIENTE 'IF' CONTROLA LA LECTURA DE
C           LAS CALIFICACIONES DE CADA ALUMNO
C
            IF ( INDCAL-NUMCAL ) 20,20,30
   30   PROALU = PROALU/NUMCAL
        PRINT *,'EL ALUMNO ',INDALU,' TIENE ',PROALU,' DE PROMEDIO'
        PROGPO = PROGPO + PROALU
        INDALU = INDALU+1
C
C       EL SIGUIENTE 'IF' CONTROLA EL PROCESO DE CADA ALUMNO
C
        IF ( INDALU-NUMALU ) 10,10,40
   40 PROGPO = PROGPO/NUMALU
      PRINT *,'EL PROMEDIO GENERAL DEL GRUPO ES: ',PROGPO
      STOP
      END
```

```
C     EJEMPLO 6.3          FORMATOS, DATOS ALFANUMERICOS
C
C     EL VECTOR 'NOMBRE' CONTENDRA EL NOMBRE DE CADA ALUMNO.
C     SE GUARDARA UN CARACTER EN CADA ELEMENTO DEL VECTOR,
C     POR LO QUE SOLO PODRAN GUARDARSE NOMBRES DE 32 O MENOS CARACTERES
C
      DIMENSION CALIF(10),NOMBRE(32)
      WRITE(6,10)
   10 FORMAT(' TECLEE EL NUMERO DE ALUMNOS DEL GRUPO (I2)')
      READ(5,20)NUMALU
   20 FORMAT(I2)
      WRITE(6,30)
   30 FORMAT(' TECLEE EL NUMERO DE CALIFICACIONES POR ALUMNO (I1)')
      READ(5,40)NUMCAL
   40 FORMAT(I1)
      PROGPO = 0.0
C
C     LA VARIABLE 'INDALU' ES EL CONTADOR DE ALUMNOS
C
      INDALU = 1
   50    WRITE(6,60)INDALU
   60    FORMAT(' TECLEE EL NOMBRE DEL ALUMNO NUMERO ',I2,'  (A32)')
         READ(5,70)NOMBRE
   70    FORMAT(32A1)
         WRITE(6,80)NUMCAL,NOMBRE
   80    FORMAT(' TECLEE LAS ',I1,' CALIFICACIONES DE: ',32A1,
     -          '  (F3.0)')
         PROALU = 0.0
C
C        LA VARIABLE 'INDCAL' ES EL CONTADOR DE
C        CALIFICACIONES POR CADA ALUMNO
C
         INDCAL = 1
   90       READ(5,100)CALIF(INDCAL)
  100       FORMAT(F3.0)
            PROALU = PROALU+CALIF(INDCAL)
            INDCAL = INDCAL+1
C
C           EL SIGUIENTE 'IF' CONTROLA LA LECTURA DE
C           LAS CALIFICACIONES DE CADA ALUMNO.
C
            IF ( INDCAL-NUMCAL ) 90,90,110
  110    PROALU = PROALU/NUMCAL
         WRITE(6,120)INDALU,NOMBRE,PROALU
  120    FORMAT(1X,I2,'.- ',32A1,5X,F5.2)
         PROGPO = PROGPO + PROALU
         INDALU = INDALU+1
C
C        EL SIGUIENTE 'IF' CONTROLA EL PROCESO DE CADA ALUMNO
C
         IF ( INDALU-NUMALU ) 50,50,130
  130 PROGPO = PROGPO/NUMALU
      WRITE(6,140)PROGPO
  140 FORMAT(//,' PROMEDIO GENERAL DEL GRUPO: ',F5.2)
      STOP
      END
```

```
C     EJEMPLO 6.4          ITERACIONES CON LA PROPOSICION "DO"  (6)
C
C     TODAS LAS ITERACIONES SE REALIZAN CON LA PROPOSICION "DO"
C     INCLUYENDO UN "DO" IMPLICITO PARA LA LECTURA DE LAS CALIFICACIONES
C
      DIMENSION CALIF(10),NOMBRE(32)
      WRITE(6,10)
   10 FORMAT(' TECLEE EL NUMERO DE ALUMNOS DEL GRUPO (I2)')
      READ(5,20)NUMALU
   20 FORMAT(I2)
      WRITE(6,30)
   30 FORMAT(' TECLEE EL NUMERO DE CALIFICACIONES POR ALUMNO (I1)')
      READ(5,40)NUMCAL
   40 FORMAT(I1)
      PROGPO = 0.0
C
C     EL SIGUIENTE "DO" CONTROLA EL PROCESO DE CADA ALUMNO
C
      DO 110 INDALU=1,NUMALU
         WRITE(6,50)INDALU
   50    FORMAT(' TECLEE EL NOMBRE DEL ALUMNO NUMERO ',I2,'  (A32)')
         READ(5,60)NOMBRE
   60    FORMAT(32A1)
         WRITE(6,70)NUMCAL,NOMBRE,NUMCAL
   70    FORMAT(' TECLEE LAS ',I1,' CALIFICACIONES DE: ',32A1,
     -          '  (',I1,'F3.0)')
C
C        EL SIGUIENTE "READ" LEE DE UNA SOLA VEZ TODAS LAS
C        CALIFICACIONES POR MEDIO DE UN "DO" IMPLICITO DE LECTURA,
C        POR LO QUE LOS DATOS DEBERAN ESTAR TODOS EN LA MISMA LINEA
C
         READ(5,80) ( CALIF(INDCAL),INDCAL=1,NUMCAL )
   80    FORMAT(9F3.0)
         PROALU = 0.0
C
C        EL SIGUIENTE "DO" ACUMULA TODAS LA CALIFICACIONES
C        EN LA VARIABLE "PROALU"
C
         DO 90 INDCAL=1,NUMCAL
            PROALU = PROALU+CALIF(INDCAL)
   90    CONTINUE
         PROALU = PROALU/NUMCAL
         WRITE(6,100)INDALU,NOMBRE,PROALU
  100    FORMAT(1X,I2,'.- ',32A1,5X,F5.2)
         PROGPO = PROGPO + PROALU
  110 CONTINUE
      PROGPO = PROGPO/NUMALU
      WRITE(6,120)PROGPO
  120 FORMAT(//,' PROMEDIO GENERAL DEL GRUPO: ',F5.2)
      STOP
      END
```

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA PROGRAMACION Y
COMPUTACION ELECTRONICA

BREVE HISTORIA DEL LENGUAJE

OCTUBRE, 1984.

Palacio de Minería Calle de Tacuba 5 primer piso . Deleg. Cuauhtemoc 06000 México, D.F. Tel.: 521-40-20 Apdo. Postal M-2285

2.- BREVE HISTORIA DEL LENGUAJE

La introducción en el mercado de las computadoras de programas almacenados permitió el nacimiento de una nueva profesión, el programador de computadoras. Desde entonces, se han producido significativos avances en el campo de la programación de computadoras, específicamente en el desarrollo de técnicas para hacer menos dificultoso este trabajo para el programador humano.

Muchos autores reconocen que el honor de haber sido el primer programador (mejor dicho, la primera) corresponde a una simpática dama que murió casi un siglo antes de que apareciera la primera computadora de programas almacenados. Ada Augusta, condesa de Lovelace, quien viviera una vida extraordinaria. Nació en 1815; fué uno de los muchos descendientes del prolífico poeta inglés Lord Byron. Unos pocos meses después de su nacimiento, sus padres se separaron y ella no volvió a ver a su padre nunca más.

Por las fechas de su matrimonio con el conde de Lovelace, en -- 1835, Ada Augusta se relacionó con Charles Babbage, quien estaba en ese momento empezando su proyecto de la máquina analítica. Poseedora de aptitudes para las matemáticas y el pensamiento mecánico, se ofreció a trabajar con Babbage en su proyecto y, en 1842, tradujo del inglés una primera descripción al italiano de la máquina, añadiendo muchas notas de su cosecha. Se refirió a "ciclos de operación" y al repetido uso de las tarjetas como estructuras del tipo de subrutinas y se refirió también a la computación no numérica y a la manipulación simbólica. Observó que la máquina analpitica no "originaba nada" y que sólo podía hacer "aquello que uno sabía cómo ordenarle que realizara". Una de sus notas fué una descripción detallada para calcular los números de Bernoulli con la máquina analítica, que para muchos fué el primer "programa". Ada y Charles trabajaron juntos posteriormente en una treta para aplicar la máquina analítica al problema de los montos en las carreras de caballos. Durante el resto de su vida Ada se dedicó al juego dilapidando una parte considerable de la fortuna de Lovelace. Murió de cáncer en 1852.

Al igual que los idiomas sirven de vehículo de comunicación entre los seres humanos, existen lenguajes que realizan la comunicación entre los seres humanos y las computadoras. Estos lenguajes permiten expresar los programas o el conjunto de instrucciones que el operador humano desea que la computadora ejecute. Los lenguajes de computadora toman diferentes formas; los de las primeras, como la ENIAC y la EDSAC se componían en el lenguaje real de las máquinas mismas. En lenguaje de máquina, las instrucciones se expresan simplemente como una serie de dígitos binarios, o bits (binary digits). La dificultad de programar las máquinas primitivas de esta manera limitaba drásticamente su utilidad y proporcionaba un fuerte incentivo para que se desarrollaran lenguajes de programación más orientados hacia la expresión de soluciones con la notación de los problemas mismos. Un programa especialmente diseñado podía entonces, ser ejecutado para que realizara la traducción al lenguaje real usado por la máquina.

Los primeros lenguajes de programación se conocieron como lenguajes ensambladores, un ejemplo de los cuales es TRANSCODE, desarrollado para la computadora FERUT de la Universidad de Toronto por Pat Hume y Beatrice Worsley. En los lenguajes ensambladores se define un código especial (llamado mnemónico) para cada una de las operaciones de la máquina y se introduce una notación especial para especificar el dato con el cual debe realizarse tal operación. Un programa especial, denominado ensamblador, traduce las instrucciones simbólicas del lenguaje a las instrucciones de máquina necesarias para que sean ejecutadas. Los lenguajes ensambladores son todavía muy populares en ciertas aplicaciones; a pesar de que se ha avanzado notablemente en los lenguajes de programación de máquina, esto no basta para satisfacer las necesidades de todo lo que el programador desea hacer.

A mediados del decenio 1950-1960 aparecieron los primeros lenguajes de programación de propósito general, uno de los cuáles revolucionó muy pronto el campo de la programación. Se llamó FORTRAN (FORmula TRANSlating system) y fué publicado en 1954. El líder del proyecto FORTRAN fué John Backus, quien trabajó para la IBM y desarrolló un método formal para definir la sintaxis de los lenguajes de programación, la Forma Backus-Naur o BNF. FORTRAN fué realizado en 1957, con nuevas versiones que fueron apareciendo en 1958, 1960 y 1962, la última de las cuales se conoció con el nombre de FORTRAN IV. Se trata de un lenguaje dirigido a la solución numérica de problemas científicos; es fácil de entender, leer y escribir. Con el FORTRAN, el usuario está capacitado de inmediato para escribir un programa aunque sepa muy poco acerca de las características físicas de la máquina en la cual el programa va a ser ejecutado. Sin duda, el lenguaje es independiente de la máquina y en teoría, a pesar de algunas dificultades prácticas, los programas FORTRAN escritos para una máquina deben ser fáciles de transferir a otra. No ocurre lo mismo con los programas escritos en ensamblador o en código de máquina. El más fuerte impacto que tuvo el lenguaje FORTRAN en la industria de las computadoras se debió a que permitía a los usuarios programar sus propias soluciones, sin necesidad de recurrir a la ayuda de un programador profesional.

En un comienzo, FORTRAN no fué totalmente aceptado a causa, sobre todo, del temor que inspiraban sus altos costos de traducción. Al contrario de los ensambladores, los lenguajes de alto nivel, a causa de su generalidad, requieren traductores más complejos conocidos como compiladores, que por su propia complejidad son también más costosos de ejecutar. A pesar de estos problemas, su empleo aumentó y con el paso de los años los costos de compilación han sido reducidos sustancialmente, por lo que en la actualidad FORTRAN ha llegado a ser el lenguaje de programación más ampliamente utilizado en el mundo, y también un factor muy importante en el cada vez más difundido uso de las computadoras. La aparición de compiladores rápidos orientados al uso de los estudiantes, como PUFFT, desarrollado en la Universidad de Purdue, y WATFOR y WATFIV, desarrollados en la Universidad de Waterloo, han

hecho que la enseñanza de la programación con FORTRAN sea más simple; ello ha permitido que la computación misma llegue a un mayor número de estudiantes.

Otros lenguajes de programación han seguido rápidamente los pasos de FORTRAN; el lenguaje ALGOritimico, ALGOL, fué diseñado por un comité internacional en 1958 y revisado en 1960. Se trata de un lenguaje muy efectivo para resolver una amplia variedad de problemas con aplicaciones en matemáticas numéricas, pero no es adecuado (al igual que FORTRAN) para manejar datos no numéricos. Aún hoy, ALGOL es más popular en Europa que en Norteamérica.

Tanto FORTRAN como ALGOL están dirigidos básicamente a la computación científica; en mayo de 1959, el Departamento de Defensa de Estados Unidos convocó a una reunión para discutir el problema de desarrollar un lenguaje común para aplicaciones en negocios. Respondieron al llamado cerca de 40 representantes de los usuarios, de las dependencias del Gobierno, de los fabricantes de computadoras y de otras partes interesadas. La versión inicial de COBOL (COmmon Business Oriented Language) apareció en diciembre de 1959.

Los objetivos de COBOL consideraban la expresión natural de los programas (es decir, en inglés), lo que permitiría el aprendizaje fácil del lenguaje, la amplia documentación del mismo y la independencia de la máquina, lo cual facilitaría la transferencia de los programas de COBOL de una instalación a otra. A pesar de que las especificaciones de COBOL han sido revisadas varias veces desde su primera versión, el lenguaje mismo ha permanecido esencialmente sin cambios. En la actualidad se utiliza en las aplicaciones de procesamiento de datos para los negocios.

BASIC (Beginner's All-purpose Symbolic Instruction Code), un lenguaje científico de programación que fué diseñado con el objeto de

hacer su aprendizaje y su uso tan fácil como sea posible, lo desarrollaron en Dartmouth College en 1965 John Kemeny y Tom Kurz. El sistema BASIC fué el primero en utilizarse en una red o base distribuída, y -- también el primero en estar disponible en tiempo compartido o modo interactivo. Cada comando proporcionado por el usuario desde una terminal BASIC provoca una respuesta inmediata de la computadora, lo cual le permite al usuario tener un control más estricto sobre el procesamiento - de su programa. BASIC continúa siendo muy popular en la actualidad, y el tiempo compartido ha llegado a ser la forma común de operación, con más lenguajes y otras facilidades adicionales para el usuario.

En septiembre de 1963, un comité compuesto por personal de IBM y de clientes se formó con objeto de generar un lenguaje que pudiera atraer a mas usuarios , pero que continuara siendo una poderosa herramienta para el ingeniero. En el momento de su inicio, se creyó que el comité úncamiente extendería el FORTRAN, pero después de realizar el estudio de FORTRAN, ALGOL y COBOL y de entrevistar a gran variedad de usuarios, el comité decidió desarrollar un nuevo lenguaje. El 1o de marzo de 1964 el comité presentó un informe con el nuevo programa propuesto. (Inicialmente denominado NPL., por New Programming Language, cambió de nombre a petición del National Physical Laboratory) El lenguaje fué revisado en junio y diciembre de ese año y denominado finalmente PL/I. El primer manual oficial se publicó al inicio de 1965 y el primer compilador de PL/I fué terminado en el sistema 360 de IBM en el mes de agosto de 1966.

Debido a que PL/I es un lenguaje muy general, tiene una amplísima variedad de aplicaciones; su uso ya en aumento y muchos piensan que llegará a desplazar a sus progenitores — FORTRAN, ALGOL y COBOL—.Recientemente se ha producido algunos compiladores orientados al uso de los estudiantes, como el sistema PL/C de la Universidad de Cornell y el SP/k de la Universidad de Toronto, los cuales se espera que aumenten la aceptación de PL/I como un lenguaje para la enseñanza.
(Tomado de "Ciencia de las Computadoras", J.P Tremblay, P.B. Bunt, Mc. Grace Hill).

CAPITULO II

LENGUAJE FORTRAN

ING. HERIBERTO OLGUIN R.
ING. ANTONIO PEREZ A.
M. EN C. RICARDO CIRIA M.

CAPITULO II

1.- INTRODUCCION

2.- BREVE HISTORIA DEL LENGUAJE

3.- ELEMENTOS BASICOS

- 3.1.- JUEGO DE CARACTERES
- 3.2.- NUMEROS
- 3.3.- ESTRUCTURA DE UN PROGRAMA
  - 3.3.1.- Proposición END
  - 3.3.2.- Proposición STOP
- 3.4.- HOJA DE CODIFICACION

4.- CONSTANTES Y VARIABLES

- 4.1.- CONSTANTES ENTERAS
- 4.2.- CONSTANTES REALES
- 4.3.- VARIABLES ENTERAS
- 4.4.- VARIABLES REALES

5.- EXPRESIONES ARITMETICAS

- 5.1.- OPERADORES ARITMETICAS
- 5.2.- EXPRESIONES ARITMETICAS
  - 5.2.1.- REGLAS PARA FORMAR ARIT -EXPR
- 5.3.- PROPOSICION DE ASIGNACION

6.- ENTRADA Y SALIDA (1a. Parte)

- 6.1.- PROPOSICION PRINT
- 6.2.- PROPOSICION READ

7.- TRANSFERENCIA DE CONTROL

- 7.1.- ETIQUETAS
- 7.2.- PROPOSICION GO TO
- 7.3.- PROPOSICION IF ARITMETICA

8.- ARREGLOS

- 8.1.- VARIABLES CON SUBINDICE
- 8.2.- DECLARACION DIMENSION
- 8.3.- REGLAS PARA FORMAR SUBINDICES

9.- ENTRADA Y SALIDA (2a. Parte)

- 9.1.- PROPOSICION READ
- 9.2.- PROPOSICION WRITE
- 9.3.- PROPOSICION FORMAT

10.- ITERACIONES

- 10.1.-PROPOSICION DO
- 10.2.-ANIDAMIENTO
- 10.3.-PROPOSICION CONTINUE

11.- PROPOSICION IF LOGICA

- 11.1.-EXPRESIONES LOGICAS
- 11.2.-PROPOSICION IF LOGICA

12.- FUNCIONES

- 12.1.-FUNCIONES PROPORCIONADAS POR EL COMPILADOR
- 12.2.-FUNCIONES DEFINIDAS POR EL USUARIO

CAPITULO II Lenguaje Fortran (17)

1.- Introducción al lenguaje FØRTRAN

El lenguaje FØRTRAN, cuyo nombre corresponde a las primeras letras de las palabras inglesas FORmula (fórmula) y TRANslation (traducción), es un lenguaje de programación orientado a problemas matemáticos y se emplea en casi todas las computadoras del mundo. Debido a su parecido con el lenguaje aritmético común, el FØRTRAN simplifica la preparación de problemas que pueden resolverse mediante una computadora. Los datos e instrucciones se pueden organizar mediante una secuencia de enunciados FØRTRAN; éstos constituyen el llamado Programa Fuente.

Todas las computadoras que "entienden" el lenguaje FØRTRAN. tienen lo que se llama un Compilador Fortran, llamado también traductor o intérprete, el cual analiza los enunciados FØRTRAN y los traduce a un Programa Objeto, el cual queda en Lenguaje de Máquina.

Un programa escrito en lenguaje FØRTRAN se puede procesar en cualquier máquina que tenga un Compilador FØRTRAN. Esto nos indica que el lenguaje es independiente para cada máquina, o sea que el compilador se debe preparar en cada caso teniendo en cuenta la máquina que ha de usarse en particular; puesto que las máquinas difieren en su organización interna, se ha desarrollado un número de "dialectos" del Lenguaje FØRTRAN , cada uno de los cuales es apropiado para una clase de máquinas. Las diferencias entre los varios dialectos son mínimas y se ajustan el uno al otro fácilmente.

3.- ELEMENTOS BASICOS

3.1 Juego de Caracteres

El alfabeto FØRTRAN está constituído de caracteres que son símbolos familiares de escritura y de teclados de máquinas de escribir, así como de dispositivos especiales de perforación; dichos caracteres son:

Alfabéticos:	A B C D E F G H
	I J K L M N
	* Ø P Q R S T U V W X Y Z
Numéricos:	0 1 2 3 4 5 6 7 8 9
Símbolos:	+ - * / = . , ( ) ' a

De este alfabeto se construyen todos nuestros símbolos, expresiones y enunciados que se utilizan en el lenguaje FØRTRAN.

3.2 Números

Los números pueden representarse en diferentes formas, las cuales se asemejan a los símbolos de la aritmética general; pero debido a la estructura interna de las computadoras se establecen las convenciones de: Punto Fijo y Punto Flotante que proporcionan facilidades para su manejo en FØRTRAN. Los símbolos de punto fijo se usarán solamente con números enteros y los cálculos asociados se denominarán aritmética de los enteros o modo entero; mientras que la aritmética de los números reales se hará en la forma de punto flotante y se llamará aritmética de los reales o modo real. Debido a que también es necesario distinguir las constantes (números que no cambian durante toda la ejecución de un programa) de las variables (Números que pueden cambiar), surgen cuatro clases de símbolos para los números.

3.3.1 El enunciado END

Este se lee simplemente END e informa al compilador que el programa fuente ha terminado y debe ser el último enunciado de cualquier programa FØRTRAN.

3.3.2, El enunciado STØP

Este aparece simplemente como STØP y es el que nos indica que ha terminado la ejecución y en el caso de IBM - 1130 la computadora se detiene y el operador tendrá que hacer que continúe trabajando. Debido a ello se recomienda que se utilice el enunciado CALL EXIT, el cual pasa el control a un programa monitor que hace que la computadora continúe ejecutando los otros programas que siguen a continuación.

Tanto el STØP como el CALL EXIT podrán aparecer después de cualquier enunciado.

4.- Constantes y Variables

4.1 Constantes enteras.

Dependiendo del tipo de computadora se podrán representar por un cirto número de dígitos, si el entero es negativo, los dígitos deberán ser precedidos del signo menos; si el entero es positivo el signo es opcional.

Ejem. Símbolos para constantes enteras pueden ser entre otras:

1976 +1 0 +1976 -1976

Símbolos que no se aceptan para constantes enteras:

7483282 (dependiendo de la computadora utilizada, puede ser demasiado grande)

1976: (el punto decimal no se permite)

4.2 Constantes reales

Dependiendo del tipo de computadora, las constantes reales se podrán representar por varios dígitos, con punto decimal pudiéndose colocar al principio de los dígitos, al final o entre dos dígitos cualesquiera. Cuando aparece un punto en una constante su tratamiento será de punto flotante. Si la constante real es precedida de un signo menos, se indicará que es negativa, si es positiva el signo es opcional.

Ejem. Símbolos para constantes reales pueden ser entre otras:

1976. -.00001976 +12.345 -12.345

-.007 .007 5.348 0.3

Símbolos que no se aceptan para constantes reales:

123456789.32

5343 (falta el punto decimal)

Para representar las constantes reales existe también la llamada forma exponencial; esta la podemos representar mediante una letra E y una constante entera de uno o dos dígitos, positiva o negativa Esta constante entera es un exponente del número diez; el signo menos es para los exponentes negativos y para los positivos, el signo es opcional. En FORTRAN, la presencia del exponente hace que el uso del punto decimal sea opcional.

Ejem. Forma exponencial	Forma no exponencial
1.328E2	132.8
1.328E02	132.8
1.328E00	132.8
-4.724E-03	-.004724
+7.61E3	7610.
-6432E-3	-6.432

4.3 Variables enteras

Estas se representan por combinaciones de una a seis letras y/o dígitos, no se permiten otros caracteres y el primer caracter deberá ser una de las letras I, J, K, L, M ó N. El primer caracter de una variable es el que indica si es entera o real. Durante la ejecución de un programa, las variables enteras deberán restringirse a valores enteros.

Ejem. Símbolos para variables enteras pueden ser, entre otros:

NUMCT KILO N1 N2 M10 KONT

IIALC JCLAV MARY KONTI L1976

Símbolos no aceptables para variables enteras:

CUENT (el primer caracter debe ser I, J, K, L, M ó N).

Kontador (demasiados caracteres)

12.34 (sólo se aceptan letras y números)

4.4. Variables reales

Estas se representan por combinaciones de una a seis letras y/o dígitos, no se permiten otros caracteres y el primer caracter tiene que ser necesariamente una letra diferente a I,J, K, L, M ó N. Durante la ejecución de un programa dichas variables se deben restringir a valores reales.

(19)

Ejem. Símbolos para variables reales pueden ser, entre otros:

FUERZ	VELOC	ACELI	CUENT	A1	A2
ALFA	VIELA	RA42	XT	PROD	SUMA

Símbolos no aceptables para variables reales:

A3.8 (el punto no es letra o número)
CORRIEN (demasiados caracteres)
3 BASO (el primer caracter debe ser una letra)
MIIMCT (el primer caracter no puede ser M)

5.- Expresiones Aritmética

(20)

5.1 Operadores Aritméticos

Las operaciones aritméticas y los símbolos que se utilizan en FØRTRAN son:

		Ejem.	Algebra	FØRTRAN
Adición	+		a + b	A + B
Sustracción	-		a - b	A - B
Multiplicación	*		a b	A * B
División	/		$\frac{a}{b}$	A / B
Exponenciación	**		a^2	A * B
			a^2	A** 2

5.2 Expresiones aritméticas

En base a lo expuesto anteriormente podemos ahora formular expresiones aritméticas en lenguaje FØRTRAN y nos daremos cuenta que son muy similares a las expresiones aritméticas del algebra común.

Expresiones FØRTRAN	Expresiones Comunes
A**2-B**2	$a^2 - b^2$
B**2-4.*A*C	$b^2 - 4ac$
(A+B)/2.	$\frac{1}{2}(a+b)$
2*K-J+N	2k-j+n
C+B-3.*A	c+b-3a

5.2.1 Reglas para las expresiones aritméticas

Las reglas a las que debemos sujetar las expresiones aritméticas son necesarias debido a la estructura de las computadoras y al observarlas tendremos un ahorro en el tiempo de ejecución de un programa.

Regla 1 Si nos fijamos en las expresiones FØRTRAN anteriores nos damos cuenta que : Todas las constantes y variables en una expresión deben estar en el mismo modo, esto es, todas deben ser enteras o todas deben ser reales. (existe una excepción que mencionaremos más adelante).

Es necesario consultar los manuales de cada máquina, (27)
ya que como hemos mencionado anteriormente dependerá
esta regla del tipo de computadora. Por lo pronto
la consideraremos como se ha indicado.
Si A=5., B=8., C=2. y D=1.6
Entonces (A+B)/C se calcula en el siguiente orden:
5.+8.=13. 13./2.=6.5
Mientras que A+B/C se calcula en el siguiente orden:
8./2.=4. 5.+4.=9.
Ahora si deseamos calcular (A+C)**2 Conducirá a:
5.+2.=7 7.**2=49.
Mientras que A+C**2 Conducirá a:
2.**2=4' 5.+4.=9
Ahora si: (A*B)/(C*D)=40./3.2=12.5
Entonces: A*B/C*D=40./C*D=20.*D=32.
Finalmente si tenemos paréntesis dentro de
otros paréntesis se tiene:
(A*(B+C)**2=(A*10**2=50.**2=2500.
B+C tiene la más alta prioridad por encontrarse
en el paréntesis más interno.
(A*B+C)**2=(40.+2)**2=42.**2=1764.
A*(B+C)**2=A*10.**2=A*100.=500.
A*B+C**2=A*B+4.=40.+4.=44
Debemos teher cuidado en expresar lo que deseamos
realizar.

Regla 4 No deberemos colocar un signo de operación antes de un signo más o menos, esto es, no deberemos poner dos signos de operación juntos.

Ejem. A*B 1+-J M-+N A/-B

Estas expresiones deberán sistituirse por:

A*(-B) I+(-J) M-(+N) A/(-B)

5.3 Proposición de Asignación

Se forman con las expresiones presentadas anteriormente y nos indican los cálculos particulares que deben hacerse. Su forma es:

Variable = Expresión aritmética

El significado del signo = es el de asignación, esto es, que deberá calcularse el valor de la expresión a la derecha del signo = y su valor se asignará a la variable que se encuentre a la izquierda del signo, la cual tiene una localidad en la memoria de la computadora.

Ejem. Si A=5., B=8., C=2. y D=1.6

X=(A+B)/C se le asignará a la X el valor 6.5

ALO=(A+B)**2 se le asignará a ALO el valor 169.

RAI+SQRT(B*C) se le asignará a RAI el valor 4.

Algo diferente al álgebra normal es el enunciado A=A+3. el cual no debe alarmarnos ya que indica que a la localidad de memoria con el nombre A se le asignará elnuevo valor A+3. esto es:

Si A=5. y A=A+3. entonces :

A=5.+3. A=8. o sea que la variable A se le asigna el valor de 8. y el valor anterior que fué 5. se pierde.

6.- ENTRA Y SALIDA (1a. PARTE)

Todo programa FORTRAN que realice algún cálculo ó resuelva algún problema, debe informar al usuario el resultado de sus cálculos mediante los dispositivos con que cuenta la computadora para ello, como podría ser una impresora ó una pantalla de televisión. De igual forma, la mayoría de los programas procesan la información que un dispositivo externo les proporciona, como sería una lectora de tarjetas perforadas, una unidad de cinta ó el teclado de una terminal.

Las instrucciones con que cuenta el lenguaje FORTRAN para hacer estas operaciones se llaman instrucciones de entrada y salida debido a que hacen entrar o salir información desde el programa hacia algún dispositivo externo.

Primeramente se muestran las instrucciones de entrada y salida read y PRINT en sus formas más sencillas.

6.1 Proposición PRINT.

La proposición PRINT es la encargada de mostrar en algún dispositivo de salida el valor de las variables que el programador desee; su forma general más sencilla es la siguiente:

PRINT * , out list.

En la forma anterior, el asterisco es un indicador de que la forma de impresión de los valores se hará de acuerdo a un standard definido por el lenguaje, es decir, en vez del asterisco se podrá poner un indicador que especificará la forma en que deseemos que se impriman los valores de las variables (cuantas columnas en blanco, cuantos decimales después del punto, etc).

Este indicador se llama indicador de FORMATO y se estudiará ampliamente más adelante; por el momento, bastará con que utilicemos el asterisco para no preocuparnos por esos detalles.

Lo que está representado por outlist es la lista de las variables que desamos que se impriman, por ejemplo:

A= 13.5
B= 44.44
I= 123
PRINT *, A, B, I.

Produciría lo siguiente:

1,3500000E+00 4.4440000E+01 123

La impresión en formato exponencial de las variables reales "A" y "B" se debe al standard del lenguaje cuando se utilizó el * en vez del indicador de formato.

6.2 Proposición READ

La proposición READ efectúa la operación contraria de la proposición PRINT, es decir, toma un número que se proporciona por algún dispositivo externo y lo asigna a una variable.

La forma general de la proposición READ es completamente similar a la proposición PRINT, es decir:

READ * , in list.

En donde nuevamente el asterisco indica que se tomen los datos de entrada sin importar la forma en la que venga; por ejemplo, dada la instrucción.

READ *, X, Y, Z.

y suponiendo que los datos externos fueron:

22 - 18.37 4.24E+5

Después de ejercitar la instrucción READ la variable X tendría el valor 22, la variable Y valdría -18.37 y la variable Z sería igual a 425000.

7.- Transferencia de Control

7.1 Etiquetas

Debido a que los enunciados de un programa FØRTRAN se ejecutan en el orden que aparecen y que en muchas ocasiones queremos transferir la ejecución a otros enunciados si se satisface una cierta condición, FØRTRAN nos permite numerar dichos enunciados. Un número de enunciado debe ser una constante entera de uno a cinco caracteres. sin el signo más o menos; el número se coloca a la izquierda del enunciado.

Ejem.
```
  3 CONT = CONT+1.
 24 RAIZ = (A**2+B**2)**.5
```

7.2 Proposición GO TO

Este toma la forma GØ TØ N en donde N es un número de enunciado.

El GØ TØ produce un salto incondicional; así GØ TØ 3 envía la ejecución al enunciado número 3 que puede ser la instrucción de conteo del ejemplo anterior. GØ TØ 24 pesa el control al enunciado 24 que puede ser el del ejemplo anterior.

Ejem. Supongamos que unos de los enunciados de un programa son:

```
      I = 1
      ISUM = 0
    1 ISUM = ISUM+1
      I = I+1
      GØ TØ 1
```

Esto nos representa la suma de los números enteros, desde luego es necesario ponerle otros enunciados pero por el momento nos aclara lo indicado.

7.3 IF ARITMETICO

La instrucción IF aritmética es la parte del lenguaje FORTRAN que se encarga de efectuar una transferencia condicional del flujo de control, es decir, de hacer una transferencia de control como la instrucción GOTO pero, a diferencia de éste, que siempre transfiere el control a un solo lugar, el IF aritmético puede transferir el control a uno de tres lugares diferentes, dependiendo del valor que tenga una expresión dada.

Lo que determina el lugar a transferir es el valor de una expresión aritmética, teniéndose tres destinos diferentes que se escogen de acuerdo al valor negativo, igual a cero o positivo que tenga la expresión; de lo anterior se desprende el nombre de la instrucción (IF aritmética), ya que el destino de la transferencia depende de un valor aritmético.

La forma general de la instrucción IF aritmética es la siguiente:

IF (arit-expresión) ln1, ln2, ln3

En donde arit-expresión es cualquier expresión aritmética y ln1, ln2 y ln3 son etiquetas de líneas de destino; al ejecutarse esta instrucción, se evalúa la expresión aritmética y después se transfiere el control a la línea ln1 si la expresión resultó negativa, a la línea ln2 si la expresión es igual a cero ó a la línea ln3 si la expresión es positiva.

Al escribir programas FORTRAN, es frecuente que se requiera saber si una variable determinada es igual a un cierto valor o no y, dependiendo de la respuesta, efectuar algún cálculo u otro diferente; es por esto que la expresión aritmética más común dentro de un IF aritmético es la resta de una variable menos una constante.

Para averiguar si la variable es igual al valor dado, se resta a la variable la constante dada; de esta manera, si el resultado es igual a cero, sabremos que la variable es igual a la constante; en caso contrario, si el resultado es positivo, la variable es mayor que la constante y si el resultado es negativo, la variable es menor que la constante; veamos un ejemplo:

```
      IF ( A - 7.5 ) 12, 25, 88

C AQUI SE TRANSFIERE EL CONTROL SI "A" ES MENOR QUE 7.5
  12 - - -
     - - -

C AQUI SE TRANSFIERE CUANDO "A" ES IGUAL A 7.5
  25 - - -
     - - -
C. Y AQUI CUANDO "A" ES MAYO QUE 7.5
  88 - - -
     - - -
```

Si al programador no le interesan los tres resultados que proporciona el IF aritmético sino sólo dos de ellos, se puede repetir la misma etiqueta en dos de los destinos del IF aritmético. Por ejemplo, si en un programa hay que efectuar un cálculo cuando la variable NUM sea menor o igual a la variable LIMIT, y otro cálculo diferente cuando NUM sea mayor que LIMIT, se utilizarpia el siguiente IF aritmético.

```
      IF ( NUM - LIMIT ) 20, 20, 40
C CALCULO CUANDO NUM <= LIMIT
  20. - - -
      - - -
C. CALCULO CUANDO NUM > LIMIT
   40 - - -
      - - -
```

Finalmente no hay que 9lvidar que, en forma similar a la instrucción GOTO, la instrucción IF aritmética siempre transferirá el control a partir de ella hacia algún otro punto dentro del programa por lo que la instrucción que siga inmediatamente después de la instrucción IF aritmética sólo se ejecutará si tiene una etiqueta y el control de ransferiere a ella desde algún otro lugar del programa, por lo cual, es conveniente que alguno de los destinos del IF aritmético se coloque inmediatamente después de la instrucicón IF.

8.- Arreglos

Frecuentemente tratamos con un grupo de variables que forman ó pertenecen a una clase o colección. Cuando las variables forman un conjunto ordenado, pueden relacionarse unas con otras por la notación de subíndices; entonces designamos esa colección como arreglo y las variables que pertenecen a ésta serie son elementos del arreglo. A veces se emplea como sinónimo de arreglo el nombre de matriz y, en consecuencia, hablamos de elementos de la matriz.

8.1 Variables con subíndice

Un conjunto de números que pueda arreglarse en un renglón ó columna se considera como un arreglo lineal ó unidimensional, y ésta serie puede llamarse vector. Identificamos los elementos de un vector renglón ó columna por un sólo subíndice.

Ejem. La columna de números del vector llamado A, consiste de los elementos A, hasta An inclusive y se representa como sigue:

Notación acostumbrada	Notación FØRTRAN
A_1	A (1)
A_2	A (2)
A_3	A (3)
⋮	⋮
Ai	A(i)
⋮	⋮
An	A(N)

Cada una de estas A(1), en donde I varía de I a N, son el nombre de una variable, el conjunto de todas ellas es lo que llamamos arreglo.

Si se usan dos subíndices para identificar los elementos de un arreglo se considera éste como un arreglo bidimensional. Los cuadros de un tablero de ajedrez, pueden considerarse como un arreglo bidimensional. Y si llamamos a cualquiera de los cuadros con la variable CTAJ tendremos 64 variables; pero como el tablero tiene 8 renglones y 8 columnas, podemos referirnos al cuadro que se encuentra en el renglón 3 y la columna 5 con la variable CTAJ (3,5).

Dependiendo del tipo de computadora será el núm de subíndices que podremos asignarle a un arreglo; en IBM - 1130 sólo se admiten arreglos con un máximo de tres subíndices.

Las variables que se utilicen para designar arreglos deberán observar las reglas que se dieron anteriormente al hablar de variables enteras y reales considerando que para los cinco caracteres alfanuméricos son independientes de los índices que se encuentran entre paréntesis.

8.2 Declaración DIMENSION

Siempre que en un programa utilicemos variables con subíndices deberemos poner como primer enunciado el DIMENSIØN, el cual indica al compilador qué tanto espacio de memoria se debe reservar para las variables con subíndices. Su forma es:

DIMENSIØN u, v, w, ...

Donde u, v, w, ... son nombres de variables, cada una de las cuales va seguida por el máximo número de elementos en el arreglo correspondiente. Deberán observarse las siguientes reglas:

Regla 1 Cada variable con subíndices se debe mencionar en un enunciado DIMENSIØN antes de su primer uso en el programa.

Regla 2 Los símbolos representados anteriormente por u, v, w, ... deben tener la forma:

nombre de variable (máximo número de elementos)

el número entre paréntesis debe ser una constante entera sin signo.

Ejem. DIMENSION A(20), B(4,8), CARR(5,3,4)

Esto indica que el compilador reservará 20 localidades para el arreglo A, sus veinte variables serán A(1), A(2), ..., A(20) al mismo tiempo se reservarán 32 (4X8) localidades para las variables B(1,1), B(1,2), B(1,3), ..., B(1,8), B(2,1), B(4,1), B(2,8), B(3,1), B(3,2), ..., B(3,8), B(4,1), B(4,2), ..., B(4,8) y por último se reservarán 60 (5X3X4) localidades para las variables del arreglo CAR, con tres subíndices cada una.

Regla 3 El arreglo que se use en particular, dentro del programa podrá tener menos elementos que los especificados en la magnitud del enunciados DIMENSION, pero no más.

Regla 4 La variable tal como aparece en el enunciado DIMENSIØN debe tener exactamente el mismo número de subíndices que en cualquier otra parte del programa.

8.3 Reglas para formar subíndices

Regla 1 Un subíndice debe ser un entero, puede ser constante, variable ó una de las expresiones aritméticas siguientes:

A * V + b A * V - b

en donde v es una variable entera y a y b son constantes enteras sin signo.

Ejem. Algunos subíndices pueden ser:

1 1972 10*KONT 2*I J

1976*N-8 2*I-4 2*I+3

No se pueden usar como subíndice:

1+I -I 2-10*CONT -1932 -KILO

Regla 2 Un subíndice sólo debe tomar valores positivos

Regla 3 Un subíndice en sí no debe ser una variable con subíndices. Así X(I(2) no es permitido.

Regla 4 Un símbolo que representa un arreglo, una variable con subíndice, no debe usarse sin subíndices para representar otra variable diferente en el mismo programa. Esto es A(I) y A no deben referirse a variables diferentes. Como siempre hay una excepción que por ahora no tocaremos.

Ejem. Los símbolos para variables reales con subíndices podrían incluir:

X(I) SUM(K+2) A(1, 2*J+1) B(INT) C(I,J)

Para variables enteras con subíndices. podemos tener:

INT(M,N) I(J) ICTA(J,2*I).

9.- Entrada y Salida (2a. Parte)

Estos, como su nombre lo indica, sirven para introducir y sacar información de la computadora.

9.1 Proposición READ

Este enunciado tiene la forma READ (I, N) LISTA I y N son enteros sin signo y LISTA representa una lista de nombres de variables para las cuales se leerán valores. I designa el tipo de periférico de entrada que se utilice (lectora de tarjetas, consola, etc.). N es el número de un enunciado FØRMAT asociado al READ.

Ejem. El enunciado READ (2, 101) J, B, H

Producirá la lectura de tres números: un entero y dos reales y se almacenarán en las localidades de la memoria de la computadora designadas con las variables J, B y H en su orden. Las comas que separa estos nombres de variables en el READ son indispensables, 2 es la unidad de entrada y 101 un FØRMAT.

9.2 Proposición WRITE

Este tiene la forma WRITE (I, N) LISTA I y N son enteros sin signo y LISTA representa una lsita de variables para las cuales se imprimen valores. I designa el tipo de periférico de salida que se utilice (impresora, cinta, etc.). N es el número de un enunciado FØRMAT asociado al WRITE.

Ejem. El enunciado WRITE (3,108) L,X,Y

Producirá que se impriman los valores de las variables L, X y Y que se encuentren en las localidades de memoria con esos nombres, en el formato especificado por el enunciado número 108 y por la unidad de salida número 3; las comas que separan estos nombres de variables en el WRITE son indispensables.

9.3 Declaración FORMAT

Este tipo de enunciado no inician por si mismos los cálculos, no producen transferencia de control ni estimulan el flujo de información, pero proveen al compilador FØRTRAN de los detalles esenciales para la traducción del programa fuente en FØRTRAN al programa objeto en lenguaje de máquina ó para la conversión de datos a la entrada o la salida.

Si queremos introducir datos a la computadora lo podemos hacer mediante un enunciado que esté dentro del programa, como A = 3.1416, ésto es lo que podríamos llamar inicializar una variable; y el programa se compilaría cada vez que quisieramos darle un valor diferente a A, lo cual resulta muy costoso, ya que las compilaciones son laboriosas. Para evitar esto se usa el enunciado READ y los valores que se le den a A podrán estar en tarjetas de datos, los cuales son independientes de programa fuente.

El enunciado FØRMAT

Este tiene la forma: N FØRMAT (, , , ...) en la cual N es el número del enunciado FØRMAT y corresponde al N de los enunciados READ y WRITE. Los espacios entre las comas están disponibles para las especificaciones del tipo que se describen más adelante, siendo el número de espacios uno o más, de acuerdo a las necesidades del programador.

La especificación I: Iw

Aquí I indica un valor entero y W es un entero que indica el número de columnas o ancho de campo, que ocupa ese valor en la tarjeta de entrada ó en el papel de impresión. El número w deberá incluir un lugar para el signo de ese valor, siendo + opcional.

Ejem. Valor de los datos
de entrada o salida: 1130 +1620 -370 0 +14
Especificación : I4 I5 I4 I1 I3

La especificación F:Fw.d

Aquí F indica un valor real, w indica el número de columnas que ocupará el valor en la tarjeta de entrada o en el papel de impresión; d indica el número de cifras que se encontrarán después del punto decimal. w deberá incluir un lugar para el signo y otro para el punto decimal.

Ejem. Valor de los datos de
entrada ó salida: 32.787 -.007 1130. +3.70
Especificación: F6.3 F5.3 F5.0 F5.2

La especificación E: Ew.d

Aquí E indica un valor real en forma exponencial y w indica la anchura de campo para ese valor y debe de incluir el signo, si lo hay, el punto decimal, el lugar para la letra E, un lugar para el signo del exponente, si es negativo, y dos lugares para el exponente; d indica el número de dígitos a la derecha del punto decimal.

Ejem. Valor de los datos
de entrada o salida: .1403E04 -.7E-02 .1442E+04
Especificación: E8.4 E7.1 E9.4

Es conveniente que cuando seemos sacar información de la computadora, tomemos en cuenta para el ancho del campo lo siguiente:

1.- El signo, aún cuando el + generalmente no se imprime.
2.- El punto decimal para las especificaciones F y E
3.- Por lo menos un dígito a la izquierda del punto decimal, puesto que muchas máquinas imprimirán allí un cero si otro dígito no ocurre.

Suficientes lugares para todos los dígitos significativos deseados, debido a que para los dígitos que no se les deja espacio se truncan o redondean.

5.- Cuatro lugares para el exponente de la especificación E.

6.- El primer lugar se deja en blanco para el control de carro.

10.- Iteración

10.1 Proposición DO

Este toma la forma:

DØ K I = L, M, N.

DØ K I = L, M

La segunda forma sólo se aplica cuando N=1, lo que es bastante frecuente.

K representa un número de enunciado

I representa una variable entera

L, M, N son variables enteras constantes sin signo.

El DØ produce la ejecución repetida de todos los enunciados que le sigue, hasta el enunciado número K. La primera vez que se ejecutan estos enunciados la variable I es igual a L, en cada paso subsiguiente I se incrementa en la cantidad N, hasta hacerse mayor ó igual a M en el paso final; en este momento se termina el llamado lazo DØ y el control pasa al enunciado que está a continuación del enunciado K. Así, L es el valor inicial de la variable I y M su valor final. I se llama el índice del enunciado DØ y su valor corriente se puede usar en cálculos durante la ejecución del lazo. Todos los enunciados que le siguen al DØ hasta el número K inclusive constituyen el rango del DØ. También es posible que la variable I no se encuentre en ninguno de los enunciados del rango del DØ y esto nos indica que se realice la ejecución de todos los enunciados del rango del DØ M entre N veces (la parte entera de este cociente M/N). Deberemos tomar en cuenta que: el índice I se incrementa secuencial y automáticamente durante la ejecución del lazo y que se puede, en estos momentos, tratar como cualquier variable entera; el índice I queda indefinido después de terminado el lazo DØ y puede utilizarse para cualquier uso general. El enunciado K no debe ser un enunciado de especificación ni una transferencia de control esto cinluye cosas como GØ TØ, IF y DØ, así como FORMAT, END y algunos otros. Debemos considerar que no se puede desde ningún punto del programa llegar a un enunciado dentro del rango de un DØ.

Ejem. Utilizaremos un DØ para sumar los número enteros del 1 al 100, ejemplo que ya hemos visto anteriormente.

```
      ISUM = 0
      DØ 1 I = 1,100
    1 ISUM = ISUM+1
      STOP
```

Nos damos cuenta que el DØ tiene la misma función que un IF, un GØ TØ y un contador; como podrá observarse con el ejemplo anterior.

10.3 PROPOSICION CONTINUE

En ocasiones, un programa FORTRAN requiere que se transifera el control hacia un punto en el cual no se efectúe ninguna operación; por ejemplo, suponga que se tiene una instrucción DO para iterar a través de los elementos de un arreglo a fin de contar quellos elementos que sean positivos e ignorar los que sean iguales a cero ó negativos; veamos el ejemplo:

```
      NUM = 0
      DO 20 I=1,n
      IF (VEC (I) ) 20, 20, 10
10    NUM = NUM + 1
20    NUM = NUM
```

La instrucicón con la etiqueta número 20 es la salida del IF en la cual el contador no debe incrementarse; al mismo tiempo que es el punto de regreso para la instrucción DO pero,como se ve claramente, esta instrucción no debe realizar ninguna - operación, lo cual se obtiene de este ejemplo por medio de la instrucción NUM= NUM lo cual, sin embargo, podría confundirnos.

Afortunadamente, el lenguaje proporciona una instrucción que no hace nada para usarla en estos casos, esta instrucción es CONTINUE y puede usarse en cualquier lugar que el programador desee; regresando al ejemplo anterior;

```
      NUM = 0
      DO  20 I = 1, n
           IF (VEC (I) ) 20, 20, 10
10    NUM = NUM + 1
20    CONTINUE
```

A pesar de que las construcciones de programa FORTRAN como la anterior, en la que la instrucción CONTINUE se requiere, no son muy comunes; una práctica muy extendida dentro de la programación FORTRAN consiste en cerrar todas las proposiciones DO con proposiciones CONTINUE aunque no se necesiten; de esta manera el programador puede ver de inmediato en donde termina el rango de las proposiciones DO.

Por ejemplo, si se requiere sumar todos los elementos de una motriz de dos dimensiones con el sig. programa:

```
      SUMA = 0.0
      DO 10  I = 1,n
      DO 10  I = 1,M
10 Suma = SUMA + MAT (I, J)
```

Es mucho más claro hacerlo de la siguiente manera:

```
      DO 20  I = 1 , N
        DO 10 J = 1, M
            SUMA = SUMA + MAT (I, J)
10        CONTINUE
20    CONTINUE
```

11.- Proposición IF (Lógico)

(40)

11.1 Expresiones Lógicas

Para formar las expresiones lógicas (L) utilizaremos los operadores de compraración y los de relación.

Operadores de comparación:

Símbolo Matemático	Significado	Símbolo FØRTRAN	Significado Inglés.
<	Menor que	.LT.	Less than
>	Mayor que	.GT.	Greater than
≤	Menor o igual a	.LE.	Less or equal
≥	Mayor o igual a	.GE.	Greater or equal
=	igual a	.EQ.	Equal
≠	Diferente a ó No igual a	.NE.	Not equal

Operadores de relación:

	Unión	.OR.	ó ("o inclusive)
	Intersección	.AND.	y ("al mismo tiempo)
—	Complemento	.NOT.	no

Para valuar una expresión lógica se hará con las siguientes prioridades:

1.- Expresiones entre paréntesis
2.- Operadores aritméticos
3.- Operadores de comparación (.LT., .GT., .LE., .GE., .EQ. H .NE.)
4.- .NOT.
5.- .AND.
6.- .OR.

En caso de igual jerarquía la evaluación será de izquierda a derecha.

11.2 Proposición IF (Lógico)

(41)

El IF lógico es de la forma:

IF (L) S

L= expresión lógica que puede tener dos valores: Verdadero o Falso.

S= cualquier enunciado FØRTRAN diferente de: un DØ, un enunciado de especificación o de otro IF lógico.

Si L es falso (.FALSE.) entonces se ignora S y la computación continúa al siguiente enunciado. Si L es verdadero (.TRUE.) el enunciado S se ejecuta en seguida.

Resulta interesante hacer notar que si L es relativamente complicada, éste IF puede ser el equivalente de varios IF aritméticos.

Ejem.(1) X=5. y=0.5

```
IF (X.GT.3..AND. Y .LE.2)  Z=X**3+X*Y
```

Significa que si X>3. y (al mismo tiempo) y≤2. se asignará a 2 el valor que se obtenga al calcular X^3+XY, esto es Z=125.+2.5=127.5

(2)

```
IF (A.LE.Z.AND.B.GE. Y .OR.C.GT.Z) GØ TØ 12
```

Significa que si A≤X y (al mismo tiempo) B≥Y es verdadero ó C>Z es verdadero ó ambos, entonces se transfiere el control al enunciado 12.

(3)

```
      I = 1
      ISUM = 0
    1 ISUM = ISUM+1
      I = I+1
      IF (I.LE.100) GØ TØ 1
      STOP
```

Esto nos indica que sólo sumaremos los números enteros del 1 al 100

12.- Funciones

12.1 Proporcionados por el Compilador

Estas funciones predefinidas que proporciona el lenguaje FØRTRAN son de tipo de biblioteca. Para utilizarlas usaremos el nombre de la función seguido de un argumento que deberá estar entre paréntesis. Dichos argumentos pueden ser variables simples ó con subíndices, constantes, expresiones aritméticas u otras funciones predefinidas en FØRTRAN

Para IBM - 1130 tenemos:

NOMBRE	FUNCION EJECUTADA	NUM. DE ARGUMENTOS	TIPO DE ARGUMENTO(S)	TIPO DE FUNCION
SIN	Seno trigonométrico (argumento en radianes)	1	Real	Real
COS	Coseno trigonométrico (argumento en radianes)	1	Real	Real
ALOG	Logaritmo natural	1	Real	Real
EXP	Argumento de potencia del número a.	1	Real	Real
SQRT	Raíz cuadrada	1	Real	Real
ATAN	Arco Tangente	1	Real	Real
ABS	Valor absoluto	1	Real	Real
IABS	Valor absoluto	1	Entero	Entero
FLOAT	Convertir argumento de entero a real	1	Entero	Real
IFIX	Convertir argumento de real a entero	1	Real	Entero
SIGN	Transferencia de signo (Arg.1 recibe signo de Arg. 2)	2	Real	Real
ISIGN	Transferencia de signo (arg.1 recibe signo de Arg.2)	2	Entero	Entero
TANH	Tangente Hiperbólica	1	Real	Real

Ejem. SQRT (B**2-4.*4.*A*C) indica que a lo que se encuentra entre paréntesis se le sacará la raíz cuadrada. SIN (BETA) indica que se obtendrá el seno trigonométrico de el valor de la variable BETA.

13.- Subprogramas

Los subprogramas, también llamados subrutinas, son programas que pueden ser puestos en uso por otros programas cuando sea necesario.

Las funcion-s de biblioteca ó funciones del sistema constituyen una variedad de subprogramas .

13.1 FUNCIONES

Cuando el valor de una variable depende de una ó más variables ó constantes y además de una serie de cálculos, y dicha variable ha de calcularse repetidamente y en diferentes puntos de un programa, es posible definirla como una Función. En otras palabras, Además de las funciones con que cuenta la biblioteca del sistema, el usuario puede escribir sus propias funciones para uso específico de su programa.

Tomemos un ejemplo para visualizar lo anterior:

Supongamos que para un programa en especial, en el cual trabajamos con grados en lugar de radianes, deseamos calcular continuamente SENØ (X), sin el uso de funciones sería necesario transformar el argumento deseado de grado a radianes y después llamar a la función del sistema SIN (X). A continuación presentamos una función que calculará SENO (X), (X en grados):

```
      FUNCTION SENØ  ( X )
      X = X * 3.14 15 92/180
      SENØ = SIN (X)
      RETURN
      END
```

que es llamada desde el programa como:

```
      GRAD = SENØ (GRADØS)
```

En base a este ejemplo podemos generalizar el uso de la proposición FUNCTION.

a) Debe ser codificada en forma independiente del programa que la usará, es decir, no debe aparecer "dentro" del programa.

b) Debe empezar con la palabra FUNCTIØN

```
      FUNCTION nombre (parámetro )
```

c) A continuación se escribe el nombre con que será llamada.

d) Después, entre paréntesis y separados por comas, aparecen los argumentos.

EJEMPLOS.-

```
      FUNCTION RAIZ 1 (A,B,C)
      RAIZ1= (B+SQRT (B **2 - 4. * A * C )) / (2,* A)
      RETURN
    END
    FUNCTIØN  RAIZ2 (A, B, C)
      RAIZ2 = ( -B - SQRT (B**2.4 * A * C )) / (2.*A )
      RETURN
    END
C     EC. SEGUNDØ GRADØ
      READ (2.100) A, B, C
100   FORMAT  (3F10.5)
      X1 = RAIZ1 (A,B,C)
      X2 = RAIZ2 (A,B,C)
      WRITE (3,200) A,B,C, XI,X2
200   FORMAT (5 , F10.5)
      CALL EXIT
      END
```

Este ejemplo es solamente para mostrar el uso de la proposición FUNCTIØN y no contempla algunas situaciones como raíces complejas, etc.

13.2 SUBRUTINAS

Como es fácil notar, la proposición FUNCTIØN nos "regresa" un sólo valor y lo hace a través de su nombre. En muchos casos es conveniente ó necesario que se nos regrese más de un valor, para éstos casos usamos la proposición o enunciado:
SUBRUTINE.

Una subrutina es un subprograma que puede "recibir" cualquier número de parámetros (desde cero hasta un número determinado por el tipo de compilador) y puede "regresar" diferentes valores calculados.

Veamos algunos ejemplos:

Supongamos que al imprimir resultados de un cierto programa tenemos que escribir algún título usando los primero renglones de la hojas. En tal caso podemos hacer uso de una subrutina como sigue:

```
      SUBRØUTINE  ENCA
      ØRITE (3,200)
200   FØRMAT (/.1X, 'REPORTE SEMANAL' . / )
      RETURN
   END
```

Como vemos no hemos pasado ningún parámetro ó valor a la subrutina. Para que se ejecute ésta se debe hacer uso de la proposición CALL, de la siguiente forma:

```
CALL ENCA
```

dentro del programa y en el lugar donde deseemos que ocurra la impresión.

Discutamos ahora un ejemplo muy simple para ejemplificar el uso de parámetros. Hagamos una subrutina que "recibe" como entrada dos números, los sume y el resultado lo "regrese" en otra variable Sean A y B los números a sumar, y C la variable en donde se pondrá el resultado.

```
      SUBROUTINE   SUMA  (A,B,C)
        C= A + B
        RETURN
    END
```

Es importante detenerse a ver el significado de los parámetros para las subrutinas:

La subrutina anterior SUMA puede ser llamada de diversas formas:

```
      CALL   SUMA  (AA,BB,CC)
      CALL   SUMA  (4, 7, X )
      etc.
```

Como vemos, las variables A,B y C que aparecen en la subrutina son variables mudas o dormidas y solo tienen sentido dentro de la subrutina. Veamos lo anterior:
Supóngase el siguiente programa:

```
      X1 = 3.
      X2 = 4.
      CALL  SUMA  (X1, X2, X3)
      SUM= X3
      WRITE (3,200) X1, X2, X3, SUM
200   FØRMAT(4 F10.5)
      CALL EXIT
      END
```

Se propone como ejecicio al lector que haga las veces de la máquina y escriba lo que ésta imprimiría.
La máquina imprimirá :

3.0 4.0 7.0 7.0

Una de las facilidades más útiles en subrutinas es la de pasar arreglos como parámetros, ej:

```
      SUBROUTINE MAXIM    (A, MAX)
      DIMENSION           A ( 10)
      - - - - -
      - - - - -
      - - - - -
      - - - - -
      - - - - -
      RETURN
   END
```

Supóngase que ésta subrutina encuentra el elemento del arreglo A (10) con mayor valor y lo regresa a través de la variable MAX. Es importante notar que si pasamos como parámetro uno ó más arreglos hay que dimensionarlos otra vez dentro de la subrutina, lo cual se puede hacer de al menos dos formas: 1) poniendo la dimensión que aparece en el programa que lo llama;
2) Poniéndole dimensión 1(uno)
Ejemplo:

```
DIMENSION A (10) , B (20)
- - - - -
- - - - -
- - - - -
- - - - -

CALL ØRDEN   (A)
CALL MAXIM   (B)
CALL MAXIM   (A)
- - - - - -
- - - - - -
- - - - - -
- - - - - -
- - - - - -
CALL EXIT
END.
```


Caso 1:

```
   SUBROUTINE ØRDEN  (X)
   DIMENSION         X (jQ)
   - - - - -
   - - - - -
   - - - - -
   - - - - -
   RETURN
END.
```

Caso 2 :

```
   SUBROUTINE MAXIM (Y)
   DIMENSION     Y  (j)
   - - - - -
   - - - - -
   - - - - -
   - - - - -
   RETURN
END.
```

13.3 COMMON.-

Como es posible visualizar en los párrafos anteriores, las variables usadas en las subrutinas, o mejor dicho, dentro de las subrutinas, son totalmente independientes a las avariables usadas en el programa principal. Muchas veces es conveniente que tanto las subrutinas como el programa que las llama tengan variables en COMUN. Para lograr ésto existe la declaración.

COMMON

La forma general de esta proposición es:

COMMON lista de variables

donde "lista de variables" es un conjunto de variables y/o arreglos separados por comas a las cuales queremos adjudicarles la propiedad anterior, es decir, sean comunes a varios subprogramas.

Ej.

```
COMMON    A,B, X (10), AB (30)
```

Esta delcaración debe aparecer al principio de cualquier programa o subrutina en que se desee usar. Veamos un ejemplo.

```
C       SUMA DE DOS NUMEROS
        COMMON A, B, C
        A= 3
        B= 7
        CALL SUMA
        Z = C
        WRITE (3,200) A, B, C, Z
200     FORMAT (4 F10.5)
        CALL EXIT
        END.
```



```
SUBROUTINE SUMA
COMMON A, B, C
C = A + B
RETURN
END
```

Este programa debe imprimir:

```
 3.0        7.0        10.0          10.0
```

Una propiedad importante del COMMON es que si un arreglo es especificado en COMMON que dá automáticamente dimensionado, es decir, no hay que especificar dicho arreglo a través de la declaración DIMENSION.

En las siguientes páginas se muestran veintiún programas, que incluyen sus diagramas de flujo, condificaciones, datos y resultados; el objeto es que al lector pueda complementar la parte teórica con la práctica, amén de que deberá hacer los propios y procesarlos en una computadora a su alcance.

DIVISION DE EDUCACION CONTINUA
FACULTAD DE INGENIERIA U.N.A.M.

INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA

SISTEMAS Y PROGRAMAS.

OCTUBRE , 1984.

Palacio de Minería Calle de Tacuba 5 primer piso Deleg. Cuauhtemoc 06000 México, D.F. Tel.: 521-40-20 Apdo. Postal M-2285

FICHEROS BASIC

En esta sección se explicará cómo crear y acceder a la información almacenada en dispositivos externos. Esta información se almacenará en ficheros Basic.

Ficheros

Un fichero es una colección de datos que está en asociación con un dispositivo físico concreto. Ha de utilizarse ficheros Basic cuando se almacenan programas en disco y más adelante cargarlos a memoria.

Además de los programas los ficheros pueden almacenar datos a los que los programas pueden tener acceso.

Ficheros de Programas

Los fiecheros de programas suelen contener una representación binaria, o en ASCII, de los programas. Estos ficheros, que están creados por BASIC o por otro lenguaje de alto nivel, son los que realmente utiliza el COMPUTADOR cuando ejecuta un programa.

Por ejemplo, tecleando RUN SAMPLE en respuesta al mensaje orientativo inicial de BASIC, comunica a BASIC que cargue el fichero SAMPLE a partir de la unidad de disco, RUN supone que SAMPLE es un fichero.

Ficheros de datos

Los ficheros de datos retienen la información que un programa procesa o crea. Por ejemplo, un programa de procesamiento de palabras procesa un fichero de datos llenado con texto. El operador puede efectuar la edición correctora de este fichero cuando así se desee y luego, quizá, almacenarle como otro fichero de datos aparte.

Por lo general, los ficheros de datos son de utilidad cuando un programa requiere, o produce, grandes cantidades de datos. Normalmente, los ficheros empleados para la introducción o salida a partir de programas serán ficheros de datos, en oposición a ficheros de programas.

Dispositivos de ficheros

Hay varios dispositivos que pueden retener ficheros, bien sea para introducción a su programa, bien sea para salir del mismo, bien sea para ambas cosas. Especifique un dispositivo particular cuando nombre un fichero; BASIC identifica varios nombres de dispositivo como parte de un

nombre fichero BASIC válido. Los dispositivos que soportan fiecheros BASIC se indican en la Tabla siguiente:

Dispositivos de ficheros BASIC

Dispositivo de fichero	Nombre del dispositivo	Entrada	Salida
Grabadora casete	CASI:	X	X
Unidad de disco A	A:	X	X
Unidad de disco B	B:	X	X
Teclado	KYBD:	X	
Pantalla de visualización	SCRN:		X
Impresora de línea # 1	LPTI:		X
Impresora de línea # 2	LPT2:		X
Impresora de línea # 3	LPT3:		X
Adaptador # 1 de comunicaciones asíncronas	COM1:	X	X
Adaptador # 2 de comunicaciones asíncronas	COM2:	X	X

Un medio que puede almacenar mucha información, tal como un disco flexible, puede contener muchos ficheros. Cada fichero en el disco se referencia con su propio y singular nombre de fichero. Un fichero de disco puede utilizarse como una fuente y como un destino para datos; en la carga de un programa a partir de un fichero de disco se utiliza ese fichero como una fuente y en la conservación de un programa como un fichero de disco, se emplea el fichero como un destino.

Algunos dispositivos están asociados con solamente un fichero y por consiguiente, con sólo un nombre de fichero. Además, algunos dispositivos pueden utilizarse exclusivamente como ficheros de salida o exclusivamente como ficheros de entrada. Por ejemplo, a la pantalla de visualización puede actuar solamente como un destino para datos, es decir, es exclusivamente un fichero de salida. Un ejemplo opuesto es el teclado que es exclusivamente un fichero de entrada y no se puede enviar ninguna información al teclado.

Algunos dispositivos pueden utilizarse como ficheros BASIC sin que se especifique el nombre del dispositivo. Por ejemplo, no se tiene que especificar explícitamente el teclado o la presentación visual como ficheros de entrada, o de salida, cuando se utiliza sentencias como INPUT o PRINT. Hay muchos casos en los que se utilizará entrada y salido tipo fichero a partir de ficheros

sin referencia explícita al dispositivo.

Ficheros de discos

Hay dos tipos de ficheros de discos: ficheros secuenciales y ficheros aleatorios. La principal diferencia entre estos tipos de fichero es como se accede a sus datos y cómo se almacenan los mismos.

La ventaja de los ficheros secuenciales es que son fáciles de utilizar. Se requieren pocas operaciones costosas de servicio para poner a punto un fichero secuencial y el procedimiento de acceso es sencillo. Los ficheros aleatorios, por el contrario, ocupan menos espacio en un disco que los ficheros secuenciales. Los ficheros aleatorios son también más versátiles, puesto que se puede llegar a cualquier punto en el fichero.

FICHEROS DE DISCOS SECUENCIALES

Un fichero secuencial es simplemente una serie de caracteres sin ningún formato intrínseco. Los datos irán automáticamente a donde indique el puntero (es decir, al final del fichero). De este modo, añadiendo un cuarto item al fichero anterior se tendría:

```
ITEM 1
ITEM 2
ITEM 3
ITEM 4
      <-------- Número extremo de fichero secuencial
```

Cuando se comienza la lectura de datos a partir de un fichero secuencial, el primer elemento de datos procede del principio del fichero. Cuando se continúa la lectura del fichero, los datos se recuperan en el orden en que se escribieron originalmente.

```
ITEM 1 <-------- Principio de fichero secuencial
ITEM 2           ITEM 1 es el primer dato leído
ITEM 3
ITEM 4
       <-------- Final de fichero secuencial
ITEM 2 <-------- Principio de fichero secuencial
ITEM 3           ITEM 2 es el siguiente dato leído
ITEM 4
       <-------- Final de fichero secuencial
```

Todos los dispositivos de ficheros BASIC soportan la estructura de fichero secuencial. La mayoría de los dispositivos de hecho, son inherentemente secuenciales.

Acceso de fichero secuencial

Los pasos en el acceso a un fichero secuencial son como sigue. En primer lugar, el fichero ha de abrirse. La apertura de un fichero hace que BASIC ponga aparte espacio de memoria para transferir datos al, y desde el, dispositivo en el que reside el fichero y para el seguimiento del proceso de acceso del fichero.

Una vez que se haya abierto un fichero, se puede tener acceso al mismo para salida (escritura de datos para el fichero) o entrada (lectura de datos desde el fichero). No se puede utilizar un fichero secuencial para salida y entrada al mismo tiempo. Cuando se especifica salida de datos para un fichero, también ha de especificarse si los datos se añadirán al final del fichero o al principio. En el segundo caso, se perderá cualquier dato que ya estuviera en el fichero.

Después de efectuar las debidas operaciones con un fichero secuencial, el fichero debe cerrarse. El cierre de un fichero hace que BASIC escriba cualquier dato restante desde la memoria al dispositivo en donde se almacenará y luego renuncie a la memoria reservada para manipular acceso de fichero.

Apertura de un fichero secuencial

La sentencia OPEN crea un canal para un fichero residente en un dispositivo específico. Hay dos formatos para la sentencia OPEN:

OPEN espec. fichero FOR modo AS (#) num. fich.

o

OPEN modo 2, (#) num. fich. espec. fichero.

Espec. fichero es una especificación de fichero con validez (nombre de fichero). Si se está teniendo acceso a un fichero secuencial en un dispositivo distinto de un disco o casete, entonces, consiste solamente en el nombre del dispositivo, como se indica en la Tabla anterior.

Los parámetros modo y modo2 son expresiones de cadenas que especifican cómo el programa accederá al fichero secuencial. Estos parámetros se definen como sigue:

Modo	Modo 2	Tipo de acceso
OUTPUT	O	Salida secuencial
INPUT	I	Entrada secuencial
APPEND	(no disponible)	Adición secuencial a fichero existente

El parámetro num.fich. es una expresión entera que las posteriores sentencias de acceso a fichero utilizarán para especificar el fichero. La expresión num.fich. debe ser un entero.

Un ejemplo de una sentencia OPEN de fichero secuencial es:

```
300 OPEN "B:BLACKHOL.MC2" FOR APPEND AS #6
```

Otro ejemplo de sentencia OPEN de fichero secuencial es:

```
20 OPEN "I", 2, "A:GOGETEM.RAH"
```

Escritura para un fichero secuencial

Una vez que se haya abierto para salida un fichero secuencial, las siguientes sentencias pueden utilizarse para introducir información en el fichero:

```
PRINT#
PRINT#USING
WRITE#
```

Estas sentencias actúan casi como sus contrapartidas de salida de visualización. La diferencia, en este caso, radica en que estas sentencias incluyen un número de fichero y que escriben datos para un fichero, y no para la presentación visual. Por ejemplo, las siguientes sentencias escribirán datos para número de fichero 2:

```
190 REM VALOR=4
200 PRINT#2,43.33; VALOR
210 PRINT#2,USING"-   ##.#    ";923800,6.555E-5
```

La ejecución de estas sentencias dará lugar a los siguientes datos en número de fichero 2:

```
43.33 4 @ 92.4E4 @ 65.6E-4
```

Caracteres de espacio

Secuencias de caracteres de avance de línea-retorno del carro

La sentencia WRITE# se comporta también como su contrapartida de salida de presentación visual. Cuando una WRITE# da salida a datos para un fichero secuencial, los datos numéricos se separan con comas y las cadenas se separan con comillas (").

Lectura a partir de un fichero secuencial

Un fichero secuencial que está abierto para introducción puede ser objeto de acceso con las sentencias siguientes:

```
INPUT#
LINE INPUT#
INPUT#
```

Las sentencias INPUT# y LINE INPUT# actúan con ficheros secuenciales como las sentencias INPUT Y LINE INPUT lo hacen con la introducción a partir del teclado.

La sentencia INPUT# tiene el formato siguiente:

```
INPUT #filenum, variable , variable ...
```

La sentencia INPUT# lee los datos en la posición actual en el fichero secuencial y asigna los datos a la variable correspondiente en la sentencia. El tipo de los datos que son objeto de lectura y asignación a una variable debe coincidir con el tipo de variable; de no ser así, dará lugar a un error "Type Mismatch" (falta de coincidiencia de tipo).

Por ejemplo, supongamos que los siguientes caracteres formaran parte de un fichero secuencial:

```
Fred, Smith Frieda, Smythe
```

La siguiente sentencia INPUT# interpretaría estos caracteres como tres cadenas:

```
90 INPUT #8, NAME1$, NAME2$, NAME3$
```

Para ilustrar el uso de las comillas como delimitadores, consideremos lo siguiente:

```
150 POSITION$="Duke"
160 FULLNAMES="ellington, Edward, Kennedy"
170 PRINT#3, POSITION$; FULLNAME$
```

La sentencia INPUT#

```
INPUT#3, POSITION$, FULLNAMES
```

asignaría Duke Ellington a POSITION$ y Edward a FULLNAMES

Para conseguir una entrada correcta, la sentencia PRINT # debe delimitar las cadenas como sigue (téngase presente que las comillas deben ser objeto de salida con el empleo de la función CHR$(34)):

```
170 PRINT#3, CHR$(34); POSITION$;CHR$(34);
             CHR(34);FULLNAME$;CHR$(34)
```

Esta sentencia escribirá lo siguiente para el número de fichero 3:

```
"Duke" "Ellington, Edward, Kennedy"
```

Una sentencia INPUT # subsiguiente interpretará las comillas como delimitadores. En consecuencia, las dos cadenas se introducirán adecuadamente.

La sentencia LINE INPUT # interpreta, de forma algo diferente, los datos del fichero secuencial. El formato de la sentencia LINE INPUT # es:

```
LINE INPUT # num.fich. var.cadena
```

El único delimitador que LINE INPUT # identificará es una secuencia de retorno del carro/avance de línea. La sentencia LINE INPUT # se utiliza cuando los espacios o las comas en un item de cadena son importantes y deben conservarse.

Cierre de un fichero secuencial

El paso final en el acceso a un fichero secuencial es el cierre del fichero. Siempre ha de cerrarse un fichero secuencial abierto cuando haya acabado con el mismo (por ejemplo, al final de un programa) o cuando se quiera utilizar el fichero en otro modo (salida, entrada o adición). Si el fichero no se cierra adecuadamente, pueden perderse datos.

La sentencia CLOSE cierra ficheros. Esta sentencia tiene el formato siguiente:

```
CLOSE # num.fich. , # num.fich. ...
```

Si no se da ningún parámetro de número de fichero, se cierran todos los ficheros actualmente abiertos. De cualquier otro modo, la sentencia CLOSE cierra solamente aquellos ficheros cuyo número de fichero se haya especificado.

El cierre de un fichero escribirá el contenido de una memoria intermedia que BASIC mantiene para ese fichero para el dispositivo en donde reside el fichero.

Un ejemplo de acceso de fichero secuencial

Reunamos, ahora, algunos de los conceptos anteriores para mostrar como crear y leer ficheros secuenciales.

A continuación se muestra un programa que crea un fichero secuencial de cadenas en la unidad de disco A. El fichero de disco se abre primero para salida de programa y luego se llena con la serie de cadenas que introduzca a partir del teclado. El fichero de disco se cierra cuando la letra Q se introduce por sí misma. El fichero se reabre luego para entrada de programa y la presentación visual se abre como un fichero secuencial para salida de programa. A continuación, el programa transfiere datos entre los ficheros de presentación visual y disco, efectuándose la lectura de las cadenas a partir del disco y haciéndoles salir a la presentación visual.

```
10  ' ****EJEMPLO DE FICHERO SECUENCIAL****
20  '
30  '    Este programa crea y llena un fichero secuencial en
40  '    un disco en la unidad A.  Los datos en el fichero están
50  '    constituidos por cadenas introducidas en el teclado.
60  '    Una vez que esté completo el fichero, el programa le
70  '    permitirá inspeccionar las entradas una a una.  Obser-
80  '    ve que, en este caso, se tiene acceso a la presentación
90  '    visual como un fichero secuencial para salida (esto es
100 '    lo mismo que si se hubieran utilizado las sentencias
    '    PRINT normales).
110 '    Este programa requiere BASIC Avanzado o de Disco y un
115 '    disco formateado para unidad de disco A.
120 CLS:KEY OFF
130 PRINT "INSTALAR DISCO PARA FICHERO EN UNIDAD A"
140 PRINT "PULSAR CUALQUIER TECLA CUANDO ESTE PREPARADO"
150 IF INKEY$=""THEN 160
160 CLS
170 OPEN "A:SEQTEST.DAT" FOR OUTPUT AS 1 ' Abrir fichero
        secuencial en unidad A
180 PRINT "INTRODUCIR CADENA A INTRODUCIRSE EN FICHERO"
190 INPUT"SI DESEA ACABAR FICHERO, INTRODUCIR Q-", A$
        ' Introducir cadena para pasar a fichero
200 IF A$="Q"THEN 260
210 PRINT #1, A$'Salida a fichero n.º1-"A:SEQTEST.DAT"
220 GOTO 180 ' Obtener otra entrada
230 '
240 ' Transferir ahora datos de fichero a fichero de
      visualización
250 '
260 CLOSE ' Necesita cerrar fichero secuencial antes de
          utilizarlo para entrada
```

```
270 CLS
280 PRINT"AHORA PUEDE INSPECCIONAR LAS ENTRADAS DE FI-
        CHERO UNA A UNA."
290 PRINT
300 PRINT "PARA VER CADA ENTRADA, TECLEAR ENTER"
310 PRINT
320 IF INKEY$=""THEN 320
330 OPEN "A:SEQTEST.DAT" FOR INPUT A$ 1 ' Volver a abrir el
        fichero para introducción a programa
340 OPEN "scrn:" FOR OUTPUT A$ 2 ' Esta vez utilizar la
        presentación visual como un fichero
350 INPUT #1, R$ ' Obtener siguiente registro a partir de
        "A:SEQTEST-DAT"
360 PRINT #2, F$'Salida a fichero de presentación visual
370 IF EOF(1) THEN 400 ' Comprobar marca de final de fichero
        en fichero de entrada
380 IF INKEY$=""THEN 380
390 GOTO 350 ' Obtener otro registro
400 CLOSE " Todo esta hecho; cerrar ficheros
410 PRINT
420 PRINT "¡Esto es todo¡"
430 END
```

FICHEROS DE DISCO ALEATORIOS

La caracterfstica singular de los ficheros de disco aleatorios es que están estructurados. En un fichero aleatorio, los items escritos en el fichero están organizados en unidades denominadas registros . Cada registro de un fichero aleatorio contiene uno a más items separados y colocados en el registro en un orden especffico.

A los ficheros aleatorios se tiene acceso para entrada y salida a razón de un registro cada vez. Los registros en un fichero aleatorio están numerados, lo que permite hacer referencia a un registro especffico por medio de su número, con lo que el acceso será inmediato. Por ejemplo, si se desea la lectura de un cierto item en un fichero, se puede conseguir en el registro que contiene el item sin necesidad de tener que leer a través de todos los registros que le preceden.

Acceso de fichero de disco aleatorio

Para tener acceso a un fichero aleatorio en un disco, se necesita abrir primero un canal para el fichero aleatorio, con lo que se asigna un número de fichero a un nombre de fichero. La apertura de fichero prepara también una memoria intermedia de fichero aleatorio en memoria. Esta memoria

intermedia es el enlace entre un programa y un fichero de disco.

Antes de que se pueda emplear la memoria intermedia de fichero, ha de especificarse el formato de los registros que se utilizará. En este caso, se definen la longitud y el orden de los items que comprende un registro.

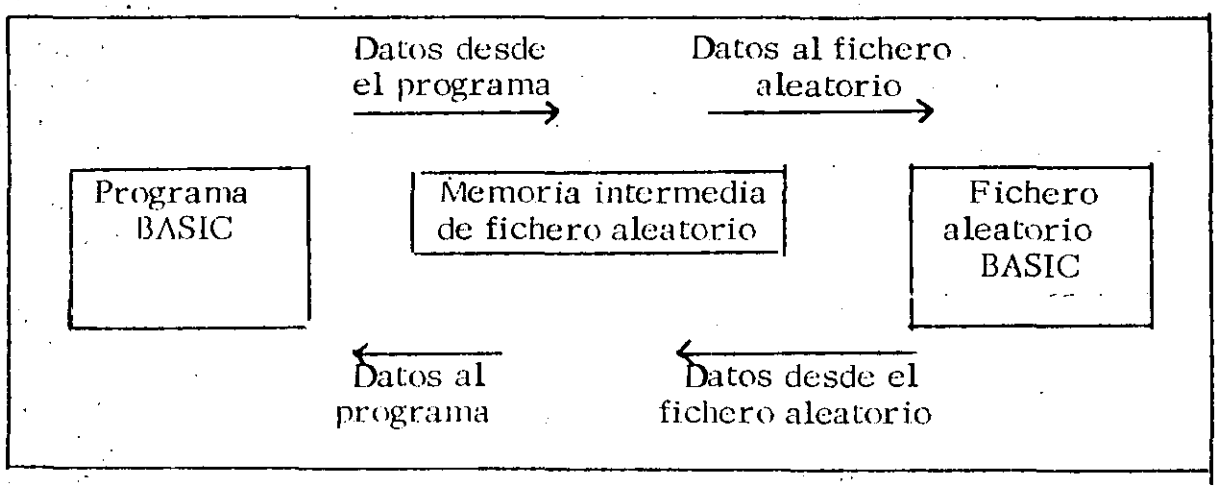
Los datos para el fichero se colocan en la memoria intermedia de fichero, con lo que se consituye un registro de fichero aleatorio. Cuando todos los items en un registro se hayan colocado en la memoria intermedia, el registro se tranfiere desde dicha memoria intermedia al disco, en donde reside el fichero.

Análogamente, para recuperar datos a partir de un fichero aleatorio, hay que cargar un registro desde el fichero a la memoria intermedia de fichero aleatorio. Los items en el registro se introducen, entonces, para el programa mediante acceso a la memoria intermedia. Obsérvese que siempre que esté abierto un fichero aleatorio, puede emplearse para salida o entrada, a diferencia con los secuenciales.

El proceso del acceso a un fichero aleatorio se muestra en la Figura siguiente.

Cuando se haya acabado con un fichero aleatorio, se le debe cerrar.

El cierre de un fichero aleatorio hace que cualquier dato en la memoria intermedia sea escrito para el fichero y también deja libre a la memoria, que estaba apartada, para encargarse del proceso de acceso a fichero aleatorio.



Apertura de un fichero aleatorio

Cuando se desea tener posibilidad de acceso a un fichero aleatorio, primero se debe abrir el fichero. El formato de la sentencia OPEN es:

OPEN FILESPEC AS [#] filenum [LEN=reclen]

ó

OPEN "R", [#] filenum, filespec [,reclen]

No tiene que especificarse entrada o salida en una sentencia OPEN de fichero aleatorio, puesto que los ficheros aleatorios pueden utilizarse para entrada y salida.

El parámentro "filespec" debe ser una expresión de cadena que proporcione un nombre de fichero válido. Obsérvese, sin emargo, que el único dispositivo permitido para ficheros es una de las unidades de disco (A o B). El parámetro "filenum" es una expresión entera tal como se define para la apertura de ficheros secuenciales.

Téngase presente que "reclen" no puede superar el valor especificado en la opción /S:b cuando se invoca primero BASIC, puesto que/S:b establece la longitud máxima de cualquier memoria intermedia de fichero aleatorio. El valor por defecto para b es de 128 octetos y el valor máximo para b es de 32.767.

Escritura de datos para la memoria intermedia

Una vez que se establezca el formato para un registro de un fichero, han de emplearse las sentencias LSET y RSET para colocar datos en los campos reservados de la memoria intermedia. El formato para estas sentencias es:

LSET stringvar=X$
RSET stringvar=X$

El parámetro "stringvar" es una variable de cadena definida en una sentencia FIELD anterior y X$ es una expresión de cadena. Si X$ tiene una longitud inferior a la que tiene la variable de cadena a la que está asignada. LSET justificará a la izquierda la cadena (rellenarla con espacios) en el espacio de memoria intermedia asignado a la variable. Análogamente RSET justificará a la derecha la cadena, volviendo a rellenar con espacios, en el campo adecuado de la memoria intermedia del fichero aleatorio. Si X$ tiene una longitud superior a la que tiene la variable de cadena, se suprimen los caracteres adicionales más a la derecha en X$.

Escrituras de registros para ficheros

Una vez que se haya construido completamente un registro (es decir, cuando se haya llenado la memoria intermedia asignando valores a todas las variables de cadena en el registro), el registro se escribe para el fichero con la sentencia PUT. El formato para este sentencia es:

PUT # filenum , número de registro

El parámetro filenum es el número de fichero especificado en la sentencia OPEN. Cuando no se especifica un parámetro de número de registro, PUT cargará el registro en la siguiente posición disponible, como si fuera un fichero secuencial. De cualquier otro modo, puede elegir a donde desea que vaya el registro en el fichero incluyendo el número de registro, que puede ser una expresión entera desde 1 a 32.767.

Por ejemplo, supongamos que el fichero B:PAYROLLDAT se ha abierto como número de fichero 3 en el ejemplo anterior. La siguiente sentencia colocará un nuevo registro en la novena posición en el fichero:

```
340 PUT #3, 10
```

La acción de esta sentencia puede ilustrarse como sigue:

Fichero B:PAYROLLDAT

Registro n.°	Datos
1	Brian 2.54
2	Sarah 5.44
3	Robin 23.90
.	.
.	.
.	.
10	George 2.13
.	.
.	.
.	.
300	Billy Joe .33

Memoria intermedia de fichero

George	2.13

Si no se quiere especificar un número de registro en una sentencia PUT subsiguiente, el siguiente registro escrito será el número de registro 11.

Lectura de los registros a partir de ficheros

Para poder recuperar registros a partir de un fichero aleatorio, ha de emplearse la sentencia GET, cuyo formato es:

GET # filenum numero de registro

El parámetro filenum es el mismo que para la sentencia PUT. El número de registro especifica qué registro ha de leerse en la memoria intermedia.

Por ejemplo, la siguiente sentencia efectuará la lectura del registro en el lugar 274 en la memoria intermedia:

```
540 GET #3, 274
```

La acción de esta setencia puede ilustrarse como sigue:

Fichero B:PAYROLLDAT

Registro n.°	Datos
1	Brian 2.54
2	Sarah 5.44
3	Robin 23.90
.	.
.	.
.	.
274	Dennis 2.11
.	.
.	.
.	.
300	Billy Joe .33

Memoria intermedia de fichero

Dennis	2.11

Si no se especificara un número de registro, se obtendrá simplemente el registro después del último leído. Por consiguiente, la siguiente sentencia leería, ahora, el registro en el lugar 275 del fichero en el ejemplo anterior:

```
590 GET #3
```

Si se pretende conseguir un registro que no haya sido anteriormente objeto de lectura, sucederá una de dos cosas. Si el número de registro es inferior al mayor número de registro escrito para el fichero, la lectura de datos es impredecible. Si el número especificado es superior al mayor número escrito, el registro recuperado estará constituido por cadenas nulas. Si un número de registro en una u otra sentencia GET o PUT es menor que 1 o mayor que 32.767, entonces, resultará un error Bad Record Number (número de registro incorrecto).

Lectura de los datos a partir de la memoria intermedia

Una vez que la sentencia GET haya cargado un registro en la memoria in-

termedia de fichero aleatorio, se puede tener acceso a los datos en la memoria intermedia utilizando las variables de cadena definidas en la sentencia FIELD para esa memoria intermedia o tratando a la memoria intermedia como un fichero secuencial e introduciendo caracteres con una sentencia INPUT# o LINE INPUT#.

Para conseguir un item específico de un registro recuperado, ha de utilizarse las variables de cadena definidas en la sentencia FIELD. Por ejemplo, supongamos que la memoria intermedia contiene los datos siguientes:

Beau	15.32

Puesto que la memoria intermedia se definió con la sentencia:

```
210  FIELD #3, 30 AS E$, 4 AS P$
```

podría leerse el nombre a partir de la memoria intermedia con la sentencia siguiente:

```
CURRENTNAME$ = E$
```

Ello asignaría el nombre de Beau a la variable de cadena denominada CURRENTNAME$

Ejercicios

1. - Encontrar el promedio de los primeros cien números enteros.

```
10  'NOMBRE : PROMEDIOS
20  SUM=0
30  I  =1
40  MIENTRAS (I < 100)

50        SUM=SUM+I
60        I=I+1
70  FIN (MIENTRAS)
80  SUM=SUM/100
90  PRINT "EL PROMEDIO DE LOS PRIMEROS 100 ENTEROS ES";SUM
100 END
```

2. - Multiplicar una matriz de 3 x 3 por un escalar

```
10  'NOMBRE: MULTIPLICA MATRIZ
20  DIM MATRIZ (3,3),  RESUL (3,3)

30  INPUT "DIGITE EL ESCALAR"; ESC%
40  FOR J=0 T02
50    FOR I=0 TO 2
60       READ MATRIZ (I,J)
70       RESUL (I,J) = MATRIZ (I,J)* ESC%
80    NEXT I
90  NEXT J

100 ' IMPRIME MATRIZ

110   FOR X=0 TO 2
120     FOR Y=0 TO 2
130       PRINT RESUL(X,Y),
140     NEXT Y
150     PRINT
160   NEXT X
170   END
180   DATA  1,2,4,8,7,6,5,9,2
```

3. - Obtener los números primos entre el 1 y el 100.

```
10  'NOMBRE : NUMEROS PRIMOS

20  FOR I=1 TO 100'
30      J=2
40      BAND=0
50      MIENTRAS (BAND=0 AND J < I)

60          IF (I/J - INT(I/J))=0 THEN BAND=1
70          J=J+1
80      FIN (MIENTRAS)
90      IF BAND=0   THEN PRINT I
100 NEXT I

110 END
```

4. - Generar 10 números aleatroios y ordenarlos ascendentemente.

```
10  'NOMBRE: CLASIFICA NUMEROS

20  DIM NUM-ALE (10)

30  FOR I=1 TO 10

40      NUM-ALE (I)= RDN(.3)*10
50  NEXT I
60  I=1
70  FOR K=J TO 10
80      MIENTRAS (I < 10)

100         IF NUM-ALE(K)>=NUM-ALE (I+1)  THEN GO TO 140
110         TEMO=NUM-ALE (I)
120         NUM-ALE (I)=NUM-ALE (K)
130         NUM-ALE (K)=TEMO

140         I=I+1
150     FIN (MIENTRAS)
160 NEXT K
170 FOR K= 1 TO 10
180     PRINT  NUM-ALE (K),
190 NEXT K
200 END
```

5. - Encontrar la solución a ecuaciones de 2° grado $ax^2 + bx = c = 0$

```
10  'NOMBRE: SOLUCION DE ECUACIONES
20  BAND=0
30  INPUT "DAME LOS COEFICIENTES A, B Y C"; A, B, C
40  MIENTRAS (A=0 AND B=0 AND C=0)
50      PRINT "ECUACION NULA, LOS COEFICIENTES SON CERO"
60      BAND=1
70  FIN (MIENTRAS)
80  IF BAND =0 THEN DTRMINANT =(B**Z) - (4*A*C)

90  MIENTRAS (DTRMINANT< 0 AND BAND =0)

100     PRINT "ERROR: NO HAY SOLUCION"
110     BAND =1
120 FIN IMIENTRAS)

130 MIENTRAS (BAND=0 AND A<>0)

140     X1=(B+DTRMINANT)/2*A

150     X2=(B-DTRMINANT)/2*A
160     BAND=1
170     PRINT'PRIMERA SOLUCION;X1
180     PRINT 'SEGUNDA SOLUCION; X2
190 FIN (MIENTRAS)
200 SINO$ = "NO"
210 INPUT"DESEAS RESOLVER OTRA ECUACION (S/N)"; SINO$
220 IF SINO$="S"  THEN GO TO 20
230 END
```

SISTEMAS Y PROGRAMAS

FORTRAN *

Ing. José Flores

* Notas tomadas del curso de Introducción a la Programación y Computación Electrónica, DECFI, Octubre 1983, impartido por Ing. Heriberto Olguín R., Ing. Antonio Pérez A. y M. en C. Ricardo Ciria M.

Octubre, 1984.

INTRODUCCION A LA PROGRAMACION Y COMPUTACION

ELECTRONICA

E Q U I P O

OCTUBRE, 1984.

ORDENADORES, MINIS Y MICROS

HASTA hace diez años el término ordenador abarcaba a cualquier equipo dedicado al proceso de datos. Hoy día, la evolución de estas máquinas nos obliga incluso a establecer su clasificación.

Tipos de ordenadores

Atendiendo a su configuración podemos distinguir tres tipos de ordenadores:

1. *Ordenadores analógicos*

Son aquellos que manejan señales eléctricas y suelen aplicarse a problemas de simulación. Su programación está plasmada (cableada) en los circuitos que lo integran.

2. *Ordenadores digitales.*

Admiten su programación por medio de lenguajes y manejan un alfabeto (código binario: 0-1) mediante el cual a través de cadenas de ceros y unos, se puede representar cualquier caracter.

3. *Ordenadores híbridos*

Participan de las caracteristicas de los dos anteriores. La entrada de datos suele estar controlada por un convertidor analógico/digital, la información es procesada por un ordenador digital y la salida es canalizada a través de un convertido digital/analógico.

En lo sucesivo nos referiremos siempre, excepto cuando se indique lo contrario, a ordenadores digitales.

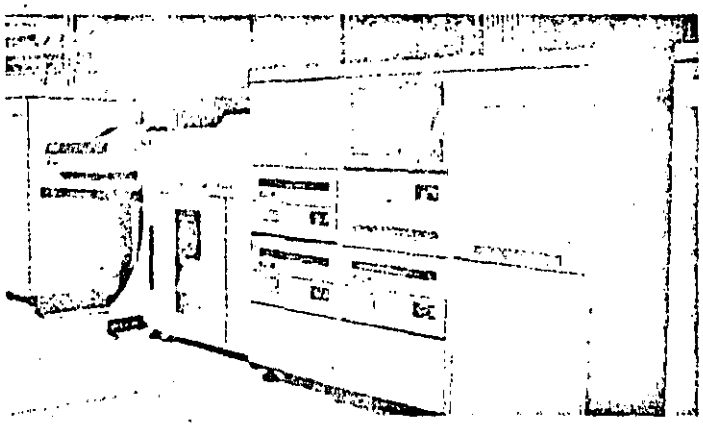
Los ordenadores digitales

Según la capacidad y potencia de esta categoria de sistemas, podemos distinguir tres clases de ordenadores bien diferenciados:

1. *Ordenadores.*
2. *Miniordenadores.*
3. *Microordenadores.*

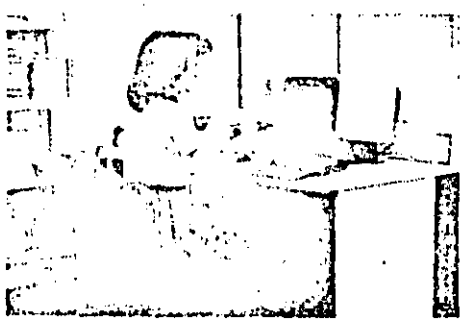
A medida que descendemos en esta escala, nos encontramos con equipos menos potentes aunque más baratos y versátiles.

En general cada uno de ellos tiene unas características ideales para un tipo de usuarios, por lo tanto ninguno anula a los anteriores, aunque existe la tendencia a la sustitución de los grandes equipos por sistemas de miniordenado-



La implantación de grandes equipos permite realizar complicados sistemas de proceso de datos. A cambio, obliga a instalaciones costosas y contar con personal de alta especialización.

La instalación de miniordenadores es bastante menos exigente que la de un gran equipo. Por lo demás su manejo es mucho más sencillo y no exige personal altamente especializado para su explotación.



El microordenador supone un nuevo paso en el acercamiento de la informática al usuario. Su empleo resulta tan cómodo como el de una simple máquina de calcular evolucionada.

Conceptos básicos

Hardware y software

Empezaremos por dar una definición de ambos conceptos:

Hardware

Si miramos un diccionario inglés-español, encontraremos como traducción de hardware, ferretería o quincallería. En efecto, nada más expresivo para definir las unidades físicas que constituyen un sistema de ordenador.

Software

En contraste con el equipo físico (hardware), se utiliza software para referirse a todos los programas que se pueden utilizar en un sistema de ordenador.

Si comparamos el ordenador con el cerebro humano, vemos que el hardware hace las veces de las memorias y demás componentes físicos del cerebro, mientras el software se encarga del soporte lógico que utiliza el cerebro para razonar.

Aunque hemos definido software como todos los programas utilizados en el ordenador, más especificamente, se aplica este término a aquellos programas que ayudan a sacar el máximo partido al equipo. En esta acepción, un programa encargado de la contabilidad de una empresa no formaria estrictamente parte del software, en cambio si formaria parte el programa de utilidad que se encargue de transcribir la información contenida en un lote de tarjetas perforadas a una cinta magnética.

En general, dado que un equipo no puede funcionar sin un software de base, este es suministrado por el fabricante junto con el ordenador. Posteriormente, y según las necesidades que surgan en la explotación, se puede ir ampliando o modificando. Los programs más usuales dentro del software de base son los siguientes:

— Programas para cálculos rutinarios.
— Programas de edición.
— Ensambladores y compiladores
— Sistema operativo.
— Programas de utilidad
— Programas de depuración.

ORDENADORES, MINIS Y MICROS

res o micro-ordenadores distribuidos, con lo que se gana en autonomía sin perder cohesión.

Ordenadores

Para el proceso de datos a gran escala, tanto en su componente de gestión como científica, es necesario el empleo de grandes equipos. Como ejemplo de aplicación científica, para la que resulta apropiado un gran ordenador, podemos citar el mantenimiento de una base de datos con la información de todos los cables de una central nuclear. En este caso no sólo hace falta una gran capacidad de almacenamiento, sino que para calcular recorridos ideales la potencia de cálculo debe ser grande. También en el campo de la gestión hay aplicaciones que sólo se pueden mantener con un gran ordenador; un ejemplo que en la actualidad está levantando muchas polémicas, es la mecanización de la información del censo de ciudadanos para los servicios de seguridad del estado. Normalmente, la adopción de grandes ordenadores obliga a realizar fuertes inversiones, tanto por lo caros que resultan los equipos como por las instalaciones auxiliares que necesitan: aire acondicionado, locales diáfanos y amplios, etc. También el equipo humano dedicado a su explotación debe ser numeroso. Por todo ello, sólo es recomendable su implantación si la complejidad o características de las aplicaciones no se adaptan a sistemas más asequibles.

Miniordenadores

El término miniordenador suele conducir a engaño; los equipos así denominados sólo son «mini» en el tamaño y precio, pero suelen prestar exactamente los mismos servicios que un ordenador mediano. Incluso si se distribuyen convenientemente y se conectan entre sí los miniordenadores necesarios, pueden sustituir con éxito a un equipo grande, evitando la centralización que éste supone y acercando al usuario final los equipos. De entre sus muchas aplicaciones podemos destacar las siguientes:

- *Control de procesos.*

En función de las señales que recibe el miniordenador, con las que se describe el estado de proceso, emite las señales necesarias para la correción del mismo. Algunos de los procesos controlados son: control de cadenas de montaje, operaciones de control de calidad, inspección de material, etc.

- *Comunicaciones.*

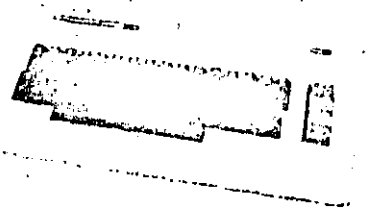
Tal vez en este área sea donde la evolución de los miniordenadores se encuentra en constante desarrollo, sus aplicaciones típicas son: reserva de plazas, transmisión de mensajes, etc.

- *Sistemas de información.*

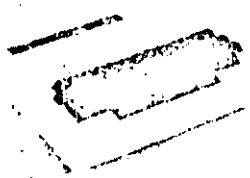
El miniordenador puede sustituir en algunos casos a equipos más grandes, realizando las típicas labores de mecanización como: sistemas comerciales, financieros, de gestión, científicos, etc.

Microordenadores

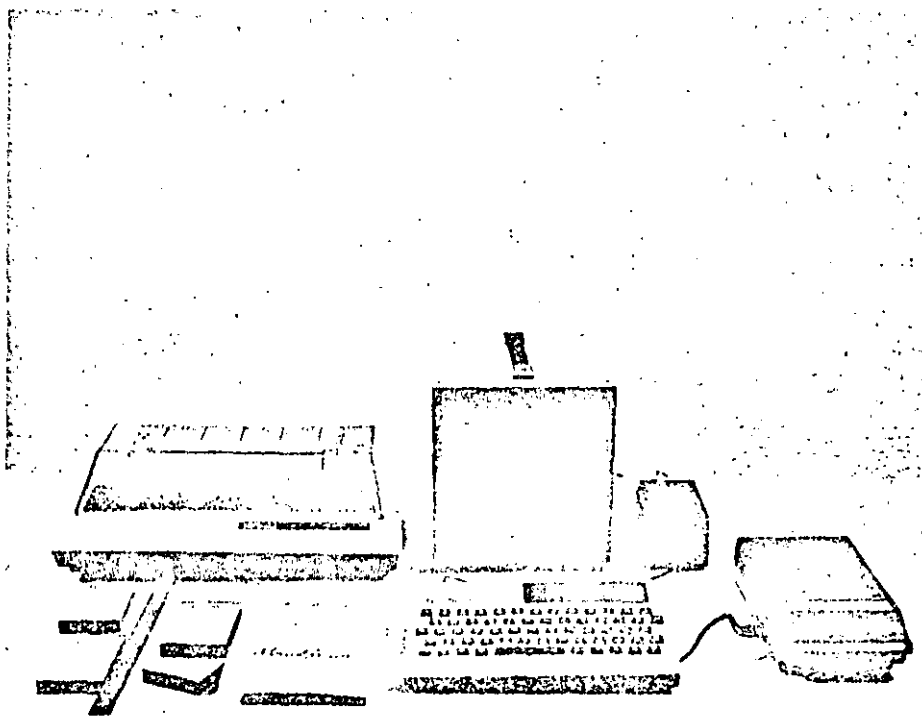
En la actualidad los microordenadores constituyen uno de los sectores más importantes del mercado informático. Cuando surgieron los primeros minior-



La irrupción de los ordenadores personales ha supuesto el definitivo acercamiento de los sistemas informáticos al usuario no especializado



Los ordenadores personales son pequeños microordenadores capaces de utilizar como dispositivos periféricos aparatos de tipo doméstico, como por ejemplo, receptores de TV, magnetófonos a cassette...



Con los ordenadores personales la informática abandona su característica de alto coste. Con una inversión razonable puede llegar a disponerse de un sistema microinformático de notables posibilidades de aplicación.

denadores se dedicaron a ofrecer unas prestaciones que no estaban cubiertas hasta ese momento, pero durante los años 70 los miniordenadores invadieron el sector más bajo de los grandes equipos. Los pasos se han vuelto a repetir con los micro-ordenadores; en la década de los 70, ocuparon un espacio vacío, si bien, en la actualidad, se están haciendo cargo de muchas de las áreas que anteriormente se cubrían con miniordenadores. Veamos algunas de las principales zonas de aplicación de los microordenadores:

• *Control de periféricos.*

Para conseguir descargar al ordenador principal de determinas tareas suele recurrirse a los microordenadores; estos se ocupan del control de los periféricos, por ejemplo: terminales, lectores de tarjetas, impresoras, etc.

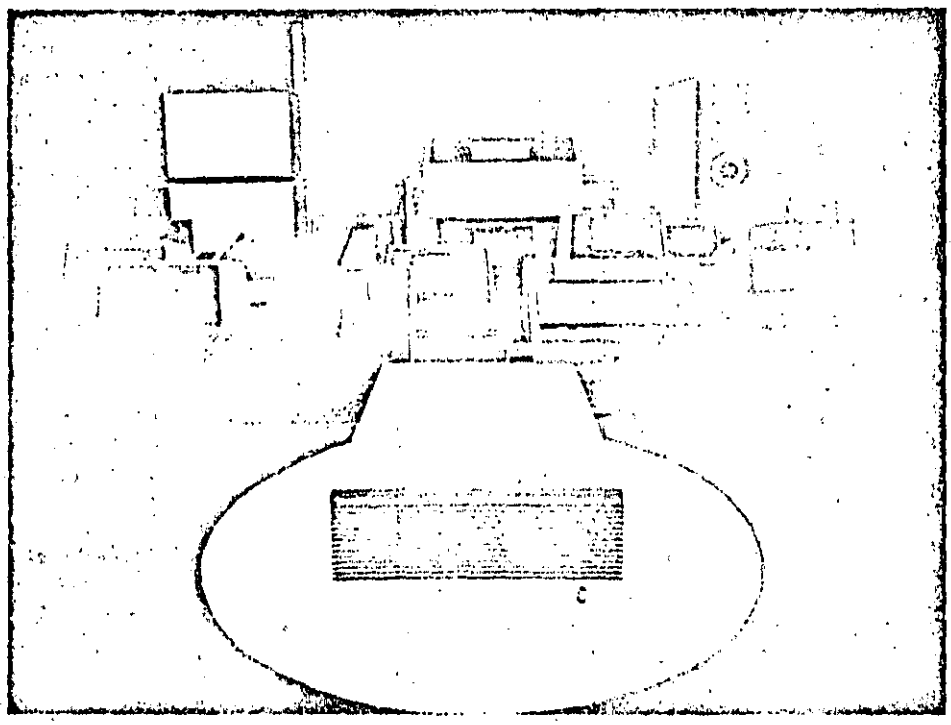
• *Toma de datos*

El microordenador puede recibir datos de diversas fuentes, los puede tratar según el programa que esté cargado, y reproducirlos en la unidad deseada. Normalmente, la forma de introducir los datos en el microordenador es mediante conexiones a instrumentos especiales de hospitales, laboratorios, etc.

• *Ordenadores personales.*

El bajo costo de los ordenadores personales (microordenadores) y la posibilidad que ofrecen de introducir la informática en el hogar, ha supuesto que su popularidad haya crecido espectacularmente.

Los ordenadores personales están basados en un microprocesador (un «cerebro», integrado en un espacio extraordinariamente reducido, capaz de dirigir, controlar y coordinar toda la actividad del sistema). Los ordenadores personales pueden conectarse a una memoria secundaria, generalmente cintas normales de cassette o discos flexibles. Su empleo es muy simple debido a que incorporan un sistema operativo interactivo y su programación se puede realizar en un lenguaje de alto nivel (normalmente BASIC). En los establecimientos especializados se pueden encontrar tanto los equipos como cualquier programa de los muchos desarrollados para las más diversas necesidades.



La profusión de periféricos especializados en las más diversas tareas, ha hecho que la informática se adapte a la prestación de los servicios más dispares, resolviendo cualquier tipo de tratamiento de información

Glosario

Ordenador:
Sistema electrónico dedicado al proceso de datos, con gran capacidad para el almacenamiento de datos y elevada velocidad de cálculo.

Miniordenador:
Sistema para el tratamiento de información de características (potencia, capacidad..., precio) inferiores a las de los ordenadores. Su estructura circuital se basa en la aplicación exhaustiva de componentes electrónicos de alta escala de integración.

Ordenador personal
Pequeño ordenador basado en un microprocesador. Todo ordenador personal es un microordenador, aunque no todo microordenador es un ordenador personal.

Sistema operativo
Conjunto de programas que supervisan el funcionamiento de un ordenador y facilita su utilización. Un ejemplo para aclarar su labor puede ser el siguiente: si el usuario quiere visualizar un programa en una pantalla se lo indica al sistema operativo que se encargará de buscar el programa en la memoria auxiliar en que se encuentre, pasarlo a la memoria principal, realizar la edición y avisar de cualquier anomalía que haya podido suceder.

Lenguaje de alto nivel
Lenguaje de programación que permite a los usuarios escribir programas mediante una notación con la que están familiarizados.

Lenguaje BASIC
Lenguaje de programación de alto nivel orientado al aprendizaje de las tareas de programación.

PERIFERICOS

TERMINALES

El terminal es un periférico de doble función: de entrada y de salida. El órgano que actúa como periférico de entrada es el teclado alfanumérico y el que actúa como periférico de salida es el monitor o pantalla de visualización. Este segundo órgano periférico (de salida) es, normalmente, un monitor de tubo de rayos catódicos similar a un receptor de televisión doméstico.
En un terminal cabe distinguir y evaluar cinco grupos de características:

- Relativas al teclado.
- Relativas a la pantalla.
- Relativas al conjunto operativo.
- Método de comunicación con el ordenador.
- Características físicas del conjunto.

Características del teclado

El teclado suele estar constituido por un bloque de teclas alfanuméricas, si bien la mayor parte de los teclados actuales incorporan, además, un teclado decimal de tipo calculadora para facilitar la introducción de datos numéricos.

- *Tipo de teclado:* Atendiendo a la distribución de las teclas cabe distinguir dos categorías de teclados:

a) De tipo QWERTY.
b) De tipo AZERTY.

La clasificación responde al orden de las teclas alfabéticas situadas en la fila superior, empezando por la tecla situada más a la izquierda.
Por lo que respecta a las teclas que conforman el teclado se define una segunda clasificación relativa a la forma en que éstas establecen el contacto:

a) Mecánicas.
b) De contacto reed.
c) Capacitivas.
d) De núcleo magnético.
e) De efecto Hall.

Aunque el teclado no sea de tipo mecánico, es posible obtener un «click» audible a título de realimentación fisiológica que permita el reconocimiento de la pulsación.

- *Teclas especiales:* El teclado suele incorporar teclas que corresponden a funciones especiales definidas por el fabricante. Esta opción simplifica la tarea de introducción de órdenes a través del teclado.

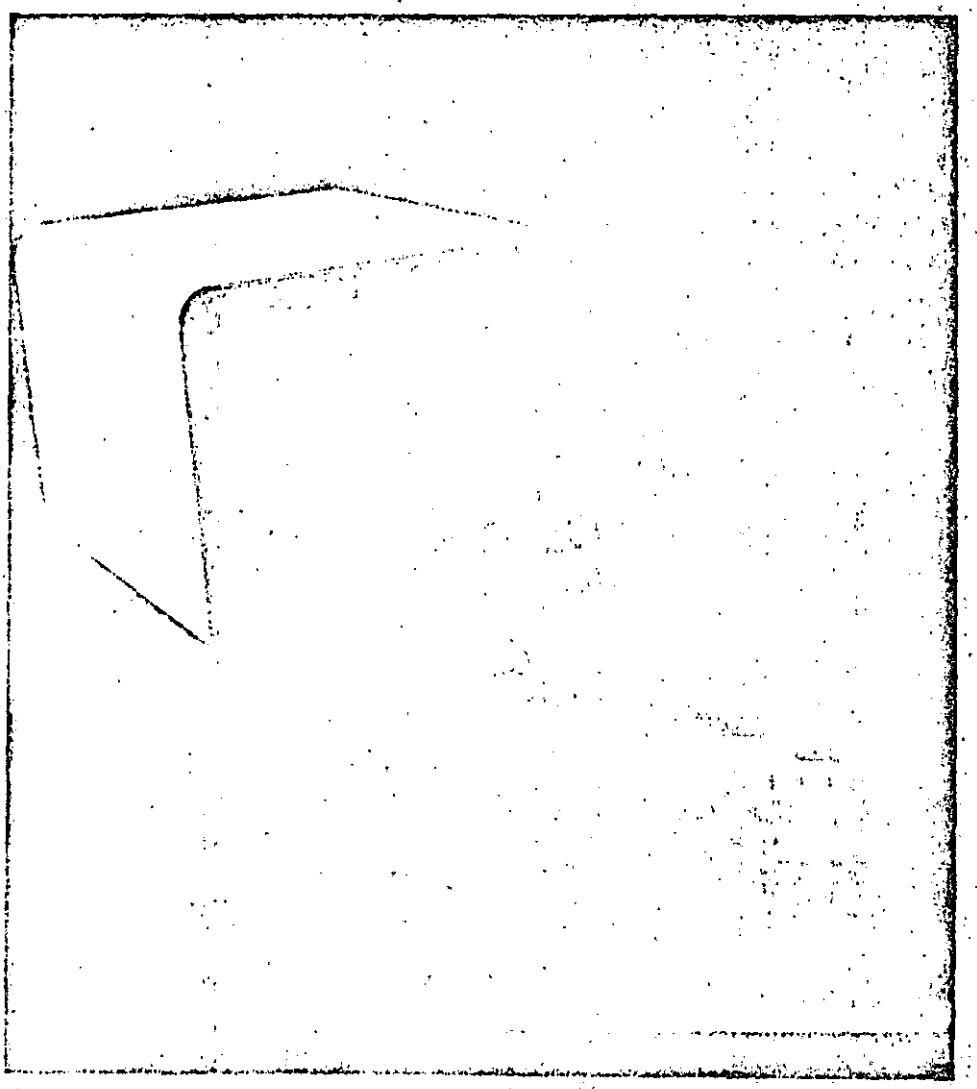
- *Pulsación de varias teclas simultáneamente:* Para evitar la aparición de errores al pulsar varias teclas a la vez, los teclados suelen acogerse a uno de los tres métodos que se indican a continuación:

1. Sobrepulsación de dos teclas (2 Key rollover): cuando se pulsa una tecla, las demás quedan bloqueadas hasta no soltar la primera.
2. Inhibición de N teclas (N Key lockout): cuando se pulsan varias teclas a la vez no se genera salida.
3. Sobrepulsación de N teclas (N Key rollover): cuando se pulsa una tecla se genera su código y al pulsar otra a la vez, se genera el código de la segunda.

La tercera solución es la que permite una escritura más rápida, aunque el primero de los métodos suele ser suficiente al visualizar el resultado de las pulsaciones sobre la pantalla.

Características de la pantalla

- *Tipo de pantalla:* Normalmente, se emplea un tubo de rayos catódicos similar a un receptor de televisión, aunque hay terminales de tipo portátil que utilizan un display de cristal líquido o de descarga de gas. Actualmente, se empiezan a utilizar como periféricos de visualización las pantallas de plasma, con lo cual se reducen las dimensiones y los caracteres alcanzan una mejor definición.



Los terminales son periféricos que tienen como misión introducir en un ordenador programas o datos, así como obtener de él información o resultados. Los terminales están formados por una pantalla y un teclado.

TERMINALES

• *Monócroma o color:* Las pantallas de tubo de rayos catódicos pueden ser monócromas (un solo color sobre fondo distinto) o de color (empleadas, normalmente, en terminales con posibilidad de gráficos). Las pantallas monócromas se suelen emplear en colores verde, blanco y ámbar y, normalmente, tienen dos posibilidades:

— Video normal: los caracteres aparecen iluminados sobre un fondo oscuro.
— Video invertido: los caracteres aparecen en color oscuro (color del fondo en video normal) sobre fondo de color claro (color de presentación en video normal).

• *Tamaño de la pantalla:* Se indica por la medida de su diagonal expresada en pulgadas.

• *Número de líneas:* Equivale al número de filas horizontales para la visualización de caracteres que caben en la pantalla. Un número de líneas habitual es de 24 ó 25. Cuando se ha ocupado la totalidad de la pantalla y se sigue escribiendo, hay dos posibilidades:

1. Scroll: Todas las líneas suben una posición, desapareciendo la primera de ellas y quedando la línea inferior libre para recibir los nuevos caracteres.
2. No Scroll: Se pasa a escribir en la primera línea de la pantalla, borrándose los caracteres escritos anteriormente según se van introduciendo los nuevos.

Estas dos posibilidades son seleccionables, normalmente, actuando sobre un microinterruptor interno.

• *Número de caracteres por línea:* Es el número de caracteres que cabe en cada línea visualizada en la pantalla. Un número frecuente de caracteres es de 80 por línea.

• *Capacidad de gráficos:* Depende del circuito electrónico denominado «controlador de pantalla» asociado al terminal. Lo más corriente es que el terminal disponga, al menos, de un juego de caracteres de los denominados semigráficos.

Características del conjunto operativo

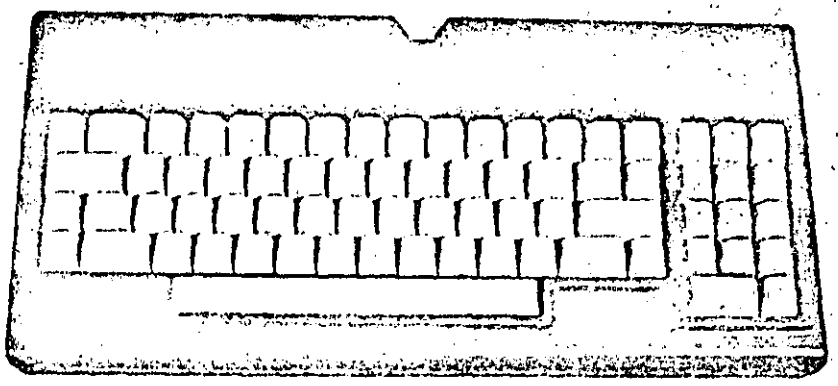
• *Control del cursor:* El cursor puede ser llevado al comienzo de pantalla (HOME: esquina superior izquierda), al principio de línea y a una posición anterior o posterior a la actual. De los desplazamientos se ocupan las teclas «de control del cursor».

• *Juegos de caracteres:* Al igual que en las impresoras, es posible seleccionar —mediante microinterruptores internos— la posibilidad de escritura de caracteres propios de diversos idiomas (Ñ, española; ß, alemana ...).

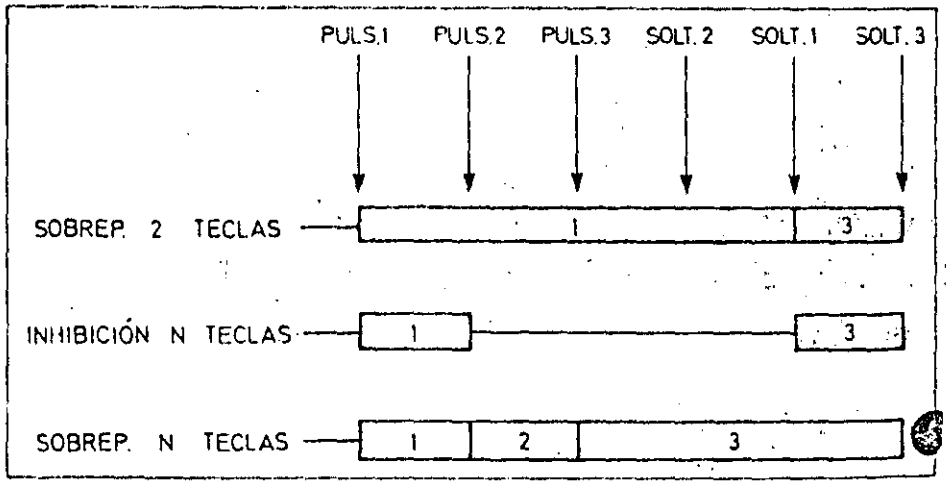
• *Zona de memoria:* El terminal debe disponer de una cierta cantidad de memoria RAM para utilizarla como buffer o almacén temporal de los datos a visualizar en la pantalla. Dependiendo de esta cantidad de memoria se podrá almacenar el contenido de varias pantallas para su posterior visualización. Esta posibilidad resulta de gran interés en los terminales orientados a la generación de gráficos, ya que puede ser necesario alternar la visualización de diversos gráficos correlativos.

• *Inteligencia:* Algunos terminales son sistemas electrónicos inteligentes, basados en microprocesador y con una notable zona de memoria. Estos son capaces de realizar ciertas funciones por sí mismos, sin necesidad de tener que recurrir al ordenador central.

• *Posibilidad de conexión de impresora:* La mayoría de los terminales permiten el acoplamiento directo de una impresora externa. A veces, también, admiten la incorporación directa de unidades de almacenamiento: disco, cinta, etc.



El teclado está formado normalmente por un bloque de teclas alfanuméricas y un bloque de teclas decimales que facilitan la introducción de datos numéricos. Algunos teclados incorporan, además, una serie de teclas a las que pueden asignarse funciones especiales.



Un problema que se presenta normalmente con cualquier tipo de teclados es el que se produce cuando se pulsan dos o más teclas simultáneamente. Estas son las soluciones más comúnmente adoptadas.

Comunicación con el ordenador

• *Tipo de interface:* Las interfaces de comunicación más empleadas en terminales son las de tipo serie:

— RS 232.

— Bucle de 20 mA.

• *Velocidad de transmisión de datos:* Es la velocidad con la que se transfieren los datos entre el ordenador y el terminal. Se expresa en «baudios» o en bits por segundo. Los terminales suelen admitir la selección de distintas velocidades de transmisión, lo que les permite adaptarse a las características de cualquier ordenador. Algunas velocidades normalizadas son 600, 1.200, 2.400, 4.800 y 9.600 baudios.

• *Modo de transmisión:* La comunicación entre el ordenador y el terminal puede realizarse de acuerdo a uno de los dos «modos» siguientes:

1. Half duplex: una línea en los dos sentidos.

2. Full duplex: dos líneas, una en cada sentido.

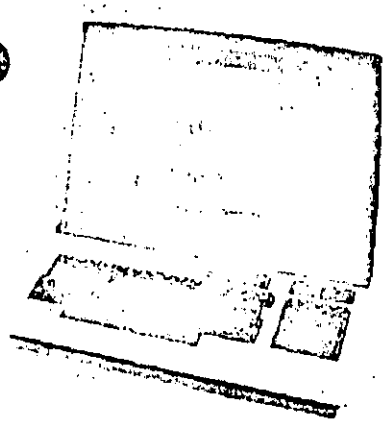
• *Control de paridad:* La detección de errores en la comunicación se realiza detectando la condición de paridad par o impar transmitida a través de uno de los bits de cada palabra binaria.

La característica de bit de paridad par o impar suele ser seleccionable.

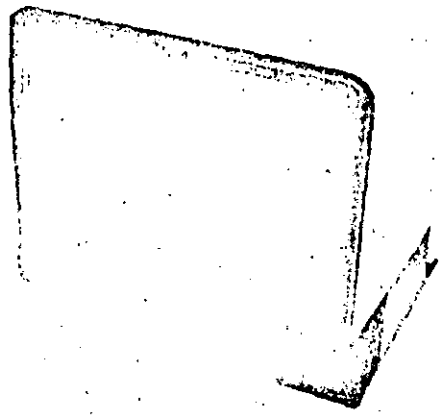
Características físicas del conjunto

• *Teclado integrado o separado:* El teclado puede estar incluido en el mismo mueble que el resto del terminal o en un soporte independiente, unido al resto mediante un cable de conexión. Esta segunda posibilidad es la que resulta más cómoda y ergonómica.

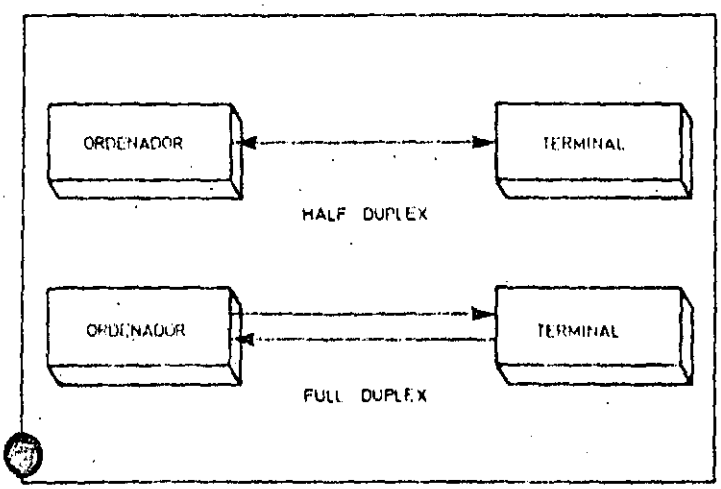
Como características adicionales relativas a la estructura física del terminal conjunto cabe citar el consumo energético y su peso y dimensiones.



Hasta hace poco la mayoría de los terminales integraban en una misma carcasa el teclado y la pantalla. En la actualidad se tiende a separar estos dos elementos.



La pantalla está formada por un tubo de rayos catódicos similar al utilizado en los receptores de televisión. Actualmente se están haciendo ensayos con pantallas de cristal líquido y con pantallas de plasma.



Existen dos formas mediante las cuales se comunican los terminales con el ordenador: una línea en los dos sentidos (half-duplex) y dos líneas, una en cada sentido (full-duplex).

Los terminales portables que pueden conectarse al ordenador a través de línea telefónica pueden ser de gran utilidad para revisar precios y stocks en los grandes almacenes o en los supermercados.

DISCOS MAGNETICOS

LOS discos son soportes de tipo magnético que se utilizan para el almacenamiento de la información en los sistemas ordenadores. Actualmente los discos son el principal medio de almacenamiento que utilizan los ordenadores que requieren un rápido acceso a los datos en forma aleatoria.

Al hablar de discos hay que hacer una primera distinción o clasificación:

1. Discos rígidos o duros.
2. Discos flexibles (floppy disk).

Discos rígidos

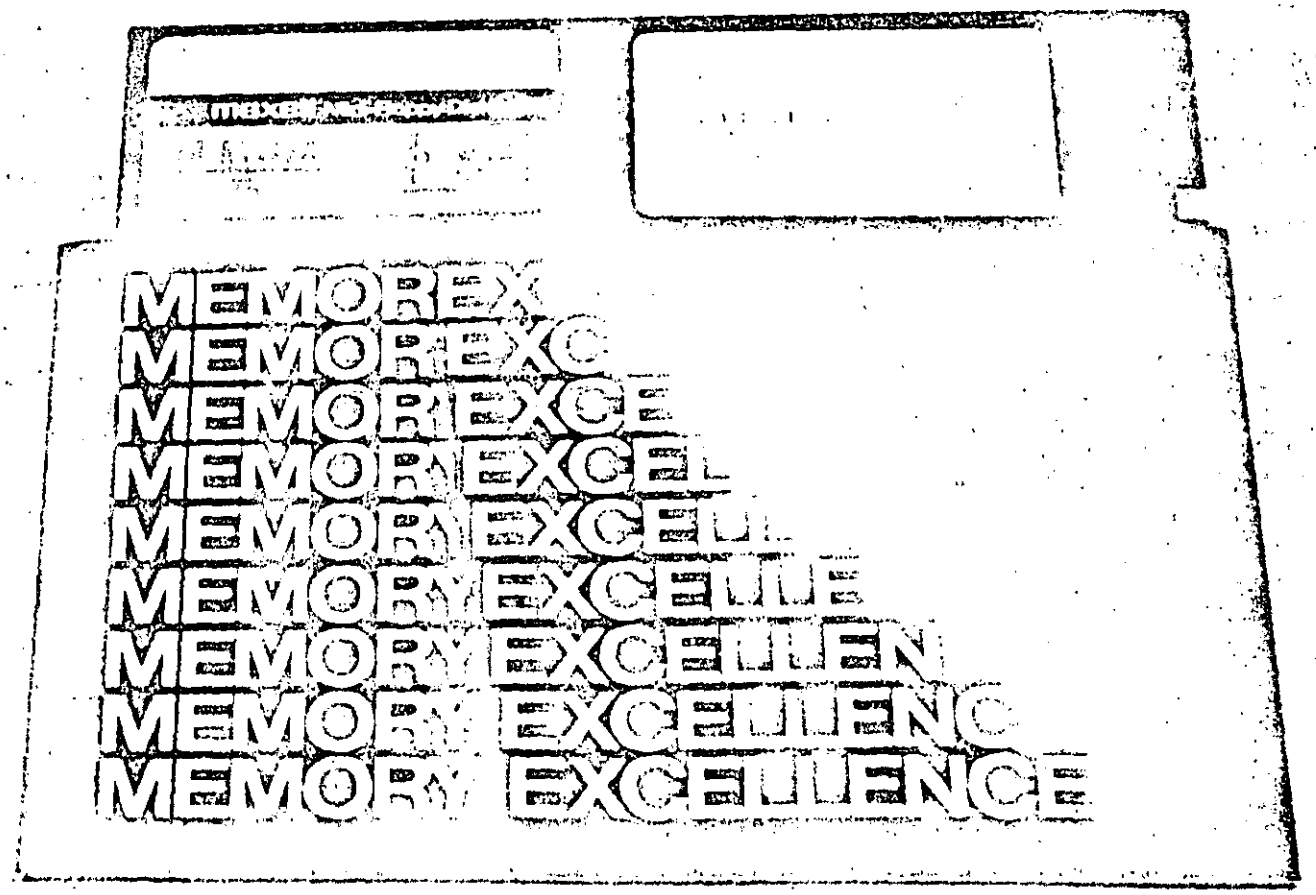
Los discos rígidos suelen estar construidos a partir de una base de aluminio recubierta de un material magnético sobre el que se graban los datos. Los tamaños normalizados que se emplean son de 14" y 8", siendo ésta la medida de su diámetro, existiendo últimamente también discos rígidos de 5 y 1/4 pulgadas.

Los discos rígidos pueden ser fijos o removibles. Los discos fijos vienen ya en su unidad de lectura y escritura y no pueden extraerse de la misma. Los discos removibles vienen normalmente en un contenedor especial para facilitar su manejo, denominado disk-pack. Normalmente estos contenedores llevan más de un disco rígido, unidos todos ellos mediante un eje, con lo que se consiguen unas capacidades de almacenamiento de datos del orden de los 100 megabytes por unidad contenedora.

Los discos rígidos fijos pueden ser de tecnología Winchester (lanzada por IBM en el año 1973), caracterizada porque la cabeza de lectura no toca físicamente al disco, sino que, por efecto aerodinámico de rotación del disco a una velocidad de unos 160 Km/h, el aire arrastrado hace que la cabeza de lectura permanezca suspendida a unas micras de distancia del disco, distancia suficientemente pequeña para que los datos puedan leerse y escribirse.

Discos flexibles

Los discos flexibles están hechos de material plástico de Mylar, recubierto



Los discos magnéticos son los soportes de información más utilizados en el campo de los sistemas microordenadores. Dentro de esta categoría de soportes, el predominio corresponde a los discos flexibles también denominados disquettes.

DISCOS MAGNETICOS

de una capa de óxido magnético. Poseen un agujero central que les sirve para encajar en el mecanismo de rotación y un pequeño agujero de control en sus proximidades, que sirve como índice para referenciar el comienzo de cada pista. El disco se protege mediante una cubierta de cartón cuyo interior es antiestático y autolimpiante. Una abertura en este envoltorio de protección permite a la cabeza lectora el acceso a los datos.

Los discos flexibles suelen ser de tres tamaños:

- 8 pulgadas.
- 5 y 1/4 pulgadas.
- Microfloppies

Los dos primeros son tamaños normalizados de diámetro del disco, mientras que los microfloppies —que son los más recientes— no tienen todavía un tamaño normalizado; los diversos fabricantes actuales producen microfloppies de 3", 3 1/4", 3 1/2" y 4".

La lectura de la información contenida en el disco flexible se realiza mediante una cabeza lectora que entra en contacto directo con el disco a través de la ranura practicada en la funda de protección. Hay que abstenerse, por tanto, de tocar los discos sobre dicha ranura. Esta hay que protegerla del polvo, así como proteger el disco de una temperatura elevada que pueda causar su deformación induciendo a errores en la lectura de los datos.

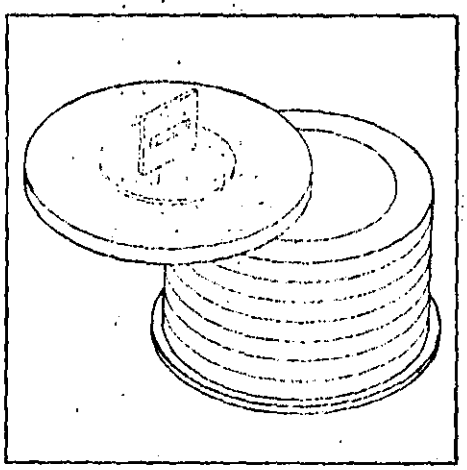
Características

La información se graba en el disco sobre pistas circulares, no en forma de espiral como ocurre en un disco de música. Para pasar a leer información de una pista a otra, la cabeza lectora debe desplazarse concéntricamente.

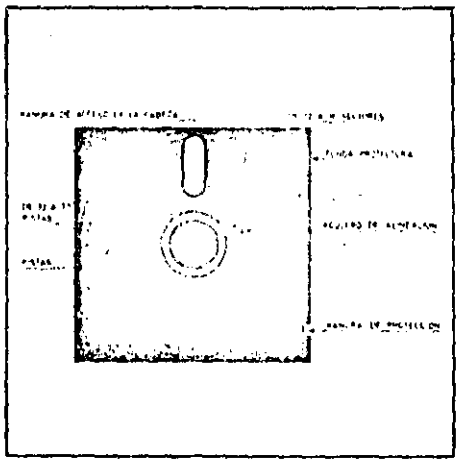
El disco se considera dividido en varias secciones llamadas *sectores*. Un sector es la parte mínima de disco que el sistema es capaz de leer o escribir. Un sector de una pista contiene 128 ó 256 bytes de información en un disco flexible, y 256 ó 512 bytes, en un disco rígido.

Las características más importantes a considerar en los diversos tipos de discos son:

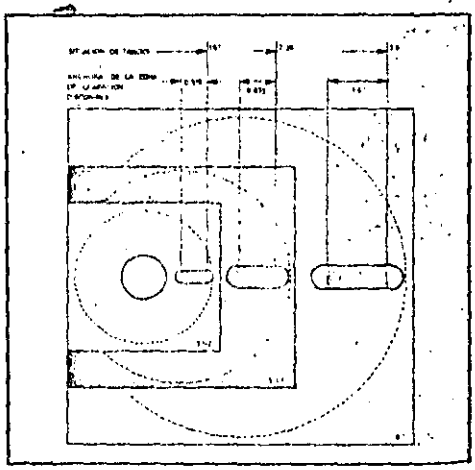
1. *Capacidad total de almacenamiento:* Es la cantidad de bits de información que puede almacenar el disco y, por tanto, una de sus características más importantes. Esta capacidad suele medirse en múltiplos del «byte» (pala-



Los discos rígidos múltiples de tipo removible están alojados normalmente dentro de un contenedor (disk-pack) que los protege y facilita su manipulación.



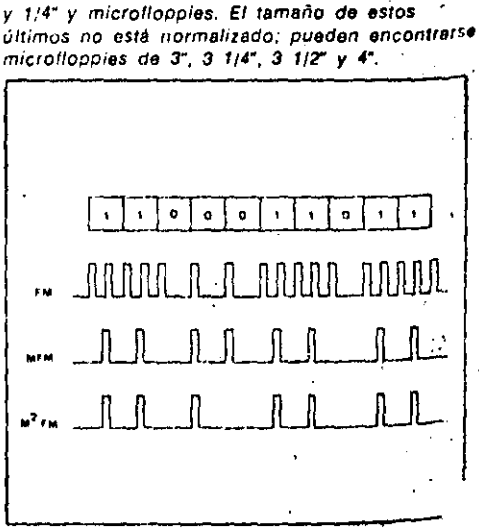
Los discos flexibles, fabricados con material plástico, se presentan dentro de una funda protectora de cartón cuya superficie interior es antiestática y autolimpiante.



Los discos flexibles suelen ser: de 8", de 5" y 1/4" y microfloppies. El tamaño de estos últimos no está normalizado; pueden encontrarse microfloppies de 3", 3 1/4", 3 1/2" y 4".

MICROFLOPPIES MAS COMUNES

FABRICANTE	HITACHI	TABOR	SONY	TANDON	IBM
Tamaño	3"	3 1/4"	3 1/2"	3 1/2"	4"
Método de codificación	FM/MFM	FM/MFM	FM/MFM	FM/MFM	FM
Densidad (bpi)	4473/8946	4625/9250	3805/7610	3128/6255	6865
Capacidad por cara (Kbits)	125/250	250/500	218,8/437,5	125/250	358
Veloc. rotación (r.p.m.)	300	300	600	300	262 a 415
Tiempo de acceso (ms)	3	10	15	3	40
Pistas por cara	40	80	70	40	46
Número de caras	2	1	1	2	1



La «densidad» —simple o doble— es una característica de los discos magnéticos que depende del método de codificación adoptado: FM, MFM o M²FM.

bra binaria de 8 bits), como son el Kilo-byte y el Mega-byte. Los discos rígidos tienen mayor capacidad que los flexibles, debido a su propia tecnología; el aluminio tiene menor deformación con los cambios de temperatura que el material plástico y, por tanto, sus pistas pueden estar más próximas unas de otras.

La capacidad, además de depender del tamaño del disco, depende también de otras características que se verán a continuación.

2. *Número de pistas:* Es el número de pistas circulares en las cuales se almacena la información. Normalmente se indica como característica la densidad de pistas; esto es, el número de pistas por pulgada (TPI).

3. *Número de caras:* Los discos pueden estar grabados por una sola cara o por las dos caras, con lo cual aumenta la capacidad de almacenamiento.

4. *Simple o doble densidad:* Los discos pueden estar grabados con tres codificaciones distintas:

— FM: Modulación de frecuencia. Al comienzo de cada célula de bit se escribe un impulso de sincronismo y luego hay un impulso en el medio de la célula si el bit es 1, no habiéndolo si es un cero.

— MFM: Modulación de frecuencia modificada. Cuando el bit es un 1 se cambia la dirección de la magnetización en el medio de la célula, pero suprimiendo la señal de sincronismo. Si el bit es un cero se da sólo la señal de sincronismo, pero si el cero esta entre dos unos, se suprime la señal de sincronismo y la señal de dato.

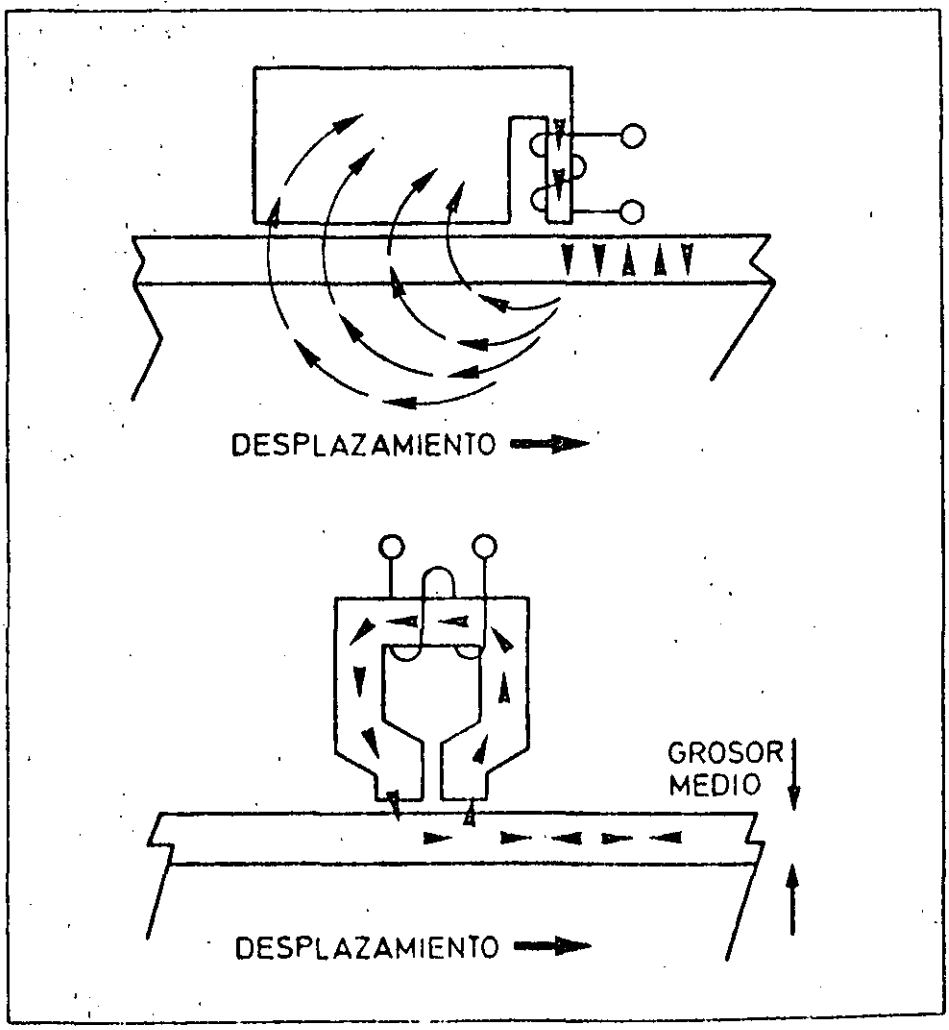
— M²FM: Modulación de frecuencia doblemente modificada. Se escribe una señal de dato si la información es un 1 y se escribe una señal de sincronismo al comienzo de la célula si la célula es un cero y no había ninguna señal ni de dato ni de sincronismo en la célula anterior.

Cuando el disco está grabado con la codificación FM se dice que es de simple densidad y cuando está grabado con las codificaciones MFM o M²FM, se dice que es de doble densidad. Estos últimos sistemas permiten almacenar un mayor número de bits de información por pulgada del disco, ya que eliminan la grabación de muchos impulsos de sincronismo.

5. *Grabación vertical o longitudinal:* Los bits de información pueden estar orientados en el material magnético de forma vertical o de forma longitudinal. La tecnología de grabación vertical actualmente sólo es posible en los discos rígidos, pero con esta tecnología se podrá conseguir en el futuro un gran aumento de la densidad de información por pulgada en los discos flexibles.

6. *Velocidad de rotación:* Se expresa en r.p.m. y es la velocidad de giro del disco alrededor de su eje.

7. *Tiempo de acceso:* Se suele dar como característica el tiempo de acceso de pista expresado en milisegundos.



Otra de las características de los discos magnéticos es el sistema de grabación empleado: vertical (gráfico superior) o longitudinal (gráfico inferior), denominación que deriva de la orientación con la que se grabe la información en el material magnético.

Comparación entre discos rígidos y flexibles

— Los discos rígidos tienen una mayor capacidad de almacenamiento que los flexibles, debido al hecho esencial de la mayor densidad de pistas por pulgada. Un disco de tecnología Whinchester de 8 pulgadas tiene, por ejemplo, más capacidad que 4 discos flexibles de 8 pulgadas. En modelos equiparables, la capacidad de almacenamiento de un disco rígido es del orden de 20 Mbytes, mientras que los discos flexibles tienen capacidades de 1 Mbyte.

— El tiempo de acceso a la información es menor en los discos rígidos que en los flexibles.

— Sin embargo, los discos flexibles son mucho más manejables que los discos rígidos, siendo su precio muy inferior, así como el coste de la unidad correspondiente de lectura y escritura.

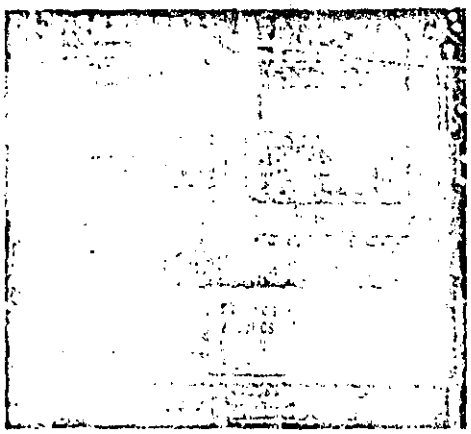
LA UNIDAD CENTRAL DE PROCESO

TAL como ya se ha indicado en anteriores capítulos, la unidad central de proceso (CPU) es el verdadero «cerebro» del ordenador. Sabemos también que en los microordenadores, la unidad central de proceso está integrada en un chip denominado microprocesador.

La misión encomendada a la CPU es la de ejecutar los programas, tanto los pertenecientes al software de base, como los de aplicación o los creados por el propio usuario. En ambos casos, es necesario procesar instrucciones, operar datos y controlar la actuación de las unidades implicadas.

A la hora de estudiar la arquitectura de la unidad central de proceso cabe distinguir entre su estructura exterior (líneas y buses de comunicación) y su organización interna.

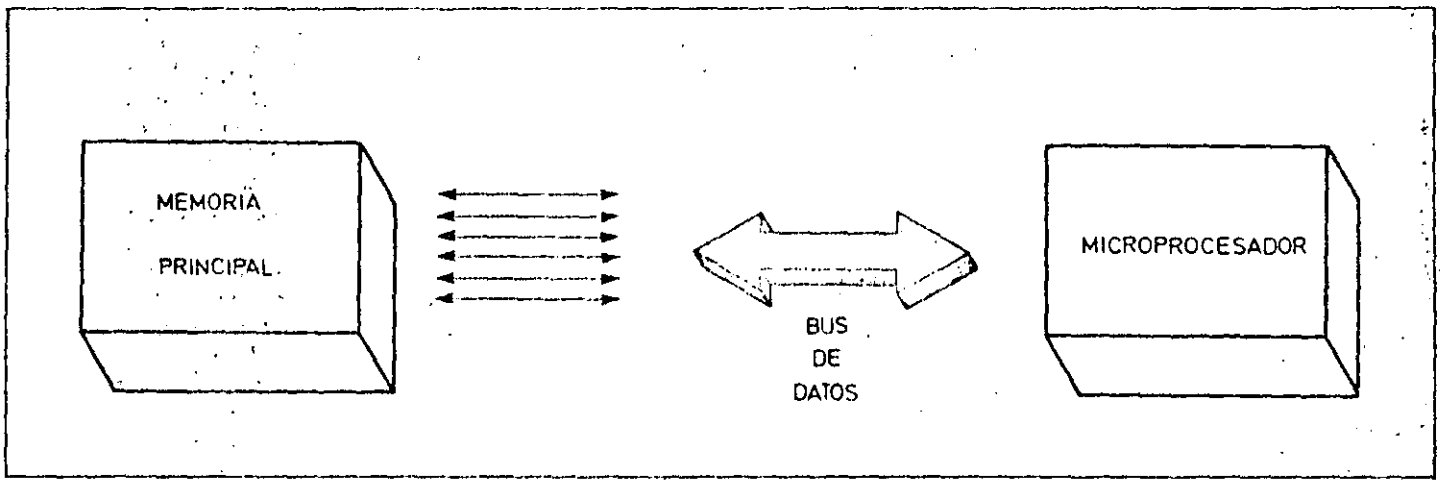
La estructura externa de la CPU es la que le permite comunicarse con las restantes unidades que integran el sistema ordenador. Esta, consta de tres buses o grupos de líneas: bus de datos, bus de direcciones y bus de control.



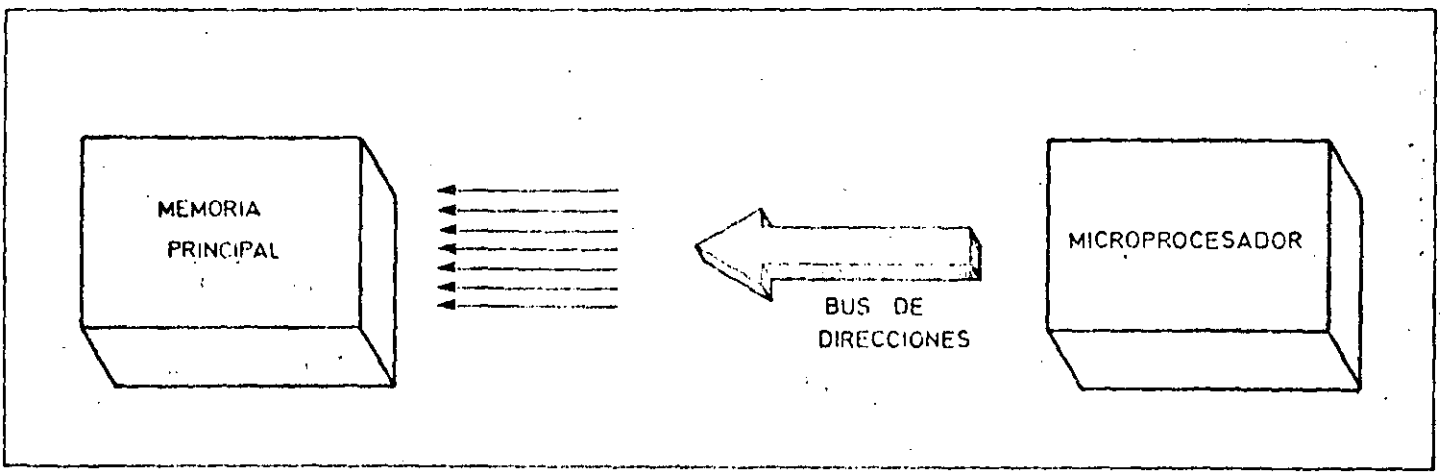
El microprocesador constituye la unidad central de proceso de los microordenadores. En la fotografía aparece la CPU del microordenador OLIVETTI M.20: el microprocesador de 16 bits Z 8001 de la firma Zilog.

Bus de datos

Es utilizado por la CPU para realizar el intercambio de instrucciones y datos con el exterior; este intercambio se realiza a través de un conjunto de líneas, una por cada bit, tanto desde la CPU



El bus de datos permite establecer el intercambio de información entre el microprocesador (CPU de los sistemas microordenadores que ocupan nuestro interés) y las restantes unidades del sistema.



El bus de direcciones canaliza los bits de las palabras binarias de dirección. La palabra transferida apunta a la posición en la cual se va a escribir o a leer información a través del bus de datos.

LA UNIDAD CENTRAL DE PROCESO

hacia el exterior como en sentido inverso.

Una de las características principales de un microprocesador (CPU de los microordenadores) es el número de bits que puede transferir el bus de datos (4, 8, 16...). Cuanto mayor sea este número, más tipos de instrucciones y datos se podrán manejar, con lo que se facilitará la labor al usuario.

Bus de direcciones

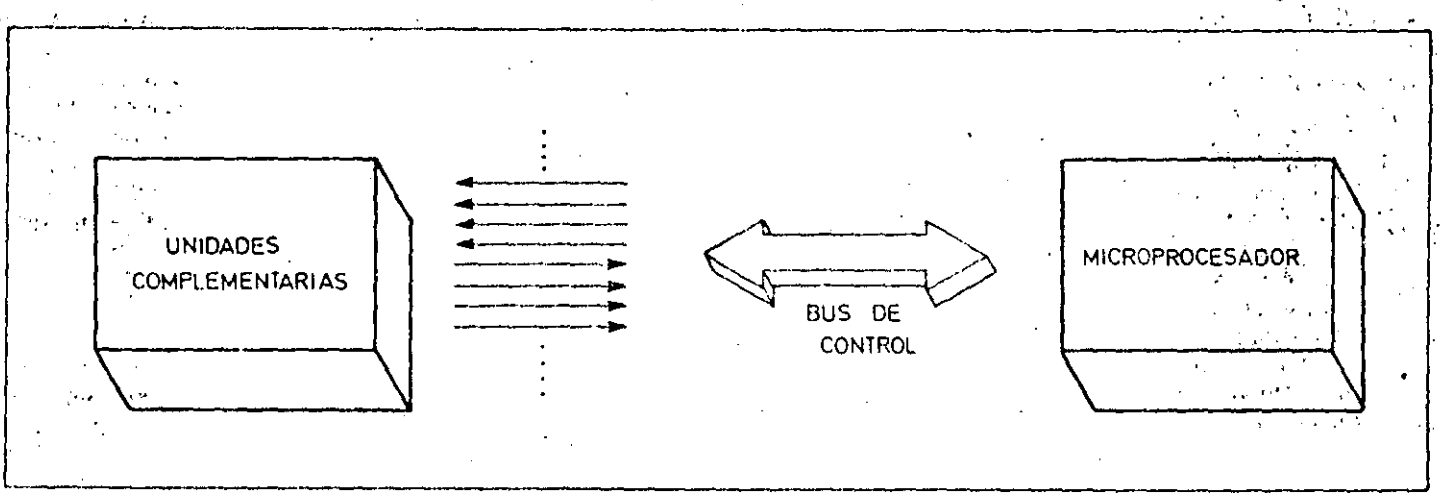
Consiste en un canal constituido por líneas de direcciones que indican la posición de memoria en la que se encuentra la información a tratar. Una vez direccionada la posición, la información almacenada pasará a la CPU a través del bus de datos. Para determinar el volumen de memoria directamente accesible por la CPU, hay que tener en cuenta el número de líneas que integran el bus de direcciones de la CPU. Generalmente, los microprocesadores de 8 bits disponen de un bus de direcciones de 16 líneas.

Bus de control

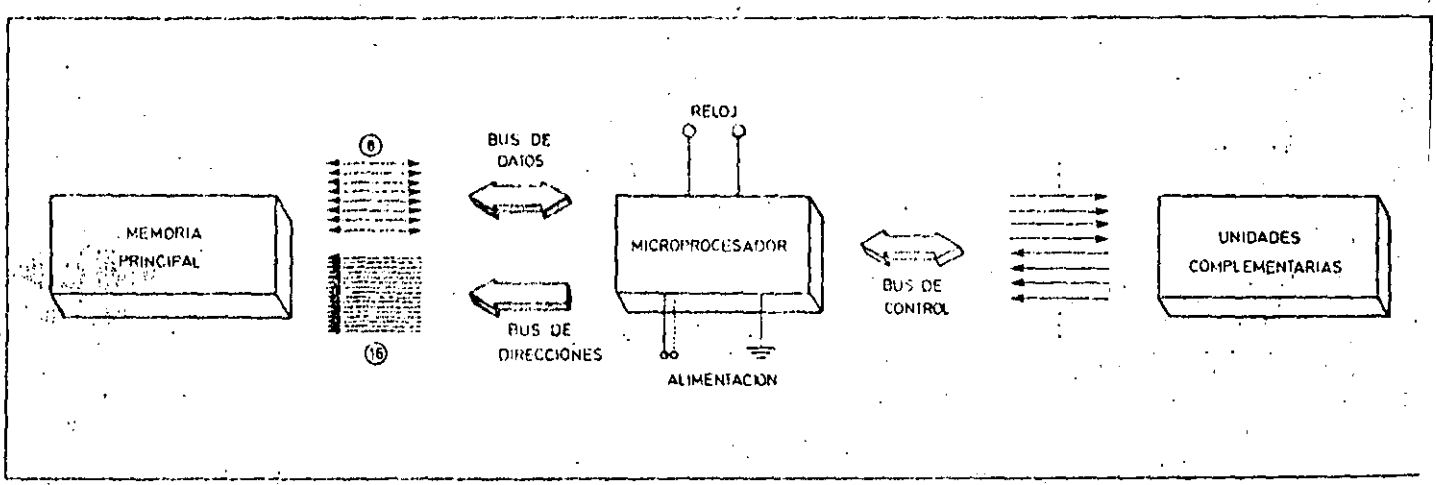
Está formado por un número variable de líneas a través de las que controla a las unidades complementarias. Evidentemente, el número de líneas necesarias depende directamente del tipo de CPU utilizada.

Además de los tres buses indicados, la CPU necesita una fuente de alimentación y un reloj para sincronizar las secuencias de operaciones. Este último entrega una señal periódica a la CPU que ésta utilizará para sincronizar todas las actividades operativas. Con todo lo expuesto anteriormente ya tenemos una idea clara de la organización exterior del microprocesador, a través de la que controla y transfiere la información. Veamos ahora cómo se organiza internamente la CPU para realizar su función.

Los dos componentes básicos que forman la CPU son la unidad de control y la unidad aritmético-lógica.



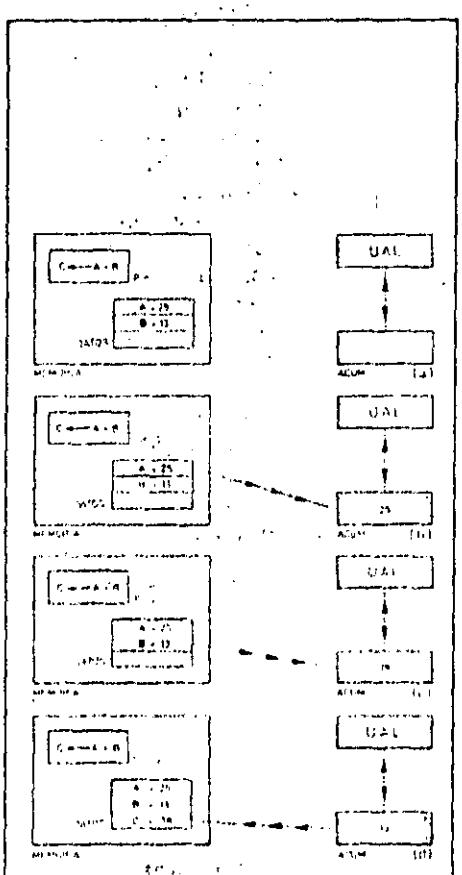
El bus de control agrupa a un conjunto variable de líneas, tanto de entrada como de salida, que permiten a la CPU gobernar a las diversas unidades del sistema conjunto.



Organización externa de un microprocesador convencional de 8 bits actuando como unidad central de proceso de un sistema microordenador.

Unidad de control

Se encarga de interpretar las instrucciones del programa y desencadenar las operaciones necesarias para su ejecución. Para ello debe controlar el funcionamiento de las unidades externas e internas implicadas.
Dispone de un registro contador de instrucciones, que apunta a la dirección de memoria en que se encuentra la instrucción a ejecutar. Evidentemente, este registro está conectado al bus de direcciones y posee tantos bits como líneas tiene el citado bus. Dado que el orden en que se deben ejecutar las instrucciones de un programa es secuencial, el contenido del contador de instrucciones se incrementará en una uni-



a) Situación inicial.
b) Carga del primer sumando en el acumulador.
c) Adición del segundo sumando al contenido del acumulador.
d) Almacenamiento del resultado.

Esquema de funcionamiento de la unidad aritmético-lógica durante el proceso de ejecución de una instrucción de suma.

Conceptos básicos

Códigos binarios

Un sistema de codificación permite transformar cadenas de caracteres basados en un alfabeto en cadenas basadas en otro distinto. Un ejemplo típico de codificación es el sistema MORSE, que transforma palabras de un alfabeto de 27 caracteres, en palabras construidas con los símbolos { . , — , b } ; en este ejemplo el beneficio obtenido es la facilidad de transmisión de los mensajes codificados.
La diferencia fundamental entre sistema de numeración y sistema de codificación, es que éste último se limita a transformar carácter a carácter mediante una tabla de conversión, mientras que los sistemas de numeración tradicionales utilizan algoritmos para las conversiones.
A continuación, vamos a estudiar dos tipos de códigos binarios decimales (BCD), que codificarán cadenas basadas en los alfabetos: { 0, 1 }, y { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }.

Código binario natural (BCD)

Es el más utilizado de los códigos ponderados (aquellos que para construir la tabla de conversiones asignan un determinado «peso» a cada posición). Está basado en la siguiente tabla:

DECIMAL	CODIGO BINARIO NATURAL
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Tal como se observa, el método consiste en sustituir cada cifra decimal por los cuatro dígitos de su representación binaria.

El sistema para la codificación se ilustra con el siguiente ejemplo:

$C_{(10} = 1609{,}28$

1 ≡ 0001
6 ≡ 0110
0 ≡ 0000 → 1 0110 0000 1001 , 0010 1
9 ≡ 1001
2 ≡ 0010
8 ≡ 1000

La conversión en sentido inverso se realizaría de forma análoga.

Código binario-decimal EXCESO-3

La propiedad fundamental de este sistema de codificación es que transforma dígitos complementarios a 9 (suman 9) en dígitos complementarios a 1.
El método para la construcción de la tabla de codificación consiste en: Dado un dígito decimal N, su codificación será la representación binaria de N + 3, así llegamos a la siguiente tabla:

DECIMAL	CODIGO BINARIO EXCESO-3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

En efecto, 3 y 6 (que son complementarios a 9) son transformados en 0110 y 1001 (que son complementarios a 1).
Veámos como ejemplo, en este caso, una decodificación de EXCESO-3 a Decimal.

$C_{(\text{EXCESO-3})} =$
$= 100\ 1001\ 0011\ 1100\ ,\ 0101\ 1011\ ;$

0100 ≡ 1
1001 ≡ 6
0011 ≡ 0
1100 ≡ 9
0101 ≡ 2
1011 ≡ 8

$\rightarrow C_{(10} = 1609{,}28$

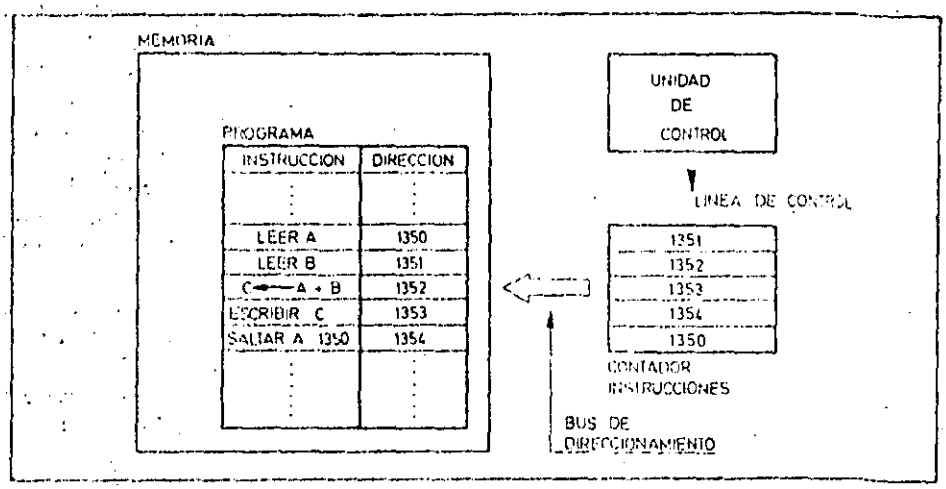
LA UNIDAD CENTRAL DE PROCESO

dad cada vez que se ejecute una instrucción, excepto cuando la propia ejecución ordene a la unidad de control una alteración en el orden de proceso. La unidad de control también dispone de un registro especial (registro de instrucciones) al que transfiere la instrucción desde la memoria con objeto de facilitar su ejecución.

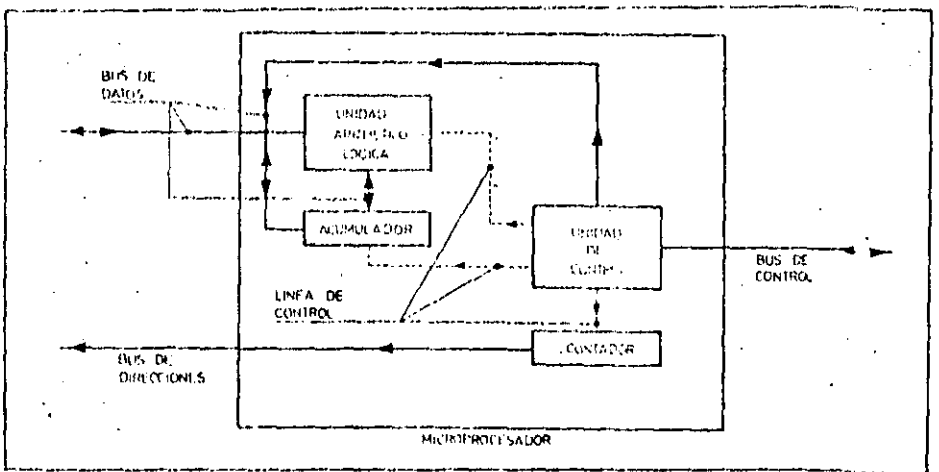
Unidad aritmético-lógica

La unidad aritmético-lógica (UAL) es la encargada de operar los datos, de forma lógica o aritmética, de acuerdo a las órdenes que recibe de la unidad de control. Podemos afirmar que la unidad de control maneja las instrucciones y la aritmético-lógica los datos. La forma de distinguir si una información es instrucción o dato, es obligando a que la primera palabra procesada sea una instrucción; a continuación, las propias instrucciones se encargan de indicar la naturaleza de la información a tratar. La unidad aritmético-lógica puede disponer de uno o dos registros especiales, denominados *acumuladores*, con los que opera bien sea directamente con su contenido o bien con su contenido más la información presente en el bus de datos.

El número de bits de estos registros coincide con el número de bits que constituyen la palabra directamente operable por la CPU (8 bits en los microprocesadores de 8 bits).



Evolución del contenido del registro contador de instrucciones durante la ejecución del programa que aparece almacenado en memoria. Salvo al ejecutar instrucciones de salto, el contenido del citado registro se incrementa en una unidad.



Arquitectura interna típica de una unidad central de proceso constituida, por ejemplo, por un chip microprocesador.

Glosario

¿Cuáles son los buses y líneas que constituyen la arquitectura externa de la unidad central de proceso?

Las líneas de alimentación y de reloj y los buses de datos, direcciones y control.

¿Qué misión cumple el reloj?

Permite a la CPU secuenciar correctamente las operaciones y actividades de proceso.

¿Qué diferencia existe entre los tres buses?

El bus de datos sirve para realizar el intercambio de instrucciones y datos; el de direcciones apunta a la posición de la memoria en la que se va a leer o escribir información, y el de control se encarga de gobernar y controlar a las unidades complementarias.

¿Cuáles son las características principales de una CPU?

El número de bits de que consta la palabra directamente operable (longitud de palabra) y la capacidad de direccionamiento de memoria.

¿Cuáles son las unidades básicas que integran la CPU?

La unidad de control que procesa las instrucciones y la unidad aritmético-lógica que se encarga de operar los datos. La primera dispone de un registro-contador que le indica la dirección de memoria en que se encuentra la instrucción y la segunda de uno o dos registros acumuladores donde realiza las operaciones.

¿A través de los buses de la CPU se puede transmitir información bit a bit?

No. Sólo tiene sentido la transmisión de palabras completas: todos los bits que conforman una palabra simultáneamente.

UNIDADES DE DISCO

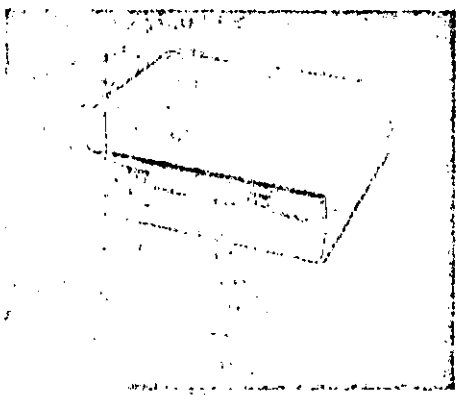
LAS unidades de disco son los periféricos de almacenamiento más utilizados en los sistemas microordenadores. Mediante este periférico los datos pueden ser almacenados y leídos cuando sea preciso. Las operaciones de lectura y escritura en el disco se realizan por medio de cabezas que en un principio eran metálicas, si bien, a partir de 1975, éstas se vieron sustituidas por cabezas cerámicas con mejor curva de respuesta y mayor duración.

• *Escritura de datos*

La escritura de los datos en el disco se realiza por medio de una cabeza que está constituida, básicamente, por una ferrita con dos bobinados. Al pasar la corriente eléctrica, en uno u otro sentido a través de los bobinados, crea un campo magnético que puede ser norte-sur o sur-norte. Este campo magnético emitido hace que las microparticulas del material magnético del disco se orienten en uno u otro sentido al pasar bajo la cabeza.

• *Lectura de datos*

El medio magnético del disco gira por debajo de la cabeza de lectura a una velocidad constante, constituyéndose en la fuente de un campo magnético variable, debido a la distinta orientación de las micropartículas. Este campo magnético se capta en el entrehierro de la cabeza, con lo que aparece una tensión inducida, de una u otra polaridad, en las bobinas. Las bobinas son complementarias, esto es: la polaridad de las tensiones inducidas es opuesta en cada una de ellas.



Las unidades de disco son los periféricos de almacenamiento más comúnmente utilizados en los sistemas microordenadores. Dentro de esta categoría, el predominio corresponde a las unidades de disco flexible.

Características constructivas

• *Simple cabeza o doble cabeza:* Las unidades de disco pueden ser de una o dos cabezas. Un disco de una sola cara puede ser escrito y leído en una unidad de dos cabezas, mientras que un disco de dos caras sólo puede operar en una unidad de doble cabeza.

En las unidades de una cabeza el patín, mandado por un electroimán, ejerce una presión sobre el disco del orden de 10 a 15 gramos, de tal forma que la cabeza entra en contacto con el disco. Esto no ocurre en las unidades de disco rígido de tecnología Winchester, en donde no existe contacto físico de la cabeza con el disco.

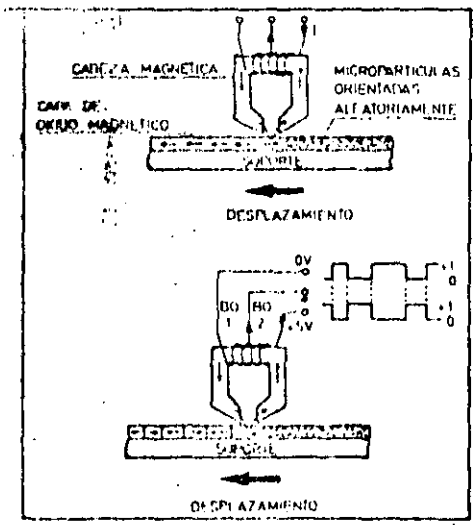
En las unidades de dos cabezas, la presión la ejercen las propias cabezas, que pueden estar enfrentadas o no. Si las cabezas no están enfrentadas se consigue reducir el desgaste del disco.

• *Movimiento del disco:* La velocidad de rotación del disco alrededor de su eje debe ser constante. Para ello las unidades de disco de 8" emplean un motor del tipo síncrono, efectuándose la transmisión por medio de una polea. La adaptación a las distintas frecuencias de la alimentación de red (50 Hz o 60 Hz) se consigue por medio del cambio de la polea.

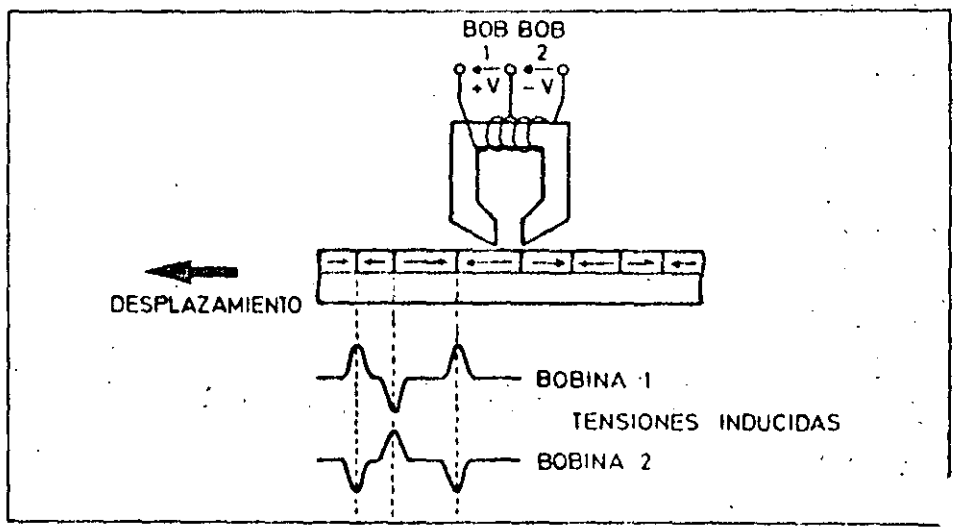
Algunos modelos de 8", al igual que los de 5 y 1/4", utilizan, en cambio, un motor de corriente continua alimentado a 12 V con control electrónico de la velocidad. Este motor tiene la ventaja de ser más pequeño e indiferente a la frecuencia de la tensión de red con la que se alimenta la unidad.

La velocidad de rotación del disco es la que da la velocidad de transferencia de los datos al ordenador. Una velocidad estándar de 300 rpm permite una velocidad de transferencia de datos de 250 Kbaudios.

• *Movimiento de las cabezas:* El movimiento de las cabezas hacia el interior y el exterior del disco se puede efectuar



Escritura de informaciones en un disco magnético. La cabeza emite un campo magnético que orienta las partículas que desfilan bajo el entrehierro. Este campo es creado en las bobinas por efecto de la circulación de la corriente.



La lectura del disco la realiza la cabeza captando en el entrehierro el campo magnético variable originado por la distinta orientación de las partículas sobre el medio magnético.

UNIDADES DE DISCO

de dos formas, dependiendo del tipo de motor:

— Motor paso a paso.
— Motor lineal.

Si el motor es paso a paso existen tres tipos de desmultiplicación de la velocidad:

a) Mediante banda flexible.
b) Mediante guía en espiral.
c) Mediante guia helicoidal.

Con motores paso a paso se consiguen tiempos de acceso de pista a pista del orden 3 a 40 msgs, con un tiempo posterior de estabilización de 8 a 45 msgs; mientras que con motores lineales el tiempo de acceso de pista a pista es del orden de 5 msgs con un tiempo de estabilización de 12 msgs.

• *Posicionamiento de las cabezas:* El posicionamiento de las cabezas en la pista correspondiente del disco se consigue de dos formas:

a) Bucle cerrado: las cabezas encuentran su posición mediante un dato de referencia almacenado en la superficie del disco.

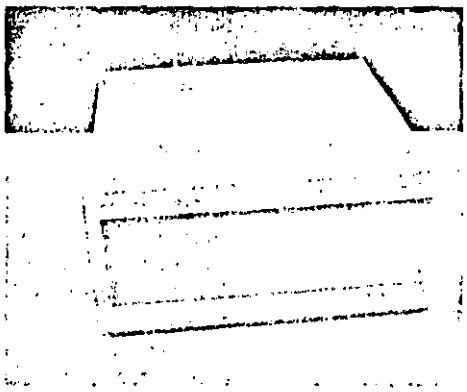
b) Bucle abierto: en este caso no hay dato de referencia en el disco. La precisión del sistema se limita a la que tenga el motor paso a paso y a la precisión de la armadura mecánica radial que puede ser de uno de los tres tipos anteriormente mencionados.

El posicionamiento por bucle abierto tiene el inconveniente de los cambios de temperatura. Si el disco es rígido, de aluminio y la armadura radial es de acero, nos encontramos con dos materiales que tienen distinto coeficiente de dilatación y, por tanto, para que no haya errores, las pistas del disco habrán de estar más alejadas una de otra, con lo que se consigue menor capacidad de almacenamiento de datos (ver tabla adjunta).

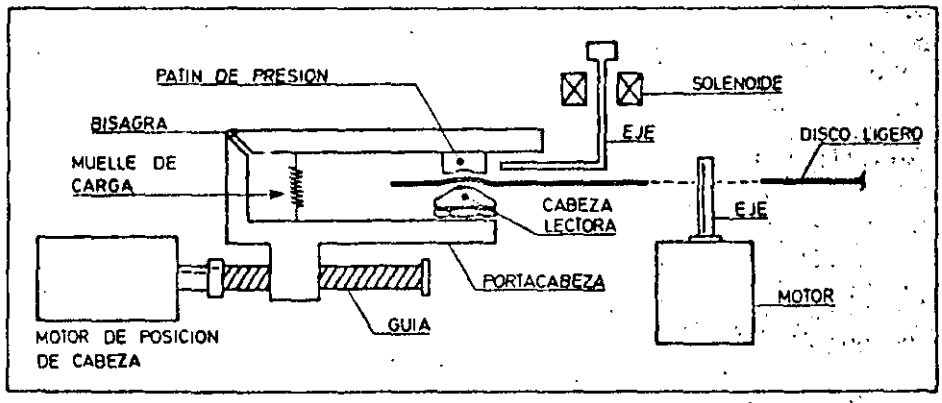
En los sistemas de bucle abierto se consiguen densidades de 200 a 300 pistas por pulgada (tpi), mientras que en los sistemas de bucle cerrado se consiguen densidades de 600 pistas por pulgada.

Características operativas

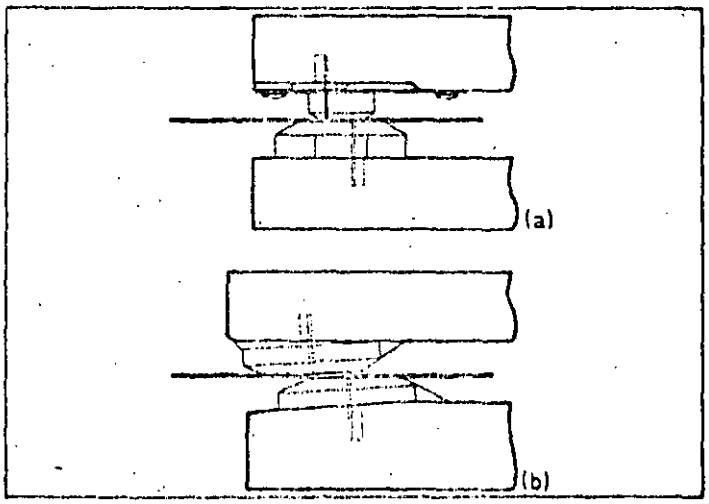
Si la unidad es de disco rígido, además



Los discos rígidos pueden ser de tipo fijo o «removible». El equipo de la fotografía es una unidad compacta de disco rígido fijo, de tecnología Winchester.



En las unidades de disco de una sola cabeza, el patín, mandado por un electroimán, ejerce una presión sobre el disco del orden de 10 a 15 gramos.



En las unidades de dos cabezas, la presión la ejercen las propias cabezas que pueden estar enfrentadas (a) o algo desplazadas entre sí (b). En este último caso se logra reducir el desgaste del disco.

Zona mecánica de una unidad para discos flexibles de 5 y 1/4 pulgadas. En la parte superior se observa la armadura metálica del patín de presión que cierra el mecanismo de sujeción del disco.

de indicarse si es de tecnología Winchester o no, debe indicarse si los discos que se emplean son fijos o removibles.

Además de las características del disco, tales como capacidad total de almacenamiento, densidad de información por pulgada, etc., la unidad de lectura/escritura tiene otras características propias:

• *Tiempo de acceso:* Normalmente se especifican dos valores, expresados ambos en msgs.

a) Tiempo de acceso pista a pista: Es el tiempo que tardan en pasar de una pista a posicionarse y empezar a adquirir datos en la pista contigua.

b) Tiempo medio de acceso: es el valor medio de los tiempos que tardan las cabezas en distintos movimientos aleatorios entre distintas pistas.

• *Velocidad de transferencia de datos:* Es la velocidad a la que se comunican los datos al ordenador una vez que las cabezas están posicionadas en la pista. Se expresa en baudios o en Kbaudios. Depende lógicamente de la velocidad de rotación del disco.

• *Método de grabación:* Debido a que actualmente las unidades de disco están controladas electrónicamente por medio de un microprocesador, se incorporan en la propia unidad los sistemas de codificación y decodificación:

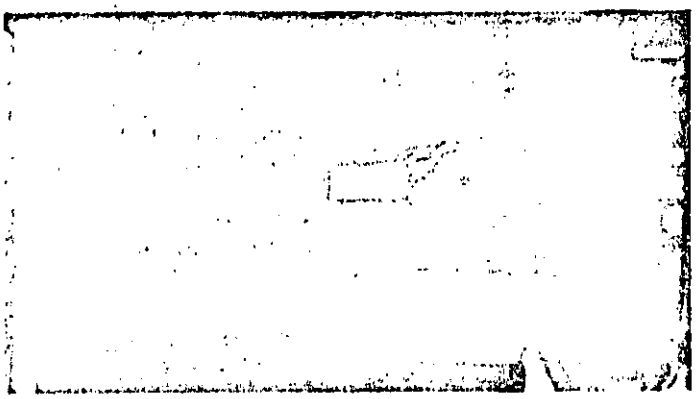
a) FM.
b) MFM.
c) M²FM.

pudiendo la unidad, por tanto, operar con discos de simple densidad o bien de doble densidad.

• *Tipo de interface:* Existen tres tipos de interface normales en las unidades de disco:

a) Niveles TTL.
b) RS232.
c) Bus IEEE 488.

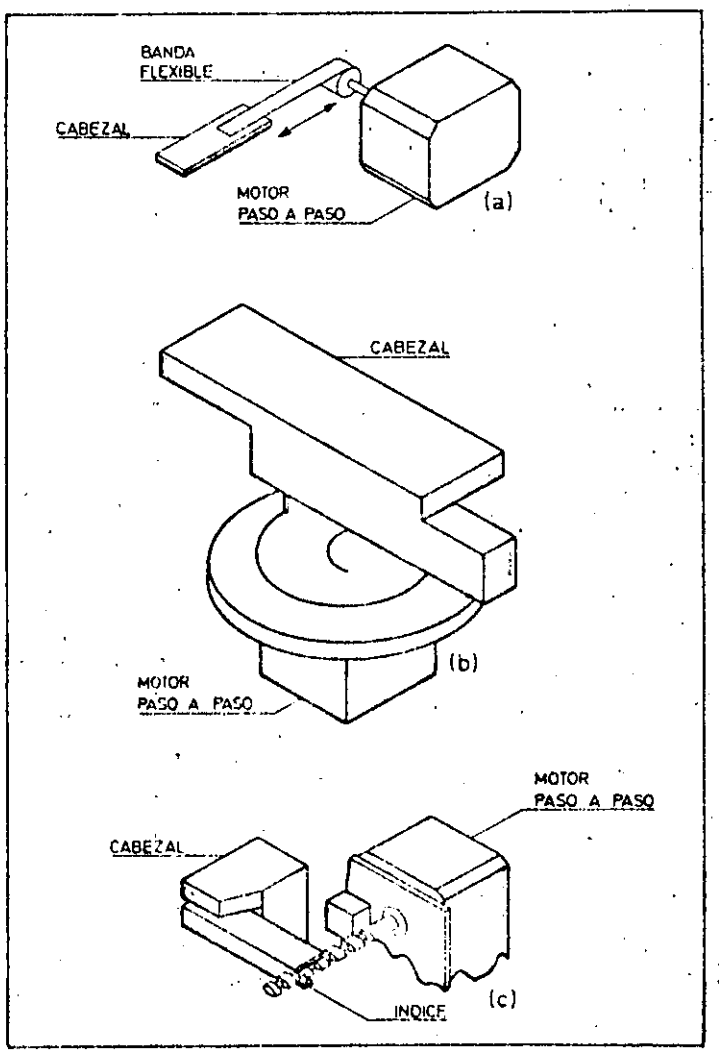
• *Dimensiones:* Las unidades de disco rígido pueden ser para discos de 14", 8" ó 5 1/4". Las unidades de disco flexible pueden ser para discos de 8", 5 1/4" o microfloppies (menores de 4").



Detalle del soporte de la cabeza de lectura y escritura de la unidad de disco de la fotografía anterior.

CAPACIDADES DE ALMACENAMIENTO SEGUN EL POSICIONAMIENTO DE LAS CABEZAS (Con tiempos de acceso medios)

DISCOS	BUCLE ABIERTO Motor paso a paso	BUCLE CERRADO Motor lineal
5 1/4"	Hasta 20 Mbytes 85 a 200 mseg	20 M a 50 Mbytes 35 a 50 mseg
8"	10 M a 40 Mbytes 70 a 80 mseg	30 M a 100 Mbytes 30 a 50 mseg
14"	30 M a 40 Mbytes 70 a 80 mseg	30 M a 300 Mbytes 30 a 50 mseg



Los motores paso a paso admiten tres tipos o variantes para la desmultiplicación de velocidad: a) mediante banda flexible, b) mediante guía en espiral y c) por medio de guía helicoidal.

LA MEMORIA PRINCIPAL DE LOS MICROORDENADORES

La memoria principal de los microordenadores está dividida en dos unidades de almacenamiento de información: *la memoria RAM* y *la memoria ROM*.

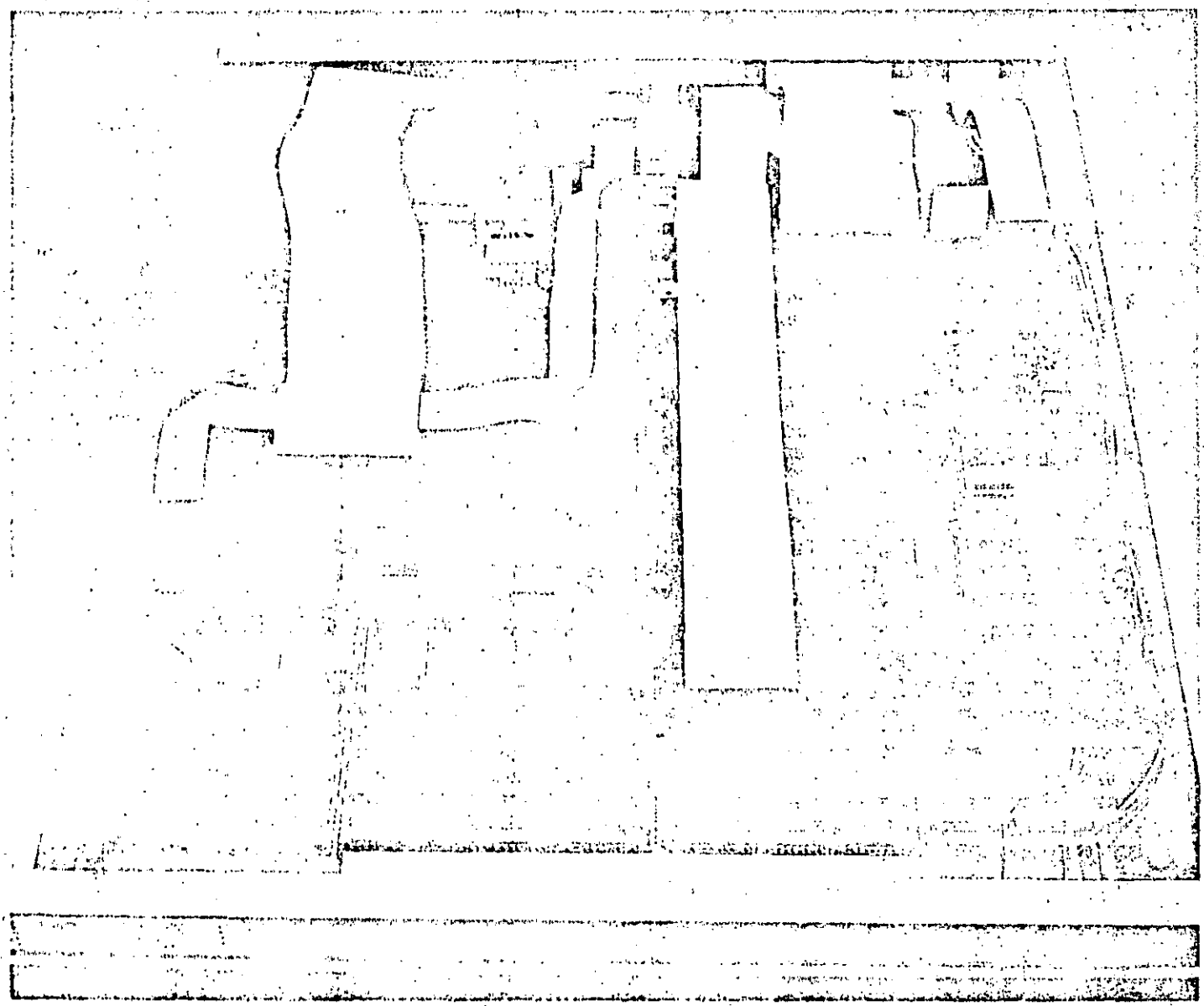
El concepto de memoria se aplica a todo dispositivo electrónico que pueda almacenar información. De esta forma, consideramos como memoria de un ordenador tanto a la memoria central utilizada por la CPU para la ejecución de programas, como a la auxiliar que servirá para almacenar información de forma masiva. La diferencia principal entre estas dos clases de memorias es que la auxiliar no puede ser procesada directamente por la CPU; por ejemplo, si disponemos de un programa almacenado en un diskette (memoria auxiliar), y queremos ejecutarlo, es imprescindible cargarlo previamente en la memoria principal.

En este capítulo nos ocuparemos del estudio de la memoria principal, tanto de la zona RAM como ROM, concretando los ejemplos al caso de los sistemas microordenadores.

Memoria RAM

La memoria RAM (Random Acces Memory) se suele denominar también memoria de lectura/escritura (R/W, Read/Write), ya que en ella se puede leer o escribir información, indistintamente.

Los medios de comunicación de la memoria con la CPU (por ejemplo, microprocesador) son el *bus de direcciones*, mediante el cual se apunta a la dirección de memoria que ocupa o va a ocupar la información, y el bus de da-



La memoria principal de los sistemas microordenadores consta de dos zonas de almacenamiento, memoria ROM y memoria RAM.

LA MEMORIA PRINCIPAL DE LOS MICROORDENADORES

Conceptos básicos

Aritmética binaria

En este apartado vamos a describir las cuatro operaciones aritméticas básicas: adición, sustracción, multiplicación y división, con el sistema de numeración binario. De esta forma obtendremos una idea de cómo funciona la unidad aritmético-lógica de la CPU en su componente aritmética.

Adición binaria

La suma binaria se realiza dígito a dígito, según la siguiente tabla:

PRIMER SUMANDO	SEGUNDO SUMANDO	RESULTADO	ACARREO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Al igual que en la suma decimal «7 + 8 = 5 más un dígito de acarreo para la cifra situada a la izquierda», en la operación binaria equivalente se verifica que «1 + 1 = 0 y acarreamos 1 para la cifra siguiente»; el fundamento de este resultado es que 1 + 1 = 2 que en binario es 10.

1	1	0	1	1	0	ACARREO
	1	1	0	1	1	0 SUMANDO-1
	1	1	0	1	1	1 SUMANDO-2
1	1	0	1	1	0	1 RESULTADO

Luego, 110110 + 110111 = 1101101.

Sustracción binaria

El procedimiento es semejante al utilizado en aritmética decimal, si bien la resta binaria se ajusta a la siguiente tabla:

MINUENDO	SUSTRAENDO	RESULTADO	PENDIENTE
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Las cantidades pendientes se suman a la siguiente cifra del sustraendo. Por ejemplo:

1	0	1	1	0	1	MINUENDO
0	1	1	0	1	1	SUSTRAENDO
1	0	0	1	0		PENDIENTE
0	1	0	0	1	0	RESULTADO

Con lo que: 101101 - 011011 = 10010

Multiplicación binaria

La tabla de multiplicar para un número bit en la siguiente:

MULTIPLICANDO	MULTIPLICADOR	RESULTADO
0	0	0
0	1	0
1	0	0
1	1	1

El procedimiento para multiplicar números binarios de más de un bit es similar al utilizado comúnmente para la multiplicación de números decimales:

Paso 1: Efectuar el producto de la cifra menos significativa del multiplicador por todas las del multiplicando.

Paso 2: Repetir el Paso 1 para todas las restantes cifras del multiplicador, escribiendo el resultado parcial, desplazado un lugar hacia la izquierda, debajo del resultado anterior. Si el bit tratado del multiplicador es 0, omitir la operación y desplazar una posición adicional el próximo resultado parcial.

Paso 3: Cuando no queden más dígitos del multiplicador, efectuar la suma de todos los resultados.

```
                1 0 1 1 0 1
              x       1 0 1 1
          ---------------------
                1 0 1 1 0 1
              1 0 1 1 0 1
            0 0 0 0 0 0
          1 0 1 1 0 1
          ---------------------
          1 1 1 1 0 1 1 1 1
```

División binaria

Se realiza aplicando un procedimiento análogo a la división decimal.

Paso 1: Se toman, de izquierda a derecha, los dígitos necesarios del dividendo para formar un número mayor o igual que el divisor, a este número le llamaremos número intermedio, y se coloca un + 1 en el resultado.

Paso 2: Se resta el divisor del número intermedio y se añade al resultado la siguiente cifra del dividendo.

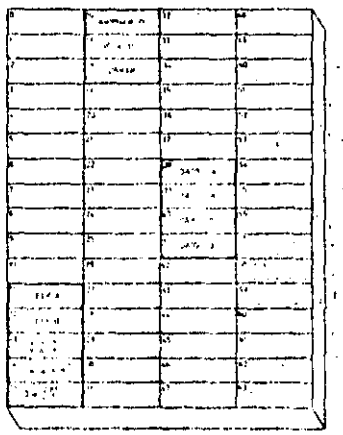
Paso 3: Si el número obtenido en el Paso 2 es mayor que el divisor, se coloca un «1» en el resultado y se repite el Paso 2, en caso contrario se coloca «0» en el resultado, se añade una nueva cifra del dividendo y se repite el Paso 2. En ambos casos, el nuevo número intermedio es el resultado de la última resta y las cifras añadidas.

Paso 4: Cuando se haya tomado el último dígito del dividendo se da por finalizada la operación, siendo el resto el número intermedio si éste es inferior al divisor, si no lo es se repite un paso más el proceso.

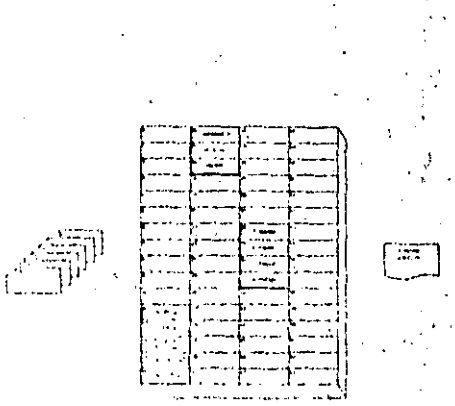
```
  1 0 1 0 0 0 | 1 1 0
- 0 1 1 0     ---------
    1 0 0 0     1 1 0
  - 0 1 1 0
      0 1 0 0
    - 0 1 1 0
        0 1 0
```

tos, a través del que se transmite la instrucción o dato apuntado por el bus de direcciones. Esta transmisión puede efectuarse en los dos sentidos, es decir, desde el exterior hacia la memoria o desde la memoria hacia el exterior.

La forma de determinar si la información va a ser leída o escrita en memoria es a través de una señal de control con dos estados posibles: uno implica lectura y otro escritura. En ambos casos la operación se realizará a través del bus de datos. Otra señal de control autorizará o no la utilización de la memoria. Todas las operaciones con la memoria



Representación de la memoria principal de un ordenador en la que está almacenado un programa para el cálculo de la hipotenusa de un triángulo rectángulo en función de los catetos.

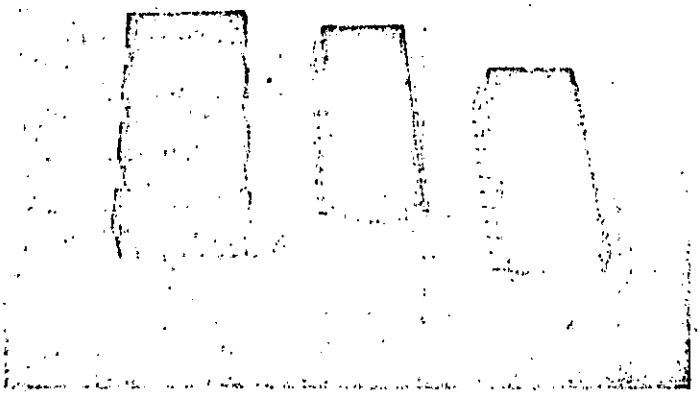


Después de ejecutar el programa anterior con los datos reflejados en las fichas (a), la memoria queda tal como se indica en el gráfico (b). La zona (c) muestra el correspondiente listado final.

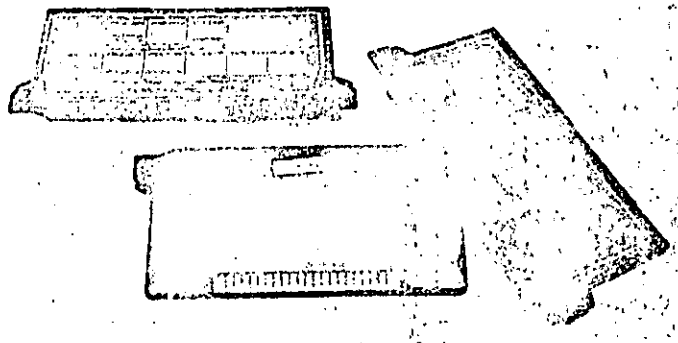
están controladas por la unidad de control integrada en la CPU (microprocesador).
Dentro del ordenador o microordenador, la memoria RAM se utilizará tanto para almacenar programas y datos como para guardar resultados intermedios.
Otra característica de la memoria RAM es su volatilidad; la falta de alimentación eléctrica hace desaparecer toda la información que estuviera almacenada en ella. Esto no debe suponer un grave problema, puesto que el usuario debe tener almacenados sus programas y datos en una memoria auxiliar no volátil (cinta magnética, diskette, etc.), de forma que el único riesgo ante una falta de energía se reduzca a la pérdida de las modificaciones efectuadas durante la sesión en curso.

Memoria ROM.

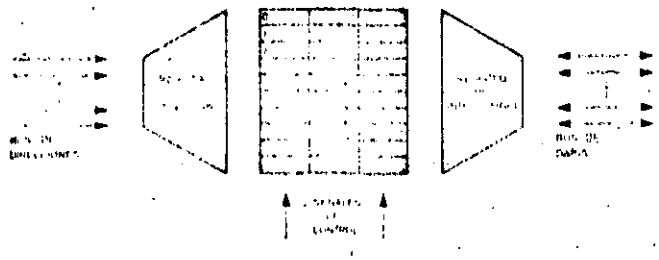
La memoria ROM (Read Only Memory) sólo permite la operación de lectura, de forma que los programas grabados en ella por el fabricante pueden ser utilizados, pero nunca modificados.
La comunicación con el microprocesador se efectúa, al igual que en las memorias de tipo RAM, a través de los buses de direcciones y datos. No obstante, en este caso, el bus de datos sólo permite la salida de información desde la memoria hacia el exterior y no al revés.
La señal de control sólo interviene para autorizar la utilización de la memoria ROM.
Evidentemente, las memorias de este tipo no son volátiles dado que su contenido es fijo y no puede reprogramarse. Por lo demás, si se perdiera la



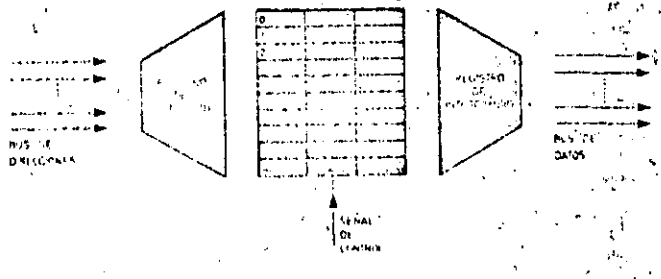
La zona de memoria de los modernos sistemas microordenadores está constituida a base de la asociación de circuitos integrados de memoria cuya capacidad por chip puede variar desde algunos bytes hasta varias decenas de Kbytes.



Algunos microordenadores permiten la ampliación de la memoria principal por medio de la incorporación de módulos exteriores de memoria RAM o ROM.



Estructura típica de una unidad de memoria RAM. Obsérvese que en este tipo de memorias (de lectura y escritura) los datos pueden entrar o salir de la unidad a través del correspondiente bus de datos.



Organización de una unidad de memoria ROM. Dada su característica de memoria de sólo lectura, los datos se canalizan sólo en un sentido, del interior de la memoria hacia el exterior a través del bus de datos.

LA MEMORIA PRINCIPAL DE LOS MICROORDENADORES

información contenida en ellas, quedarían inutilizables.

La memoria principal de un microordenador puede ampliarse, incrementando el número de unidades conectadas. La única limitación consiste en la capacidad de direccionamiento del bus de direcciones. Normalmente, los microprocesadores de 8 bits (CPU de los microordenadores de 8 bits) suelen disponer de 16 líneas en dicho bus, con lo que pueden llegar hasta $2^{16} = 65.536$ direcciones distintas. Como cada dirección está ocupada por una palabra de 8 bits (byte), el tope máximo de ampliación de este tipo de microordenadores es de 64 Kbytes (64 × 1.024 = 65.536).

Otro indicador importante para caracterizar una memoria es el *caudal*, así se denomina al número máximo de informaciones leídas o escritas en la memoria por unidad de tiempo. La unidad más usual para medir el caudal es el Kilobyte por segundo o el Megabyte por segundo. Por ejemplo, podemos hablar de un caudal de 15 Kilobytes por segundo, si el bus de datos puede transmitir 15 × 1.024 = 15.360 bytes por segundo.

BREVE DESCRIPCION DE LA EJECUCION DEL PROGRAMA PARA EL CALCULO DE HIPOTENUSAS

INSTRUCCION	EJECUCION
11. LEER A	La unidad de control acepta por el dispositivo de entrada un dato que se almacena en la palabra 38 de la memoria (a la que simbólicamente se la llama A).
12. LEER B	Análogo.
13. SI A = 0 IR a 18	La unidad aritmético-lógica ejecuta la instrucción. Comprueba si el valor cargado en la palabra 38 (A) es igual a 0. En caso afirmativo, se altera la secuencia y se ejecuta la instrucción 18; en caso negativo, se continúa en secuencia.
14. C ← $A^2 + B^2$	La unidad aritmético-lógica calcula la expresión ($A^2 + B^2$) y carga el resultado en la palabra 40 (C). Esta variable es utilizada como valor intermedio.
15. D ← $\sqrt{C}$	Se calcula la raíz cuadrada de la palabra 40 (C) y se almacena el resultado en la palabra 41 (D).
16. IMPRIMIR D	La unidad de control imprime por un dispositivo de salida el valor contenido en la palabra 42 (D).
17. IR a 11	Mediante una alteración en el registro contador de instrucciones de la unidad de control se produce una variación en la secuencia de ejecución, de forma que la siguiente instrucción será la contenida en la dirección 11.
18. PARAR	La unidad de control da por terminada la ejecución del programa, con lo que la memoria utilizada queda libre.

Glosario

¿Se puede ejecutar directamente un programa almacenado en una cinta magnética?

No. La intervención de la memoria principal es esencial para la ejecución de un programa, por tanto, habrá que pasarlo previamente de la memoria auxiliar (cinta magnética) a la memoria principal para que sea posible su ejecución.

¿Qué tipo de información se almacena en la memoria RAM?

Tantos datos como instrucciones, ya que este tipo de memoria está reservada para el usuario que puede escribir y leer en la misma a través de los programas que construya.

¿Se puede utilizar la memoria ROM para almacenar datos?

No. Unicamente puede contener los programas cargados por el fabricante (sistema operativo, programas de utilidad, etc.) y algunos datos fijos. El usuario sólo puede utilizar dichos programas (leerlos), pero nunca modificarlos.

¿En qué consiste la volatilidad de la memoria?

En la pérdida de su contenido ante la falta de fluido eléctrico. Las memorias RAM son volátiles, pero la ROM y la mayoría de las memorias auxiliares no lo son.

¿Se puede ampliar la memoria principal de un microordenador?

Sí. Siempre que el bus de direcciones pueda apuntar hasta la última palabra de la unidad de memoria utilizada en la ampliación.

¿A qué se denomina caudal?

A la cantidad de información por unidad de tiempo que se puede transmitir entre la memoria y la CPU. Las unidades utilizadas para representar el caudal son el Kilobyte por segundo y el Megabyte por segundo.

UNIDADES DE ENTRADA/SALIDA

La función de las unidades de entrada/salida —E/S o I/O, según utilicemos notación castellana o anglosajona INPUT/OUTPUT— es adaptar la información procedente del exterior para que sea interpretable por el ordenador, así como adaptar la información suministrada por el ordenador para que pueda ser tratada por los periféricos.

Intercambio de información con el exterior

Las unidades periféricas no forman parte de la unidad central de proceso, de ahí que, necesariamente, haya que habilitar el intercambio de información entre la CPU y los periféricos.

Si comparamos al ordenador con el cuerpo humano, la CPU es el cerebro, mientras que las unidades periféricas más importantes serían los órganos en los que residen los cinco sentidos. De poco serviría tener un cerebro privilegiado, si la información obtenida a través de cualquier sentido (la vista, por ejemplo), no fuera interpretable por aquél.

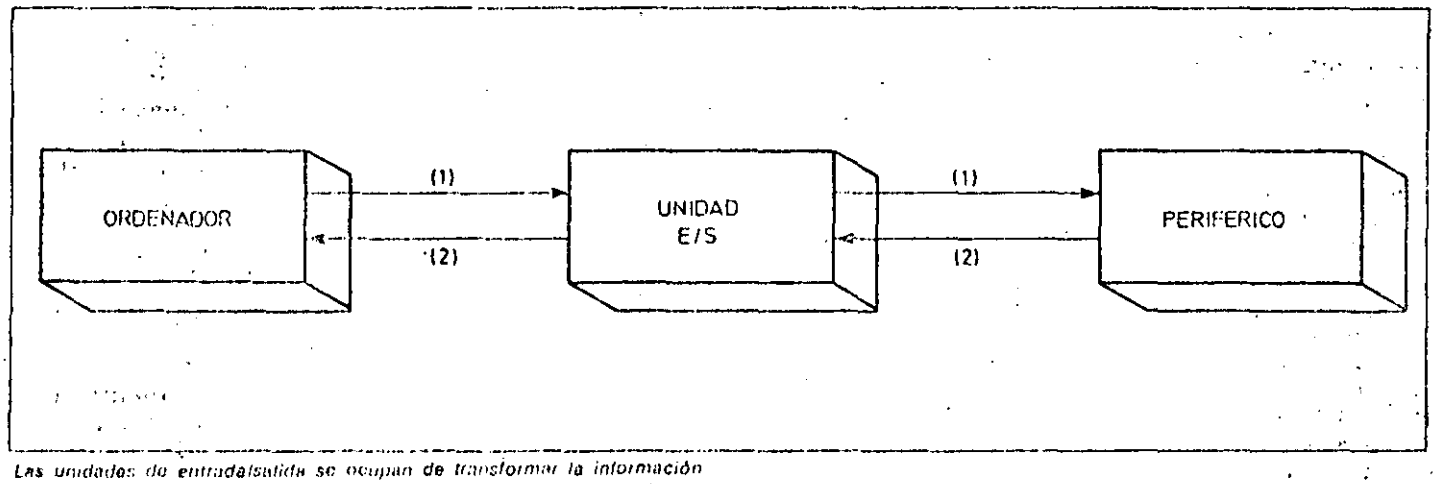
Este papel está reservado al sistema nervioso en el cuerpo humano y a las unidades de E/S en el ordenador.

Justificación de las unidades de entrada/salida

Estas unidades sirven para canalizar las transferencias de información entre el ordenador, mini o micro, y los dispositivos periféricos exteriores. Por supuesto, su actuación es controlada por la CPU.

Un ordenador puede disponer de varias unidades de E/S que, a su vez, pueden controlar varios periféricos del mismo tipo. Las principales ventajas obtenidas con su empleo son las siguientes:

1. La velocidad de trabajo de la CPU es muy superior a la de los periféricos.



Las unidades de entrada/salida se ocupan de transformar la información representada en formato del ordenador a información interpretable por el dispositivo periférico (1) y viceversa (2).

En los microordenadores actuales, las unidades de entrada/salida suelen estar constituidas por circuitos integrados programables especializados en este tipo de tareas.

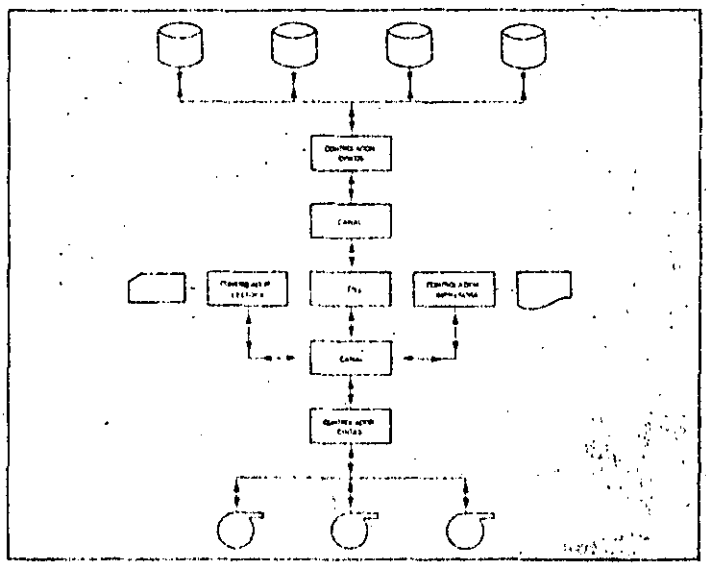


Diagrama de la arquitectura de la unidad central de un ordenador en el que intervienen los dos tipos genéricos de unidades de E/S, canales y controladores de periféricos.

UNIDADES DE ENTRADA/SALIDA

Mediante las unidades de E/S se consigue la independencia entre ambas.

2. Las unidades periféricas pueden tener distinta forma de tratar la información, incluso dentro de unidades del mismo tipo esta característica varía según los fabricantes. Mediante las unidades de E/S se pueden adaptar muy diversos tipos de periféricos, independientemente de que sus formatos sean distintos a los del ordenador.

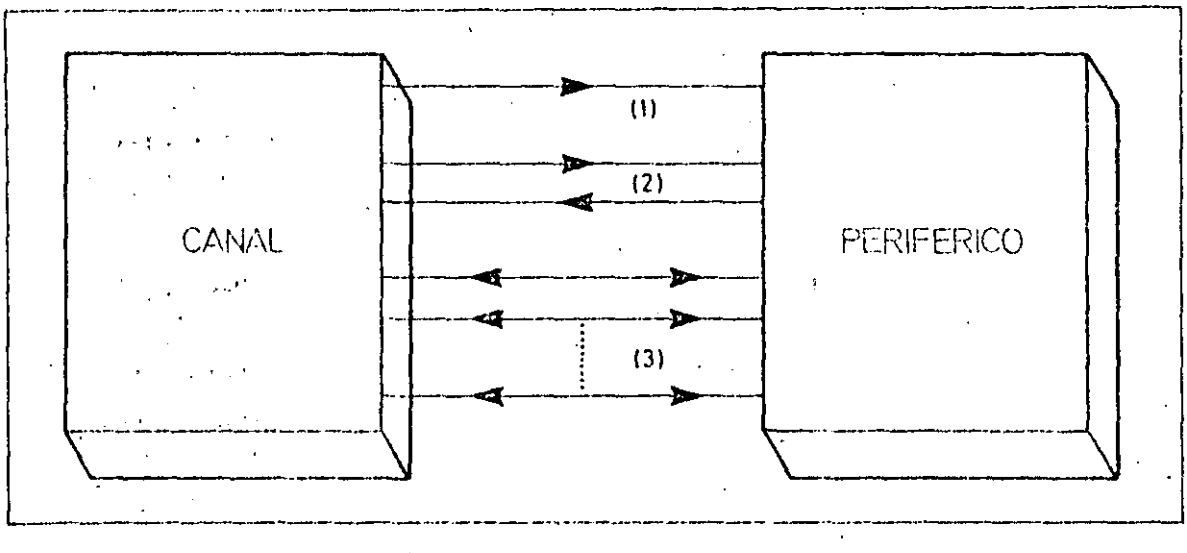
3. Las unidades de E/S también sirven de intermediarias entre las lógicas binarias del ordenador y de los periféricos, que pueden ser distintas.

Tipos de unidades de E/S

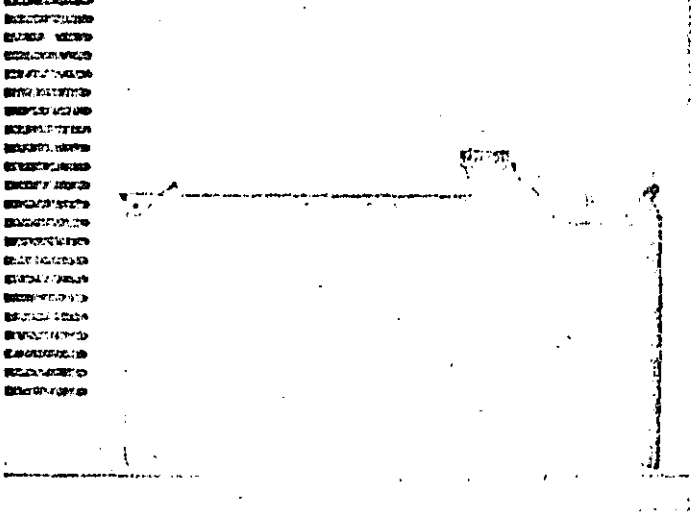
El intercambio de información entre la memoria del ordenador y el exterior se realiza, normalmente, a través de dos tipos de unidades de E/S: los canales y los controladores de periféricos.

Canales

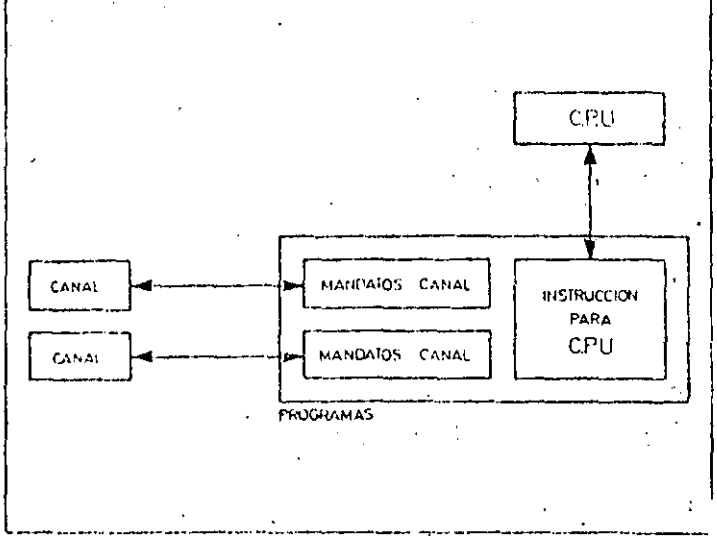
El número de operaciones por segundo que puede ejecutar cualquier ordenador es muy superior al de transferencias de información por segundo, esto obligaba a mantener bloqueada a la CPU mientras se realizaban las operaciones de entrada/salida. La solución a este problema fueron los *canales*. Cuando la CPU necesita realizar una transferencia de información con un periférico lento, por ejemplo, una impresora, no tiene por qué esperar a que ésta termine de escribir una línea para ordenar la escritura de la siguiente. Simplemente, «lanza» todas las órdenes de escritura al canal y continúa ejecutando otras instrucciones del programa o incluso dándolo por acabado. Será responsabilidad del canal gestionar adecuadamente todas las operaciones de salida que le han sido transferidas.



Esquema básico de la estructura de un bus de adaptación (interface) genérico: (1) señal de lectura/escritura; (2) líneas para la sincronización de operaciones de transferencia; (3) líneas de transferencia de información.



Algunas unidades de entrada/salida pueden requerir una circuitería electrónica compleja. En la fotografía aparece una unidad de E/S encargada de la adaptación de un puesto de trabajo al sistema Secoinsa Serie-20.



Los canales liberan a la CPU de tareas supletorias, mientras la CPU ejecuta instrucciones, los canales pueden ocuparse de gestionar las operaciones de transferencia con los periféricos.

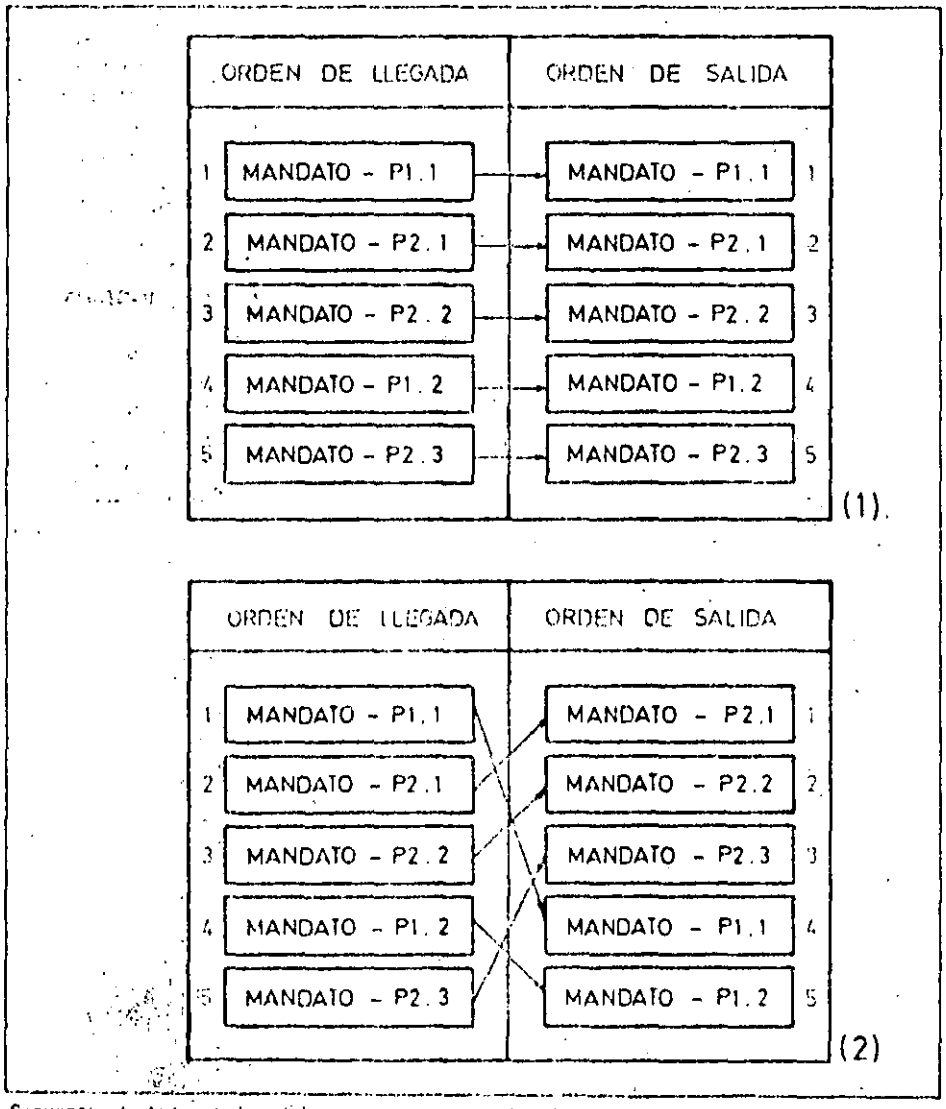
Este tipo de dispositivos sólo se utilizan en ordenadores y minordenadores. La mayoría de los microordenadores no disponen de canales, realizándose el intercambio de información directamente entre la unidad de control de la CPU y los controladores de periféricos.

Controladores de periféricos

Se encargan de gestionar una o varias unidades periféricas de un mismo tipo; para ello tienen que ser capaces de:

- Interpretar las instrucciones que reciben o entregan del/al ordenador. Esto se realiza a través de circuitos que adaptan y reconocen las señales de «interface» del canal o de la unidad central de proceso.
- Controlar al periférico asociado según sus características; para ello decodifican la operación que se les ordena ejecutar (lectura, escritura, rebobinado...) y, en el sentido inverso, emiten información del estado del periférico (ocupado, preparado, rebobinado...).

La complejidad de los controladores de periféricos puede ser muy distinta y suele estar en consonancia con la complejidad del propio periférico. Evidentemente, es muy distinto controlar una lectora de tarjetas, que realiza necesariamente un trabajo secuencial y



Secuencia de órdenes de salida, para un mismo orden de llegada, según se aplique la alternativa de prioridad «primero en llegar, primero en salir» (1) o «prioridad exterior» (2), en este último caso, se considera prioritario al procesador P2.

Glosario

¿Cuáles son las principales ventajas introducidas por las unidades de E/S?

1. Desbloqueo de la CPU al encargarse de forma autónoma de gestionar las transacciones de información.
2. Adaptación de los distintos formatos con que trabajan los periféricos a los formatos del ordenador.
3. Control y adaptación de las distintas lógicas binarias de los periféricos y la CPU.

¿Qué tipos de unidades de E/S existen?

Los *canales* que son los intermediarios entre la unidad central de proceso del ordenador y los periféricos, y los *controladores de periféricos* que pueden gestionar la utilización de uno o varios periféricos del mismo tipo.

¿Los controladores de periféricos son parte de los periféricos?

En algunos casos, cuando el controlador se encarga de un único periférico, se le puede considerar como parte del periférico, aunque, en general, el controlador es independiente del periférico.

¿Qué misiones específicas tienen encomendadas los controladores de periféricos?

Depende mucho del tipo de unidades controladas, pero, en general, se encargan de ejecutar las instrucciones recibidas de la CPU y de devolver a ésta información del estado del periférico.

¿En qué consiste la «interface»?

Son las especificaciones necesarias para poder conectar los periféricos a las unidades de control o canales. Por extensión, suele denominarse de esta forma al dispositivo o unidad que se ocupa de adaptar y establecer este tipo de comunicaciones.

UNIDADES DE ENTRADA/SALIDA

exclusivamente de entrada de datos, que controlar una unidad de discos que puede ser utilizada tanto para entrada como para salida de datos e, incluso, de forma no secuencial. En los casos más complejos se puede llegar a utilizar microprocesadores para realizar la labor de controlar los periféricos.

«Interface» entre la CPU y las unidades periféricas

Se denomina «Interface» a las especificaciones de conexión necesarias para adaptar las unidades periféricas a la CPU o a los canales. En un sentido más amplio, también se entiende por «Interface» a todos los componentes necesarios para adaptar las señales del ordenador a las de los periféricos y viceversa. La «Interface» está compuesta fundamentalmente por dos tipos de hilos: los que llevan la información a transcribir y los que se encargan de sincronizar las operaciones e indicar si éstas son de entrada o salida.

Prioridades de acceso

En el caso de que dos o más procesadores soliciten al mismo tiempo un periférico, la unidad de E/S tiene varias alternativas para decidir a cuál de ellos atenderá primero:

1. *Primero en llegar, primero en salir*

Este método es el más sencillo, consiste en asignar los recursos según el orden en que han sido solicitados. La unidad de E/S mantiene una cola de transacciones a efectuar en el orden de llegada y las va realizando en el mismo orden.

2. *Prioridad exterior*

El sistema acepta unas prioridades marcadas desde el exterior, de forma que atenderá las solicitudes antes o después según tengan más o menos prioridad, respectivamente. El defecto de este procedimiento es que un procesador con máxima prioridad, que tenga muchas transacciones, puede llegar a bloquear el sistema.

3. *Asignación cíclica*

Consistente en atender, cíclicamente y siempre en el mismo orden, a todos los procesadores que hayan solicitado la intervención de un periférico. Este método no es posible utilizarlo más que en determinado tipo de sistemas.

Conceptos básicos

Representación de números en coma fija

Este tipo de representación sólo se utiliza para números enteros, tanto positivos como negativos. En el caso de que fuera necesario trabajar con números reales con esta representación, el programa sería el responsable de controlar los decimales. Existen, básicamente, tres formas distintas de representar un número en sistema binario con coma fija:

1. *Representación en verdadera magnitud y signo*

Este método consiste en codificar en el primer bit de la palabra el signo del número mediante un «1», en el caso de números negativos, o un «0», para los positivos, y representar con los restantes bits el valor absoluto en sistema binario. Ejemplo:

NUMERO DECIMAL	REPRESENTACION (Verdadera magnitud y signo)
4	0 0 0 0 0 1 0 0
– 4	1 0 0 0 0 1 0 0
25	0 0 0 1 1 0 0 1
– 25	1 0 0 1 1 0 0 1

2. *Representación en complemento a 1*

El bit destinado al signo, al igual que en el caso anterior, valdrá «0» ó «1», según estemos representando un número positivo o negativo, respectivamente; en el resto de los bits de la palabra se codificará la verdadera magnitud del número en binario si éste es positivo y en complemento a uno, es decir, cambiando los unos por ceros y viceversa, si es negativo. Ejemplo:

NUMERO DECIMAL	REPRESENTACION (En complemento a 1)
4	0 0 0 0 0 1 0 0
– 4	1 1 1 1 1 0 1 1
25	0 0 0 1 1 0 0 1
– 25	1 1 1 0 0 1 1 0

3. *Representación en complemento a 2*

Para representar un número en complemento a 2, hasta con codificarlo en verdadera magnitud y signo si es positivo, y seguir los siguientes pasos si es negativo:

Paso 1: Representar el número en verdadera magnitud.

Paso 2: Obtener su complemento a 1.

Paso 3. Sumarle una unidad a la configuración resultante. Ejemplo:

NUMERO DECIMAL	REPRESENTACION (En complemento a 2)
4	0 0 0 0 0 1 0 0
– 4	1 1 1 1 1 1 0 0
25	0 0 0 1 1 0 0 1
– 25	1 1 1 0 0 1 1 1

Justificación de las representaciones en complemento

El hecho de haber elegido dos sistemas aparentemente tan extraños para la representación de números, como son los complementos a 1 y 2 se debe a que, ambos casos, mediante un único circuito se realizan las operaciones de suma y resta, mientras que con la representación en verdadera magnitud y signo serían necesarios dos circuitos distintos. Veámoslo con un ejemplo:

- En complemento a 1:

4 – 25 = 0 0 0 0 0 1 0 0 +
+ 1 1 1 0 0 1 1 0 =
= 1 1 1 0 1 0 1 0 = – 21

4 + 25 = 0 0 0 0 0 1 0 0 +
+ 0 0 0 1 1 0 0 1 =
= 0 0 0 1 1 1 0 1 = 29

Tanto en el caso de la suma como en la resta hemos utilizado la operación (circuito) suma binaria.

- En complemento a 2:

4 – 25 = 0 0 0 0 0 1 0 0 +
+ 1 1 1 0 0 1 1 1 =
= 1 1 1 0 1 0 1 1 = – 21

4 + 25 = 0 0 0 0 0 1 0 0 +
+ 0 0 0 1 1 0 0 1 =
= 0 0 0 1 1 1 0 1 = 29

Igual que en el caso anterior, con un único circuito «suma» podemos sumar y restar.

- En verdadera magnitud y signo:

4 – 25 = 0 0 0 0 0 1 0 0 –
– 1 0 0 1 1 0 0 1 =
= 1 0 0 1 0 1 0 1 = – 21

4 + 25 = 0 0 0 0 0 1 0 0 +
+ 0 0 0 1 1 0 0 1 =
= 0 0 0 1 1 1 0 1 = 29

En este caso, para efectuar las operaciones restar y sumar hemos necesitado dos circuitos totalmente distintos, basados en la resta y suma binaria, respectivamente.

PLOTTERS

Los plotters son periféricos de salida que efectúan dibujos de trazo continuo al recibir las instrucciones correspondientes de un ordenador, o dicho de otro modo, a partir de un programa un plotter puede realizar los planos que corresponden a un diseño. Su aplicación principal es en oficinas de ingeniería como elemento final de salida «hard copy» (copia impresa) de los sistemas CAD (Computer Aided Design: diseño ayudado por ordenador) o CAM (Computer Aided Manufacturing: fabricación ayudada por ordenador). Si, por ejemplo, se quiere realizar el diseño de una estructura, el sistema deberá disponer de un teclado y de una pantalla de rayos catódicos con posibilidad de gráficos; mediante el teclado se realizan los cálculos correspondientes, así como las diferentes correcciones en el dibujo de la estructura que aparece en la pantalla. Una vez que ya se tiene en la pantalla el dibujo final corregido, se pasa al papel dibujándolo mediante el plotter. De igual manera se puede utilizar en electrónica para diseño de circuitos impresos: el plotter dibuja el plano del circuito impreso, el plano de montaje de los componentes y toda la información necesaria para la realización práctica del diseño electrónico.

Funcionamiento

Por la forma de realizar el dibujo los plotters se pueden dividir en dos tipos:

- *De plumas:* Los dibujos se efectúan mediante plumas con tinta que se aplica sobre un papel normal.
- *Electrostáticos:* La pluma se reemplaza por una punta catódica y se dibuja sobre un papel electrosensitivo. Son más rápidos, pero de menor precisión que los de plumas. Se pueden utilizar, también, como impresoras rápidas, con velocidades de escritura que llegan a 1.625 líneas por minuto.

Los plotters que utilizan plumas con tinta pueden ser de dos tipos:

a) *De mesa:* el tamaño del papel es normalmente DIN A-3 o DIN A-4. Este se fija por efecto electrostático o mediante regletas imantadas. La pluma se desplaza por una guía o carro que a su vez es capaz de moverse en la dirección perpendicular sobre otras guías. La mesa puede ser horizontal (flatbed) o inclinada (beltbed).

b) *De tambor:* las plumas se desplazan a lo largo de la generatriz de un cilindro en el cual se enrolla el papel. Al mismo tiempo este cilindro o tambor puede girar en uno u otro sentido mediante un motor paso a paso. Se emplea el papel en rollo y, normalmente, permiten realizar dibujos de mayor tamaño que los plotters de mesa.

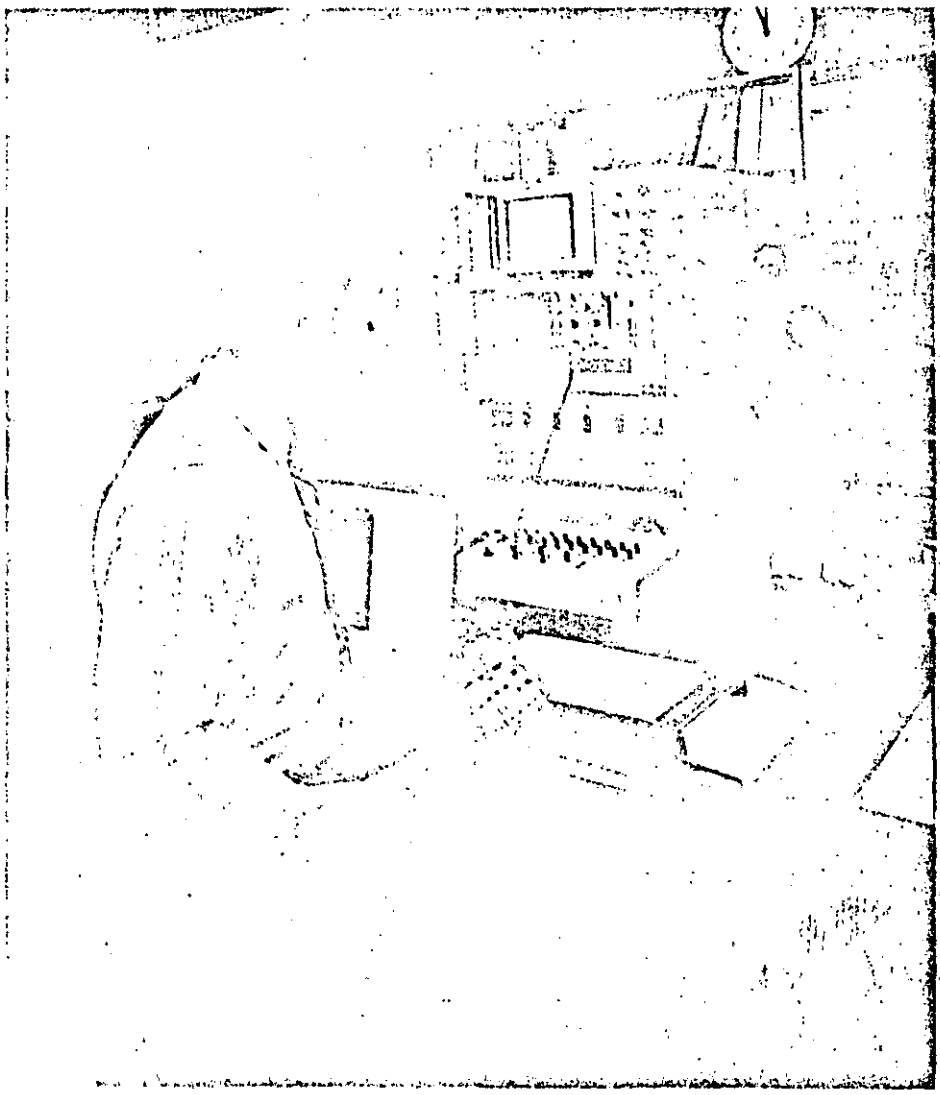
Características

Las características más importantes a la hora de evaluar un plotter son:

— *Paso incremental:* Debido a que el desplazamiento de las plumas por el papel se realiza mediante motores paso a paso, los desplazamientos son por incrementos. El paso incremental es el mínimo desplazamiento que puede realizar la pluma. En los plotters pequeños, el paso incremental es del orden de 0,1 mm ó 0,05 mm, mientras que en los grandes puede ser de 0,025 mm ó 0,0125 mm. De esta característica depende la resolución de los dibujos.

— *Resolución:* Es una característica análoga a la anterior y se expresa también en milímetros o en pulgadas. En los electrostáticos se expresa por el número de puntos por pulgada, normalmente de 100 a 200.

— *Precisión posicional estática:* Es la



Los plotters o trazadores gráficos se utilizan profusamente en el diseño de circuitos electrónicos: para dibujar el trazado del circuito impreso, el plano de montaje de los componentes...

PLOTTERS

precisión que tiene el sistema en posicionar la pluma en unas determinadas coordenadas. Se expresa su valor absoluto en milímetros o en pulgadas.

— *Velocidad de dibujo:* Es la velocidad máxima a la que se desplaza la pluma por el papel. Se expresa en mm/seg o en pulgadas por segundo (i.p.s.). Puede ser del orden de 100 mm/seg en los plotters pequeños, y de hasta 762 mm/seg (30 i.p.s.) en los grandes. En las características se dan dos tipos de velocidades:

a) *Axial:* es la velocidad de la pluma en su desplazamiento a lo largo de su guía.

b) *Diagonal:* es la velocidad resultante en el desplazamiento combinado de la pluma y del carro o del tambor.

La velocidad total de un dibujo no sólo depende de esta velocidad máxima, sino también de otros dos factores:

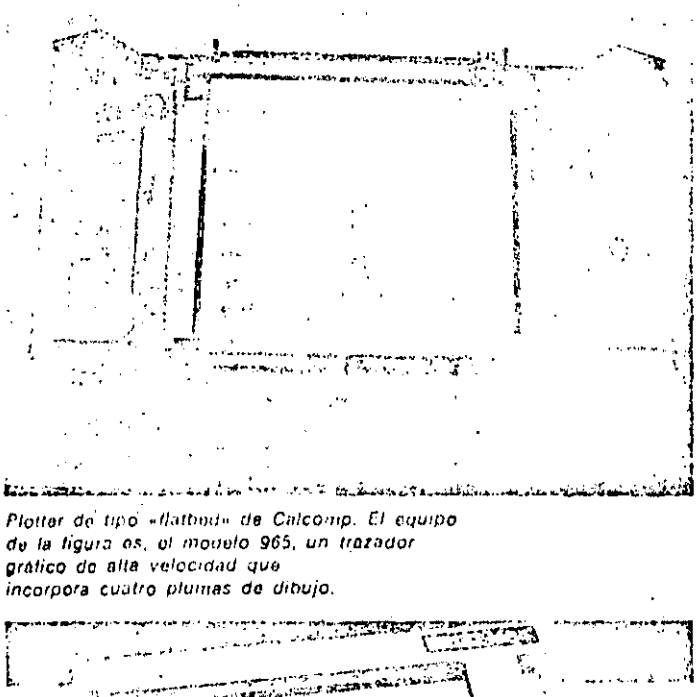
1. Aceleración: cuanto mayor sea la aceleración, antes se alcanza la velocidad máxima. Con una aceleración de 4 g se alcanza esa velocidad en una fracción de pulgada, y ello permite realizar prácticamente todo el dibujo a la velocidad máxima.

2. Tiempo de respuesta de las plumas: las plumas se aplican contra el papel mediante electroimanes y, lógicamente, tardan un tiempo tanto en subir como en bajar. Tiempos típicos de respuesta son de 2 mseg en subir y de 10 mseg en bajar.

— *Superficie de dibujo:* Son las dimensiones máximas del dibujo que puede realizarse con el plotter.

— *Número de plumas y colores:* Los plotters pueden disponer de distintas plumas de varios colores para la realización de los gráficos.

— *Funcionamiento on-line y off-line:* El plotter puede funcionar conectado directamente al ordenador (on-line), para lo cual algunos disponen de un buffer del mismo tipo que las impresoras. Sin embargo, debido a la poca velocidad de dibujo comparado con la velocidad de trabajo del ordenador, el



Plotter de tipo «flatbed» de Calcomp. El equipo de la figura es, el modelo 965, un trazador gráfico de alta velocidad que incorpora cuatro plumas de dibujo.

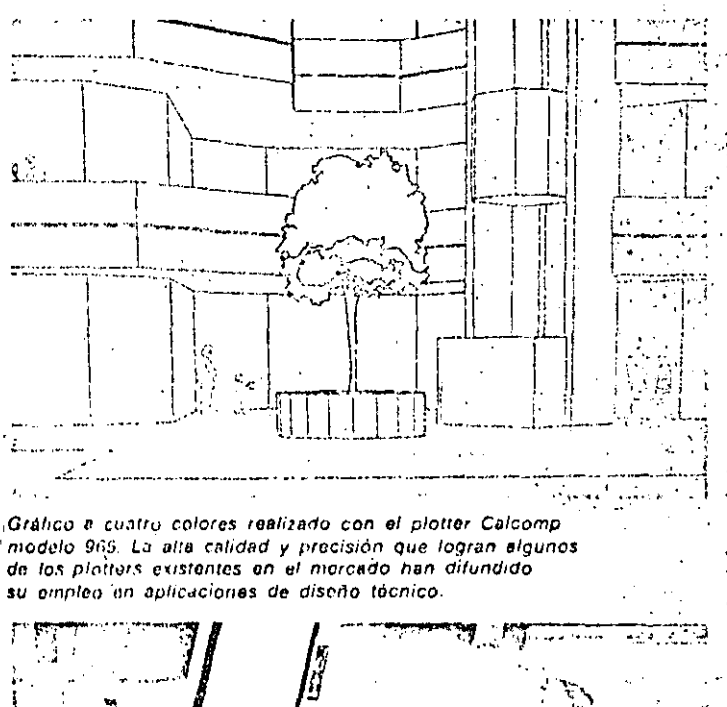


Gráfico a cuatro colores realizado con el plotter Calcomp modelo 965. La alta calidad y precisión que logran algunos de los plotters existentes en el mercado han difundido su empleo en aplicaciones de diseño técnico.

Plotter de mesa de tipo «bellhead». El modelo de la figura es el M84 de Calcomp, un trazador gráfico profesional orientado a la confección de gráficos y planos a color.

El campo de aplicación de los trazadores gráficos se incrementa día a día. De su empleo casi exclusivo en tareas de diseño técnico especializado, han saltado a aplicaciones de gestión y representación gráfica general.

funcionamiento normal de los plotters es off-line: la información correspondiente al dibujo a realizar se graba en una cinta magnética o en un disco y, posteriormente, mediante un controlador, se transfiere esa información al plotter.

— *Programas internos:* Los plotters provistos de microprocesadores internos son capaces de almacenar programas para el dibujo de caracteres o curvas clásicas. Mediante estos programas se pueden obtener sencillamente:

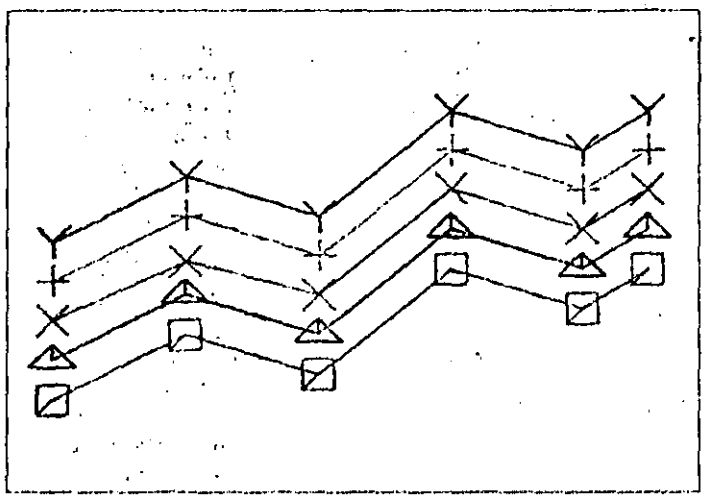
1. Generación de vectores: especificando las coordenadas de un punto de destino la pluma puede ir hasta ese punto. Las coordenadas pueden ser absolutas o relativas a la posición inicial de la pluma.
2. Generación de caracteres: el programa interno es capaz de generar y dibujar caracteres a partir del código ASCII correspondiente.
3. Generación de ejes y cuadrículas: se pueden dibujar líneas continuas, de trazos, marcas, etc.
4. Sombreados y entramados: útiles para la creación de gráficas.
5. Generación de círculos y arcos: los arcos se pueden dibujar especificando el radio y los ángulos de comienzo y final.
6. Distintos tipos de líneas: las líneas se pueden dibujar continuas, de trazos, de puntos, etc.
7. Generación de símbolos de dibujo.

— *Tipo de interface:* Las interfaces más empleadas normalmente son:

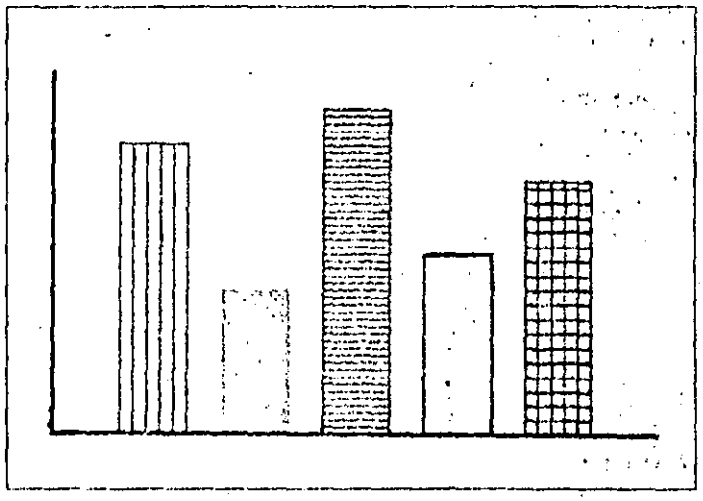
Paralelo: puede ser del tipo centronics, como en las impresoras, o de otros tipos, por ejemplo:

RS 232.
Bucle de 20 mA.
IEEE 488.

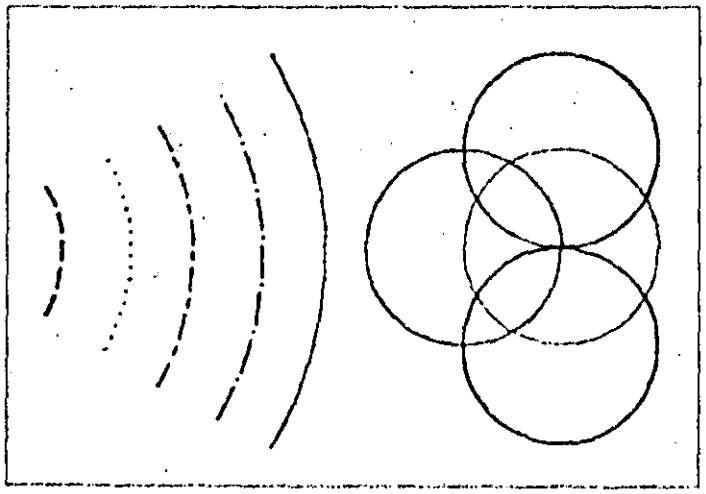
Otras características adicionales son: la tensión de alimentación (normalmente alterna), el consumo, la disipación de calor, temperatura, humedad de funcionamiento, etc.



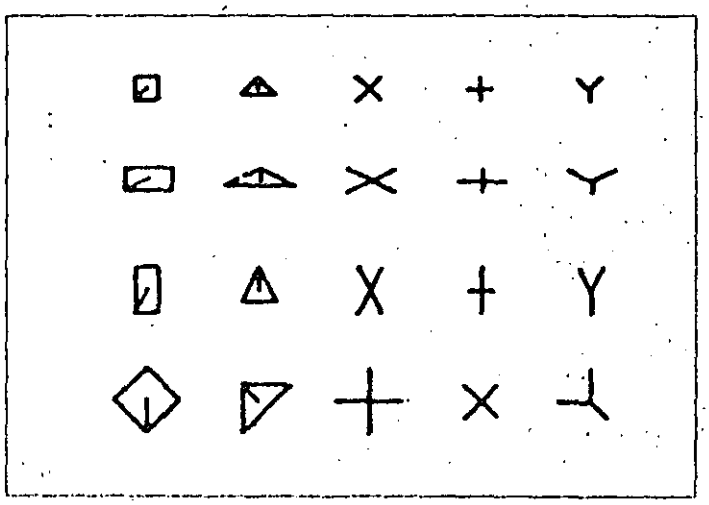
Los plotters provistos de microprocesador son capaces de almacenar programas para el trazado de curvas clásicas. Por ejemplo, para la generación de vectores.



Otros programas internos pueden ocuparse de la generación de sombreados y tramas para la confección de representaciones gráficas.



Uno de los programas internos almacenado en el plotter puede ocuparse del trazado automático de círculos y arcos de circunferencia, sin más que especificar el radio y los ángulos inicial y final.



Dentro de los programas internos, activables por comando, que suelen estar almacenados en los plotters comerciales, cabe destacar los destinados a la generación automática de símbolos y caracteres gráficos.

LA UNIDAD CENTRAL DE PROCESO

En capítulos anteriores hemos visto la configuración interna y el funcionamiento de la CPU de los microordenadores. Estas unidades centrales de proceso utilizan un microprocesador para realizar todas sus labores. Sin embargo, tanto los miniordenadores, como los ordenadores propiamente dichos, disponen de otros tipos de CPU, que aun teniendo las mismas misiones, éstas se realizan de formas muy distintas que en los microprocesadores. A continuación realizaremos un repaso a los principales tipos de unidades centrales de proceso.

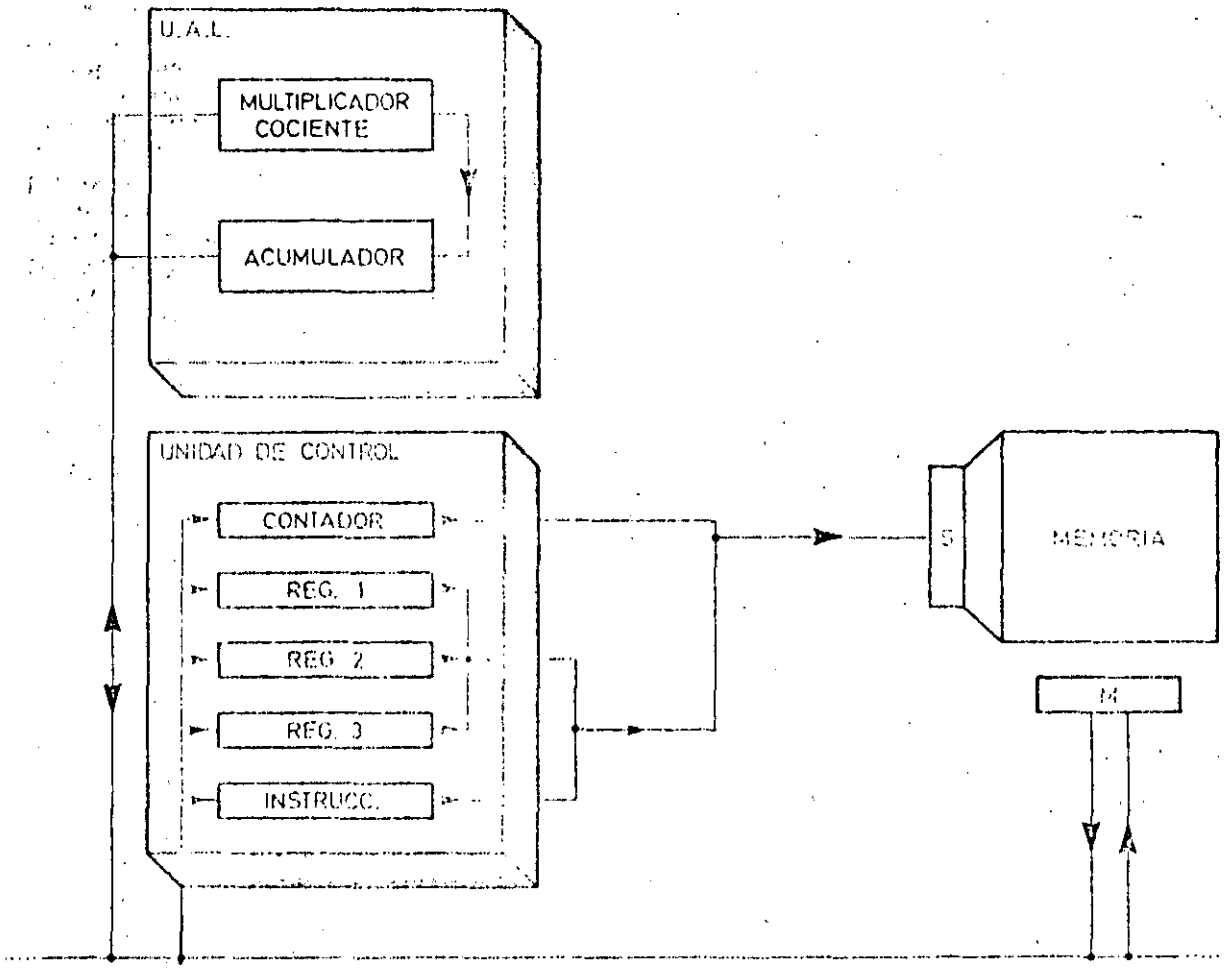
CPU, de ordenadores científicos

En la figura se ha representado esquemáticamente un ejemplo del típico ordenador científico. Dispone de tres registros-índice en la unidad de control, un acumulador y un multiplicador-cociente en la unidad aritmético-lógica. La longitud de sus instrucciones es fija: una palabra de máquina. La búsqueda en memoria de una de ellas se realiza, por tanto, durante un ciclo de reloj.

Una instrucción consta del código de operación, del bit para direccionamiento indirecto, de dos bits para direccionar cada uno de los tres registros-índice de la unidad de control y de la dirección en memoria. Los operandos también se codifican sobre una palabra de memoria en doble longitud. Con este tipo de ordenadores se consigue una alta velocidad en las operaciones de cálculo, a cambio de perder potencia en la gestión y utilización de ficheros. Estas características coinciden plenamente con los requerimientos para el proceso de datos de tipo técnico o científico, ya que el volumen de datos tratados no es excesivamente grande, pero con ellos se realizan muchos cálculos. Debido a la existencia de registros-índice y al multiplicador-cociente se logra también alta precisión en las operaciones (se redondea en posiciones de poco «peso») y se puede llegar a trabajar con números muy elevados.

CPU, de ordenadores de gestión

Las máquinas destinadas a la gestión suelen tratar caracteres (en vez de palabras como las científicas), la longitud



Esquema de la CPU de un ordenador científico. Con el empleo de registros índice y de un multiplicador-cociente se reduce el tiempo de ejecución de las operaciones, logrando, a la vez, una gran precisión en los resultados.

LA UNIDAD CENTRAL DE PROCESO

de las instrucciones es variable y generalmente trabajan con dos direcciones. Las cadenas de caracteres van limitadas por una marca de fin de cadena que ocupa una posición binaria en memoria y puede ser controlada mediante instrucciones especiales. Cuando una instrucción se refiere a una cadena de caracteres, contiene la dirección del último carácter de la misma. Evidentemente este tipo de ordenador está preparado para trabajar con datos alfanuméricos (en vez de numéricos, como los científicos).

A pesar de su carácter poco matemático, estos ordenadores deben ser capaces de realizar operaciones como: sumar o multiplicar. La forma de realizar estas operaciones es carácter a carácter. Vamos a describir, como ejemplo, la instrucción «suma de dos números»:

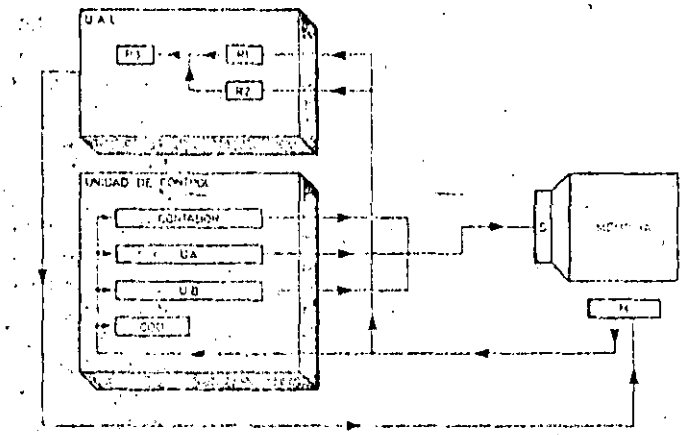
La instrucción contiene el código de la operación (suma) en un carácter y las direcciones de los últimos caracteres de los dos operandos (número a sumar). Por tanto, en la unidad de control se dispone de, al menos, cuatro registros: dos para las direcciones de los últimos caracteres de los números a sumar, otro registro COD para el código de operación y un cuarto para el contador de instrucciones.

La operación de suma se realiza en serie, carácter por carácter. Los caracteres del primero y del segundo operando se memorizan, respectivamente, en dos registros R_1 y R_2, y el arrastre es conservado, de un paso al siguiente, en el biestable R_3. La operación de suma continúa mientras no se detecte la señal de fin de cadena.

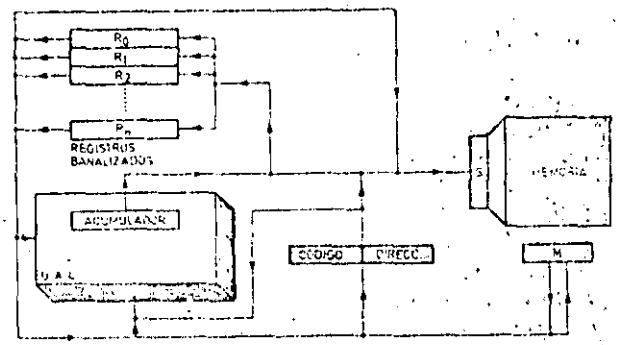
CPU, de máquinas ambivalentes

En la tercera generación de ordenadores se comercializaron máquinas de distintos tipos, pero con el mismo conjunto de instrucciones, y aptos para resolver tanto problemas de gestión, como problemas científicos. Surgió también un nuevo tipo de registros banalizados que pueden ser utilizados tanto como registros de direccionamiento como para almacenar operandos.

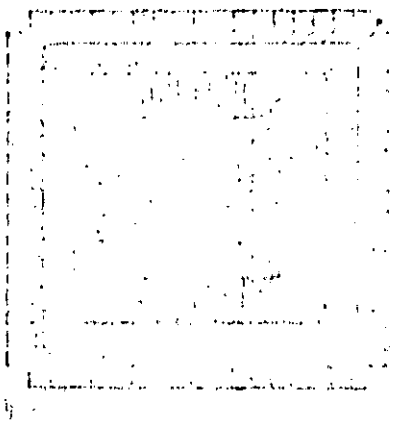
En la figura se representa una configuración típica de un sistema ambivalente. El sistema posee un conjunto de



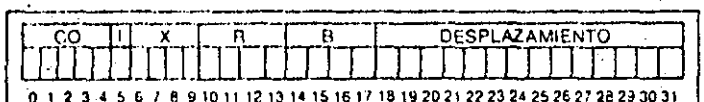
Ejemplo de CPU para un ordenador de gestión. En los registros UA y UB se almacenan las direcciones de los últimos caracteres de los números a sumar; el registro COD contiene el código de operación.



Esquema de una CPU típica de cualquier ordenador de la tercera generación. Un único sumador actúa como unidad aritmético-lógica y como unidad de cálculo de direcciones.



Unidad central de proceso de un ordenador de 32 bits. Con los microprocesadores actuales no se alcanza, aún, la potencia y velocidad proporcionadas por una CPU de este tipo



CO - 5 bits — Código de operación (hasta 32 operaciones).
I - 1 bit — Tipo de direccionamiento (0 ⇒ directo, 1 ⇒ indirecto). (excepto si X = 0000).
X - 4 bits — Contiene la posición de memoria en que se encuentra el número de registro direccionado (excepto si X = 0000)
R - 4 bits — Dirección del registro con el primer operando.
B - 4 bits — Dirección del registro de base (si B = 0000 hace referencia al contador de instrucciones).
Desplaz. 14 bits — Capaz de direccionar 2^{14} = 16 Kbytes a partir del registro base.

Configuración típica de una instrucción de 32 bits.

registros banalizados utilizables como registros aritméticos, como registros de direccionamiento, como registros de base o como registros índice. En esta máquina existe un único sumador, en el que se realizan todos los cálculos, que actúa como unidad aritmética lógica y como unidad de cálculo de direcciones. No dispone de contador de instrucciones. Esta misión se realiza a través del registro banalizado Ro.

Características de los ordenadores mixtos de gestión y científicos

Las propiedades más importantes de estos ordenadores son las siguientes:

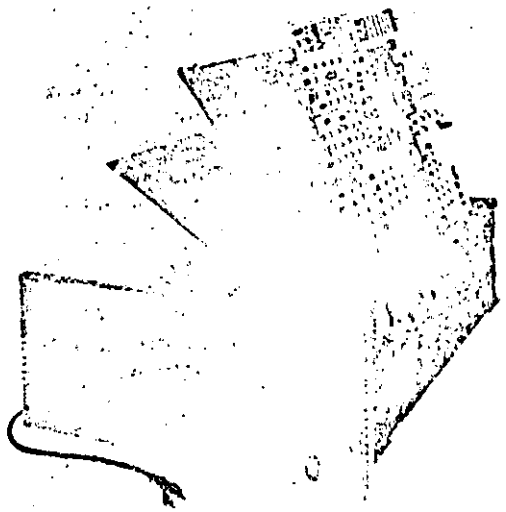
- Capacidad de direccionamiento, tanto a nivel de carácter como de palabra.
- Conjunto de instrucciones que incluye tanto las instrucciones de ordenadores de palabra (una única dirección), como las instrucciones de ordenadores de carácter (dos direcciones).
- Bus de datos, capaz de transportar tanto palabras como caracteres.

A continuación detallamos los procedimientos empleados para realizar las tres características mencionadas.

- *Direccionamiento de palabras y caracteres*

Existen dos opciones:

1. Situar sistemáticamente los fines de cadena de caracteres en los fines de palabra. De esta forma todas las instrucciones se direccionan a nivel de



La CPU de los miniordenadores y ordenadores de gran tamaño no suele estar constituida por un microprocesador, sino por elementos más complejos, capaces de ejecutar operaciones específicas del sistema para el que están diseñados.

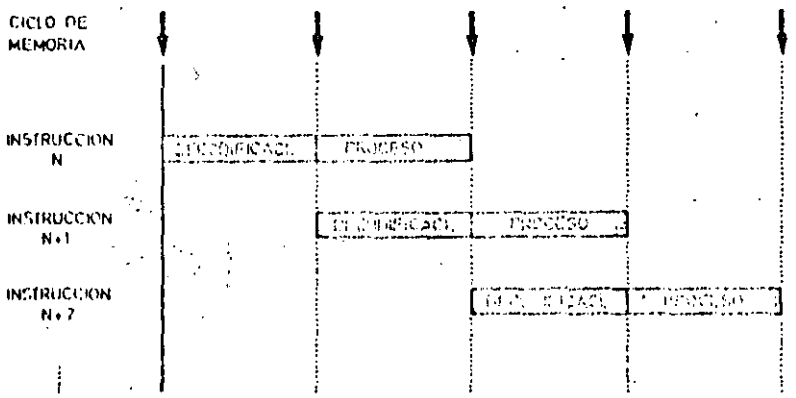


Diagrama de los tiempos teóricos de solapamiento de instrucciones. Mientras que se ejecuta la instrucción N, la instrucción N + 1 se está decodificando. Con esta técnica se doblá la velocidad de proceso de la máquina.

Glosario

¿Cuáles son las principales características de los ordenadores científicos?

Deben ser muy precisos y rápidos en el manejo de números. La CPU de estos ordenadores dispone para ello de una unidad aritmético-lógica muy potente que trabaja con palabras.

¿Cuáles son las principales características de los ordenadores de gestión?

Su principal misión estriba en el manejo de datos. Se caracterizan, por tanto, en la buena gestión de ficheros. La CPU trabaja, básicamente, con cadenas de caracteres.

¿Existen ordenadores capaces de resolver eficazmente problemas científicos y de gestión?

En efecto. En el comienzo de la tercera generación aparecieron los primeros ordenadores destinados al proceso de datos en general.

¿Qué son los registros banalizados?

Son aquellos capaces de funcionar a la vez como registros aritméticos y como registros de direccionamiento. En los ordenadores mixtos, normalmente, no se dispone de contador de instrucciones en la unidad de control. Esta función la realiza un registro banalizado.

¿Qué tipo de direccionamiento se utiliza en los ordenadores mixtos?

Pueden utilizar direccionamiento a nivel de carácter o a nivel de palabra.

¿Qué relación existe entre el tamaño de la palabra procesada por la CPU y el de la palabra almacenada en memoria?

Existen dos opciones: que el tamaño de las palabras de memoria se ajuste al de las instrucciones más cortas o con el de las instrucciones más largas.

LA UNIDAD CENTRAL DE PROCESO

palabra. Esta opción es, en general, inaceptable.

2. Utilizar únicamente direcciones a nivel de carácter. De esta forma los ordenadores, cuya longitud de palabra es 2^n caracteres, desaprovechan los n últimos bits de la dirección en las instrucciones que direccionen palabras.

• *Instrucciones de longitud variable*

Las operaciones con palabras son ideales para realizar instrucciones de tipo registro-memoria, en cambio las operaciones con caracteres se adaptan mejor a instrucciones de tipo memoria-memoria. Esto obliga a que las instrucciones tengan diferentes longitudes. Mediante la decodificación de uno o dos bits, situados normalmente en el principio del código de operación, la unidad de control conocerá inmediatamente la longitud de la instrucción que está tratando. Evidentemente los incrementos en el contador de instrucciones se realizarán según las longitudes de las instrucciones procesadas.

• *Bus de datos*

En los ordenadores mixtos existen dos tipos de procesamiento para los operandos:

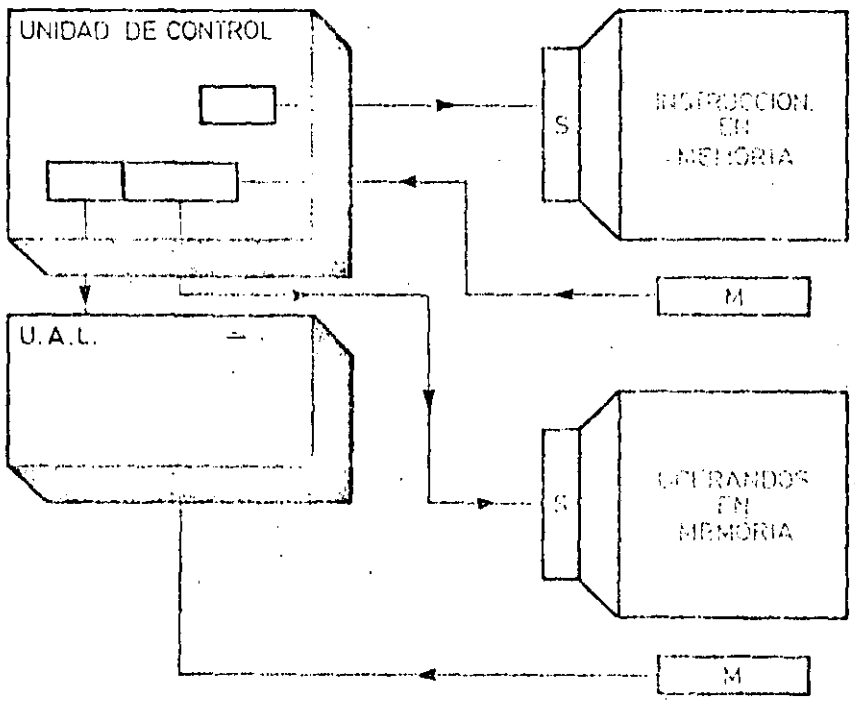
— Los datos se transfieren carácter por carácter. En este caso las instrucciones con palabras necesitan varias pasadas por el bus.

— El bus de datos tiene la misma dimensión que la palabra. Los distintos caracteres de una palabra se transfieren en paralelo, pero se procesan en la unidad aritmético-lógica en serie, mediante el operador de caracteres.

En cuanto a la relación entre instrucciones y longitud de la palabra de memoria existen dos posibles elecciones:

— La palabra de memoria tiene la misma longitud que las instrucciones cortas y, por tanto, las instrucciones largas exigirán dos accesos a memoria y un doble registro de instrucción. Cuando se ejecuta una instrucción corta, el contador de instrucciones se incrementa en una unidad. Cuando se ejecuta una instrucción larga el contador se incrementa en dos unidades.

— La palabra de memoria tiene la longitud de las instrucciones largas. En este caso los procedimientos serán más complejos que en el caso anterior. Es necesario utilizar un registro tampón largo en el registro de instrucciones, igualmente largo.



Máquina teórica con bus de instrucciones y bus de datos independientes. Este tipo de configuración se realiza en la práctica mediante software.

LA UNIDAD ARITMETICO-LOGICA

COMO ya vimos anteriormente la unidad aritmético-lógica de un ordenador es la encargada de manipular y operar con los datos. Abreviadamente se la suele designar por las siglas: UAL.
La UAL es capaz de efectuar un determinado número de operaciones elementales, en consonancia con el repertorio de instrucciones del ordenador. Estas operaciones sólo pueden ser de tres tipos: operaciones lógicas, operaciones aritméticas y operaciones de desplazamiento.
Para resolver cualquier operación compleja es necesario reducirla a operaciones elementales que sean procesables por la UAL. Tanto la estructura como el procedimiento operativo y el repertorio funcional suelen ser muy parecidos en todas las unidades aritmético-lógicas del mismo tipo. Las diferencias sensibles surgen al comparar unidades de distintos tipos.

Procedimientos operativos

Sólo existen dos procedimientos básicos operativos para la UAL: en serie y en paralelo.

- *Procedimiento en serie*

Un circuito lógico que trabaja en serie, ejecuta las operaciones secuencialmente, bit a bit.
Suponemos que las entradas al operador en serie son dos registros, A y B, de cuatro bits cada uno:

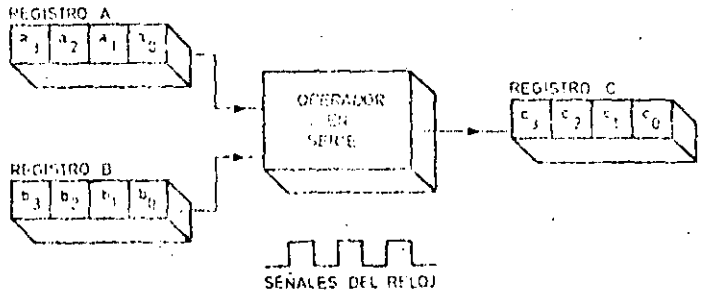
$[a_3][a_2][a_1][a_0]$ y $[b_3][b_2][b_1][b_0]$

respectivamente, y el resultado es un registro C, compuesto por los bits $[c_3][c_2][c_1][c_0]$
El operador realiza la operación con A y B, bit a bit, depositando los resultados sucesivos en los bits del registro C. Cada operación elemental precisa un impulso del reloj y la UAL sólo opera sobre un par de elementos, por lo que los operadores en serie son más económicos que los operadores en paralelo, aunque más lentos.
La UAL procede de la siguiente forma:
1. Opera sobre los bits a_0 y b_0 y almacena el resultado en c.
2. Opera sobre los bits a_1 y b_1 y almacena el resultado en c_1.
3. Opera sobre los bits a_2 y b_2 y almacena el resultado en c_2.
4. Y, finalmente, opera sobre los bits a_3 y b_3 y almacena el resultado en c_3.

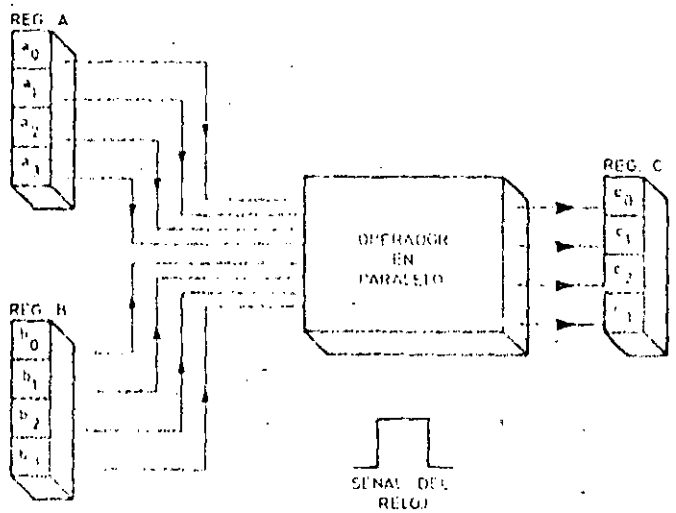
- *Procedimiento en paralelo*

Un circuito lógico trabaja en paralelo cuando efectúa simultáneamente la operación sobre todos los bits que componen cada dato. Suponiendo los mismos registros de entrada, A y B, que en el caso anterior, y el mismo registro de salida C, la operación se lleva a cabo en una sola fase y, por tanto, es necesario un único impulso del reloj. El resultado se almacena, completo, en el registro C.
Para conseguir esto, el operador paralelo utiliza cuatro circuitos operativos elementales y, en consecuencia será más caro que el operador en serie, aunque también será cuatro veces más rápido.
El procedimiento consta de un único paso:

1. Operación con los registros A y B, y



Un operador en serie trabaja secuencialmente con pares de bits. Para completar una operación sobre dos registros de cuatro bits cada uno, se necesitan, por tanto, cuatro impulsos de reloj.



Un operador en paralelo trabaja simultáneamente con todos los bits de los registros de entrada. Una operación elemental sobre dos registros se ejecuta mediante un único impulso de reloj.

LA UNIDAD ARITMETICO-LOGICA

almacenamiento del resultado en el registro C.

Los microprocesadores actuales proceden generalmente en paralelo, debido a la mayor velocidad alcanzada por este procedimiento, y a que la microelectrónica permite hoy día integrar los componentes necesarios en un volumen físico reducido.
Los operadores en paralelo pueden, a su vez, subdividirse según su tipo de registros asociados. Los dos grupos más importantes se describen a continuación:

Operadores con dos registros de entrada y un registro de salida

Este es el caso más sencillo. Los dos registros de entrada memorizan los datos con los que se va a operar; el registro de salida contiene el resultado de la operación. Los tres registros son independientes entre sí. Su funcionamiento es análogo al descrito anteriormente para el procedimiento en paralelo.

Operadores con acumulador, en paralelo

En este caso tan sólo se dispone de un único registro de entrada. El segundo dato y el resultado comparten un registro especial, llamado acumulador. Su funcionamiento es el siguiente:

1. Se carga en el acumulador el primer dato, procedente del registro de entrada.
2. Se carga el segundo dato en el registro de entrada. Los dos operandos están ahora almacenados, uno en el acumulador y otro en el registro de entrada.
3. Se realiza la operación y se sustituye el contenido del acumulador, por el resultado de dicha operación.

Los microprocesadores actuales trabajan, generalmente, con unidades aritmético-lógicas de este último tipo. Es decir, la UAL es un circuito operador en paralelo con acumulador.

Organización de la UAL

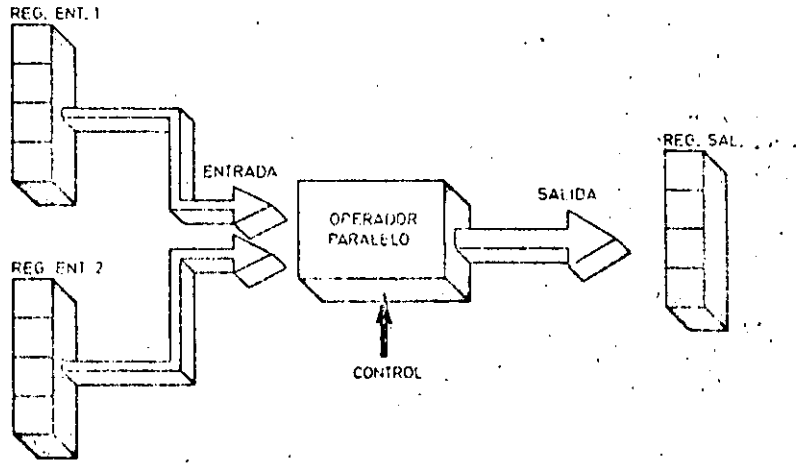
La organización de la UAL se completa con las señales externas, mediante las que es gobernada por la unidad de control (denominadas comandos de operación), y con determinado número de salidas hacia unos biestables, denominados biestables de estado, que indican alguna condición especial, surgida al ejecutar una operación en la UAL (resultado igual a cero, desbordamiento de la capacidad del registro, etc.).
Es necesario, también, garantizar la sincronización del funcionamiento de la UAL. Esta tarea está encomendada a un conjunto de señales de control que, estrictamente, no acceden a la UAL, sino que actúan sobre ella a través de los registros de entrada y del acumulador.

Operaciones de la UAL

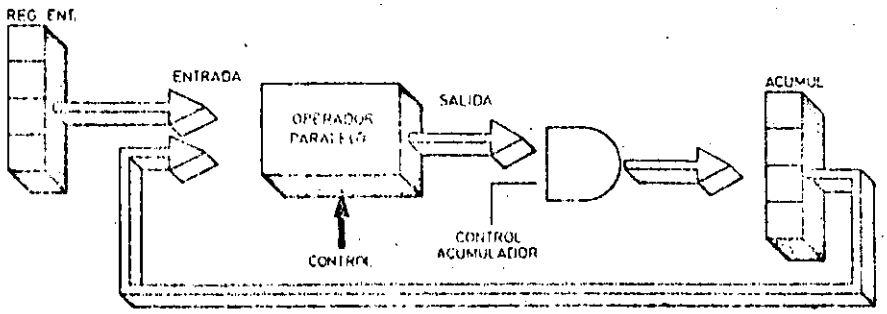
Las operaciones que puede realizar una UAL se pueden clasificar en tres grandes grupos: operaciones lógicas, operaciones aritméticas y operaciones de desplazamiento.

- *Operaciones lógicas*

Las operaciones lógicas que ejecuta



Ejemplo de operador en paralelo, sin acumulador. La entrada y la salida de este tipo de circuitos se encuentran totalmente separadas.



Operador paralelo con acumulador. Uno de los datos entra por el registro de entrada y el segundo por el acumulador. El resultado de la operación se almacena en el acumulador.

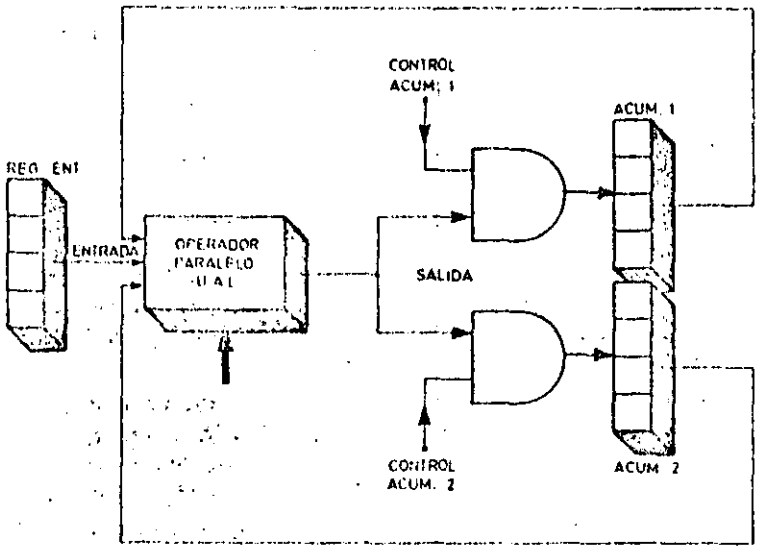
una UAL, coinciden, por lo general, con las funciones lógicas booleanas, y están implementadas en su interior mediante circuitos operativos situados entre el registro de entrada y el acumulador. Como ejemplo veremos la resolución circuital de la función producto lógico (AND).

A partir de la tabla de verdad asociada a la operación booleana AND se deducen las siguientes condiciones que debe cumplir el circuito de la UAL.

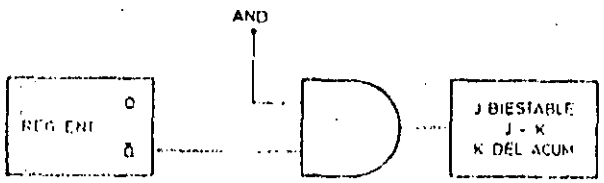
1. Si el bit almacenado en el registro de entrada es igual a 0, el biestable del acumulador debe ser puesto a 0.

2. Si, por el contrario, el bit almacenado en el registro de entrada es igual a 1, el contenido del acumulador permanece inalterado.

La función AND se implementa, en consecuencia, conectando la salida Q del registro de entrada a la entrada K del biestable J-K del acumulador, y uniendo la salida de este biestable con su entrada J.



Configuración típica de una unidad aritmético-lógica con dos acumuladores. Este tipo de estructura proporciona una gran versatilidad en ordenadores de tamaño medio.



ENT	ACU	ACU
0	0	0
0	1	0
1	0	0
1	1	1

Circuito destinado a efectuar la operación producto lógico de dos bits en la UAL y tabla de verdad correspondiente. La entrada AND está conectada a la salida del acumulador.

Glosario

¿Qué tipos de operaciones puede realizar la UAL?

Tres. Operaciones lógicas, operaciones aritméticas y operaciones de desplazamiento. En cualquiera de los casos, las operaciones deben ser elementales, de forma que para realizar una operación compuesta, ésta se debe desglosar antes en varias operaciones elementales.

¿Qué procedimientos operativos puede seguir la UAL?

Procedimiento en serie, que consiste en el tratamiento secuencial de los bits de las entradas, y procedimiento en paralelo, que consiste en operar con todos los bits de las entradas, a la vez.

¿Qué ventajas o inconvenientes tienen cada uno de los procedimientos operativos?

El procedimiento operativo en serie es más barato que el procedimiento en paralelo. Sin embargo, el procedimiento en paralelo es más rápido que el procedimiento en serie. La mayoría de los microprocesadores utilizan procedimientos en paralelo.

¿Dentro de los procedimientos en paralelo, qué tipos de operadores existen?

Dos. Uno está formado por los operadores con dos registros de entrada y uno de salida. El otro se compone de un único registro de entrada y un registro especial, llamado acumulador.

¿Además de los operadores, qué señales son necesarias para completar la organización de la UAL?

Un grupo de señales denominadas comandos de operación, mediante las cuales la unidad de control gobierna a la UAL; y un segundo grupo de señales encargadas de garantizar la sincronización del funcionamiento.

¿Qué son los registros de condición?

Son unos biestables que indican alguna condición especial surgida al ejecutar una operación, y que pueden ser utilizados por el programa para tomar determinadas decisiones.

LA UNIDAD ARITMETICO-LOGICA

• *Operaciones aritméticas*

Las dos únicas operaciones básicas que es capaz de ejecutar una UAL son la suma y la resta. Normalmente estas operaciones se realizan sobre números codificados en binario natural. A continuación vamos a construir el circuito encargado de la suma aritmética.
El circuito utilizará tres entradas: A y B con los bits de los dos números a sumar, y C, con el acarreo procedente de la etapa anterior. Tendrá dos salidas: Y para el resultado de la suma, y Z para el acarreo a propagar en la siguiente etapa. Sin más que observar la tabla de verdad de esta operación se pueden construir las funciones de salida y, a partir de ellas, el circuito lógico sumador.

• *Operaciones de desplazamiento*

Estas operaciones alteran la posición relativa de los bits almacenados dentro de un registro. Ejecutan dos tipos fundamentales de desplazamiento: corrimiento de todos los bits una posición hacia la izquierda, o corrimiento hacia la derecha.

TABLA DE VERDAD Y FUNCIONES BOOLEANAS DE SALIDA PARA EL CIRCUITO LOGICO SUMADOR

C	A	B	Y	Z
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

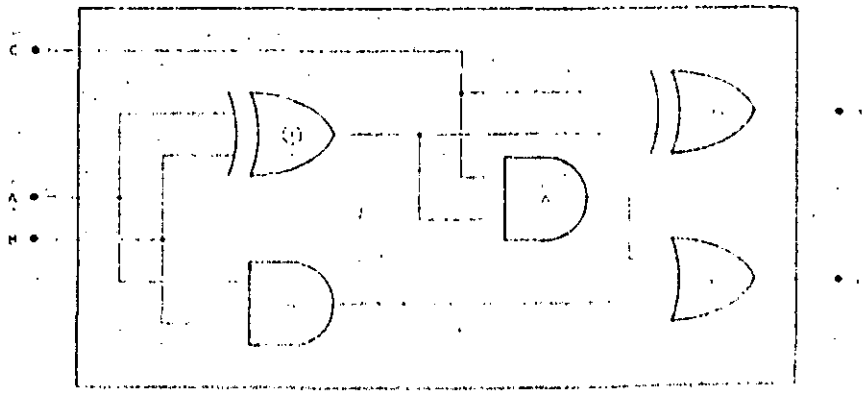
$$y = (A \wedge \bar{B} \wedge \bar{C}) \vee (\bar{A} \wedge B \wedge \bar{C}) \vee (\bar{A} \wedge \bar{B} \wedge C) \vee (A \wedge B \wedge C) =$$

$$= A \oplus B \oplus C$$

$$z = (A \wedge B \wedge \bar{C}) \vee (A \wedge \bar{B} \wedge C) \vee (\bar{A} \wedge B \wedge C) \vee (A \wedge B \wedge C) =$$

$$= (A \wedge B) \vee (C \wedge (A \oplus B))$$

Tabla de verdad y funciones lógicas que realiza un circuito lógico sumador. La entrada «C» corresponde al acarreo de la última operación y la salida «Z» al acarreo de la operación en curso.



Realización práctica, mediante puertas lógicas, de un circuito sumador de dos bits con salida de suma (Y) y de acarreo (Z)

Conceptos Básicos

Máquinas de Turing (2)

En el número anterior se describieron los fundamentos teóricos de las máquinas de Turing.
A título de ejemplo vamos a describir en éste una máquina de Turing capaz de buscar el último «1» del bloque en que se encuentre y quedarse parada en esa posición. El conjunto de instrucciones es el siguiente:

```
0 • r 1
0 1 r 1
1 • • 2
1 1 1 0
2 • l 3
2 1 l 3
3 • s 3
3 1 s 3
```

comprobemos que funciona:

ESTADO	CINTA POSICION ACTIVA			INSTRUCCION
0	11	1	1 •	0 1 r 1
1	111	1	•	1 1 1 0
0	111	1	•	0 1 r 1
1	...1111	•	.....	1 • • 2
2	...1111	•	.....	2 • 1 3
3	...111	1	•	3 1 s 3

En efecto, la posición activa termina situada en el último «1» del bloque en que se encontraba al comenzar la ejecución.
No debemos asustarnos ante la complejidad de máquinas que resuelvan problemas más complicados: la única utilidad de las máquinas de Turing es teórica, y tan sólo sirven para temas relacionados con la Teoría de la Computabilidad.

UNIDADES DE MEMORIA

UNA Memoria es un dispositivo capaz de almacenar información binaria. Las características tecnológicas de la unidad de almacenamiento o memoria de un ordenador quedan perfectamente determinadas por las características inherentes a su celda básica de almacenamiento o «punto de memoria». El punto de memoria es el elemento físico capaz de almacenar un dígito de información binaria o bit.

Clasificación de las memorias

La clasificación de las unidades de memoria puede hacerse a partir de diferentes conceptos de referencia. Comúnmente se establece una primera clasificación general atendiendo a la jerarquía que corresponde la unidad de memoria dentro del sistema de proceso.

La jerarquía es un concepto de clasificación que obedece a dos propiedades de todas las memorias: velocidad de trabajo y capacidad de almacenamiento.

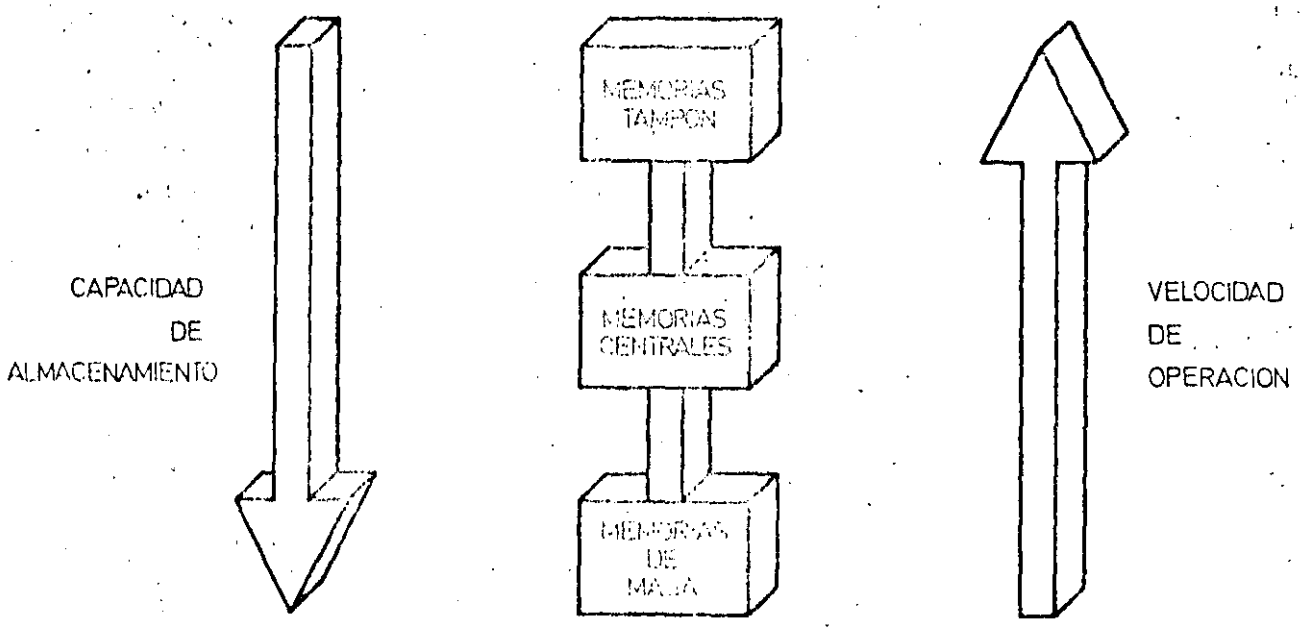
Los diversos tipos de memorias catalogados en los sucesivos órdenes jerárquicos son los siguientes:

• *Memorias tampón:* Son memorias de baja capacidad y alta velocidad. Actúan habitualmente como memorias auxiliares en la transferencia de información entre la unidad central de proceso y las unidades de entrada y salida.

Los denominados registros banalizados o de propósito general de algunos microprocesadores son memorias de este tipo.

• *Memorias centrales:* Bajo esta denominación se incluyen las memorias de trabajo que se hallan asociadas a la CPU y forman parte del sistema organizado.

Su misión consiste en almacenar los programas y los datos y resultados implicados en la ejecución de las sucesivas instrucciones.



Clasificación jerárquica de las memorias. Las más rápidas son las que tienen menor capacidad de almacenamiento. Por el contrario, las memorias de masa son las que pueden almacenar mayor cantidad de información, aunque su velocidad de operación es muy baja

UNIDADES DE MEMORIA

• *Memorias de masa:* Son memorias de acceso directo o aleatorio y de elevada capacidad. Se emplean como bloques de almacenamiento auxiliar. Su velocidad de transferencia de información es muy elevada.
Para que la CPU pueda tratar determinada información almacenada en una memoria de masa ésta debe pasar inicialmente al interior de la memoria central del sistema. En virtud del tipo de transferencia empleado, la característica básica de las memorias de masa es su caudal de transferencia o número de bytes de información que puede transferir por unidad de tiempo. Se expresa en Kbytes/seg o Mbytes/seg.

Las memorias de masa que alcanzan mayor difusión en el campo de los microordenadores son los discos magnéticos flexibles o «Floppy disk».

Características generales de las memorias

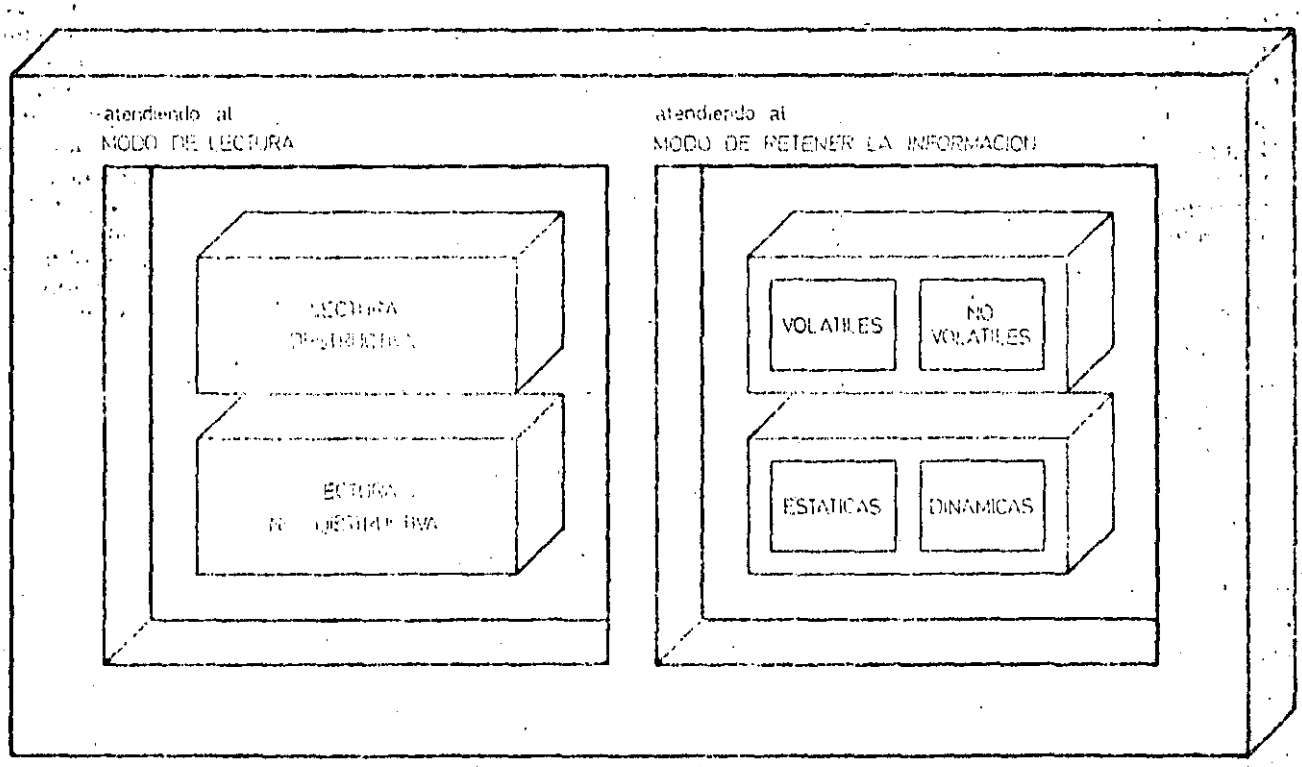
La característica más comúnmente utilizada para describir a una memoria en concreto es su capacidad de almacenamiento. Existe, sin embargo, otra serie de particularidades que permiten conocer mejor a este tipo de elementos:

• *Tiempo de acceso:* Es la media del intervalo de tiempo transcurrido desde que se solicita un dato a la unidad de memoria hasta que ésta lo entrega.

• *Ciclo de memoria:* Se define como el tiempo que transcurre desde que se solicita un dato a la unidad de memoria hasta que ésta se halla disponible para efectuar una nueva operación (lectura o escritura).

• *Tiempo medio de acceso:* Se define como el tiempo de acceso a una posición intermedia de la memoria emplazada respecto de los límites de acceso inmediato y extremo.

• *Acceso aleatorio:* Una memoria es de



Clasificación de las memorias, atendiendo a sus características de lectura y de modo de retener la información

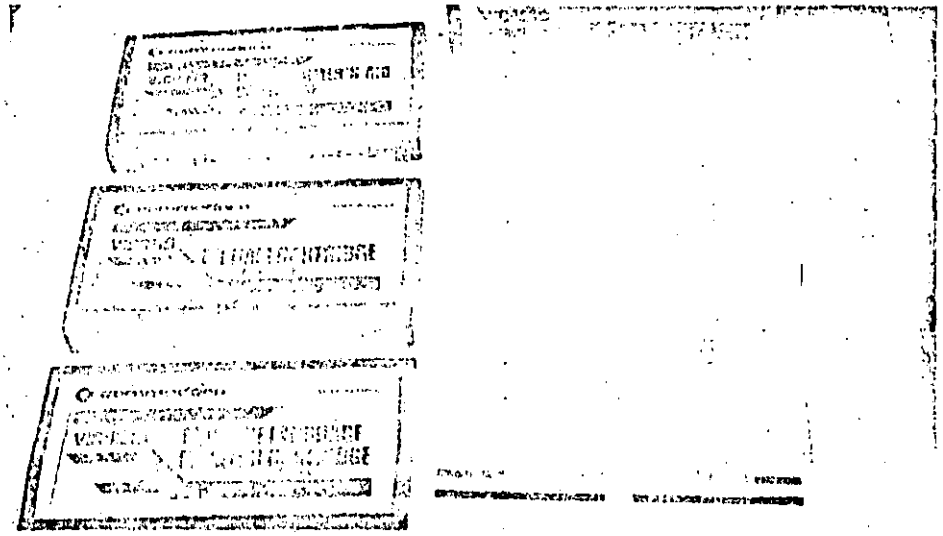
acceso aleatorio cuando el tiempo de acceso a cualquier palabra de información almacenada es de valor constante. La memoria central de los sistemas de proceso debe ser, necesariamente, de acceso aleatorio.

Clasificación básica de las memorias centrales

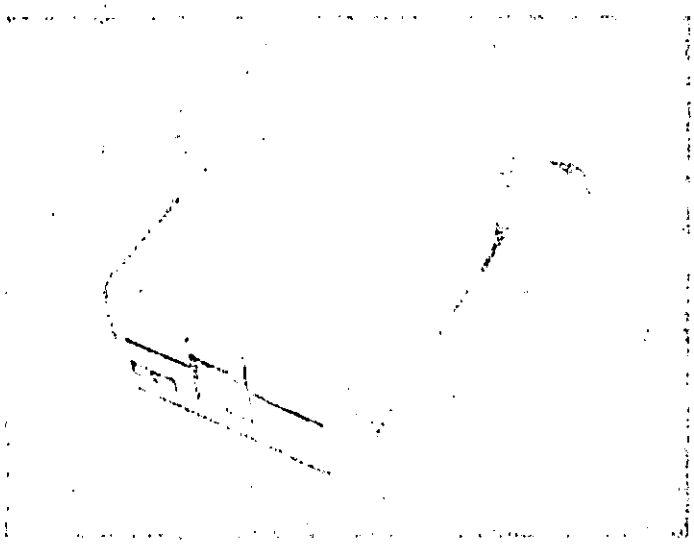
La memoria central de un sistema basado en microprocesador consta, por lo general, de varias unidades de almacenamiento que poseen una característica básica en común: son memorias de acceso aleatorio.

Dentro del campo de las memorias centrales se establecen varios conceptos que dan lugar a diversas clasificaciones. Estos conceptos de referencia son dos: modo de lectura y retención de la información almacenada.
Atendiendo al «modo de lectura» podemos establecer la siguiente clasificación de las memorias centrales:

• *Memorias de lectura destructiva:* Al leer determinada posición de memoria, la información almacenada desaparece. Este tipo de memorias precisan de una regeneración del contenido después de efectuada la operación de lectura.



Las memorias de casi todos los sistemas informáticos presentes en el mercado admiten ampliaciones por medio de módulos de expansión como los de la figura.



Una lectora de discos flexible: es capaz de leer y escribir datos en un soporte magnético o «disco», que constituye un tipo de memoria de masa muy común. Estos soportes pueden llegar a almacenar del orden de varios Megabytes de información.

Conceptos básicos

Memorias de núcleos de ferrita

Un núcleo de ferrita es un cilindro capaz de magnetizarse en dos direcciones diferentes, a cada una de las cuales se asocia uno de los dos estados lógicos «uno» o «cero».
Para almacenar un estado binario se hace pasar una corriente eléctrica por un hilo conductor que atraviesa al núcleo. La dirección de la corriente eléctrica en el hilo magnetiza al núcleo en un sentido o en otro. En cualquier caso la corriente debe superar a un valor límite I_0.

Escritura en una memoria de núcleo magnético

Cuando la intensidad de la corriente que atraviesa el núcleo no supera el valor límite I_0 el campo magnético inducido desaparece al cesar la corriente. Si, por el contrario, la corriente es mayor, el campo magnético permanece en ausencia de corriente. A I_0 se le denomina punto de no retorno. De esta forma, para grabar un uno lógico se hace pasar por la ferrita una corriente superior a I_0, que produce un campo magnético de valor B. Para grabar un cero se hace pasar una corriente inferior a $-I_0$, con lo que el campo inducido toma un valor $-B$.

Lectura en una memoria de núcleos magnéticos

Para leer el contenido de un núcleo previamente grabado se emplea un segundo hilo conductor, por el que se hace pasar una corriente eléctrica de valor $-I_0$. Si el campo tiene una intensidad B_1 se produce una corriente inducida en el primer hilo de un determinado valor que se identificará como de uno lógico. Cuando el valor del campo almacenado sea $-B_1$ la corriente inducida corresponderá al de un cero lógico.

Método de selección en una memoria de núcleos magnéticos

Los núcleos de ferrita se disponen en forma de matriz: cada fila representa una dirección (normalmente compuesta por 8 núcleos o bits. Para seleccionar la fila se manda la dirección de memoria a un circuito combinacional que determina el hilo en que se encuentra el octeto direccionado. Por este hilo se hace pasar una intensidad $I/2$. Por todas las columnas de la matriz de memoria está circulando continuamente una corriente de intensidad $I/2$. Los únicos 8 núcleos de ferrita atravesados por una intensidad I ($I/2 + I/2$) son los correspondientes al octeto seleccionado.

UNIDADES DE MEMORIA

• *Memorias de lectura no destructiva:* Las operaciones de lectura no provocan la pérdida de la información almacenada.

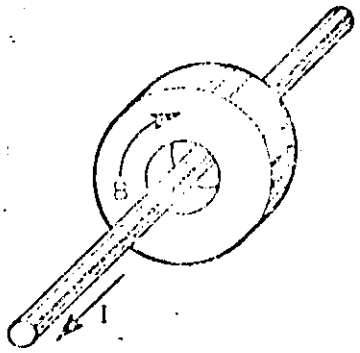
Atendiendo al modo de «retener la información» cabe hacer dos clasificaciones parciales:

• *Memorias volátiles o no volátiles:* Las memorias volátiles son aquellas que requieren la presencia de una fuente de alimentación. Al desconectar ésta, se pierde la información almacenada.

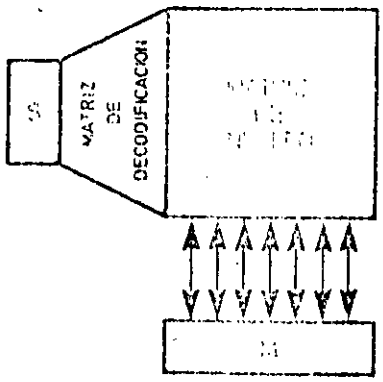
La información almacenada en las memorias no volátiles se conserva aun cuando se desconecta la fuente de alimentación de la unidad de memoria.

• *Memorias estáticas o dinámicas:* La información almacenada en una memoria estática permanece inalterable mientras no se modifique por actuación externa.

La información almacenada en una memoria dinámica sufre una degradación con el tiempo, de tal forma que llega a desaparecer al cabo de un intervalo más o menos prolongado. Para evitar esta pérdida de información deben enviarse periódicamente unos pulsos denominados «de refresco» que reposicionan la información almacenada.



Si por el centro de un núcleo de ferrita se hace pasar una corriente eléctrica de valor I, éste se magnetiza en una dirección (en el sentido de las agujas del reloj, en este caso). Si el valor de la corriente es – I, la dirección de magnetización es la contraria.



Para acceder a una determinada posición de memoria de una matriz de núcleos de ferrita se lleva la dirección de la posición a un circuito de decodificación que seleccionará el octeto buscado. El contenido de este octeto se deposita en un registro denominado M en la figura.

Glosario

¿Qué es una memoria tampón?

Es una memoria de baja capacidad, utilizada, casi exclusivamente, como registro de almacenamiento temporal de información binaria. Habitualmente actúa como memoria auxiliar en las transferencias de información entre la CPU y las unidades de Entrada/Salida.

¿Qué es la memoria central?

Es la memoria donde se almacenan los programas, los datos y los resultados implicados en la ejecución de un proceso.

¿Qué son las memorias de masa?

Son memorias de acceso directo o aleatorio, de elevada capacidad de almacenamiento. La información almacenada en ellas debe pasar, al comienzo de la ejecución de un determinado proceso, al interior de la memoria central. Ejemplos de memorias de masa son las unidades de cinta, los discos flexibles o las unidades de casete de audio.

¿Qué es una memoria RAM?

Un tipo de memoria central, de acceso aleatorio en la que se pueden efectuar dos tipos de operaciones: de lectura y escritura. En ausencia de alimentación la información almacenada en ella se «volatiliza» o desaparece.

¿Qué es una memoria ROM?

ROM son las siglas inglesas de «Read Only Memory», que quiere decir, memoria sólo de lectura. Las memorias ROM son memorias centrales en las que la información se graba durante el proceso de fabricación, y queda permanentemente almacenada en ella. También se llama memoria muerta.

¿Qué es una memoria de burbujas magnéticas?

Es un tipo de memoria RAM en el que la información binaria se almacena en forma de presencia o ausencia de dominios de magnetización en un medio magnético.

Los dominios magnéticos o burbujas se desplazan sobre el plano de una superficie magnética excitados por determinados campos magnéticos.

Su velocidad de trabajo es, hoy por hoy, sustancialmente inferior a la de las memorias de tecnología MOS. En contrapartida, su capacidad de almacenamiento es muy superior.

DIRECTORIO DE ASISTENTES AL CURSO:
"INTRODUCCION A LA COMPUTACION ELECTRONICA Y PROGRAMACION"
1984

NOMBRE Y DIRECCION	EMPRESA O INSTITUCION
1. MARIA ELENA ALVARADO FUENTES Juan de la Barrera Edif. C-2-11 Depto. 201 U. Gral. José Ma. Morelos y Pavón Cuautitlán Izcalli Edo. de México 588 38 54	SECRETARIA DE MARINA DIRECCION DE COMUNICACIONES NAVALES Revillagigedo 11-4° piso Centro Deleg. Cuauhtémoc México, D.F. 521 39 73 y 512 39 73
2. EMMANUEL ALVAREA RAMIREZ Arista 166 Col. Carrera Deleg. Gustavo A. Madero 07070 México, D.F.	SUBDIRECCION DE INFORMACION Y COORDINACION Insurgentes Sur 30 Col. Juárez Deleg. Cuauhtémoc 06600 México, D.F. 566 00 15
3. OCTAVIO ARIAS LUNA Galeana 21 Col. B. Xaltocan Xochimilco 16090 México, D.G. 676 88 76	SUBDIRECCION GENERAL DE OBRAS MARITIMAS Insurgentes Sur 669 México, D.F. 687 53 68
4. DANIEL BECERRIL ALBARRAN Hda. de la Condesa 85 Col. Prados del Rosario Deleg. Azcapotzalco 02410 México, D.F. 382 72 14	CIA. DE LUZ Y FUERZA DEL CENTRO, S.A. Melchor Ocampo 171 Col. Anáhuac Deleg. Miguel Hidalgo 11769 México, D.F. 592 09 57
5. JAIME CEBALLOS HUERTA Paseo de Antioquía 50-B-2 Col. Lomas Estrella Deleg. Iztapalapa 09890 México, D.F.	ACEITES POLIMERIZADOS, S.A. DE C.V. Inguarán 5719 Col. Aragón Inguarán 07000 México, D.F. 781 90 33
6. NEIZAHUALCOYOTL CELIS PINEDA Albuferas 5 Col. Aguilas Deleg. Alvaro Obregón 01710 México, D.F. 593 75 98	GRUPO I C A Minería 145 Col. Escandón México, D.F. 516 04 60

7. LUIS SALVADOR CERVERA AMBRIZ
Nicolás San Juan 223
Col. Del Valle
Deleg. Benito Juárez
03100 México, D.F.
543 41 55

CIA. DE LUZ Y FUERZA DEL CENTRO, S.A.
Melchor Ocampo 171-400
Col. Tlaxpana
Deleg. Cuauhtémoc
México, D.F.
566 42 80

8. JESUS CORTES RODRIGUEZ
Marina Nacional 200 Edif. 15-34
Deleg. Miguel Hidalgo
México, D.F.
271 31 30

SRIA. DE DESARROLLO URBANO Y ECOLOGIA
Belén de las Flores.
Deleg. Alvaro Obregón
México, D.F.
516 31 88

9. CARLOS CROWLEY PEREZ
Antares 51
Col. Prados de Coyoacán
04810 México, D.F.
677 70 94

CIA. DE LUZ Y FUERZA DEL CENTRO, S.A.
Melchor Ocampo 171-4° piso Of. 400
Col. Tlaxpana
Deleg. Cuauhtémoc
592 09 12

10 FERNANDO CHAVEZ GIRON
Artes 30
Col. Estanzuela
Deleg. Gustavo A. Madero
México, D.F.
781 12 05

SRIA. DE COMUNICACIONES Y TRANSPORTES
Doctor Vertiz 1243
Col. Vertiz Narvarte
Deleg. Benito Juárez
México, D.F.
575 05 75

11. ALBERTO DAMIAN GARCIA
Rubén M. Campos 2712-102
Col. Villa de Cortes
Deleg. Benito Juárez
03530 México, D.F.
579 72 12

SRIA. DE COMUNICACIONES Y TRANSPORTES
Xola y Universidad
Col. Narvarte
Deleg. Benito Juárez
México, D.F.
519 07 90

12. EDUARDO DEL CORRAL GRAJALES
Calle Avila Camacho 19
Col. El Hizachal
53840 Edo. de México

S A R H
Insurgentes Sur 30-1er piso
Col. Juárez
06600 México, D.F.
566 00 15

13. ALFONSO CARLOS ESPINOSA MAYA
Cto. Río San Pedro 58-A
Fracc. Real del Moral
Deleg. Iztapalapa
09010 México, D.F.
222 10 63

ENEP ARAGON, UNAM.
Rancho Seco s/n
Sn. Juan de Aragón
Cd. Nezahualcoyotl
Edo. de México
796 04 88

14. ANDRES B. GARCIA HERNANDEZ
2a. Cda. de Alberto Salinas
Mz. 3 - Lte. 23
Col. Aviación Civil
Deleg. Venustiano Carranza
México, D.F.
558 52 16

S C T
Xola y Universidad
Col. Narvarte
Deleg. Benito Juárez
México, D.F.
590 93 52

15. LUIS GARCIA SANCHEZ
Cancún Mz. 284 Lte. 3
Col. Padierna
Deleg. Tlalpan
México, D.F.

BUFETE DE PROYECTOS, S.A. DE C.V.
Francisco Rojas González 142
Col. Prado Hermita
Deleg. Benito Juárez
03590 México, D.F.
539 08 67 y 672 73 85

16. HEBERTO GONZALEZ KARG
Eliot 12-302
Col. Polanco
Deleg. Miguel Hidalgo
México, D.F.
254 31 87

CASA DE BOLSA DE TRABAJO
Paseo de la Reforma 392
Col. Juárez
24 00 56

17. ALFREDO LEONARDO HURTADO MELGOZA
6 de Diciembre No. 1767
Sector Hidalgo
44260 Guadalajara, Jal.
21 64 58

CENTRO DE ENSEÑANZA TECNICA INDUSTRIAL
El Chaco 3223
Col. Providencia
Guadalajara, Jal.
41 32 24

18. VICTOR LOPEZ MENDIZABAL
Ing. José J. Reinoso 68
Col. Constitución de 1917
Deleg. Iztapalapa
09260 México, D.F.
691 37 79

S A R H
Insurgentes Sur 30
Col. Juárez
Deleg. Cuauhtémoc
México, D.F.
591 18 35

19. MARIO LUNA SAAVEDRA
Beethoven 46-3
Col. Peralvillo
Deleg. Cuauhtémoc
06220 México, D.F.
597 74 80

TELEFONOS DE MEXICO, S.A.
Río Sena 49-8° piso
Cuauhtémoc
06500 México, D.F.
525 15 30 ext. 110

20. SABINO MARQUEZ MONTERO
J.J. Coronado 2
Col. Del Maestro
Veracruz, Ver.
7 34 57

PROMOTORA DE LA VIVIENDA VERACRUZANA
Facultad de Ingeniería
Hidalgo 1104
Centro
Veracruz, Ver.
7 24 04

21. EDUARDO MARTINEZ MARTINEZ
Zempoaltecas y Tlahuicas Edif. 4-2
Col. Las Trancas
Deleg. Azcapotzalco
02450 México, D.F.

D.G.C.O.H. SUBDIRECCION DE INFORMATICA
San Antonio Abad 213
Col. Obrera
Deleg. Cuauhtémoc
06800 México, D.F.
588 32 27

22. ROBERTO MERINO CARRION
1a. Cerrada de Zaragoza No. 5 Altos
Cuauhtémoc
06300 México, D.F.
546 86 03

S A R H
Dirección Gral. de Control de Ríos
Insurgentes Sur 30-1er piso
Col. Juárez
Deleg. Cuauhtémoc
06600 México, D.F.
566 00 15

23. J. MIGUEL MONTOYA R.

S.C.T.
Dirección Gral. de Obras Marítimas

24. AGUSTIN MORON REYES
Norte 60 No. 3823
Col. Río Blanco
Deleg. Gustavo A. Madero
México, D.F.
537 83 34

DIRECCION GRAL. DE OBRAS MARITIMAS
Insurgentes Sur 664-9° piso
México, D.F.
687 53 68

25. LEOBARDO PACHECO LOPEZ

S.C.T.

26. EFREN PEREZ BARRERA
Av. I.P.N. 1618 Andador D-25
Col. Magdalena de las Salinas
Deleg. Gustavo A. Madero
07760 México, D.F.
586 55 77

INSTITUTO MEXICANO DEL SEGURO SOCIAL
Eje Vial Fortuna s/n
Col. Magdalena de las Salinas
Deleg. Gustavo A. Madero
07760 México, D.F.
754 69 22 exts. 2359 y 2360

27. EVELIA PEREZ RAMOS
Xavier Sorondo 260
Col. Villa de Cortes
Deleg. Benito Juárez
03530 México, D.F.
590 54 45

S C T
Av. Xola y Universidad
Col. Narvarte
Deleg. Benito Juárez
México, D.F.
519 51 34

28. JOSE RAUL RAMIREZ SALAZAR
Oriente 12 No. 177
Col. Reforma
Cd. Nezahualcoyotl
91597 524 37

DIRECCION GRAL. DE OBRAS MARITIMAS
Insurgentes Sur 664
México, D.F.
687 53 68

29. WALTER RAKOWSKY PRIBIK
Calle B Edif. 25 No. 13
Col. U.A.P.R.S.
Deleg. Coyoacán
04800 México, D.F.
684 77 86

UNIVERSIDAD AUTONOMA METROPOLITANA
La Purísima y Michoacán
Iztapalapa
México, D.F.
686 03 02

30. SHARYN IVONNE RODRIGUEZ MONROY
Gral. José Santos Pte. No. 5
Col. Héroes de la Revolución
C.P. 53840
589 13 85

INSTITUTO POLITECNICO NACIONAL
Av. de Los Maestros 217
Sto. Tomás
547 87 76

31. ANTONIA RODRIGUEZ RODRIGUEZ
Acolhuacán 95
Col. Arenal 3a. Sección
Deleg. Venustiano Carranza
15600 México, D.F.

SUBSECRETARIA DE INFRAESTRUCTURA
Providencia 807
Col. Del Valle
México, D.F.
523 28 15

32. FERNANDO RODRIGUEZ SALAZAR
Calle 32 No. 19
Col. Independencia
Naucalpan México
C.P. 53830
294 40 19

PAPELERA ATLAS, S.A.
Av. 16 de Septiembre No. 25
Col. El Prieto
Deleg. Naucalpan
53370 Edo. de México
576 59 00

33. BEATRIZ NATALIA ROMO MOHLER
Av. del Taller
Retorno 48 U.6 Dpto. 6235
Col. Jardín Balbuena
Deleg. Venustiano Carranza
15900 México, D.F.
762 06 81

S.C.T.
Av. Universidad y Xola
Basamento del I.C.S.C.O.P.
Col. Narvarte
03028 México, D.F.
519 92 21

34. ANTONIO RUEDE ALVAREZ
Muitles No. 8
San Mateo Tlaltenango
Deleg. Cuajimalpa
05600 México, D.F.
812 28 13

TELEFONOS DE MEXICO, S.A. DE C.V.
Río Sena 49-8° piso
Col. Cuauhtémoc
06500 México, D.F.
525 37 85

35. APOLINAR SOTELO FLORES

S.C.T.
Dirección Gral. de Obras Marítimas

36. MARTIN SANCHEZ MURILLO
Francisco Ramírez 79
Col. Daniel Garza
Deleg. Miguel Hidalgo
11830 México, D.F.
277 18 13

37. LUIS SANCHEZ OLGUIN
Bismark 35
Col. Moderna
Deleg. Benito Juárez
03510 México, D.F.
590 51 38

CABLEVISION, S.A.
Av. Col. Del Valle 620
Col. Del Valle
Deleg. Benito Juárez
México, D.F.
687 91 55

38. MIGUEL SERRANO SALDAÑA
Calle 25 No. 116
Col. Edo. de México
México, D.F.
590 93 52

S.C.T.
Av. Fernando 247
Col. Alamos
Deleg. Benito Juárez
México, D.F.
590 93 92

39. RODOLFO URIBE REYNOSO
Callejón Lázaro Cárdenas 95
Col. El Chamizal
Naucalpan, Edo. de México
358 06 65

DIRECCION GRAL. DE CONSTRUCCION Y OPERACION HIDRAULICA
San Antonio Abad 231
Col. Obrera
Deleg. Cuauhtémoc
México, D.F.
588 32 27

40. ORLANDO ULISES ZAMUDIO PARTIDA
Ote. 178 No. 1513
Col. Héroes de Churubusco
Deleg. Iztapalapa
09090 México, D.F.
82 50 36

41. VICTOR MANUEL ZAPATA ROSALES
Ciclamen 69
Bo. Xaltocan
16090 Xochimilco
676 27 84

S.C.T.
Dirección Gral. de Obras Marítimas
Av. Insurgentes 664
México, D.F.