Víctor Sánchez

COMPUTACION
lenguajes y programación

COMPUTACION: LENGUAJES Y PROGRAMACION

principales conceptos básicos de computación; 2) Lenguajes y programación, en el que se hace especial hincapié en el uso del diagrama de flujo; 3) FORTRAN IV , en el que se exponen con numerosos ejemplos las distintas proposiciones de dicho lenguaje; 4) Lenguajes de ensamble, que permite conocer en detalle un lenguaje orientado a un computador, y 5) Sistemas de Operación, que da una visión general de los objetivos y estructura de ellos, como asimismo de los principales conceptos utilizados.

Julio Ortúzar
Jefe de Servicios de Computación
CELADE

PREFACIO

El Centro Latinoamericano de Demografía (CELADE) , como usuario del material estadístico demográfico, ha comprobado que en las oficinas productoras de datos estadísticos de muchos países de la región se evidencia una gran escasez de recursos humanos especializados en Procesamiento Electrónico de Datos (PED) , y en algunos casos simplemente no existe. Paralelamente se ha observado que hay una alta tasa de deserción de este personal hacia otras organizaciones, en especial a la empresa privada. Estas circunstancias crean serios problemas a dichas oficinas, ya que el PED y sus especialistas son cada vez más necesarios para proporcionar en forma precisa y oportuna la información requerida por los organismos de planificación tanto económica como social.

Desde hace unos seis años, CELADE ha venido promoviendo la realización de seminarios y cursos orientados a capacitar personal en PED en el área de estadística y ciencias sociales o participando en ellos. Por ejemplo, se pueden citar dos seminarios sobre tabulaciones censales (Santiago de Chile, 1970 y San José de Costa Rica, 1971) organizados juntamente con la Oficina del Censo de los Estados Unidos; también un curso de programación organizado con la colaboración del Centro Interamericano de Enseñanza de Estadística (CIENES) , (Santiago de Chile, 1973). Sin embargo, estos esfuerzos aislados demostraron ser insuficientes para la adecuada difusión y evaluación de técnicas avanzadas en el Procesamiento Electrónico de Datos estadísticos en materia de población.

Por las consideraciones anteriores, CELADE se propuso iniciar un curso de PED orientado especialmente al personal que trabaja en las Oficinas de Estadísticas de los países de la región, el cual, en lo posible, se impartiría en forma periódica una vez por año.

El primero de estos cursos se inició en julio de 1975 y tuvo una duración de cuatro meses. Abarcó un vasto programa docente que incluyó desde materias básicas de PED , hasta conferencias sobre tópicos altamente especializados. En su desarrollo, los participantes abordaron el estudio de temas como la validación y corrección automática de datos, técnicas y medios para el procesamiento de encuestas y censos, además de examinar en detalle conceptos y métodos de uso de lenguajes como FORTRAN y ensamblador, uso y explotación de sistemas de operación, teleprocesamiento, etc., haciendo el mayor uso de computador posible en cada uno de los temas.

El presente libro, que agrupa parte del material entregado en el primer curso de procesamiento electrónico de datos aplicado a ciencias sociales y que corresponde a aquellos temas expuestos en él por el profesor Víctor Sánchez C., pretende ser una guía y manual de consulta para el estudiante de computación. Está compuesto por cinco capítulos que son: 1) Introducción a computación, donde se dan a conocer los

Para editar el presente libro se ha contado con el respaldo, colaboración o esfuerzo de un gran número de personas del Centro Latinoamericano de Demografía (CELADE). A riesgo de omitir a algunas de ellas, deseo expresar mi sincero agradecimiento a la Directora de CELADE , señorita Carmen A. Miró, quien permitió e impulsó la ejecución de esta tarea, consciente de la importancia que tiene en el campo de la investigación el uso de técnicas y recursos modernos; al Jefe del Area de Procesamiento de Datos e Información, señor Arthur M. Conning; al Jefe del Servicio de Computación, señor Julio Ortúzar, y a los señores Abel Packer, Pedro Sust y Nelson Piro, quienes tuvieron la infinita paciencia de revisar los distintos capítulos que componen el libro y hacer sugerencias o críticas a su contenido. Deseo agradecer al personal que labora en los Servicios Editoriales y de Publicaciones la dedicación que mostraron para solucionar todos aquellos problemas que estaban fuera del área de trabajo del autor. También deseo expresar mi reconocimiento a la secretaria del Servicio de Computación, señorita Raquel Vicuña, que puso especial interés en comprender y reproducir símbolos y términos propios de computación, no siempre fáciles y asequibles.

El autor

CENTRO LATINOAMERICANO DE DEMOGRAFIA
CELADE: J.M. Infante 9. Casilla 91. Teléfono 257806
Santiago (Chile)

CELADE: Ciudad Universitaria Rodrigo Facio
Apartado Postal 5249
San José (Costa Rica)

Serie E, Nº 23

VICTOR SANCHEZ C.

COMPUTACION

LENGUAJES *y* PROGRAMACION

CENTRO LATINOAMERICANO DE DEMOGRAFIA

Santiago de Chile, 1976

INDICE

CONCEPTOS BASICOS DE COMPUTACION

LENGUAJES Y PROGRAMACION

FORTRAN IV

LENGUAJE DE ENSAMBLE

INTRODUCCION A SISTEMAS DE OPERACION

CONCEPTOS BASICOS DE COMPUTACION

1. QUE ES UN COMPUTADOR

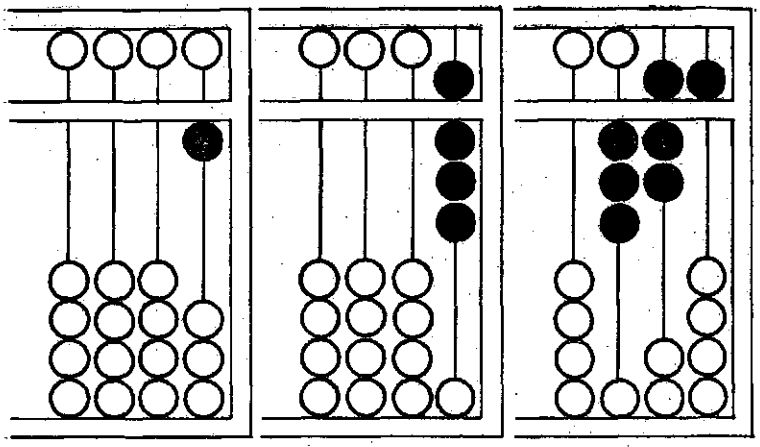
A. Desarrollo histórico

Hace unos 10 000 años, el cambio de clima producido en la tierra hizo que el hombre primitivo, que hasta entonces había tenido una vida nómada, se radicara en los valles del Nilo, Tigris y Eufrates para dedicarse a la agricultura, lo que le significó tener que resolver un tipo de problemas distintos a los que había tenido hasta entonces, como, por ejemplo, tomar en cuenta los días, las temporadas de lluvia o de sol, la cantidad y tipos de semillas, las tierras que debía sembrar, etc. La resolución de estos problemas trajo como consecuencia la necesidad de contar y se cree que inicialmente utilizaron los dedos de las manos y de los pies, como aún lo hacen algunas tribus de Nueva Guinea, y después, granos de maíz, piedrecillas, etc.

Cada vez perfeccionaron más el manejo de las piedrecillas, hasta lograr, hace unos 5 000 años, diseñar un tablero de arcilla con muescas en las que se podían colocar aquéllas, cuyo movimiento a través del tablero permitía realizar operaciones aritméticas elementales.

Posteriormente, cuando se inició el comercio, se presentó la necesidad de registrar las transacciones y es probable que esto se hiciera mediante marcas en árboles o en rocas, método que fue evolucionando en la medida en que se iban tornando más complejas las operaciones hasta lograr los registros en tablillas de barro hechas por los sumerios durante el período de 3 700 a 3 000 años A.C.

Aproximadamente 2 600 años A.C., los chinos inventaron el *ábaco*, al mismo tiempo que los japoneses inventaban un aparato parecido llamado *soroban* que, como el ábaco, funcionaba con una técnica similar a la del tablero de arcilla.



REPRESENTACION EN UN ABACO DE LAS CANTIDADES 1,8 y 375

Pasaron varios siglos antes de que hubiera un nuevo avance significativo y éste fue el desarrollo de los *logaritmos* por el escocés John Napier, en 1617. Mediante los logaritmos, las operaciones de multiplicación y división se realizan efectuando sumas y restas respectivamente. El mismo John Napier inventó un dispositivo conocido como "rodillos de Napier", que consiste en un conjunto mecánico de láminas de hueso, cada una con los *dígitos 1 al 9* y éstos con sus múltiplos en columna debajo de ellos. Con este dispositivo se podían efectuar multiplicaciones directas.

Utilizando el mismo concepto de los logaritmos, William Oughtred inventó en 1621 la *regla de cálculo*, de enorme utilidad en ingeniería, donde muy a menudo se necesita obtener o verificar la magnitud de algunos cómputos cuyo grado de precisión no interesa mayormente.

Fue en 1642 en Francia, donde Blaise Pascal inventó la primera *máquina sumadora*, que consistía en hileras de ruedecillas con dientes numerados de 0 a 9. Frente a cada hilera había una ventanilla en la que aparecía el número correspondiente a la cantidad de vueltas completas que había hecho cada ruedecilla. Lo importante es que al completar 10 vueltas una de las ruedecillas hacía girar la siguiente en un décimo de vuelta (acarreo de uno a la posición superior). El funcionamiento es exactamente el mismo que tienen los actuales medidores y cuenta-kilómetros.

Casi 30 años más tarde, en 1671, el filósofo y matemático alemán Gottfried von Leibnitz perfeccionó la máquina sumadora de Pascal,

logrando construir en 1674 la primera máquina calculadora, con la que podían ejecutarse las cuatro operaciones aritméticas.

Hacia 1725, Basile Bouchon diseñó un telar que operaba mediante papel perforado. En 1728, el ingeniero francés M. Falcon diseñó uno que operaba por medio de tarjetas perforadas colocadas en forma de cadena sin fin. Sólo las agujas que coincidían con los agujeros podían penetrar y sus hilos formaban el diseño.

En 1801, Joseph Marie Jacquard obtuvo un *telar automático* operando con tarjetas perforadas.

En 1812, Charles Babbage, profesor de matemáticas en la Universidad de Cambridge, diseñó una máquina llamada *máquina de diferencias* destinada a calcular e imprimir tablas matemáticas. El método se basaba en el hecho de que un polinomio de grado n tiene su diferencia n-ésima constante. Charles Babbage continuó con sus proyectos y en 1820 diseñó la *máquina analítica*, capaz de realizar una secuencia determinada de cálculos y que tenía la habilidad de almacenar números, imprimir resultados y repetir ciclos de operaciones durante la computación. Lamentablemente, no se logró materializar el invento porque los principios enunciados sobrepasaban las posibilidades técnicas de esa época. Esos principios correspondían en gran parte a los contenidos en el concepto de computador.

En 1850, D.D. Parmalee patentó en los Estados Unidos la *primera sumadora impulsada por teclas*, que permitía realizar las sumas operando sobre una columna de dígitos cada vez.

En 1872, Frank Stephen Baldwin inventó la primera *calculadora reversible*, en los Estados Unidos y pocos años más tarde, en 1878, W.F. Odhner diseñó, en Rusia, una máquina similar.

En 1880, el Dr. Herman Hollerith, estadístico que trabajaba en la Oficina del Censo de los Estados Unidos, empezó a diseñar un sistema mecánico que le permitiera registrar, calcular y tabular datos de los censos. El sistema consistió, finalmente, en registrar los datos en tarjetas mediante una perforadora operada manualmente. Con un dispositivo mecánico que cubría la superficie de la tarjeta, se lograba establecer circuitos eléctricos cerrados cuando una serie de clavijas pasaban a través de las perforaciones y tomaban contacto con recipientes llenos de mercurio, lo que hacía que la información fuera registrada en discos contadores.

El Dr. Hollerith organizó en 1896 la Tabulating Machine Company, la que en 1911 se fusionó con otras dos compañías para formar la Computing Tabulating Recording que pasaría a ser, en 1924, la International Business Machines Corporation (IBM Corporation).

Durante bastante tiempo se trabajó con las máquinas de tarjetas perforadas o máquinas de registro unitario (Unit Record), existiendo, por lo tanto, la limitación de velocidad de procesamiento y, además, la derivada del hecho de que cada máquina realiza funciones independien-

tes de las que realizan las restantes máquinas que componen un sistema.

Esas limitaciones llevaron al profesor Howard Aiken, de la Universidad de Harvard, a trabajar conjuntamente con ingenieros de la IBM para lograr una sola máquina que realizara todas las funciones que anteriormente aparecían desligadas entre sí, con lo que se obtuvo, en 1944, el MARK I, máquina electromecánica con 72 acumuladores de suma y 60 juegos de interruptores para fijar constantes. Las instrucciones se daban por medio de interruptores, tableros de alambre y cinta perforada. Después de un siglo, el sueño de Charles Babbage se había hecho realidad.

La primera máquina que utilizó tubos electrónicos para calcular fue la ENIAC (Electronic Numerical Integrator and Computer), diseñada por J. Presper Eckert y el Dr. John W. Mauchly de la Moore School of Engineering de la Universidad de Pensylvania, entre 1942 y 1946. Las instrucciones de la máquina se programaban en paneles de control, intercambiables, en tarjetas o en cintas de papel perforadas. La limitación de esta máquina radicaba en la poca capacidad de almacenamiento y en el hecho de tener que dar las instrucciones a medida que se iba avanzando en el trabajo.

En mayo de 1949, en Inglaterra, en la Universidad de Cambridge, se dio a conocer el EDSAC (Electronic Delayed Storage Automatic Computer), primer *computador de programa almacenado.*

En abril de 1951, entró en funcionamiento en la Oficina del Censo el UNIVAC (Universal Automatic Computer), desarrollado por la Eckert Mauchly Computer Company, fundada en 1946 por los diseñadores del ENIAC y adquirida en 1949 por la Remington Rand. Las características de este computador lo hicieron atrayente para empresas comerciales y fue así como en 1954 se realizó la primera instalación comercial en la General Electric Appliance Park en Louisville, Kentucky.

En 1952 fue terminado el EDVAC (Electronic Discrete Variable Automatic Computer) diseñado también por J. Presper Eckert y el Dr. John W. Mauchly, para el ejército de los Estados Unidos.

IBM instaló su primer computador en 1953: el IBM 701 y en 1954 el IBM 650, que utilizaba como forma de memoria un tambor magnético, elemento adicional a la memoria de tubos de rayos catódicos. Estos últimos constituyen la característica de la *primera generación de computadores.*

Mientras tanto, se gestaba un cambio radical en la construcción de computadores: el reemplazo de los tubos por transistores, elementos mucho más pequeños, más baratos y que casi no generan calor. El hecho, además de que requieren muy poca energía, los hizo revolucionar la tecnología.

El cambio de tubos a transistores se hizo primeramente en los computadores militares en 1956 y posteriormente, en 1958, en los com-

putadores comerciales, constituyéndose así la *segunda generación de computadores.*

Se puede decir que el primer computador transistorizado fue el ISI-609 construido por la Information System Inc., en 1958. Otros posteriores son el Solid State 80, el Philco 2000 y los IBM 7000, 1600 y 1400. De estos últimos, los más conocidos han sido el IBM 1620 y 1401.

En 1963, IBM anunció el Sistema IBM/360, en el que se introduce un nuevo cambio tecnológico importante: la tecnología de la lógica en estado sólido para producir un paquete miniatura y la creación de circuitos integrados de material semiconductor. La primera, monta transistores delgados y pequeños diodos hechos de silicón en una base de cerámica y los conecta con un circuito impreso. A continuación, el módulo completo se monta en plástico. Este cambio es la característica de la *tercera generación de computadores.*

En el Sistema /370 de IBM se han incorporado nuevamente avances tecnológicos: la tecnología del sistema monolítico y el almacenamiento monolítico. Si anteriormente un pedacito de silicón del tamaño de una cabeza de alfiler contenía normalmente un tipo de componente (transistor o diodo) y se necesitaban varios pedacitos y resistencias de película delgada para formar un circuito, ahora un pedacito de silicón contiene sobre 100 componentes y se pueden formar en él hasta ocho circuitos interconectados. Esto permite obtener mayor confiabilidad por la disminución del número de conexiones externas; mayor velocidad a causa de la disminución del recorrido entre circuitos; menor espacio inherente a la mayor densidad de componentes.

Además, se ha cambiado el almacenamiento de núcleos de ferrita por almacenamiento monolítico, el cual utiliza los mismos conceptos que la lógica monolítica y permite obtener, por lo tanto, las mismas ventajas que aquél, esto es, velocidad, confiabilidad, menores requerimientos de espacio, etc.

B. Sistemas de Procesamiento de Datos

a) *Definición de conceptos*

Procesamiento de datos: se puede definir como una serie de acciones planificadas y operaciones sobre los datos para lograr un resultado deseado.

Los procedimientos y dispositivos usados constituyen un *Sistema de Procesamiento de Datos.* Estos pueden variar en tamaño, complejidad, velocidad, costo o aplicación. Sin embargo, independientemente de los datos que se van a procesar o de los elementos o dispositivos que se van a utilizar, todo procesamiento de datos involucra tres consideraciones básicas:

Los datos originales o *entrada* (input) al sistema
El *proceso* planificado en el sistema
El resultado final o *salida* (output) del sistema.

Es conveniente definir lo que se entiende por *dato* como asimismo lo que se considera *información*. Dato es cualquier señal, palabra cifra, símbolo, etc. que represente una idea, objeto, condición o situación. No es tan fácil definir lo que es información. Al menos, habrá varias consideraciones previas antes de intentar establecer el ámbito que interesa de la palabra. Primero es necesario plantear la interrogante: ¿Es un dato información? La respuesta debe ser no aunque un dato contiene potencialmente información, ésta no será útil hasta que no se establezca una relación de ese dato con otro u otros datos. Por ejemplo:

HISTORIA - ROBLES - 1950

cada uno de estos datos en sí no aporta información útil. Se puede especular bastante para poder extraer alguna utilidad de ellos y con toda seguridad que eso se logra, pues habrá personas que no saben que historia se escribe con h y robles con b, pero ¿es esa la información que le interesaba transmitir a quien escribió esos datos? No se sabe.

En cambio, si esos datos aparecen insertos en una tabla como la siguiente:

Clasificación de libros

Materia	*Autor*	*Año*
Historia	*Robles*	*1950*

se entrega la información que se esperaba proporcionar. Nuevamente, sin embargo, el conocimiento que se ha obtenido al leer la tabla puede ser de utilidad para una o más personas y no representar ningún beneficio o, lo que es más, carecer totalmente de significación para otras personas.

De aquí también se puede determinar que cada dato lleva implícitas dos nociones: la de identificación de un concepto (materia, autor, año) y el valor particular del dato (Historia, Robles, 1950) o, lo que es lo mismo, lleva implícitas las ideas de función y argumento.

Ejemplo:

Inventario de CELADE Santiago-Chile
al 31/12/74

Código	Descripción	Valor $	Fecha de Ingreso
AB01	Escritorio	10,35	01/04/73

Los datos propiamente tales son: AB01-Escritorio-10,35-01/04/73 y se han seleccionado y organizado de acuerdo con:

Usuario	CELADE
Problema	Inventario
Tiempo	31/12/74
Lugar	Santiago-Chile
Función–1	Código
Argumento–1	AB01
Función–2	Descripción
Argumento–2	Escritorio
Función–3	Valor ($)
Argumento–3	10,35
Función–4	Fecha Ingreso
Argumento–4	01/04/73

La organización, según argumento, puede ser, por ejemplo:

Clasificación en orden ascendente de los códigos

Clasificación en orden alfabético

Clasificación por fecha de ingreso

Clasificación por fecha de ingreso y en orden alfabético, etc.

El conjunto de datos organizados como el del ejemplo muestra la posibilidad de establecer relaciones entre algunas columnas de ellos. Todas estas relaciones constituyen un primer nivel de información. Las que se efectúen entre líneas o grupos de líneas formarán niveles más bajos.

b) *Sistema manual*

Un sistema manual consistiría en lápices, lapiceras, libros, formularios, carpetas, tarjetones, archivadores y otros elementos manuales que permitan registrar datos, elaborarlos y presentar los resultados. Es evidente que el tiempo que transcurre desde la recepción de los datos hasta la entrega de resultados irá aumentando de acuerdo con el volumen de aquéllos y a la mayor complejidad que presenten los cálculos que deban efectuarse.

c) *Sistema mecánico*

Dadas las características del sistema manual, especialmente en lo que se refiere a demora en la entrega de resultados, lo normal es que se trate de optimizar algunas etapas de los procesos y para ello es necesario recurrir a equipos mecánicos como son: máquinas de escribir, de sumar, de calcular, de contabilidad, etc. Esto trae como consecuencia la implantación de un sistema mecánico de procesamiento.

Cuando el volumen de información es demasiado grande y, además, es necesario dejar registrados los datos para futuras nuevas utilizaciones o reprocesos, se recurre a los equipos de registro unitario que operan sobre tarjetas perforadas. Se tiene así:

i) *Máquina perforadora.* Permite registrar información numérica, alfabética y caracteres especiales mediante perforaciones hechas en tarjetas. Para ello se cuenta con teclados especiales.

Una ligera pulsación de la tecla en el teclado determina el accionamiento eléctrico de uno o varios punzones que atraviesan la tarjeta, produciéndose una, dos o tres perforaciones por columna, según haya sido la tecla oprimida.

Las perforaciones permiten que otras máquinas procesen los datos registrados en la tarjeta mediante dispositivos especiales que permiten interpretar las perforaciones.

ii) *Máquina verificadora.* Permite corroborar que los datos hayan sido registrados correctamente en la perforadora, a la cual es similar, excepto que en vez de perforar, detecta si la información que contiene la columna de la tarjeta corresponde a la tecla oprimida. Cualquier diferencia que exista hace que la máquina se detenga y emita una señal. Es posible intentar hasta tres verificaciones en la columna; si al término de ellas la diferencia se mantiene, la máquina hace una muesca en el borde superior de la tarjeta sobre la columna errónea. Si la tarjeta es verificada sin encontrar ninguna columna con error, la máquina hace una muesca en el borde derecho de la tarjeta.

iii) *Máquina clasificadora.* Permite acomodar las tarjetas en el orden que se desee, que puede ser numérico, alfabético o de caracteres especiales. También es posible revisar automáticamente las tarjetas para saber si su perforación está de acuerdo con el tipo elegido, mientras se realiza la operación de clasificación.

iv) *Máquina intercaladora.* Las operaciones básicas que pueden hacerse en esta máquina son las siguientes: seleccionar tarjetas específicas de un archivo, es decir, aquellas que cumplen una cierta condición; combinar dos archivos para formar uno solo, ya sea con o sin selección de tarjetas; verificar el orden o serie de las tarjetas de un archivo y, finalmente, comparar dos archivos de tarjetas y seleccionar de ambos aquellas tarjetas que no tengan compañeras en el otro.

v) *Máquina interpretadora.* Esta máquina lee la información perforada en una tarjeta y la imprime en la misma tarjeta.

vi) *Máquina reproductora.* Permite copiar en otra tarjeta toda la información contenida en una tarjeta, o parte de esa información. También es posible repetir esa información, o parte de ella, en un conjunto de tarjetas que le sigan. La primera función se denomina reproducción y la segunda multiperforación (gung-punch).

vii) *Máquina calculadora.* La calculadora lee factores perforados en tarjetas, efectúa cálculos utilizando dichos factores y perfora los resultados ya sea en la tarjeta de donde fueron leídos los datos, o bien en determinadas tarjetas que le sigan. Los cálculos consisten en combinaciones de operaciones aritméticas básicas.

viii) *Máquina de contabilidad.* Esta máquina lee información per-

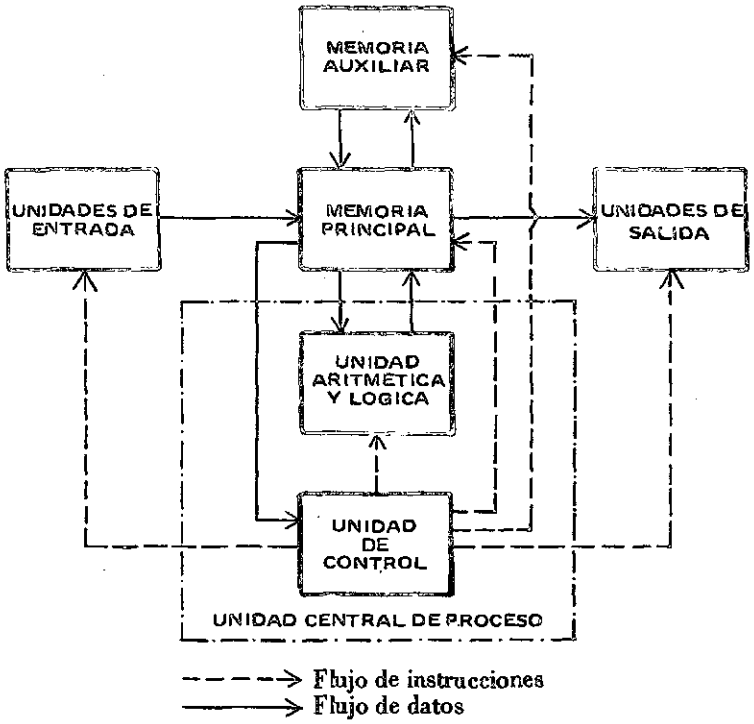
forada en tarjetas, suma, resta, compara, selecciona e imprime resultados. También se puede perforar tarjetas sumarias simultáneamente con la preparación de planillas.

d) *Sistema electrónico*

El Sistema Electrónico de Procesamiento de Datos (EPD) es el que se conoce comúnmente por *computador*, computadora u ordenador.

Está formado por un conjunto de unidades que se puede agrupar según su función en:

Unidades de Entrada
Unidades de almacenamiento (memoria)
Unidad Aritmética y Lógica } Unidad Central de Proceso
Unidad de Control
Unidades de Salida



La transmisión de instrucciones y datos entre unidades se realiza mediante conexiones y circuitos electrónicos.

Las operaciones que puede realizar el computador son de tipo aritmético, esto es, adición, substracción, multiplicación y división y, además, la operación lógica de comparar datos entre sí. La gran ventaja que tiene el computador sobre las calculadoras de escritorio, en lo que se refiere, por ejemplo, a operaciones aritméticas, es que realiza éstas a velocidades electrónicas que se expresan en milisegundos (ms=1/1000seg) microsegundos (us=1/1000000 seg.) y nanosegundos (ns=1/1000000000 seg.)

En la Unidad de Memoria principal se almacena el conjunto de instrucciones (programa) que indicará a la Unidad de Control las operaciones a realizar, la secuencia correcta de ella y los elementos que intervendrán en la operación. Generalmente, una de las primeras instrucciones que se ejecuta es la de almacenar los datos que serán procesados.

Las características mencionadas: conexión entre unidades, velocidades electrónicas de cálculo, ejecución de la totalidad de los pasos correspondientes a un proceso basándose en el programa almacenado, la posibilidad de retener los datos que se operarán, como asimismo los resultados parciales y finales, y además, el hecho de poder "elegir" o "decidir" qué grupo de instrucciones realizará de acuerdo con el resultado de comparaciones entre elementos, hacen que el factor humano prácticamente no intervenga en forma directa, lo que permite disminuir la posibilidad de error por mala interpretación de información escrita, por mal registro de datos u operación defectuosa con ellos, logrando al mismo tiempo mayor precisión y menor demora en la obtención de los resultados.

Si bien es cierto que no se elimina totalmente la posibilidad de registro o lectura de información defectuosa por falla de unidades electro-mecánicas, es muy baja la probabilidad de que ellos ocurran y existe como factor de seguridad una serie de elementos técnicos que verifican, en forma automática, que las operaciones se realicen correctamente y cualquier problema que se presente es detectado y se detiene el proceso.

Los pasos que se realizan en un proceso electrónico de datos son los siguientes:

i) Registro de los datos e instrucciones en un elemento de entrada tal como: cinta de papel perforada, tarjeta perforada, cinta magnética, caracteres en tinta magnética o caracteres ópticos.

ii) Introducción de instrucciones y datos a la memoria a través de un dispositivo de entrada, el cual los convierte en impulsos eléctricos, los que permiten magnetizar, en un sentido u otro, los elementos de almacenamiento.

iii) Las instrucciones son leídas por la unidad de control desde la memoria. Una vez que las interpreta, emite las órdenes a los componentes del sistema que intervienen en la operación.

iv) La lectura, interpretación y ejecución de las instrucciones se

realizan tomando cada una de ellas en forma secuencial. La ejecución implica transferencia de datos desde la memoria a la unidad aritmético-lógica donde se efectúan las operaciones aritméticas o de comparación.

v) Los datos, resultados parciales o resultados finales son enviados a la unidad de almacenamiento para mantenerlos hasta que sean usados nuevamente en procesos posteriores o para ser enviados a dispositivos de salida cuando lo ordene la unidad de control.

vi) La información es enviada desde la unidad de almacenamiento al exterior a través de un dispositivo de salida, el cual la registra en un elemento de salida que puede ser: cinta de papel perforada, tarjeta perforada, cinta magnética o papel impreso. Es posible también obtener información mediante dispositivos de despliegue visual.

Es fácil comprender por qué aumenta cada día el uso de computadores. No hay disciplina, especialidad o área de trabajo en que no pueda aplicarse con enormes ventajas el computador. Con la ayuda de éste se puede abordar y resolver problemas que en otras condiciones hubiera sido imposible solucionar por la cantidad de tiempo que habría significado hacerlo.

Para dar una idea de la amplitud del campo en que pueden utilizarse estas máquinas, se mencionan algunos de los innumerables problemas en que ellas se aplican:

Empresas e Industrias
- Cálculo de sueldos y salarios
- Contabilidad
- Control de inventarios
- Registro de personal
- Facturación
- Control de producción
- Control de proyectos mediante camino crítico
- Aplicación de programación lineal
- Aplicación de la teoría de esperas
- Simulación de sistemas
- Reposición de elementos de máquinas
- Control de procesos
- Sistemas de tiempo real

Universidades
- Control de curriculum de alumnos
- Selección y control de matrícula
- Estadística médica
- Bienestar estudiantil (becas y otros beneficios)
- Educación programada
- Traducción de lenguajes

Hospitales
- Registro de enfermos. Historia clínica
- Diagnósticos
- Estadísticas
  - Mortalidad infantil
  - Historias de embarazos

Otros
- Dirección y control de cohetes
- Dirección de submarinos
- Recuperación de información con palabras claves
- Tabulación de censos
- Encuestas de hogares.

2. ELEMENTOS Y DISPOSITIVOS DE ENTRADA-SALIDA

A. Elementos de Registro de Datos

a) *Tarjeta.*

La tarjeta es el medio de registro más utilizado hasta el momento. A continuación se presentan dos sistemas: el UNIVAC y el IBM Hollerith, de los cuales este último es el de mayor difusión.

i) *Tarjeta* UNIVAC. Es una tarjeta rectangular, de cartulina, que mide 3 1/4 pulgadas de ancho y 7 3/8 pulgadas de largo. A lo ancho está subdividida en dos zonas, cada una con capacidad para cuarenta y cinco columnas, en cada una de las cuales es posible representar un carácter sea éste numérico, alfabético o especial, de tal manera que una tarjeta puede contener hasta noventa caracteres. El registro de los caracteres se obtiene mediante una o más perforaciones circulares distribuidas en las seis posiciones de las columnas.

En la parte superior de cada zona se puede imprimir el contenido de cada columna.

ii) *Tarjeta* IBM. Tiene las mismas dimensiones que la tarjeta UNIVAC. Horizontalmente, está dividida en doce líneas, diez de las cuales tienen numeración impresa partiendo con el número nueve en el borde inferior y siguiendo en orden descendente hasta llegar a cero a medida que se acerca al borde superior. Las dos líneas restantes, que no tienen numeración, corresponden a la línea once, inmediatamente encima de la línea cero, y la doce, cercana al borde superior de la tarjeta.

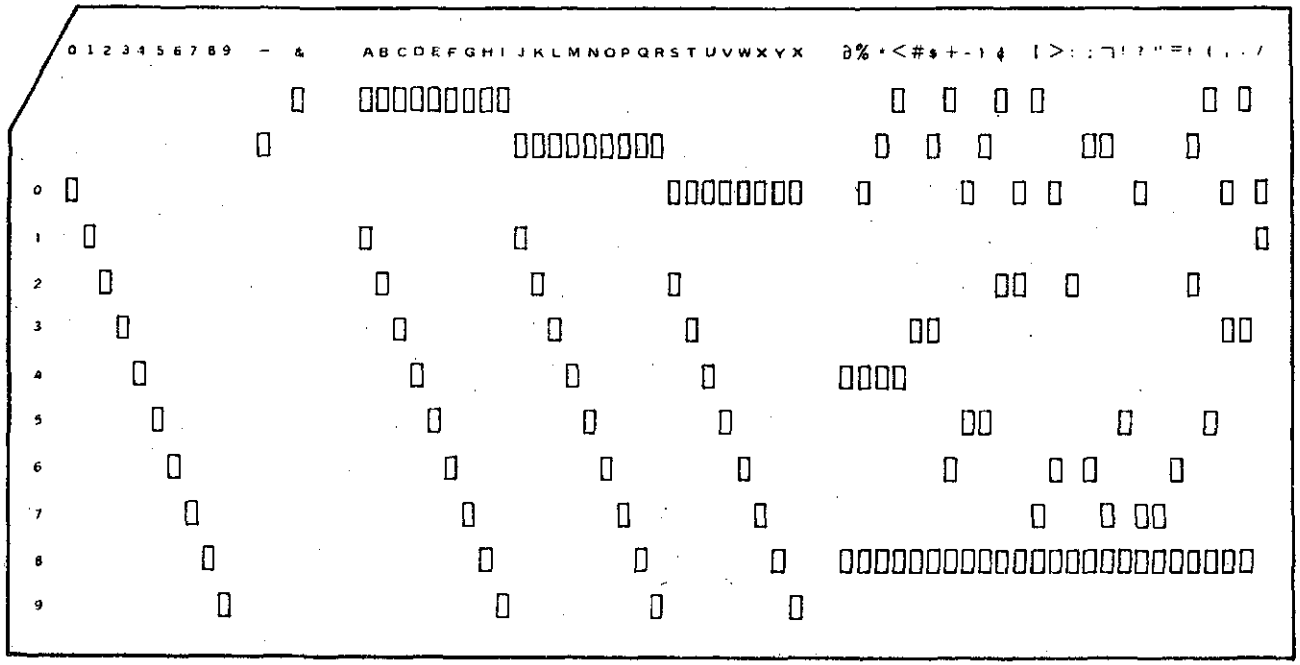
Las doce líneas están subdivididas en dos áreas: el área numérica y el área de zonas. El área numérica contiene las líneas cero a nueve para representar los dígitos respectivos y el área de zonas, las líneas cero, once y doce, para representar caracteres alfabéticos o especiales. Luego, la línea cero actúa como zona o con su valor numérico, según la información que se desee representar.

La tarjeta tiene, a lo largo, capacidad para ochenta columnas, cada una de las cuales puede contener un carácter numérico, alfabético o especial, de tal manera que una tarjeta puede contener hasta ochenta caracteres. Sin embargo, aun cuando es poco corriente hacerlo, es posible subdividir la tarjeta en dos zonas con seis líneas cada una, lo mismo que la tarjeta UNIVAC, con lo cual se aumenta la capacidad total a ciento sesenta caracteres.

El registro de la información se obtiene mediante una o más perforaciones rectangulares distribuidas en las doce posiciones de las columnas. El código más usado es el Código Decimal Codificado en Binario Ampliado para el Intercambio de información (Extended Binary Coded Decimal Interchange Code- EBCDIC). En este código los dígitos se representan por una perforación en la línea en que aparece impreso el dígito y los caracteres alfabéticos por dos perforaciones en la columna, una que corresponde a *zona* y la otra a *dígito*. De acuerdo con lo anterior, las letras A a la I tendrán una perforación en la ZONA 12 y, además, una en el dígito 1,2,... ó 9. Las letras J a la R tendrán perforación en la ZONA 11 y, además, una en el dígito 1,2,... ó 9 y, finalmente, las letras S a Z tendrán perforación en la ZONA 0 y, además una en el dígito 2,3,... ó 9. Los caracteres especiales se representan con una, dos o tres perforaciones dentro de la columna.

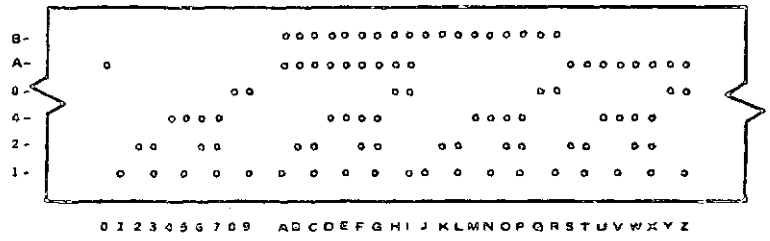
Con el objeto de facilitar la manipulación de las tarjetas, éstas tienen normalmente la esquina superior izquierda recortada, lo que permite ver rápidamente aquellas tarjetas que están colocadas en posición anormal, por error o porque sea necesario perforarlas de nuevo. También con ese recorte es fácil distinguir tarjetas de otros grupos, a las que se les ha recortado la esquina superior derecha.

A continuación, la figura muestra una tarjeta en la que se han representado todos los caracteres. Tal como en la tarjeta UNIVAC, en el borde superior es posible imprimir el contenido de cada columna. En el borde izquierdo aparecen impresos los dígitos correspondientes a las líneas0 a 9. En la tarjeta real esos dígitos se repiten a lo largo de la tarjeta, al mismo tiempo se tiene impresa, entre las líneas 0 y 1, y debajo de la línea 9, la numeración de las columnas desde 1 a 80.


0 1 2 3 4 5 6 7 8 9 - &
A B C D E F G H I J K L M N O P Q R S T U V W X Y X
∂ % * < # $ + - ¬ ¢
| > : ; ¬ ! ? " = ! (, . /

IBM ha puesto en uso una nueva tarjeta que tiene las características siguientes: mide 3 1/4 pulgadas de largo por 2 5/8 pulgadas de ancho, lo que significa que es más chica que la tarjeta usada hasta ahora. Sin embargo, está dividida en tres zonas, cada una con capacidad para seis líneas a lo ancho y treinta y dos columnas a lo largo, lo que permite registrar en total noventa y seis caracteres.

El código utilizado es el Decimal Codificado en Binario (Binary Coded Decimal -BCD). En la figura siguiente se representan los caracteres numéricos y los alfabéticos en el código BCD. Una perforación en la línea designada con A representa el 0 o la ZONA 0 de la tarjeta anterior, una perforación en la línea B representa la ZONA 11 y perforación en ambas líneas representa la ZONA 12. Las perforaciones son circulares de un diámetro aproximado de 1 mm.



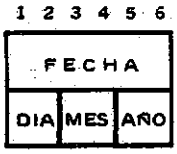
iii) *Diseño de Tarjetas.* Una vez que se ha terminado el análisis de un problema y de él se ha concluido, entre otras cosas, que los datos se registrarán en tarjetas, es necesario hacer una distribución racional de los datos dentro de ellas. Evidentemente, no hay una forma estándar para el diseño, pues dependerá del problema de que se trate. Sin embargo, deberían considerarse, al menos, los siguientes criterios:

La información debe quedar agrupada de acuerdo con una relación lógica. Por ejemplo, en primer lugar, los datos de identificación y a continuación el resto de la información, manteniendo siempre una estructura jerárquica en la que existe una dependencia natural.

Para aquellos datos que tendrán variación en magnitud a lo largo del tiempo, debe preverse espacio suficiente para que absorba esa variación.

En aquellos problemas en que los datos ocupen más de una tarjeta, debe identificarse ésta y al mismo tiempo repetir la identificación mínima de la persona o el objeto relacionado con los datos.

El espacio que ocupa cada dato se denomina campo y si es necesario segmentarlo, dará origen a subcampos. Como ejemplo se puede dar el campo FECHA, que se divide en los subcampos DIA-MES-AÑO, ocupando en total seis columnas.



iv) *Tipos de tarjetas.* Al hablar de tipo de tarjeta se hace referencia a la estructura, a la forma y a la función que tiene, independientemente de su diseño.

Los tipos básicos de tarjetas se indican a continuación:

Tarjeta de transcripción. Es la que se perfora a partir de un documento.

Tarjeta dual. Es la que se perfora a partir de datos escritos sobre la misma tarjeta.

Tarjeta de marca sensible. Es la que se perfora automáticamente a partir de marcas hechas con lápiz de grafito en lugares especialmente dispuestos para ellas.

Tarjetas con talón. Se denomina así a la tarjeta que tiene una parte que puede ser desprendida de ella. Esta parte (el talón) podrá estar a la izquierda o la derecha de la tarjeta. Normalmente se usan para comunicación de consumos (energía eléctrica, agua, gas); con la tarjeta se acude a efectuar el pago y se recibe el talón como comprobante. La parte restante, que puede ser del tamaño normal de una tarjeta, o menor, es utilizada por la compañía o empresa.

Tarjetas de forma continua. Es aquella que está unida por la parte superior y por la inferior con otras tarjetas similares formando un formulario continuo.

Tarjeta de doble uso. Es aquella que tiene impresa desde la columna 1 hasta la 40 y, girándola en 180°, muestra nuevamente la misma impresión en la mitad izquierda. Se puede utilizar solamente cuando la información ocupa como máximo cuarenta columnas. Una vez que las tarjetas así perforadas dejan de usarse en su primera mitad, puede perforarse la mitad que ha quedado libre, con nueva información.

Tarjetas porta-perforación. Son aquellas que se han construido de tal manera que todas las posibles posiciones perforables están semiperforadas. Se usan cuando la cantidad de información es pequeña y fácilmente registrable en forma manual por una persona que debe utilizar un punzón o estilete adecuado.

Tarjetas con ventana. Son aquellas en las que se ha cortado una ventana rectangular en el lado derecho para permitir la colocación de marcos de micro-película. Para usar este tipo de tarjeta se necesita un dispositivo lector. En el lado izquierdo de la tarjeta se puede perforar información en la forma normal.

Tarjeta-cheque. Tiene el formato de un cheque, de acuerdo con las estipulaciones del Banco respectivo. Puede entonces ser cobrada en la forma usual y posteriormente procesada.

b) *Cinta de papel.*

El uso de la cinta de papel perforada, es bastante menor que el de la tarjeta, por cuanto tiene varias limitaciones que la hacen poco atractiva para el registro de datos. El hecho de ser un elemento continuo no permite intercalar o quitar información, como tampoco corregir o modificar datos. Sin embargo, su misma continuidad le permite registrar información de longitud variable mayor que la que pudiera contener una tarjeta; además, existe una serie de máquinas como es el caso de las cajas registradoras que, junto con hacer la operación normal de registro, perforan la misma información en forma paralela y automática en cinta de papel, con lo cual se ahorra el tiempo de transferencia de datos desde un documento hasta el elemento de registro.

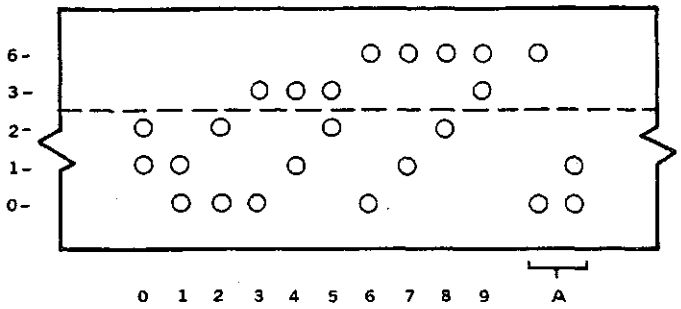
Los códigos más usados ocupan cinco, siete u ocho canales de información, ubicados en forma paralela a lo largo de la cinta. Los datos se registran mediante perforaciones circulares a lo ancho de la cinta y sobre los canales mencionados (columna).

i) *Cinta de cinco canales.* Código dos de cinco. En este código se asignan a cada uno de los canales los valores 0, 1, 2, 3 y 6 respectivamente. Los dígitos 1 a 9 se representan con dos perforaciones, de tal manera que la suma de los valores correspondientes a los canales perforados dé el dígito que se registra. El cero se representa con la combinación de los canales de valores 1 y 2 perforados.

Los caracteres alfabéticos y especiales se representan mediante combinaciones de dos dígitos. Por ejemplo, la letra A se representa con los dígitos 6 y 1 perforados. El tipo de instrucción de transferencia de datos a memoria, o desde ella, determina la interpretación como dígito simple o como pares de ellos.

La ventaja de representar cada carácter con sólo dos perforaciones está en el hecho de poder detectar en forma automática la menor o mayor cantidad de perforaciones por columna, lo que se interpreta en forma inmediata como error causado por el dispositivo.

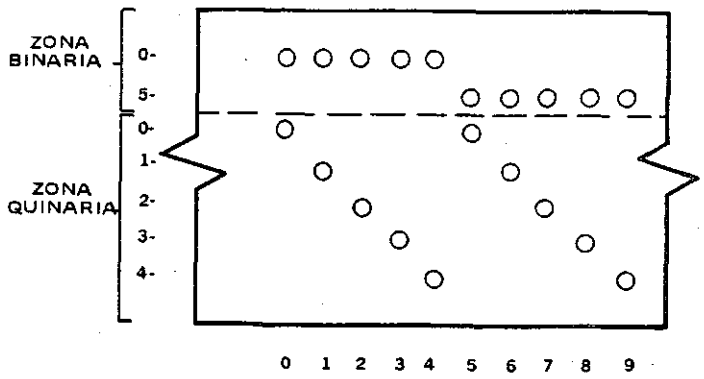
Entre los canales 2 y 3 se tiene un canal con perforaciones de menor diámetro, el cual cumple un doble objetivo: servir para que en él engrane una ruedecilla con dientes que arrastran la cinta cuando aquélla gira, por lo que se le denomina canal de arrastre y, además, servir como punto de referencia para determinar cuál es el anverso o reverso de la cinta.



Código de teletipo (código n de cinco). Se denomina también código n de cinco porque los caracteres, tanto numéricos como alfabéticos y especiales, se representan con una combinación variable de perforaciones a lo ancho de la cinta, esto es, puede haber una, dos, tres o cuatro perforaciones. Los cinco canales perforados en la columna indican que la información que sigue es alfabética y se interpretará así hasta que aparezca la combinación que indica que a continuación siguen dígitos.

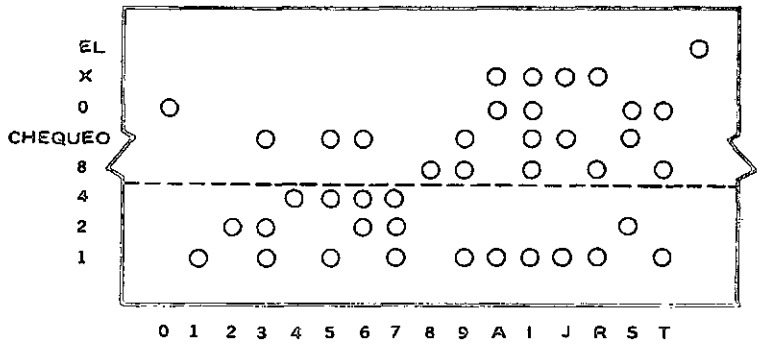
ii) *Cinta de siete canales.* Se utiliza en esta cinta el código biquinario, para lo cual la cinta se subdivide en dos zonas: la zona que queda sobre el canal de arrastre y que contiene las posiciones binarias (dos) o, lo que es lo mismo, dos canales, con valores asignados 0 y 5, respectivamente, y la zona que queda bajo el canal, que contiene las posiciones quinarias o cinco canales con valores asignados 0,1,2,3 y 4, respectivamente.

Los dígitos 0 a 9 son codificados por una combinación de perforaciones de una posición binaria y una posición quinaria, de tal manera que la suma de los valores asignados a los canales perforados da el dígito respectivo. Los caracteres alfabéticos y especiales se representan con combinaciones de dos dígitos.



iii) *Cinta de ocho canales.* Se utiliza el código Decimal Codificado en Binario. La representación de los caracteres es la misma que se utiliza en la nueva tarjeta IBM. En este caso se denomina canal X a la línea B de la tarjeta y canal 0 a la línea A. Se han agregado dos canales: el de CHEQUEO o de PARIDAD, ubicado entre los canales 0 y 8, destinado a completar un número impar de perforaciones en la columna y el canal EL (End Of Line) ubicado sobre el canal X para indicar, cuando tiene perforación, que ha terminado un grupo de datos (registro).

El canal de arrastre está ubicado entre los canales 8 y 4.



El canal de chequeo o de paridad permite detectar errores en la perforación causados por dispositivos defectuosos.

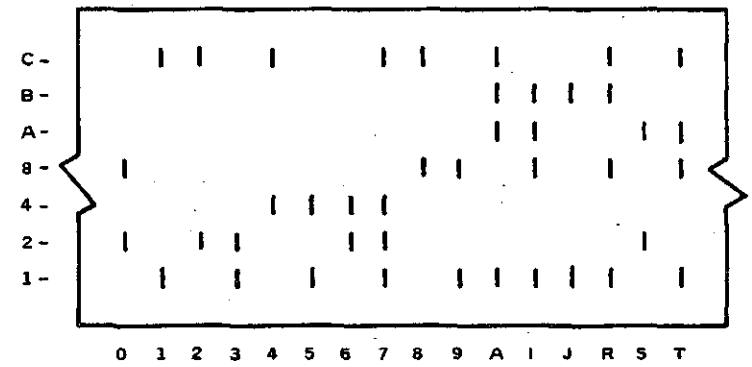
c) *Cinta magnética*

Es un elemento muy utilizado para registrar información por la capacidad de almacenamiento que tiene, como asimismo por la rapidez con que es posible transferir los datos hacia y desde ella. Normalmente se registran datos y resultados, sean estos últimos parciales o totales. Sin embargo, los programas que realizarán el proceso de la información pueden ser también almacenados.

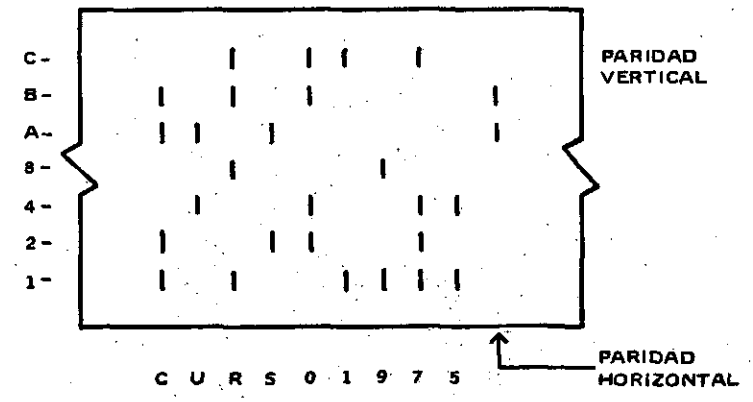
La cinta magnética es similar a la cinta usada en las grabadoras de sonido en cuanto a su función, aspecto y manejo. Difiere sólo en sus dimensiones y en el hecho de ser fabricada con especificaciones mucho más estrictas.

Es una cinta de plástico que tiene normalmente media pulgada de ancho, recubierta en una de sus caras con una película ferromagnética, la que se obtiene mezclando partículas microscópicas de óxido de hierro con un agente que sirve como adhesivo. En esta cara se efectúa el registro de los datos mediante la magnetización de pequeñas áreas discretas en canales ubicados en forma paralela a lo largo de la cinta. Tal como en la cinta de papel, el carácter se estructura a lo ancho de ella. Existen dos tipos de cinta: uno que contiene siete canales y el otro con capacidad para nueve canales.

Dado que los conceptos son los mismos para los dos tipos de cintas, se expondrá solamente la de siete canales, en la que el código utilizado para representar información es el Decimal Codificado en Binario visto anteriormente. En la cinta magnética se utiliza un canal (C) para completar un número *par* de áreas discretas a lo ancho de ella (columna). El área discreta se designa con el nombre BIT (BInary digiT) el que se utilizará en lo sucesivo para identificar, en general, a todos aquellos elementos que pueden tomar sólo dos estados que representen *a su vez ausencia o presencia.* A diferencia del código BCD, utilizado en la tarjeta y en la cinta de papel, el cero se representa en la cinta magnética combinando los canales 8 y 2.



El chequeo efectuado con la paridad por carácter, como se ha visto anteriormente, se realiza con el objeto de detectar errores causados por el dispositivo de grabación. La paridad por carácter se conoce como PARIDAD VERTICAL para diferenciarla de la PARIDAD HORIZONTAL, que se realiza a lo largo de cada canal. En esta verificación se crea, al término de cada grabación, una columna de bits, los que se obtienen al completar un número par de ellos a lo largo de cada canal.



La orden de grabar o no un bit en el canal C llega en forma independiente de la orden de grabar los bits restantes del carácter. Es posible entonces detectar la grabación errónea de un carácter, por falla del dispositivo, mediante la paridad horizontal, dado que al grabar esa columna habrá una discrepancia entre lo que se debe registrar verticalmente y lo que debe quedar al contabilizar los bits de cada canal. Es fácil comprobar lo anterior considerando la figura que representa el trozo de cinta; si en la grabación de la letra R, por defectos del dispositivo se omite el bit del canal 8, al efectuarse la grabación de la columna de paridad horizontal tendrá que registrarse un bit en el canal 8, para completar un número par. Al mismo tiempo, como se tiene un bit en el canal B y otro en el canal A, deberá grabarse un bit en el canal C nuevamente, para completar la paridad. Sin embargo, en el canal C se tienen cuatro bits, lo que significa que NO debe aparecer un bit en la columna de paridad horizontal. Esta discrepancia acusa el error de grabación.

En idéntica forma se realiza la grabación en la cinta de nueve canales, de ahí que no interesa mayormente analizar el código utilizado, que es de difícil memorización. En esta cinta, la columna recibe el nombre de BYTE, es decir, ocho bits de información y un bit de paridad.

La proximidad que existe entre una columna y otra determina la DENSIDAD de grabación, que se mide normalmente en "caracteres por pulgada" en la cinta de siete canales y en "Bytes por pulgada (BPI)" en la de nueve canales. Las densidades más usadas son de 800 y 1600 bytes por pulgada.

Las longitudes de cinta que se utilizan comúnmente son: 2400, 1200, 600 y 300 pies, con una longitud mínima de 50 pies en cada caso. Se puede determinar así la cantidad teórica de bytes por cinta. Por ejemplo, para la cinta de 2400 pies se tendrá:

Bytes por cinta = 2400 * 12 * 800 = 23.040.000

Bytes por cinta = 2400 * 12 * 1600 = 46.080.000

d) *Caracteres con tinta magnética.*

Estos caracteres se utilizan fundamentalmente en cheques y otros documentos bancarios. El código que se usa se denomina MICR (Magnetic in Character Recognition) y consta de dígitos y símbolos que se imprimen con tinta que contiene partículas de óxido de hierro. Además, los caracteres tienen una forma especial que hace más fácil su posterior reconocimiento.

e) *Caracteres ópticos.*

Tal como los caracteres con tinta magnética, éstos deben imprimirse con un estilo especial que permite un reconocimiento fácil. Sin embargo, existen dificultades en la lectura, a causa de las diferencias en

calidad de impresión, del método de reconocimiento, etc. La ventaja que tienen respecto a otros elementos es el tiempo que se ahorra al no tener que perforar o grabar la información para que pueda ser introducida al computador.

f) *Formulario continuo.*

Es el elemento de salida por excelencia, dado que en él se imprimen los resultados finales o parciales de un proceso en el sistema y lenguaje utilizado corrientemente por el ser humano. Se puede diseñar todo tipo de formularios y planillas que permitan una mejor comprensión de la información obtenida como asimismo una economía de tiempo en cuanto a posterior ordenamiento de los resultados como a eliminación de trámites y procedimientos innecesarios.

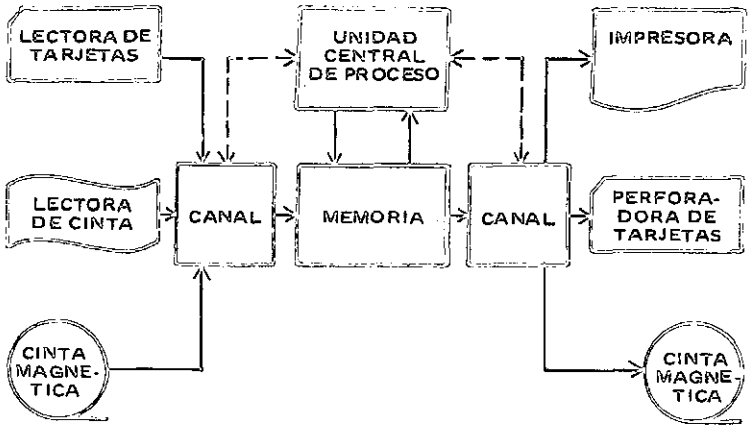
B. Entrada/Salida

a) *Comunicación con la unidad de almacenamiento.*

Una de las características del Sistema EPD, que facilita las operaciones simultáneas necesarias para la máxima utilización de los recursos propios del sistema, es el empleo de dispositivos que permiten equilibrar las velocidades de operación de las máquinas de entrada/salida, que son aparatos electromecánicos y las velocidades del resto de las unidades del sistema, que son electrónicos. Estos dispositivos se llaman CANALES y para cumplir su función "amortiguadora" tienen una pequeña memoria en la que reciben información, con lo que se logra dejar libre a la Unidad Central de Proceso (UCP) de la labor de recibir o enviar información a velocidades electromecánicas, destinando ese tiempo ganado a proceso de datos. Cuanto la UCP requiere datos o tiene que entregarlos, se comunica con el canal a velocidades electrónicas.

De hecho, el canal es un pequeño computador que se dedica sólo a las operaciones de entrada/salida. En algunos casos, se usan exclusivamente para acoplamiento de dispositivos de alta velocidad, como las cintas magnéticas y se denominan canales SELECTORES. En otros, pueden atender simultáneamente varios dispositivos de baja velocidad, como lectoras de tarjetas, impresoras, etc. y se denominan canales MULTIPLEXOR; opcionalmente, estos canales pueden atender dispositivos de alta velocidad.

Cuando se atienden varios dispositivos de baja velocidad en forma simultánea, se dice que el canal trabaja en modalidad MULTIPLEX o modalidad BYTE. Cuando se conectan varios dispositivos de alta velocidad a un canal multiplexor, sólo uno de ellos podrá funcionar a la vez, diciéndose entonces que el canal opera en modalidad RAFAGA (modo BURST). Los canales selectores siempre trabajan en modalidad ráfaga.



b) *Dispositivos de entrada*

i) *Lectora de tarjetas.* Hay dos sistemas que permiten la transmisión de los datos perforados en las tarjetas: El sistema de escobillas y el de células fotoeléctricas. El primero consiste en doce escobillas que barren en forma paralela las doce líneas de la tarjeta. Cuando la escobilla pasa por una perforación, hace contacto con un tambor de transmisión, lo cual cierra un circuito, hecho que se traduce en el envío de una señal o impulso eléctrico. En el sistema de células fotoeléctricas, el principio es el mismo, dado que se tienen doce células, una para cada línea de la tarjeta. El circuito se cierra cuando la luz emitida por un foco luminoso pasa a través de la perforación activando la célula respectiva.

Las velocidades de lectura de tarjetas varían desde 100 hasta 2 000 tarjetas por minuto, siendo la velocidad promedio de aproximadamente 700.

Para efectuar una operación continua, los dispositivos de lectura poseen depósitos de alimentación de tarjetas con capacidades que oscilan entre 1 000 y 3 000 tarjetas, contando, además, con bolsillos receptores de tarjetas leídas.

ii) *Lectora de cinta de papel.* Igual que en las lectoras de tarjetas existen dos sistemas de transmisión de datos: el de escobillas o electromecánico y el de células fotoeléctricas. El primero permite velocidades de hasta 100 caracteres por segundo, y con el segundo se puede llegar hasta 2 000 caracteres por segundo.

iii) *Lector de caracteres con tinta magnética.* La técnica utilizada se basa en cabezas lectoras de tinta magnética, que producen señales eléctricas cuando detectan la pasada de caracteres magnéticos. Los impulsos eléctricos emitidos son analizados por circuitos especiales para

determinar, por comparación con tablas almacenadas, cuál es el carácter que ha sido leído.

iv) *Lector de caracteres ópticos.* Hay muchos tipos de lectores de caracteres ópticos, algunos de los cuales pueden leer marcas especiales, números, caracteres alfabéticos e incluso caracteres en general, impresos manualmente. Existen también varios métodos de reconocimiento de caracteres ópticos, de los cuales se podrían señalar los siguientes:

Reconocimiento mediante la detección de ausencia o presencia de puntos. Se identifica el carácter comparando los puntos detectados con patrones de punto almacenados en la memoria.

Reconocimiento por análisis del carácter hecho con células foto-eléctricas y comparación con patrón almacenado.

Reconocimiento por búsqueda de características especiales y comparación con patrón almacenado.

Reconocimiento por detección de partes no impresas del carácter y determinación, con ellas, de códigos establecidos previamente.

c) *Dispositivos de entrada o salida*

i) *Lectora-perforadora de tarjetas.* Algunas máquinas pueden realizar indistintamente la lectura o la perforación, para cada una de cuyas funciones cuentan con una estación, que se activa de acuerdo con la operación que deba realizarse. Para la perforación, se tienen doce punzones correspondientes a cada una de las líneas de la tarjeta, y se obtienen velocidades que varían desde 100 hasta 500 tarjetas por minuto. Normalmente, las lecto-perforadoras tienen dos bolsillos de recepción, uno de los cuales puede recibir tarjetas seleccionadas por programa, operación que se llama selección de bolsillo.

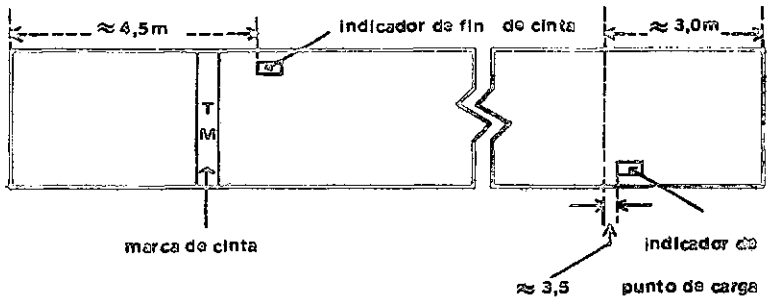
ii) *Lectora grabadora de cinta magnética.* Para colocar la cinta en el dispositivo, se enrolla en carretes de plástico de 6 1/4, 8 1/2, ó 10 1/2 pulgadas de diámetro. Dado que la operación de grabación borra automáticamente cualquier información previa que hubiese en la cinta, se ha diseñado un anillo de plástico que protege la información contra accidentales órdenes de grabación y que puede ser colocado en una ranura del carrete. Si el carrete es montado en la lectora-grabadora con el anillo, éste oprime un botón de la unidad, lo que permite grabar o leer información. Por el contrario, si se saca el anillo para montar el carrete, el botón queda en la ranura sin ser oprimido, lo que impide la grabación de información, no así la lectura.

Debido a la precisión con la que debe efectuarse la grabación o la lectura, es necesario detectar exactamente donde se debe iniciar la operación. Por otra parte, como es imposible grabar hasta el final de la cinta, es necesario también detectar cuando se aproxima dicho final. Ambas cosas pueden efectuarse con la ayuda de "marcadores fotosensibles".

Aproximadamente a tres metros desde el comienzo de la cinta, en la cara brillante de ella y en el borde cercano al operador, se coloca una

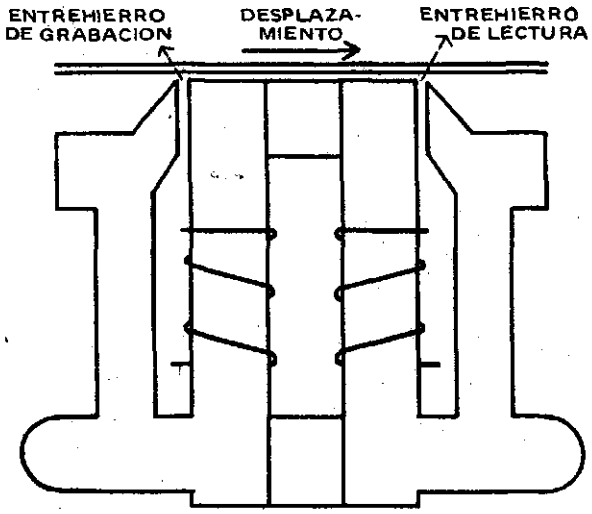
plaquita que permite reflejar (REFLECTIVE SPOT) la luz emitida por un foco luminoso, con lo que se activará una célula fotoeléctrica, la que pondrá en funcionamiento los circuitos que ubican el dispositivo de grabación exactamente frente al punto de carga (LOAD POINT). Las placas reflectoras son pequeños trozos de plástico recubiertos con una delgada película de aluminio. Pueden ser colocados en forma manual, dado que se adhieren con la sola presión de los dedos.

Para indicar que el término de la cinta se aproxima, se utiliza una placa similar a la anterior, la que se coloca aproximadamente a cuatro metros y medio del extremo de la cinta, en la cara brillante, pero en el borde opuesto al operador. Esta marca, como se dijo anteriormente, indica nada más que la cinta se va a terminar, es decir, se puede continuar grabando información bajo la exclusiva responsabilidad del programador. Si la operación es de lectura, la Unidad de Cinta no reconoce el indicador de fin de cinta (END OF TAPE), en cuyo caso es un carácter especial o marca de cinta (TAPE MARK) el que señala el término de la información.



La grabación o lectura de la información en la cinta magnética se logra a través de una cabeza-lecto-grabadora, de las que existe una por canal, y que magnetiza o interpreta las pequeñas áreas discretas que representan los datos.

Para efectuar tanto la grabación como la lectura, hay en la cabeza lecto-grabadora dos aberturas o entrehierros, uno para cada función. La ventaja de este sistema es la de poder verificar en el entrehierro de lectura lo que se acaba de grabar en la abertura de grabación. Cualquier discrepancia es acusada inmediatamente como error por falla del dispositivo.



La grabación destruye la información que contiene la cinta. La lectura, en cambio, puede repetirse un número indefinido de veces sin que eso signifique alterar los datos contenidos. Todos los elementos magnéticos en los que se registre información tienen estas características.

Para grabar o leer la información en la cinta magnética, es necesario que ésta vaya siendo transferida de su carrete original a otro que está fijo en el dispositivo. La operación de devolver la cinta a su carrete original se llama rebobinado.

Para cargar la cinta en la unidad deben darse los pasos siguientes:

– Pasar el extremo libre de la cinta a través de los rodillos y enrollarla en el carrete de la unidad.

– Verificar que la placa reflectora, indicadora del punto de carga, pase a la derecha del dispositivo con las cabezas lecto-grabadoras.

– Cerrar la puerta de la unidad.

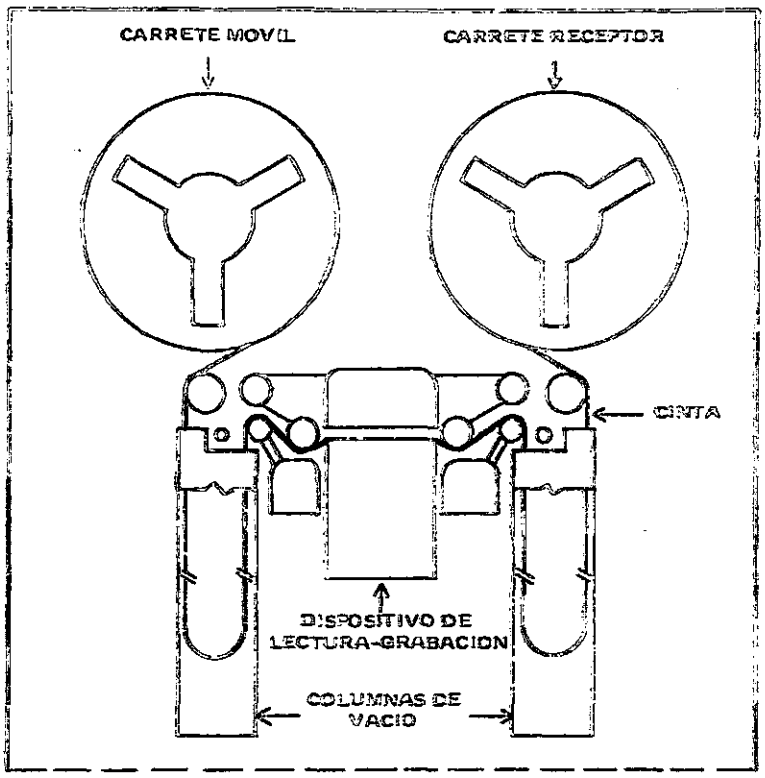
– Oprimir la tecla de carga, con lo que baja el dispositivo detector de comienzo y fin de cinta, se introduce ésta en las columnas de vacío (las que permiten amortiguar los cambios bruscos de velocidad) y se rebobina la cinta a baja velocidad. Al detectarse la placa reflectora, indicadora del punto de carga, se detiene el rebobinado.

– Oprimir la tecla de iniciación (START) con lo cual se deja la unidad lista (READY) para ser utilizada por el sistema.

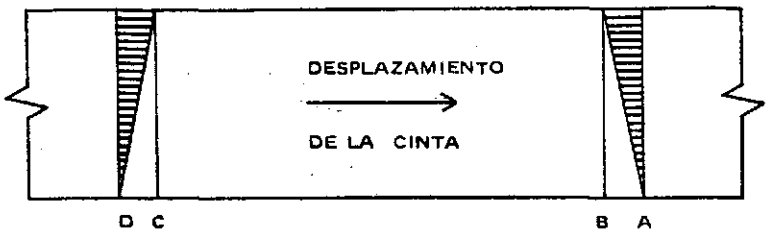
Para efectuar lectura o grabación, es necesario que la cinta adquiera una cierta velocidad denominada velocidad de trabajo.

Existen nuevas unidades en que los carretes están en depósitos de plástico que protegen la cinta del polvo y de los riesgos que significa el

manejo. Estos depósitos se montan en la unidad y se produce en forma automática el enhebrado de la cinta, rebobinado y posicionamiento de las cabezas lecto-grabadoras.



La cinta magnética se encuentra detenida mientras se ejecutan instrucciones que no hagan referencia a ella. Se pondrá en movimiento sólo cuando la unidad reciba la orden de grabar o leer. Desde el momento en que llega la orden hasta aquel en que se llega a la velocidad de trabajo, transcurre un lapso en el que la cinta ha avanzado un determinado espacio. El trozo de cinta recorrida se conoce como "Espacio entre registros" (INTER RECORD GAP-IRG) o "espacio entre bloques" (INTER BLOCK GAP—IBG) y es de aproximadamente 0,75 pulgadas para cinta de siete canales y 0,60 pulgadas para cinta de nueve canales.

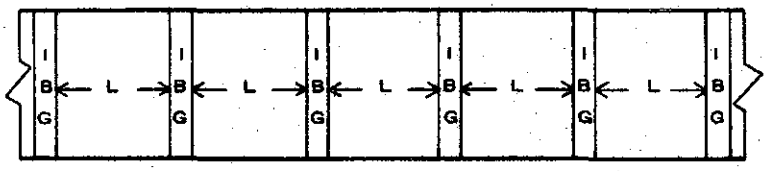


Los espacios AB y CD son los denominados "espacios entre bloques". El trecho comprendido entre ambos se define como REGISTRO DE CINTA (RC) o registro físico.

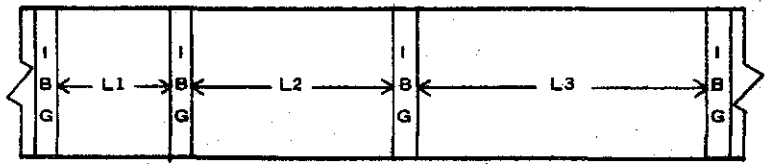
Los datos agrupados considerando algún criterio que facilite su proceso forman un REGISTRO LOGICO (RL). Si RC=RL se dice que la cinta está DESBLOQUEADA. Al mismo tiempo se puede tener:

CINTA DESBLOQUEADA { registros lógicos de LONGITUD FIJA / registros lógicos de LONGITUD VARIABLE

– DESBLOQUEADA, longitud fija



– DESBLOQUEADA, longitud variable

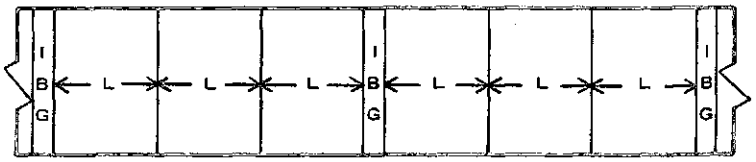


Si RC=n*RL (n entero mayor que uno) se dice que la cinta está BLOQUEADA. La cantidad de registros lógicos por registro de cinta (BLOQUE) se designa como "razón de bloqueo" o "factor de bloqueo".

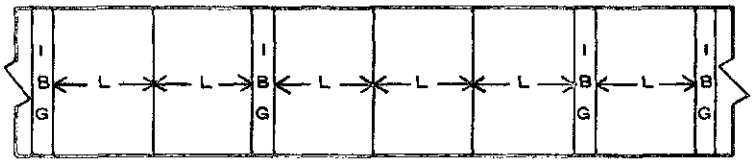
En las figuras que se presentan a continuación, se han separado los registros lógicos entre sí por una línea, aun cuando en la práctica no existe esa separación física. Las subdivisiones que se pueden lograr dentro de una cinta bloqueada son:

- CINTA BLOQUEADA
  - registros lógicos de LONGITUD FIJA
    - IGUAL razón de bloqueo
    - DISTINTA razón de bloqueo
  - registros lógicos de LONGITUD VARIABLE
    - IGUAL razón de bloqueo
    - DISTINTA razón de bloqueo

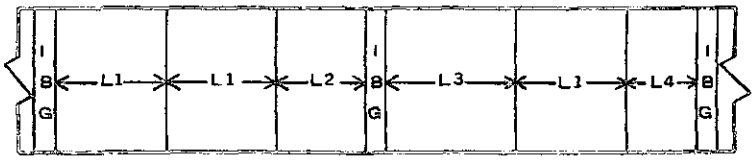
– BLOQUEADA, longitud fija, IGUAL razón de bloqueo



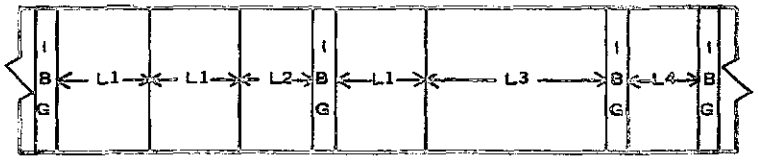
– BLOQUEADA, longitud fija, DISTINTA razón de bloqueo



– BLOQUEADA, longitud variable, IGUAL razón de bloqueo



– BLOQUEADA, longitud variable, DISTINTA razón de bloqueo



Existen actualmente dos formas para registrar información en una cinta magnética: la primera requiere que los datos estén registrados en otro medio, desde el cual se leerán a la memoria del computador y desde ella se enviarán a la cinta magnética. La otra forma es directa, esto es, se tiene una máquina con teclado, independiente del computador, con la cual se puede grabar la información a medida que se oprimen las teclas. La grabación puede ser hecha en pequeños discos (diskette) y desde ellos llevar los datos a cinta de nueve canales, mediante un convertidor; puede ser grabada en cintas pequeñas (cassettes) y de allí transferida a cintas de siete o nueve canales; puede ser grabada directamente en cintas de siete o nueve canales o puede ser grabada en disco magnético mediante un pequeño computador y transferida posteriormente a cinta magnética mediante control de un supervisor.

Este tipo de máquinas, que se denomina sistema de entrada de datos (data entry), se inició en la década de 1950 con el equipo Unityper, que permitía grabar directamente en una cinta UNIVAC, compatible con las usadas por un computador, a razón de 50 caracteres por pulgada en bloques de 120 caracteres, separados por un espacio (gap) de 1,25 pulgada.

En 1964 se desarrolló un dispositivo en el estado de Nueva York, mediante el cual el operador podía efectuar la corrección de los datos con sólo pulsar sobre el error el carácter corregido. Se logró entonces la posibilidad de que cada unidad funcionara como registradora (perforadora) y verificadora.

En esa oportunidad se hizo un estudio que reveló que el aumento de producción había sido del orden del 20 al 40 por ciento.

En 1968 la Communitype Corporation presentó su Data Jotter Modelo 90, que registraba la información en cassette de cinta magnética no compatible.

También la IBM puso en el mercado máquinas similares que utilizaban cassette.

A mediados de 1969 se iniciaron las primeras entregas de dispositivos multiteclados. The Logic Corporation y la Computer Machinery Corporation fueron los primeros en introducir el concepto de entrada multi-estación denominado "grupos". En el Sistema Logic Corp's 720, todos los datos introducidos en los teclados son procesados y almacenados en un disco magnético en el procesador central, el que puede controlar hasta 60 estaciones de teclado. Cada registro es verificado en el mismo disco y corregido, en caso necesario, en la forma convencional. Después de la verificación, los registros son transferidos a cinta magnética compatible.

Las funciones de un sistema de entrada de datos son:

Digitación
Verificación

Validación
Dígito verificador
Chequeo de rango
Consistencia
Campo numérico o alfabético, etc.
Control de formato de registros
Salto de campos
Duplicación de campos
Inserción de ceros, etc.
Salida
Funciones de control
Asignación de tareas a los operadores
Separación de archivos de datos
Identificación de los archivos
Estadística
Estas funciones pueden ser realizadas por un supervisor y también en forma automática.
Errores

A continuación se dan características de algunos equipos:

Olivetti DE 523

Puede grabar y verificar datos entregando el resultado en cassette de cinta magnética no compatible
Relleno de ceros por la izquierda
Relleno de blancos por la derecha
Verifica dígito (Módulo 10 y Módulo 11)
Pantalla con capacidad para 310 caracteres
Contabiliza registros

Olivetti DE 520

Convertidor de cassette a cinta magnética compatible de 7 canales 556/800 bpi o 9 canales 800/1 600 bpi
Características de la cassette de cinta magnética
Longitud aproximada 280 pies (85.3 m)
Capacidad hasta 230 000 caracteres equivalentes a 2 200 tarjetas.

IBM 3740

Tiene la estación de datos 3741 para grabación y verificación o la estación doble de datos 3742 con una mayor capacidad, o ambas estaciones a la vez y tiene además pantallas con 3 ó 6 líneas de 40 caracteres. La información se graba en pequeños discos (diskettes).
Funciones normales:
Entrada
Verificación
Actualización y búsqueda

Contabilización de registros

Realiza programas similares a los de la perforadora, esto es, relleno de ceros, relleno de blancos, salto de campos, duplicaciones, etc.

Verifica dígito (Módulo 10 y módulo 11)

Convertidora 3747

Convierte de diskette a cinta de 9 canales

Desarrolla una velocidad de 300 registros por minuto con detección de errores

Convierte de cinta a diskette para actualización en la 3741 o 3742

Tiene bolsillo con capacidad para 20 diskettes

Características del diskette

Capacidad para, aproximadamente, 1 900 registros de 128 caracteres cada uno

National NCR 736-101

Registra y lee datos en cintas pequeñas de 9 canales, 800 bpi

Tiene verificación y corrección de errores

Rellena ceros por la izquierda

Contabiliza registros

Realiza funciones automáticas programables (duplicación, salto de campos, etc.)

Consolidador National NCR 736-201/202

Las cintas pequeñas producidas por la 736-101 se consolidan en cinta de 9 canales de 2 400 pies

Las cintas se leen con una NCR 736-201, se consolidan en otra igual y la transferencia es controlada por NCR 736-202

Sistema Singer 1500

Tiene el terminal Singer 1501 que permite la entrada de datos con verificación y validación de ellos. Posee una pantalla de despliegue de datos que muestra el carácter real introducido, en 4 u 8 líneas de 32 caracteres cada una. Los datos pueden ser grabados directamente en mini-cintas con capacidad para 122 400 caracteres (900 registros de 136 caracteres).

Se puede conectar a los dispositivos de cinta magnética modelo 1511 (cinta de 7 canales con 556 u 800 bpi). Modelos 1512 ó 1513 (cinta de 9 canales con 800 bpi) o al Modelo 1514 (cinta de 9 canales con 1 600 bpi). Para efectuar la conversión se tiene el programa de "Entrada avanzada de datos para maxi-cintas" que permite convertir la información contenida en mini-cintas dè la serie 1500 a código EBCDIC, Honeywell o ITS, grabándola en registros que pueden variar desde 10 a 1 250 caracteres.

Funciones normales del programa "Entrada de datos generales"

Entrada

Verificación

Validación mediante dígito de verificación y uso de 3 acumuladores para controles de totales

Duplicación automática, salto de campos, supresión e inserción de registros, etc.

Otros programas con que cuenta la serie 1500 son: "Entrada avanzada de datos" que permite examen de rango, búsqueda y comparación con tablas, múltiples verificaciones de dígitos, inserción de constantes, etc. "Programas utilitarios", "Programas de clasificación/intercalación", etc.

Otros equipos periféricos de la serie 1500 son: Impresora Modelo 1525, Impresora de línea Modelo 1552, Modem de datos Modelo 2024 y Adaptador de Comunicaciones Sincrónico Modelo 1535.

Computer Machinery Corporation CMC-5

Consola de supervisión CMC-5: contiene todos los elementos que necesita el supervisor de entrada de datos para controlar y dirigir las operaciones de hasta 16 estaciones de teclado. Se encuentran incluidos en ella un computador compacto de uso general equipado con 8 K palabras (K=1 000 de capacidad de memoria, una unidad de discos magnéticos, una unidad de cinta magnética y un panel de control del supervisor.

Unidad de cinta magnética:

CMC 231 graba en 7 canales con 556/800 bpi

CMC 232 graba en 9 canales con 800 bpi

CMC 235 graba en 9 canales con 1 600 bpi

Unidad de disco magnético:

CMC 721 tiene capacidad para: 18 000 registros de 112 caracteres, 100 formatos registros (opcionalmente se puede aumentar a 240 formatos principales/alternativas y 60 grupos de formato múltiple)

Pantalla video CMC 103 con capacidad para cuatro líneas de 32 caracteres cada una y 16 caracteres de función y estado

Estación de teclado CMC 103 conectado a la pantalla

Verificación de grabación

Control de dígito verificador (Módulo 10 y Módulo 11)

Balance automático de campos

Chequeo de rango

Chequeo de consistencia de campos

Estadísticas de operación

Relleno de ceros por la izquierda, etc.

De las características de los sistemas de entrada de datos vistos

anteriormente, se puede sacar como conclusión que tienen una gran ventaja sobre las tarjetas perforadas, las que aparecen hoy como un soporte de difícil archivo, voluminoso y caro y no pueden ser utilizadas nuevamente como medio de registro de información distinta. La eficiencia en la operación de perforación y verificación es menor que en las operaciones respectivas en los nuevos sistemas. El ambiente de trabajo es ruidoso.

Todo esto hace prever que en la preparación de datos, las tarjetas se aplicarán cada vez más a campos específicos y restringidos, y tienden a ser reemplazadas por los sistemas de grabación directa en cinta compatible.

iii) *Teclado-impresoras.* Son muy parecidas a las máquinas de escribir y se utilizan para comunicación entre el operador y el sistema. A través de ellas el operador recibe mensajes del sistema y puede, en respuesta, entregar comandos que instruyen al sistema acerca de los pasos siguientes que debe realizar. Los mensajes y otro tipo de información que entrega el sistema se imprimen en formulario continuo. Las respuestas del operador se efectúan por medio del teclado y también quedan registradas en el formulario continuo.

Es frecuente que este tipo de máquinas se emplee también para teleprocesamiento, como terminales remotos. Esto significa que una persona puede efectuar consultas a través del teclado a un computador ubicado a distancia y obtener una respuesta casi en forma inmediata.

iv) *Pantallas.* Estos dispositivos proporcionan un medio de comunicación visual entre el usuario y el sistema, para lo que se utiliza una pantalla de tubo de rayos catódicos similar a las pantallas de televisión. Es posible desplegar en dichas pantallas: gráficos, tablas, caracteres alfanuméricos, etc.

Por medio de un lápiz de luz, electrónico, es posible reacomodar, borrar o agregar información en la pantalla para almacenarla posteriormente en la memoria. Igualmente, se pueden entrar mensajes formados por caracteres alfanuméricos y otros símbolos mediante un teclado. Antes de ser almacenados, los mensajes son desplegados en la pantalla para verificación. Sus aplicaciones se han difundido con extraordinaria rapidez, principalmente con propósitos educativos, de control industrial, financiero y de investigación académica.

d) *Dispositivos de salida.*

i) *Impresora.* Los dispositivos de impresión producen registros de salida desde la memoria del computador, en medios detectables por seres humanos. Se denominan impresoras de impacto aquéllas que producen la impresión al oprimir papel y cinta entintada contra el tipo de carácter adecuado, cuando éste pasa frente a la posición que corresponde imprimir.

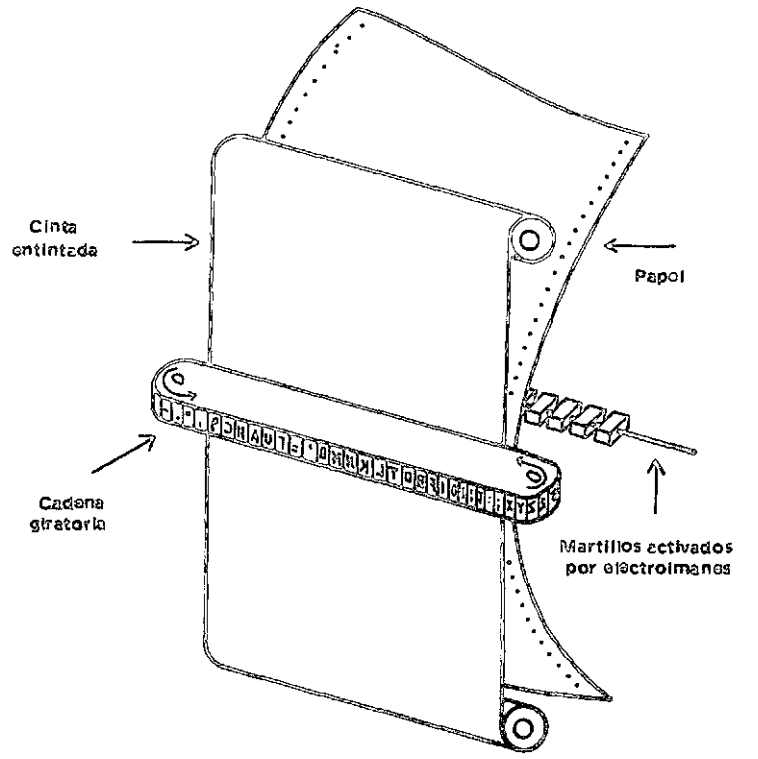
La mayoría de las impresoras pueden imprimir entre 300 y 2 000

líneas por minuto y la cantidad de posiciones de impresión en la línea varía entre 80 y 160, de los que el más común es el de 132.

Las altas velocidades logradas se obtienen a base de la combinación de movimientos de: un dispositivo transportador del papel o formulario continuo, una cadena o tambor en el cual se encuentran ensamblados y resaltados juegos completos de caracteres y martillos accionados por electroimanes frente a cada posición de impresión. El hecho de colocar varios juegos de caracteres hace bajar el tiempo de impresión a causa de que no es necesario esperar la pasada del mismo carácter para imprimirlo.

El espaciamiento vertical se logra, generalmente, por medio de una cinta perforada que se coloca como cinta de control de carro, pero también es posible obtenerlo mediante programa.

Mayores velocidades de impresión se han podido obtener con impresoras sin impacto en las que se ha llegado hasta sobrepasar las 5 000 líneas por minuto. La impresión se obtiene a base de cargas eléctricas sobre papel impregnado químicamente. Desgraciadamente, con este tipo de impresión no se puede obtener copias y la calidad de la impresión no es buena.



ii) *Graficador.* Son dispositivos que permiten dibujar gráficos a base de la información proporcionada desde la memoria. Es posible obtener gráficos de puntos o de línea contínua.

3. UNIDAD DE ALMACENAMIENTO (MEMORIA)

A. Representación de Información

La unidad de almacenamiento, más comúnmente llamada memoria, es la unidad donde se guarda toda la información que permite realizar un proceso, como asimismo los resultados obtenidos de él. La información puede ser, por tanto:

Conjunto de instrucciones al computador
Datos que intervienen en el proceso
Resultados parciales o finales obtenidos

Un requisito básico para entender el funcionamiento de los computadores es el conocimiento de la forma en que se guardan los datos e información en general. Dado que los elementos que componen un computador son transistores, diodos, núcleos magnéticos, cables, etc. y que todos ellos son biestables, esto es, pueden tener dos estados, los datos deben ser representados a base de esas dos posibilidades.

El número restringido de posibilidades induce a utilizar otro sistema de representación que no sea el decimal, dado que con éste tendrían que considerarse diez estados posibles para cada dígito. Pero antes de entrar a analizar otros sistemas es necesario comprender el que se utiliza a diario, es decir, el decimal.

a) *Sistemas numéricos*

i) Sistema decimal: Es el primero con notación de posición que tuvo amplia difusión, creado por los hindúes y luego transmitido a Europa por los árabes. Su éxito se debe, fundamentalmente, a que incorporaron un símbolo que representa la ausencia de datos, el cero, además de utilizar dos conceptos importantes como son: el valor absoluto y el valor relativo a la posición. En el sistema romano, por ejemplo, se hace uso del concepto de valor absoluto, pero no del de valor relativo a la posición.

En el sistema decimal, los valores absolutos son los dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9 y los valores de posición corresponden a potencias de diez. (Por definición, todo número, exceptuando el cero, elevado a cero da como resultado uno). El número 5 041,25 se interpreta de la siguiente manera:

$$5*10^3 +0*10^2 +4*10^1+1*10^0+2*10^{-1}+5*10^{-2}$$

o sea que en la práctica se trabaja con los coeficientes de potencias de diez. La potencia estará dada por la posición que ocupe el coeficiente, dentro del número, en relación con la coma decimal.

La coma será reemplazada en lo sucesivo por el punto decimal, que es el carácter que reconoce normalmente el computador para separar la parte entera de la fraccionaria. En cambio, se seguirá utilizando la palabra dígito para designar a los coeficientes, aun cuando la palabra lleve implícita la idea de diez y se esté trabajando en otros sistemas.

En resumen, los elementos que usa el sistema decimal son:

Base = 10

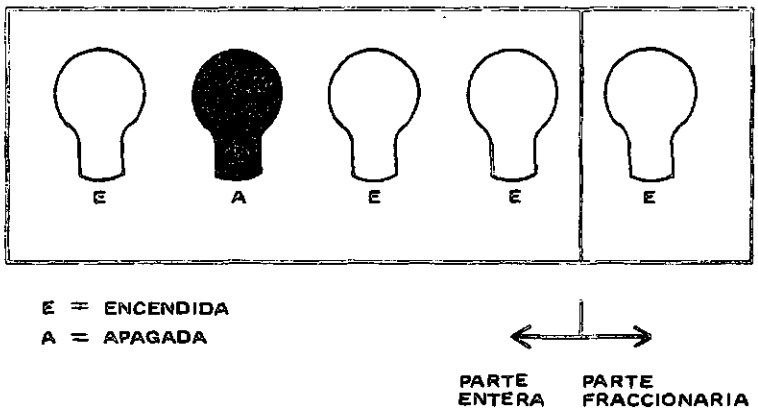
Coeficientes = 0,1,2,3,4,5,6,7,8 y 9

ii) *Sistema binario:* es el sistema que se usará en los computadores a causa de que la base es 2 y por lo tanto los coeficientes son sólo 0 y 1, posibles de representar fácilmente con los elementos del computador.

El número 1011.1 se interpreta de la misma manera que en el sistema decimal, cambiando, por supuesto, la base. Se tiene así:

$1*2^3+0*2^2+1*2^1+1*2^0+1*2^{-1}$

Si se pensara que este número debe quedar representado en un sistema de ampolletas, quedaría en la forma siguiente:



iii) *Conversión del sistema decimal al binario:* a pesar de que el computador realiza las conversiones internamente y en ellas no interviene en ningún momento el usuario, es conveniente conocer la equivalencia entre un sistema y otro para entender la estructura que tendrá la información cuando está almacenada en memoria, como asimismo para poder interpretar "vaciados" de memoria que no se realicen en el sistema decimal.

Para evitar confusiones de escritura, los números se encerrarán

entre paréntesis y como subíndice se colocará la base del sistema respectivo. El problema planteado será, por ejemplo, convertir el número 11.5 del sistema decimal al binario

$$(11.5)_{10} \quad = \quad (\ ?\ )_2$$

La conversión se realiza en dos etapas: conversión de la parte entera y conversión de la parte fraccionaria.

Para convertir la parte entera se divide ésta por dos, el resultado se coloca a la izquierda del dividendo y debajo del resultado, el residuo; se repite la misma operación con el resultado obtenido y se continúa de la misma manera hasta obtener como resultado el valor cero. Los residuos que se han logrado en las divisiones sucesivas son los coeficientes de las potencias de dos, que forman la parte entera en el sistema binario.

El hecho de colocar los resultados siempre hacia la izquierda permite obtener los coeficientes ocupando sus posiciones definitivas.

Cocientes	0	← 1	← 2	← 5	←	11:2
	↓	↓	↓	↓		
Residuos	1	0	1	1		

luego,

$$(11)_{10} \ = (1011)_2$$

Para convertir la parte fraccionaria, se multiplica ésta por dos; la parte fraccionaria del resultado obtenido se coloca a la derecha del multiplicador y debajo de ella la parte entera; se repite la misma operación con el resultado obtenido y se continúa de la misma manera hasta obtener como parte fraccionaria el valor cero o cuando se considera que la precisión conseguida es suficiente, dado que la conversión de la parte fraccionaria puede continuar indefinidamente en muchos casos.

La parte entera de los resultados conseguidos corresponde a los coeficientes buscados. Al colocar aquéllos siempre hacia la derecha, los coeficientes quedan en sus posiciones definitivas.

2*	0.5 → 0.0		Fracción
	↓		
	1		Entero

luego,

$$(0.5)_{10} \quad = \quad (0.1)_2$$

y por consiguiente,

$$(11.5)_{10} \quad = \quad (1011.1)_2$$

Otro ejemplo:

$$(69.48)_{10} = (\ ?\ )_2$$

Conversión de la parte entera

Cocientes	0	← 1	← 2	← 4	← 8	← 17	← 34	←	69:2
	↓	↓	↓	↓	↓	↓	↓		
Residuos	1	0	0	0	1	0	1		

$$(69)_{10} = (1000101)_2$$

Conversión de la parte fraccionaria

2*	0.48	→0.96	→0.92	→0.84	→0.68	→0.36	→0.72	→0.44	→0.88	→0.76 Frac-
	↓	↓	↓	↓	↓	↓	↓	↓	↓	ciones
	0	1	1	1	1	0	1	0	1	Enteros

Se puede continuar la operación indefinidamente

$$(0.48)_{10} = (0.011110101)_2$$

luego,

$$(69.48)_{10} = (1000101.011110101)_2$$

iv) *Conversión del sistema binario al decimal:* para convertir un número escrito en el sistema binario al sistema decimal, basta con desarrollar el primero como una sumatoria de potencias de dos. El resultado de la sumatoria será el número en el sistema decimal.

Ejemplo: $(1101.111)_2 = (\ ?\ )_{10}$

el número binario corresponde:

$$1*2^3+1*2^2+0*2^1+1*2^0+1*2^{-1}+1*2^{-2}+1*2^{-3}$$

$$8 + 4 + 0 + 1 + 0.5 + 0.25 + 0.125$$

$$(1101.111)_2 = (13.875)_{10}$$

v) *Sistema hexadecimal:* con el objeto de facilitar el manejo externo de los datos escritos en el sistema binario, se hace uso del sistema hexadecimal, en el que la base es 16 y los coeficientes son: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. Se han reemplazado los coeficientes 10, 11, 12, 13, 14 y 15 por las letras para evitar confusión en la escritura, pero siguen conservando su valor.

Los métodos de conversión del sistema decimal al hexadecimal y de éste al decimal son los mismos que se han utilizado con el sistema binario.

Por ejemplo:

$$(213.25)_{10} = (\ ?\ )_{16}$$

- Conversión de la parte entera

$$\begin{array}{cccc} 0 & \leftarrow 13 & \leftarrow & \lfloor 213:16 \\ \downarrow & \downarrow & & \\ 13 & 5 & & \end{array}$$

el valor 13 debe cambiarse por la letra respectiva y se tiene

$$(213)_{10} = (D5)_{16}$$

Conversión de la parte fraccionaria

$$\begin{array}{l} 16*\lfloor 0.25 \rightarrow 00 \\ \qquad\quad \downarrow \\ \qquad\quad 4 \end{array}$$

luego,

$$(0.25)_{10} = (0.4)_{16}$$

y de ahí se tiene:

$$(213.25)_{10} = (D5.4)_{16}$$

Lo mismo ocurre con la operación inversa, por ejemplo:

$$(AB1.4C)_{16} = (\ ?\ )_{10}$$

basta con interpretar el número hexadecimal, esto es:

$$A*16^2+B*16^1+1*16^0+4*16^{-1}+C*16^{-2}$$

o lo que es lo mismo:

$$10*16^2+11*16^1+1*16^0+4*16^{-1}+12*16^{-2}$$
$$2560+176+1+0.125+0.046875$$

luego,

$$(AB1.4C)_{16} = (2737.296875)_{10}$$

La conversión del sistema hexadecimal al binario es bastante sencilla, pues consiste solamente en escribir en binario el valor decimal correspondiente a cada dígito hexadecimal. Así, por ejemplo:

$$\begin{array}{ccccccc} & & & AB1.4C & & & \\ 1010 & & 1011 & & 0001 & . \ 0100 & 1100 \\ 10 & & 11 & & 1 & 4 & 12 \end{array}$$

$$(AB1.4C)_{16} = (101010110001.01001100)_2$$

Esta característica se aprovecha para facilitar la conversión del sistema decimal al hexadecimal. En vez de hacerla directamente, se convierte primero a binario y de binario a hexadecimal. Para ello, se forman grupos de cuatro dígitos binarios desde el punto a la izquierda y desde el punto a la derecha. Si faltan dígitos binarios a la izquierda se agregan

ceros, si faltan a la derecha se pueden agregar ceros o calcular, si es posible, más dígitos significativos.
Ejemplo:

$$(269.42)_{10} = (\ ?\ )_2 = (\ ?\ )_{16}$$

Conversión de la parte entera

0	←1	←2	←4	←8	←16	←33	←67	←134	←269:2
↓	↓	↓	↓	↓	↓	↓	↓	↓	
1	0	0	0	0	1	1	0	1	

$$(269.)_{10} = (100001101.)_2$$

$$(100001101.)_2 = (\ ?\ )_{16}$$

se forman grupos de cuatro dígitos binarios desde el punto hacia la izquierda

0001	0000	1101
1	0	D

$$(269.)_{10} = (100001101.)_2 = (10D.)_{16}$$

Conversión de la parte fraccionaria

2* | 0.42	→ 0.84	→ 0.68	→ 0.36	→ 0.72	→ 0.44	→ 0.88	→ 0.76	→ 0.52
	↓	↓	↓	↓	↓	↓	↓	↓
	0	1	1	0	1	0	1	1

$$(0.42)_{10} = (0.01101011)_2$$

$$(0.01101011)_2 = (\ ?\ )_{16}$$

se forman grupos de cuatro dígitos binarios desde el punto hacia la derecha

.0110	1011
6	B

$$(0.42)_{10} = (0.01101011)_2 = (0.6B)_{16}$$

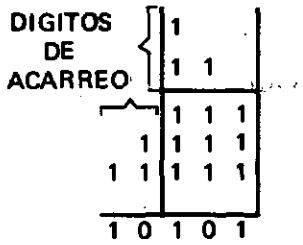
y el número completo será:

$$(269.42)_{10} = (100001101.01101011)_2 = (10D.6B)_{16}$$

vi) *Operación de suma y resta con binarios y con hexadecimales:* al efectuarse la adición en el sistema decimal se realiza la suma de los dígitos de cada columna y en cada una de ellas el valor obtenido se divide por diez, el resultado de esta división es el "acarreo" que se agrega a la columna siguiente y el residuo es el dígito que se coloca bajo la columna computada.

En los otros sistemas se sigue el mismo procedimiento. Por ejemplo, obtener el resultado de la suma de tres sietes en el sistema binario y en el decimal.

BINARIO



DECIMAL

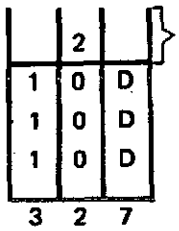
En la suma binaria, para mayor facilidad, se puede utilizar la tabla siguiente:

+	0	1
0	0	1
1	1	10

DIGITO DE ACARREO

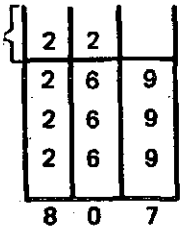
Aplicar el mismo sistema a la suma de tres veces doscientos sesenta y nueve en el sistema hexadecimal, y en el decimal.

HEXADECIMAL



DIGITOS DE ACARREO

DECIMAL



Si se tiene un número N con X dígitos se define como complemento a B a la diferencia $B^X - N$ en que B es la base del sistema al cual pertenece el número.
Ejemplos:

$N = (15735)_{10}$

Complemento a $10 = 10^5 - 15735$

$N=(101110)_2$

Complemento a $2 = 2^6 - 101110$

$N=(ABCD)_{16}$

Complemento a $16 = 16^4 - ABCD$

La resta se puede realizar a base del complemento del sustraendo; para ello se suma al minuendo el complemento obtenido y a continuación se le resta al resultado el valor de B^X. A simple vista, tal vez, no signifique mayor facilidad esta forma de efectuar la resta, sin embargo, su aplicación interna en el computador se traduce en una mayor velocidad de ejecución.

Ejemplos:

Restar en el sistema decimal

$$25742 - 18931$$

esto se debe convertir en:

$$25742 + \underbrace{(10^5 - 18931)}_{\text{complemento a diez}} - 10^5$$

la expresión se puede escribir también como

$$25742 + (100000 - 18931) - 100000$$

en que el complemento es

$$\begin{array}{r} 100000 \\ -18931 \\ \hline 81069 \end{array}$$

luego, el complemento a diez de 18931 es 81069. Nótese que el complemento se puede obtener fácilmente restando de nueve (la base menos uno) cada uno de los dígitos del número y luego sumando 1 al dígito del extremo derecho.

Continuando con la operación se tiene:

$$\begin{array}{r} 25742 \\ +81069 \\ \hline 106811 \end{array}$$

y finalmente

$$\begin{array}{r} 106811 \\ -100000 \\ \hline 6811 \end{array}$$

que es el resultado pedido.

Es importante observar que si se tiene un acumulador o registro con capacidad para cinco dígitos no será necesario realizar la resta de 100000 porque el resultado se habrá obtenido en la etapa anterior, como se ve a continuación:

$$\begin{array}{r} 2\ 5\ 7\ 4\ 2 \\ +\ 8\ 1\ 0\ 6\ 9 \\ \hline 0\ 6\ 8\ 1\ 1 \end{array}$$

Restar en el sistema binario

$$101111 - 100011$$

esto se convierte en:

$$101111 + \underbrace{(2^6 - 100011)}_{\text{complemento a dos}} - 2^6$$

o lo que es lo mismo

$$101111 + (1000000 - 100011) - 1000000$$

en que el complemento es

$$\begin{array}{r} 1000000 \\ -100011 \\ \hline 011101 \end{array}$$

No es necesario hacer la resta para obtener el complemento, ya que se puede determinar en forma rápida "invirtiendo" cada bit (cada 1 se convierte en 0 y cada 0 se convierte en 1) y luego sumando 1 al dígito del extremo derecho.

Al sumar el complemento al minuendo se tiene:

$$\begin{array}{r} 1\ 0\ 1\ 1\ 1\ 1 \\ +\ 0\ 1\ 1\ 1\ 0\ 1 \\ \hline 0\ 0\ 1\ 1\ 0\ 0 \end{array}$$

que es el resultado pedido (se ha considerado un registro con capacidad para seis dígitos).

Restar en el sistema hexadecimal

$$FFFFF - ABCDE$$

esto se convierte en:

$$FFFFF + \underbrace{(16^5 - ABCDE)}_{\text{complemento a dieciséis}} - 16^5$$

o lo que es lo mismo

$$FFFFF + (100000 - ABCDE) - 100000$$

en que el complemento es

$$\begin{array}{r} 100000 \\ -ABCDE \\ \hline 54322 \end{array}$$

el resultado se obtiene fácilmente restando de quince (la base menos uno) cada uno de los dígitos del número y luego sumando 1 al dígito del extremo derecho.

Al sumar el complemento al minuendo se tiene:

$$\begin{array}{rrrrrr} & F & F & F & F & F \\ + & 5 & 4 & 3 & 2 & 2 \\ \hline & 5 & 4 & 3 & 2 & 1 \end{array}$$

que es el resultado pedido.

b) *Representación de datos*

En todos los computadores se utiliza el modo binario para representar información, aun cuando la estructura de los datos e instrucciones sea distinta y la unidad de información sea otra.

La unidad de información que se considerará a continuación será el BYTE, que consta de ocho bits de información más un bit de paridad. Dado que al bit de paridad no tiene acceso el usuario, se contemplarán solamente los ocho bits de información cuya representación será la siguiente:

BYTE

i) *Representación en un byte.* En un byte se puede representar:

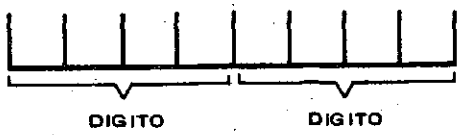
- un carácter alfanumérico
- dos dígitos decimales o hexadecimales
- un número expresado en binario puro

Para representar un carácter alfanumérico, el byte se considera dividido en dos grupos de cuatro bits cada uno, designándose la primera mitad como PARTE ZONA del carácter y la segunda como PARTE DIGITO. Esta estructura se conoce como FORMATO ZONA CARACTER (ZONA–DIGITO o ZONA–NUMERO).

Ejemplos:

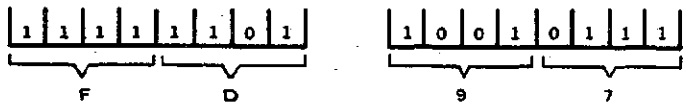
Representación de los caracteres 1 y D usando el código EBCDIC

Para representar dígitos decimales o hexadecimales, también es necesario considerar el byte subdividido en dos grupos de cuatro bits cada uno y en ellos se estructuran los dígitos

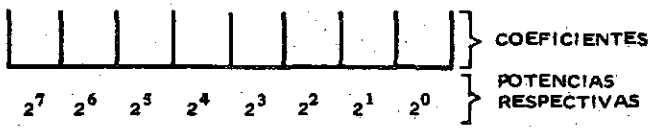


Ejemplos:

Representación de los pares de dígitos FD y 97

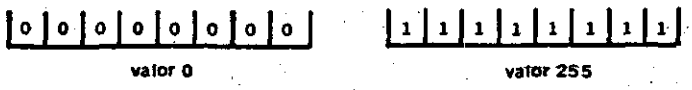


Para representar un número en binario puro debe considerarse una sumatoria en que cada uno de los bits es coeficiente de una potencia de 2 y éstas aparecen en el mismo orden que en el sistema binario, esto es, 2^0 en el extremo derecho y 2^7 en el extremo izquierdo. De acuerdo con el valor de las potencias se designa a los bits de la derecha como "bits de orden inferior" y a los de la izquierda como "bits de orden superior".



Ejemplos:

Representación de los valores 0 y 255



ii) Representación de información numérica entera (punto fijo). Para representar información numérica entera, los bytes se pueden juntar en grupos de:

– dos bytes para formar una media palabra (HALFWORD – H)
– cuatro bytes para formar una palabra (FULLWORD – F)

En ambos casos el bit de orden superior representa el signo. Si la cantidad es positiva, el bit mencionado tendrá valor cero y si la cantidad es negativa, este bit tendrá valor uno. Los bits restantes corresponden a coeficientes de potencias de 2, tal como en la representación de un número binario puro. Las cantidades negativas se representan en complemento a dos.

Ejemplos:

MEDIA PALABRA

UNA PALABRA

en el bit 0 se representa el signo.

Representar el valor 255 en media palabra y -127 en una palabra.

valor 255

valor -127

iii) Representación de información numérica real (punto flotante). Para representar información numérica real, los bytes se pueden juntar en grupos de:

– cuatro bytes para obtener precisión simple
– ocho bytes para obtener doble precisión
– dieciséis bytes para obtener precisión ampliada

en los tres casos, la representación del dato se realiza a base de una CARACTERISTICA (primer byte) y una MANTISA (bytes restantes). Para hacer más fácil la comprensión, considérese el ejemplo siguiente:

el valor $(182.35)_{10}$ puede expresarse también como:

$$\underbrace{0.18235}_{\text{MANTISA}} * \underbrace{10^3}_{\text{EXPONENTE}}$$

en igual forma el valor $(-0.00045)_{10}$ puede expresarse como:

$$\underbrace{-0.45}_{\text{MANTISA}} * \underbrace{10^{-3}}_{\text{EXPONENTE}}$$

En ambos casos se dice que el dato está normalizado, esto es, los dígitos significativos están inmediatamente después del punto decimal. La MANTISA es una fracción y el EXPONENTE representa el desplazamiento del punto decimal, hacia la derecha cuando es positivo y hacia la izquierda cuando es negativo; al mismo tiempo indica la cantidad de lugares que deben desplazarse.

Para obviar el problema de la representación del signo del exponente se define la CARACTERISTICA, que es igual a la suma de un valor constante positivo más el valor del exponente. De esta forma los exponentes positivos aumentan el valor de la característica y los negativos lo disminuyen, siempre manteniendo como punto de referencia el valor constante que corresponde al exponente cero.

Con esta definición y eligiendo el valor 50 como constante, la representación de los valores $(182.35)_{10}$ y $(-0.00045)_{10}$ será:

0.18235	53	−0.45	47
MANTISA	CARACTERIS-TICA	MANTISA	CARACTERISTICA

En la memoria de bytes, para representar los datos reales se utiliza el sistema hexadecimal. Esto significa que la fracción normalizada debe ser hexadecimal y el exponente debe corresponder a una potencia de dieciséis. Al mismo tiempo, la constante usada para formar la característica es 64. Finalmente, en memoria todo debe quedar en binario, esto es, la característica y los dígitos hexadecimales que forman la mantisa.

Ejemplos:

Representar en punto flotante el valor $(182.35)_{10}$

Conversión de la parte entera

$$\begin{array}{ccccc} 0 & \leftarrow & 11 & \leftarrow & \underline{182:16} \\ \downarrow & & \downarrow & & \\ B & & 6 & & \end{array}$$

Conversión de la parte fraccionaria

$$\begin{array}{ccccccccc} 16 * \underline{0.35} & \rightarrow & 6 & \rightarrow & 6 & \rightarrow & 6 & \rightarrow & 6 \\ & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ & & 5 & & 9 & & 9 & & 9 \end{array}$$

$$(182.35)_{10} = (B6.5999)_{16}$$

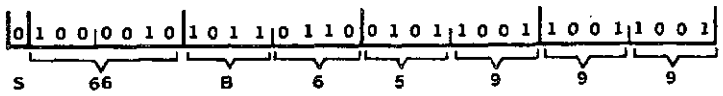
$$B6.5999 = 0.B65999 * 16^2$$

luego,

CARACTERISTICA = 64 + 2 = 66

$(66)_{10} = (1000010)_2$

y en memoria se tendrá:



En doble precisión se aumenta la cantidad de dígitos hexadecimales de la mantisa a *catorce* y en precisión ampliada a *veintiocho*, valor que se obtiene al contabilizar los dígitos del resultado de la multiplicación de dos cantidades de doble precisión con el máximo de dígitos significativos cada una.

iv) *Representación decimal (dígitos empaquetados).* La gran mayoría de los problemas llamados comerciales, de negocios, o administrativos, como es el caso de liquidación de sueldos y salarios, facturación, descuento de documentos bancarios, etc., trabaja con la representación decimal, en la que se almacenan los valores con que se procesa de tal forma que en el byte del extremo derecho queda un dígito y el signo y en los bytes restantes, dígitos empaquetados, esto es, dos por byte. La estructura es la que se indica a continuación:

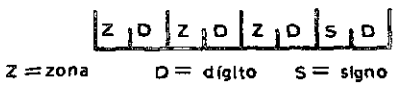
Ejemplos:

Representación de los valores –453 y 157893

obviamente, cada dígito queda registrado internamente en binario, al igual que los signos - y +. El primero tiene la misma configuración que el dígito hexadecimal D y el segundo la del dígito C.

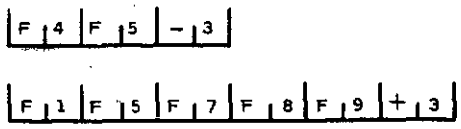
Este tipo de representación, que se conoce como FORMATO EMPAQUETADO, permite una longitud máxima de dieciséis bytes; por lo tanto, se puede almacenar una cantidad máxima de 31 dígitos decimales más el signo.

Otra estructura de datos es la conocida como FORMATO ZONA (difiere del formato ZONA–CARACTER en la estructura del byte de orden inferior) que tiene la forma siguiente:



Ejemplo:

Representación de los valores –453 y 157893



Nótese que la representación de los dígitos es la misma que en el formato carácter, luego, la diferencia entre ambos formatos está en el conjunto y en éste corresponde al byte del extremo derecho en que aparecen almacenados signo y dígito. En el formato carácter se tendría en un byte el dígito y en otro el signo.

v) *Datos lógicos.* Se denominan así los caracteres en general cuyo formato es el FORMATO ZONA CARACTER visto anteriormente. Existen instrucciones que permiten manejar desde un byte hasta dieciséis millones de bytes en una sola operación, como es el caso de transferencia de datos desde un lugar a otro de la memoria.

B. Dispositivos de Almacenamiento

Existen dos tipos de almacenamiento:

Memoria: principal, directa, primaria o de trabajo

Memoria: auxiliar, secundaria o de respaldo.

Al primer tipo pertenecen los núcleos magnéticos, películas magnéticas y el sistema monolítico puesto en uso por la IBM en el Sistema/370. Al segundo corresponden los discos magnéticos, el tambor magnético, las tarjetas magnéticas y también se puede considerar la cinta magnética cuyas características se vieron anteriormente.

a) *Memoria de trabajo*

Se dijo al comienzo del capítulo que en la Unidad de Almacenamiento se guarda toda la información que permite realizar un proceso como asimismo los resultados obtenidos de él. Esto es válido para todos los dispositivos de almacenamiento, cualquiera que sea su tipo, pero muy en particular para la memoria de trabajo, pues en ella es donde se guarda el programa de instrucciones que permitirá efectuar el proceso, como también a ella llegarán los datos, sea de almacenamiento externo (tarjetas, cinta perforada, etc.) o de memorias auxiliares (disco, tambor, etc.) y de ella saldrán los resultados al exterior o a las memorias auxiliares.

La importancia de la memoria de trabajo se refleja en el hecho de que algunas de sus características le confieren mayor o menor potencia

al computador en general. Esas características son: tamaño, velocidad, dirección, modo de operación y elementos de representación.

i) *Tamaño:* está determinado por la cantidad de caracteres que puede contener y es variable de un computador a otro. Es posible incrementar el tamaño de la memoria de acuerdo con las necesidades de uso del sistema de PED que se vayan produciendo, para lo que la construcción se efectúa a base de módulos y éstos se miden con la unidad K, que representa 1000 caracteres (excepto en algunos sistemas como el /360/370 en que K representa 1 024 caracteres o bytes). El incremento, en todo caso, tiene un límite y ese límite determina si el computador es pequeño, mediano o grande.

ii) *Velocidad:* la velocidad es la característica que está íntimamente ligada con el avance tecnológico. Ella se refiere a la rapidez con que llega una instrucción o un dato desde la memoria a la unidad de control. A esto se denomina "tiempo de acceso".

Tomando en cuenta que durante todo el proceso estarán moviéndose instrucciones y datos desde la memoria a la unidad de control y viceversa, el tiempo de acceso es un factor importante en la velocidad total de la Unidad Central de Proceso (UCP). Este es el motivo por el cual las memorias tienden cada vez a ser más compactas para lograr en esta forma mayor velocidad, producto de un camino más corto por recorrer.

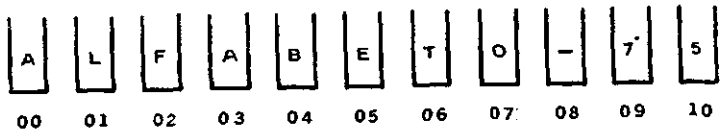
iii) *Dirección:* para poder utilizar la información almacenada en memoria, es necesario que pueda ser ubicada rápidamente y sin mayor dificultad. La única forma de conseguir esto es que cada dato y cada instrucción queden guardados en lugares que sean identificados fácilmente o, lo que es lo mismo, que tengan una dirección.

Es bastante común, y no por eso menos útil, el símil que se hace entre la memoria de trabajo de un computador y el sistema de casillas utilizado en correos. En el sistema de casillas, cada una de ellas tiene un número de orden o "dirección" con la cual se puede tener acceso rápidamente a la información que se encuentra en su interior. Nótese que esta dirección es totalmente independiente de la información que hay en la casilla, que pueden ser cartas, revistas, documentos, etc. De la misma manera, en la memoria de trabajo la dirección es independiente de lo que haya almacenado, que pueden ser números, letras, caracteres especiales o combinaciones de ellos.

Ahora bien, la estructura de la "casilla" de la memoria dependerá del computador. En relación a esto se puede decir que hay dos métodos de dirección que inciden en la estructura. En uno, la dirección se refiere a la ubicación de un carácter y en el otro se refiere a varias posiciones consideradas como un todo.

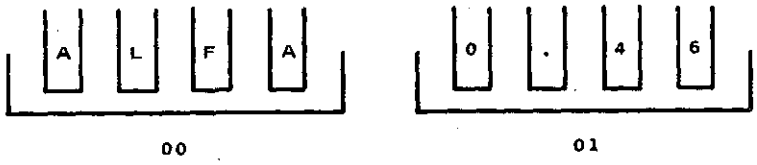
Las figuras que siguen muestran en forma gráfica los dos métodos:

– dirección de un carácter



contenido de la dirección (celda) 04 = B
contenido de la dirección (celda) 08 = –

– dirección de un conjunto de caracteres



contenido de la dirección (celda) 00 = ALFA
contenido de la dirección (celda) 01 = 0.46

Al conjunto de caracteres se le denomina PALABRA. Nuevamente, de acuerdo con el computador, se tiene: palabra de longitud fija y palabra de longitud variable. No obstante lo anterior, cuando se trata de palabra de longitud fija se dirige al conjunto. Si la palabra es de longitud variable, cada carácter del conjunto tiene dirección, pero sólo se hace referencia al primero de ellos (dirección de orden menor o carácter del extremo izquierdo) o al último. Para indicar la longitud de cada conjunto también hay varias formas: un campo al comienzo del conjunto que indica su longitud, carácter especial que indica el término de él, último carácter de conjunto con marca especial, código de la instrucción que opera sobre el conjunto que indica, además, la longitud de éste, campo en la instrucción que indica longitud, registro especial, etc.

iv) *Modo de operación:* se refiere a la forma de transferencia de información que puede ser en *serie* (carácter por carácter) o en *paralelo* (un bloque o una palabra cada vez). Evidentemente, la transferencia en paralelo proporciona mayor velocidad en los procesos.

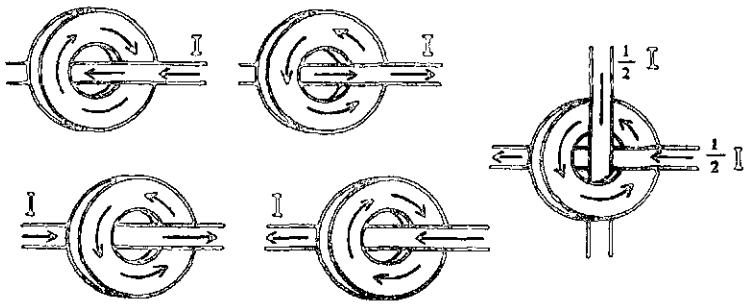
v) *Elementos de representación: Núcleos magnéticos:* un núcleo magnético es un pequeño anillo de material ferromagnético de 0.08 pulgada de diámetro y 0.025 pulgada de espesor.

Aparte de su tamaño compacto, que fue una decidida ventaja en el diseño de computadores, la característica más importante del núcleo es que puede ser fácilmente magnetizado en unas pocas millonésimas de

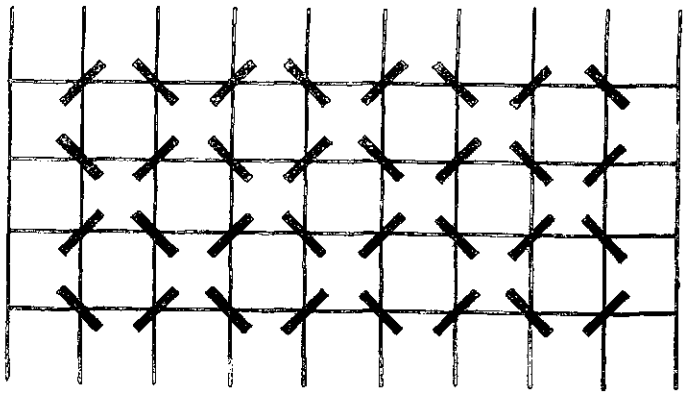
segundo y, a menos que sea cambiado deliberadamente, retiene su magnetismo durante un tiempo indefinido.

Si se hace pasar un alambre a través de los núcleos y se envía por él suficiente corriente eléctrica, los núcleos serán magnetizados. La dirección de la corriente determina la polaridad o estado magnético del núcleo. Si se invierte su sentido, cambia también el estado magnético. Se tiene así un elemento biestable cuyos estados pueden usarse para representar las condiciones 0 ó 1, ausencia o presencia de información.

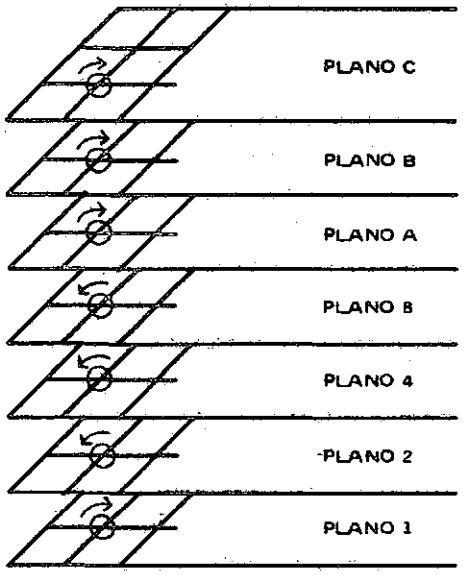
Si se hace pasar dos alambres a través de cada núcleo, de tal manera que formen un ángulo recto entre ellos, y se hace pasar por cada uno la mitad de la corriente necesaria para magnetizar un núcleo, sólo se magnetizará aquel que se encuentre en la intersección de los alambres y ningún otro núcleo es afectado.



Usando este principio, se puede colocar un gran número de núcleos en una malla de alambre, así, cualquiera de ellos puede seleccionarse para grabación o lectura sin afectar a los restantes. Tal arreglo de alambres y núcleos recibe el nombre de *plano de núcleos.*



Si se desea almacenar información en el código Decimal Codificado en Binario, serán necesarios siete planos de núcleos dado que a cada plano se le asigna un valor, incluido un plano de paridad. La figura siguiente muestra el carácter A registrado en los siete planos:



CARACTER A

Nótese que los núcleos que forman el carácter A están colocados en la intersección de los mismos dos alambres de cada plano, de ahí que si se traza una línea vertical imaginaria a través de esos núcleos se tendrá la ubicación física de un carácter almacenado en la memoria, o lo que es lo mismo, la estructura de una celda.

Una vez que la información ha sido almacenada en la memoria, debe proveerse algún medio para hacerla accesible, esto es, sacarla o leerla cuando se la necesita. Se ha visto que una polaridad magnética definida puede ser registrada en un núcleo mediante el flujo de corriente a través de un par de alambres que lo crucen, y que es enviado en forma de pulso eléctrico. Este pulso cambia el estado del núcleo a positivo o negativo, de acuerdo con la dirección que tenga el flujo de la corriente.

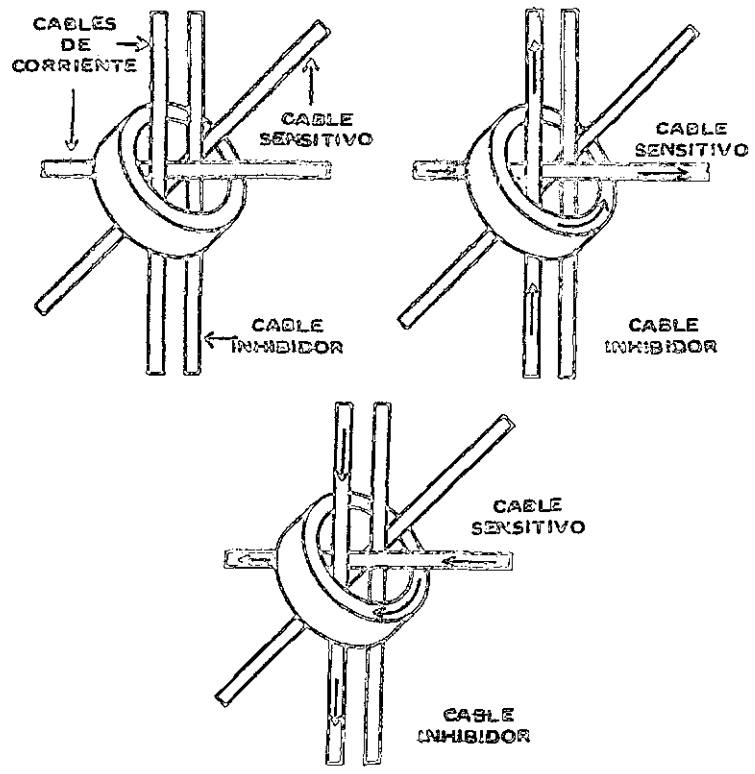
Si el estado magnético del núcleo cambia por acción del pulso, este cambio induce corriente en un tercer alambre que pasa a través del núcleo, denominado alambre o cable SENSITIVO, cuya corriente puede

detectarse para determinar si el núcleo está magnetizado en uno o en otro sentido. En otras palabras, si tiene un 1 o un 0.

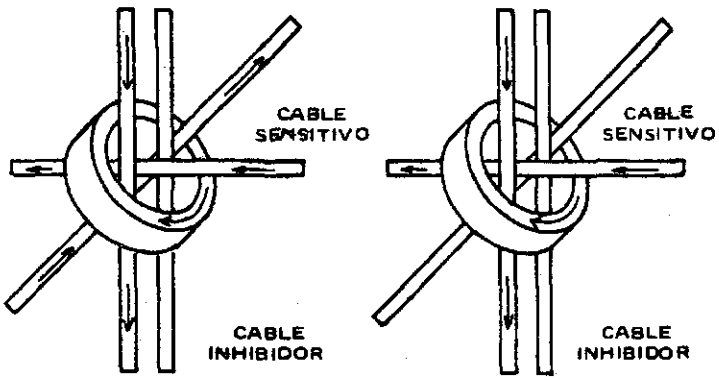
Se necesita un solo cable sensitivo para todo un plano de núcleos, ya que en cualquier plano será examinado sólo un núcleo a la vez para conocer su estado magnético. El alambre pasa por todos los núcleos del plano.

Es de hacer notar, sin embargo, que la lectura es destructiva, esto es, el proceso de lectura de un 1 cambia el núcleo a 0. Así, el computador debe reponerlo, como también dejar en 0 el núcleo que contenía 0.

Para reproducir o regenerar los 0 y 1 tal como estaban antes de la lectura, el computador trata de regrabar un 1 en todas las ubicaciones previamente leídas; al mismo tiempo, un pulso inhibidor suprime la grabación en aquellos núcleos que previamente contenían 0. El pulso inhibidor es enviado a través de un cuarto alambre y, en efecto, anula el pulso grabador en uno de los dos cables usados para magnetizar el núcleo. El cable inhibidor, tal como el cable sensitivo, pasa por todos los núcleos del plano.

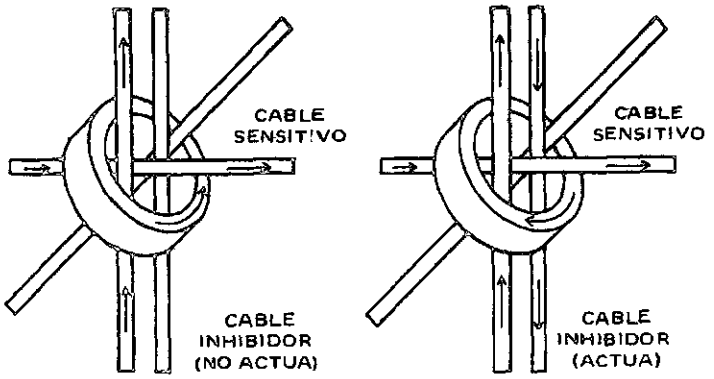


En las figuras que aparecen a continuación, se tiene grabado un bit 1 y un bit 0. En ambos casos, el cable inhibidor y el cable sensitivo no actúan, sólo lo hacen los cables utilizados para obtener la lectura del estado de un núcleo. La máquina da la orden de grabar un cero en el núcleo cuyo estado se desea detectar. Si el núcleo contiene un cero y se da orden de grabar allí un cero, el núcleo permanece en su estado original y el cable sensitivo encargado de transmitir el estado del núcleo no recibe pulso eléctrico, lo que el computador interpreta como que el núcleo está en estado cero. Por el contrario, si el núcleo está en estado 1 y se graba en él un cero, el cambio de estado producido induce a su vez corriente en el cable sensitivo, lo que es interpretado por el computador como un estado 1 en el núcleo en referencia. En este último caso, sin embargo, luego de leer correctamente el 1, el núcleo quedó con un 0 *que no era el contenido original.*



BIT 0 (EL NUCLEO TENIA UN 1) BIT 0 (EL NUCLEO TENIA UN 0)

Es necesario devolver los núcleos a sus estados originales. Así como en el proceso de lectura el computador da orden de grabar 0 en el núcleo que se va a leer, ahora se invierte el proceso y se da orden de grabar un 1 en el núcleo afectado por la lectura. Con el objeto de que se grabe efectivamente un 1 cuando corresponde, si el cable sensitivo recibió un pulso eléctrico en la etapa de lectura, en esta etapa de grabación el cable inhibidor no actúa, en cambio sí lo hace cuando el cable sensitivo no había recibido pulso. Por el cable inhibidor pasa la mitad de la corriente necesaria para magnetizar el núcleo en sentido contrario al que tiene la corriente de una de las coordenadas que trata de grabar el 1. Se anula así el efecto de la orden y en el núcleo queda un 0.



BIT 1 (REINTEGRADO) BIT 0

Como en todo dispositivo magnético, cuando se graba información en los núcleos se borra lo que había en ellos. Por el contrario, cuando se lee la información, ella permanece inalterable, lo que permite utilizarla todas las veces que se desee.

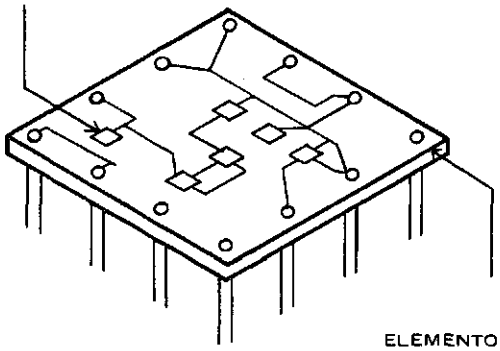
Película magnética: la película magnética, más conocida como película delgada, consiste en depósitos planos muy delgados construidos con una aleación de níquel y hierro. Estos planos metálicos están conectados por alambres ultradelgados y montados en una base aisladora tal como vidrio o plástico.

Se puede usar también en forma de alambre recubierto. La película magnética está enrollada alrededor de un alambre hecho normalmente de berilio y cobre.

La operación, en todo caso, es similar a la de los núcleos magnéticos,ya que en ambos casos se usan los planos para ubicación de los elementos.

Sistema monolítico: se utilizan los mismos conceptos de la lógica monolítica, esto es, se hace uso de un elemento de cerámica de media pulgada cuadrada con interconexiones de metal sobre las cuales se colocan recortes o pequeñas porciones de silicón. Sin embargo, en el caso del almacenamiento monolítico, en vez de implementar circuitos lógicos en los recortes de silicón, se diseñan las celdas utilizadas para contener bits de almacenamiento.

PORCION DE SILICON
CON COMPONENTES



ELEMENTO DE
CERAMICA CON INTER-
CONEXIONES

Una de esas porciones de silicón es de aproximadamente un octavo de pulgada cuadrada y puede contener 128 bits de almacenamiento y su conjunto de circuitos asociado para decodificación, dirección y consultas. Se montan dos porciones de arreglo de almacenamiento en un elemento de media pulgada cuadrada, de los cuales, un par forma un módulo de arreglo de almacenamiento. Cada módulo contiene 512 bits y se monta en una tarjeta de 3 1/2 pulgadas de alto por 4 3/4 de ancho, que contiene 12 K bits.

Las tarjetas, a su vez, se empaquetan en módulos de almacenamiento básico (BSM) de 13 1/4 pulgadas de largo por 5 1/2 de alto y 9 de ancho, que contiene 48 K bytes de almacenamiento y su conjunto de circuitos asociado.

Las ventajas del sistema monolítico sobre la memoria de núcleos son las siguientes:

Se puede obtener mayor velocidad de almacenamiento debido al camino más corto entre circuitos de almacenamiento y además por la *capacidad de lectura no destructiva* del sistema monolítico. Se vio anteriormente en la memoria de núcleos que se necesita un ciclo de regeneración de la información después de efectuada una lectura. Este ciclo no es necesario en el almacenamiento monolítico.

Las tarjetas son fácilmente reemplazables, lo cual permite que los incrementos de memoria puedan también ser instalados rápidamente.

El requerimiento de espacio es menor.

vi) *Memoria virtual:* es un espacio de memoria destinado a direcciones (espacio de dirección) que pueden ser utilizadas por un programa para referirse a instrucciones y datos.

Hay que distinguir entre espacio de dirección (memoria virtual) y el espacio de almacenamiento real (memoria principal). El primero es

un conjunto de identificadores o nombres y el segundo, de ubicaciones de memoria física en el cual deben ser colocados los datos e instrucciones para que sean procesados por la Unidad Central de Proceso (UCP).

La memoria virtual se llama así porque representa una "imagen de memoria" más bien que memoria principal. Dado que la memoria virtual no existe como memoria física, las instrucciones y datos a los cuales se refieren sus direcciones, que son los *contenidos de memoria virtual*, deben estar almacenados en alguna ubicación física. Para ello se dividen en dos partes: una que está siempre presente en almacenamiento real y otra que no lo está. Esta última debe estar en alguna ubicación desde la cual puede ser llevada a memoria principal para ser procesada por la UCP. Este requerimiento se logra usando almacenamiento de acceso directo. Además, es necesario un mecanismo que permita asociar las direcciones de memoria virtual de instrucciones y datos contenidos en memoria de acceso directo, con sus ubicaciones actuales en memoria principal cuando esos datos estén siendo procesados por la UCP. Este mecanismo es un dispositivo de traducción de dirección dinámico, ubicado en la UCP.

Con el concepto de memoria virtual, un sistema puede soportar un espacio de direcciones bastante mayor que el tamaño de la memoria principal que tenga el computador, dado que las instrucciones y datos se llevan a la memoria de trabajo sólo cuando van a ser usados y se retornan a la memoria de acceso directo cuando se necesita el almacenamiento real que usaron y ya no volverán a utilizarse, dejando así el espacio libre para otra información.

vii) *Máquina virtual:* es una simulación funcional de un sistema completo de computador, incluyendo una Unidad Central de Proceso (UCP) virtual, almacenamiento virtual, canales virtuales, dispositivos ENTRADA/SALIDA virtuales y una consola de operador virtual, todo lo cual representa para el usuario una máquina real.

Un componente llamado Programa de Control (PC) soporta operaciones paralelas de multiprogramación, que permiten que los recursos de una máquina real sean compartidos por múltiples máquinas virtuales, cada una de las cuales, y la planificación del trabajo que ella realiza, son manejados por un sistema de operación más bien que por el PC. Esto es, cada máquina virtual tiene un conjunto de programas que conforma un sistema de operación que asigna recursos de máquina y planifica la ejecución de programas tal como si el sistema de operación estuviere ejecutándose en una máquina real.

b) *Memoria auxiliar*

Permite almacenar una cantidad mucho mayor de información que la

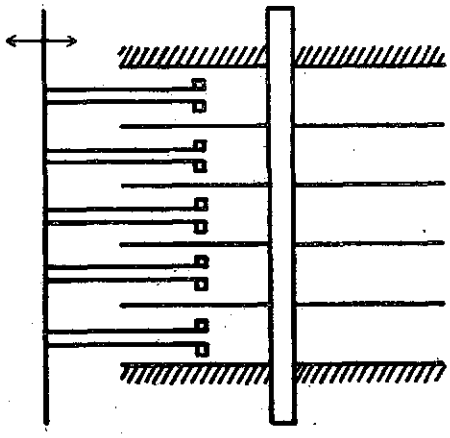
memoria de trabajo. Proporcionalmente, el costo es más bajo que el de ésta, sin embargo, el tiempo de acceso es mucho mayor.

Para que la información sea registrada en memoria auxiliar, normalmente debe pasar por la memoria de trabajo, igualmente, si se desea conocer su contenido, debe llegar primero a la memoria principal y de ahí salir al exterior. Se exceptúa el caso de cinta magnética y a veces, aunque no es lo corriente, disco magnético, cuando se tienen dispositivos que permitan grabar o leer directamente en ellos.

i) *Discos magnéticos:* es un disco de metal delgado recubierto por ambos lados con un material que puede ser magnetizado. La información se registra en canales o pistas concéntricas en las cuales los bits correspondientes a un dato se graban en forma serial, esto es, un bit al lado del otro en el mismo canal.

Normalmente, los discos se juntan en grupos para formar un módulo o un paquete. El módulo corresponde al grupo de discos fijos en un dispositivo, que consta generalmente de 25 discos. Los dispositivos pueden tener uno o dos módulos de discos. Se define como paquete el grupo de discos intercambiables, esto es, el conjunto de discos que se puede remover del dispositivo tal como los carretes de cinta magnética.

Para efectuar la lectura o grabación de información, se utiliza una cabeza lectora-grabadora o un conjunto de ellas. Evidentemente, a mayor cantidad de cabezas menor el tiempo de acceso a la información. Si hay una sola de ellas, tiene que hacer tres movimientos para ir de una pista a otra que se encuentre en otro disco; primero tiene que salir del disco en el cual se encuentra, a continuación subir o bajar para encontrar la cara del otro disco y enseguida entrar para colocarse en la nueva pista. Si se tiene una cabeza por cara, el movimiento es sólo horizontal, como se indica en la figura siguiente:



En este último caso, debido a que los brazos que llevan las cabezas lecto-grabadoras están montados rígidamente en un eje vertical, al situarse una cabeza frente a una pista determinada, las otras estarán a su vez frente a pistas de igual número, en las caras restantes. Esto permite poder grabar o leer información de todas las pistas paralelas, verticalmente, sin desplazar el mecanismo, con lo cual se puede hablar de un CILINDRO DE INFORMACION.

Las capacidades de almacenamiento varían desde 4 millones a 200 millones de caracteres. En el caso de utilización de paquetes intercambiables, se tienen dispositivos con capacidad para mantener hasta ocho paquetes simultáneamente, lo que hace subir la capacidad de almacenamiento a 1 600 millones de caracteres.

Los tiempos de acceso varían desde 75 a 25 milisegundos y las velocidades de transferencia desde 150 000 hasta 885 000 caracteres por segundo. Los discos se encuentran siempre girando a una velocidad que varía desde 1 500 a 2 400 revoluciones por minuto.

ii) *Tambor magnético:* ha sido utilizado prácticamente desde el comienzo de la construcción de computadores, como almacenamiento de información. Incluso en los primeros computadores se utilizaba como memoria principal.

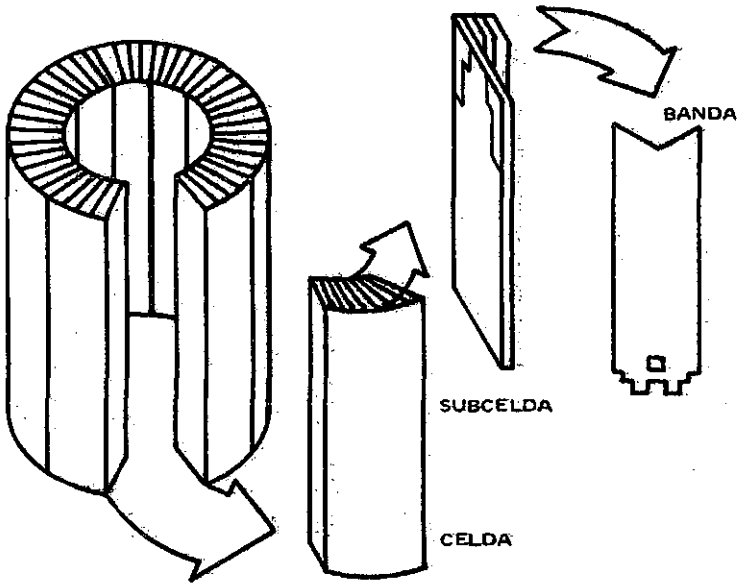
El tambor magnético es un cilindro de metal recubierto con material magnético, montado verticalmente, que gira a una velocidad de 3 500 revoluciones por minuto. Para grabar o leer información, se tienen cabezas lecto-grabadoras, una frente a cada pista existente en la superficie del cilindro. La información se graba en forma serial, esto es, un bit a continuación del otro en el mismo canal o pista. Para disminuir el tiempo de acceso, en algunos casos se utilizan dos o cuatro cabezas distribuidas por sectores, en cada pista.

Se tienen tambores con una capacidad aproximada de 4 millones de caracteres con una velocidad de transferencia de 1,25 millones de caracteres por segundo.

iii) *Tarjetas magnéticas:*

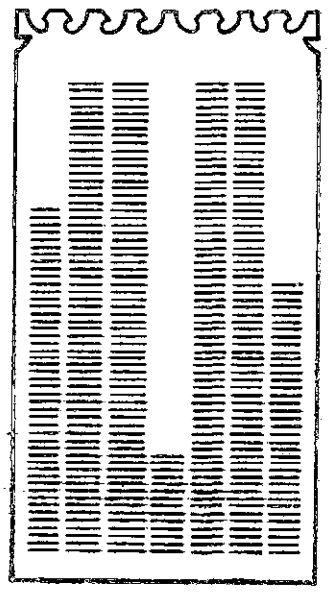
Data Cell. Esta unidad de la IBM está formada por 10 celdas intercambiables, ordenadas en forma circular, cada una de las cuales tiene 20 subceldas. Cada subcelda tiene 10 bandas magnéticas, en cada una de las cuales se tienen 100 pistas. Finalmente, cada pista tiene capacidad para un máximo de 2 000 bytes.

La banda es seleccionada de una subcelda y enrollada alrededor de un tambor giratorio que está situado bajo un bloque de cabezas lecto-grabadoras que realizan la transferencia de datos. Cuando la lectura o grabación se ha completado, la banda se devuelve a su subcelda original y ésta a su vez al arreglo de 10 subceldas. El bloque de cabezas lecto-grabadoras contiene 20 elementos magnéticos y puede situarse en cualquiera de 5 ubicaciones posibles (llamadas cilindros) proporcionando así 100 pistas de grabación por banda.



El tiempo de acceso a una lámina oscila entre 175 y 600 milisegundos, considerando la posibilidad más desfavorable, que es aquélla en que se tiene una banda en el tambor de lectura y grabación, lo que implica tener que situarla en su posición original y después llevar la nueva banda. La velocidad de transferencia es de 55 000 bytes por segundo y la capacidad es de cerca de 400 millones de bytes.

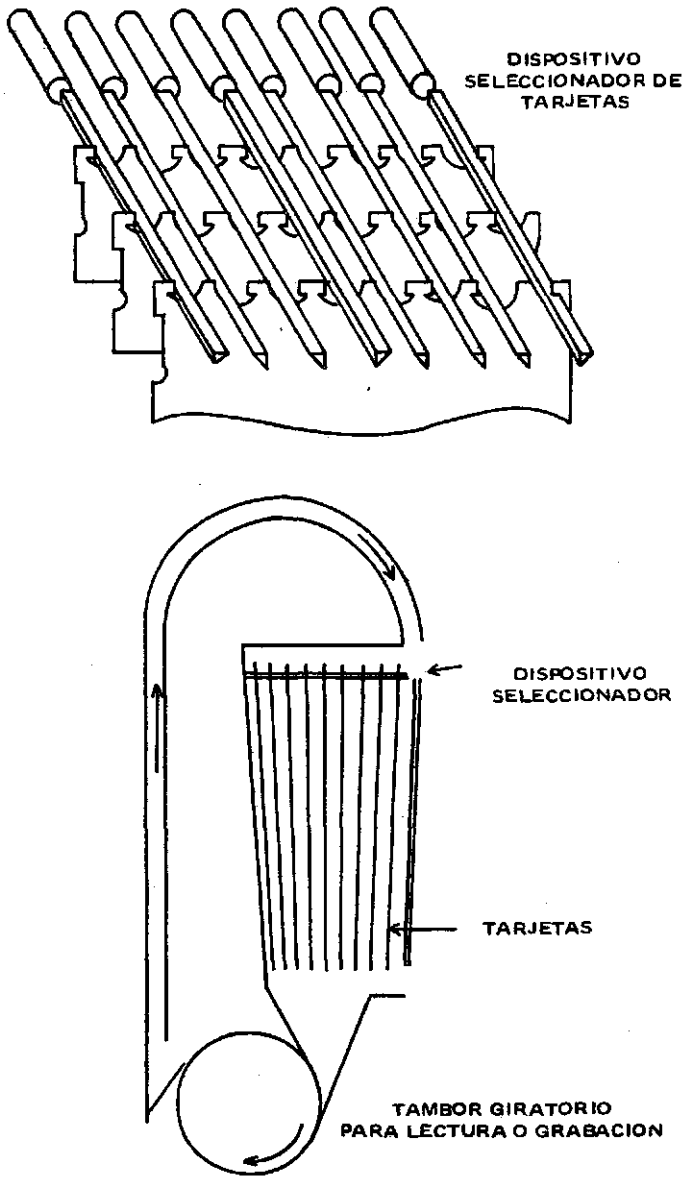
Card Random Access Memory (CRAM). Esta unidad de la National utiliza un paquete de tarjetas que contienen pistas paralelas de material magnético, en cada una de las cuales se registra la información. El número de pistas en la tarjeta y el número de tarjetas en el paquete dependen del tipo de unidad CRAM que se utilice. El computador NATIONAL NCR 315 tiene tarjetas de 3 1/4 pulgadas de ancho por 14 de largo, y la longitud de las pistas es igual a 12 1/2 pulgadas.



Tiene siete pistas, cada una con capacidad para 4 650 dígitos decimales. El paquete tiene 256 tarjetas, lo que da una capacidad de 8 332 800 dígitos.

Para seleccionar una tarjeta en la unidad CRAM, la tarjeta tiene una serie de muescas en su borde superior, las que responden en su orden a un código binario. Después de seleccionada, se pone en un tambor, en el que se graba o lee información. Concluida la operación, se retorna al paquete, aprovechando la fuerza centrífuga que la impulsa por un camino de regreso.

La velocidad de transferencia en la unidad CRAM, utilizada en el NCR 315, es de 150 000 dígitos por segundo.


DISPOSITIVO
SELECCIONADOR DE
TARJETAS
DISPOSITIVO
SELECCIONADOR
TARJETAS
TAMBOR GIRATORIO
PARA LECTURA O GRABACION

C. *Unidad Central de Proceso*

La Unidad Central de Proceso (UCP) está formada por la Unidad Aritmético-Lógica y la Unidad de Control. Se puede decir entonces que la UCP es el cerebro de un sistema EPD, pues en ella es donde se realiza el control de todas las unidades que componen el sistema, además de efectuar las operaciones aritméticas y lógicas del proceso, que implican transferencia y análisis de las instrucciones y, posteriormente, movimiento de los datos a la unidad aritmético-lógica y de los resultados obtenidos en ella.

Antes de hacer un análisis simple de los principales elementos que se utilizan para el funcionamiento de la UCP, se verán algunos conceptos del álgebra de Boole.

a) *Algebra de Boole*

Para describir los circuitos utilizados para efectuar las operaciones aritméticas y otras, como representación interna de la información que llega del exterior (codificación) e interpretación de la información almacenada para enviarla fuera del sistema (decodificación), se utiliza normalmente el álgebra de Boole, por la simplificación que ella permite en su representación, como también porque constituye un lenguaje fácil de comunicación entre los proyectistas.

El lenguaje básico y las leyes del álgebra booleana fueron expuestos por George Boole en 1854. Utiliza tres operaciones:

AND representada por el símbolo $\wedge$ $\cap$

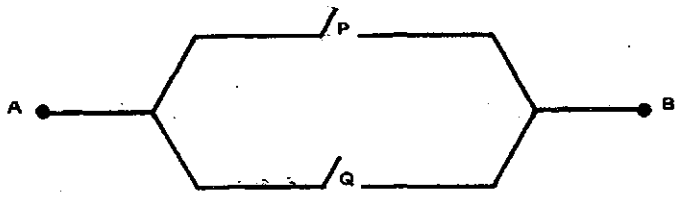
OR representada por el símbolo $\vee$ $\cup$

NOT representada por el símbolo $-$, que se coloca encima del símbolo negado.

El resultado de las operaciones lógicas es VERDADERO o FALSO, que en un circuito puede corresponder a PRESENCIA o AUSENCIA de información (uno o cero).

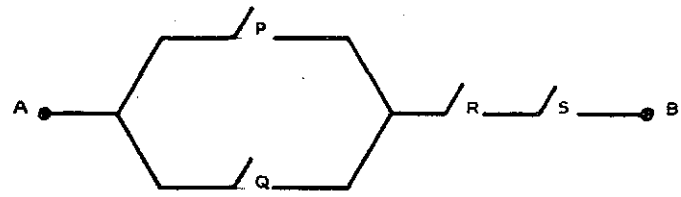
A continuación se tienen algunos circuitos sencillos representados también con el álgebra de BOOLE.

Para que circule corriente del punto A al punto B, deben estar cerrados los interruptores P y Q. Si ambos son VERDADEROS, el resultado es VERDADERO; en cualquier otro caso será FALSO.



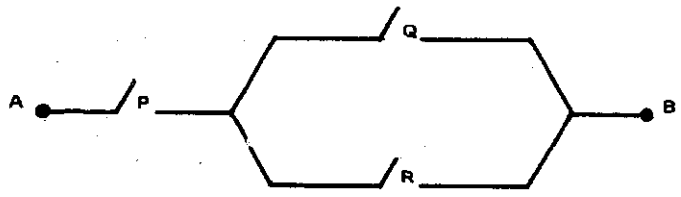
(P∪Q)

Para que circule corriente del punto A al punto B, debe estar cerrado el interruptor P o el Q. Si uno de ellos es VERDADERO, el resultado es VERDADERO. Ambos deben ser FALSOS para que el resultado sea FALSO.



(P∪Q) ∩ (R∩S)

Para que circule corriente del punto A al punto B, debe estar cerrado el interruptor P o el Q y, además, R y el S. Para obtener resultado VERDADERO, P o Q deben ser VERDADEROS y R y S deben ser VERDADEROS.



P∩(Q∪R)

Para que circule corriente del punto A al punto B, deben estar cerrados los interruptores P y el Q o el interruptor R. Si P es VERDADERO y P o R son VERDADEROS, el resultado es VERDADERO.

La ecuación $P \cap (Q \cup R)$ corresponde a una simplificación de la ecuación $(P \cap Q) \cup (P \cap R)$, en la que se sacó factor común P.

b) *Unidad Aritmético-Lógica*

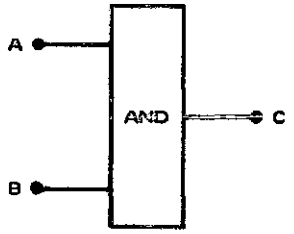
Las operaciones que se realizan en esta unidad corresponden a cálculos aritméticos en que se hace uso de la Aritmética de Punto Fijo (operaciones entre números enteros), de la Aritmética de Punto Flotante (operaciones entre números reales) y de la Aritmética Decimal (operaciones entre números empaquetados). Además, se efectúan operaciones lógicas como: desplazamiento de bits dentro de un campo, comparación de operandos lógicos (caracteres), edición de datos, conversión de datos, etc.

Dado que las direcciones de datos e instrucciones raras veces están en forma absoluta y, por el contrario, se representan a base de dos o más elementos, el cálculo que es necesario realizar para obtener la dirección definitiva también se realiza en esta unidad.

Los circuitos que se utilizan para obtener las operaciones mencionadas son bastante complejos, de manera que sólo se expondrán las ideas básicas que permitan formarse una idea del mecanismo interno del computador.

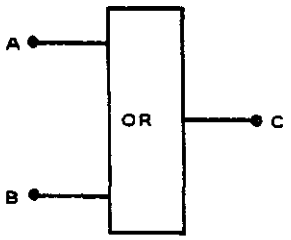
Para simplificar la representación de los circuitos se hace uso del concepto "puerta". Se tienen tres clases de puertas: puerta AND, puerta OR y puerta INVERTER. Las dos primeras tienen dos o más alambres de entrada y uno de salida, y la INVERTER tiene uno de entrada y uno de salida. El flujo de corriente se designará por 1 y el no flujo por 0.

En la puerta AND, para que haya un 1 en el alambre de salida debe haber un 1 en cada alambre de entrada.



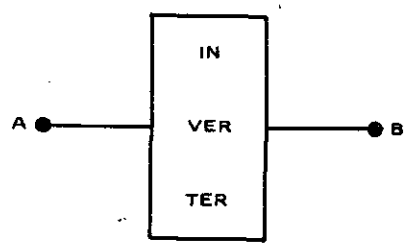
A	0	1	0	1
B	0	0	1	1
C	0	0	0	1

En la puerta OR, para que haya un 1 en el alambre de salida debe haber un 1 en uno de los alambres de entrada.



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

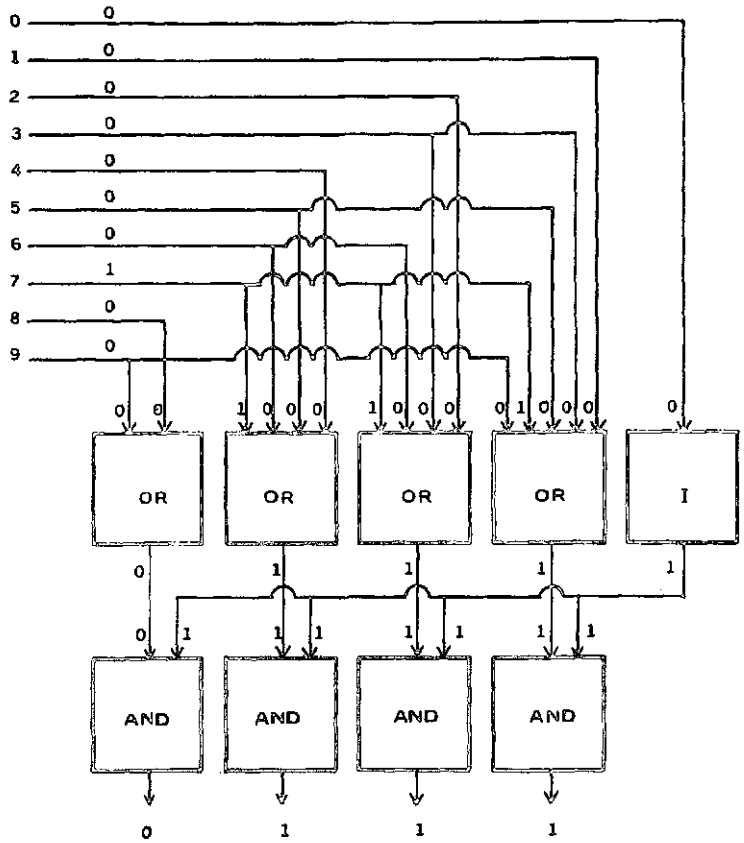
En la puerta INVERTER se efectúa, como su nombre lo indica, una inversión (negación) de la información que entra.



A	0	1
B	1	0

A continuación se ven algunos ejemplos en los que se aplican las puertas AND, OR e INVERTER:

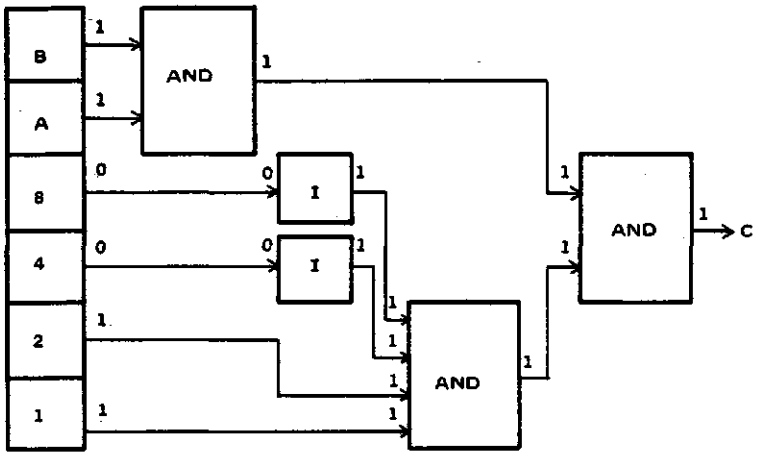
Conversión de los dígitos decimales a binario (grupo de cuatro núcleos)



La corriente que pasa por el alambre que corresponde al dígito 7 se bifurca y entra a las tres puertas OR del extremo derecho o, lo que es lo mismo, entra un 1 por ellas y debe salir, por lo tanto, un 1. Por la puerta OR del extremo izquierdo ha entrado un 0 y salido un 0, en cambio en la puerta I entró un 0 y salió un 1.

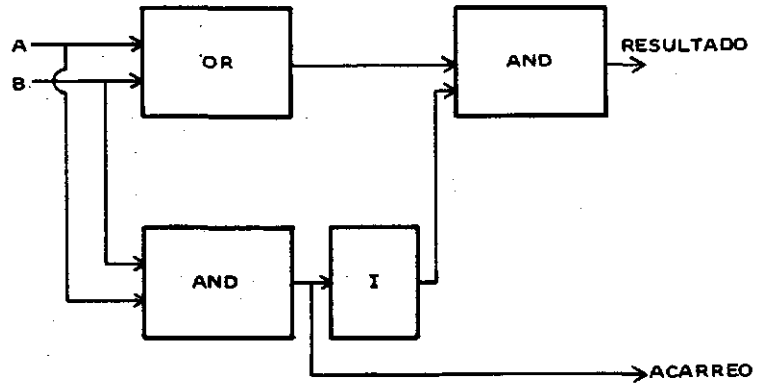
Pasando a las puertas AND, se observa que en las únicas en que entran dos unos es por las del extremo derecho, luego por ellas sale un 1; en cambio, en la del extremo izquierdo entran un 0 y un 1 y luego sale un 0.

Decodificación de un carácter en código DCB.



Considérese en este ejemplo que se tiene almacenada la letra C. El circuito que permite decodificar dicho carácter es el mostrado en la figura, esto es, cada carácter tendrá su circuito de decodificación.

Suma de dos dígitos binarios A y B



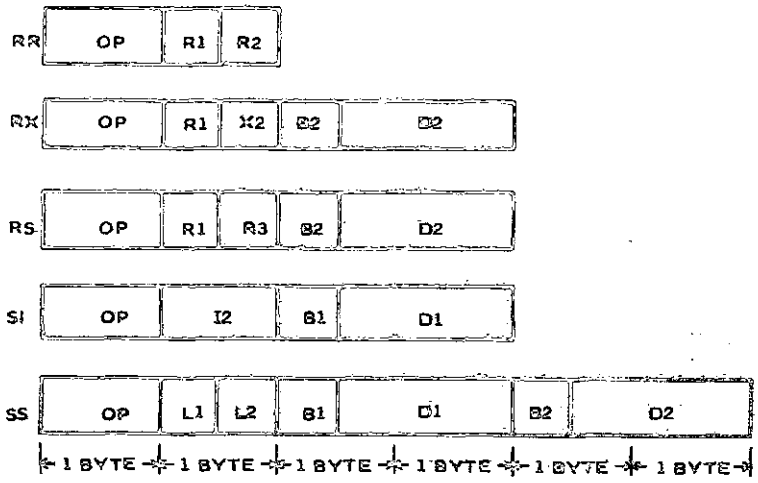
Este circuito se denomina "medio sumador" y permite sumar dos dígitos binarios. Se considera el bit de acarreo que se produce cuando ambos sumandos tienen valor 1.

c) *Unidad de control*

Esta unidad es la que "seleccionará" los circuitos necesarios para ejecutar todas las operaciones mencionadas anteriormente. Para ello tiene que analizar y ejecutar la totalidad o parte de un conjunto de instrucciones denominado "programa". Cada instrucción contiene un código que le indicará a la unidad de control cuál es la operación que debe efectuarse; al mismo tiempo le señalará cuáles son los operandos que intervienen y qué características tienen o cuáles son los dispositivos que se utilizarán.

Las instrucciones se pueden agrupar en la forma siguiente:
Instrucciones de entrada-salida
Instrucciones aritméticas
Instrucciones lógicas
Instrucciones de bifurcación
Instrucciones de control del sistema

El formato de las instrucciones dependerá del computador que se esté utilizando. Como ejemplo, se da a continuación el formato de las instrucciones utilizadas en los Sistemas/360/370 de la IBM.



El código de operación, en la mayoría de las instrucciones, ocupa el primer byte (hay excepciones en que se ocupa también el segundo byte). Normalmente, las direcciones de los operandos se obtienen a base de los elementos *D*, *B* o *D*,*B*,*X* (las instrucciones en que la dirección se da en registros constituyen la excepción). En el Sistema /360 se tienen dieciséis registros de uso general con capacidad para cuatro bytes cada

uno, con los que se realizan las operaciones aritméticas de punto fijo, el cálculo de las direcciones, control de programas, etc. Se tienen, además, cuatro registros de punto flotante para las operaciones aritméticas de punto flotante, cada uno con capacidad para ocho bytes. En el Sistema /370 se tienen registros de control cuyo número depende de los requerimientos de las funciones instaladas. Las longitudes de los campos que entran en un proceso pueden darse con el elemento L de la instrucción o en registros.

De acuerdo con lo anterior, se tiene:

Formato RR operaciones de Registro a Registro

Formato RX operaciones de Registro y memoria indeXada

Formato RS operaciones de Registro y memoria (Storage)

Formato SI operaciones de memoria (Storage) y operando Inmediato

Formato SS operaciones de memoria (Storage) a memoria (Storage)

D. Organización de Archivos

a) *Conceptos generales*

i) *Registro, archivo y volumen.* Se ha visto en los capítulos anteriores que es necesario, para poder manejar la información correspondiente a un problema, que ella esté almacenada o *registrada* en elementos o medios que permitirán su transferencia a la memoria del computador. Estos medios tienen como unidad de medida común el *registro,* cuya agrupación da lugar a los *archivos*, cualquiera que sea el tipo de éstos. Los ejemplos más comunes y claros de registro-almacenamiento son la tarjeta, la línea impresa y el espacio comprendido entre dos INTER BLOCK GAP (IBG) en una cinta magnética. Este tipo de registro recibe también el nombre de *registro físico* para diferenciarlo del *registro lógico* que se estructura de acuerdo con las relaciones lógicas que existen entre los datos. En algunos casos el registro físico puede corresponder al registro lógico, pero, en general, estará formado por dos o más de ellos.

Desde el punto de vista de la utilización de los archivos, éstos pueden clasificarse en:

Archivos maestros: Son aquellos que permanecen vigentes mientras permanezca útil un sistema de información. Se actualiza cada cierto tiempo, de manera que su información corresponda a hechos o situaciones actuales. Ejemplo: archivo de personal en un sistema de remuneraciones.

Archivos de respaldo. Son aquellos a partir de los cuales es posible regenerar archivos que han sido destruidos en etapas posteriores. Como medida de seguridad, es conveniente aplicar la técnica "abuelo,

padre, hijo", de tal manera que si el archivo "hijo" es destruido, se pueda recurrir, para regenerarlo, al archivo "padre", manteniendo todavía como respaldo el archivo "abuelo".

Archivos de transacciones. Son aquéllos que se crean especialmente para una labor específica y una vez cumplida ésta no necesitan conservarse, por cuanto su información ha perdido actualidad. En algunos casos, sin embargo, se dejan como respaldo (archivo histórico) con el objeto de tener un factor de seguridad previendo la destrucción de los informes obtenidos o la necesidad de reprocesar la información. Ejemplo: archivo de despacho de materiales en un sistema de control de inventarios.

Archivos de paso. Son aquéllos que se crean durante un proceso para ser utilizados posteriormente en el mismo ciclo de operación, pero sin que permanezcan hasta el ciclo siguiente.

Archivos de trabajo. Son aquéllos que se crean durante un proceso y no son utilizados nuevamente, pues han servido exclusivamente como áreas de respaldo. Ejemplo: archivos necesarios en un proceso de clasificación/intercalación (Sort/Merge).

Archivos de reposición. Son aquéllos que se crean durante un proceso a base de todos los valores obtenidos y grabados cada cierto tiempo, de manera que sea posible reiniciar el proceso a partir de cualquiera de esas grabaciones sin tener que empezarlo todo desde el comienzo.

Archivos de informes. Son aquéllos que corresponden a respuestas del sistema EPD. Su contenido refleja los resultados del procesamiento y son utilizados por personas, por lo tanto, deben registrarse en un medio visual, normalmente el formulario continuo.

Se entiende por volumen el medio físico de almacenamiento. Se aplica principalmente a la cinta magnética en que el volumen es el carrete de cinta y al disco magnético en que el volumen es el paquete (pack) de discos.

De acuerdo con lo anterior, se pueden tener varios archivos en un volumen (multi-file-volume) o varios volúmenes que formen un solo archivo (multi-volume-file).

ii) *Tipo de acceso.* Es la forma en que se puede recuperar la información contenida en un registro determinado. Se pueden clasificar en tres grupos:

Acceso serial. Es aquél en que para recuperar la información contenida en un registro es necesario leer cada uno de los registros anteriores del archivo. Este acceso es obligado en archivos en tarjetas perforadas, en cinta de papel perforada, en cinta magnética y en cualquier archivo cuyos registros se sitúen físicamente uno a continuación del otro, en el espacio contiguo disponible.

Acceso secuencial. Es aquél en que para obtener la información

de un registro es necesario leer los registros que le preceden en el orden lógico, independiente de su ordenamiento físico.

Acceso directo. Es aquél en que la información de un registro se puede obtener en forma directa, esto es, sin necesidad de analizar previamente otros registros. Este tipo de acceso se puede obtener en dispositivos como: discos magnéticos, tambor magnético, tarjetas magnéticas; de ahí que éstos reciban también el nombre de dispositivos de acceso directo (Direct Access Storage Device-DASD).

iii) *Tipos de procesamiento.* Es la forma en que se procesan los registros de un archivo o, lo que es lo mismo, el orden en que los registros entran en una transacción. Existen tres tipos de procesamiento: serial, secuencial y al azar (random).

Serial es aquél en que los registros se procesan de acuerdo con el orden físico en que estén almacenados.

Secuencial es aquél en que los registros se procesan de acuerdo con el orden lógico en que estén almacenados.

Al azar (random) es aquél en que los registros se procesan en cualquier orden.

b) *Tipos de organización*

La organización del archivo se refiere a la estructura lógica y física que tendrán los registros en el medio de almacenamiento. Existe, por supuesto, una relación muy estrecha entre: medio de almacenamiento, tipo de procesamiento y tipo de organización.

Los tipos de organización que se utilizan en la práctica son:

i) *Secuencial:* corresponden a esta organización todos los archivos en que la disposición física de los registros es obligatoriamente uno a continuación del otro en el espacio contiguo disponible. De acuerdo con esta característica, para recuperar la información contenida en un registro es necesario leer cada uno de los registros anteriores del archivo.

Nótese que esta definición tiene correspondencia con el acceso serial, por lo cual debiera llamarse organización serial, sin embargo, ha quedado el nombre de secuencial porque ha primado la costumbre.

Es conveniente hacer notar que aun cuando para ciertos medios de almacenamiento es obligatorio el acceso serial, no lo es el usar en esos medios la organización secuencial (serial). Perfectamente se puede tener este tipo de organización en dispositivos de acceso directo.

ii) *Secuencial indexada:* este tipo de organización, que normalmente se emplea en discos magnéticos, se caracteriza por permitir dos tipos de acceso: el secuencial y el directo. Para ello se almacenan los registros junto con una "llave" o índice, que es el que permite recuperar los registros secuencialmente o ir directamente a ellos.

Con el objeto de acelerar los procesos que utiliza esta organización, se crean, al mismo tiempo que se almacenan los registros, tres tablas: índice máximo en cada pista (track index), índice máximo en el cilindro de información (cylinder index) y el índice máximo en un grupo de cilindros (master index). En esta forma, cuando se efectúa, por ejemplo, el proceso de búsqueda de un registro, se empieza en el master index y en él se determina en qué parte del cylinder index se debe continuar. En el cylinder index, a su vez se indica el punto donde debe seguir la búsqueda en el track index y en éste, la pista donde está almacenado el registro buscado y en la pista, finalmente, la búsqueda es secuencial.

En el momento de crear el archivo se exige que los registros entren ordenados secuencialmente, lo que determina que el área destinada para los registros se vaya llenando sin dejar huecos o espacios libres. Sin embargo, con el transcurso del tiempo, lo normal es que haya necesidad de insertar nuevos registros, lo que produce "desbordes" en las pistas donde ellos se agregan. Con el objeto de mantener la organización, se crean áreas de desborde (overflow) en los cuales "caen" los registros desplazados por la inserción. A pesar de esto, es posible efectuar procesamientos secuenciales, lo cual se logra mediante indicaciones de encadenamiento que permiten situar los registros almacenados en las áreas de desborde.

iii) *Random:* este tipo de organización se logra mediante una relación existente entre la identificación del registro y su ubicación física. Hay dos métodos para obtener esta relación: algoritmo de almacenamiento y tabla de índices de registros.

Con el primer método se presentan dos tipos de problemas: uno es la generación de sinónimos, esto es, la aplicación del algoritmo da como resultado la misma dirección para distintos registros; y el otro es que quedan áreas sin usar, pues el algoritmo no las asigna a ningún registro. El primer problema se soluciona mediante un sistema de encadenamiento similar al utilizado con las áreas de desborde en la organización del punto anterior, así cada registro indicará la ubicación del sinónimo siguiente. Con el segundo método es necesario tener una tabla en la que aparecen las identificaciones de todos los registros, en forma ascendente, con sus respectivas ubicaciones.

iv) *Particionada:* este tipo de organización se refiere a archivos que se han subdividido en miembros que tienen, cada uno, una organización secuencial.

Para lograr la recuperación de los registros, se tiene una tabla en que se relaciona el nombre o identificación de cada miembro con la ubicación del primer registro que contiene.

v) *Relativa:* esta organización se aplica solamente a archivos de longitud fija que están almacenados en DASD. Los registros en este caso

se sitúan respetando dos normas básicas: que la distribución sea uniforme y que sea hecha a partir de una posición bien determinada. Se puede observar que no es necesario que exista una relación lógica entre los registros, dado que la distribución uniforme constituye una constante que multiplicada por el número de orden del registro buscado y sumado el resultado a la dirección de partida, da la ubicación de dicho registro.

c) *Procesos de computación de servicio corriente*

Un proceso de computación es una elaboración de datos haciendo uso de un computador para obtener resultados específicos.

Existen algunos procesos que se realizan en la mayoría de los sistemas de información y son los siguientes:

i) *Clasificación (Sort):* se refiere a aquellos procesos en que se ordenan los registros de un archivo de acuerdo con el contenido de determinados campos. Como ejemplo se pueden citar:

Ordenamiento de un conjunto de personas de acuerdo con la renta bruta percibida.

Ordenamiento de un conjunto de personas en forma ascendente, de acuerdo con su primer apellido.

Mortalidad por grupo de edades y sexos, dentro de un período de años.

ii) *Intercalación (Merge):* es aquél en que se obtiene un archivo en una secuencia determinada a partir de dos o más archivos que deben estar en la misma secuencia. Se hace notar que los archivos deben haber sido sometidos previamente al proceso de clasificación. De ahí que existan programas utilitarios que cumplen ambas funciones (Sort/Merge).

iii) *Pareamiento:* son aquellos procesos en que se trata de ubicar *pares* de registros, cada uno perteneciente a un archivo distinto, que cumplen una relación de correspondencia. Por ejemplo, la misma identificación.

En general, el proceso de pareamiento consiste en ubicar dos registros que tengan una misma clave o argumento de búsqueda y, en forma optativa, determinar si coinciden o si existe relación entre otros argumentos de ambos registros. Las coincidencias o diferencias que se encuentren permitirán decidir si los registros son o no correspondientes.

El problema que se presenta a menudo es que las claves o argumentos tienen errores, lo que trae como consecuencia un porcentaje alto de registros que no tienen el par correspondiente en el otro archivo. Incluso entre los registros pareados habrá también una cantidad en que los demás argumentos tengan errores. Las técnicas utilizadas

para darle solución al problema se analizan en el punto d) Explotación de archivos secuenciales.

iv) *Concatenación:* es aquél en que el objetivo es unir en uno solo, dos o más archivos, de tal manera que queden ubicados uno a continuación del otro sin que se produzca mezcla de registros.

v) *Validación:* éste es uno de los procesos de mayor importancia en el procesamiento electrónico de datos, pues con él se trata de asegurar al máximo la veracidad y corrección de un determinado conjunto de datos. Al mismo tiempo permite aplicar métodos de corrección de los errores detectados.

Los procesos de validación entregan normalmente un archivo con los registros que no contienen error en los campos validados y un listado de registros en los que se ha detectado error o anomalía.

Una vez que se ha obtenido el listado, debe procederse a un análisis de los errores para efectuar la corrección y enseguida la preparación de ésta. Esto significa, en primer lugar, que el sistema debe estar diseñado de tal manera que permita las correcciones en forma eficiente, y en segundo lugar, que el proceso de validación en sí es un proceso repetitivo, esto es, debe efectuarse tantas veces como sea necesario para dejar el archivo totalmente depurado de errores.

De acuerdo con lo anterior, se puede decir que la validación consiste en realizar diversas verificaciones o chequeos de la información, que pueden clasificarse como sigue:

Chequeo orientado a un campo
- chequeo de caracteres (alfabético, numérico, etc.)
- chequeo de contenido (cadena, rango, etc.)

Chequeo orientado a un registro
- chequeo de consistencia de campos
- chequeo de dígito verificador

Chequeo orientado a un conjunto de registros
- chequeo de consistencia de registros
- chequeo de corte de control

El "corte de control" es la acumulación de los valores de ciertos campos numéricos hasta detectar que se cumple una condición de control prefijado, momento en el que se entregan los totales acumulados.

Después que se ha detectado el error y por consiguiente su ubicación, en algunos casos se pueden aplicar métodos de corrección automática. Estos métodos son:

Método de paquetes fríos: se hace la corrección en función de otras variables para asignar el código representativo más adecuado.

Método de los paquetes calientes: se hace la corrección en función de otras variables para asignar el código representativo más

adecuado, pero este código es renovado de acuerdo con la información que va apareciendo a través del proceso.

Método deductivo: consiste en determinar, mediante razonamiento, cuáles son los valores más probables a asignar.

Método aleatorio: consiste en crear el código que se va a asignar, de acuerdo con una generación de un número al azar y una distribución acumulativa porcentual de los códigos posibles.

Método mixto: combinación de algunos de los métodos anteriores.

vi) *Edición:* corresponde fundamentalmente a la preparación de la información, de manera que ésta pueda ser transferida directamente desde algún medio de almacenamiento o después de alguna pequeña elaboración a formularios continuos.

La preparación puede consistir en formateo de registros, inserción de espacios en blancos, colocación de comas o punto decimal, etc.

Debido a las características de un sistema EPD, es corriente que la información, una vez que se ha preparado a "imagen de impresora", sea grabada en cinta o disco magnético y de ahí sea transferida al formulario continuo.

vii) *Actualización:* se refiere a la puesta al día de un archivo maestro, de tal modo que incluya todos los movimientos que se han producido desde la fecha de su creación o de la última actualización. Las operaciones que implica la actualización, en parte o en su totalidad, son:

Modificación de campos
Inserción de nuevos registros
Eliminación de registros

viii) *Cálculo:* A este tipo de procesos corresponden las operaciones aritméticas que se realizan con los datos. Varía la complejidad de los problemas que se resuelven, desde simples combinaciones de operaciones hasta complicadas secuencias de ellas. Entre los primeros están los sistemas de información administrativa en que se tiene un gran volumen de datos y con ellos se efectúan operaciones muy simples. En cambio, en los problemas de tipo científico, la situación es inversa, es decir, muy pocos datos y con ellos operaciones aritméticas que corresponden a modelos matemáticos complejos.

Es importante hacer notar, sin embargo, que los procesos de cálculo siempre van acompañados de uno o varios de los tipos de procesos vistos anteriormente.

ix) *Copia:* se refiere a la función de transferir un archivo, sin hacerle modificaciones, desde un medio de almacenamiento a otro similar o distinto. Este tipo de proceso, por su simplicidad y uso

común, se encuentra programado y forma parte del paquete de programas utilitarios. Se pueden citar por ejemplo los siguientes:

Tarjeta a tarjeta
Tarjeta a cinta magnética
Tarjeta a disco magnético
Tarjeta a impresora
Cinta magnética a tarjeta
Cinta magnética a disco
Cinta magnética a impresora, etc.

d) *Explotación de archivos secuenciales*

El uso de archivos secuenciales es inherente, en una u otra forma, a la mayoría de los procesos de computación, de ahí que se estime conveniente tratar en forma exclusiva la explotación o, lo que es lo mismo, el trato más eficiente de ellos.

i) *Técnicas de pareamiento:* se vio en el proceso de pareamiento que el problema que se presenta a menudo es que las claves o argumentos tienen errores, lo que significa no encontrar el registro que forma el par con el que se está analizando. Algunos métodos de solución son:

Archivo de referencias cruzadas. Este es un archivo similar al archivo básico, pero clasificado de acuerdo con otro argumento. En algunos casos, contiene la misma información que el archivo básico y en otros se prefiere tener registros reducidos que incluyen el campo de clasificación y la clave para volver al archivo básico. Por ejemplo, un archivo de personal clasificado de acuerdo con el número de identificación del empleado, y otro de referencias cruzadas, clasificado según el nombre. Si se trata de parear un registro cuyo número de identificación sea erróneo, no se encontrará el registro correspondiente en el archivo básico. Se busca entonces pareando por nombre en el archivo de referencias cruzadas. Si en éste se tiene la información completa, el pareo se ha logrado inmediatamente; en caso contrario, con el número de identificación correcto se buscará el resto de la información en el archivo básico.

Código Soundex. Tiene por objeto tratar de obviar los errores cometidos en la escritura de nombres, para lo cual se asigna un código al nombre y a los apellidos de acuerdo con las siguientes reglas:

El código consta de un carácter alfabético (la letra inicial del nombre o apellido) seguido de tres dígitos que dependen de las consonantes del nombre o apellido, según la tabla que sigue:

DIGITO	LETRAS
1	B,F,P,V
2	C,G,J,K,Q,S,X,Z

DIGITO	LETRAS
3	D,T
4	L
5	M,N
6	R
7	Y (seguida por vocal)
0	para rellenar, si es necesario

Las vocales y las letras h,w,e y (no seguida por vocal) no se consideran.

Las letras dobles se consideran como una sola letra.

Dos letras contiguas del mismo grupo se consideran como una sola letra.

Si la segunda letra del nombre es del mismo grupo que la inicial, no se considera.

Dos letras del mismo grupo, separadas por vocal o por y, se consideran por separado.

La h y la w no son separadores.

La s, la k y la z finales no se consideran.

Las partículas DEL, DE, LA, etc. no se consideran.

Ejemplo:

CAMPUSANO	C512
CAMPUZANO	C512
CAMPOSANO	C512
CANPUSANO	C512
CANPUZANO	C512

El problema que se puede presentar es que nombres distintos tengan códigos iguales.

Ejemplo:

PEREZ	P620
PORRAS	P620

Después que se han consignado las diferencias o igualdades, es necesario decidir si hay pareo o no. Para esto se pueden utilizar tres sistemas:

Sistema de aciertos. Consiste en contar el número de argumentos que coinciden con los del registro correspondiente y si se cumple al menos un número prefijado de argumentos, se dan por pareados los registros.

Sistema de ponderadores lógicos. Este sistema se basa en un algoritmo que representa la lógica del razonamiento del usuario y que toma en cuenta la importancia particular de las coincidencias y discrepancias de los distintos argumentos.

Sistema de ponderaciones logarítmicas. Se basa en la teoría de la

información y consiste en asignar ponderaciones a las coincidencias y discrepancias de los argumentos.

Para las coincidencias, se asigna como ponderación el logaritmo negativo de la frecuencia del valor del argumento. En el caso de las discrepancias, se asigna como ponderación el logaritmo positivo de la tasa de errores en el argumento.

Los registros serán pareados si la ponderación total obtenida es mayor o igual a un número mínimo establecido previamente sobre la base del comportamiento real del sistema en operación.

ii) *Técnicas de búsqueda.*

Búsqueda serial: Se efectúa examinando serialmente cada registro del archivo hasta encontrar el deseado. Si cada registro del archivo tiene la misma probabilidad de ser usado, el número de comparaciones con los registros almacenados en un archivo de N registros se da por la fórmula:

$$C = \frac{N + 1}{2}$$

Si las claves de búsqueda se clasifican y procesan por lotes, la búsqueda serial se hace tanto más eficiente cuanto mayor sea el número de registros requeridos. Así, si se requieren M registros, el número de comparaciones es:

$$C = \frac{N}{M}$$

Si la probabilidad de ser ocupado cambia de un registro a otro, se puede ordenar el archivo en orden decreciente de probabilidad. De esta forma se tiene:

$$C = \sum_{r=1}^{r=N} r * P_r.$$

donde:

P_r = Probabilidad de requerir el registro r-ésimo

Búsqueda binaria: Considerando que el archivo está ordenado secuencialmente, se subdivide en tres conjuntos: uno que tiene un solo elemento y éste es el que está ubicado físicamente en el centro del registro, otro que tiene todos los registros con clases menores a la del registro del centro y el tercero, que tiene todos los registros con claves mayores.

El proceso se inicia al comparar la clave de búsqueda con la del registro central; si son iguales, la búsqueda termina; en caso contrario, queda definido el conjunto en el cual se continúa. De aquí en adelante

el proceso se repite en la misma forma hasta encontrar el registro buscado.

El número medio de comparaciones se da por:

$$C = \frac{N+1}{N} \log_2 (N+1) - 1$$

y el número máximo por:

$$Cmax = \log_2 (N+1)$$

Búsqueda por bloques: En esta búsqueda, primero se subdivide el archivo en bloques de largo *s*. A continuación se hace una comparación con el último registro del primer bloque; si la clave es mayor, se sigue con el último registro del segundo bloque y así sucesivamente hasta que la clave sea menor que la del registro. Se determina así el bloque que contiene el registro buscando, el cual se detecta mediante una búsqueda secuencial dentro del bloque.

Si el archivo tiene N registros que se agrupan en bloques de largo *s*, se tendrá un total de N/*s* bloques. Las comparaciones necesarias para ubicar estos bloques variará de 1 a N/*s* y dentro de cada bloque se requieren de 1 a (*s*-1) comparaciones, luego el número medio de comparaciones será:

$$C = \frac{N+s^2}{2s}$$

y el número máximo

$$Cmax = \frac{N}{s} + s - 1$$

Se desprende de lo anterior que es necesario optimizar el largo de los bloques para obtener un C menor. Este largo es:

$$s\,opt = \sqrt{N}$$

con lo cual

$$C = \sqrt{N}$$

BIBLIOGRAFIA

1. Arnold, Robert R., Hill, Harold C. y Nichols, Ayhmer V., *Sistema Moderno de Procesamiento de Datos*, México, Limusa/Wiley, 1971, 387 p.
2. Abrams, Peter and Corvine, Walter *Basic data processing*, 2ª edición, San Francisco, Rinehart Press, 1971, 487 p.
3. CELADE, *Curso de introducción al procesamiento electrónico de datos* (PED) *para cientistas sociales*, Santiago, Chile, 1974, 117 p.
4. Davis, Gordon B., *Introducción a los computadores electrónicos, México, Continental, 1969, 583 p.*
5. Du Roscoat, J., *Conceptos de la programación de los ordenadores*, Barcelona, Toray/Mason, 1971, 364 p.
6. ECOM, *Curso básico de análisis de sistemas*, Santiago, Chile, 1972.
7. Ibáñez B., Pablo y Castro S., Eduardo, *Algoritmos de corrección automática*, Tesis, Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, 1974.
8. IBM System products division, IBM *system/370; system summary, Nueva York, 1973.*
9. IBM System products division, *A guide to de* IBM *system/370 model 135*, Nueva York, 1974.
10. Katzan, Harry Jr., *Computer organization and the system/370*, Nueva York, Van Nostrand-Reinhold, 1971, 308 p.
11. Sánchez C., Víctor y Gutiérrez G., Jorge, *Introducción a la computación*, Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, 1972, 109 p.
12. Schmidt, Richard W. y Meyers, William E., *Introducción a los ordenadores y al proceso de datos*, Madrid, Raraninfo, 1971, 452 p.

LENGUAJES Y PROGRAMACION

I. INTRODUCCION

Uno de los principales problemas con que se encuentra el investigador, el programador o el usuario de computadores en general es la representación gráfica de los procesos que tiene que resolver.

Lo normal es que trate de solucionar las dificultades a medida que se van presentando, pero, desgraciadamente, lo hace en la etapa de escritura del problema en un lenguaje que entienda el computador, saltándose así toda la etapa de clarificación y de formalización del método de solución del mismo. Es como si iniciara un viaje conociendo la meta, pero totalmente ignorante de la ruta que le conducirá a ella.

Seguramente, esto se debe a la falta de cursos que enseñen a los estudiantes a pensar en forma lógica, de manera ordenada, con método y disciplina, independientemente de las carreras que sigan o vayan a seguir. Tal vez se debe también al hecho de que los cursos de lenguajes de computador se hacen a presión en las universidades, tratando de que el alumno los aplique en la solución de problemas de otras asignaturas lo antes posible, sin que haya una coordinación previa con ellas y, lo que es más grave, sin que se emplee el tiempo necesario para que el alumno se dedique exclusivamente a construir algoritmos de solución de problemas y a plantearlos en forma de diagramas de flujo.

Otra consecuencia lamentable es la falta de documentación de los programas y sistemas. Los puntos que siguen tienen como principal objeto corregir las deficiencias ya anotadas o al menos servir de ayuda a quien desee eliminarlas.

II. ALGORITMOS Y DIAGRAMAS DE FLUJO

1. *Definición de algoritmos*

Se define el algoritmo como un conjunto finito de pasos que permiten obtener la solución de un problema. Existen algoritmos numéricos y no numéricos. Ejemplos del primer tipo son: los de las operaciones aritméticas, el algoritmo para obtener la raíz cuadrada de un número, el algoritmo para resolver un sistema de ecuaciones, etc. Ejemplos del segundo tipo son: las recetas de cocina, las instrucciones necesarias para cambiar un neumático a un auto, el algoritmo que permite colorear un mapa con sólo cuatro colores, etc.

Para aclarar mejor el concepto de algoritmo, se tienen los dos ejemplos siguientes:

Ejemplo 1. Se tienen escritos en una hoja, cuatro números distintos, enteros, desordenados. Determinar el mayor de ellos.

Si se observan los números escritos, se puede señalar, casi de inmediato, el número mayor. Pero este resultado se ha obtenido al realizar un proceso mental, del cual algunos pasos se efectúan casi inconscientemente. Al analizar detenidamente el proceso efectuado y al escribir los pasos dados, se obtiene el siguiente algoritmo:

i) Se compara el primer número con el segundo
ii) El que resulte mayor se compara con el tercero
iii) El que resulte mayor de esta segunda comparación se compara con el cuarto
iv) El que resulte mayor de esta tercera comparación es el número pedido.

Ejemplo 2. Dar las instrucciones a una persona para que sirva una taza de café con leche. Dicha persona no sabe hacerlo, pero cuenta con todos los elementos necesarios para ello, esto es, loza y servicio limpios, café y leche calientes, azúcar.

El algoritmo respectivo será:

i) echar azúcar en la taza
ii) echar café a continuación
iii) echar leche.

A pesar de que este algoritmo permite preparar una taza de café con leche, con él se pueden obtener múltiples resultados que dependerán del criterio y gusto de la persona que sirva. Luego, si se quiere una solución determinada, se debe especificar con más detalle los pasos del algoritmo. Podría ser por ejemplo:

i) echar tres terrones de azúcar en la taza
ii) echar café hasta un cuarto de taza
iii) echar leche hasta llenar la taza.

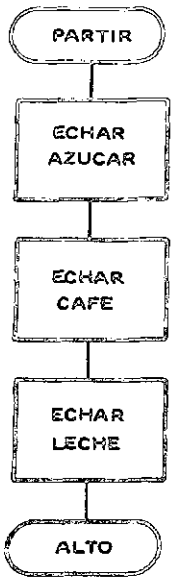
2. *Diagramas de flujo y aplicaciones*

El diagrama de flujo es la representación gráfica del algoritmo de solución de un problema. Para construirlo se cuenta con determinadas figuras geométricas o "símbolos". Así, por ejemplo, la figura siguiente:

que permite iniciar o terminar un proceso, según la leyenda que se coloque en su interior.

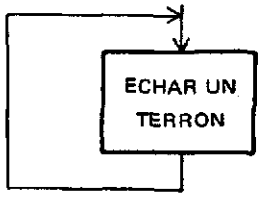
El rectángulo que permite indicar la operación que se va a realizar.

Con estos dos símbolos se puede hacer el diagrama de flujo correspondiente al algoritmo del ejemplo 2. Al considerar el algoritmo más general se tiene:

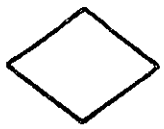


La operación  puede ser reemplazada por:

Pero aún podría ser representada mediante la introducción de un "ciclo".

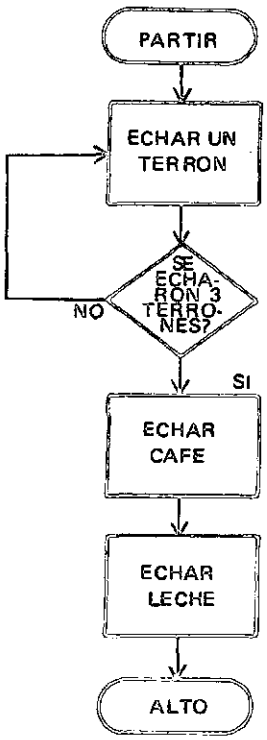


Al salir del bloque, la dirección de la flecha indica un retorno incondicional a la misma operación. Sin embargo, esto se ha transformado en un ciclo indefinido, sin término, pues no existe un elemento que permita detener la operación. Se debe establecer un límite y ese es la cantidad de azúcar que se debe echar. En otras palabras debe haber un control, en forma de pregunta, que indague si se han echado tres terrones. Si la respuesta es NO, el ciclo debe continuar; pero si es afirmativa, debe terminar, o lo que es lo mismo, salir de él para realizar la operación que sigue en secuencia. Esto se interpreta también como una "decisión" tomada a partir de las preguntas y sus posibles respuestas y en el diagrama de flujo se representa con un rombo:



que permite encerrar en él una pregunta

Utilizando este nuevo símbolo, se obtiene el diagrama de flujo que se indica a continuación:

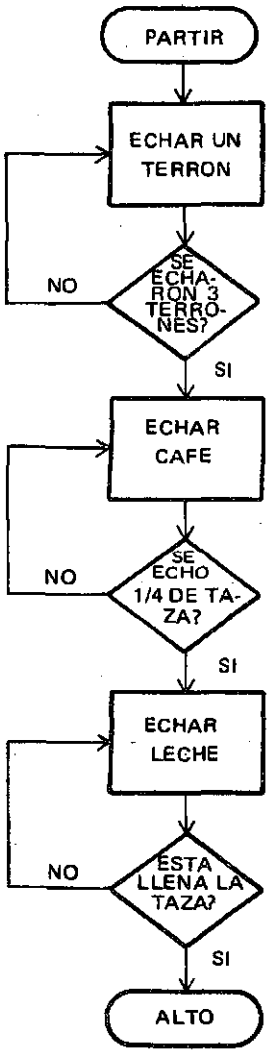


Observando el ciclo, se verifica que se ejecuta mientras no se hayan echado 3 terrones; una vez que esto ocurre, se sale de él para pasar a la operación ECHAR CAFE.

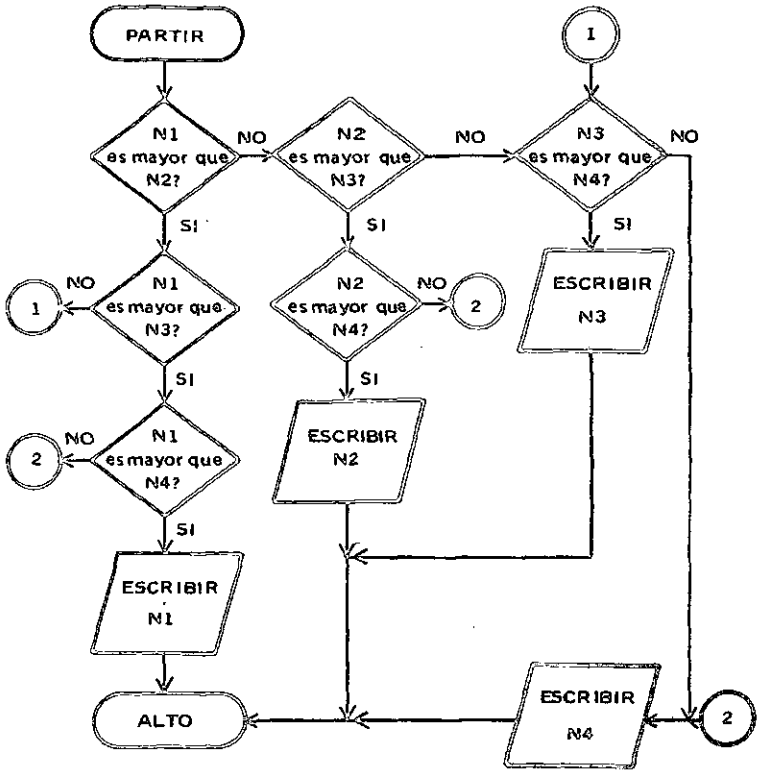
De estos diagramas se pueden deducir algunas reglas sobre las cuales se insistirá para evitar ambigüedades en la construcción de otros. Debe advertirse que de una figura de iniciación (PARTIR) sólo puede salir una flecha; a una figura de término (ALTO) pueden llegar múltiples flechas y, desde luego, no puede salir ninguna. A las figuras de operación pueden llegar una o más flechas, pero puede salir sólo una.

Finalmente, a la figura de decisión pueden llegar una o más flechas y salen tantas como sea el número de respuestas posibles. Evidentemente, este número debe ser superior a uno para que exista decisión.

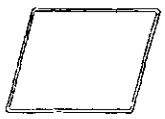
Volviendo al problema, el diagrama de flujo completo será:



Y el diagrama de flujo correspondiente al ejemplo 1 será el siguiente:



Se han introducido dos nuevas figuras en el diagrama de flujo:



El romboide, que se utiliza para indicar lectura o escritura de información, o lo que es lo mismo, ENTRADA o SALIDA de información en el proceso, y

El círculo, que permite enlazar dos o más partes del diagrama, dentro de una misma página. En el interior del conector se especifica cualquier símbolo, el que debe repetirse en el o los conectores de enlace.

Existe también el pentágono como conector, que permite enlazar dos o más partes del diagrama, que están en distintas páginas. En este caso, dentro del conector se especifican dos símbolos separados entre sí por: — ó / ó, ó; ó cualquier otro carácter especial. El primer símbolo identifica la página y el segundo al conector dentro de la página.

Se puede observar en el diagrama que las flechas no necesariamente tienen que "entrar" en una figura, pueden empalmar con otra flecha. Lo importante es que el flujo quede claro.

Una de las unidades que componen un computador es la MEMORIA, formada por "celdas" donde se guarda información. Esas celdas están numeradas en forma correlativa y el número de orden constituye la "dirección" de la celda. Se considerará el problema siguiente: Guardar en la celda 10 el resultado de la suma del contenido de la celda 15 más el contenido de la celda 20. Si se utiliza la notación ( ) para expresar "contenido de", se puede escribir:

(celda 10) = (celda 15) + (celda 20),

que se puede leer: el contenido de la celda 10 es igual al contenido de la celda 15 más el contenido de la celda 20. Pero la interpretación del signo = es un tanto ambigua, puesto que se podría interpretar como: lo que "había" en la celda 10 es igual a lo que se puede obtener si se suman los contenidos de las celdas 15 y 20.

Ese problema queda solucionado si se intepreta el signo = como "está definido por", con lo cual se lee: el contenido de la celda 10 está definido por el resultado de la suma de los contenidos de las celdas 15 y 20. También se puede decir: el resultado de la suma de los contenidos de las celdas 15 y 20 "se asigna a" la celda 10, o "se guarda en" la celda 10.

Otros símbolos que se utilizan corrientemente para definir o asignar son: la flecha en sentido de derecha a izquierda (←——) y dos puntos seguidos del signo igual (:=). En ese caso se anotará:

(Celda 10) ←—— (celda 15) + (celda 20) o
(Celda 10) := (celda 15) + (celda 20)

El símbolo de definición permite también escribir expresiones como la siguiente:

(celda 10) = (celda 10) — (celda 15)

que significa: el contenido de la celda 10 queda definido por el resultado obtenido al restar del contenido de la celda 10, el contenido de la celda 15. Si dichos contenidos fueran 1345 y 826, el nuevo valor guardado en la celda 10 será 519.

Si se generaliza y se identifican con los nombres A, B y C las celdas 10, 15 y 20 respectivamente, se podrá escribir:

$$A = B + C$$

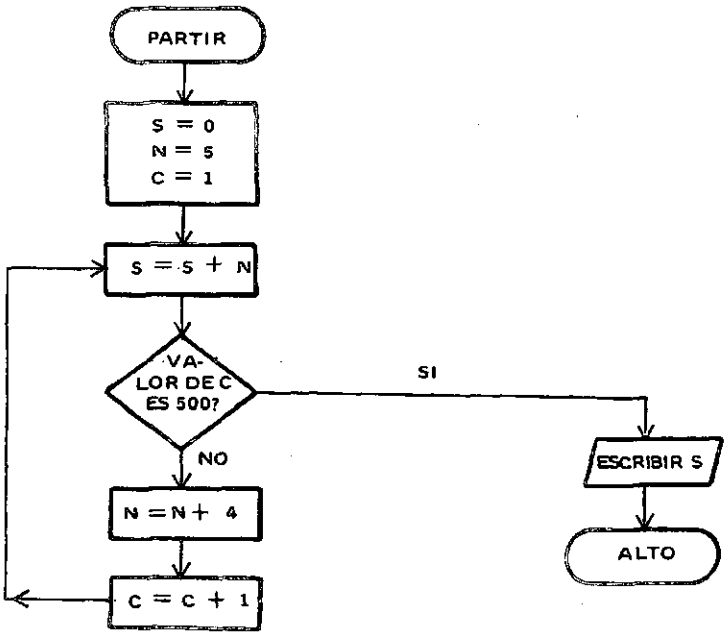
que se puede interpretar como: la *variable* A queda definida con el resultado obtenido al sumar el valor de la *variable* B al valor de la *variable* C.

Se llama la atención al hecho de haber asignado las variables A, B y C a tres celdas de memoria.

El siguiente problema se resolverá con estos nuevos conceptos:

Ejemplo 3. Hallar la suma de 500 términos de la progresión aritmética cuyo primer término es 5 con incremento 4, esto es 5, 9, 13, 17, . . . Se usarán las variables:

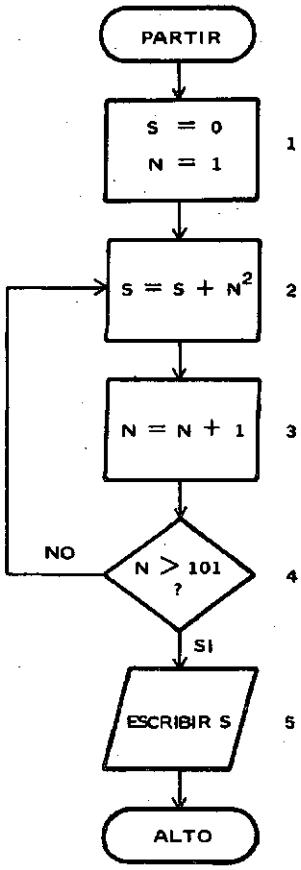
S para acumular la suma
N para formar cada término
C para controlar el número de veces que se realiza el ciclo de suma.



Ejemplo 4. Se desea tener un algoritmo que permita encontrar la suma de los cuadrados de los primeros 101 enteros positivos. En otras palabras, se quiere encontrar el valor de: $S = 1^2 + 2^2 + 3^2 + \ldots + 101^2$.

Con N se designará sucesivamente a los números 1, 2, 3, . . ., 101. Con cada valor de N se calculará su cuadrado y este valor incrementará la suma acumulativa S. Por supuesto, S debe tener un valor inicial 0.

En el ejemplo anterior se utilizó C para controlar la repetición del ciclo. Ahora se aprovechará la misma variable N y el valor que ella toma para efectuar el control.



Si se desea verificar si el algoritmo funciona, se numeran los bloques, se considera un número menor de términos, por ejemplo 5, y se construye la tabla siguiente:

Paso	Bloque Nº	Valor de las variables S	N	Control	Sí o No
Partir					
1	1	0	1		
2	2	1			
3	3		2		
4	4			2 > 5?	No
5	2	5			
6	3		3		
7	4			3 > 5?	No
8	2	14			
9	3		4		
10	4			4 > 5?	No
11	2	30			
12	3		5		
13	4			5 > 5?	No
14	2	55			
15	3		6		
16	4			6 > 5?	Sí
17	5	Escribir S			
Alto					

Nótese que estos diagramas de flujo no sugieren la idea de resolver el problema con un computador, a pesar de que se han utilizado las características de su "memoria" para avanzar en los conceptos que permiten mejorar la construcción de ellos. Esto significa que el problema puede resolverse perfectamente con cualquier sistema de procesamiento de datos, incluyendo lápiz y papel, como se acaba de realizar para controlar el funcionamiento del algoritmo de solución del problema propuesto.

Con los elementos hasta ahora estudiados se puede examinar un problema frecuente, cual es el de calcular medidas estadísticas de un juego de datos.

Ejemplo 5. Calcular la media aritmética.

El algoritmo de solución será:

i) Poner en cero los acumuladores para la suma y el contador de lecturas.

ii) Leer los datos.

iii) Contabilizar la lectura y sumar el dato al acumulador.

iv) Volver al paso ii).

Se ha llegado a un punto en que se entra a un ciclo que parece infinito, dado que siempre se retorna a leer datos. A pesar de que no se visualiza cómo "romper" esa secuencia obligada de operaciones, en la práctica, en algún momento, al tratar de realizar la lectura, se encon-

trará que los datos se han terminado y este hecho permitirá salir del ciclo.

Cualquiera que sea el medio en que esos datos estén registrados, siempre es posible hacer la pregunta ¿hay más datos?, obviamente, esta consulta debe ser hecha después de la lectura, para saber si al efectuar esta operación se encontraron datos o no.

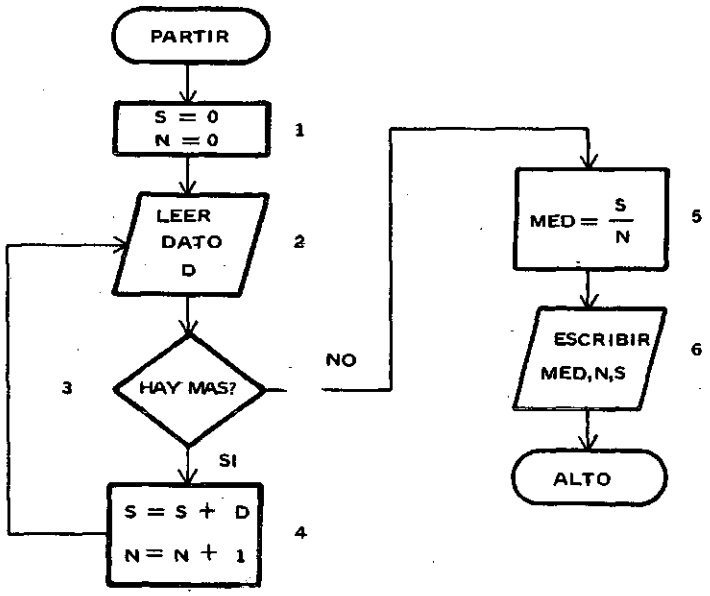
Es lo que hace el ser humano al leer datos en una hoja de papel. La vista recorre el conjunto transmitiendo la información al cerebro. Por cada dato leído, el cerebro consulta ¿hay más? y al recibir respuesta afirmativa ordena leer nuevamente. ¿Cuándo sabrá que no hay más? Cuando la vista se haya dirigido al espacio a continuación del último y haya transmitido la información encontrada. Esa información, que puede ser: espacio en blanco, un paréntesis, un punto, etc., le indicará al cerebro que no hay más datos.

En el problema planteado los pasos que están a continuación del ciclo son:

v) Calcular la media aritmética

vi) Escribir la media, el número de datos y la suma acumulada.

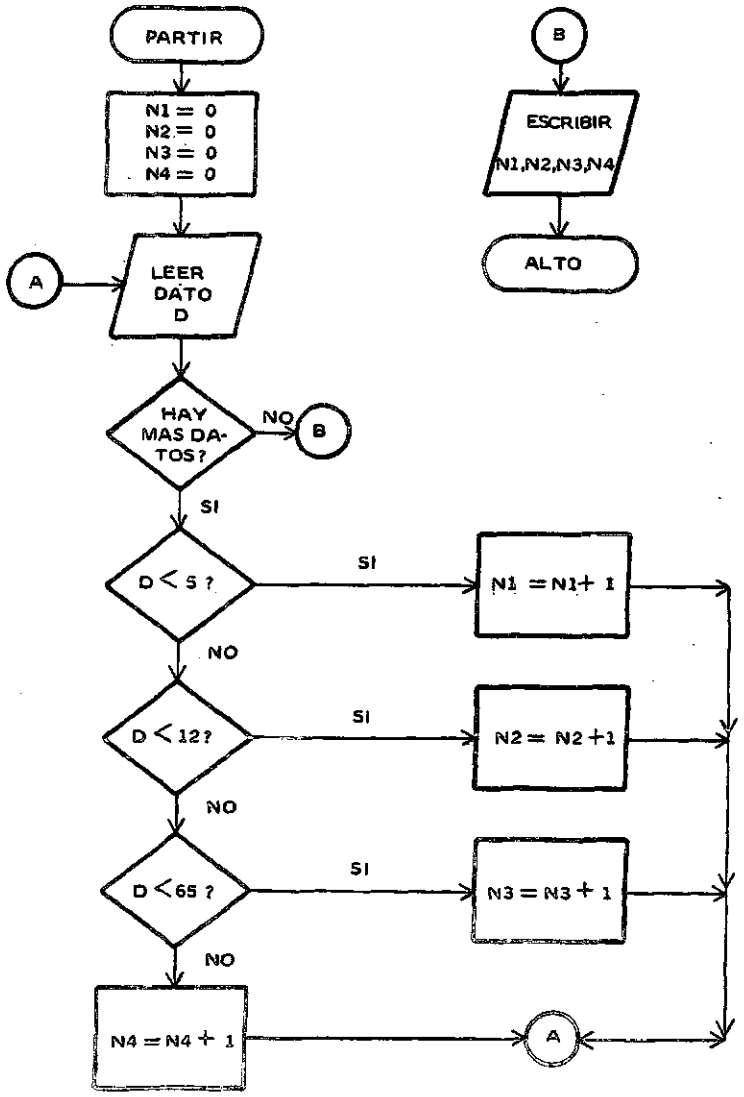
El diagrama de flujo correspondiente a este algoritmo es:



Es conveniente hacer notar que, cuando se especifican dos o más operaciones dentro de un bloque, como es el caso de los bloques 1 y 4 del diagrama anterior, la secuencia de ejecución de ellas se efectúa en el mismo orden en que están escritas. Esta secuencia, que siempre se mantiene, es independiente del punto de entrada de las flechas del bloque.

Ejemplo 6. Considerando el mismo juego de datos del ejemplo anterior, calcular la frecuencia de ellos en cada una de las 4 siguientes categorías, según su valor numérico:

- menores que 5
- de 5 a 11
- de 12 a 64
- mayores de 64



Para la solución, se han empleado los símbolos N1, N2, N3 y N4 para calcular las frecuencias de la primera, segunda, tercera y cuarta estratificación, respectivamente.

En aquellos problemas en que se trabaja con conjuntos de datos, se les asignan nombres a dichos conjuntos con el objeto de identificarlos. El nombre asignado puede ser totalmente arbitrario o puede indicar o dar idea referente al tipo de datos que contiene el conjunto. Por ejemplo, si se tienen las estaturas de niños de 7 años, se las podrá identificar con: X, ESTAT, ESTATURA7, etc. Todos ellos son nombres posibles, sin embargo, ESTAT es más claro que X, y ESTATURA7 es mucho más que los anteriores, en relación con el tipo de información contenida en el conjunto.

Si se desea identificar cada dato, será necesario indicar primero el nombre del conjunto y en seguida la posición o número de orden del dato dentro de él.

Por ejemplo, sea el conjunto de temperaturas máximas que hubo en una ciudad en un lapso de 5 días:

34,3	32,1	29,7	31,5	32,4

Si se identifica al grupo con el nombre genérico T, cada temperatura tendrá el nombre particular T_i en que i varía de 1 a 5. Así se obtiene:

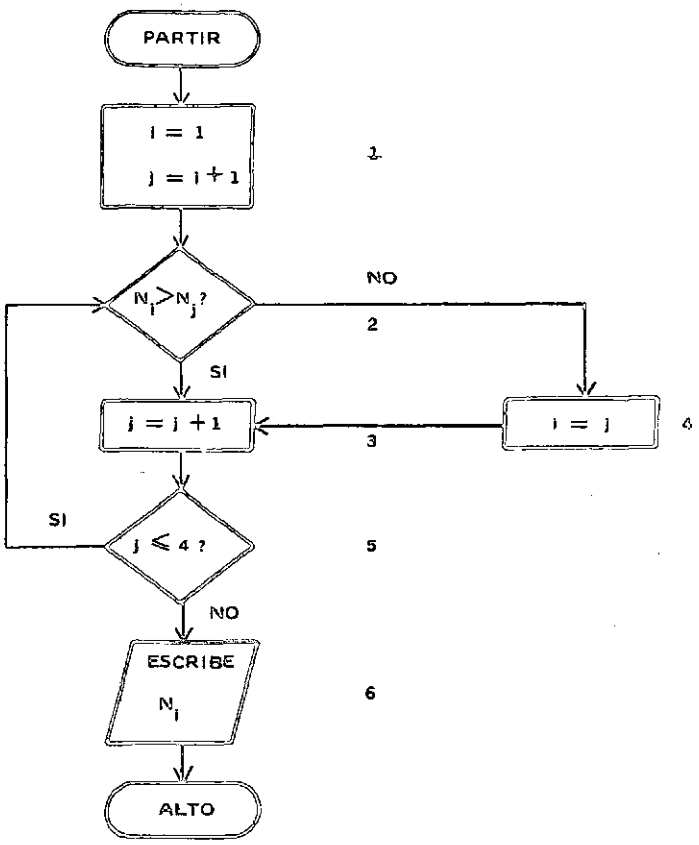
$$\begin{aligned} T_1 &= 34,3 \\ T_2 &= 32,1 \\ T_3 &= 29,7 \\ T_4 &= 31,5 \\ T_5 &= 32,4 \end{aligned}$$

El número de orden del dato se llama índice, los nombres particulares *variables con índice* o *variables indexadas* y el nombre genérico, *nombre de arreglo* siendo *arreglo* el conjunto de datos.

El índice se escribe más bajo que el nombre genérico, de ahí que corrientemente se le denomina sub-índice. Cuando se tiene que escribir el índice a la misma altura del nombre genérico, es conveniente diferenciarlo en alguna forma de él. Puede ser escribiéndolo con minúscula, o de un tamaño menor o separándolo con algún signo especial.

La razón de esta medida queda clara al considerar el elemento general de un conjunto o, como se dijo antes, un nombre particular tal como T_i. Si el índice se escribe con mayúscula a la misma altura de T, queda TI, que se confunde con el nombre de variable TI que puede haber sido utilizado, o lo será más adelante, en otra parte del diagrama, como nombre simple de variable.

Al resolver el ejemplo 1 haciendo uso de índices, se tendrá:



Para controlar el funcionamiento del algoritmo se asignan valores arbitrarios a las variables N_i. Sean ellos 5, 1, 9 y 13 para definir N_1, N_2, N_3 y N_4 respectivamente.

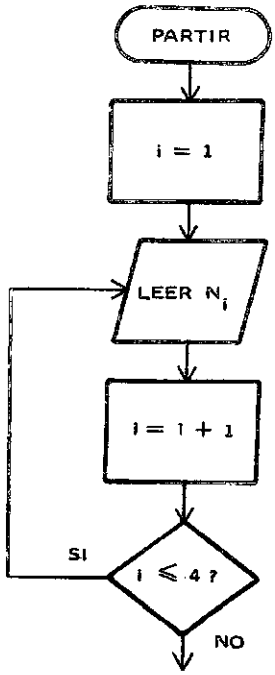
Paso	Bloque Nº	Valor de las variables				Control	Sí o No
		i	j	N_i	N_j		
Partir							
1	1	1	2	5	1		
2	2					$5 > 1$	Sí
3	3		3		9		
4	5					$3 \leqslant 4$	Sí
5	2					$5 > 9$	No
6	4	3		9			
7	3		4		13		
8	5					$4 \leqslant 4$	Sí
9	2					$9 > 13$	No
10	4	4		13			
11	3		5		?		
12	5					$5 \leqslant 4$	No
13	6	Escribe N_4					
Alto							

En el paso 11, aun cuando el índice *j* está "apuntando" a N_5, que no existe, no tiene importancia porque dicha variable no se utiliza.

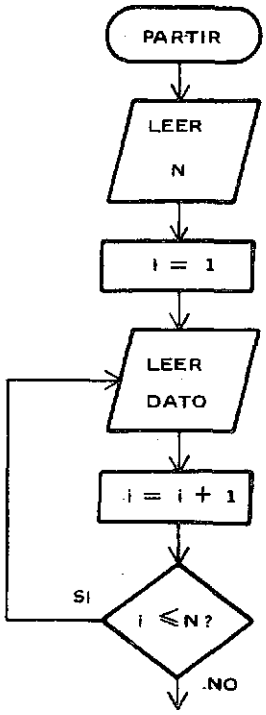
Se han seguido construyendo los algoritmos independientes de la idea de resolver los problemas mediante computador. En lo sucesivo las soluciones se orientarán por ese camino. A pesar de ello, se podrá notar que el problema podrá ser siempre resuelto con otro sistema de procesamiento de datos utilizando el mismo algoritmo.

De acuerdo con lo anterior, se presentan a continuación algunas variantes del diagrama visto para el ejemplo 1. Estas variantes suponen almacenamiento de datos en la memoria del computador.

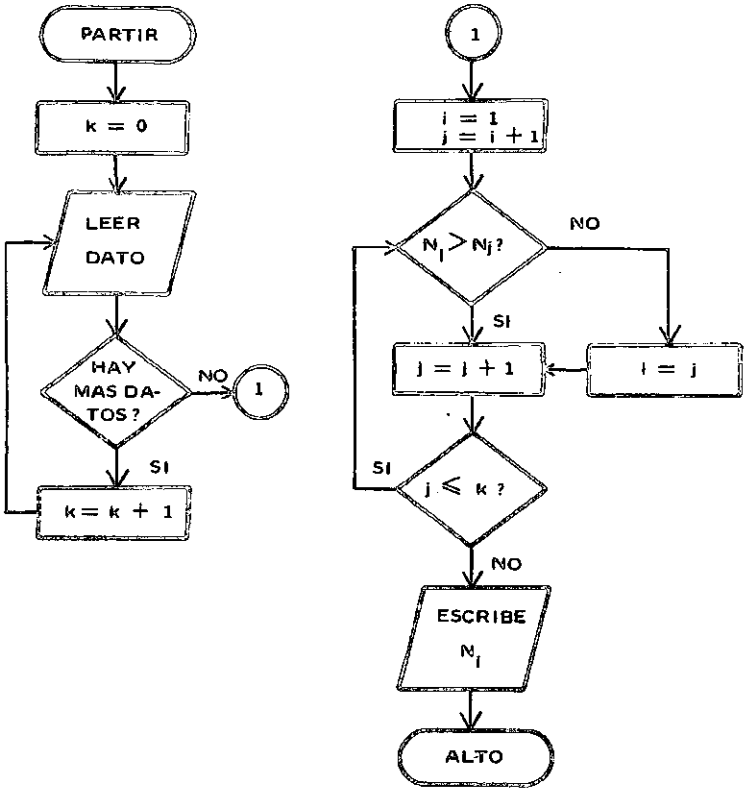
a) El número de datos es conocido (en el problema es 4).



b) El número de datos no es conocido, pero se sabe que encabeza el conjunto de datos (N será el número de datos).

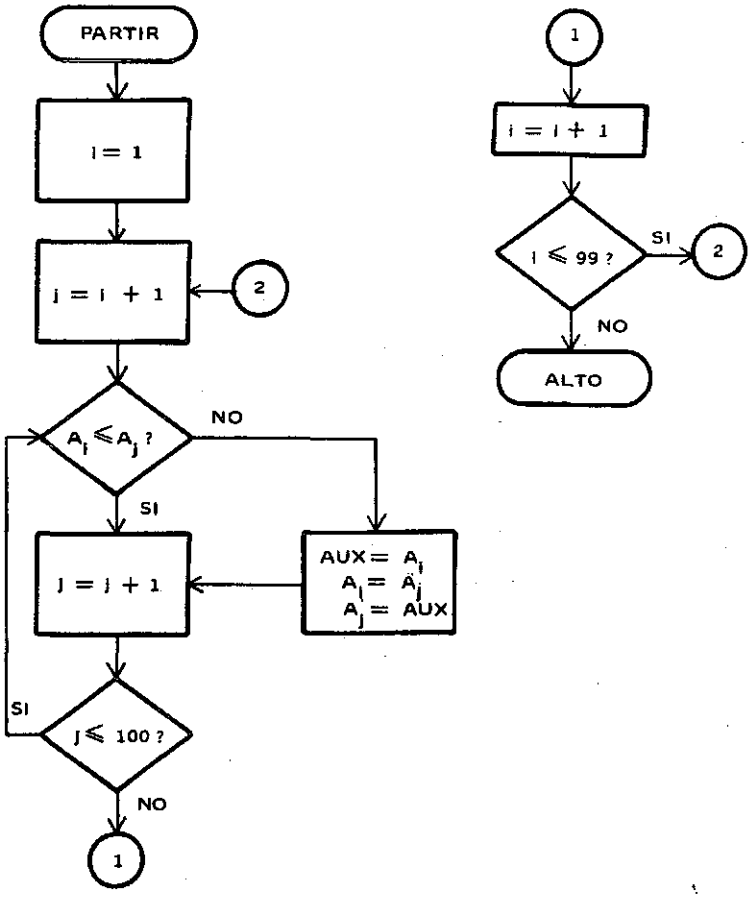


c) El número de datos no es conocido.



La solución b) es más general que la a), y la c) más que la b). Esto no significa que sea la única ni tampoco que es la mejor posible. De idéntica manera a la usada para almacenar información en memoria, se pueden sacar datos de ella.

Ejemplo 7. Se tiene un arreglo de 100 elementos, que se desean ordenar de menor a mayor. Con el objeto de simplificar los algoritmos, se omitirán los ciclos de lectura e impresión.



Se han resuelto dos problemas mediante el uso de variables con un índice. Sin embargo, este tipo de variables es ineficiente para resolver el problema del ejemplo siguiente:

Ejemplo 8. Se tienen registrados los datos de una población y entre ellos figura la edad y el estado civil de cada habitante. Se pide obtener la siguiente tabla:

Edad	Población por estado civil según edades simples						
	Estado Civil						
	Soltero	Casado	Conviviente	Separado	Divorciado	Viudo	Ignorado
1 año y menos							
.....							
37 años							
98 años							
99 años y más							
Ignorado							

Para resolver el problema debe considerarse que se ha hecho la siguiente codificación:

Edad ignorada	100
Estado civil	
Soltero	1
Casado	2
Conviviente	3
Separado	4
Divorciado	5
Viudo	6
Ignorado	7

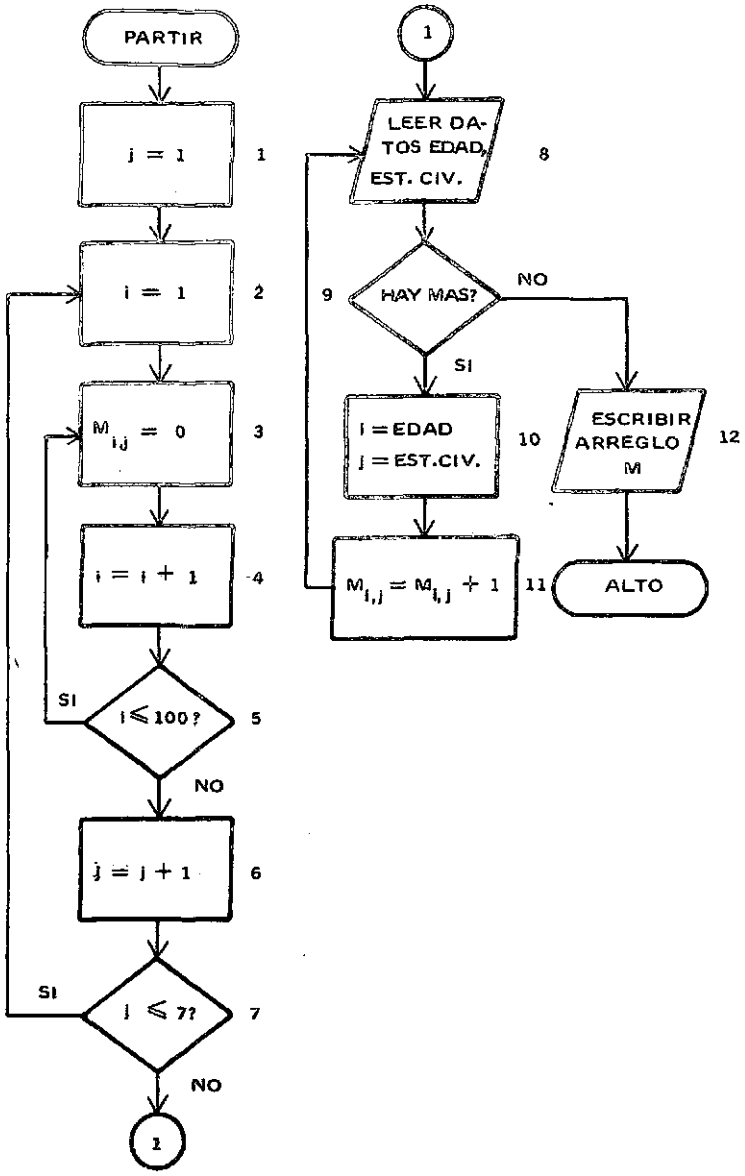
Se puede observar que la tabla contendrá 700 resultados correspondientes a 100 edades por 7 estados civiles, por lo tanto será necesario utilizar variables con índice para acumular cada uno de ellos. En caso contrario, se tendrían que crear 700 variables y darle un nombre distinto a cada una, lo que no permitiría un algoritmo eficiente, dado que sería necesario construir un ciclo de acumulación por cada variable.

Es posible resolver el problema haciendo uso del tipo de variable "indexada" que se ha visto, o sea variable con un índice. Pero la variable con un índice implica un arreglo lineal, sea renglón o columna, y la tabla pedida corresponde a un arreglo bilineal, es decir, de dos dimensiones. Se tendrá que trabajar entonces con 7 arreglos lineales, uno por cada tipo de estado civil. Sin embargo, la construcción del algoritmo presenta aún dificultades que es posible eliminar si el problema se resuelve con variables con dos índices, uno por cada dimensión del arreglo.

La idea es la misma que se vio para variables con un índice. Habrá un nombre genérico que identifica al arreglo y nombres particulares que identifican al dato. El nombre particular está formado por el nombre genérico seguido de la ubicación del dato dentro del arreglo. Cuando se trata de un arreglo de dos dimensiones, la ubicación del dato corresponde al cruce del renglón y de la columna que lo contienen. Luego, si con el primer índice se identifica el renglón y con el segundo la columna, se tendrá la posición del dato.

Si se denomina con M al arreglo del problema, el elemento general, o lo que es lo mismo, un nombre particular, será M_{ij}, i variará de 1 a 100 y j de 1 a 7. Cuando se trate de identificar a un determinado elemento del arreglo, i y j tendrán un valor numérico, será necesario entonces separarlos por coma para evitar confusión en su lectura. Ejemplo: $M_{5,4}$ es el nombre del dato que está en el cruce del renglón 5 con la columna 4, diferente de M_{54} que es el elemento 54 del arreglo lineal M.

Solución del problema:



Observaciones:

a) Si se hubiera colocado el ciclo que se repite más veces encerrando al ciclo que se repite menos, el número de bloques que se obtiene será el mismo. Sin embargo, la cantidad de veces que se ejecuten las operaciones indicadas en los bloques 2,6 y 7 será bastante mayor.

b) Los bloques 10 y 11 pueden ser reemplazados por uno solo que contenga $M_{EDAD,\ ESTCIV} = M_{EDAD,\ ESTCIV} + 1$

c) En esta solución no se ha almacenado el conjunto de datos en memoria, de tal manera que en el bloque 8 no ha sido necesario utilizar variables con índice. En todo caso, a diferencia de los problemas anteriores, ahora se han almacenado dos datos con cada orden de lectura, edad y estado civil.

d) La forma en que debe ordenarse la información y cómo deben darse las instrucciones de escritura para obtener la tabla tal como ha sido solicitada, esto es, con títulos, encabezamientos por columna, etc., es materia de capítulos posteriores, de ahí que en el bloque 12, solamente se haya especificado la orden ESCRIBIR ARREGLO M.

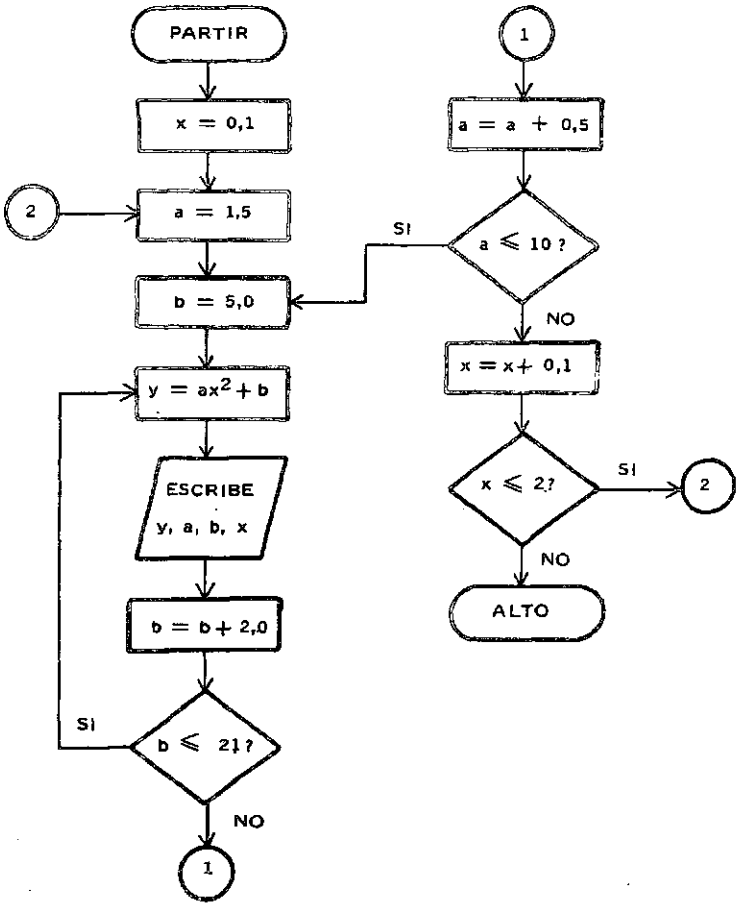
Ejemplo 9. Se desea calcular el valor de la función:

$$y = ax^2 + b \qquad \text{para } a = 1{,}5\ (10{,}0)\ 0{,}5$$
$$b = 0{,}1\ (\ 2{,}0)\ 0{,}1$$
$$x = 5{,}0\ (21{,}0)\ 2{,}0$$

la notación a = 1,5 (10) 0,5 se lee como sigue: *a* varía desde 1,5 hasta 10,0 con incrementos de 0,5. En igual forma se interpretan *b* y *x* utilizando los valores que les corresponden.

Lo anterior significa que deben efectuarse todas las combinaciones posibles entre *a*, *b* y *x*. Cada una de esas combinaciones determinará un valor para *y*.

a) Se calculará el valor de y para una combinación a, b, x, y el resultado se escribirá inmediatamente.



b) Los valores de y, calculados, se guardarán en memoria. Una forma cómoda de resolver el problema es utilizando un arreglo de tres dimensiones. Si se llama Y a dicho arreglo, el elemento general será $Y_{i,j,k}$ en que i controlará la variación de a, j la de b y k la de x.

El límite superior de cada índice se obtiene a base de: valor final, valor inicial e incremento de la variable que controla. Ellos se reemplazan en la fórmula siguiente:

$$\text{lim. sup} = \text{parte entera de } \frac{(\text{valor final} - \text{valor inicial})}{\text{incremento}} + 1$$

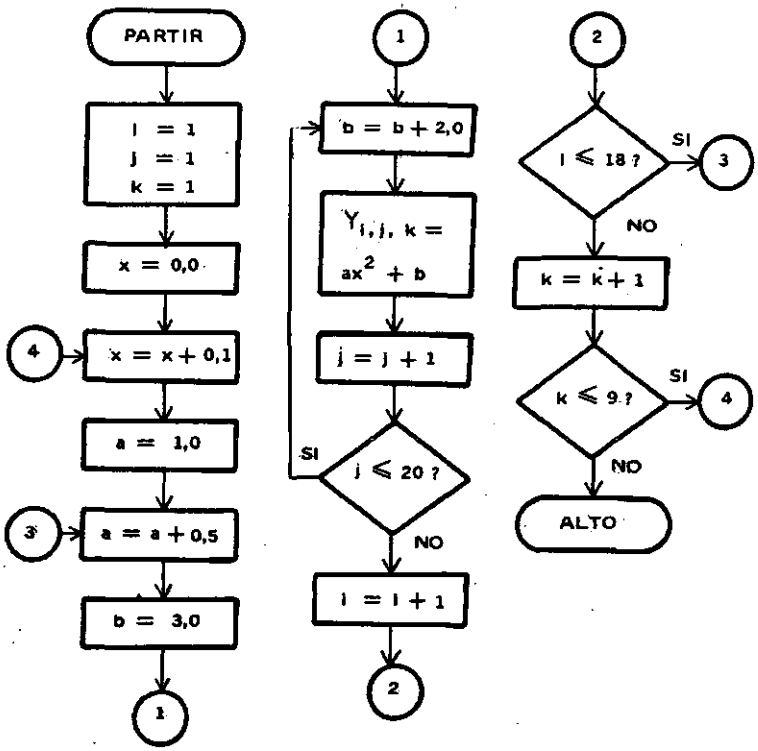
por ejemplo:

$$\text{límite superior de } i = \text{parte entera de } \frac{(10{,}0 - 1{,}5)}{0{,}5} + 1$$

límite superior de $i = 18$

lo mismo respecto a los índices j y k

Solución del problema:



Los algoritmos se pueden hacer a dos niveles:

a) a nivel de operaciones, como se ha venido realizando hasta ahora, y

b) a nivel de conjuntos de operaciones.

En el primer caso, se explica en detalle la sucesión de pasos que es necesario dar para obtener el resultado. En el segundo se sintetizan grupos o "bloques" de operaciones en una sola expresión, de tal manera que se obtiene un algoritmo que indica, en forma general, lo que hay que hacer a través del proceso.

Ejemplo 10. Se tiene un conjunto X de datos. Al comienzo de ellos, se tienen además dos valores m y n, que se utilizarán en la siguiente forma:

$$\text{si } m = 1, \text{ calcular } Y_i = X_i^{\frac{1}{2}}$$

la raíz cuadrada de X_i se obtiene a base de la fórmula de aproximación de Newton

$$Y = \frac{1}{2}\left(Y_a + \frac{X}{Y_a}\right)$$

en que Y_a representa la penúltima raíz obtenida.

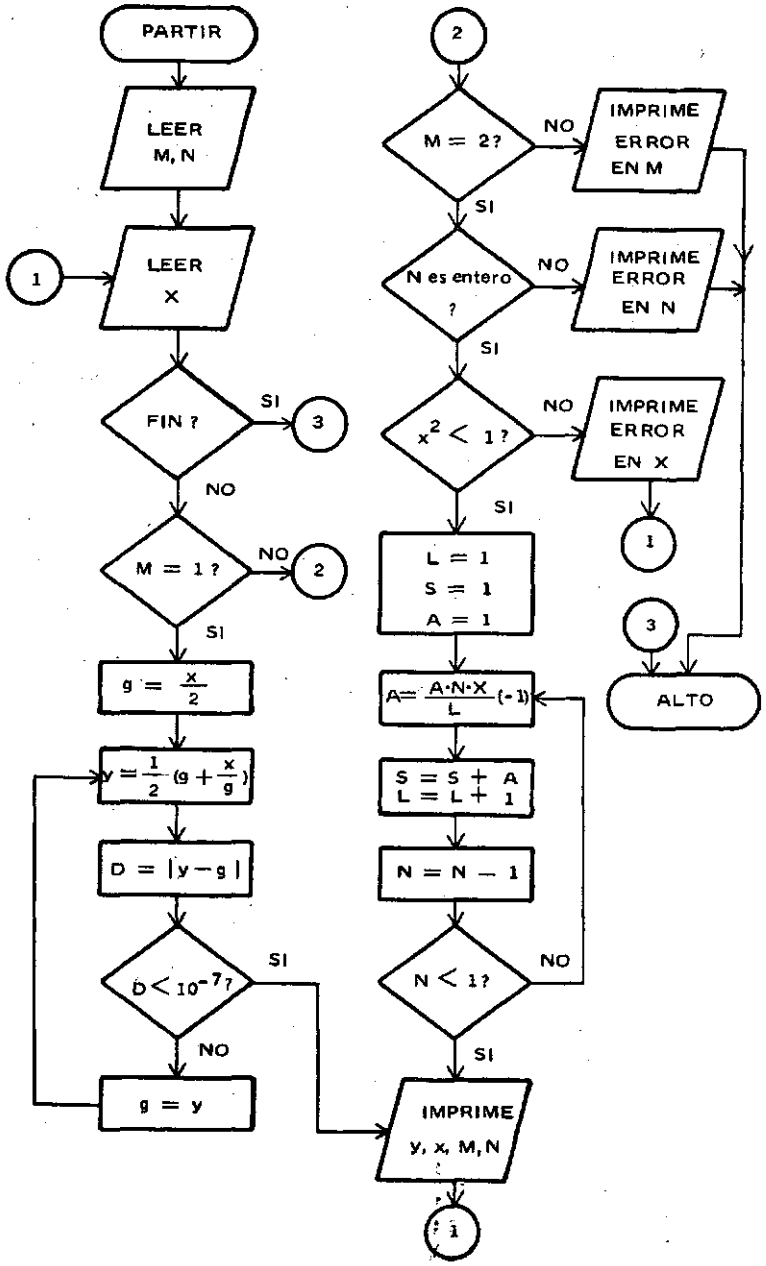
Detener el cálculo cuando la diferencia entre la penúltima y la última raíz obtenidas sea, en valor absoluto, menor que 10^{-7}

si $m = 2$, calcular

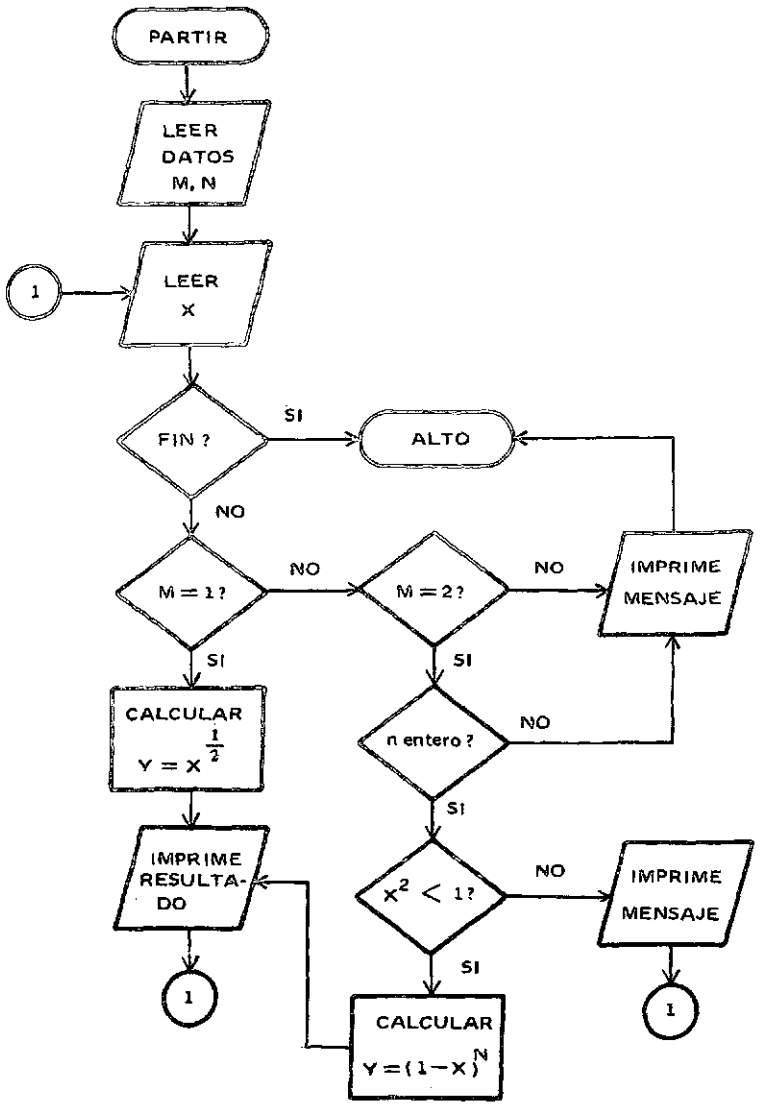
$$Y_i = (1-X_i)^n = 1 - nx_i + \frac{n(n-1)}{2!} X_i^2 - \frac{n(n-1)(n-2)}{3!} X_i^2 + \ldots$$

n debe ser entero mayor que 0 y X_i^2 debe ser menor que 1.

a) diagrama a nivel de operaciones.



b) diagrama de bloques.



En este diagrama, las expresiones $Y = X^{\frac{1}{2}}$ e $Y = (1-X)^N$ sintetizan bloques de operaciones, lo que permite obtener un diagrama más fácil de leer. Si bien es cierto que no se especifica cómo obtener los valores de Y en cada uno de los casos, se puede controlar mejor la lógica aplicada en el tratamiento de la información. Esto es de bastante utilidad cuando se tienen algoritmos previamente construidos, de manera de intercalarlos solamente en aquellos procesos que los usen; también cuando esas partes del algoritmo se deseen detallar en forma independiente.

Existe la siguiente figura para representar grupos de operaciones que no se detallarán en el diagrama:

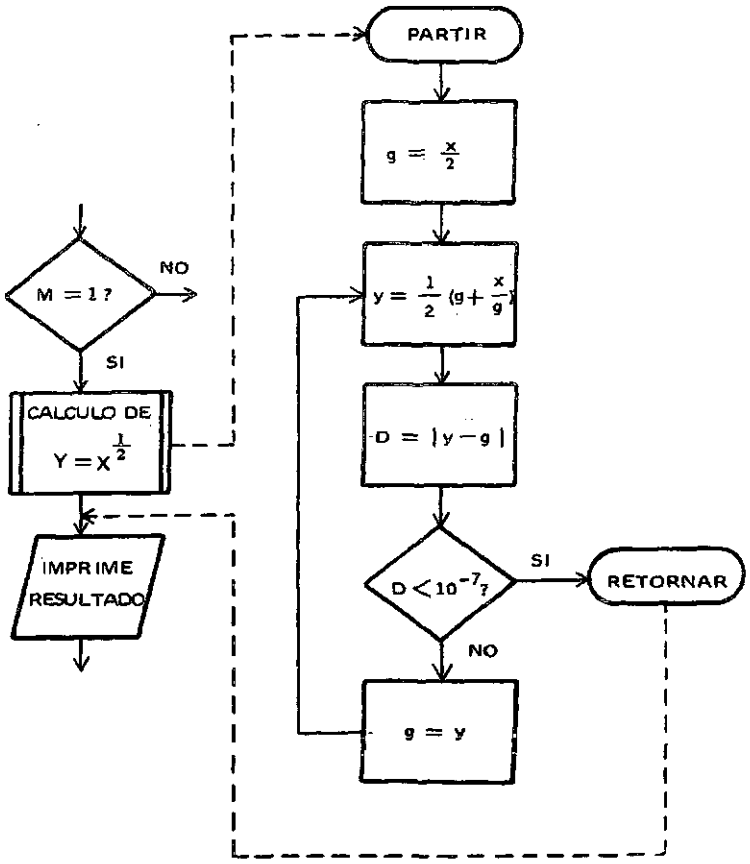
que permite indicar procesos predefinidos.

Si se considera en el diagrama anterior solamente la parte en que figura el cálculo de $Y = X^{\frac{1}{2}}$ se tendrá un ejemplo de uso de un proceso predefinido:

La parte de la derecha en el diagrama corresponde al algoritmo independiente o proceso predefinido que permite obtener $Y = X^{\frac{1}{2}}$. Este algoritmo es "llamado" por el algoritmo principal del cual recibe el dato X. Con X calcula el valor de Y, resultado que entrega al programa principal al retornar a él nuevamente.

La línea segmentada que en la práctica no se usa, se ha colocado con el objeto de hacer más claro el "salto" al proceso predefinido, como, asimismo, el retorno desde él.

c) Representación de un proceso predefinido.



Otro tipo de diagrama que interesa conocer es el *diagrama de flujo para sistemas.* Con este diagrama se describe el trayecto de la información a través de las distintas unidades que componen el sistema de procesamiento de datos.

Los problemas que figuran a continuación son ejemplos de uso del diagrama mencionado con sistemas de procesamiento de datos mecánico y electrónico.

Ejemplo 11. Se tiene un archivo de tarjetas maestras de productos, ordenadas por número de producto. En estas tarjetas está perforada, además de otros datos, la descripción y el precio unitario.

De bodega llegan formularios que tienen como información el número del producto y la cantidad que hay en bodega. Se desea sacar un informe (listado) con: número del producto, descripción, precio unitario, cantidad en bodega y valor de la existencia.

Los pasos que será necesario dar para obtener el listado pedido, si se piensa en un sistema mecánico de procesamiento de datos, son los siguientes:

a) La información contenida en el documento que llega de bodega se perfora en tarjetas y luego se verifica que haya sido perforada correctamente.

b) Las tarjetas obtenidas (tarjetas de detalle) se clasifican de acuerdo con el número del producto.

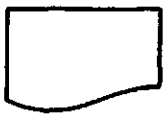
c) Las tarjetas clasificadas se intercalan con las del archivo maestro, seleccionando las tarjetas maestras sin tarjetas de detalle y las de detalle sin maestras. En este último caso se trata de un error que es necesario localizar y corregir.

Se obtiene un archivo ordenado por número de producto, en el que cada tarjeta maestra tiene a continuación su respectiva tarjeta de detalle.

d) Se reproduce (gang-punch) la información, descripción y precio unitario de las tarjetas maestras en las de detalle.

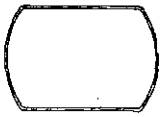
e) Se seleccionan (separan) las tarjetas maestras de las de detalle.

Para representar mediante un diagrama de flujo el proceso descrito, será necesario considerar las figuras que se indican a continuación:

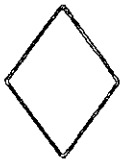


para representar documentos e informes

para tarjetas perforadas



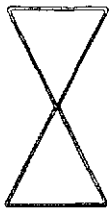
para operaciones efectuadas en máquinas con teclado



para representar arreglo en secuencia de un conjunto de ítem (SORT)

para representar la combinación de dos o más conjuntos de ítem en uno solo (MERGE)

para extraer de un conjunto uno o más conjuntos específicos de ítem.

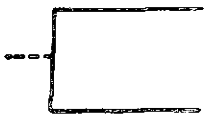


para representar la combinación con extracción; se forman dos o más conjuntos de ítem a partir de otros dos o más conjuntos (COLLATE)

para archivos fuera de línea

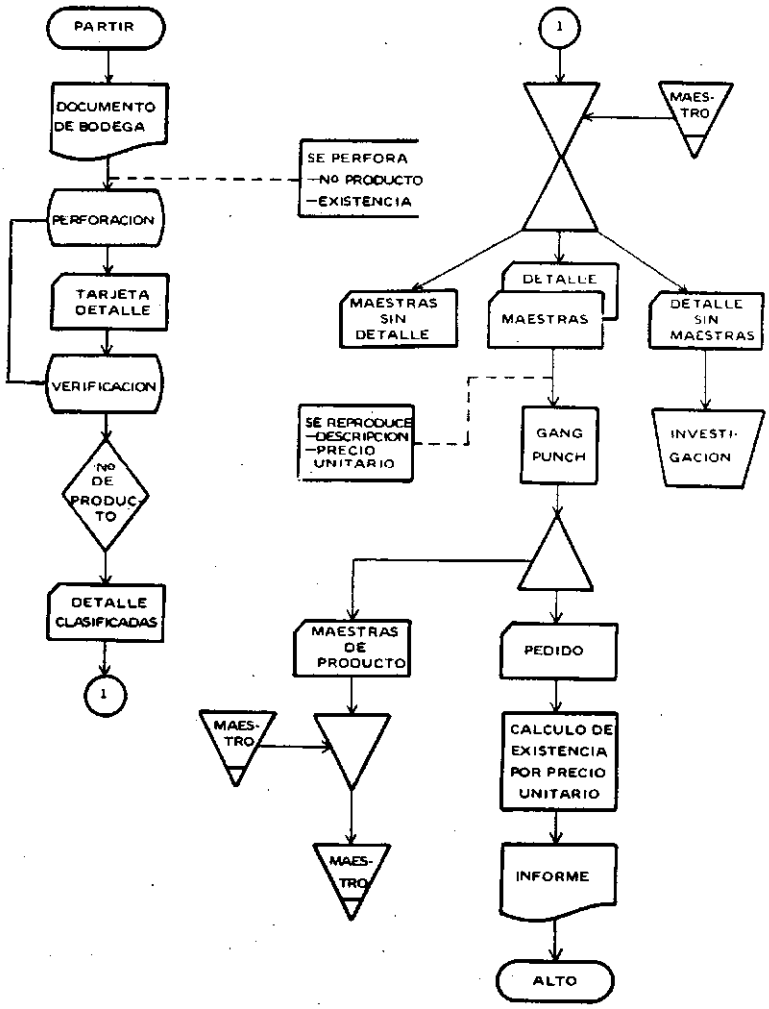
para operaciones manuales que no requieran equipo

para operaciones de máquina suplementarias al procesamiento principal



para agregar comentarios descriptivos o notas aclaratorias

Con la ayuda de estos símbolos se construye el diagrama que se indica.



Ejemplo 12. Resolver el problema anterior con un sistema de procesamiento electrónico de datos (PED). Para ello debe considerarse que:

a) La información registrada en el documento que llega de bodega ya ha sido perforada en tarjetas, pero no clasificada.

b) El archivo maestro de productos ordenados por número se tiene en cinta magnética.

c) La información de las tarjetas de detalle se clasifica y se graba en disco magnético.

Los pasos que será necesario dar para obtener el informe pedido se indican a continuación:

i) Se leen las tarjetas de detalle y la información se almacena en la memoria del computador en forma de imagen de tarjeta.

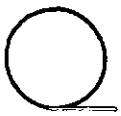
ii) La información leída se clasifica por número de producto y se graba en disco magnético.

iii) Se leen los registros del disco y de la cinta, se comparan y cuando corresponden al mismo producto, se efectúa el cálculo del valor de la existencia que hay en bodega.

iv) Terminado el cálculo, se imprimen los resultados para obtener el informe pedido.

v) Si hay registros en disco que no tienen su respectivo maestro en la cinta magnética, se imprimen para investigar el motivo.

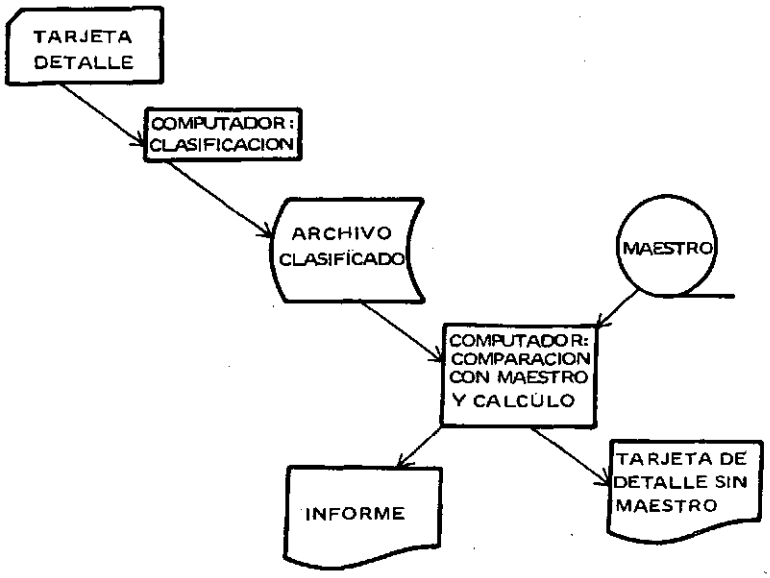
Se necesitan dos nuevos símbolos para poder construir el diagrama de flujo. Ellos son:



para representar cinta magnética en forma específica

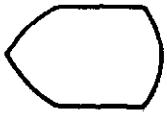
para representar disco magnético, tambor magnético, etc. Cualquier clase de almacenamiento en línea, utilizado para entrada/salida.

Se obtiene así el siguiente diagrama:

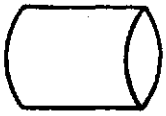


A continuación figuran otros símbolos que se utilizan:

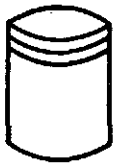
para representar entrada de información manual por medio de máquina de teclado en línea.



para el despliegue de información a través de indicadores en línea, dispositivo de video, plotters, etc.



para representar tambor magnético en forma específica



para representar disco magnético en forma específica

III. ¿QUE ES UN PROGRAMA?

En el punto anterior se vio que el diagrama de flujo se construye con el objeto de hacer el gráfico correspondiente al algoritmo de solución de un problema. Ambos, algoritmo y diagrama, constituyen un medio de comunicación entre la persona que escribe o dibuja cómo solucionar el problema y quién tiene que ejecutar cada uno de los pasos indicados para obtener dicha solución. En otras palabras, son instrucciones u órdenes que forman parte de un lenguaje y que deben ser obedecidas para poder obtener el o los resultados pedidos.

Pero cuando las instrucciones no van a ser entregadas a una persona para que las realice, sino a un computador, es necesario buscar otro medio de comunicación, o lo que es lo mismo, otro lenguaje que permita la comprensión de ellas por parte de la máquina.

En este caso, al conjunto de instrucciones escritas que componen la solución se le denomina PROGRAMA y de ahí derivan los nombres PROGRAMADOR, que es la persona que traduce el algoritmo a un lenguaje entendible por el computador y PROGRAMAR, que es el nombre que se le da a la labor de traducción. El lenguaje a su vez se llama lenguaje de PROGRAMACION.

Es conveniente señalar que lo usual es que el algoritmo sea dado al programador sin pensar en que la solución va a ser obtenida mediante la utilización de un computador. De ese modo, el programador debe tener en cuenta todas las características del lenguaje y de la máquina que va a usar, con el objeto de que ellas le permitan obtener la solución más eficiente. Su labor, en consecuencia, es en gran parte creativa.

Es distinto el caso cuando el algoritmo entregado está desarrollado en detalle a nivel de operaciones elementales. Cuando eso ocurre, que no es lo habitual, la labor de programar se transforma en un trabajo de CODIFICACION, que deja muy poco margen al programador para que aplique su capacidad de creación.

El programa, como asimismo la información con la cual trabaja, están registrados en memoria. El computador debe "saber" en qué parte de ella ha sido almacenada la primera instrucción para poder así iniciar la ejecución del programa. La forma en que se comunica esa dirección dependerá del tipo de computador que se esté utilizando. Una vez que esto se ha logrado, las instrucciones mismas informarán al computador dónde debe ubicar los datos.

IV. LENGUAJES

Para comunicarse con el computador existen dos niveles de lenguaje:

1. *Lenguaje de máquina*

El lenguaje de máquina corresponde al nivel inferior y está formado por un conjunto de instrucciones elementales o básicas, diferente, por supuesto, de un computador a otro en lo que se refiere fundamentalmente al formato de las instrucciones, cuyo efecto y estructura dependerán de las características tecnológicas que tenga el computador y cuya escritura se hace a base, exclusivamente, de dígitos y pueden representar: operaciones, "direcciones" en memoria o parámetros en general. Esto implica que la lectura o revisión de un programa presente dificultades de interpretación a quien las haga, más aún si la persona que cumple esa labor no está familiarizada con el computador utilizado o lo desconoce, le será muy difícil, si no imposible, conseguir su objetivo.

De acuerdo con las características del computador, la instrucción puede estar formada por:

- código de operación (OP)
- dirección del primer operando (D1)
- dirección del segundo operando (D2)
- dirección del resultado (D3)
- dirección de la próxima instrucción (D4)

lo que significa que podrá haber máquinas con instrucciones de una, dos, tres o cuatro direcciones. Además, en algunos computadores, no todas las instrucciones tienen igual longitud, de donde resultan máquinas con instrucciones de longitud fija y otras de longitud variable.

a) Se supondrá un computador ficticio cuya memoria está formada por 5000 celdas, cada una de ellas con capacidad para 18 dígitos. Se supone que el número 35 indica al computador que realice la operación SUMAR.

3500	35	1200	2400	3000	4000
	OP	*D1*	*D2*	*D3*	D4

En la celda 3500 se tiene una instrucción que indicará al computador lo siguiente: SUMAR al dato obtenido desde la celda 1200 el contenido de la celda 2400. Guardar el resultado en la celda 3000. La próxima instrucción se obtiene en la dirección 4000.

Todas las operaciones aritméticas se realizan en la unidad correspondiente, lo que permite hacer uso de los mismos datos todas las veces que se desee.

b) Se supondrá ahora que en el computador las operaciones son realizadas en un ACUMULADOR, siempre en la Unidad Aritmética, y que además, desde él, la transferencia de un resultado es automática.

Sin embargo, es necesario cargar el acumulador con el primer operando, para lo cual se utilizará el código de operación 32.

3500	32	1200	0000	4000
	OP	D1	D3	D4
4000	35	2400	3000	4500
	OP	D1	D3	D4

En la celda 3500 se tiene una instrucción que le indicará al computador: LLEVAR el dato contenido en la celda 1200 al ACUMULADOR. La próxima instrucción se obtiene en la dirección 4000, y dice: SUMAR el contenido de la celda 2400 al contenido del acumulador, llevar el RESULTADO a la dirección 3000. La próxima instrucción se encuentra en la dirección 4500.

En la primera instrucción se puede observar que el campo D3 no se utiliza. Si cada uno de los campos ocupara una celda, la primera instrucción ocuparía tres celdas y la segunda cuatro. Se tendrían así instrucciones de longitud variable.

c) Se considerará ahora que la operación de llevar el resultado desde el acumulador a una celda se obtendrá mediante una nueva instrucción, en que el código de operación es 31.

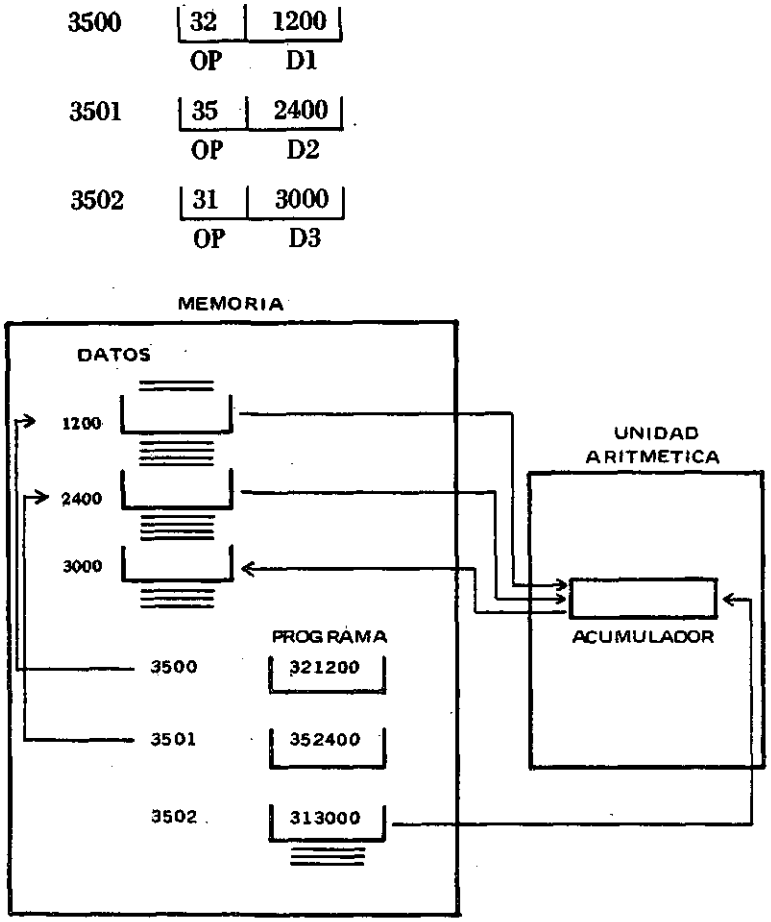
3500	32	1200	4000
	OP	D1	D4
4000	35	2400	4500
	OP	D2	D4
4500	31	3000	5000
	OP	D3	D4

La instrucción contenida en la celda 3500 indica que el dato guardado en 1200 se lleva al acumulador y la próxima instrucción está en la dirección 4000.

En la celda 4000 la instrucción dice que al dato contenido en el acumulador se le suma el valor guardado en la celda 2400 y la próxima instrucción está en la dirección 4500.

Finalmente, en la celda 4500 se tiene: guardar en la celda 3000 el resultado que está en el acumulador, y buscar la próxima instrucción en la celda 5000.

d) Si se considera que todas las instrucciones se encuentran una a continuación de la otra en el mismo orden en que van a ser ejecutadas, es posible eliminar el campo correspondiente a la dirección de la próxima instrucción. Se obtiene así la estructura siguiente:



La instrucción contenida en la celda 3500 hace que el computador lleve al acumulador el dato que está en la dirección 1200. La instrucción siguiente hace que se sume el contenido de la celda 2400 a lo que hay en el acumulador. Por último, la instrucción contenida en la celda 3502 hace que se descargue el acumulador en la celda 3000.

Con los ejemplos anteriores es posible ver con claridad las dificultades que encuentra el programador o quien tenga que leer o revisar un programa escrito en lenguaje de máquina. Es más difícil aún si se utilizan en la escritura otros sistemas numéricos.

Sólo con el ánimo de dar un punto de referencia más, se expone a continuación el mismo problema resuelto en los ejemplos, pero ahora utilizando el lenguaje de un computador real, un IBM/360. No interesa

el análisis de cada instrucción, sino el aspecto que presenta el conjunto de ellas.

```
58   50   20   0A
58   70   20   0E
1A   57
50   50   20   06
```

2. *Lenguaje simbólico*

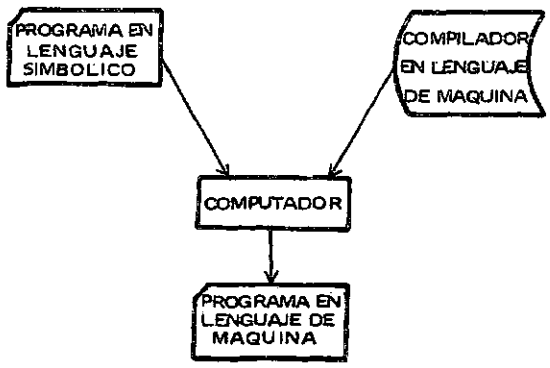
Debido a las dificultades mencionadas, fue necesario crear los lenguajes simbólicos que permiten al programador concentrarse más en la solución del problema y no tanto en la escritura de las instrucciones. Al mismo tiempo se disminuyen los errores de programación, de lectura, de perforación, etc., como asimismo el tiempo de búsqueda de ellos.

Es evidente que las instrucciones escritas en lenguaje simbólico no podrán ser "entendidas" por el computador, dado que éste conoce solamente el lenguaje de máquina. Es necesario, entonces, hacer una traducción de las instrucciones de un lenguaje a otro, para lo cual se hace uso de un programa traductor llamado COMPILADOR.

Las etapas que se deben efectuar para lograr la traducción son las indicadas a continuación:

- Se tiene el programa escrito en lenguaje simbólico
- Se almacena el programa en memoria
- Se almacena en memoria, el compilador, que está escrito en lenguaje de máquina
- Se realiza el proceso de traducción
- Como resultado, se obtiene el programa traducido, esto es, en lenguaje de máquina.

Para "graficar" este proceso se supondrá que el programa escrito en lenguaje simbólico se ha perforado en tarjetas y que el compilador está grabado en disco magnético.



Entre los lenguajes simbólicos es necesario distinguir tres categorías:

A. Los orientados a la máquina, en los que los elementos que conforman una instrucción de máquina han sido reemplazados en su totalidad, o en forma parcial, por símbolos. Dado que el conjunto de instrucciones de máquina y la estructura de éstas dependen de las características del computador, el lenguaje simbólico que considere directa o indirectamente esas características estará orientado a él. El programa que se vio escrito en lenguaje de máquina del IBM/360, al ser escrito en el lenguaje simbólico orientado a ese computador, quedará en la siguiente forma:

```
L     5,DATO1
L     7,DATO2
AR    5,7
ST    5,RESULT
```

Este lenguaje específico se llama ASSEMBLER y corresponde al tipo conocido como *ensamblador*, en que cada instrucción del lenguaje da origen a una sola instrucción de máquina. Esto significa que los algoritmos escritos en un lenguaje ensamblador resultan tan extensos como los escritos en lenguaje de máquina. Además, aun cuando la estructura de las instrucciones es más simple que en lenguaje de máquina y sus símbolos, fáciles de recordar y revisar, todavía no ofrecen mucha información a quien no esté interiorizado en las convenciones de su escritura.

B. Los *lenguajes generales o de alto nivel* permiten escribir los algoritmos en una forma fácil de comprender, pues se aproximan bastante al lenguaje que se habla a diario. Esto se debe a que las expresiones son similares a las que se escriben en matemática elemental, a que los nombres con que se designan las variables pueden ser tan descriptivos como se desee, y a que es posible colocar comentarios acerca de lo que realiza el programa, parte de programa o instrucción, en el idioma que la persona quiera, castellano, inglés u otro. Además, cada instrucción del lenguaje de alto nivel corresponde a varias instrucciones en lenguaje de máquina. Por ello es que las primeras se denominan sentencias, declaraciones o proposiciones.

Para lograr lo anterior todos los lenguajes disponen de constantes numéricas que se escriben con o sin signo, con o sin punto decimal (en reemplazo de la coma decimal). También, todos permiten usar variables y todos exigen que los nombres que identifican variables comiencen por carácter alfabético. Hay diferencia, sí, entre un lenguaje y otro en la cantidad de caracteres que componen el nombre. Por ejemplo, en BASIC puede tener hasta dos caracteres, en FORTRAN hasta seis; en cambio PL/I y COBOL aceptan hasta treinta caracteres.

En todos los lenguajes es posible "romper" la secuencia normal de

ejecución mediante sentencias de bifurcación o de "salto" de una sentencia a otra que puede estar antes o después en el programa. Estos saltos pueden ser de acuerdo con una condición o incondicionales y, en este caso, la sentencia es del tipo

GO TO X

en que X es un rótulo o etiqueta que identifica a otra sentencia que es la meta del salto. En BASIC todas las sentencias deben llevar rótulos numéricos; en FORTRAN los rótulos deben ser numéricos, pero no es obligatorio que todas las sentencias lo tengan; en PL/I y COBOL los rótulos son alfanuméricos y se utilizan igual que en FORTRAN.

Los saltos condicionales pueden estar complementados con sentencias IF que tienen una estructura distinta según sea el lenguaje al que pertenecen.

Dado que un computador sólo puede realizar las operaciones aritméticas, todos los lenguajes permiten formar expresiones aritméticas, cualquiera que sea su complejidad. Las variables que se utilicen pueden ser simples o con índices. Si se trata de estas últimas, la única exigencia de los lenguajes es que las dimensiones de los arreglos sean "declaradas" al comienzo del programa. PL/I difiere de los otros tres lenguajes en que permite usar índices cero o negativos si es necesario, no así los otros.

Ejemplo 13. Problema de la tabla estado civil-edad. En las páginas siguientes se verá, con los cuatro lenguajes mencionados, la solución del ejemplo 8, obtención de una tabla estado civil por edad. Las soluciones están escritas en Hojas de Codificación y se ha cuidado de diferenciar la letra O del número 0 escribiendo una barra diagonal sobre la letra. Cada línea de la hoja corresponde a una tarjeta.

1. *Solución en el lenguaje* FORTRAN

Esta solución está escrita en una variante de FORTRAN llamada WATFOR (un FORTRAN desarrollado en la Universidad de WATERLOO), utilizada con muy buenos resultados con propósitos educativos. Se diferencia del FORTRAN principalmente en las sentencias de entrada/salida de datos.

Las sentencias de comentario empiezan con la letra C en la primera columna. Los rótulos se codifican en las columnas 1 a 5 y el texto de la sentencia en las columnas 7 a 72.

La declaración de las dimensiones del arreglo M se hace por medio de la sentencia DIMENSION, ubicada al comienzo del programa.

Cuando se detecta *fin de datos* en la operación de lectura, se produce un salto automático a la parte del programa destinada a la impresión de los resultados. Esto se obtiene mediante la cláusula END=40 colocada en la sentencia READ.

SØLUCIØN FØRTRAN

```
C
C     SØLUCIØN FØRTRAN: TABULACIØN DE EDAD, ESTADØ CIVIL
C
            DIMENSIØN M(100,7)
C
C     1.-   INICIALIZAR M CØN CERØS:
C
            J=1
        1   I=1
        2   M(I,J)=0
            I=I+1
            IF(I.LE.100)GØ TØ 2
            J=J+1
            IF(J.LE.7) GØ TØ 1
C
C     2.-   LECTURA Y CØNTABILIZACIØN DEL CASØ:
C
        3   READ (END=40) EDAD,ECIV
            M(EDAD,ECIV)=M(EDAD,ECIV)+1
            GØ TØ 3
C
C     3.-   IMPRESIØN DE LA MATRIZ:
C
       40   I=1
        4   PRINT, (M(I,J),J=1,7)
            I=I+1
            IF(I.LE.100) GØ TØ 4
C
            STØP
C
C           LA SENTENCIA END INDICA AL CØMPILADØR QUE NØ HAY
C           MAS SENTENCIAS PØR TRADUCIR:
C
            END
```

2. *Solución en el lenguaje* PL/I

Está escrita usando algunas opciones básicas del lenguaje. En PL/I cada sentencia termina con punto y coma (;) y pueden escribirse varias sentencias en una tarjeta. Se puede codificar desde la columna 2 a la 72, en forma absolutamente libre. Los comentarios se escriben precedidos por los caracteres barra diagonal y asterisco (/*) y seguidos por asterisco, barra diagonal (*/):

```
/* ESTE ES UN CØMENTARIØ. */
```

Los rótulos se distinguen del texto de la sentencia porque se separan de ella con el signo dos puntos (:).

Las dimensiones de los arreglos se declaran mediante una sentencia DECLARE.

Para "inicializar" un arreglo con ceros, basta asignarle ceros al nombre del arreglo, sin necesidad de especificar índices.

La sentencia ØN ENDFILE (SYSIN) indica lo que debe hacerse cuando se terminan los datos.

La sentencia END indica al compilador que no hay más sentencias por traducir y además indica término del proceso.

SOLUCIØN PL/I

```
/o
  SØLUCIØN PL/I: TABULACIØN DE EDAD, ESTADØ CIVIL
                                                        o/
TABULA: PRØCEDURE ØPTIØNS(MAIN);
          DECLARE M(100,7);
  /o 1.-  INICIALIZA M CØN CERØS:          o/
        M=0;
  /o 2.-  LECTURA Y CØNTABILIZACIØN:       o/
        ØN ENDFILE (SYSIN) GØ TØ IMPRIMIR;
LEER; GET FILE (SYSIN)  LIST (EDAD,ECIV);
        M(EDAD,ECIV)=M(EDAD,ECIV)+1;      GØ TØ LEER;
  /o 3.-  IMPRIME MATRIZ M:       o/
IMPRIMIR:          I=1;
    IMPRIME:            PUT LIST (M(I,J) DØ J=1 TØ 7);
                        I=I+1;  IF I<=100 THEN GØ TØ IMPRIME;
  /o
        LA SENTENCIA END DEFINE EL FIN DE LA EJECUCIØN
        Y EL FIN DE LAS SENTENCIAS A LA VEZ:
        END.                                            o/
```

3. *Solución en el lenguaje* BASIC

BASIC es un lenguaje orientado al trabajo de terminales de tipo interactivo. Todas las sentencias de un programa escrito en BASIC deben llevar un rótulo, que es el número de secuencia de la misma. Se acostumbra numerar las sentencias de 10 en 10 para facilitar la intercalación de otras nuevas, entre las ya definidas. El programa se escribe en una máquina de teclado, conectada a un computador y en ella aparecen los diagnósticos de error. El tiempo de respuesta comunicándolos es muy corto, lo que permite corregirlos inmediatamente.

La lectura de datos, hecha mediante una sentencia INPUT, no contempla la posibilidad de detectar el fin de éstos. La sentencia INPUT solicita, cada vez que es ejecutada, que los datos se escriban en el teclado de la máquina y no tiene posibilidad de estipular que se ejecute algún fragmento distinto de programa al término de ellos. Por esto, en la adición se ha hecho uso de un truco de programación: la edad 9999999 indicará fin de datos. Al ser detectada esa "marca" se imprimirá la matriz.

Los comentarios se inician con la clave REM (de REMARKS). La declaración de las dimensiones de un arreglo se hace con una sentencia DIM (de DIMENSION). Las asignaciones se efectúan con la sentencia LET, por ejemplo:

```
LET    A = 0
LET    H = N
```

La impresión de arreglos se realiza con la sentencia MAT PRINT y la escritura se obtiene de renglón a renglón en la máquina de escribir.

Finalmente, en la solución se han utilizado las variables E1 y E2 para identificar la edad y el estado civil, respectivamente.

```
SØLUCIØN BASIC
 10  REM
 20  REM  SØLUCIØN BASIC: TABULACIØN DE EDAD, ESTADØ CIVIL.
 30  REM
 40  REM
 50   DIM    M(100,7)
 60  REM
 70  REM      1.- INICIALIZAR M CØN CERØS:
 80  REM
 90   LET    J=1
100   LET    I=1
110     LET    M(I,J)=0
120     LET    I=I+1
130     IF I<= 100 THEN 110
140   LET  J=J+1
150   IF     J<= 7 THEN 100
160  REM
170  REM      2.- LECTURA Y CØNTABILIZACIØN:
180  REM
190   INPUT E1,E2
200     IF E1 = 9999999 THEN 260
210   LET M(E1,E2) = M(E1,E2) +1
220   GØ TØ 190
230  REM
240  REM      3.- IMPRESIØN DE LA MATRIZ M:
250  REM
260      MAT PRINT M
270      END
```

Para que el programa se ejecute, una vez que se han dado todas las sentencias a través de la máquina de escribir, es necesario escribir a máquina la sentencia especial

RUN

que no es parte del programa, sino que pide al computador la ejecución de éste.

4. *Solución en el lenguaje* COBOL

COBOL es un lenguaje apropiado para resolver problemas del tipo llamado "comercial", dado que tiene una estructura sintáctica que permite hacer programas muy narrativos.

Está formado por cuatro divisiones, cada una de las cuales tiene una función determinada que cumplir. Algunas de ellas están compuestas por secciones.

Un esqueleto de programa tendrá la estructura siguiente:

IDENTIFICATION DIVISION

Fundamentalmente para identificar el programa.

ENVIRONMENT DIVISION

Para indicar el medio ambiente en que debe trabajar, computador, unidades lógicas, etc.

DATA DIVISION

Para describir archivos, registros y toda clase de variables que se utilicen en el programa.

PROCEDURE DIVISION

Contiene el algoritmo de solución del problema, traducido a sentencias del lenguaje.

Para insertar comentarios debe ponerse un asterisco (*) en la columna 7. Los nombres de división, sección, etc. e identificadores de sentencias se escriben a partir de la columna 8 y el cuerpo de las sentencias en las columnas 12 a la 72.

```
SOLUCIØN CØBØL

IDENTIFICATIØN DIVISIØN.
PRØGRAM-ID. 'TABULA'.
REMARKS.    TABULACIØN DE EDAD,ESTADØ CIVIL.
ENVIRØNMENT DIVISIØN.
INPUT-ØUTPUT SECTIØN.
     SELECT TARJETA  ASSIGN TØ 'SYSIN'  UTILITY.
     SELECT LISTADØ  ASSIGN TØ 'SYSØUT' UTILITY.
DATA DIVISIØN.
FILE SECTIØN.
FD   TARJETA  RECØRDING MØDE IS F
              LABEL RECØRDS ARE ØMITTED
              DATA RECØRD IS DATØS.
01   DATØS.
     05  EDAD PICTURE  99.
     05  ECIV  PICTURE   9.
     05  FILLER PIC X(78).
*****************************************************************
FD   LISTADØ  RECØRDING MØDE IS F
              LABEL RECØRD IS ØMITTED
```

```
        DATA RECØRD IS LINEA.
01    LINEA.
      05  FILLER          PICTURE X(132).
*****************************************************************
WØRKING-STØRAGE SECTIØN.
77    I                   PICTURE 9(3)    VALUE ZERØS.
77    J                   PICTURE 9       VALUE ZERØS.
01    MATRIZ.
      05  EDADES          ØCCURS 100 TIMES.
          10   ECIVIL     ØCCURS   7 TIMES.
               15   M     PICTURE 9(6).
01    LINEA1.
      05  FILLER          ØCCURS   6 TIMES.
          10   CELDA      PICTURE ZZZZZ9.
*****************************************************************
PRØCEDURE DIVISIØN.
INICIACIØN-DEL-TRABAJØ SECTIØN.
INICIA.
      MØVE 1 TØ J.
DEJA-I-EN-UNØ.
      MØVE 1 TØ I.
CARGA.   MØVE ZERØ TØ M (I, J)  ADD 1 TØ I
             IF I IS LESS ØR EQUAL TØ 100 THEN GØ TØ CARGA.
             ADD 1 TØ J
               IF I IS NØT GREATER 7 THEN GØ TØ DEJA-I-EN-UNØ.
      ØPEN INPUT TARJETA
             ØUTPUT LISTADØ.
*****************************************************************
LECTURA-Y-CØNTABILIZACIØN SECTIØN.
LEE.
      READ TARJETA AT END GØ TØ FIN.
      ADD 1 TØ M (EDAD,ECIV)
      GØ TØ LEE.
*****************************************************************
IMPRESIØN-DE-RESULTADOS SECTION.
FIN.
      MØVE 1 TØ I.
PREPARA.
      MØVE 1 TØ J.
MUEVE.
      MØVE M (I,J) TO CELDA (J) ADD 1 TØ J
        IF J IS NØT GREATER 7 THEN GØ TØ MUEVE.
      WRITE LINEA FRØM LINEA1 AFTER 1
        IF I IS NØT GREATER 100 THEN GØ TØ PREPARA.
CIERR-ARCHIVØS.
      CLØSE TARJETA.
      CLØSE LISTADØ.
      STØP RUN.
```

C. Por último, los *lenguajes orientados al problema.* El uso de los lenguajes de tipo general es recomendable cuando las necesidades de procesamiento de datos caen dentro de un amplio rango de aplicaciones.

En cambio cuando las aplicaciones son mucho más circunscritas y corresponden a un único tipo de problemas, se pueden reducir radicalmente los esfuerzos de programación con un programa que permite abarcar muchos casos parecidos y que tenga, además, una forma simple de introducir los parámetros específicos del problema. Este tipo de programas, independientemente del lenguaje en que se hayan escrito, se conocen como programas *orientados,* con lo que se quiere expresar que son aptos para resolver un determinado tipo de problemas.

Cuando las funciones de los programas orientados son demasiado complejas, puede hacerse difícil entregar los parámetros del problema. Este es el caso de programas orientados a simulación, a resolución de modelos lineales, al cálculo de estructuras, a la producción de tabulaciones, al análisis estadístico, etc. En estas oportunidades los parámetros son tantos que se prefiere crear *lenguajes orientados,* que facilitan la comunicación entre el usuario y el programa orientado.

Los lenguajes orientados, generalmente, siguen reglas parecidas a las de los lenguajes de alto nivel. Es frecuente que dispongan de variables, de constantes, de sentencias condicionales y de ciertos recursos aritméticos. No obstante lo anterior, no debe confundirse el concepto de *lenguaje de alto nivel* con el de *lenguaje orientado.* En tanto el primero sirve a cualquier propósito y es traducido siempre a instrucciones de lenguaje de máquina para la ejecución de los programas del usuario, cada lenguaje orientado permite sólo la realización de tareas dentro de ámbitos específicos y generalmente sólo permite expresar de una manera más cómoda los parámetros de un problema que será resuelto por el programa orientado.

CELADE utiliza, en el manejo de problemas estadísticos, procedimientos y programas orientados de excepcional utilidad. Entre ellos, los programas:

MARGINAL	orientado a la inspección estadística y evaluación de códigos de archivos
CENTS	orientado a la producción de tabulaciones en censos y encuestas de gran tamaño
SPSS y OSIRIS	orientados al análisis estadístico de encuestas.

El primero no posee un lenguaje orientado que lo maneje, de tal manera que es necesario entregarle las características de los archivos a través de tarjetas de control. En cambio, los restantes poseen poderosos lenguajes orientados que ofrecen una gama mayor de posibilidades de programación. Estos lenguajes se denominan con los mismos nombres de los programas.

CELADE ha desarrollado un programa, dotado de lenguaje orientado, el CONCOR, para cubrir el área de conversión y corrección automática de errores para masas de datos estadísticos.

V. ESTRUCTURACION DE PROGRAMAS Y MODULARIDAD

La estructuración o diseño de programas es, tal vez, el concepto más importante que debe tener siempre presente el programador. Una buena estructuración implica mayor posibilidad de funcionamiento del programa como asimismo un costo menor. Por otra parte, los programas bien diseñados son más fáciles de escribir, documentar, revisar, cambiar, entender, depurar, probar y mantener.

¿Qué es lo que hace que un programa esté bien estructurado? La respuesta, si bien es simple, es un desafío a la capacidad creadora del programador; un programa está bien estructurado si consta de dos tipos de rutinas:

i) Monitores o módulos de control.

ii) Subrutinas, cada una de las cuales realiza una función bien específica, parte de la función total del programa.

Esta definición encierra un concepto nuevo, que es el de *modularidad*, entendiéndose por tal la estructuración de un programa en *módulos* y llamando módulo a un grupo de instrucciones que realizan una función claramente definida, específicamente relacionada a la función lógica del programa.

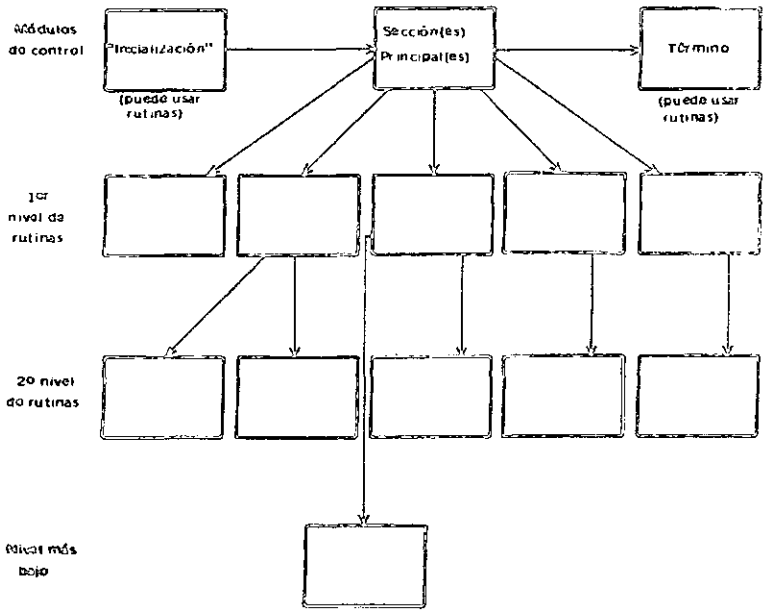
Profundizando lo anterior, se tiene que: el módulo de control debe dar una visión rápida acerca de la función total del programa. Para ello debe estar constituido fundamentalmente por llamadas a las subrutinas, que son módulos que cumplen con tareas particulares.

Se puede decir que el módulo de control está ubicado en el primer nivel de jerarquía en la estructura del programa y en su diseño es donde debe concentrar su esfuerzo el programador. Aquí es donde podrá ver claramente el panorama total del programa y por ello en esta etapa no debe entrar al detalle. Se recomienda que el diagrama de bloques correspondiente al módulo de control no exceda de los veinte bloques. Esto tiene como ventaja el poder repensar y rediagramar fácilmente el programa.

Las subrutinas en cuanto a estructura, son modelos del programa, así como éste lo es del sistema completo. Esto significa que la subrutina también puede consistir en un módulo de control en el que se hacen llamados a módulos de un nivel más bajo.

Pero es importante dejar en claro o insistir en el hecho de que modularidad no significa hacer uso de rutinas porque sí. Estas deben cumplir ciertos requisitos. Se dijo anteriormente que la rutina tiene una función específica, parte de la función lógica total. Hay que agregar que no debe haber, o debe haber muy poca, interacción con otros módulos; en otras palabras, no es dependiente de lo que ocurra en ellos, sino solamente de la información que se le entregue y de la función que cumpla. Esto significa que puede ser revisado y especificado en detalle sin considerar al resto, lo que trae como consecuencia que cada módulo podría ser programado por personas distintas. Finalmente, no deben entregarse a rutinas funciones que corresponden al módulo de control.

Recurriendo al diagrama de bloques, se puede representar claramente la estructura de un programa diseñado con el concepto de modularidad.



Es probable que el problema que se resolvió con FORTRAN, BASIC, PL/I y COBOL no sea el mejor ejemplo para mostrar la aplicación del concepto de modularidad, sin embargo, se ha tomado la solución COBOL y de ella la división de procedimiento para enfocar el mismo problema con la ayuda de las nuevas ideas sobre estructura de programas.

SOLUCIØN CØBØL

```
PRØCEDURE DIVISIØN.
     PERFØRM INICIA-Y-DEJA-CERØ-EN-M THRU LEE-Y-CØNTABILIZA.
     PERFØRM LEE-Y-CØNTABILIZA THRU TERMINA.
     PERFØRM IMPRIME-RESULTADØS THRU FUERA.
     GØ TØ FIN.
*****************************************************************
INICIA-Y-DEJA-CERØ-EN-M.
     MØVE 1 TØ J.
DEJA-I-EN-UNØ.
     MØVE 1 TØ I.
CARGA.  MØVE ZERØ TØ M (I,J)    ADD 1 TØ I
             IF I IS LESS ØR EQUAL TØ 100 THEN GØ TØ CARGA.
             ADD 1 TØ J
                IF I IS NØT GREATER 7 THEN GØ TØ DEJA-I-EN-UNØ.
     PERFØRM ABRE-ARCHIVØS.
*****************************************************************
LEE-Y-CØNTABILIZA.
     READ TARJETA AT END GØ TØ TERMINA.
          ADD 1 TØ M(EDAD,ECIV)
     GØ TØ LEE-Y-CØNTABILIZA.
TERMINA.
     EXIT.
*****************************************************************
IMPRIME-RESULTADOS.
     MØVE 1 TØ I.
PREPARA.
     MØVE 1 TØ J.
MUEVE.
     MØVE M (I,J) TØ CELDA (J) ADD 1 TØ J
        IF J IS NØT GREATER 7 THEN GØ TØ MUEVE.
     WRITE LINEA FRØM LINEA 1 AFTER 1
        IF I IS NØT GREATER 100 THEN GØ TØ PREPARA.
     PERFØRM CIERRA-ARCHIVØS.
FUERA.
     EXIT.
*****************************************************************
ABRE-ARCHIVØS.
     ØPEN INPUT TARJETA
          OUTPUT LISTADØ.
*****************************************************************
CIERRA-ARCHIVØS.
     CLØSE TARJETA.
     CLØSE LISTADØ.
*****************************************************************
FIN.
     STØP RUN.
*****************************************************************
```

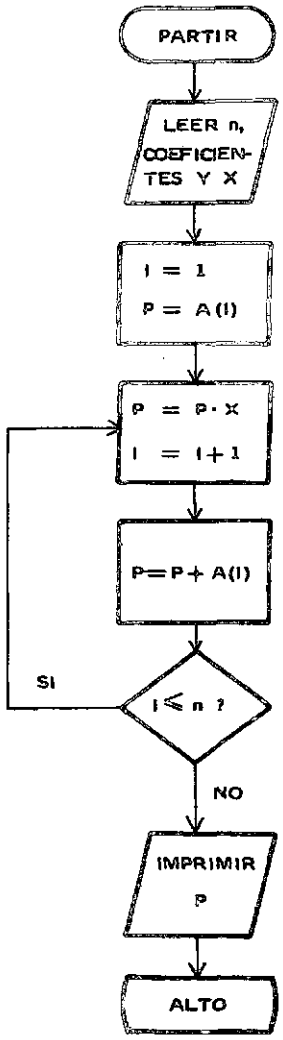
VI. PROBLEMAS RESUELTOS

1. Evaluar el polinomio de orden *n*

$$P = a_1 x^n + a_2 x^{n-1} + \ldots + a_n x + a_{n+1}$$

Solución: El polinomio

$$P = (((a_1 x + a_2)x + a_3)x + \ldots + a_n)x + a_{n+1}$$



2. Encontrar todos los números de tres dígitos que sean iguales a la suma de los cubos de sus dígitos.

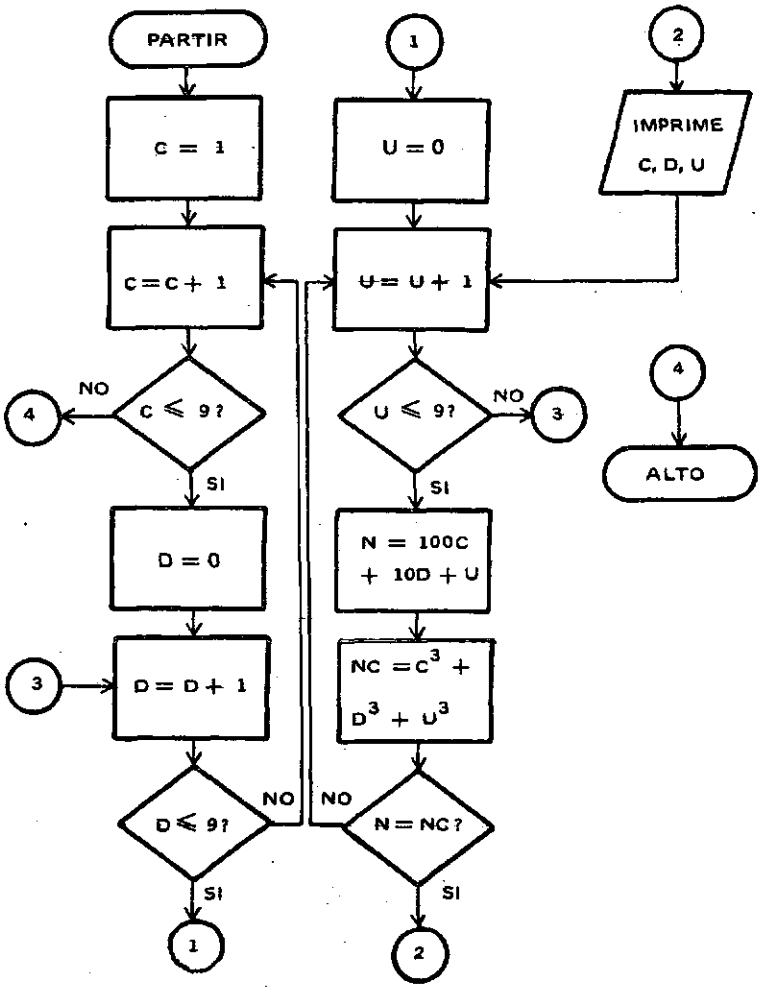
Solución: Si se denominan C, D y U a los dígitos del número, el número será:

$$N=100C+10D+U$$

por otra parte, la suma de los cubos de los dígitos corresponde a:

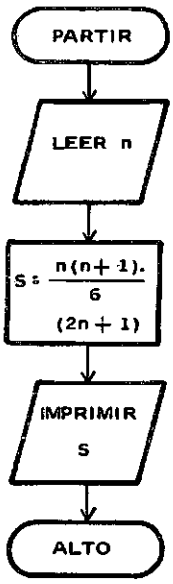
$$NC=C^3+D^3+U^3$$

Se trata de obtener que: N sea igual a NC



3. Encontrar la suma de los cuadrados de los primeros 101 enteros positivos mediante la fórmula general

$$S = \frac{n(n+1)\ (2n+1)}{6}$$ donde n es el número de término

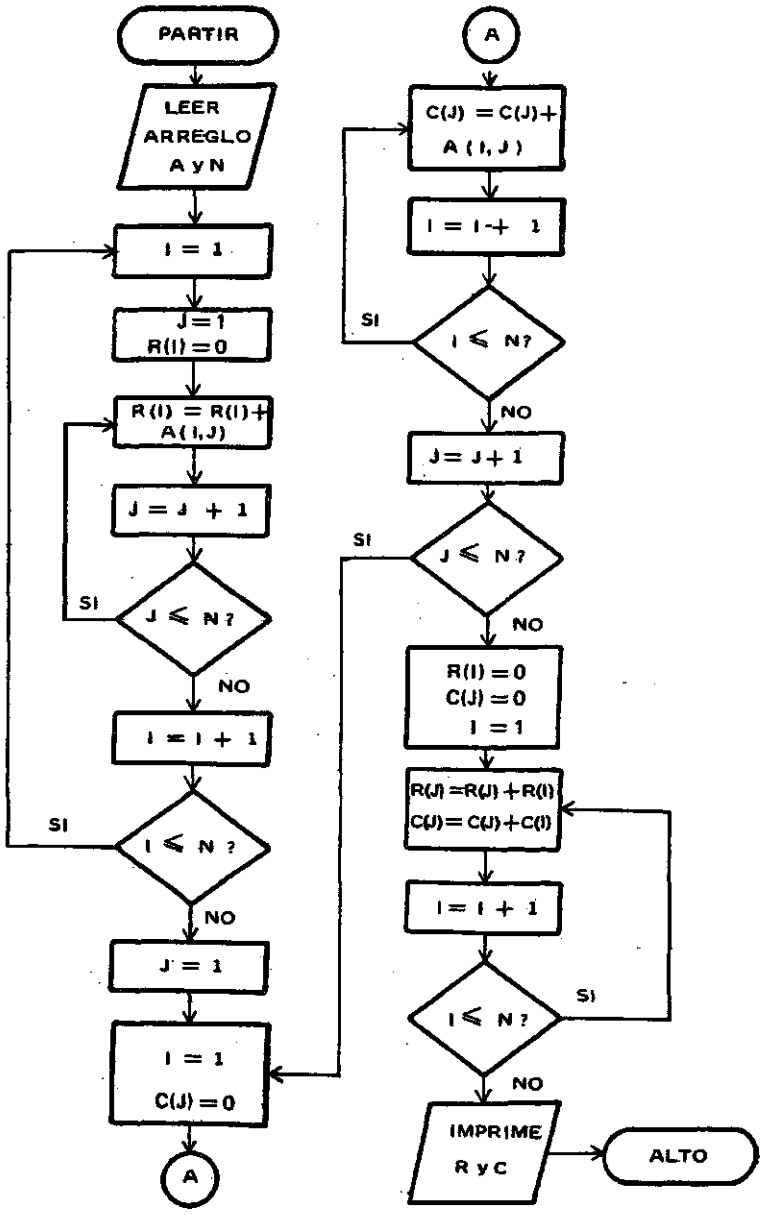


Compárese con la solución dada en el ejemplo 4 de este capítulo.

4. Se tiene un arreglo A de n*n elementos. Se pide sumar todos los elementos de cada renglón para obtener subtotales R_i y todos los elementos de cada columna para obtener subtotales C_j. Verificar que R_{n+1} sea igual a C_{n+1}.

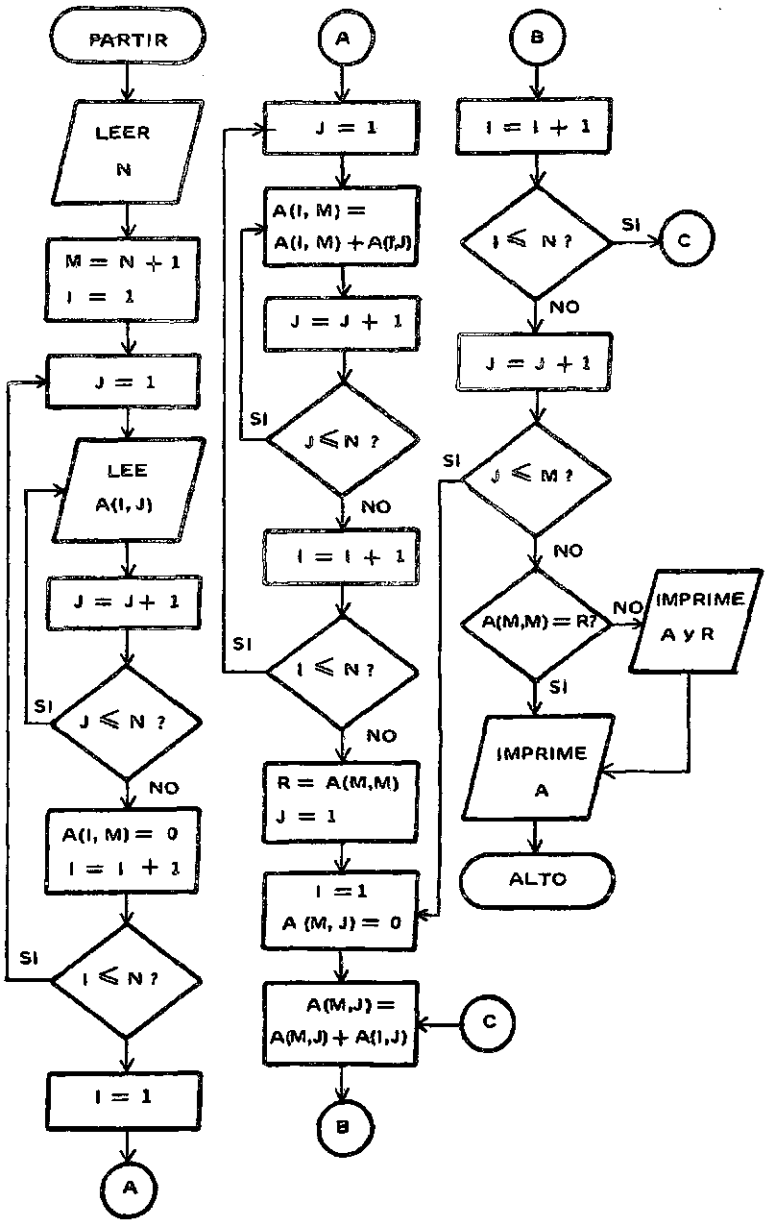
$$\begin{array}{cccccc} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & R_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & R_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & R_3 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & R_n \\ C_1 & C_2 & C_3 & \cdots & C_n & \boxed{\begin{matrix} R_{n+1} \\ C_{n+1} \end{matrix}} \end{array}$$

a) Solución en que los arreglos R y C se construyen aparte de A.



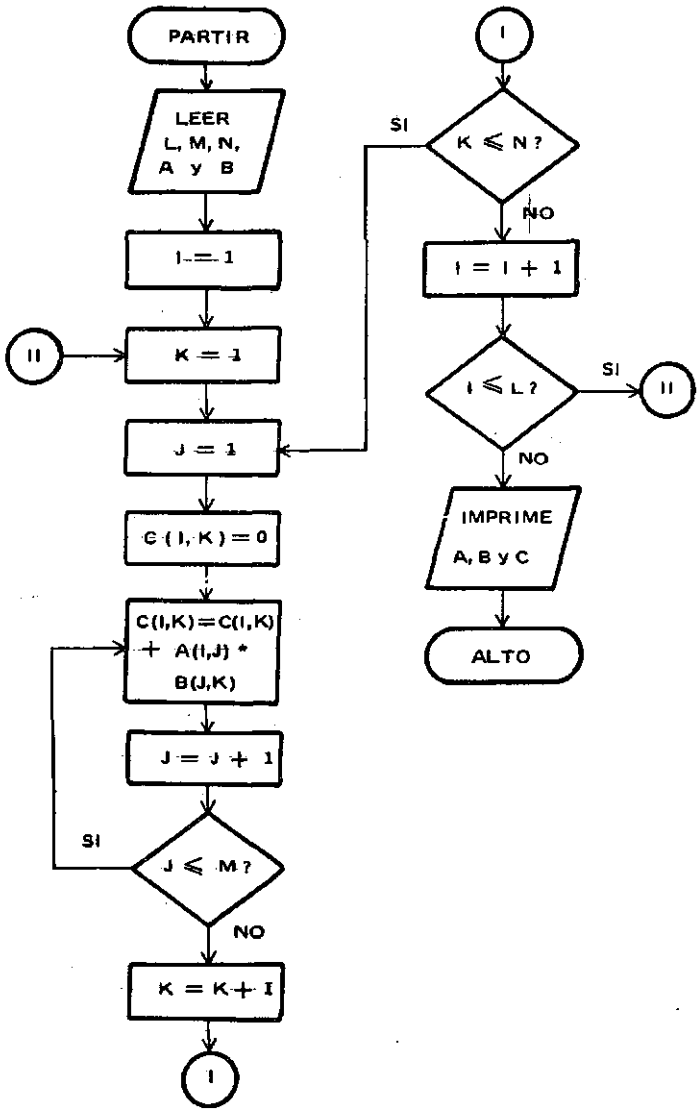
En esta solución no se ha incluido la verificación de la igualdad de R_{n+1} y C_{n+1} dado que ella se puede realizar visualmente con el resultado de la impresión.

b) Solución en que los subtotales por renglón y por columna son elementos de un arreglo A aumentado.



5. Calcular el producto de dos arreglos A (16x20) y B (20x25) a base de la fórmula del elemento general de la matriz resultante C:

$$c_{i,k} = \sum_{j=1}^{n} a_{i,j}\, b_{j,k}$$



6. Se tiene una serie de N puntos $(X_1, Y_1), (X_2, Y_2) \ldots (X_n, Y_n)$. Se pide encontrar la ecuación de la recta que pasa por los puntos. La ecuación de la recta es:

$$Y = A_0 + A_1 X$$

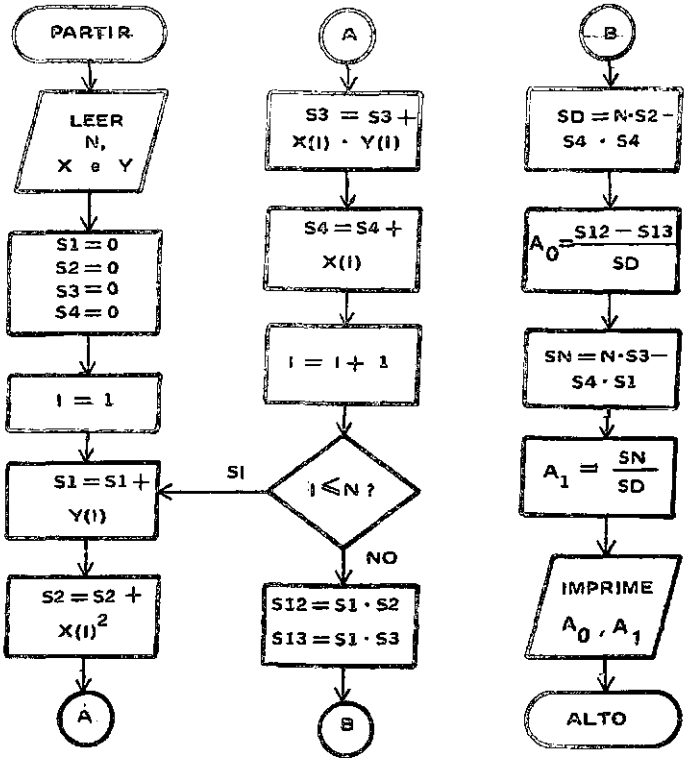
donde:

$$A_0 = \frac{\Sigma Y \Sigma X^2 - \Sigma Y \Sigma XY}{N\Sigma X^2 - (\Sigma X)^2}$$

$$A_1 = \frac{N\Sigma XY - \Sigma X \Sigma Y}{N\Sigma X^2 - (\Sigma X)^2}$$

Con el objeto de simplificar la escritura se ha utilizado la notación $\Sigma X, \Sigma XY, \Sigma X^2$, etc., en vez de:

$$\sum_{i=1}^{M} X_i, \quad \sum_{i=1}^{M} X_i Y_i, \quad \sum_{i=1}^{M} X_i^2, \text{ etc.}$$



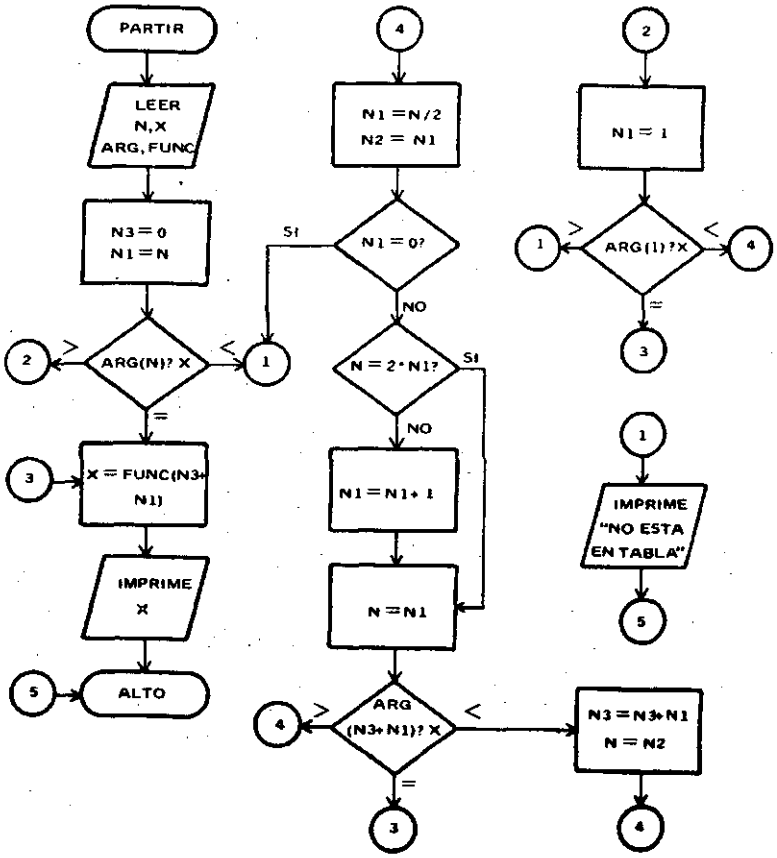
7. Se tienen dos arreglos, ARGUMENTOS Y FUNCIONES cada una con N elementos (N puede ser par o impar). Se pide buscar un dato X en el primer arreglo con la técnica siguiente:

a) Se compara el dato con el elemento que está en N/2

b) Si es menor, se compara con el que está en N/4 y así sucesivamente.

c) Si es mayor, se compara con el que está en $\frac{3}{4}$ N, etc.

d) Si es igual, el dato X se reemplaza por el elemento correspondiente del segundo arreglo.



VII. PROBLEMAS PROPUESTOS

1. Hallar la suma de 500 términos de la progresión aritmética cuyo primer término es 5 con incremento 4, esto es, 5, 9, 13, 17, ...
Utilizar la fórmula:

$$S_n = \frac{[\,2 \cdot U_1 + (n-1)d\,]\,n}{2}$$

donde: U_1 = primer término
n = número de términos
d = incremento
S_n = sumatoria

2. Calcular las raíces de una ecuación de segundo grado del tipo

$$AX^2 + BX + C = 0$$

3. Resolver el sistema de ecuaciones

$$\begin{aligned} AX + BY &= C \\ DX + EY &= F \end{aligned}$$

4. Calcular e^X a base de la fórmula

$$e^X = 1 + \frac{X}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \cdots$$

detener el cálculo cuando el último término sea menor que 10^{-7}.

5. Calcular el valor de π de acuerdo a la fórmula indicada por un dato leído N

a) Si N=1

$$\text{con } \frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

b) Si N=2

$$\text{con } \frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \cdots$$

c) Si N=3

$$\text{con } \frac{\pi}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \frac{8}{7} \cdot \frac{8}{9} \cdots$$

6. Si se tienen los dos primeros elementos de un arreglo N(1)=0 y N(2)=1. Calcular hasta el término N(1000) a base de la fórmula

$$N(I) = N(I-2) + N(I-1)$$

(Serie de Fibonacci)

7. Se tienen dos arreglos A y B, cada uno con 50 elementos. Se pide calcular

$$D=\sqrt{\sum_{i=1}^{50}(A_i - B_i)^2}$$

8. Se tiene un arreglo A con 50 elementos. Se pide transformarlo de acuerdo con:

$$a_i = a_i \cdot i$$

9. Se tiene un arreglo X de 50 por 50 elementos. Se pide encontrar el mayor de los elementos.

10. Se tiene un arreglo Y de 20 por 20 elementos. Se pide transformarlo en un arreglo YY de 400 elementos.

11. Se tiene un arreglo Z de 40 por 40 elementos. Se pide calcular

$$S=\sum_{i=1}^{40} Z_{ii}$$

12. En el arreglo anterior, se pide intercambiar el renglón K con la columna L. K y L son datos que deben ser leídos.

BIBLIOGRAFIA

1. CELADE, *Curso de Introducción al Procesamiento Electrónico de Datos (PED) para Cientistas Sociales*, Santiago, Chile, 1974, 117 p.
2. Gleim, George A., *Program Flowcharting*, San Francisco, Rinehart, 1970, 71 p.
3. IBM System Products Division, *Técnicas de diagramación*, Nueva York, 1970, 62 p.
4. Keenan, Thomas A., Forsythe, Alexander I., Organick, Elliot I. y Stenbert, Warren, *Lenguajes de diagramas de flujo*, México, Limusa/Wiley, 1973, 588 p.
5. Packer, David W., "Effective Program Design", en *Computer and Automation*, julio, 1970, vol. 19, N° 7, p. 37.

FORTRAN IV

I. INTRODUCCION

Entre los lenguajes que se utilizan para comunicación con los computadores, uno de los más conocidos es el lenguaje FORTRAN (FORmula TRANslation). Es un lenguaje simbólico de tipo general que permite resolver con facilidad la representación de algoritmos para solución de problemas científicos en términos de instrucciones al computador.

La gran mayoría de los computadores posee una versión de FORTRAN, de tal manera que un programador puede procesar sus problemas en distintos computadores sin que ello le signifique hacer cambios notables en la estructura de sus programas. Fundamentalmente, las diferencias estarán relacionadas con los dispositivos de entrada y salida de datos y con las instrucciones respectivas. Si se han diseñado los programas en forma modular, los cambios se referirán principalmente al módulo de entrada o al módulo de salida o a ambos, dejando el resto de los módulos, que son el cuerpo del programa, con su estructura original.

Es importante señalar que al aprender un lenguaje de este tipo, es bastante fácil adquirir posteriormente el dominio de otros lenguajes de la misma categoría, dado que las instrucciones básicas, si no son iguales, al menos poseen la misma lógica de funcionamiento.

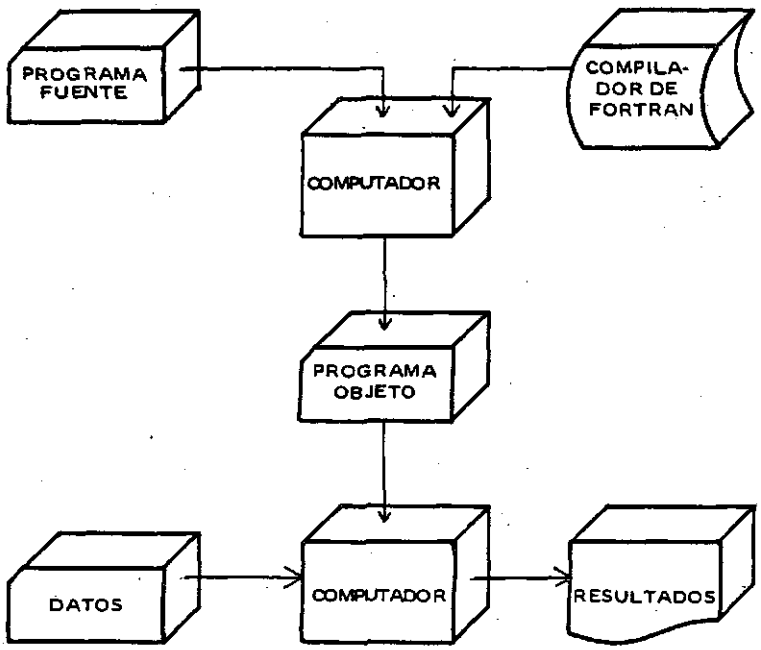
En relación al proceso de los programas, es necesario que se cumplan dos etapas:

a) Compilación o traducción del programa escrito en FORTRAN al lenguaje de la máquina que se va a utilizar; y
b) Ejecución del programa traducido a lenguaje de máquina.

En la etapa de compilación, el programa escrito en FORTRAN (programa fuente) desempeña el papel de datos, los que procesará un programa escrito en lenguaje de máquina denominado COMPILADOR.

El resultado de este proceso es el mismo programa, pero ahora traducido a lenguaje de máquina (programa objeto) y puede obtenerse en tarjetas perforadas, en cinta magnética, en disco magnético, etc., o también puede dejarse en la misma memoria de trabajo.

En la etapa de ejecución, el programa objeto procesa los datos del problema y entrega los resultados requeridos.



A continuación se describe un FORTRAN IV casi completo y en el APENDICE B se da a conocer el resto de los adelantos que tiene el lenguaje FORTRAN IV COMPLETO, aun cuando sólo están enunciados.

II. ELEMENTOS DEL LEGUAJE

Un programa escrito en un lenguaje simbólico está compuesto por proposiciones, ordenadas secuencialmente de acuerdo con lo estipulado en el algoritmo de solución del problema. Su ejecución se efectúa en el mismo orden en que se encuentran, es decir, una a continuación de la otra.

En el lenguaje FORTRAN las proposiciones se pueden agrupar en la forma siguiente:

a) de entrada y salida
b) de asignación
c) de control
d) de especificación
e) de subprograma.

Los tipos a), b) y c) se denominan *ejecutables* y las de los tipos d) y e), *no ejecutables.*

Para que el programa pueda ser leído por el computador, se perfora normalmente en tarjetas y para ello es necesario escribir las proposiciones en hojas de codificación apropiadas.

1. *Hoja de codificación*

La hoja de codificación tiene líneas con capacidad para ochenta caracteres, lo que significa que cada línea puede ser vaciada en su totalidad en una tarjeta.

Se puede subdividir la línea en campos:

Campo 1 formado por columnas 1 a 5

Campo 2 formado por columna 6

Campo 3 formado por columnas 7 a 72

Campo 4 formado por columnas 73 a 80

a) *Campo 1* (columnas 1 a 5). Se utiliza para especificar un número que identifique a la proposición. Puede variar dicho número desde 1 hasta 99999 y no puede haber dos o más proposiciones, en el mismo programa, con idéntico número.

La ubicación del número dentro del campo es arbitraria dado que una columna en blanco (ø) no se interpreta. Se tiene así que todas las líneas siguientes

1 2 3 4 5	
1 5	
1 5	
1 5	
1 5	
1 5	

contienen el número 15 como identificación.

Aun cuando todas las líneas pueden llevar número de identificación, se evita hacerlo porque eso significa emplear más tiempo en la etapa de compilación. Se identifican entonces sólo aquellas líneas que van a ser referidas.

El número de identificación sólo cumple la función de identificar. Luego, su magnitud no implica prioridad en la ejecución de la proposición.

Comentarios: Parte de la documentación de un programa se puede obtener a través de explicaciones internas acerca de lo que hace el programa o una parte de él, la forma en que deben especificarse los datos y el orden en que deben ser colocados, etc. Esto se logra escribiendo la letra

C en la columna 1, con lo cual el compilador no procesa la línea completa y sólo se imprime en el listado que se obtiene del programa fuente.

Ejemplo 1:

```
C  EL CØMENTARIØ PERMITE
C  DØCUMENTAR LØS PRØGRAMAS
C  EN FØRMA INTERNA.
```

La letra O se cruza para no confundirla con el Nº 0.

b) *Campo 2* (columna 6). Este campo contiene normalmente el carácter blanco o cero (el espacio en blanco es un carácter y como tal tiene su representación dentro del computador). Se utiliza cuando la proposición no cabe en la línea; en ese caso, se codifica cualquier carácter distinto de blanco o cero en la columna 6 de la línea siguiente para indicar que la proposición continúa en ella.

Puede haber hasta 19 líneas de continuación correspondientes a una sola proposición, pero NO PUEDE HABER MAS DE UNA PROPOSICION POR LINEA.

c) *Campo 3* (columnas 7 a 72). Es el que contiene la proposición propiamente tal. Con el objeto de dar más claridad a ésta *es posible intercalar blancos en la medida que desee el programador.* Los blancos son ignorados por el compilador excepto cuando ellos forman parte de un dato literal, en cuyo caso son considerados y tratados como blancos.

d) *Campo 4* (columnas 73 a 80). Este campo no es significativo para el compilador FORTRAN, por lo tanto es posible utilizarlo como parte de un comentario para identificar el programa o para verificar la secuencia de las tarjetas.

Ejemplo 2:

```
C  EJEMPLØ 2.
C  ESTE PRØGRAMA LEE DØS
C  DATØS, REALIZA UN CALCULØ,
C  IMPRIME EL RESULTADØ Y
C  SE    DETIENE.
      READ (1,10) B,C
      A = B + C
      WRITE (3,20) A
      STØP
10    FØRMAT(F6.2,F6.2)
20    FØRMAT(F10.3)
      END
```

Las cuatro primeras líneas del programa anterior son de comentario. A continuación cuatro sentencias ejecutables: una de ,entrada (READ), una de asignación, una de salida (WRITE) y una de control (STOP). Finalmente, dos proposiciones de especificación y una de control (END) que indica el término del programa fuente.

Las proposiciones de especificación FORMAT son las únicas de especificación que pueden ir en cualquier parte del programa. Las res-

tantes deben ir siempre al comienzo. Aquellas indican la estructura que deben tener los datos de entrada o la forma en que se imprimirán o grabarán los resultados, de tal modo que siempre se mencionan con una o más proposiciones de entrada o salida.

2. *Constantes*

Una constante es un valor escrito en el programa fuente. Como su nombre lo indica, es un elemento que permanece fijo, invariable.

Hay cinco tipos de constantes:

ENTERAS
REALES
LOGICAS
HEXADECIMALES
LITERALES

Las reales pueden ser estipuladas con PRECISION SIMPLE o con DOBLE PRECISION.

Constante ENTERA. Es un número decimal escrito SIN punto decimal. Ocupa cuatro bytes en memoria y su magnitud máxima es:

$2^{31} - 1$, que es igual a 2147483647

Ejemplo 3:

i) Constantes enteras *válidas:*

0 +99999 175 −2147483647

(el signo más (+) puede ser omitido).

ii) Constantes enteras *no válidas:*

0 99,999 .5 2147483649

Constante REAL. Puede tener una de las tres formas siguientes:

BASICA: es un número decimal escrito CON punto decimal. Ocupa cuatro bytes en memoria. Si es positivo, el signo puede ser omitido.

BASICA seguida de EXPONENTE DECIMAL: el exponente decimal consiste de la letra E o la letra D seguida de una constante entera de uno o dos dígitos, con o sin signo. La letra E especifica SIMPLE precisión (cuatro bytes en memoria), la letra D indica DOBLE precisión (ocho bytes en memoria), y se interpretan como "diez elevado a".

Constante ENTERA seguida de EXPONENTE DECIMAL.

Magnitudes: 10^{-78} (16^{-65}) hasta 10^{75} (16^{63}).

El valor 10^{-78}, para los efectos de cálculo, se considera equivalente a cero.

Precisión: En cuatro bytes se pueden representar seis dígitos hexadeci-

males (siete dígitos decimales). En ocho bytes se pueden representar catorce dígitos hexadecimales (dieciséis dígitos decimales).

Ejemplo 4:

i) Constantes reales *válidas:*

+0.0
−100.845
9999.999
1234567.E+14 esto es 1234567. por 10^{14}.
−5.4E+02, −5.4E02 esto es −5.4 por 10^{2}
−5.4E2, −54.E1 esto es −54. por 10^{1} } esto es −540.
−54E1
1234567890.123456
8.7D+02, +8.7D2, 8.7D02 esto es 8.7 por $10^{2} = 870.$

ii) Constantes reales *no válidas:*

0	no tiene punto
3,1416	tiene coma
16.28E·	no tiene la constante entera del exponente
−15.3D−97	excede la magnitud permitida
828.524.627	tiene más de un punto
−4.5D78	excede la mangitud permitida

Constante LOGICA. Esta constante especifica un valor *lógico*, "verdad" (true) o "falso" (false). Solamente hay dos constantes lógicas:

.TRUE.
.FALSE.

Cada una de estas constantes ocupa cuatro bytes en memoria. Cuando se asigna una constante lógica a una variable lógica (ver "Tipos y longitudes de variables"), se está especificando que dicha variable tomará el valor TRUE o el valor FALSE.

Al escribir la constante debe ser precedida y seguida por punto.

Constante HEXADECIMAL. Es un número hexadecimal precedido por la letra Z.

Un byte contiene dos dígitos hexadecimales. Si el número contiene un número impar de dígitos, se agrega un cero hexadecimal a la izquierda del número.

Si la longitud de la variable que va a contener a la constante es mayor que la necesaria, se rellena con ceros hexadecimales por la izquierda; si la longitud es menor que la necesaria, se trunca el número por la izquierda.

Constante LITERAL. Es una cadena de caracteres alfabéticos, numéricos y/o especiales que se puede expresar en las formas siguientes:

a) Encerrada entre apóstrofos

b) Precedida por wH en que *w* es el número de caracteres que contiene la cadena.

Cada carácter requiere un byte de almacenamiento. El número de caracteres en la cadena no puede ser mayor que 255.

```
C  EJEMPLØ 5.
C  EN ESTE PRØGRAMA SE USAN
C  CØNSTANTES ENTERAS Y
C  REALES.
        READ (1,10) B,C
C  SE MULTIPLICA B PØR LA
C  CØNSTANTE REAL 3.1416
        A = B * 3.1416
C  SE DIVIDE C PØR LA
C  CØNSTANTE ENTERA 2
        D = C / 2
        WRITE (3,20) A,D
        STØP
   10   FØRMAT (F6.2,F6.2)
   20   FØRMAT (F10.3,F10.3)
        END
```

3. *Nombres simbólicos*

Son identificadores formados por uno a seis caracteres alfanuméricos, esto es, alfabéticos (A a Z y $ que se incluye en este grupo) o numéricos (0 a 9). El primer carácter debe ser alfabético.

Ejemplo 6:

i) Nombres *válidos:*

A
ZETA1
$152
NUMERO

ii) Nombres *no válidos:*

A-B	carácter extraño, el guión
ZETA.	carácter extraño, el punto
4ALFA	empieza con carácter numérico
TEMP235	tiene más de seis caracteres

4. *Variables*

Tiene el mismo significado que en álgebra. Una variable es un símbolo que representa uno de muchos valores numéricos o uno de los dos valores lógicos.

En los problemas que se han visto, A, B, C y D son variables. Las variables se identifican con un *nombre simbólico*, el cual puede servir también como ayuda en la documentación del programa si es que se elige con una significación adecuada. Por ejemplo, el área de un rectángulo se puede calcular con la expresión siguiente:

$$S = A * B$$

pero es mucho más significativo escribir:

$$AREA = ANCHØ * LARGØ$$

A. *Tipos y longitudes de variables*

Los tipos de variables son los mismos que los tipos de constantes, esto es:

–ENTERAS
–REALES
  –SIMPLE PRECISION
  –DOBLE PRECISION
–LOGICA

A cada tipo de variable le corresponde una longitud normal y una opcional, las cuales determinan la cantidad de bytes en memoria que ocupará el valor. La longitud opcional debe ser declarada mediante una sentencia de especificación.

Tipo de variable	*long. normal*	*long. opcional*
Entera	4	2
Real	4	8
Lógica	4	1

Existen tres formas que permiten declarar el tipo de una variable:
–Especificación predefinida
–Mediante proposición IMPLICIT
–Mediante proposiciones de especificación.

a) *Especificación predefinida*

Se obtiene a través del nombre de la variable. Si el primer carácter del nombre es: I,J,K,L,M o N, el tipo de la variable queda automáticamente definido como ENTERO; cualquier otro carácter define una variable de tipo REAL con SIMPLE PRECISION. Tanto las variables enteras como las reales ocupan cuatro bytes en memoria.

Se puede observar que con la especificación predefinida no es posible obtener variables de tipo real con doble precisión y tampoco variables lógicas. En este caso es necesario recurrir a las proposiciones de especificación explícitas.

b) *Proposición* IMPLICIT

Al igual que en la especificación predefinida, la proposición IMPLICIT declara el tipo de la variable haciendo uso del primer carácter del nombre. Sin embargo, el programador tiene la opción de asignar un rango de caracteres alfabéticos a un tipo determinado de variable.

La proposición IMPLICIT anula los efectos de la especificación predefinida.

La estructura completa de la proposición como asimismo ejemplos de su utilización se verán en el capítulo "Proposiciones de especificación".

c) *Proposiciones de especificación de tipo*

Las proposiciones de especificación de tipo declaran explícitamente el tipo de una o más variables. Su efecto anula el de la proposición IMPLICIT y el de la especificación predefinida.

La estructura completa de las proposiciones como también ejemplos de la forma de utilización se verán en el capítulo "Proposiciones de especificación".

5. *Arreglos*

Si se trabaja con gran cantidad de variables, el problema mayor que se presenta en su manejo es la identificación de cada una de ellas. Junto con tener que crear el nombre es conveniente pensar en que él sea significativo, y es bastante difícil, aparte de la pérdida de tiempo que ello implicaría, asignarles nombre a cada una de tres mil, cuatro mil o más variables, como sería el caso de aquellos problemas de tipo estadístico en que el volumen de información es cuantioso. Además, sería necesario repetir instrucciones que permiten efectuar una operación con un dato, para cada uno de los datos que va a ser procesado. Por ejemplo, si se va a acumular la suma de cinco datos A, B, C, D y E en un contador S, las instrucciones serán:

```
S = A
S = S + B
S = S + C
S = S + D
S = S + E
```

La notación matemática del problema es sencilla:

$$S = \sum_{i=1}^{i=5} A_i$$

o lo que es lo mismo:

$$S = A_1 + A_2 + A_3 + A_4 + A_5$$

En rigor, lo que se ha hecho es sumar los elementos de un arreglo de nombre A. El nombre se hace extensible a los componentes del arreglo, pero además, a cada uno de ellos se lo identifica en forma concreta con el subíndice.

El subíndice indica realmente la ubicación que tiene el elemento dentro del conjunto. Se llama subíndice para diferenciarlo del superíndice que se escribe a la derecha sobre la variable, en la misma forma que un exponente.

El conjunto o arreglo recibe en matemáticas el nombre de VECTOR LINEA o VECTOR COLUMNA, dependiendo de si está escrito horizontal o verticalmente.

En FORTRAN se utiliza el mismo criterio expuesto antes. El nombre genérico es un nombre simbólico y para hacer referencia a un elemento determinado del arreglo se indica su posición, encerrada entre paréntesis, a continuación del nombre. En el ejemplo visto es necesario un subíndice para especificar la ubicación del dato. Se trata entonces de un arreglo unidimensional o lineal.

Ejemplo 7:

Se tiene la siguiente lista de datos:

1	35	−6	28	9

Si el nombre genérico que se asigna a este arreglo es A, los elementos serán:

A(1) = 1
A(2) = 35
A(3) = −6
A(4) = 28
A(5) = 9

La ubicación en memoria, de estos datos, será uno a continuación del otro, en orden ascendente según el índice.

Debido al uso de subíndices las variables toman el nombre de VARIABLES SUBINDICADAS o variables con índice.

Considérese ahora el siguiente arreglo de datos:

Columnas

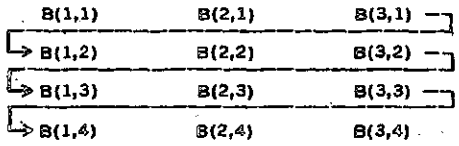
	1	2	3	4
renglón 1	51	4	36	2
renglón 2	−5	13	24	−1
renglón 3	10	25	−3	−14

Este es un arreglo bidimensional de tres por cuatro elementos. Si a este arreglo se le llama MATRIZ (igual que la denominación que se da en matemáticas a este tipo de conjuntos), éste será el nombre genérico de los doce elementos y para referirse a alguno en particular se indicará a continuación de él, entre paréntesis, primero el renglón en el que se encuentra y después la columna correspondiente. Ambos separados entre sí por coma.

Ejemplo 8:

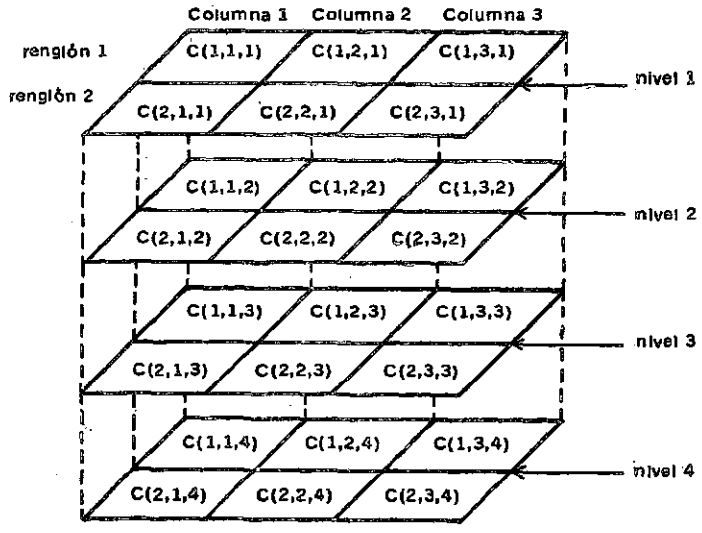
MATRIZ (3,4) es el nombre de la variable que pertenece al arreglo llamado MATRIZ y que está ubicada en el renglón 3 y en la columna 4.

Para saber la ubicación que tendrá en memoria cada uno de los elementos respecto a los demás, se hace variar "más rápido el índice de la izquierda que el de la derecha". Se tiene así, por ejemplo, en un arreglo B de tres por cuatro elementos, donde la flecha indica la secuencia de almacenamiento, que éste será:

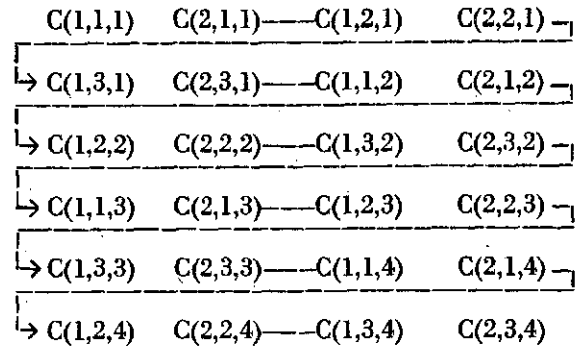


Nótese que este ordenamiento corresponde justamente al inverso del utilizado en álgebra.

Un arreglo tridimensional, supóngase C, de dos por tres por cuatro elementos se puede representar como se indica a continuación:



Su ubicación en memoria se obtendrá haciendo variar "rápido el índice de la izquierda, lento el del centro y más lento el de la derecha". Se obtiene entonces:



A. *Subíndices*

Se ha visto en los ejemplos de arreglos que el subíndice es un elemento que permite ubicar a un dato dentro del arreglo. Dado que la ubicación no puede ser fraccionada, el subíndice debe ser un número ENTERO.

FORTRAN NO ACEPTA SUBINDICES NEGATIVOS NI DE VALOR CERO.

FORTRAN IV Básico acepta un máximo de TRES subíndices, lo que equivale a arreglos de tres dimensiones. Los subíndices pueden tener una de las siguientes SIETE formas:

a) V
b) C'
c) V + C'
d) V – C'
e) C * V
f) C * V + C'
g) C * V – C'

donde:

V es una variable entera, sin signo y sin subíndices

C y C' son constantes enteras sin signo.

Cualquiera que sea la forma de subíndice utilizada, el resultado evaluado no debe sobrepasar la dimensión correspondiente a ese subíndice.

Ejemplo 9:

i) Variables subindicadas *válidas:*

ARRAY (IHOLD)
NEXT (18)
MATRIZ (I + 2)
L (I–5)
A (6 * L)
Z (2 * J + 1)
ALFA (4 * M – 3)

ii) Variables subindicadas *no válidas:*

ARRAY (–L)	la variable debe ser sin signo
LISTA (I–2.)	la constante debe ser entera
MATRIZ (–7*J)	la constante debe ser sin signo
W (M(3))	la variable debe ser sin subíndices
NEXT (0)	el subíndice no debe ser cero
PAGO (J*2)	la constante debe preceder a la variable
TOTAL (2+K)	la variable debe preceder a la constante

FORTRAN IV Completo acepta hasta SIETE dimensiones. En cuanto a los subíndices, pueden contener:

a) expresiones aritméticas (ver "Expresiones")
b) variables subindicadas
c) resultados reales que se convierten a enteros
d) referencias a funciones (ver "Subprogramas")

Ejemplo 10:

TABLA (A*2+3,B/5)
TEMP (I(5)-2,C(I2,j))
BETA (A+3.8)

B. *Reserva de Memoria para los arreglos*

Para poder reservar memoria a los arreglos, sean éstos de *datos* o de *resultados*, el compilador debe conocer el tipo del arreglo y la cantidad de elementos que contedrá. Esta información la obtiene de la proposición DIMENSION (ver "Proposiciones de especificación"). En ella se especifica el último elemento de cada arreglo, dado que la ubicación de él corresponde a los límites máximos de cada dimensión. El tipo del arreglo, o lo que es lo mismo de sus elementos, se obtiene en igual forma que el tipo de las variables.

EL PRODUCTO DE LOS LIMITES permite obtener la cantidad de elementos del arreglo y junto con el TIPO de éste, la cantidad de memoria que es necesario reservar.

```
C  EJEMPLØ 11.
C  USØ  DE ARREGLØ
       DIMENSIØN X(5)
       READ (1,10) A,B
       X(1)  = A + B
       X(2)  = A — B
       X(3)  = A * B
       X(4)  = A / B
       X(5)  = X(3) * X (2)
C  SE IMPRIMEN TØDØS LØS
C  ELEMENTØS DEL ARREGLØ X
       WRITE (3,20) X
       STØP
    10 FØRMAT (2F6.2)
    20 FØRMAT (5F10.3)
       END
```

6. *Expresiones*

A. *Expresión Aritmética*

Una *expresión aritmética* se define como: una constante, variable, referencia de función (ver "Subprogramas") o combinación de ellas entre sí con operadores aritméticos.

Se puede hacer uso de paréntesis en la misma forma que en álgebra.

a) *Operador aritmético.*

Los operadores aritméticos son símbolos que representan operaciones que deben efectuarse entre expresiones aritméticas.

Símbolo	*Operación*
**	exponenciación
*	multiplicación
/	división
+	adición
—	substracción

b) *Normas para escribir expresiones aritméticas*

Con el objeto de evitar interpretaciones erróneas de expresiones aritméticas, por ambigüedad en su escritura, es necesario cumplir las siguientes normas:

i) Toda operación entre expresiones aritméticas debe ser indicada mediante un operador.

Ejemplo: A por B debe escribirse A * B dado que AB es un nombre simbólico.

ii) No pueden aparecer dos operadores aritméticos contiguos.

Ejemplo: A por -B debe escribirse A * (-B) y no A * -B

iii) Para evaluar una expresión aritmética se procede de izquierda a derecha respetando la jerarquía siguiente:

1º Evaluación de funciones (ver "Subprogramas")
2º Exponenciación
3º Multiplicación y división
4º Adición y substracción

La excepción la constituyen operaciones de exponenciación en secuencia, en cuyo caso la evaluación se efectúa de derecha a izquierda.

Ejemplo 12:

1) A * B/C+D ** E

1º Cálculo de A * B queda X/C+D ** E
2º Cálculo de X/C queda Y+D ** E
3º Cálculo de D ** E queda Y+Z
4º Cálculo de Y+Z

2) A ** B ** C

1º Cálculo de B ** C queda A ** X
2º Cálculo de A ** X

iv) El tipo del resultado obtenido al evaluar una expresión aritmética está determinado por el tipo de las variables, constantes o resultados de funciones que conforman la expresión. La tabla que figura a continuación muestra todas las combinaciones posibles:

+ − * / **	ENTERO	REAL, SIMPLE PRECISION	REAL,DOBLE PRECISION
ENTERO	ENTERO	RSP	RDP
REAL,SIMPLE PRECISION	RSP	RSP	RDP
REAL,DOBLE PRECISION	RDP	RDP	RDP

B. *Expresión* LOGICA

Una expresión lógica se define como una constante lógica, una variable lógica, una referencia a función lógica, una expresión de relación o una combinación de ellas entre sí con operadores lógicos.

Expresión DE RELACION. Se obtiene al combinar dos expresiones aritméticas con un operador de relación.

a) *Operador de relación*

El operador de relación debe estar precedido y seguido por punto, separando las expresiones aritméticas. El resultado que se obtenga será siempre TRUE o FALSE.

Operador de relación	*Significado*
.GT.	mayor que (>)
.GE.	mayor o igual que (≥)
.EQ.	igual a (=)
.LE.	menor o igual que (≤)
.LT.	menor que (<)
.NE.	no igual a (≠)

Ejemplo 13:

Sea A = 6. K = 9 L = 3

Expresión de relación	*Valor*
A.GE.8.	falso
K.EQ.9	verdadero
L.LE.K	verdadero
A.NE.L	verdadero

b) *Operadores lógicos*

Existen tres operadores lógicos, cada uno de los cuales debe estar precedido y seguido por punto. Solamente aquellas expresiones que al ser evaluadas tienen el valor verdadero o falso pueden combinarse con los operadores lógicos.

Operador lógico	*Uso*	*Significado*
.NØT.	.NØT.A	Si A es verdad, entonces la expresión tiene el valor falso; si A es falso, entonces la expresión tiene el valor verdad.
.AND.	A.AND.B	Si A y B son verdad, entonces la expresión tiene el valor verdad. Si alguno de los dos o ambos tienen el valor falso, la expresión tiene el valor falso.
.ØR.	A.ØR.B	Si alguno de los dos o ambos tienen el valor verdad, la expresión tiene el valor verdad; si ambos tienen el valor falso, la expresión tiene el valor falso.

Las únicas secuencias válidas de operadores lógicos son: .AND. .NØT. y .ØR. .NØT.

Ejemplo 14:

Sea I = 8 X = 55.4 e Y = 100.

Expresión lógica	*Valor*
.NØT.(X.NE.Y)	falso
X.NE.Y.AND.X.LT.I	falso
Y.GT.1.ØR.X.GE.55.4	verdad

Evaluación de una expresión lógica: para evaluar una expresión lógica se debe respetar la jerarquía siguiente:

1º Evaluación de funciones
2º Exponenciación (**)
3º Multiplicación y división (* y /)
4º Adición y substracción (+ y −)
5º Relaciones (.GT.,.GE.,.EQ.,.LE.,.LT.,.NE.)
6º .NØT.
7º .AND.
8º .ØR.

Ejemplo 15:

1) Y.GT.A + D ** I.AND..NØT.(X.NE.Y).ØR.N

Pasos:

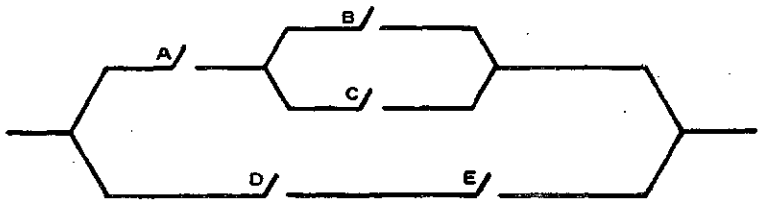
1º Evaluación de D ** I; queda Y.GT.A + Z.AND..NØT.(X.NE.Y) .ØR.N
2º Evaluación de A+Z; queda Y.GT.W.AND..NØT.(X.NE.Y).ØR.N
3º Evaluación de X.NE.Y; queda Y.GT.W.AND..NØT.V.ØR.N
4º Evaluación de Y.GT.W; queda U.AND..NØT.V.ØR.N
5º Evaluación de .NØT.V; queda U.AND.T.ØR.N
6º Evaluación de U.AND.T; queda S.ØR.N
7º Evaluación de S.ØR.N

2) (A.AND.(B.ØR.C)).ØR.(D.AND.E)

Pasos:

1º Evaluación de B.ØR.C.; queda (A.AND.Z).ØR.(D.AND.E)
2º Evaluación de A.AND.Z; queda Y.ØR.(D.AND.E)
3º Evaluación de D.AND.E; queda Y.ØR.X
4º Evaluación de Y.ØR.X

La expresión de este ejemplo es equivalente al circuito que se indica a continuación:



III. PROPOSICIONES

Se definen como proposiciones aquellas expresiones cuya traducción, hecha por el compilador, equivalen, en la mayoría de los casos, a varias instrucciones en lenguaje de máquina o a reservas de espacio de memoria, creación de tablas de símbolos, etc.

1. *Proposición de asignación: aritmética y lógica*

La proposición de asignación, como su nombre lo indica, permite asignar a una variable el resultado de una expresión aritmética o de una expresión lógica. Para ello se utiliza el operador de definición, que es un signo igual (=) y que se traduce como "se define por".

i) Estructura de la proposición

a = b

donde:

a: es cualquier variable con o sin subíndices

b: es cualquier expresión aritmética o lógica

ii) Función. La variable que figura al lado izquierdo del símbolo de definición se define por el valor que resulta al evaluar la expresión que está al lado derecho del símbolo. Si la variable que se está definiendo tenía algún valor, éste queda borrado por el nuevo. El tipo de la variable definida tiene prioridad sobre el tipo de resultado obtenido para la expresión, cuando ésta sea aritmética.

Si *b* es una expresión aritmética, *a* debe ser una variable real o entera. Si *b* es una expresión lógica, *a* debe ser una variable lógica.

Ejemplo 16:

Suponer que el tipo de las siguientes variables ha sido especificado como se indica a continuación:

Variable	*Tipo*
A,B,C,D	Reales precisión simple
E,F	Reales precisión doble
G,H,I,J	Enteras
L,M	Lógicas

De acuerdo con esas especificaciones se ilustra el funcionamiento de la proposición de asignación con los ejemplos que siguen:

A = B	El valor de B define a la variable A
C = E*D	El valor de la expresión es de doble precisión La parte más significativa de ese valor define a C
F = G	El valor de la variable G se convierte a real de doble precisión y define a la variable F
H = D	La parte entera de la variable D se asigna a la variable H

I = I+1	El valor de I es reemplazado por el valor de I más 1
L = .FALSE.	El valor de L es reemplazado por FALSE
M = 3..NE.C	Si la constante real 3. no es igual al valor de la variable C, se asigna a M el valor TRUE; en caso contrario se le asigna FALSE.

A. *Problemas propuestos*

a) Escribir las proposiciones de asignación aritmética que corresponden a las siguientes fórmulas originales:

i) $\pi = 3{,}14159$

ii) $p = \gamma \cdot Z + pa$

iii) $F = \frac{\gamma}{2}(bH^2 - bh^2)$

iv) $e = \frac{500H}{\mu(\frac{\delta}{2}+500)}$

v) $Q = 0{,}785D^2 \cdot 0{,}82 \cdot (2gH)^{\frac{1}{2}}$

vi) $R = \frac{4\alpha^2 \rho L}{\pi}$

vii) $t_3 = t_4 + \frac{\frac{x_3}{k_3}(t_1 - t_4)}{\frac{x_1}{k_1}+\frac{x_2}{k_2}+\frac{x_3}{k_3}}$

viii) $M = \frac{Ebh^3T}{6{,}6DN}$

ix) $P_5 = a_0x^5 + a_1x^4 + a_2x^3 + a_3x^2 + a_4x + a_5$

x) $c = \frac{U^2+V^2}{U^2-V^2}$

b) Suponiendo que se tienen definidas las variables A,B,C,D,I y J con los siguientes valores:

A=1. B=1.5 C=5. D=3. I=2 J=3

indicar cuál es el valor que se obtiene en las proposiciones aritméticas que figuran a continuación:

```
i)   M    =   I/J
ii)  N    =   J/I
iii) BO   =   -1./(2.*C)+D**I/(4.*A**J)
iv)  Y    =   (A*1.E-6+B*D**J)**(I/J)
v)   Y    =   A+B/C-D**J
vi)  CE   =   1.112*D*B*C/(D-C)
vii) ALFA =   C**I/B*(D**I)
viii)BETA =   (-C-D)**I-(C-D)**(I+J)
ix)  PX   =   A*B**J+C*B**I+D*B+A*C*D
x)   L    =   (B+2*B)**((J+I)/I)
```

2. *Proposiciones de Control*

Se entiende por proposiciones de control aquellas que permiten alterar la secuencia normal de ejecución de un programa.

A. *Proposición* GO TO

Esta proposición permite transferir el control de la ejecución a otra proposición ubicada antes o después de ella. Esto se conoce como BIFURCACION y más comúnmente como SALTO dentro del programa; SALTO hacia ADELANTE o SALTO hacia ATRAS.

Existen tres clases de GO TO:

a) GO TO *Incondicional*

Es el que permite realizar el salto sin que haya una condición previa para su ejecución, esto es, el salto se efectúa siempre.

i) Estructura dc la proposición

```
GO TO X
```

donde:

X: es el número de identificación de una proposición ejecutable.

ii) Función. Produce una bifurcación en el programa, de tal manera que la próxima proposición a ejecutar será aquella que tiene el número X.

Ejemplo 17:

Calcular la suma de los enteros mayores que cero

```
C  EJEMPLØ 17.
C  GRUPØ DE PRØPØSICIØNES QUE
C  PERMITE CALCULAR LA SUMA DE
C  LØS  ENTERØS MAYØRES QUE CERØ
       NS = 0
       I  = 1
```

```
10  NS = NS + I
    I  = I + 1
    GØ TØ 10
```

Se puede observar que este grupo de proposiciones constituye un programa que, teóricamente, se ejecutará en forma indefinida dado que no hay un elemento que permita romper el ciclo obligado por la proposición GO TO

```
C  EJEMPLØ 18.
C  PRØBLEMA QUE SE PUEDE
C  PRESENTAR EN EL USØ DE GØ TØ
       NS   = 0
       I    = 1
       GØ TØ 5
       I    = I + 1
    5  NS   = NS + I
       STØP
       END
```

El problema que muestra el grupo de proposiciones del ejemplo 18 es el que se deriva de tener una proposición (puede ser un grupo) sin identificación inmediatamente después del GO TO, lo que significa que ella nunca será ejecutada.

b) GO TO *computado*

Es el que permite realizar el salto de acuerdo con el valor que tenga en ese momento una variable determinada, esto es, hay un cierto control sobre la ejecución de la proposición. Sin embargo, pueden presentarse los mismos problemas que se vieron en el GO TO incondicional: ciclo indefinido o grupo de proposiciones que no se ejecuta.

i) Estructura de la proposición

GO TO $(x_1, x_2, \ldots, x_n)$,i

donde:

los x_i : son números de identificación de proposiciones ejecutables, *i* es una variable entera, sin subíndices que debe cumplir con $1 \leqslant i \leqslant n$

ii) Función. El salto o bifurcación se realiza a la proposición cuyo número de identificación está en el lugar indicado por la variable *i*. Si el valor de *i* está fuera del rango permitido, se ejecuta la proposición que figura a continuación del GO TO (algunos compiladores indican error).

Ejemplo 19:

```
C  EJEMPLØ 19.
C  USØ DE GØ TØ EN
C  SUMA REBUSCADA DE ENTERØS
       M = 0
       NS = 0
       I = 1
```

```
    5   M = M + NS/100
        GØ TØ (20,10),M
   20   NS = NS + I
        I = I + 1
        GØ TØ 5
   10   WRITE (3,51)NS
        STØP
   51   FØRMAT (I6)
        END
```

En el ejemplo 19 se hace uso de la característica del GO TO Computado, de tomar la proposición siguiente cuando la variable está fuera de rango (eso ocurrirá cuando M tenga valor 0). Cuando M tenga valor 1, el salto se realiza a la proposición 20 y cuando M sea 2 el salto se efectúa a la proposición 10 en que se imprime el valor obtenido por NS. ¿Cuántos enteros se suman? Conviene tener cuidado al hacer uso de la característica mencionada, dado que algunos compiladores, como se indicó antes, acusan error y otros producen resultados imprevisibles sin dar ninguna indicación.

c) GO TO *asignado*

Similar a la proposición anterior, en ésta se *asigna* un valor a la variable que indicará el lugar o meta del salto mediante una proposición ASSIGN.

i) Estructura de la proposición

ASSIGN i TO m

GO TO m,(x_1,x_2,...,x_n)

donde:

los x_i : son números de identificación de proposiciones ejecutables, *i* es un número de identificación de proposición y debe corresponder a uno de los *x*, *m* es una variable entera, sin subíndices, a la cual se le asigna el valor de *i*.

ii) Función. El salto o bifurcación se realiza a la proposición cuyo número de identificación se le haya asignado a la variable *m*. Este número debe estar entre los x_i.

Ejemplo 20:

```
C  EJEMPLØ 20.
C  USØ DE ASSIGN Y
C  GØ   TØ ASIGNADØ
        ASSIGN 20 TØ L
        READ (1,51)B,C
     5  GØ TØ L,(20,30,10)
    20  A = B + C
        ASSIGN 10 TØ L
        GØ TØ 5
```

```
30   A = B * C - 5.4
     ASSIGN 10 TØ L
     GØ TØ 5
10   Z = A ** 2
     WRITE (3,52) Z
     STØP
51   FØRMAT (F6.2,F6.2)
52   FØRMAT (F10.3)
     END
```

En este caso se trata de realizar uno de dos procesos de cálculo de acuerdo con el valor asignado a la variable L con la primera proposición ASSIGN. Un proceso corresponde a las proposiciones 20 y 10, que son las que se ejecutan en el ejemplo y el otro a las proposiciones 30 y 10. La primera proposición ASSIGN puede ser colocada en el momento de ejecución del programa con el valor 20 o con el valor 30, según sea el proceso que se desea efectuar.

B. *Proposición* IF

Esta proposición permite efectuar un salto, de acuerdo con un resultado aritmético o lógico obtenido. Existen, por lo tanto, dos tipos de IF:

a) IF *aritmético*

El salto se realiza de acuerdo con el resultado obtenido al evaluar una expresión aritmética.

i) Estructura de la proposición

IF (a) x_1,x_2,x_3

donde:

x_1,x_2,x_3: son números de identificación de proposiciones ejecutables y

a: es una expresión aritmética.

ii) Función. El salto o bifurcación se realiza a la proposición x_1 cuando el valor de *a* es menor que cero, a x_2 si es igual a cero y a x_3 si es mayor que cero. Se pueden realizar las siguientes combinaciones:

$x_1=x_2$ salto si el valor de *a* es menor o igual a 0
$x_2=x_3$ salto si el valor de *a* es mayor o igual a 0
$x_1=x_3$ salto si el valor de *a* es distinto de 0

Ejemplo 21:

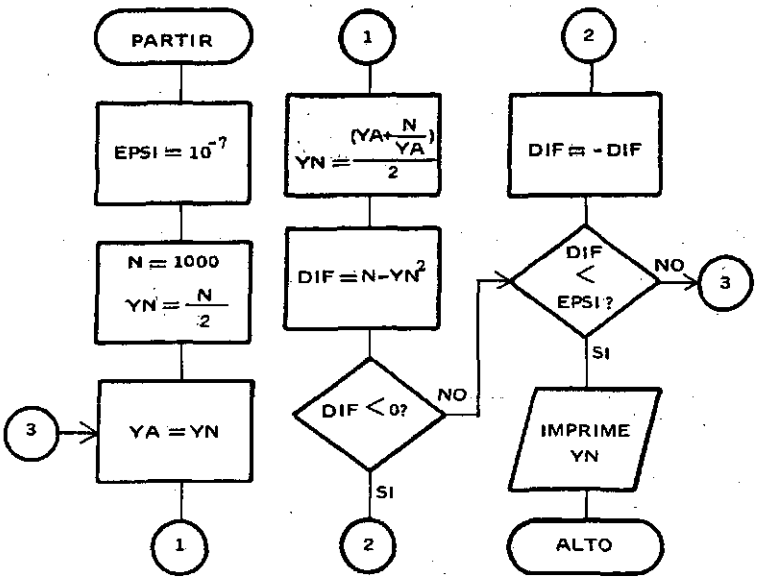
Programar el cálculo de $\sqrt{1000}$ utilizando la fórmula de aproximación de Newton:

$$y_n = \frac{1}{2}\left(y_a + \frac{N}{y_a}\right)$$

en que:

N : es el número cuya raíz se desea
y_a : es la raíz última obtenida (o el valor de partida)
y_n : es la nueva raíz

El error de aproximación debe ser menor que 10^{-7}




```
C  EJEMPLØ 21.
C  USØ DE IF ARITMETICØ.
C  SE CALCULA LA RAIZ CUADRADA DE N
C  CØN  LA FØRMULA DE NEWTØN
         EPSI  = 1.E-7
         ZN = 1000.
         YN = ZN/2.
       1 YA = YN
         YN = (YA + N/YA)/2.
         DIF = ZN - YN * YN
         IF (DIF) 2,3,3
       2 DIF = -DIF
       3 IF (DIF - EPSI) 4,1,1
       4 WRITE (3,51) YN
         STØP
      51 FØRMAT (F10.3)
         END
```

Observaciones:

i) Si se cambia la proposición ZN = 1000, por una proposición que lea el número, el programa queda más general.

ii) En la primera proposición IF, si el valor de DIF es igual a 0, se puede saltar a imprimir inmediatamente.

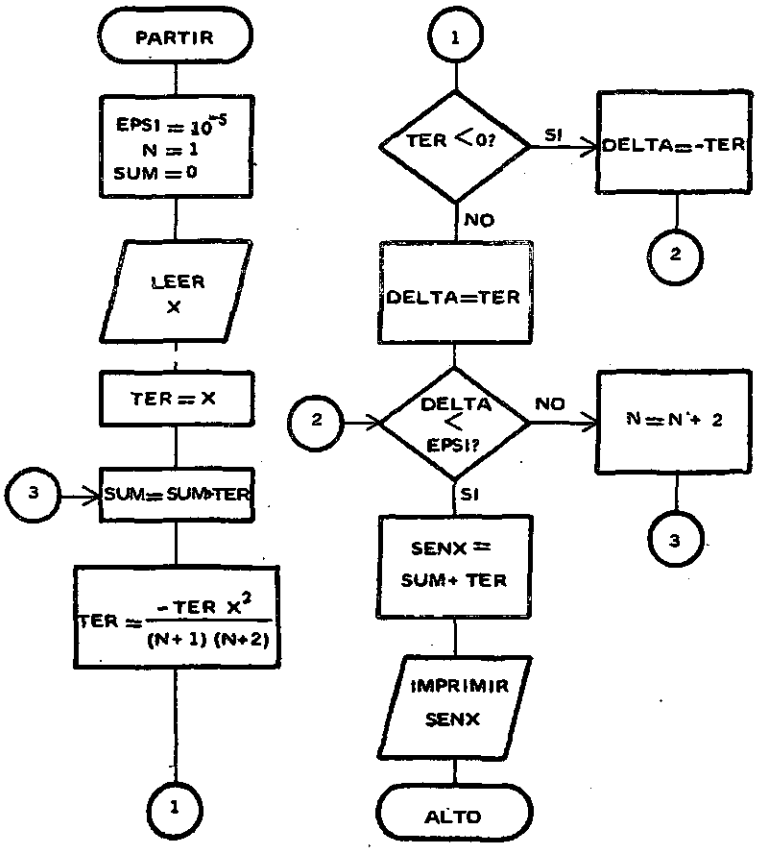
iii) La segunda proposición IF contiene en forma indirecta la comparación entre DIF y EPSI.

Ejemplo 22:

Programar el cálculo de:

$$\text{senx} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \pm \ldots$$

Detener el cálculo cuando el último término sea, en valor absoluto, menor que 10^{-5}.



```
C  EJEMPLØ 22.
C  USØ DE IF ARITMETICØ.
C  SE CALCULA SEN(X) UTILIZANDØ UNA
C  SERIE
       EPSI = 1.E-5
       ZN = 1.
       SUM = 0.
       READ (1,51) X
       TER = X
    6  SUM = SUM + TER
       TER = -TER*X*X/(ZN+1)/(ZN+2)
       IF (TER) 1,2,2
    1  DELTA = -TER
       GØ TØ 3
    2  DELTA = TER
    3  IF (DELTA - EPSI) 4,5,5
    5  ZN = ZN + 2.
       GØ TØ 6
    4  SENX = SUM + TER
       WRITE (3,52) SENX
       STØP
   51  FØRMAT (F6.2)
   52  FØRMAT (F10.3)
       END
```

b) IF *lógico*

El salto se realiza de acuerdo con el resultado obtenido al evaluar una expresión lógica.

i) Estructura de la proposición

IF(a)s

donde:

a: es una expresión lógica

s: es cualquiera proposición ejecutable, excepto una proposición DO u otra proposición IF lógica.

ii) Función. Se ejecuta la proposición s si el resultado de la expresión lógica es verdadero (TRUE); por el contrario, si es falso (FALSE) se ejecuta la proposición que sigue en secuencia al IF.

Ejemplo 23:

Hacer un programa que lea dos valores B y X. Calcular con dichos valores

$$A=B^2+\frac{x^2}{2}$$

si se cumple que:

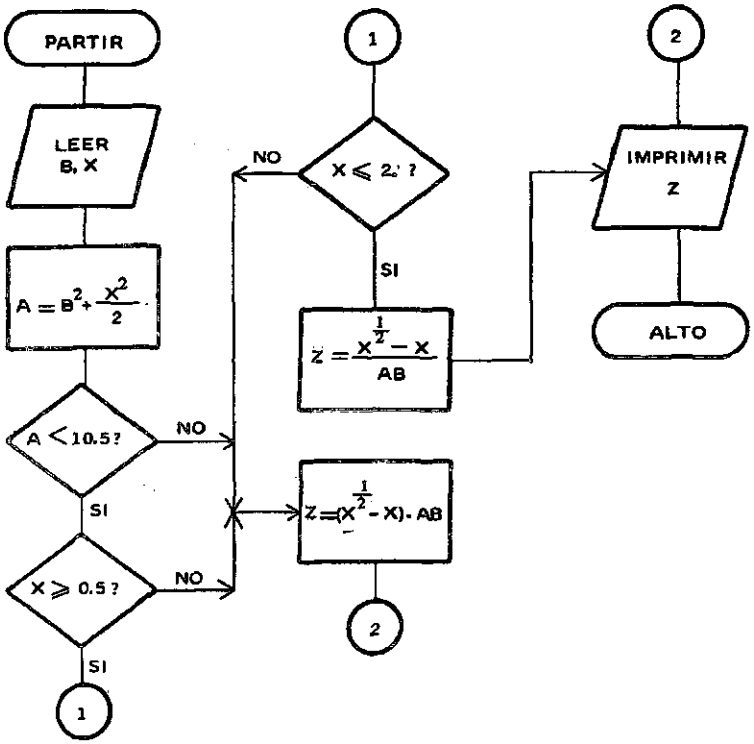
$$A<10.5 \text{ y } 0.5 \leqslant x \leqslant 2.$$

calcular

$$Z = \frac{x^{\frac{1}{2}} - x}{AB}$$

en caso contrario, calcular

$$Z = AB(x^{\frac{1}{2}} - x)$$




```
C  EJEMPLØ 23.
C  USØ  DE IF LØGICØ.
       READ (1,51) B,X
       A = B ** 2 + X * X / 2.
       IF(A.LT.10.5.AND.(X.GE..5.AND.X.LE.2.)) GØ TØ 1
       Z = A * B *(X ** .5 - X)
       GØ TØ 2
     1 Z = (X ** .5 - X)/(A * B)
     2 WRITE (3,52) Z
       STØP
    51 FØRMAT (F6.2,F6.2)
    52 FØRMAT (F10.3)
       END
```

Sugerencia: Resolver el mismo problema utilizando IF aritmético.

C. *Proposición* DO

Esta proposición permite ejecutar repetidamente, por un número de veces fijo, una o más proposiciones que conforman el "ciclo DO".

i) Estructura de la proposición

$$\text{DO x i} = m_1, m_2, m_3$$

donde:

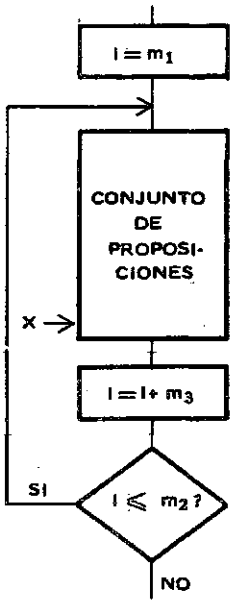
x: es el número de identificación de una proposición ejecutable que aparece después del DO en el programa.

i: es una variable entera sin subíndices llamada variable del DO.

m_1, m_2, m_3: pueden ser constantes enteras, sin signo, mayores que cero, o variables enteras, sin signo, sin subíndice, mayores que cero. El valor de m_2 no puede exceder de $2^{31}-1$. El valor m_3 es opcional, si no aparece, se subentiende que es 1, en este caso la coma que le precede debe eliminarse.

ii) Función. En el momento de ejecutarse la proposición DO se asigna a la variable *i* el *valor inicial* m_1, enseguida, se ejecuta el grupo de proposiciones hasta aquella que tiene el número de identificación *x*. Finalizada la ejecución se suma a la variable *i* el incremento m_3 y el resultado es comparado con el *valor final* m_2. Si es menor o igual que él, se ejecuta nuevamente el grupo de proposiciones; en cambio, si el valor de la variable i excede al valor de comparación, el proceso continúa en la proposición siguiente a la que tiene el número de identificación *x*.

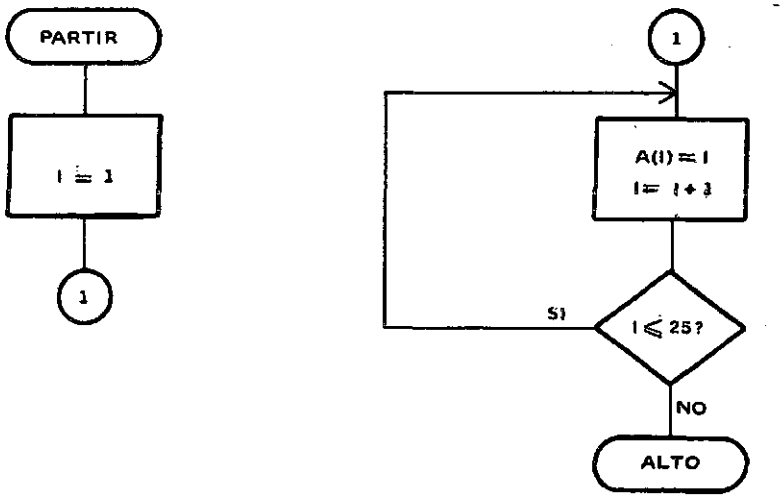
Aun cuando ocurra que m_2 sea menor que m_1, las proposiciones que forman el ciclo serán ejecutadas una vez, puesto que la comparación se realiza al término del ciclo. El diagrama de flujo que figura a continuación corresponde al funcionamiento de la proposición DO.



Se dice que las proposiciones que forman el ciclo están en el rango del ciclo DO.

Ejemplo 24:

Formar un arreglo A con 25 elementos de tal manera que cada elemento contenga el número real equivalente a su número de orden, esto es, A(I)=I



```
C  EJEMPLØ 24.
C  SØLUCIØN UTILIZANDØ IF
        DIMENSIØN A(25)
        I = 1
   1    A(I) = I
        I = I + 1
        IF (I.LE.25)GØ TØ 1
        STØP
        END
C  SØLUCIØN UTILIZANDØ DØ
        DIMENSIØN A(25)
        DØ 1 I = 1,25
   1    A(I) = I
        STØP
        END
```

Ejemplo 25:

En el mismo programa para el problema del ejemplo 24, calcular el producto de todos los elementos impares.

```
C  EJEMPLØ 25.
C  UTILIZACIØN DE DØ
        DIMENSIØN A(25)
        DØ 1 I = 1,25
   1    A(I) = I
        PA = 1.
        DØ 2 I = 1,25,2
   2    PA = PA * A(I)
        STØP
        END
```

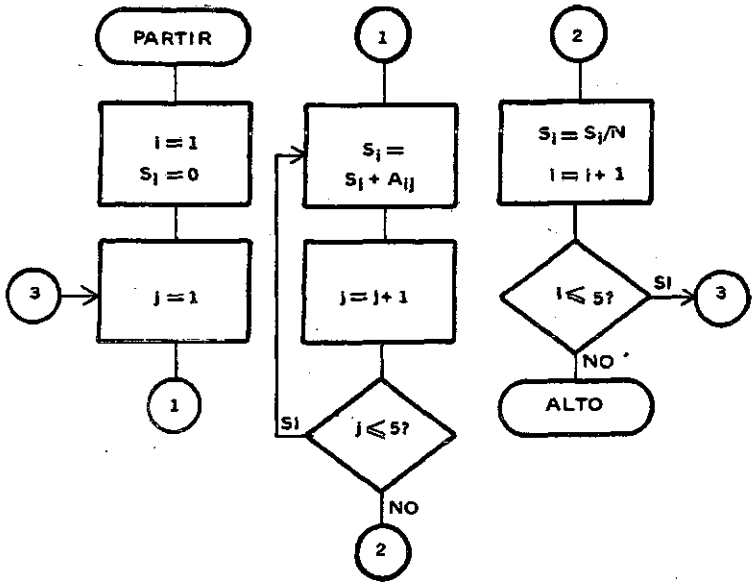
iii) *Normas que deben observarse al utilizar la proposición* DO

1) Los parámetros de la proposición DO, esto es, i,m_1, m_2 y m_3, no pueden ser cambiados por ninguna proposición que esté en el rango del ciclo DO.

2) Puede haber proposiciones DO en el rango del ciclo DO, formando así un nido de DO. Para que esta estructura sea válida, *todas las proposiciones del* DO *interior deben estar en el rango del* DO *que lo contiene.*

Ejemplo 26:

Se tiene un arreglo de cinco por cinco elementos. Se pide programar el cálculo del promedio de los elementos de cada renglón




```
C  EJEMPLØ 26.
C  APLICACIØN DE UN NIDØ DE DØ
       DIMENSIØN A(5,5),S(5)
       DØ 15 I = 1,5                          <- RANGO DEL DO EXTERNO
       S(I) = 0.
       DØ 10 J = 1,5                          <- RANGO DEL DO INTERNO
    10 S(I) = S(I) + A(I,J)
    15 S(I) = S(I) / 5.
       STØP
       END
```

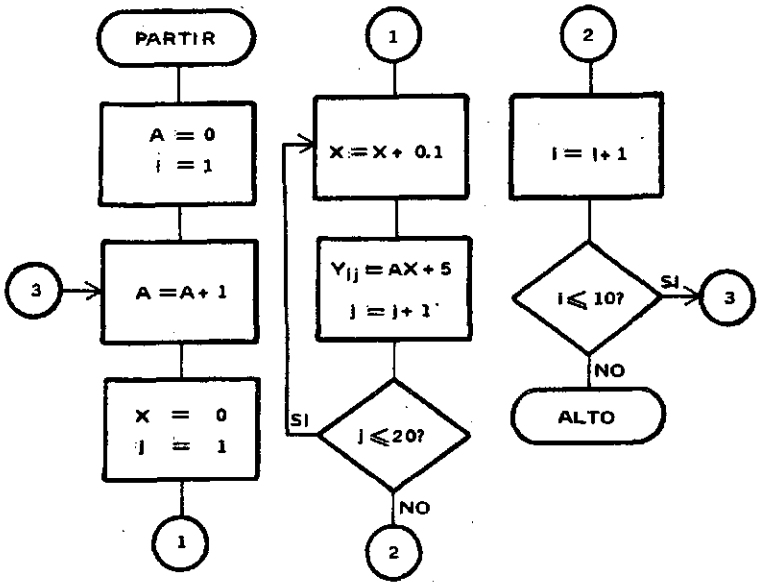
Ejemplo 27:

Programar el cálculo de: $Y = ax + 5.$ para

VALOR FINAL

VALOR INICIAL — VALOR DE INCREMENTO

a = 1. (10.) 1.
x = .1 (2.) .1




```
C  EJEMPLØ 27.
C  APLICACIØN DE UN NIDØ DE DØ
       DIMENSIØN Y(10,20)
       A = 0.
       DØ 10 I = 1,10
       A = A + 1.
       X = 0.
       DØ 10 J = 1,20
       X = X + .1
  10   Y(I,J) = A * X + 5.
       STØP
       END
```

RANGO DEL DO INTERNO

RANGO DEL DO EXTERNO

3) Es permitido el salto desde el interior de un ciclo DO.

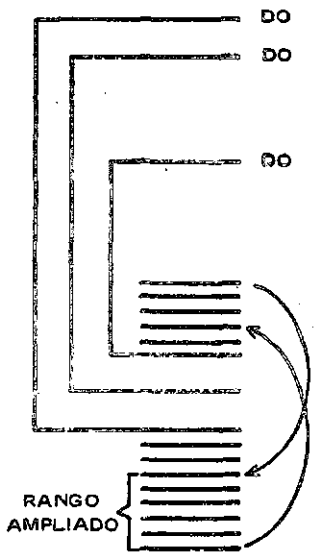
4) Se entiende como rango ampliado de un DO al conjunto de proposiciones comprendidas entre el punto meta de salto, para un salto efectuado desde el interior de un ciclo DO, y el punto donde se ordena el retorno a dicho ciclo, ambos puntos incluidos. Deben cumplirse las siguientes reglas:

El salto al interior de un ciclo DO se puede efectuar solamente si es desde un rango ampliado del DO.

El rango ampliado de un DO no debe contener otra proposición DO que también tenga rango ampliado, si este último está en el mismo módulo de programa que el primer DO.

Los parámetros de la proposición DO, esto es, i, m_1, m_2, m_3, no pueden ser cambiados en el rango ampliado del DO.

Ejemplo 28:



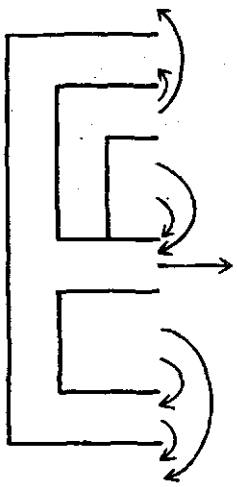
5) El número de identificación de una proposición que figura como la última en el rango de más de un DO (ejemplo 27) no puede ser utilizado como meta de salto en ningún GO TO o IF aritmético que no estén en el rango del DO más interno.

6) La última proposición en el rango de un ciclo DO no puede ser una proposición GO TO, PAUSE, STOP, RETURN, IF aritmético, DO o un IF lógico que contenga a alguna de ellas.

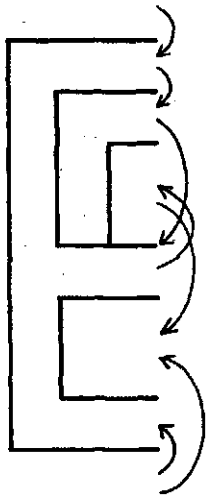
7) Se puede saltar a un subprograma desde el interior de un ciclo DO, como asimismo, retornar del subprograma al ciclo.

A continuación se indican gráficamente ejemplos de saltos permitidos y no permitidos.

Saltos permitidos



Saltos no permitidos



D. *Proposición* CONTINUE

Esta proposición puede ser colocada en cualquier parte de un programa, por supuesto, en un lugar que corresponda a una proposición ejecutable, sin que la secuencia de ejecución se vea afectada.

i) Estructura de la proposición

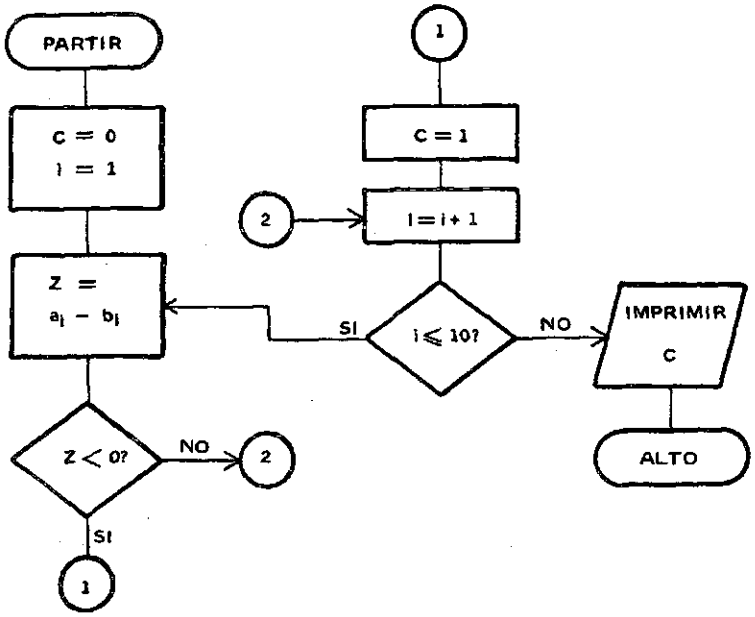
CONTINUE

ii) Función. Es una proposición "vacía" o de no-operación. Su utilidad se manifiesta al ser colocada como última proposición en un ciclo DO en aquellos casos en que éste termina con proposición GO TO, PAUSE, STOP, RETURN, IF aritmético, DO o IF lógico que contenga a alguna de ellas.

Ejemplo 29:

Se tienen dos listas, A y B, de diez elementos cada una. Se pide averiguar si alguna diferencia $A_i - B_i$ es menor que cero. Si así ocurre, hacer C=1, en caso contrario hacer C= 0.

En la solución de este tipo de problemas es necesario tener en cuenta que el proceso del ciclo DO siempre debe pasar por la última proposición de dicho ciclo, dado que inmediatamente, a continuación de la ejecución de ella, se produce el incremento de la variable del DO y su comparación con el valor final.




```
C  EJEMPLØ 29.
C  USØ  DE LA PRØPØSICIØN CØNTINUE
      DIMENSIØN A(10),B(10)
      C = 0.
      DØ 1 I = 1,10
      IF (A(I) - B(I)) 2,1,1
    2 C = 1.
    1 CØNTINUE
      WRITE (3,51) C
      STØP
   51 FØRMAT (F10.3)
      END
```

Observación: En el programa se ha eliminado el paso $Z=a_i-b_i$ porque se puede colocar directamente en la proposición IF.

Ejemplo 30:

Considerando las dos listas del problema anterior, para cada diferencia negativa de a_i-b_i, incrementar el primero en 2.5 y decrementar el segundo en 2.5, luego acumular la sumatoria del producto de ambos y volver a verificar la diferencia de los mismos elementos, esto es, para el mismo *i*.

```
C  EJEMPLØ 30.
C  USØ  DE LA PRØPØSICIØN CØNTINUE
      DIMENSIØN A(10),B(10)
      S = 0.
      DØ 1 I = 1,10
    2 IF (A(I) - B(I)) 3,1,1
    3 A(I) = A(I) + 2.5
      B(I) = B(I) - 2.5
      S = S + A(I) * B(I)
      GØ TØ 2
    1 CØNTINUE
      WRITE (3,51) S
      STØP
   51 FØRMAT (F10.3)
      END
```

E. *Proposición* PAUSE

Se utiliza para producir una detención momentánea del proceso con el objeto de permitir alguna intervención del operador.

i) Estructura de la proposición. Tiene tres formatos:

PAUSE
PAUSE n
PAUSE 'mensaje'

donde:

n: es una constante entera sin signo de hasta cinco dígitos
'mensaje': es una constante literal

ii) Función. Junto con producirse una detención momentánea del proceso, se imprime en la máquina de escribir de la consola PAUSE 0, PAUSE y el valor de *n* o PAUSE y el mensaje, de acuerdo con el formato que ha sido utilizado. Una vez que se termina la acción planificada durante la pausa (análisis de resultados intermedios, cambio de formulario, montaje de discos o cintas magnéticas, etc.), se puede ordenar que continúe el proceso. En este caso la reiniciación se efectúa en la proposición siguiente a PAUSE.

F. *Proposición* STOP

Se utiliza para producir la detención definitiva del proceso.

i) Estructura de la proposición. Tiene dos formatos:

STOP
STOP n

donde:

n: es una constante entera sin signo de hasta cinco dígitos.

ii) Función. Se produce la detención definitiva del proceso y si se ha utilizado el segundo formato se imprime además STOP y el valor de *n* en la máquina de escribir de la consola. El segundo formato se emplea cuando se desea detectar el recorrido que ha tenido el proceso hasta llegar a obtener un resultado determinado. En este caso, se distribuyen proposiciones STOP con distintos valores de *n* en los puntos de término del programa que se desea controlar, de tal manera que al detenerse el proceso e imprimirse STOP y el valor de *n*, se ubica inmediatamente el punto de detención.

G. *Proposición* END

Se utiliza con el objeto de indicar el término físico de un programa fuente, sea éste programa principal o subprograma.

i) Estructura de la proposición

END

ii) Función. Indica al compilador el término de las proposiciones, por tanto, no es una proposición que produzca detención del proceso para lo cual está destinada STOP. El campo destinado a número de identificación debe aparecer en blanco.

H. *Problemas propuestos*

a) Programar

$$Y = \frac{(A+B)^2}{(A-B)} \qquad \text{si } A-B>0$$

$$Y = \frac{A+2(A+B)}{A-B} \qquad \text{si } A-B<0$$

$$Y = 5(A+B)-B \qquad \text{si } A-B=0$$

b) Programar

$$F = \frac{7(A+B)+C}{C+A+D} \qquad \text{si } C+B<A$$

$$F = \frac{C-A}{(D-B)A+C} \qquad \text{si } C+B \geq A$$

c) Programar

$$F = A(A+B-|C+D|)\cdot 1/2$$

d) Programar

$$Z = |X| - 2|X - |Y||$$

e) Se pide el valor de L al terminar el siguiente programa:

```
      L = 3
      L = (L / 4) * 4 + L
      IF (L – 15) 1,6,6
1     L = L + 2
      IF (L – (7 * L) / 4) 6,4,4
4     L = L – 1
6     STØP
      END
```

f) Programar

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} \pm \cdots$$

detener el cálculo cuando el último término sea, en valor absoluto, menor que 10^{-5}.

g) Programar

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

igual criterio de detención que en el problema anterior.

h) Programar el cálculo de π a base de:

i) $\dfrac{\pi}{4} = 1 - \dfrac{1}{3} + \dfrac{1}{5} - \dfrac{1}{7} \pm \cdots$

ii) $\dfrac{\pi^2}{6} = 1 + \dfrac{1}{2^2} + \dfrac{1}{3^2} + \dfrac{1}{4^2} + \cdots$

iii) $\dfrac{\pi^2}{12} = 1 - \dfrac{1}{2^2} + \dfrac{1}{3^2} - \dfrac{1}{4^2} \pm \cdots$

iv) $\dfrac{\pi^2}{8} = 1 + \dfrac{1}{3^2} + \dfrac{1}{5^2} + \dfrac{1}{7^2} + \cdots$

Detener el proceso cuando el valor absoluto del último término calculado sea menor que 10^{-7}.

i) Programar

$$\log 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} \pm \cdots$$

igual criterio de detención que en el problema h).

j) Programar el cálculo de e a base de:

i) $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots$

ii) $\frac{1}{e} = 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} \pm \cdots$

detener el cálculo cuando el valor absoluto del último término sea menor que 10^{-8}.

k) Programar

$$(1+x)^n = 1 + nx + \frac{n(n-1)}{2!} x^2 + \frac{n(n-1)(n-2)}{3!} x^3 + \ldots$$

n debe ser entero > 0; $x^2 < 1$

l) Programar

$$(1+x)^{-n} = 1 - nx + \frac{n(n+1)}{2!} x^2 - \frac{n(n+1)(n+2)}{3!} x^3 \pm \cdots$$

n debe ser entero > 0; $x^2 < 1$

m) Programar

$$(1+x)^{\frac{1}{2}} = 1 + \frac{1}{2}x - \frac{1 \cdot 1}{2 \cdot 4} x^2 + \frac{1 \cdot 1 \cdot 3}{2 \cdot 4 \cdot 6} x^3 - \frac{1 \cdot 1 \cdot 3 \cdot 5}{2 \cdot 4 \cdot 6 \cdot 8} x^4 \pm \ldots$$

debe cumplirse que $x^2 \leqslant 1$, detener el proceso cuando el valor absoluto del último término sea menor o igual que 10^{-8}.

n) Programar

$$\log x = \frac{x-1}{x} + \frac{1}{2}\frac{(x-1)^2}{x} + \frac{1}{3}\frac{(x-1)^3}{x} + \ldots$$

igual criterio de detención que en el problema anterior.

o) $$\log\left(\frac{x-1}{x-1}\right) = 2\left[\frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \frac{1}{7x^7} + \ldots\right]$$

debe cumplirse que $x^2 > 1$. Igual criterio de detención que en el problema m).

3. *Proposiciones de entrada/salida (input/output)*

Permiten al programador transferir información desde dispositivos de entrada a memoria y de ésta hacia dispositivos de salida.

Los elementos que participan en el proceso de entrada/salida son:

a) Conjunto de datos (DATA SET)

b) Dispositivo de entrada/salida

c) Proposición de entrada o salida que contiene además la lista de variables (lista de entrada/salida)

d) Proposición de formato que indica cómo están agrupados los datos o cómo deben agruparse.

Para referirse al conjunto de datos, es decir, para indicar al computador *cuál es y dónde está el conjunto que se desea transferir*, se utiliza una constante entera sin signo o variable entera, sin signo, sin subíndices. Aun cuando de esta forma se identifica simbólicamente la unidad, se ha preferido llamar a esa constante o variable "número de referencia del conjunto de datos" dado que es a éste al que se desea indicar y no a un dispositivo en especial.

Es necesario tener en cuenta que estos números de referencia se asignan *en forma permanente a los dispositivos con que cuenta la instalación*; esto significa que las asignaciones hechas pueden variar de un computador a otro de acuerdo con la configuración que tenga cada uno.

En los ejemplos que se han visto, como asimismo en los que se desarrollarán más adelante, se utilizan las siguientes asignaciones:

Número de referencia	Tipo de dispositivo permitido	Tipo de operación permitida
1	Lectora de tarjetas	Entrada
3	Impresora	Salida

Hay dos tipos de proposiciones de entrada/salida: las proposiciones secuenciales y las de acceso directo (no disponibles las últimas en el Soporte Básico de Programación FORTRAN IV Básico). Las proposiciones secuenciales proporcionan las facilidades para la ubicación, selección y recuperación de datos organizados secuencialmente. Son independientes del dispositivo, pues un conjunto de datos secuenciales puede residir en cualquier tipo de volumen.

Las proposiciones de acceso directo proporcionan facilidades para la ubicación, selección y recuperación de datos en un orden especificado por el usuario. Son sólo válidas cuando el conjunto de datos va a residir o ya reside en dispositivos de memoria de acceso directo.

A. *Lista de entrada/salida*

Cuando se ejecuta una proposición de entrada, la información es llevada a memoria a ubicaciones que no son necesariamente contiguas. Si se trata de recuperar información desde memoria, casi siempre ella es reunida desde distintas ubicaciones y emitida al exterior mediante una proposición de salida. Esas posiciones de memoria donde quedará la información o desde donde se recuperará, se especifican con un conjunto de nombres simbólicos que constituyen la lista de entrada/salida.

Los nombres simbólicos que puede contener la lista son: nombres de variables, con o sin subíndice, y nombres de arreglos. En el primer caso se transferirá un solo valor por cada nombre, en cambio si se trata de nombres de arreglos se transferirán tantos datos como elementos tenga cada arreglo. La cantidad de elementos de un arreglo se determina a base de la especificación hecha en la proposición DIMENSION y el orden en que se transfieren es el que corresponde al almacenamiento en memoria.

La transferencia de arreglos en la forma vista tiene dos limitaciones: deben entrar o salir de memoria todos los elementos del arreglo y además deben hacerlo en el orden en que quedan en memoria. Si se desea eliminar esas restricciones, es necesario hacer uso de la estructura DO (DO implícito) para listas de entrada/salida. Esta estructura va separada por coma a continuación de una lista (o sublista) de entrada/salida, ambas encerradas entre paréntesis.

Ejemplo 31:

a) Lista de datos. Se desea imprimir: las variables x,y,z el arreglo lineal C con 10 elementos, el primer y el quinto elementos del arreglo lineal D.

X,Y,Z,C,D(1),D(5)

b) Lista de datos con DO implícito. El mismo caso anterior.

X,Y,Z,(C(I),I=1,10),D(1),D(5)

sublista

c) Variables simples con DO implícito. Se desea repetir la transferencia de X,Y,Z por cada elemento de C.

(X,Y,Z,C(I),I=1,10)

sublista

d) Se desea transferir cada elemento impar del arreglo junto con el valor del índice correspondiente

(I,C(I),I=1,10,2)

sublista

e) Se desea transferir un número N de elementos siendo N menor o igual que el número máximo de elementos.

N,(C(I),I=1,N)

f) Transferencia de un arreglo bidimensional:

i) ((E(I,J),I=1,5),J=1,6)

varía el índice I más rápido que el índice J, esto es, corresponde al orden que tienen los elementos en memoria.

ii) ((E(I,J),J=1,6),I=1,5)

varía el índice J más rápido que el índice I

g) Transferencia de un arreglo tridimensional

(((F(I,J,K),I=1,3),J=1,2),K=1,4)

varía rápido I, más lento J y más lento aún K.

B. *Proposición* READ

Esta proposición permite introducir datos a la memoria principal.

i) Estructura de la proposición

READ (a,b,ERR=c,END=d) lista

donde:

a: es un número de referencia de conjunto de datos (data set), dado en forma de constante entera sin signo o variable entera sin signo, sin subíndices.

b: es opcional y si se utiliza puede ser el número de identificación de la proposición FORMAT, que describe el registro que se va a leer, el nombre de un arreglo que contiene las especificaciones de formato o un nombre definido en una proposición NAMELIST. Si *b* no se utiliza significa que los datos serán leídos sin formato.

ERR=c: es opcional y si se utiliza, c es el número de identificación de una proposición contenida en el mismo programa que contiene a READ. Esa proposición será la meta de salto en caso de detectar error durante la transferencia.

END=d: es opcional y si se utiliza, *d* es el número de identificación de una proposición contenida en el mismo programa que contiene a READ. Esa proposición será la meta de salto en caso de detectar el término del conjunto de datos.

lista: es opcional y corresponde a una lista de entrada.

ii) Función. Se transfieren datos a la memoria principal desde el conjunto de datos identificados por *a*. Si se lee sin formato, los datos leídos deben haber sido *grabados* anteriormente con una proposición WRITE sin formato. En caso de detectar error o término del conjunto

de datos (fin de archivo), la próxima proposición a ejecutar está señalada por c o *d* respectivamente, siempre que se utilicen estos parámetros. Si la lista no es colocada, se efectúa el salto de un registro. Los parámetros ERR=c y END=d se pueden colocar en cualquier orden cuando se usan ambos.

Ejemplo 32.

1) READ(1,51,ERR=20,END=50)A,B,C,D
2) READ(N,57,END=100)I,J,(A(I),I=1,50)
3) READ(1,60)((B(I,J),I=1,10),J=1,15)
4) READ(10)N,(C(I),I=1,N)
5) READ(11,65)(D(I),E(I),I=1,50),(F(J),J=1,30)

C. *Proposición* WRITE

Esta proposición permite sacar datos desde la memoria principal.

i) Estructura de la proposición

WRITE (a,b) lista

donde:

a: es un número de referencia de conjunto de datos (ver proposición READ)

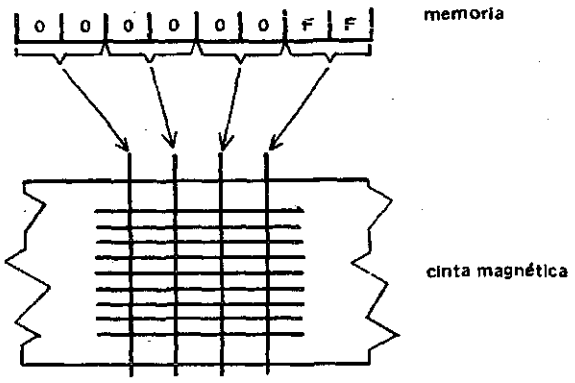
b: es opcional y si se utiliza puede ser: el número de identificación de la proposición FORMAT que describe el registro que se va a imprimir (grabar), el nombre de un arreglo que contiene las especificaciones de formato o un nombre definido en una proposición NAMELIST. Si *b* no se utiliza, significa que los datos serán *grabados* sin formato.

lista: es opcional y corresponde a una lista de salida.

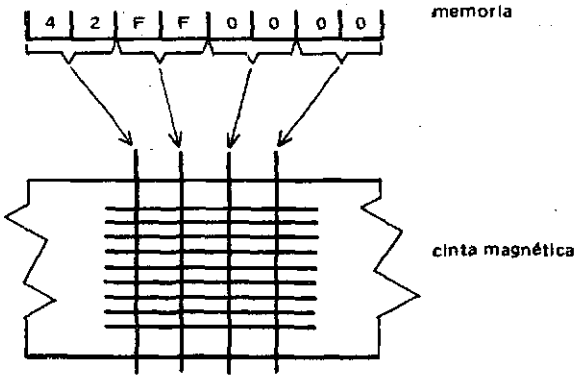
ii) Función. Se transfieren datos desde la memoria principal al conjunto de datos identificados por *a*. Si se graba sin formato, los datos grabados ocupan cuatro u ocho bytes según sea entero, real de precisión simple o real de precisión doble.

Ejemplo 33:

1) Grabación del dato entero 255 en cinta magnética sin usar formato:



2) Grabación del dato real 255 en cinta magnética *sin* usar formato:



Si se hubiera grabado con formato, cada carácter ocuparía un byte en la cinta magnética, de aquí que, números que tengan más de cuatro caracteres ocuparán más bytes en cinta magnética o en otro dispositivo magnético que si se graba sin formato.

Si no se pone lista de salida significa que se hará uso de la proposición FORMAT, en la cual se tiene como argumento a constantes literales (ver "Constante literal en una proposición FORMAT").

Ejemplo 34:

```
i)   WRITE (a,51)A,B,C,D
ii)  WRITE (3,52)A,B,C,(D(I),I=1,50)
iii) WRITE (11)A,B,C,(D(J),E(J),J,J=1,100)
iv)  WRITE (11,53)N,(A(K),K=1,N)
v)   WRITE (3,54)
```

Ejemplos de READ y WRITE haciendo uso de arreglos de códigos de formato y de la proposición NAMELIST, se pueden ver en el capítulo "Otras formas de READ y WRITE".

D. *Proposición* FORMAT

Aun cuando esta proposición no es de entrada/salida, sino que de especificación, es conveniente analizarla en este capítulo, pues ella está íntimamente ligada a las proposiciones de entrada/salida READ y WRITE.

i) Estructura de la proposición. Tiene tres formas básicas:

X FORMAT ($C_1, C_2, \ldots, C_n$)
X FORMAT ($C_{11}, C_{12}, \ldots, C_{1n}/C_{21}, C_{22}, \ldots, C_{2n}/ \ldots /C_{n1}, C_{n2}, \ldots, C_{nn}$)
X FORMAT ($C_{11}, C_{12}, \ldots, C_{1n}, (C_{21}, C_{22}, \ldots, C_{2n}), \ldots, (C_{n1}, C_{n2}, \ldots, C_{nn}) \ldots$)

donde:

X: es el número de identificación de la proposición FORMAT
C_i y C_{ij}: son códigos de formatos, esto es, son aquellos elementos mediante los cuales se describen campos de datos o especifican características que permiten la transferencia de información.

ii) Función. La proposición FORMAT permite describir los datos que van a ser transferidos con proposiciones READ y/o WRITE. De acuerdo a la estructura de FORMAT utilizada queda definido el (los) registro (s) FORTRAN, esto es, se especifica la cantidad de datos que tendrá cada registro físico.

1) Primera forma básica:

X FORMAT ($C_1, C_2, \ldots, C_n$)

| ← REGISTRO FORTRAN → |

Todos los datos descritos con los códigos de formato que están entre paréntesis (argumento del FORMAT) deben estar contenidos en un REGISTRO FORTRAN, por ejemplo, una tarjeta o una línea de impresión.

Si hay más códigos de formatos que datos, los códigos sobrantes se ignoran. Si hay menos códigos de formato que datos, se define un nuevo registro FORTRAN y la descripción de los campos se inicia otra vez con el código del extremo izquierdo (paréntesis izquierdo).

2) Segunda forma básica:

X FORMAT($C_{11}, \ldots, C_{1n}$ / $C_{21}, \ldots C_{2n}$ / ... / $C_{n1}, \ldots, C_{nn}$)

| ← REGISTRO FORTRAN → | REGISTRO FORTRAN | ← → | REGISTRO FORTRAN | ←

Todos los datos descritos con los códigos de formato que están entre el paréntesis izquierdo y el primer operador de división (slash), deben estar contenidos en un registro FORTRAN; todos los que están descritos con los códigos que figuran entre el primer operador de división y el segundo, en un nuevo registro FORTRAN y así sucesivamente.

Si hay más códigos o menos códigos, se aplican las normas indicadas en la "Primera forma básica".

Se pueden saltar registros de entrada o introducir registros de salida compuestos por blancos usando operadores de división consecutivos. Si hay *n* operadores consecutivos al comienzo o al final del argumento del FORMAT, se saltan o insertan *n* registros. Si aparecen *n* operadores consecutivos en cualquier otra parte del argumento, se saltan o insertan *n-1* registros.

Ejemplo 35:

X FORMAT (///...////)

se saltan o insertan tres registros al iniciarse la transferencia y se saltan o insertan cuatro registros cuando se detectan los cuatro operadores que figuran al término del argumento. Si la transferencia continúa, el control de formato se inicia nuevamente en el paréntesis izquierdo.

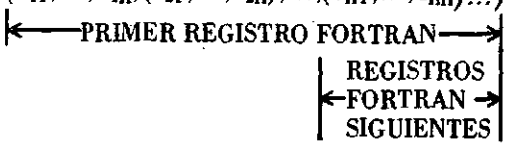
X FORMAT(.../////...)

los cuatro operadores que figuran entre los códigos de formato permiten saltar o insertar tres registros.

La utilidad principal que tienen los operadores de división consecutivos es poder dejar líneas en blanco en la salida, lo que permite darle más claridad a los resultados impresos.

3) Tercera forma básica:

X FORMAT ($C_{11},\ldots,C_{1n},(C_{21},\ldots,C_{2n}),\ldots,(C_{n1},\ldots,C_{nn})\ldots$)



Todos los datos descritos con los códigos de formato, que están entre el primer paréntesis izquierdo y el último paréntesis derecho (primer nivel), deben estar contenidos en el PRIMER REGISTRO FORTRAN. Si hay más códigos que datos, los códigos de formato sobrantes se ignoran. Si hay menos códigos, la descripción de los datos se obtiene con los códigos que están en el último grupo encerrado en paréntesis de segundo nivel, más los códigos que figuran hasta el último paréntesis derecho. Este conjunto de códigos define los REGISTROS FORTRAN SIGUIENTES.

Dentro de los paréntesis de segundo nivel se acepta un tercer nivel de paréntesis. El cuarto nivel no es aceptado.

Además de permitir la definición de registros FORTRAN, los paréntesis se usan también para encerrar un grupo de códigos de formatos que se repiten un número a de veces. El factor de repetición se especifica inmediatamente antes del paréntesis izquierdo que encierra el grupo.

Ejemplo 36:

X FORMAT(C_1,2(C_2,C_3),...,C_n) es idéntico a
X FORMAT(C_1,C_2,C_3,C_2,C_3,...,C_n)
X FORMAT(...,a(...,a(...),...,a(...),...),...) es válido
X FORMAT(...,a(...,a(...,a(...),...),...),...) no es válido

4) Carácter de control de carro.

El primer carácter de un registro de salida, cuando se trata de impresión, es considerado carácter de control del carro de la impresora, esto es, no se imprime y se interpreta de acuerdo con la siguiente tabla:

Carácter	*Significado* (acción del carro de la impresora)
blanco	avanza una línea antes de imprimir
0	avanza dos líneas antes de imprimir
1	avanza a la primera línea de la página siguiente
+	no avanza

Cualquier otro carácter se interpreta como blanco (b). La forma más común de especificar el carácter es colocándolo entre apóstrofos al comienzo del argumento del FORMAT.

Ejemplo 37:

X FORMAT('0',...)

se indica que el carro de la impresora debe avanzar dos líneas antes de que se realice la impresión.

E. *Códigos de formato*

Los códigos de formato son:

I para describir campos de datos enteros
F para describir campos de datos reales
E para describir campos de datos reales con exponente E
D para describir campos de datos reales con exponente D
L para describir campos de datos lógicos
G para describir campos de datos enteros, reales o lógicos
A para describir campos de caracteres
Z para describir campos de datos hexadecimales
H para indicar datos literales
'literal' para indicar datos literales
X para indicar que un campo va a ser saltado en entrada o llenado con blancos en salida

T para especificar la posición en un registro FORTRAN a partir de la cual se va a realizar la transferencia de datos.

En las estructuras de códigos de formato se utilizan los símbolos siguientes:

a: es una constante entera sin signo, opcional, usada para indicar el número de veces que se ocupará el mismo código. Si se omite, el código se ocupa una sola vez.

w: es una constante entera sin signo, distinta de cero, que especifica la cantidad de caracteres que tiene el campo, esto es, si hay signo, punto, etc., deben considerarse en w.

d: es una constante entera sin signo, que especifica el número de lugares decimales a la derecha del punto decimal, esto es, indica la cantidad de dígitos de la parte fraccionaria de un número.

p: es opcional y representa un factor de escala de la forma *nP*, en que *n* es una constante entera, con o sin signo. Para usarlo se aplica la fórmula siguiente:

$$\text{CANTIDAD EXTERNA} = \text{CANTIDAD INTERNA} * 10^{n}$$

s: es una constante entera sin signo, que especifica el número de dígitos significativos en salida o la parte fraccionaria en entrada.

r: es una constante entera sin signo, que especifica una posición dentro de un registro FORTRAN.

1. *Código de Formato I*

i) Estructura del código

a I w

donde:

a : es factor de repetición
w: es la cantidad de caracteres

ii) Función. Se utiliza para transferir datos enteros. Si es ENTRADA de datos, los blancos que encabezan la información, los que estén intercalados con los dígitos o en el extremo derecho son considerados como ceros. La magnitud no debe exceder la capacidad máxima definida para constante entera. Si es salida de datos, se presentan dos posibilidades, además de la normal, que es cuando *w* es igual a la cantidad de caracteres que tiene el dato. Una posibilidad es que *w* sea mayor que la cantidad de caracteres, en cuyo caso las posiciones excedentes de la izquierda se rellenan con blancos. La otra posibilidad es que *w* sea menor que la cantidad de caracteres, situación en la cual se imprimen *w* asteriscos.

Ejemplo 38:

Leer seis valores enteros perforados en una tarjeta. Los dos prime-

ros tienen tres y cinco caracteres respectivamente. Los cuatro restantes tienen cuatro caracteres cada uno. Imprimir los seis valores de tal manera que queden tres datos por línea. Los tres primeros con seis caracteres cada uno y los tres siguientes con siete caracteres cada uno.

```
C  EJEMPLØ 38.
C  USØ  DE CØDIGØ DE FØRMATØ !
       READ (1,51) I,J,K,L,M,N
       WRITE (3,52) I,J,K,L,M,N
       STØP
  51   FØRMAT (I3,I5,4I4)
  52   FØRMAT ('b',3I6/' ',3I7)
       END
```

El espacio en blanco (b) que aparece entre apóstrofos se utiliza como carácter de control de carro e indica que avance una línea antes de imprimir.

Suponiendo que los datos de entrada sean:

Columna 1 de la tarjeta

```
                    ↓
                    b10-32bbb4b523bbb-27b+28
```

Observación:

El carácter b se ha utilizado para representar el espacio en blanco tal como en el formato 52 del ejemplo 38. El carácter b significa que no se perfora nada en entrada y en salida no se imprime nada en esas posiciones.

El resultado de la impresión será:

posición 1 de la línea

```
                    ↓
                    bbbb10b–3200bbb405
                    bbb2300bbbb–27bbbbb28
```

Ejemplo 39:

Los mismos datos del ejemplo 38 están perforados ahora, los dos primeros con cinco caracteres cada uno en la primera tarjeta, los dos siguientes con tres y cuatro caracteres respectivamente en la segunda tarjeta y los dos últimos con cinco caracteres cada uno en la tercera tarjeta. Al imprimirlos se desea que el primer dato quede en la primera línea y dos datos en cada una de las líneas siguientes.

```
C  EJEMPLØ 39.
C  USØ  DE CØDIGØ DE FØRMATØ !
       READ (1,51) I,J,K,L,M,N
       WRITE(3,52) I,J,K,L,M,N
```

```
      STØP
51    FØRMAT (215/13,14)
52    FØRMAT ('Ø',13/'Ø',14,16/('Ø',12,15))
      END
```

La perforación de los datos será la siguiente:

columna 1

```
ØØØ10–32        1a tarjeta
4Ø523           2a tarjeta
ØØ–27ØØØ28      3a tarjeta
```

y la impresión será:

posición 1

```
Ø10             1a línea
****ØØØ405      2a línea
**ØØ–27         3a línea
28              4a línea
```

La impresión se realiza de acuerdo con la correspondencia siguiente:

Variable	*Código de formato*
I	I3
J	I4
K	I6
L	I2
M	I5
N	I2

Nótese que la variable J, cuyo valor es–3200, aparece impresa con ****, esto es, no se contemplaron los caracteres suficientes en *w* y se imprimieron *w* asteriscos. Lo mismo ocurre con la variable L, que tiene valor 2300 y *w* indica impresión de dos caracteres.

2. *Códigos de formato F, E y D*

i) Estructura del código

```
p a F w.d
p a E w.d
p a D w.d
```

donde:

- p: es factor de escala
- a: es factor de repetición
- w: es cantidad de caracteres
- d: es parte fraccionaria

ii) Función. Se utilizan para transferir datos reales. Estos no deben exceder la magnitud máxima que corresponde a constantes reales.

En ENTRADA, el dato puede tener, opcionalmente, exponente que debe estar precedido por una constante de al menos un dígito, con o sin signo, con o sin punto decimal. Si el dato tiene punto decimal, éste tiene prioridad sobre la indicación dada por *d* en el código de formato. Si el dato tiene exponente decimal E, D o constante entera con signo y en el código de formato se coloca factor de escala, éste no tiene efecto sobre el dato. Los espacios en blanco que figuren en el dato se consideran ceros.

Los códigos F, E y D se pueden usar para leer indistintamente datos con exponente E, D o constante entera con signo. En cualquier caso, tiene prioridad el tipo de la variable leída.

En SALIDA, en el código de formato F, en el valor *w*, están incluidos los dígitos de la parte entera, los de la parte fraccionaria, el punto decimal y el signo, si el valor es negativo. Los códigos E y D se pueden intercambiar para imprimir reales de precisión simple o doble, el resultado es impreso con la letra que corresponde al tipo de la variable que figura en la lista de salida. Deben contemplarse en *w*, a menos que se utilice factor de escala P, posiciones para: signo (si el dato es negativo), punto decimal, dígitos significativos y la letra D o E seguida de constante entera de dos dígitos y signo. Si hay espacio suficiente contemplado en *w*, aparece impreso el dígito cero antes del punto decimal. Como norma general, la constante 7 se debe sumar a la cantidad de dígitos significativos que se desea imprimir.

Si se utiliza factor de escala P con código de formato E o D, no tiene efecto sobre la magnitud del dato impreso sino en su estructura, dado que el punto decimal se desplaza a la izquierda o a la derecha y el exponente disminuye en el primer caso y aumenta en el segundo.

Ejemplo 40:

Se tienen los siguientes datos:

12345–78987654321

El programa que figura a continuación (REAL*8 C,D declara las variables C y D de doble precisión):

```
C EJEMPLØ 40.
C USØ DE CØDIGØS DE FØRMATØ E Y D
C CØN FACTØR DE ESCALA P
      REAL*8 C,D
      READ (1,51) A,B,C,D
      WRITE(3,52) A,B,C,D
      WRITE(3,53) A,B,C,D
      STØP
   51 FØRMAT(2E5.3,2D5.3)
```

```
 52   FØRMAT('  ',E10.5,E14.5)
 53   FØRMAT('  ',1P2E13.4,-1P2D13.4)
      END
```

imprime de acuerdo con los datos leídos, los resultados que se indican:

```
.12345E␣02␣␣-0.78980E␣01
.76543D␣02␣␣␣0.21000D␣02
␣␣␣1.2345E␣01␣␣-7.8980E␣00␣␣␣0.0765D␣03␣␣␣0.0210D␣03
```

Ejemplo 41:

Se tienen los siguientes datos:

```
␣45.3␣␣28␣14E+1␣4.D2␣2.+1␣␣3−2
␣453␣␣453␣␣453␣␣453␣
```

El programa que figura a continuación:

```
C  EJEMPLØ 41.
C  USØ  DE CØDIGØ DE FØRMATØ F
       READ (1,51)A,B,C,D,E,F
       WRITE(3,52)A,B,C,D,E,F
       READ (1,53)A,B,C,D
       WRITE(3,54)A,B,C,D
       STØP
  51   FØRMAT(2F5.2,4F5.0)
  52   FØRMAT(6F8.2)
  53   FØRMAT(1PF5.2,2PF5.2,−1PF5.2,−2PF5.2)
  54   FØRMAT(4F10.4)
       END
```

imprime de acuerdo con los datos leídos, los resultados que se indican.

```
␣␣45.30␣␣␣2.80␣␣140.00␣␣400.00␣␣␣20.00␣␣␣␣0.03
␣␣␣4.5300␣␣␣␣0.4530␣␣453.0000␣4530.0000
```

Ejemplo 42:

Se tienen los siguientes datos:

15.995.4-152.5D-1	1a tarjeta
15.995.4-152.5E-1	2a tarjeta

El programa que figura a continuación lee estos datos; la primera tarjeta, que contiene un dato con exponente D, la lee con código de formato E y la segunda, que contiene un dato con exponente E, la lee con código de formato D. Se hace uso nuevamente de la proposición REAL*8 para definir las variables X, Y y Z, de doble precisión.

```
C  EJEMPLØ 42.
C  USØ  DE CØDIGØS DE FØRMATØ E Y D
       REAL*8 X,Y,Z
```

```
      READ (1,51)A,B,C
      WRITE(3,52)A,B,C
      READ (1,53)X,Y,Z
      WRITE(3,54)X,Y,Z
      STØP
51    FØRMAT(2E5.2,E7.1)
52    FØRMAT(F8.1,2E12.3)
53    FØRMAT(2D5.2,D7.1)
54    FØRMAT(3D12.3)
      END
```

Los resultados que se imprimen son los que siguen:

ßßß16.0ßßß0.540EßØØßßß0.525EßØ1
ßß0.160DßØ2ßßß0.540DßØØßßß0.525DßØ1

Ejemplo 43:

Un profesor tiene un curso de 20 alumnos. Cada uno de ellos tiene tres notas correspondientes a pruebas parciales. El profesor ha preparado una tarjeta por alumno con la siguiente estructura:

Número del alumno	(NA) columnas 1 y 2
Nota 1	(C1) columnas 3 y 4
Nota 2	(C2) columnas 5 y 6
Nota 3	(C3) columnas 7 y 8

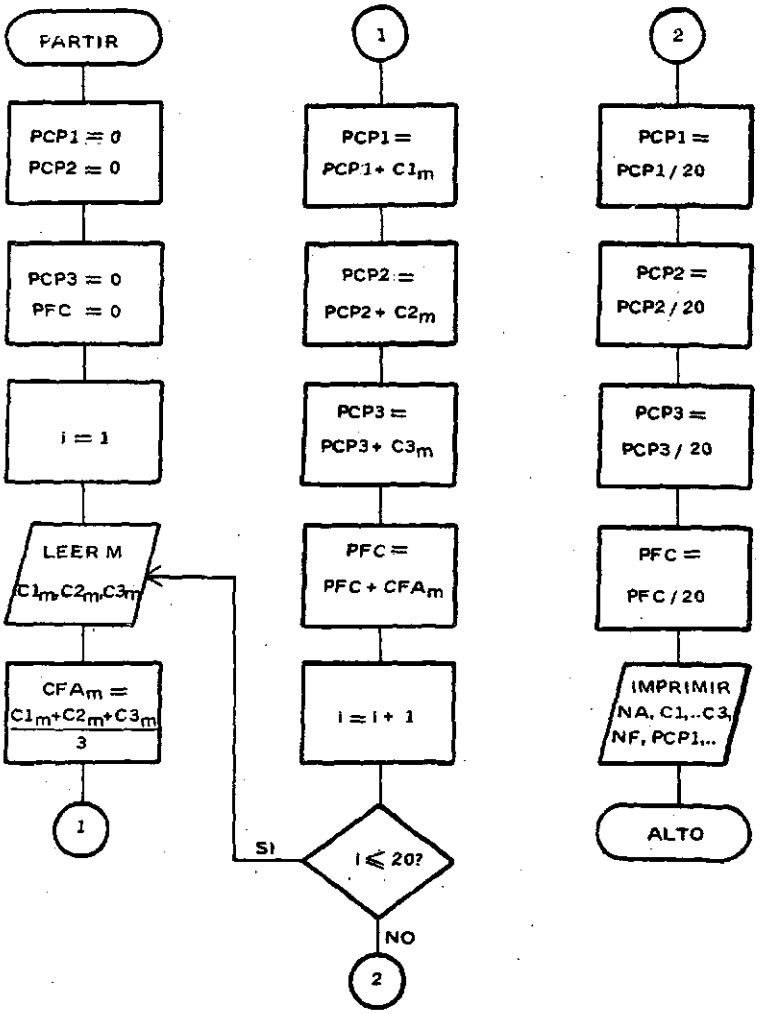
Se desea programar el cálculo de:

—promedio por alumno (nota final)	CFA
—promedio del curso en cada prueba	PCP
—promedio final del curso	PFC

Se supondrá que el número del alumno varía de 1 a 20, pero que las tarjetas pueden entrar desordenadas.

En una primera solución se considera que están todas las tarjetas y en una segunda solución que faltan tarjetas; en este último caso se terminará la lectura cuando se detecte una tarjeta en blanco.

a) Primera solución. El número del alumno servirá como índice para cuatro arreglos, que tendrán las tres notas parciales y la nota final. En un solo ciclo se incluye la lectura de los datos de un alumno, el cálculo de su nota final y la acumulación para el cálculo del promedio del curso.

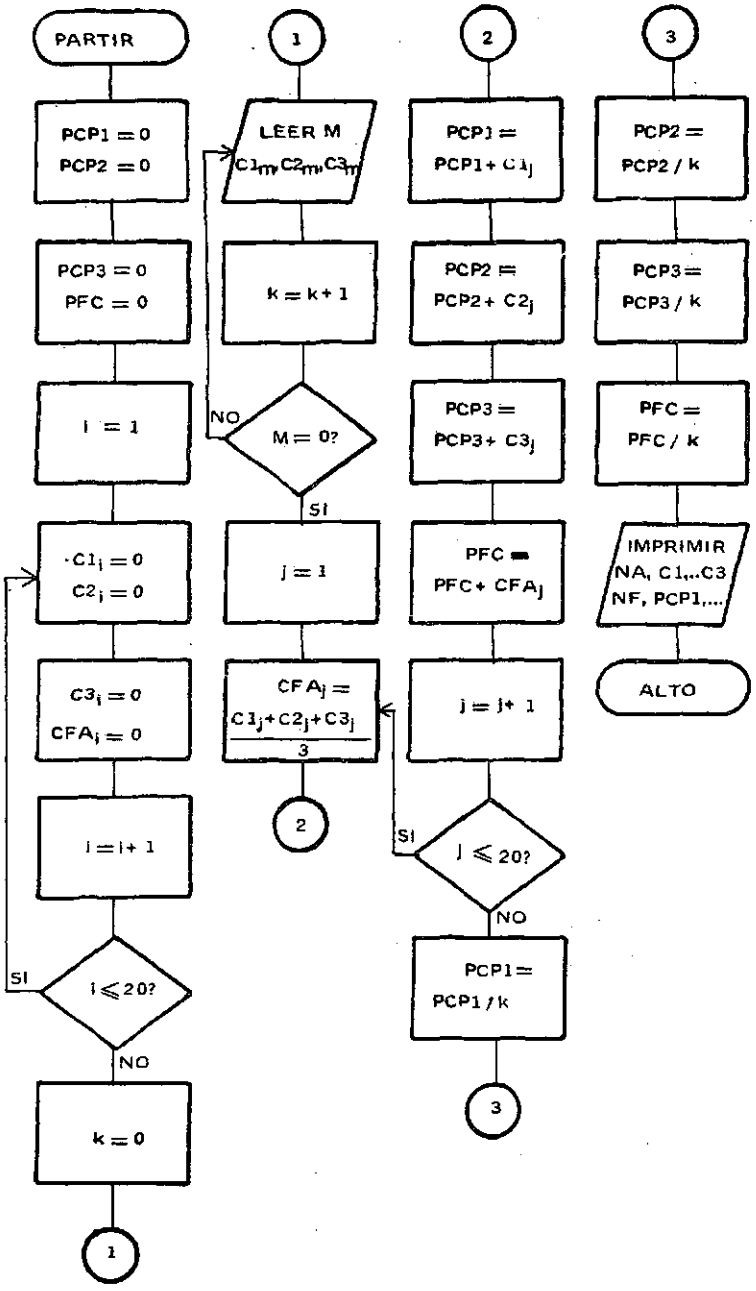


Observación:

A pesar de que el programa que se obtiene se ha minimizado en cuanto a número de instrucciones, el hecho de tener una proposición de lectura que es lenta, dentro de un ciclo en que aparecen proposiciones de asignación, hace que todo el proceso sea lento. Es preferible tener un ciclo de lectura en que se almacene toda la información para efectuar a continuación todos los cálculos.

```
C  EJEMPLØ 43. (SØLUCIØN A)
C  CALCULØ DE NØTAS DE UN CURSØ
       DIMENSIØN C1(20),C2(20),C3(20),CFA(20)
       PCP1 = 0.
       PCP2 = 0.
       PCP3 = 0.
       PFC  = 0.
       DØ 10 I=1,20
       READ (1,51) N,C1(N),C2(N),C3(N)
       CFA(N) = (C1(N)+C2(N)+C3(N))/3.
       PCP1 = PCP1 + C1(N)
       PCP2 = PCP2 + C2(N)
       PCP3 = PCP3 + C3(N)
   10  PFC  = PFC  + CFA(N)
       PCP1 = PCP1 / 20.
       PCP2 = PCP2 / 20.
       PCP3 = PCP3 / 20.
       PFC  = PFC  / 20.
       WRITE (3,52) (I,C1(I),C2(I),C3(I),CFA(I),I=1,20),
      2PCP1,PCP2,PCP3,PFC
       STØP
   51  FØRMAT(I2,3F2.1)
   52  FØRMAT(20(' ',I4,4F7.2/)//F12.2,3F7.2)
       END
```

b) Segunda solución. Dado que se supone que las tarjetas pueden entrar desordenadas, nuevamente se utilizará el número del alumno como índice de los arreglos en que se almacenará la información.


PARTIR
PCP1 = 0
PCP2 = 0
PCP3 = 0
PFC = 0
I = 1
C1i = 0
C2i = 0
C3i = 0
CFAi = 0
I = I + 1
I ≤ 20?
SI
NO
K = 0
1
LEER M
C1m, C2m, C3m
K = K + 1
M = 0?
NO
SI
J = 1
CFAj = (C1j + C2j + C3j) / 3
2
PCP1 = PCP1 + C1j
PCP2 = PCP2 + C2j
PCP3 = PCP3 + C3j
PFC = PFC + CFAj
J = J + 1
J ≤ 20?
SI
NO
PCP1 = PCP1 / k
3
PCP2 = PCP2 / k
PCP3 = PCP3 / k
PFC = PFC / k
IMPRIMIR
NA, C1,..C3
NF, PCP1,...
ALTO

```
C  EJEMPLØ 43. (SØLUCION B)
C  CALCULÓ DE NØTAS DE UN CURSØ
       DIMENSIØN C1(20),C2(20),C3(20),CFA(20)
       PCP1 = 0.
       PCP2 = 0.
       PCP3 = 0.
       PFC = 0.
       DØ 10 I=1,20
       C1(I) = 0.
       C2(I) = 0.
       C3(I) = 0.
   10  CFA(I)= 0.
       L = 0
   11  READ (1,51) N,C1(N),C2(N),C3(N)
       L = L + 1
       IF (N.NE.0) GØ TØ 11
       DØ 12 J=1,20
       CFA(J) =(C1(J)+ C2(J)+ C3(J))/3.
       PCP1 = PCP1 + C1(J)
       PCP2 = PCP2 + C2(J)
       PCP3 = PCP3 + C3(J)
   12  PFC = PFC + CFA(J)
       PCP1 = PCP1 / L
       PCP2 = PCP2 / L
       PCP3 = PCP3 / L
       PFC = PFC / L
       WRITE(3,52)(I,C1(I),C2(I),C3(I),CFA(I),I=1,20)
       WRITE(3,53)PCP1,PCP2,PCP3,PFC
       STØP
   51  FØRMAT(I2,3F2.1)
   52  FØRMAT(' ',I4,4F7.2)
   53  FØRMAT(///F12.2,3F7.2)
       END
```

Observación: La variable L se utiliza para contabilizar la cantidad de tarjetas leídas y calcular, con ese valor, los promedios que se desean.

3. *Código de formato L*

i) Estructura del código

a L w

donde:

a: es factor de repetición
w: es cantidad de caracteres

ii) Función. Se utiliza para transferir datos lógicos. En ENTRADA el dato debe estar formado por: un blanco al menos, seguido

por la letra T o la letra F y a continuación cualquier tipo de caracteres. La letra T causa que sea asignado el valor VERDADERO (TRUE) a la variable que figura en la lista de entrada/salida. La letra F causa que sea asignado el valor FALSO (FALSE) a dicha variable. En SALIDA se imprime la letra T o la F, según sea el valor de la variable, verdadero o falso respectivamente. La letra se ajusta a la derecha en el campo de salida, precedida por *w-1* blancos.

Ejemplo 44:

Se tienen los siguientes valores, perforados en tres tarjetas.

```
bTbbbFALbTALbFIN          1a tarjeta
bTRUEbbbFALSEbbTbF        2a tarjeta
bT+*                      3a tarjeta
```

El programa que figura a continuación lee esas tarjetas e imprime los valores leídos:

```
C  EJEMPLØ 44.
C  USØ  DE CØDIGØ DE FØRMATØ L
       LØGICAL  A,B,C,D,E*1
       READ(1,51)A,B,C,D
       WRITE(3,52)A,B,C,D
       READ(1,53)A,B,C,D
       WRITE(3,52)A,B,C,D
       READ(1,51)E
       WRITE(3,52)E
       STØP
  51   FØRMAT(4L4)
  52   FØRMAT(' ',2L4,L8,L1)
  53   FØRMAT(2L7,2L2)
       END
```

los resultados que se obtienen impresos son:

```
bbbTbbbFbbbbbbbbTF        1a línea
bbbTbbbFbbbbbbbbTF        2a línea
bbbT
```

4. *Código de formato G*

Este código se utiliza en reemplazo de los códigos I,F,E,D o L.

i) Estructura del código

p a G w.s

donde:

p: es factor de escala
a: es factor de repetición
w: es cantidad de caracteres
s: es parte fraccionaria de entrada y dígitos significativos en salida.

ii) Función. Permite la transferencia de datos enteros, reales o lógicos, que correspondan a variables de su mismo tipo.

En ENTRADA se mantienen las mismas normas dadas para los códigos I,F,E,D y L. Si las variables son enteras o lógicas, la parte *s* del código de formato se puede omitir; si se especifica, es ignorada.

En SALIDA, igualmente, se mantienen las mismas normas dadas para cada código en particular. Si las variables son enteras o lógicas, la parte *s* se puede omitir; en caso contrario, se ignora. Para datos reales, la parte *s* indica el número de dígitos significativos que se desea imprimir, como también si el dato se desea imprimir con o sin exponente decimal. En este último caso, si el número es mayor o igual que 0.1 y menor o igual que 10**s (0.1≤x≤10**s), el número es impreso sin exponente decimal. En caso contrario, se imprimirá con el exponente decimal que le corresponda al tipo de variable, esto es, E o D.

Aun cuando el resultado sea impreso sin exponente decimal, el valor que se imprime aparece desplazado hacia la izquierda cuatro lugares que corresponden a la letra E o D seguida de dos dígitos con o sin signo. Si el exponente es positivo, no se imprime el signo, pero el espacio queda en blanco y debe reservarse. Lo anterior significa que es necesario contemplar en el valor *w* los cuatro espacios del exponente, el del punto decimal, el del signo si el dato es negativo y al menos un dígito que preceda al punto decimal. En total, entonces, *w* será igual a siete más la cantidad de dígitos significativos que se desea imprimir.

Ejemplo 45:

Suponiendo que se tienen tres tarjetas con los siguientes datos:

␣105224	1a tarjeta
␣49535.4E+16.2D−185.3+1␣4834	2a tarjeta
␣TITO␣F	3a tarjeta

El programa que figura a continuación contiene dos proposiciones de especificación: LOGICAL U,V para indicar que las variables U y V son lógicas, y REAL*8 D que declara que la variable D es de doble precisión (longitud 8 bytes).

```
C EJEMPLØ 45.
C USØ DE CØDIGØ DE FØRMATØ G
      LØGICAL U,V
      REAL*8 D
      READ(1,51) I,J
```

```
      WRITE(3,54) I,J
      READ (1,52) A,B,C,D,E
      WRITE(3,55) A,B,C,D,E
      READ (1,53) U,V
      WRITE(3,53) U,V
      STØP
51    FØRMAT(G4.2,G3)
52    FØRMAT(G5.2,3G6.1,1PG5.2)
53    FØRMAT(G5.1,G2)
54    FØRMAT(' ',G6.2,G6)
55    FØRMAT(' ',G6.3,3G10.3,0PG7.2)
      END
```

Con este programa se obtienen los resultados siguientes:

␣␣105␣␣224	1a línea
******␣␣54.0␣␣␣␣0.620␣␣␣␣␣␣853.␣␣␣4.8␣␣␣␣	2a línea
␣␣␣T␣F	3a línea

5. *Código de formato Z*

i) Estructura del código

a Z w

donde:

a: es factor de repetición
w: es cantidad de caracteres

ii) Función. Permite la transferencia de datos hexadecimales.

En ENTRADA, los blancos que figuren en el campo leído se consideran ceros hexadecimales. Cada byte contiene dos dígitos hexadecimales, de tal manera que si se lee un número impar de dígitos el dato se ajusta a la derecha y se rellena con un cero hexadecimal por la izquierda. Lo mismo ocurre cuando el campo es mayor que el necesario para los caracteres leídos, esto es, se rellena con ceros hexadecimales por la izquierda. Si el campo es menor que el necesario para los caracteres que se leen, se recortan los caracteres de orden superior.

En SALIDA, si el número de caracteres del dato es menor que w, el campo se rellena con blancos por la izquierda. Si el número de caracteres es mayor que w, se pierden los caracteres de orden superior. Ejemplo 46:

Se tienen las siguientes tarjetas de datos:

␣A␣BFFA␣1869ABCDEF45.5␣255	1a tarjeta
AABFAFBFEDFFFFFF0	2a tarjeta

El programa que figura a continuación lee la información de esas tarje-

tas e imprime. Se declara la variable B como de doble precisión con la proposición REAL*8 B.

```
C EJEMPLØ 46.
C USØ  DE CØDIGØ DE FØRMATØ Z
       REAL*8 B
       READ (1,51) I,A,B,C,J
       WRITE(3,52) I,A,B,C,J
       READ (1,53) J,A,I
       WRITE(3,54) J,A,I
       WRITE(3,55) J,I
       STØP
  51   FØRMAT(2Z4,Z10,F5.3,I3)
  52   FØRMAT(' ',I10,Z2,Z14,2Z10)
  53   FØRMAT(Z3,Z4,Z11)
  54   FØRMAT(' ',Z2/' ',Z3/' ',Z8)
  55   FØRMAT(2I10)
       END
```

Los resultados que se obtienen son:

```
ØØØØØ2571A000001869ABCDEFØØ422D8000ØØ000000FF
AB
AFB
FFFFFFF0
ØØØØØØ171ØØØØØØØ-16
```

En la primera lectura la variable I se define con los dígitos hexadecimales A0B ajustados a la derecha. Estos corresponden a:

$$A*16^2+0*16^1+B*16^0$$
$$10*256+0+11$$
$$2571$$

que es el valor que se imprime con la primera proposición WRITE.

La operación inversa se efectúa con las variables C y J, que se definen con los valores 45.5 y 255 respectivamente.

$$(45.5)_{10} = (2D.8)_{16} = 0.2D8*16^2$$

la característica será entonces:

$$(64+2)_{10} = (66)_{10} = (42)_{16}$$

como el signo es positivo no afecta a este valor y queda:

422D8000, que es el valor impreso.

En cuanto a la variable J.

$(255)_{10} = (FF)_{16}$ e internamente 000000FF

6. *Código de formato A*

i) Estructura del código

a A w

donde:

a: es factor de repetición
w: es cantidad de caracteres

ii) Función. Permite la transferencia de caracteres que quedan almacenados o lo están, en formato de carácter. Dado que cada carácter ocupa un byte, la cantidad de ellos que es transferida depende de la longitud con que haya sido definida la variable, la que puede ser de cualquier tipo.

En ENTRADA, si *w* es menor que la cantidad de bytes (cb) reservada para la variable leída, se leen *w* caracteres y se ajustan a la izquierda y se rellenan los bytes restantes con blancos. Si *w* es mayor que la cantidad de bytes reservada se saltan *w-cb* caracteres y se leen *cb* caracteres.

En SALIDA, si *w* es menor que la cantidad de bytes reservada, se imprimen *w* caracteres del extremo izquierdo de la variable. Si *w* es mayor que la cantidad de bytes, se imprimen *cb* caracteres precedidos por *w-cb* blancos.

Ejemplo 47:

La siguiente tarjeta de datos:

A+BCDEFG$-*HIJKL1234MNOPQ única tarjeta

es leída por el programa que figura a continuación:

```
C  EJEMPLØ 47.
C  USØ  DE CØDIGØ DE FØRMATØ A
       LØGICAL U
       REAL*8 A
       READ (1,51) I,A,B,U
       WRITE(3,52) I,A,B,U
       STØP
  51   FØRMAT(A4,2A8,A2)
  52   FØRMAT(' ',A2,A12/' ',A4,A6)
       END
```

el cual imprime los resultados que se indican en seguida:

```
A+ØØØØDEFG$ -*H          1a línea
1234ØØMN                 2a línea
```

Ejemplo 48:

Hacer un programa que entregue el gráfico de la función seno *x*. Los caracteres que se utilizan son los siguientes:

*I

El programa que figura a continuación entrega el gráfico de la función seno x, la cual se obtiene a base de una serie:

```
C  EJEMPLØ 48.
C  USØ  DE CØDIGØ DE FØRMATØ A
C  GRAFICØ DE LA FUNCIØN SEN
       DIMENSIØN A(130)
       READ (1,51) BLANCØ,ASTER, EJE
       DØ 10 I=1,130
   10  A(I)=EJE
       WRITE(3,52)A
       DØ 11 I=1,130
   11  A(I)=BLANCØ
       A(65)=EJE
       X = 0.
       DX= 0.07854
       EPSIL = 1.E-5
   12  TER = X
       N = 1
       SUM = 0.
   13  SUM = SUM + TER
       TER = -TER *X*X/(N+1)/(N+2)
       IF (TER)14,15,15
   14  DELTA =-TER
       GØ TØ 16
   15  DELTA = TER
   16  IF (DELTA - EPSIL) 18,17,17
   17  N = N + 2
       GØ TØ 13
   18  Y = SUM + TER
       K = 65 + 50*Y
       A(K)=ASTER
       WRITE(3,52)A
       IF (K - 65)19,20,19
   19  A(K)=BLANCØ
       GØ TØ 21
   20  A(K)=EJE
   21  X = X + DX
       IF (X.LE.6.2832)GØ TØ 12
       STØP
   51  FØRMAT(3A1)
   52  FØRMAT(' ',130A1)
       END
```

7. *Constante literal en una proposición* FORMAT *y código de formato H*

a) *Constante literal.*

Son cadenas de caracteres alfanuméricos y especiales, incluido el carácter blanco, que van como argumento de la proposición FORMAT

encerrados entre apóstrofos. Si aparece un carácter apóstrofo en la cadena de datos, debe ir seguido inmediatamente por otro apóstrofo. Al almacenar el argumento del FORMAT, se hace un análisis de los caracteres y al encontrarse dos apóstrofos seguidos se guarda sólo uno.

En ENTRADA, los caracteres de la tarjeta reemplazan a los caracteres de la cadena de caracteres *uno a uno*. Si se usan apóstrofos, se lee un número de caracteres igual al que está entre ellos. Si se usa código de formato H, se leen *w* caracteres.

En SALIDA se imprimen los caracteres que figuran entre apóstrofos o los *w* caracteres que siguen al código de formato H.

Ejemplo 49:

Se tienen tres tarjetas con los siguientes datos:

RESULTADØ	1a tarjeta
SE␣PRUEBA␣APØS''TRØFØ	2a tarjeta
RESULTADØ␣CØDIGØ␣H␣EN␣SALIDA	3a tarjeta

El programa que figura a continuación:

```
C EJEMPLØ 49.
C USØ  DE LITERAL Y CØDIGØ DE FØRMATØ H
       READ (1,51)
       WRITE(3,51)
       READ (1,52)
       WRITE(3,52)
       READ (1,53)
       WRITE(3,53)
       WRITE(3,54)
       WRITE(3,55)
       STØP
 51    FØRMAT(' PRUEBA DE LITERAL')
 52    FØRMAT(' TITULØ CØN CØDIGØ H ')
 53    FØRMAT(19H PRUEBA DE CØDIGØ H)
 54    FØRMAT(' LITERAL NØ NECESITA W')
 55    FØRMAT(20HCØDIGØ H NECESITA W)
       END
```

lee las tres tarjetas mencionadas e imprime los siguientes resultados:

```
RESULTADØ
SE PRUEBA APØS''TRØFØ
RESULTADØ CØDIGØ H
LITERAL NØ NECESITA W
CØDIGØ H NECESITA W
```

Se puede observar que con las órdenes de lectura se produjo el reemplazo de los argumentos de los FORMAT 51, 52 y 53 por el contenido de las tarjetas. De la lectura e impresión de la segunda tarjeta se concluye que al aparecer dos apóstrofos, como parte de los caracteres

encerrados entre otros dos, como ocurre si hay un literal en el argumento del FORMAT, sólo se almacena uno de ellos; en cambio, al leer dos apóstrofos para reemplazar un argumento o parte de él,se almacenan ambos.

8. *Código de formato X*

i) Estructura del código

w X

donde:

w: es cantidad de caracteres

ii) Función. El código de formato X causa que en ENTRADA se salten *w* caracteres y en SALIDA se inserten *w* blancos.

Ejemplo 50:

Se tiene la siguiente tarjeta de datos:

␣1234.5678.9␣365.423101

Las primeras cinco columnas definen a la variable N (número de proceso), a continuación es necesario saltarse siete columnas. Las dos columnas siguientes definen a la variable J, las cuatro que siguen a la variable A y las cinco últimas a la variable B.

Se desea calcular:

X = A + B**J

y la impresión debe tener la estructura siguiente:

posición 20

RESULTADO␣DEL␣PROCESO␣#.....

} 3 líneas en blanco

posición 5

DATOS:

} 1 línea en blanco

posición 7 posición 27 posición 47

A␣=␣␣␣ B␣=␣␣␣ J␣=␣␣␣

posición 5

↓

RESULTADO:

} 1 línea en blanco

posición 7

↓

XЬ=ЬЬЬ

El programa que figura a continuación resuelve el problema planteado. En ENTRADA, con la proposición 51 FORMAT (I5,7X,...) se produce la eliminación de las siete columnas que no interesan. En SALIDA, se hace notar que como argumento de la proposición FORMAT pueden aparecer mezclados todos los códigos de formato y constantes literales.

El resultado pedido se ha colocado inmediatamente después del programa.

```
C  EJEMPLØ 50.
C  USØ  DE CØDIGØ DE FØRMATØ X
       READ (1,51) N,J,A,B
       X = A + B**J
       WRITE(3,52) N,A,B,J,X
       STØP
  51   FØRMAT(I5,7X,I2,F4.1,F5.3)
  52   FØRMAT(' ' ,19X, 'RESULTADØ DEL PRØCESØ #',
      2I5/// ' ',4X,'DATØS:'//' ',6X,'A =',3X, F6.2,8X,
      3'B =',3X,F6.3,8X,'J =',3X,I4//' ',4X,
      4'RESULTADØ:'//' ',6X,'X =',3X,F11.3)
       END
```

ЬЬЬЬЬЬЬЬЬЬЬЬЬЬЬЬЬЬЬЬRESULTADO DEL PROCESO # 1234

ЬЬЬЬDATOS:

ЬЬЬЬЬЬA =ЬЬЬ 65.40ЬЬЬЬЬЬЬЬЬB =ЬЬЬ23.101ЬЬЬЬЬЬЬЬЬJ =ЬЬЬ 3

ЬЬЬЬRESULTADO:

ЬЬЬЬЬЬX =ЬЬЬ 12393.383

9. *Código de Formato T*

i) Estructura del código

Tr

donde:

r: indica posición dentro del registro FORTRAN

ii) Función. Se especifica la posición dentro de un registro FORTRAN a partir de la cual se iniciará la transferencia de datos.

La entrada y la salida pueden empezar en cualquier posición usando el código de formato T. Cuando la salida es impresa, la correspondencia entre *r* y las posiciones de la línea no es exacta debido a que el primer carácter es considerado como control de carro y no se imprime, luego la posición en la línea corresponde a *r–1*.

Ejemplo 51:

Se tienen los siguientes datos:

1234567890987654321

El programa que figura a continuación:

```
C EJEMPLØ 51.
C USØ DE CØDIGØ DE FØRMATØ T
      READ (1,51) XY,MN,JK,AB
      WRITE(3,52) JK,AB,MN,XY
      STØP
   51 FØRMAT(T12,F5.2,T1,I5,T17,I3,T6,E6.3)
   52 FØRMAT(T42,I3,T2,E12.6,T31,I5/' ',T2,E12.5)
      END
```

lee los datos mencionados e imprime los siguientes resultados:

posición 1 posición 30 posición 41

0.678909E␣03␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣12345␣␣␣␣␣321
␣0.87654E␣03

F. *Otras formas de* READ *y* WRITE

a) *Uso de arreglos de códigos de formato*

En las proposiciones READ y WRITE es posible utilizar, en vez de una proposición FORMAT, un arreglo con códigos de formato. El nombre de dicho arreglo reemplaza, en la estructura de las proposiciones READ y WRITE, al número de identificación de la proposición FORMAT.

Deben cumplirse las siguientes normas:

1) La información que se guarda en el arreglo debe ser idéntica a la que contiene la proposición FORMAT en su argumento, esto es, se elimina solamente el número de identificación y la palabra clave FORMAT.

2) Debe utilizarse un arreglo, aun cuando éste tenga que tener un solo elemento.

3) Si el argumento almacenado contiene literales y dentro de alguno de éstos hay doble apóstrofo, se deberá utilizar el argumento sólo en salida, en caso contrario debe ocuparse el código de formato H.

Debe tenerse en cuenta que cada carácter ocupa un byte, sin embargo, es conveniente especificar un arreglo con mayor capacidad que la necesaria para guardar todos los caracteres del argumento, de tal manera que si se cambia este último no sea necesario tener que modificar las dimensiones del arreglo cada vez.

Ejemplo 52:

El programa siguiente almacena códigos de formato en un arreglo y utiliza éste para entrada y salida.

```
C  EJEMPLØ 52.
C  USØ  DE ARREGLØ DE CØDIGØS
       DIMENSIØN FMT(3)
       READ (1,50) FMT
       READ (1,FMT) L,A,B
       X = (A + B)**L
       Y = (A - B)**L
       M = L * 2
       WRITE(3,FMT) M,X,Y
       STØP
  50   FØRMAT (3A4)
       END
```

Los datos que lee el programa son:

```
                    (I4,2F7.2)
                    ØØ2ØØ10.5ØØØ11.1
```

y se obtienen los siguientes resultados:

```
                    ØØ4Ø466.56ØØØ0.36
```

b) *Proposición* NAMELIST

Es una proposición de especificación, que permite utilizar las proposiciones READ y WRITE sin especificar lista de entrada/salida. Para lograr esto se declaran con NAMELIST nombres de listas de entrada/salida y a dichos nombres se refieren las proposiciones READ y WRITE.

1) *Estructura de la proposición*

NAMELIST /X_1/$lista_1$/X_2/$lista_2$/,.../X_n/$lista_n$

donde:

Los X_i : representan nombres de lista
las $listas_i$: son listas de entrada/salida

2) *Función*

Se asignan nombres simbólicos a listas de entrada/salida. Estos nombres no deben ser los mismos de variables o arreglos y deben aparecer encerrados entre operadores de división. Los de variables o arreglos pueden pertenecer a más de una lista. Los declarados en la proposición NAMELIST deben utilizarse sólo en proposiciones en entrada/salida.

3) *Estructura de los datos de entrada para* NAMELIST

Los datos deben tener un formato especial para ser leidos con nombres declarados en la proposición NAMELIST.

i) El primer carácter en cada registro que se va a leer debe ser blanco.

ii) El segundo carácter en el primer registro de un grupo de registros de datos debe ser & (epsilón), seguido inmediatamente por el nombre declarado en NAMELIST. Este nombre no debe contener blancos y debe ser seguido por un blanco.

iii) El nombre debe ser seguido, después del blanco, por los ítem de datos, separados entre sí por coma, opcional después del último ítem de datos.

iv) El término del grupo de datos se señala mediante

&END

v) El ítem de datos tiene la estructura siguiente:

nombre simbólico = constante(s)

donde:

nombre simbólico es el nombre de una variable, con o sin subíndices, o nombre de arreglo

constante (s) puede ser de cualquier tipo. Si es lógica, puede tener la forma T y F o .TRUE. y .FALSE.. Si son varias constantes que definen a un nombre de arreglo, deben ser, en cantidad, menor o igual al número de elementos del arreglo. Si la misma constante se repite en forma sucesiva, puede adoptarse la notación i*constante, siendo *i* el número de veces que la constante se repite.

vi) Los nombres simbólicos que aparecen en los datos de entrada deben estar declarados como parte de al menos una lista en la proposición NAMELIST; sin embargo, el orden que ellos tengan es arbitrario.

vii) En las listas de la proposición NAMELIST no puede haber nombres que sean parámetros formales (ver "Subprogramas").

viii) Los blancos que figuren después de enteros y exponentes son tratados como ceros.

ix) Si en la primera tarjeta no encuentra el nombre simbólico con el cual se está leyendo, lo sigue buscando en los grupos NAMELIST siguientes.

4) *Estructura de los datos de salida*

Los datos se escriben conservando la estructura que tendrían al ser leídos a través de la proposición NAMELIST. Los campos de salida están diseñados para contener todos los dígitos significativos del dato. Los arreglos son escritos por columna.

Ejemplo 53:

El programa que figura a continuación:

```
C EJEMPLØ 53.
C USØ  DE PRØPØSICIØN NAMELIST
       NAMELIST /NØM1/L,A,B/NØM2/M,X,Y
       READ (1,NØM1)
       X = (A+B)**L
       Y = (A-B)**L
       M = L*2
       WRITE(3,NØM2)
       STØP
       END
```

lee los siguientes grupos de datos, en procesos separados

```
&NØM1 A=10.5,L=2,B=11.1&END          proceso 1
                                   2 líneas en blanco
&NØM1 A=10.5                         proceso 2
L=2,B=11.1,&END
```

y en ambos se obtienen los mismos resultados que se indican enseguida:

```
&NØM2
M=          4,X=  466.55957     ,Y=  0.35999930
&END
```

Obsérvese que se imprime & en vez de & . Sin embargo, la configuración o representación interna es la misma.

Ejemplo 54:

Se tienen los siguientes datos:

```
&NOM1 A = 8.5,B = 1.5,L = 2&END       la tarjeta
&NOM2 C = 5.5,D = 4,5,I = 3&END       2a tarjeta
```

El programa que figura a continuación lee la primera vez con el nombre simbólico NØM2, que no se encuentra en la primera tarjeta. Se salta la primera y consulta en la segunda si está el nombre buscado. El proceso se realiza en forma correcta; sin embargo, el primer grupo NAMELIST se pierde.

```
C EJEMPLØ 54.
C USØ  DE PRØPØSICIØN NAMELIST
       NAMELIST /NØM1/L,A,B/NØM2/I,C,D/NØM3/M,X,Y
       READ(1,NOM2)
```

```
      X = (C+D)**I
      Y = (C-D)**I
      M = I*2
1     WRITE(3,NØM3)
      READ(1,NØM1,END=2)
      X = (A+B)**L
      Y = (A-B)**L
      M = L*2
      GØ TØ 1
2     STØP
      END
```

Los resultados que se obtienen son:

```
&NOM3
M =      6,X = 1000.000    , Y = 1.0000000

&END
```

G. *Otras proposiciones secuenciales de entrada/salida*

a) Proposición END FILE

i) Estructura de la proposición

END FILE *a*

donde:

a: es una constante entera sin signo o variable entera sin signo y sin subíndices, que representa un número de referencia de conjunto de datos.

ii) Función. Se define el final o término del conjunto de datos asociado con *a*. Una proposición WRITE después de END FILE define el comienzo de un nuevo conjunto de datos.

b) Proposición REWIND

i) Estructura de la proposición

REWIND a

donde:

a : es una constante entera sin signo o variable entera sin signo y sin subíndices que representa un número de referencia de conjunto de datos.

ii) Función. Causa que una proposición READ o una proposición WRITE, inmediatamente posterior, se refieran al primer registro del primer conjunto de datos asociado con *a*

c) Proposición BACKSPACE

i) Estructura de la proposición

BACKSPACE a

donde:

a : es una constante entera sin signo o variable entera sin signo y

sin subíndices, que representa un número de referencia de conjunto de datos.

ii) Función. Causa que se efectúe el retroceso de un registro lógico en el conjunto de datos asociado con *a*. Si el conjunto de datos estaba en su comienzo al darse la orden, ésta no tiene efecto.

Al término de un archivo deben especificarse dos proposiciones BACKSPACE si se desea recuperar el último registro lógico grabado.

Ejemplo 55:

```
C  EJEMPLØ 55.
C  USØ DE LAS PRØPØSICIØNES
C  END  FILE, REWIND Y BACKSPACE
       DIMENSIØN A(20),B(20),C(20)
       N = 8
    1  READ (1,51) A,B,C
       WRITE(N,51) A,B,C
       IF(C(1).NE.O.) GØ TØ 1
       END FILE N
    2  READ (1,51) A,B,C
       WRITE(N,51) A,B,C
       IF(C(1).NE.O.) GØ TØ 2
       END FILE N
       REWIND N
    3  READ (N,51,END=4) A,B,C
       WRITE(3,52) A,B,C
       GØ TØ 3
    4  READ (N,51,END=5) A,B,C
       WRITE(3,52) A,B,C
       GØ TØ 4
    5  READ (N,51,END=6)
       GØ TØ 5
    6  BACKSPACE N
       BACKSPACE N
       READ (N,51,END=7) A,B,C
       WRITE(3,52) A,B,C
    7  STØP
   51  FØRMAT(20F4.1)
   52  FØRMAT('  ',20F6.1)
       END
```

Con este programa se graban dos archivos en un carrete de cinta magnética, cada uno de los cuales está formado por dos registros físicos y cada registro físico por tres registros lógicos. El término de cada archivo se obtiene con la proposición END FILE que causa la grabación de una marca de fin de archivo. La proposición REWIND rebobina la cinta al punto de carga. Las dos proposiciones BACKSPACE posicionan la cinta en el último registro lógico grabado.

H. *Problemas propuestos*

a) ¿Cuántas tarjetas se leen con la siguiente serie de proposiciones:

```
      READ (1,3)(A(I),I=1,5), (B(I),I=1,7)
3     FØRMAT (2F8.3,3(F5.2/F4.1))     ?
```

b) Se tienen los siguientes datos:

$$A = 2.5 \ , \ B = -37.72 \ , \ C = -732.5 \ , \ I = 876$$

i) Ordenarlos en tarjetas y leerlos

ii) ¿Cómo quedan los resultados con:

```
      WRITE (3,2) A,B,I,C
2     FØRMAT (2F8.3,I5)        ?
```

c) Se tiene una tarjeta perforada como se indica a continuación:

```
0203532.4729399087
```

Se pide la impresión, luego de ejecutar el siguiente programa:

```
      READ (1,3) I,AI,BJ,L,DATØ
      WRITE(3,5) AI,BJ,DATØ,L,I
      STØP
1     FØRMAT (3F9.2/(I4))
3     FØRMAT (I4,F6.4,F3.2,I3,F7.2)
5     FØRMAT (3F5.3/(I3))
      END
```

d) Se tiene una tarjeta con los siguientes caracteres

```
ABXDGFØERRTSRANQPYØR
```

se piden los resultados luego de procesar el siguiente programa:

```
      READ (1,3) A,B,C,D
      WRITE(3,5) A,C,B,D
      STØP
3     FØRMAT (4A5)
5     FØRMAT (A1,A3,A4,A2)
      END
```

4. *Proposiciones de Especificación*

Las proposiciones de especificación proporcionan al compilador información acerca de la naturaleza de los datos, como también información que le permite asignar a dichos datos ubicaciones en memoria o reservar memoria para resultados.

Toda proposición de especificación debe preceder a la primera proposición ejecutable del programa fuente.

A. *Proposición* DIMENSION

i) Estructura de la proposición

DIMENSIØN $a_1(k_1), a_2(k_2), ..., a_n(k_n)$

donde:

los a_i son nombres de arreglos

los k_i representan las dimensiones del arreglo. Cada k_i está compuesto de una a siete constantes enteras, sin signo, separadas entre sí por coma cuando hay más de una. Cada constante representa el valor máximo que puede tener el subíndice respectivo dentro del arreglo. Cada k_i puede contener variables enteras, de longitud 4 bytes, sólo cuando la proposición DIMENSION en la cual ellas aparecen forma parte de un SUBPROGRAMA (véase el capítulo Subprogramas "Dimensiones en tiempo de ejecución").

ii) Función. Permite asignar memoria a los arreglos que se utilizan dentro del programa fuente.

Ejemplo 56:

```
C  EJEMPLØ 56.
C  USØ  DE PRØPØSICIØN DIMENSIØN
       DIMENSIØN A (10,10)
       S = -3.
       DØ 10 I = 1,4
       S = S + 4.
       DØ 10 J = 2,4
       A(I,1) = S
  10   A(I,J) = A(I,J-1) + 1.
       WRITE(3,51)((A(I,J),J=1,4),I=1,4)
       STØP
  51   FØRMAT(4(F6.2,5X))
       END
```

Los resultados que se obtienen con el programa anterior son los siguientes:

1.00	2.00	3.00	4.00
5.00	6.00	7.00	8.00
9.00	10.00	11.00	12.00
13.00	14.00	15.00	16.00

B. *Proposiciones de tipo*

Existen dos clases de proposiciones de tipo: la proposición IMPLICIT y las proposiciones de especificación explícitas.

a) *Proposición* IMPLICIT

Debe ser la primera proposición en un programa principal y la segunda en un subprograma y no puede haber más de una en ninguno de los dos.

i) Estructura de la proposición

IMPLICIT $tipo_1 * s_1(a_{11}, a_{12}, ...), ..., tipo_n * s_n(a_{n1}, a_{n2}, ...)$

donde:

los $tipo_i$: pueden ser algunas de las siguientes palabras claves:

INTEGER, REAL, LOGICAL

los s_i: son constantes enteras sin signo, opcionales y representan alguna de las longitudes permitidas para el tipo al cual están asociadas. Si no se coloca, debe eliminarse el asterisco que le precede en la estructura

los a_{ij} : representan un carácter o un rango de caracteres alfabéticos (A, B, ...,Z,$). El rango se representa a su vez por el primero y por el último carácter, separados entre sí por un operador de resta y manteniendo el mismo orden en que están en el conjunto de caracteres alfabéticos, esto es, el rango de C a J se representa por (C–J) y no (J–C).

ii) Función. Se especifica el tipo y longitud de todas las variables, arreglos y funciones del usuario (véanse los "Subprogramas"), cuyos nombres empiecen con una letra en particular.

Tiene prioridad sobre la declaración predefinida de tipo.

Ejemplo 57:

1) IMPLICIT INTEGER (A–F), REAL (I–N)

Todas las variables cuyos nombres empiecen con las letras incluidas en el rango A a F son declaradas enteras y aquéllas cuyos nombres empiecen con las letras del rango I a N son declaradas reales de simple precisión.

2) IMPLICIT INTEGER * 2 (A–E, Ø–$), REAL*8 (J,K)

Todas las variables cuyos nombres empiecen con las letras de los rangos A a E y O a $ son declaradas enteras de longitud dos bytes y aquéllas cuyos nombres empiecen con las letras J y K son declaradas reales de doble precisión.

b) *Proposiciones de especificación de tipo explícitas*

i) Estructura de las proposiciones

$Tipo * S\ a_1 * s_1(k_1)/X_1/, a_2 * s_2(k_2)/X_2/, ..., a_n * s_n(k_n)/X_n/$

donde:

Tipo es INTEGER, REAL, LOGICAL

S y s_i: son constantes enteras sin signo, opcionales y representan alguna de las longitudes permitidas para el tipo al cual están asociadas. Si no se coloca, el asterisco que le precede en la estructura debe eliminarse

los a_i: son nombres de variables, arreglos o funciones (véanse los "Subprogramas")

los k_i: son opcionales y representan las dimensiones del arreglo (véase "Proposición DIMENSION")

los x_i: son opcionales y representan valores de datos iniciales. Si no se coloca, se omiten los operadores de división//.

ii) Función. Declara el tipo de una variable, arreglo o resultado de una función por su nombre, independiente del primer carácter de éste. Se puede, además, dar la dimensión de los arreglos cuyo tipo se declara (en ese caso, no deben aparecer en la proposición DIMENSION).

Si se desea asignar valores iniciales a las variables o arreglos que se estén declarando, o ambas cosas a la vez, esos valores se encierran entre operadores de división inmediatamente a continuación de las variables o arreglos que se inicialicen. Este se efectúa para el arreglo o variable inmediatamente precedente. Debe haber correspondencia entre el tipo del dato y la variable respectiva, exceptuando el caso de constantes literales o hexadecimales. Si los valores que se desea asignar se repiten en forma sucesiva, se puede utilizar la notación *i** constante, en que *i* indica el número de veces que se repite la constante. Para inicializar arreglos, éstos deben estar dimensionados en la misma proposición o en proposiciones DIMENSION o COMMON precedentes. No se pueden asignar valores iniciales a nombres de función.

La declaración de tipo mediante proposiciones de especificación explícitas tiene prioridad sobre la proposición IMPLICIT y sobre la declaración predefinida.

Ejemplo 58:

Se tienen los siguientes datos:

ABCDEFGHIJKLMNØPQRSTUVWXYZ$

El programa que figura a continuación:

```
C EJEMPLØ 58.
C USØ DE LAS PRØPØSICIØNES DE ESPECIFICACIØN
C INTEGER, REAL, LØGICAL
      INTEGER*2 ALFA, BETA, GAMA*4/2555/
      INTEGER A/'DATØ'/,B/ZFF/
      REAL*8 C(2,3)/5*1.,2./,D*4/Z42FF0000/
      LØGICAL*1 L,M,N/ØF/
      LØGICAL Ø,P,Q/T/,R/.FALSE./
      READ (1,51) ALFA, BETA,L,M,Ø,P
      WRITE (3,52) ALFA,BETA,GAMA,A,B,C,D
      WRITE (3,53) L,M,N,Ø,P,Q,R
      STØP
```

```
51    FØRMAT(2A4,2A3,2A6)
52    FØRMAT(' ',2A8,I6,A6,I6/(' ',3D11,3I)
53    FØRMAT(' ',2A4,L4,2A8,2L4)
      END
```

lee los datos mencionados y entrega los siguientes resultados impresos:

```
bbbbbbCDbbbbbbGHbb2555bbDATØbbb255      1a línea
bb0.100DbØ1bb0.100DbØ1bb0.100DbØ1       2a línea
bb0.100DbØ1bb0.100DbØ1bb0.200DbØ1       3a línea
bb0.255EbØ3                             4a línea
bbKbbNbbbFbbbbQRSTbbbbWXYZbbbbTbbbF     5a línea
```

c) *Proposición* DOUBLE PRECISION

i) Estructura de la proposición

DOUBLE PRECISION $A_1(k_1)$, $a_2(k_2)$,..., $a_n(k_n)$

donde:

los a_j: representan nombres de variables, arreglos o funciones (ver "Subprogramas")

los k_j: son opcionales y representan las dimensiones del arreglo (ver "Proposición DIMENSION")

ii) Función. Se declara en forma explícita que las variables, arreglos o resultados de funciones que aparecen en la proposición son reales de doble precisión.

Ejemplo 59:

i) DOUBLE PRECISION A,X,Z,
ii) DOUBLE PRECISION I,J(20),B(10,10)
iii) DOUBLE PRECISION C(5,4,5),N,Y

C. *Problemas propuestos*

a) Se pide el valor de A y de S al final del programa siguiente:

```
      INTEGER A
      S = 0.
      A = 5
    5 DO 10 I = 1,A
   10 S = S+A*I
      A = A+2
      IF(A - 10)5,5,6
    6 IF(S - 100.)7,7,8
    7 A = 1
      GO TO 9
    8 A = 2
    9 GO TO (11,12),A
   11 S = 0.
   12 STOP
      END
```

b) Programar el cálculo de la suma de los cuadrados de los 100 elementos de una lista

$$S = \sum_{i=1}^{100} a_i^2$$

c) Se tiene un arreglo bidimensional con 20 renglones y 10 columnas. Programar el cálculo de la suma de los elementos

d) Se tienen dos matrices A(M,N) y B(M,N). Programar

$$C = A + B$$

e) Se tienen dos matrices A(i,j) y B(j,k). Se pide programar el cálculo de:

$$C(i,k) = A(i,j) * B(j,k)$$

f) Se tienen dos listas A y B con 50 elementos cada una. Programar el cálculo de los elementos de un arreglo C, como sigue:

si $a_i \neq 0$ C_i es la suma de los elementos de B, menos b_i
si $a_i = 0$ C_i es la suma de los elementos de A, hasta a_i

g) Se tienen dos listas A y B de *n* elementos cada una ($N < 500$). Se pide una lista C de N elementos formados de la siguiente manera:

$$C_i = \sqrt{\frac{|b_i + a_i|}{1 + |a_i|}}$$

para el cálculo de la raíz usar la fórmula iterativa

$$X_{i+1} = \frac{1}{2}\left(X_i + \frac{A}{X_i}\right)$$

h) Tabular la siguiente función:

$$Z = \frac{X^2 - 2XY - Y^2}{2X - 3Y}$$

para

$$X = -1.(0.1)1.$$
$$Y = -1.(0.1)1.$$

Si el denominador es menor o igual a 0.01 suponerlo 0 y hacer $Z = 10**75$

i) Tabular la siguiente función:

$$Y = AX^2 + B$$

para $X = 0.1(0.1)2.$
$A = -5.(1.)5.$
$B = 0.(0.5)10.$

j) Se tiene una lista de valores, LISTA (1000). Se pide: el cuadrado de los elementos, el cubo de los elementos y el valor recíproco de ellos. Además la suma de los cuadrados, de los cubos y de los recíprocos.

k) Dadas 100 tarjetas, en cada una de las cuales se tienen perforados tres valores (separados entre sí por blanco), escribir un programa para leer las tarjetas y colocar los valores en listas A, B y C respectivamente.

l) Escribir un programa que lea 100 valores. Los 50 primeros se almacenan en un arreglo A y los restantes en un arreglo B.

m) Se tiene en memoria una lista J de cien elementos. Escribir los valores que corresponden a elementos de índice impar.

n) Se pide la salida del siguiente programa:

```
      DIMENSIØN A(12)
      S = 1.
      DØ 10 I=1,10,3
      S = S * I
      A(I) = S
      A(I+1) = 2 * S
   10 A(I+2) = S + 5
      WRITE(3,11)(A(I),I=1,12,2),(A(I),I=2,12,2)
   11 FØRMAT(F7.2,F5.0,(F7.2,2F6.1))
      STØP
      END
```

ñ) Escribir un programa que imprima los cien elementos de una matriz A, cuatro por línea, con la organización que se indica:

A(1) = A(2) = A(3) = A(4) =
A(5) = A(6) = etc.

o) Se pide indicar la salida de resultados de los siguientes programas:

i)

```
      DIMENSIØN A(100)
      S = 0.
      DØ 10 I=1,6
      S = S + 1.
   10 WRITE (3,1) S
    1 FØRMAT(4(F6.2,5X))
      STØP
      END
```

ii)

```
      DIMENSIØN A(100)
      A(1) = 1
      DØ 10 I=1,14
   10 A(I+1) = A(I)+1.
```

```
      WRITE (3,1) (A(I), A(I+1), A(I+2),I=1,3)
    1 FØRMAT(3(F6.2,5X))
      STØP
      END
```

iii)

```
      DIMENSIØN A(100)
      S = 0.
      DØ 10 I =1,16
      S = S + 1.
   10 A(I) = S
      WRITE(3,1)(A(I),I=1,16)
    1 FØRMAT(4(F6.2,5X))
      STØP
      END
```

iv)

```
      DIMENSIØN A(10,10)
      S = -3.
      DØ 10 I=1,4
      DØ 10 J=2,4
      S = S + 4.
      A(I,1) = S
   10 A(I,J) = A(I,J-1)+ 1.
      WRITE(3,1)((A(I,J),J=1,4),I=1,4)
    1 FØRMAT(4(F6.2,5X))
      STØP
      END
```

v)

```
      DIMENSIØN A(10,10)
      A(1,1) = 1.
      DØ 10 I=1,4
      DØ 10 J=2,4
      A(I+1,1)=A(I,1)+4.
   10 A(I,J)=A(I,J-1)+1.
      WRITE(3,1)((A(I,J),J=1,4),I=1,4)
    1 FØRMAT(4(F6.2,5X))
      STØP
      END
```

p) Se pide la impresión que entrega el siguiente programa:

```
      DIMENSIØN A(10),B(10)
      S = 0.
      A(1)=S
      A(2)=S + 1.
      DØ 10 I=3,10
      S = S + 1.
   10 A(I)=A(I-1)*(S+1) - A(I-2)*S
```

```
      K = 0
      DØ 20 I=1,10
      IF(A(I)-5)20,20,3
    3 K = K + 1
      B(K) = A(I)
   20 CØNTINUE
      WRITE(3,2)(B(J),J=1,K)
    2 FØRMAT(T10,F7.2,T30,F9.5,T70,F5.1)
      STØP
      END
```

q) Se pide la impresión que entrega el siguiente programa:

```
      DIMENSIØN B(10),A(10,10)
      DØ 10 I=1,5
   10 B(I) = I
      DØ 15 I=1,5
      A(I,1) = B(I)
      A(I,2) = A(I,1) + 1.
      DØ 15 J=3,5
      A(I,J) = 0.
      L = J - 1
      DØ 15 K=1,L
   15 A(I,J) = A(I,J) + A(I,K)
      WRITE(3,3)((A(L,K),L=1,5),K=1,5)
      STØP
    3 FØRMAT(5F10.2/(5F9.3))
      END
```

r) Los resultados de una encuesta fueron perforados con la siguiente disposición en tarjetas:

Nombre de la encuestada	columnas 1 a 24
Estado Civil	columna 25 (0 = soltera, 1 = casada y otros)
Nacionalidad	columnas 26 a 29
Número de hijos	columnas 30 y 31

Se desea saber:

El promedio de hijos
El porcentaje de madres solteras
El porcentaje de madres con más de tres hijos
El porcentaje de hijos naturales

Se ha colocado una última tarjeta que tiene perforado un nueve en columna 25 para utilizarla como fin de archivo.

5. *Subprogramas*

En programación se presenta con mucha frecuencia la necesidad de

tener que realizar repetidas veces partes de un programa, que corresponden a cálculos en los que la única variación se efectúa en los valores que toman las variables. Por ejemplo, en el cálculo siguiente:

PDX=((((A5*X+A4)*X+A3)*X+A2)*X+A1)*X+A0

se pueden tener los coeficientes constantes y realizar el cálculo de PDX para distintos valores de X o mantener constante X y efectuar el cálculo para distintos juegos de coeficientes. En ambos casos, el tener que detallar toda la proposición de asignación cada vez que se necesite un resultado, constituye una pérdida de tiempo que será mayor cuanto mayor sea la complejidad del cálculo o el número de proposiciones que haya que repetir.

Se evita esta situación con los subprogramas, denominados también rutinas o subrutinas, que pueden ser "llamados" por el programa principal o monitor cada vez que se necesite el resultado deseado.

Es importante tener presente que el uso de subprogramas permite darle modularidad a un programa, esto es, estructurarlo a base de módulos, cada uno de los cuales podría ser programado por personas distintas a base de: información que se entrega al módulo, proceso de cálculo y resultado(s) que debe entregar. Es necesario tener claro que el uso en sí de subprogramas no constituye modularidad dado que los subprogramas deben cumplir con normas precisas de construcción y funciones a realizar, dentro de la estructura total del programa.

Se tienen dos clases de subprogramas: subprograma FUNCTION y subprograma SUBROUTINE. Se ven además en este capítulo, las "funciones de proposición" y las funciones estándar.

A. *Funciones de proposición*

Las funciones de proposición se definen (declaran) y son "llamadas" dentro de la misma unidad de programa.

i) Estructura de la función de proposición

nombre $(a_1, a_2, \ldots, a_n)$ = expresión

donde:

nombre: es la identificación de la función

los a_i: son variables sin signo, sin subíndices, distintos entre sí llamados *parámetros formales* (argumentos vacíos). Debe haber al menos un parámetro formal

expresión: es cualquier expresión aritmética o lógica que no contenga variables con subíndices. Si dentro de la expresión aparece una función de proposición, ésta debe estar definida previamente.

ii) Función. Se establece un procedimiento de cálculo (la expresión) en el cual intervienen los parámetros formales. Dichos parámetros formales son reemplazados, uno a uno, por parámetros actuales o reales, cuando se realiza la llamada de la función.

La llamada de la función se ejecuta al aparecer el nombre de la función en una proposición de asignación, seguido de los parámetros actuales encerrados entre paréntesis.

Los parámetros actuales deben corresponder en tipo, número y orden con los parámetros formales.

Para la declaración del tipo de la función se aplican las mismas reglas que para la declaración del tipo de variables.

Una vez que los parámetros actuales reemplazan a los formales en el procedimiento de cálculo, se evalúa la expresión y el valor obtenido reemplaza a su vez la llamada de la función en la proposición de asignación.

Los nombres de los parámetros formales pueden aparecer en varias funciones de proposición y pueden asimismo ser utilizados como nombres de variables, pues constituyen en la práctica elementos distintos.

La función de proposición debe aparecer antes de cualquier proposición ejecutable. Si en ella aparece en la parte expresión una llamada de otra función de proposición, esta última debe estar previamente definida. La función de proposición no puede llamarse a sí misma.

Ejemplo 60:

a) *Definiciones válidas*

i) SUMC(A,B,C) = A*A+B*B+C*C
ii) POL(A1,A2,A3) = (A1*X+A2)*X+A3
iii) PROM(X,Y,Z) = (X+Y+Z)/3.
iv) VLOG(A,B,C) = A.AND.B.OR.C

b) *Definiciones no válidas*

i) ALFA(3.,X) = A*X+3. una constante aparece como parámetro formal
ii) POL(A(1),A(2)) = A(1)*X+A(2) aparecen variables con subíndices como parámetros formales y además en la expresión
iii) POLIN(X) = A(1)*X+A(2) variables con subíndices en la expresión
iv) TAB(X,Y) = X**2+Y*TAB(A,B) llamada a sí misma de la función

Ejemplo 61:

En una tarjeta se tienen perforados los siguientes datos:

1 15–1 2 25 3 –35–3 1 –2 25 15

que son leídos por el programa que figura a continuación:

```
C EJEMPLØ 61.
C USØ DE FUNCIØN DE PRØPØSICIØN
      DIMENSIØN X(8),Z(7)
```

```
      VALØR (XZ) = ((A3*XZ+A2)*XZ+A1)*XZ+A0
      READ (1,51) X,A3,A2,A1,A0
      Z(1) = VALØR(X(1))**2 + 5.
      Z(2) = .5 + VALØR (X(2)**2) *2.
      Z(3) = 3. * VALØR (X(3)+1.5)
      DØ 15 I=4,7
15    Z(I) = VALØR (X(I))*VALØR(X(I+1))
      WRITE(3,52) Z
      STØP
51    FØRMAT(12F3.1)
52    FØRMAT(' ',3(F10.2,2X))
      END
```

El programa entrega los resultados que se indican:

␣␣␣␣14.00␣␣␣␣␣␣17.28␣␣␣␣␣␣7.13
␣␣␣␣70.69␣␣␣␣␣␣195.75␣␣␣␣−1343.25
␣␣3805.88

Ejemplo 62:

```
C  EJEMPLØ 62.
C  USØ  DE FUNCIØN DE PRØPØSICIØN
      ALFA(Y) = (A*Y + B)*Y + A*B
      BETA(X) = (3*X + A)/(3*X + B) + ALFA (X)
      READ (1,51)A,B
      Y = ALFA(.5)
      Z = BETA(.5)
      WRITE(3,52) Y,Z
      STØP
51    FØRMAT(2F3.1)
52    FØRMAT('  ',2F10.3
      END
```

El programa anterior lee los siguientes datos:

2 3

y entrega los resultados que se indican a continuación:

␣␣␣␣8.000␣␣␣␣8.778

Ejemplo 63:

```
C  EJEMPLØ 63.
C  USØ  DE FUNCIØN DE PRØPØSICIØN
      LØGICAL*1 L,M,U,V,W,X,Y,Z,A,B,C,VALØG
      VALØG(A,B,C)=.NØT.A.AND.B.ØR.C
      READ (1,51) U,V,W,X,Y,Z
      IF(VALØG(U,V,W)) GØ TØ 1
      L = .FALSE.
      WRITE(3,52) L
    1 IF(VALØG(X,Y,Z)) GØ TØ 3
      M = .FALSE.
```

```
      2  WRITE(3,52)M
         STØP
      3  M = .TRUE.
         GØ TØ 2
     51  FØRMAT (6L2)
     52  FØRMAT (' ',2(L3,2X))
         END
```

El programa anterior lee los siguientes datos:

F T F T F T

y entrega el resultado que se indica a continuación:

T

B. *Funciones estándar*

Corresponden a procedimientos de cálculo de uso frecuente y que, por este motivo, han sido incorporados al lenguaje FORTRAN. Esto significa que es posible llamar a las funciones estándar en la misma forma en que se efectúa el llamado de las funciones de proposición, esto es, en una proposición de asignación se coloca el nombre de la función estándar, y a continuación encerrados entre paréntesis se especifican el o los argumentos.

A continuación se entrega una lista de las funciones estándar de uso más común. En el Apéndice A aparece una lista completa.

Función	Nombre	Definición
Logaritmo natural	ALOG DLOG	$y = \log_e x$ ó $y = \ln x$
Logaritmo decimal	ALOG10 DLOG10	$y = \log_{10} x$
Exponencial	EXP DEXP	$y = e^x$
Raíz cuadrada	SQRT DSQRT	$y = \sqrt{x}$ ó $y = x^{1/2}$
Seno	SIN DSIN	$y = \sin x$
Coseno	COS DCOS	$y = \cos x$
Tangente	TAN DTAN	$y = \tan x$
Cotangente	COTAN DCOTAN	$y = \cot an x$
Valor absoluto	IABS ABS DABS	$y = \lvert x \rvert$

Nota: Los nombres que empiezan con D corresponden a doble precisión, los otros a precisión simple, excepto IABS, que es el valor absoluto de un entero. El tipo del argumento debe corresponder a la precisión que se desea. Los argumentos de las funciones trigonométricas deben darse en radianes [–].

Ejemplo 64:

Programar el cálculo de:

$$y = e^{\log_e \sqrt{\frac{\operatorname{sen} x + \cos x}{|A+B|}}}$$

Con la ayuda de las funciones estándar, la solución se obtiene fácilmente:

```
Y = EXP(ALOG(SQRT((SIN(X)+COS(X))/ABS(A+B))))
```

C. *Subprograma* FUNCTION

Consiste en la proposición FUNCTION seguida de otras proposiciones entre las que debe aparecer, al menos una vez, una proposición de asignación en que la variable que figura a la izquierda del símbolo de asignación sea el nombre del subprograma, y una proposición RETURN. Al final de todas debe estar la proposición END.

La llamada del subprograma FUNCTION se efectúa en la misma forma que la llamada de una función de proposición.

El subprograma FUNCTION constituye un módulo independiente del programa principal, por lo cual se pueden definir en él: variables, arreglos y números de identificación de proposición, iguales a los definidos en el programa principal o en otros subprogramas. No puede contener proposiciones SUBROUTINE u otra proposición FUNCTION, ni llamarse a sí mismo.

a) *Proposición* FUNCTION

i) Estructura de la proposición

Tipo FUNCTION nombre * s (a_1,a_2,...,a_n)

donde:

Tipo puede ser: INTEGER, REAL, DOUBLE PRECISION o LOGICAL. Es optativo colocarlo

nombre: es el identificador de la función

*s: representa una de las longitudes permitidas para el tipo asociado.

Puede ser colocada o no, siempre que se haya especificado Tipo.

los a_i: son parámetros formales. Debe haber al menos uno.

ii) Función. Las proposiciones que siguen a la proposición

FUNCTION constituyen una declaración o definición de un procedimiento de cálculo, en el cual deben figurar los parámetros formales, que son reemplazados por los actuales cuando el subprograma es llamado (véase "Parámetros actuales en Subprogramas").

Normalmente se entrega un resultado a través del nombre del subprograma. Dicho nombre debe aparecer, al menos una vez, a la izquierda del símbolo de definición en una proposición de asignación. Es posible entregar más de un resultado haciendo uso de parámetros formales, los cuales tendrán que figurar en proposiciones de asignación igual que el nombre del subprograma.

La relación entre parámetros formales y actuales es la misma que en las funciones de proposición.

Si no se especifica "Tipo", éste se indica en alguna de las otras formas posibles, esto es: declaración predefinida, declaración explícita o por medio de la proposición IMPLICIT. Si se utiliza esta última dentro del subprograma, debe estar inmediatamente a continuación de la proposición FUNCTION. Cuando no se hace uso de la declaración predefinida, debe declararse el tipo también en el programa llamador.

b) *Proposición* RETURN

i) Estructura de la proposición

```
RETURN
RETURN i
```

donde:

i: es una variable sin subíndices, sin signo o constante entera sin signo. Se utiliza solamente en subprogramas SUBROUTINE (véase el capítulo "Proposición RETURN en subprograma SUBROUTINE).

ii) Función. Permite el retorno al programa o subprograma que ha llamado al subprograma en el que está la proposición RETURN considerada.

Indica, entonces, la conclusión lógica del cálculo y retorna el control y al menos un resultado numérico al programa llamador. Puede existir más de una proposición RETURN en un subprograma.

El primer formato se puede utilizar en subprogramas FUNCTION o SUBROUTINE. El segundo formato sólo se puede ocupar en subprogramas SUBROUTINE.

Ejemplo 65:

```
C EJEMPLØ 65.
C USØ  DE SUBPRØGRAMA FUNCTIØN
       READ (1,51) X,Y,Z
       ZETA = PRUEBA (X,Y,Z,U)
       WRITE(3,52) X,Y,Z,ZETA,U
       STØP
  51   FØRMAT(3F2.1)
```

```
52    FØRMAT(' DATØS',3F5.1/' RESULTADØS',2F7.2)
      END

      FUNCTIØN PRUEBA(A,B,C,D)
      PRUEBA =(2.*A + 3. *B)/4.*C
      D = PRUEBA ** 2
      RETURN
      END
```

El programa anterior lee los siguientes datos:

453040

y entrega los resultados que figuran a continuación:

DATØS␣␣4.5␣␣3.0␣␣4.0
RESULTADØS␣␣18.00␣324.00

Ejemplo 66:

En una tarjeta se tienen perforados los datos que se indican:

5101520253035404550556065707580859095l0

Los datos son leídos por el programa siguiente:

```
C  EJEMPLØ 66.
C  USØ  DE SUBPRØGRAMA FUNCTIØN
      DIMENSIØN A(20)
      READ (1,51)A
      PR = PROM(A)
      WRITE(3,52)
      WRITE(3,53)A,PR
51    FØRMAT(20F2.1)
52    FØRMAT(' MATRIZ A'/)
53    FØRMAT(' ',10F4.1/' ',10F4.1/' PRØMEDIØ',F7.2)
      STØP
      END
      FUNCTIØN PRØM(X,N)
      DIMENSIØN X(N)
      S = 0.
      DØ 10 I=1,N
10    S = S + X(I)
      PRØM = S/N
      RETURN
      END
```

que entrega los resultados que figuran a continuación:

MATRIZ A

␣0.5␣1.0␣1.5␣2.0␣2.5␣3.0␣3.5␣4.0␣4.5␣5.0
␣5.5␣6.0␣6.5␣7.0␣7.5␣8.0␣8.5␣9.0␣9.5␣1.0
PRØMEDIØ␣␣4.80

En el subprograma PROM se puede observar que el arreglo X tiene como dimensión una variable entera y el valor de esa variable es pasado al subprograma en el momento de llamarlo. Mayores detalles se pueden ver en el capítulo "dimensionamiento en tiempo de ejecución (objeto)".

Ejemplo 67:

```
C EJEMPLØ 67.
C USØ DE SUBPRØGRAMA FUNCTIØN
      READ (1,51)D,E,F
      Z = 1.5
      SQ = RAIZ(D,E,F)
      SQ = SQ * Z
      WRITE(3,52)SQ
      STØP
   51 FØRMAT(3F4.1)
   52 FØRMAT(1H0,F7.3)
      END
      FUNCTIØN RAIZ(A,B,C)
      VALØR(X) = (U*X+V)*X+W
      Z = 0.5
      W = Z**3 + 1.5*A
      V = Z**2 + A*B
      U = Z + A*B/C
      RAIZ =VALØR(Z)+A+B-C
      ZETA = VALØR(Z)
      WRITE(3,52)ZETA
   52 FØRMAT(' ',F9.2)
      RETURN
      END
```

El programa anterior lee los datos siguientes:

3.0 1.5–2.0

y entrega los resultados que figuran a continuación:

bbbb8.31
b22.219

Ejemplo 68:

```
C EJEMPLØ 68.
C USØ DE SUBPRØGRAMA FUNCTIØN
      REAL*4 LISTA
      INTEGER ALFA, SI,SU,EPSIL
      READ (1,51) A,B,C
      SI = ALFA (A,B,C)
      WRITE(3,52) SI
      SU = EPSIL(A,B,C)
      WRITE(3,52) SU
      SV = LISTA(A,B,C)
```

```
      WRITE(3,53) SV
      STØP
51    FØRMAT(3F2.1)
52    FØRMAT(' ',I5)
53    FORMAT(' ',F8.3)
      END
      FUNCTIØN ALFA(X,Y,Z)
      INTEGER ALFA
      ALFA = X*X + Y*Y + Z*Z
      RETURN
      END
      FUNCTIØN EPSIL (X,Y,Z)
      IMPLICIT INTEGER(E-F)
      EPSIL = X*Y*Z
      RETURN
      END
      FUNCTIØN LISTA (X,Y,Z)
      REAL*4 LISTA
      LISTA = X*Y*Z
      RETURN
      END
```

El programa anterior lee los datos siguientes:

354555

y entrega los resultados que figuran a continuación:

62
86
86.625

Ejemplo 69:

```
C EJEMPLØ 69.
C USØ  DE SUBPRØGRAMA FUNCTIØN
       IMPLICIT REAL(M-N)
       INTEGER SS,BETA,SW*2,GUIA*2
       READ (1,51) A,B,C
       SR = NETA (A,B,C)
       WRITE(3,53) SR
       ST = GAMA (A,B,C)
       WRITE(3,53) SI
       SS = BETA (A,B,C)
       WRITE(3,52) SS
       SW = GUIA (A,B,C)
       WRITE(3,52) SW
       STØP
51     FØRMAT(3F2.1)
52     FØRMAT(' ',I5)
53     FØRMAT(' ',F8.3)
       END
```

```
FUNCTIØN NETA(X,Y,Z)
REAL NETA
NETA = X*X + Y*Y + Z*Z
RETURN
END
INTEGER FUNCTIØN GUIA*2(X,Y,Z)
GUIA = (X+Y+Z)/3
RETURN
END
FUNCTIØN GAMA(X,Y,Z)
GAMA = (X+Y+Z)/3.
RETURN
END
INTEGER FUNCTIØN BETA(X,Y,Z)
BETA = (X+Y+Z)/3
RETURN
END
```

El programa anterior lee los datos siguientes:

354555

y entrega los resultados que figuran a continuación:

```
ØØ62.750
ØØØ4.500
ØØØØ4
ØØØØ4
```

D. *Subprograma* SUBROUTINE

Consiste en la proposición SUBROUTINE seguida de otras proposiciones entre las que debe aparecer, al menos una vez, una proposición RETURN y al final de todas, la proposición END.

En muchos aspectos es similar al subprograma FUNCTION y así constituye también un módulo, independiente del programa principal, con iguales características que aquél, en lo que se refiere a nombres de variables y números de identificación de proposición.

Las normas sobre parámetros formales y actuales son las mismas que en el subprograma FUNCTION, exceptuando el hecho de que en el subprograma SUBROUTINE puede no haber parámetros formales, en cuyo caso se omiten los paréntesis.

Difiere el subprograma SUBROUTINE del FUNCTION en la forma de ser llamado, dado que esta acción se realiza a través de una proposición CALL. Además, no necesariamente retorna un resultado numérico, el procedimiento de cálculo puede consistir en copiar una matriz o en transponerla, etc.

No puede contener proposiciones FUNCTION ni SUBROUTINE y tampoco puede llamarse a sí mismo.

a) *Proposición* SUBROUTINE

i) Estructura de la proposición

SUBROUTINE nombre(a_1,a_2,...a_n)

donde:

nombre: es el identificador de la subrutina
los a_i: son parámetros formales.

ii) Función. Las proposiciones que siguen a la proposición SUBROUTINE constituyen una declaración o definición de un procedimiento de cálculo. Si hay parámetros formales, se reemplazan por los parámetros actuales cuando el subprograma es llamado (véase "Parámetros actuales en Subprogramas").

Puede utilizar uno o más parámetros formales para retornar resultados al programa llamador. El nombre de la subrutina no puede aparecer en ninguna otra proposición en el subprograma.

Si se utiliza una proposición IMPLICIT, debe aparecer inmediatamente después de la proposición SUBROUTINE.

b) *Proposición* CALL

i) Estructura de la proposición

CALL nombre (a_1,a_2,...,a_n)

donde:

nombre; es el identificador de un subprograma SUBROUTINE
los a_i: son parámetros actuales.

ii) Función. Permite llamar a un subprograma SUBROUTINE

Ejemplo 70:

El programa siguiente lee un dato con el cual define a la variable X. Transfiere el dato leído al subprograma PRUEBA a través de una proposición CALL. El subprograma PRUEBA entrega al programa principal tres resultados a través de los parámetros formales P2, P3 y A. Los dos primeros se definen con proposiciones de asignación y el último con una proposición READ. Observar que el parámetro formal A es distinto del parámetro actual A.

```
C EJEMPLØ 70.
C USO DE SUBPRØGRAMA SUBRØUTINE
      READ (1,51) X
      CALL PRUEBA(X,A,B,C)
      SUMA = A + B + C
      WRITE(3,52) SUMA
```

```
      STØP
51    FØRMAT(F3.1)
52    FØRMAT(' ',F7.3)
      END
      SUBRØUTINE PRUEBA(X,P2,P3,A)
      P2 = 1.5*X**2 - 0.5
      P3 = 2.5*X**3 -1.5*X
      READ(1,51) A
      RETURN
51    FØRMAT (F4.0)
      END
```

Datos leídos:

```
.5
20.
```

Resultado obtenido:

```
 19.438
```

Ejemplo 71:

```
C EJEMPLØ 71.
C USØ  DE SUBPRØGRAMA SUBRØUTINE
       DIMENSIØN A(10),B(10)
       N = 10
       CALL LEA (A,N)
       CALL CALC(A,B,N,P)
       CALL IMP (A,B,N,P)
       STØP
       END
       SUBRØUTINE LEA (X,M)
       DIMENSIØN X(M)
       READ (1,51) X
       RETURN
51     FØRMAT(10F3.1)
       END
       SUBRØUTINE CALC(X,Y,M,PR)
       DIMENSIØN X(M),Y(M)
       PR = PRØM (X,M)
       CALL ØRDEN(X,M)
       DØ 10 I=1,M
10     Y(I) = X(I)*X(I)
       RETURN
       END
       FUNCTIØN PRØM (X,M)
       DIMENSIØN X(M)
       PRØM = 0.
       DØ 10 I=1,M
10     PRØM = PRØM + X(I)
       PRØM = PRØM/M
       RETURN
```

```
      END
      SUBRØUTINE ØRDEN(Z,N)
      DIMENSIØN Z(N)
      M = N - 1
      DØ 15 I=1,M
      K = I + 1
      DØ 10 J=K,N
      IF(Z(I).LE.Z(J))GØ TØ 10
      AUX = Z(I)
      Z(I) = Z(J)
      Z(J) = AUX
10    CØNTINUE
15    CØNTINUE
      RETURN
      END
      SUBRØUTINE IMP (X,Y,M,PR)
      DIMENSIØN X(M),Y(M)
      WRITE(3,52)
      WRITE(3,53) X
      WRITE(3,54)
      WRITE(3,53) Y
      WRITE(3,55) PR
      RETURN
52    FØRMAT(' MATRIZ DATØ'//)
53    FØRMAT(' ',5F5.1)
54    FØRMAT(' MATRIZ RESULTADØ'//)
55    FØRMAT(' PRØMEDIØ',F7.2)
      END
```

El programa y subprogramas anteriores procesan los datos siguientes:

1 3 4 2 7 10 5 6 9 8

y entregan los resultados que aparecen a continuación:

```
MATRIZ DATØ
  1.0   2.0   3.0   4.0   5.0
  6.0   7.0   8.0   9.0  10.0
MATRIZ RESULTADØ
  1.0   4.0   9.0  16.0  25.0
 36.0  49.0  64.0  81.0 100.0
PRØMEDIØ  5.50
```

c) *Proposición* RETURN *en subprogramas* SUBROUTINE

El retorno normal desde un subprograma FUNCTION o SUBROUTINE al programa llamador se efectúa con la proposición RETURN, sin argumento. En el primer caso, el retorno se realiza a la misma proposición

que realizó el llamado, en el segundo, la vuelta es a la proposición siguiente a la proposición CALL que ejecutó el llamado.

En los subprogramas SUBROUTINE se puede volver a otros puntos del programa llamador, distintos del normal, mediante la estructura siguiente de la proposición RETURN.

i) Estructura de la proposición

RETURN i

donde:

i: es una variable, sin subíndices, sin signo, que señala el punto de retorno.

ii) Función. Produce el retorno al programa llamador, al punto señalado por *i*. Entre los parámetros formales debe haber uno o más asteriscos separados entre sí, o de los otros parámetros formales, por coma. El valor de *i* está dado por:

$1 \leqslant$ valor de $i \leqslant$ número máximo de asteriscos

Cada uno de los asteriscos es reemplazado por un parámetro actual de la forma

&n ó $n

donde:

n : es un número de identificación de proposición
& : se coloca si se utiliza el código EBCDIC
$: se coloca si se utiliza el código BCD.

Si el valor de *i* es *uno*, el retorno se produce a la proposición cuyo número de identificación reemplazó al *primer* asterisco, si el valor de *i* es *dos*, a aquélla cuyo número reemplazó al *segundo* asterisco, etc.

Ejemplo 72:

```
C EJEMPLØ 72.
C RETØRNØ A PUNTØS DISTINTØS DEL NØRMAL
      L = 0
    1 READ (1,50) I
      CALL SALTØ (I,$10,L,$20)
      WRITE(3,51) L
      IF(L.EQ.1) GØ TØ 1
      STØP
   10 WRITE(3,52) L
      GØ TØ 1
   20 WRITE(3,53) L
      STØP
   50 FØRMAT(I2)
   51 FØRMAT(' ',I3,' I<0')
   52 FØRMAT(' ',I3,' I=0')
   53 FØRMAT(' ',I3,' I>0')
```

```
      END
      SUBRØUTINE SALTØ (M,*,N,*)
      N = N + 1
      IF (M) 5,6,7
    5 RETURN
    6 RETURN 1
    7 RETURN 2
      END
```

El problema anterior lee los datos siguientes:

```
-5
 0
13
```

y entrega los resultados que aparecen a continuación:

```
ßß1ßIß<ß0
ßß2ßIß=ß0
ßß3ßIß>ß0
```

E. *Parámetros utilizados en subprogramas*

a) *Parámetros actuales*

Parámetros actuales o reales son aquéllos que se transfieren al subprograma por el programa que lo llama. Reemplazan a los parámetros formales y deben corresponder en orden, número y tipo con éstos.

Los argumentos actuales pueden ser:

1) Cualquier tipo de constante, excepto la constante hexadecimal. Si se trata de una constante literal, debe especificarse de igual manera que en la proposición FORMAT, esto es, entre apóstrofos o precedida por wH; sin embargo, el valor pasado al subprograma corresponde sólo a la cadena de caracteres

2) Cualquier tipo de variable, excepto las definidas por proposiciones ASSIGN

3) Cualquier tipo de nombres de arreglos (véase "Parámetros formales")

4) Cualquier tipo de expresión

5) Nombres de subprogramas (véase "Proposición EXTERNAL")

6) Números de identificación de proposiciones (sólo para subprogramas SUBROUTINE).

Ejemplo 73:

```
C EJEMPLØ 73.
C USØ  DE LITERAL CØMØ PARAMETRØ ACTUAL
       IX = MAT('PRUEBAS')
       WRITE(3,52) IX
   52  FØRMAT(' ',I4)
       STØP
       END
```

```
      FUNCTIØN MAT(B)
      REAL*8 Y,B
      Y = B
      WRITE(3,52) Y
52    FØRMAT(' ',A12)
      MAT = 1
      RETURN
      END
```

El programa anterior entrega los resultados siguientes:

```
bbbPRUEBAS
b1
```

b) *Parámetros formales*

Los parámetros formales se conocen también como parámetros vacíos, flotantes o fantasmas. Son reemplazados por los parámetros actuales y deben corresponder en orden, número y tipo con éstos.

En el caso de nombres de arreglos que figuren como parámetros formales, el parámetro actual debe ser:

i) para arreglos unidimensionales, un arreglo del mismo tipo y de dimensión igual o mayor que el del subprograma.

ii) para arreglos de más de una dimensión, un arreglo del mismo tipo y de dimensiones iguales. Se exceptúan los arreglos que se dimensionan en tiempo objeto.

Ninguno de los parámetros formales puede aparecer en proposiciones COMMON, EQUIVALENCE o NAMELIST.

Para un parámetro formal al cual se le asigna un valor dentro del subprograma, se debe hacer corresponder un parámetro actual que sea variable o arreglo, esto es, no debe utilizarse como parámetro actual una constante o expresión. El ejemplo que figura a continuación muestra lo que puede ocurrir al no cumplir con esta norma.

Ejemplo 74:

El programa que figura a continuación:

```
C EJEMPLØ 74.
C USØ  DE SUBPRØGRAMA SUBRØUTINE
       INTEGER X,Y,Z
       READ (1,51) X,Y,Z
       CALL CAMBIØ(5,6,7)
       X = X * 5
       Y = Y * 6
       Z = Z * 7
       WRITE(3,52) X,Y,Z
       STØP
51     FØRMAT(3I1)
52     FØRMAT(' ',3I4)
       END
```

```
SUBRØUTINE (CAMBIØ(I,J,K)
I =  I + 1
J =  J + 2
K =  K + 3
RETURN
END
```

lee los datos siguientes:

567

Se definen así con la lectura las variables: X=5., Y=6. y Z=7..Dado que las variables X, Y y Z no han sido transferidas a la subrutina y, por lo tanto, no han sufrido cambio, los resultados que se deberían obtener serían:

	X = X*5	o sea	X = 25
	Y = Y*6	o sea	Y = 36
y	Z = Z*7	o sea	Z = 49

Sin embargo, al ser transferidas las *direcciones* de las constantes 5, 6 y 7, reemplazaron a las *direcciones* de I, J y K respectivamente y sus contenidos fueron modificados, esto es, donde debería haber 5 quedó 6, en 6 quedó 8 y en 7 quedó 10. De ahí que el resultado que se obtiene es:

	X = X*6	o sea	X = 30
	Y = X*8	o sea	Y = 48
y	Z = Z*10	o sea	Z = 70

X30Y48Z70

F. *Llamadas de subprogramas*

El tipo de llamada de un subprograma está definido por la forma de especificar los parámetros formales. Se tiene así:

a) *Llamada por valor*

Corresponde a los subprogramas que aparecen en los ejemplos vistos. En ellos los parámetros formales aparecen separados entre sí por coma y todos encerrados entre paréntesis a continuación del nombre del subprograma. En este caso se reserva almacenamiento en el subprograma para los parámetros formales. Cuando el subprograma es llamado, el *valor* del parámetro actual es llevado al almacenamiento reservado en aquél, desde el programa llamador. Cuando termina el proceso del subprograma, el resultado es transferido otra vez al argumento actual en el programa llamador.

b) *Llamada por nombre*

Los parámetros formales deben encerrarse entre operadores de división

y separarse entre sí por coma. En este caso el subprograma no reserva almacenamiento para el parámetro formal. Para realizar los cálculos se utiliza el almacenamiento que corresponde al parámetro actual en el programa llamador.

Ejemplo 75:

```
C  EJEMPLØ 75.
C  LLAMADA PØR NØMBRE
       DIMENSIØN A(5),B(5),C(5)
       READ (1,51) A,B,N,
       CALL XNØM(A,B,C,N,SUM)
       WRITE(3,52) SUM,A,B,C
       STØP
  51   FØRMAT(10F2.0,I2)
  52   FØRMAT(' ',F5.1/(' ',5F5.1))
       END
       SUBRØUTINE XNØM(/X/,/Y/,/Z/,/N/,/S/)
       DIMENSIØN X(N),Y(N),Z(N)
       S = 0.
       DØ 15 I=1,N
       S = S + X(I)*Y(I)
  15   Z(I) = X(I) + Y(I)
       RETURN
       END
```

El programa anterior lee los siguientes datos:

```
1 2 3 4 5 6 7 8 910 5
```

y entrega los resultados que figuran a continuación:

```
130.0
  1.0  2.0  3.0  4.0  5.0
  6.0  7.0  8.0  9.0 10.0
  7.0  9.0 11.0 13.0 15.0
```

G. *Entradas múltiples en subprogramas*

Aparte de la entrada normal a un subprograma SUBROUTINE, efectuada mediante la proposición CALL, y a un subprograma FUNCTION, realizada con la llamada del subprograma en una proposición de asignación, es posible tener múltiples entradas utilizando la proposición ENTRY dentro del subprograma llamado. Cada ENTRY en el subprograma define un punto de entrada distinto del normal. La proposición CALL o la referencia al subprograma FUNCTION utiliza esos puntos así definidos como nombre del subprograma.

y entregan los resultados que figuran a continuación:

```
RB=          2275000.00

RETØRNØ N  1    ISW=0
RB=          2275000.00
RI =           227499.88

RETØRNØ NØRMAL ISW=1
RB=          2275000.00
RI =           227499.88
RL=          2047500.00

RETØRNØ N  2    ISW=2
RB=          2275000.00

RETØRNØ N  1    ISW=0
RB=          2275000.00
RI =           341249.94

RETØRNØ NØRMAL ISW=1
RB=          2275000.00
RI =           341249.94
RL=          1933750.00

RETØRNØ N  2    ISW=2
RB=          2275000.00

RETØRNØ N  1    ISW=0
RB=          2275000.00
RI =           341249.94

RETØRNØ NØRMAL ISW=1
RB=          2275000.00
RI =           341249.94
RL=          1933750.00

RETØRNØ N  2    ISW=2
```

H. *Proposición* EXTERNAL

i) Estructura de la proposición

EXTERNAL $a_1, a_2, a_3, \ldots, a_n$

donde:

los a_i: son nombres de subprogramas que se pasan como parámetros actuales a otro(s) subprograma(s).

ii) Función. Es una proposición de especificación que declara como símbolos externos al programa llamador aquellos nombres de subprogramas que se pasan como parámetros actuales a otro u otros subprogramas.

Debe preceder las definiciones de funciones de proposición y a todas las proposiciones ejecutables.

Ejemplo 77:

Se tienen los siguientes datos:

5 1 3 5 7 9 2 4 6 810

que son procesados por el programa y subprogramas que figuran a continuación:

Ejemplo 77:

```
C EJEMPLØ 77.
C USØ  DE PRØPØSICIØN EXTERNAL
       EXTERNAL AMUL,REST
       DIMENSIØN X(5),Y(5),Z(5)
       READ (1,51) M,X,Y
       N = -1
       CALL SUB(X,Y,Z,N,M,W,AMUL)
       WRITE(3,52)W,X,Y,Z
       CALL SUB(X,Y,Z,N,M,W,REST)
       WRITE(3,52)W,Z
       N = 2
       CALL SUB(X,Y,Z,N,M,W,AMUL)
       WRITE(3,52)W,Z
       CALL SUB(X,Y,Z,N,M,W,REST)
       WRITE(3,52)W,Z
       STØP
  51   FØRMAT(I2,10F2.0)
  52   FØRMAT(' ',F7.2/(5F7.2))
       END
       SUBRØUTINE SUB(A,B,C,/N/,/M/,/X/,RUT)
       DIMENSIØN A(M),B(M),C(M)
       IF(N)1,1,7
   1   X= RUT(A,B,C,M)
   2   IF(X)3,6,4
   3   X = -X
   4   X = SQRT(X)
   5   RETURN
   6   X = 0.
       RETURN
   7   X = RUT(B,A,C,M)
       GØ TØ 2
```

o también a:

```
DIMENSION  X(10)
COMMON     A,B,C,D,X
```

Ejemplo 79:

```
COMMON     I,J/ALFA/X,Y,Z//K,L/ALFA/U,V,W
```

define un área común en blanco que contiene las variables I,J,K y L y un área común de nombre ALFA que contiene las variables X,Y,Z,U,V y W.

Ejemplo 80:

Si se tienen dos variables A y B, y un arreglo C de cinco por cinco elementos, todos de doble precisión; dos variables D y E, de precisión simple; tres variables enteras, I,J,K, de cuatro bytes de longitud; dos variables enteras, M y N, de dos bytes de longitud y una variable lógica L, de un byte de longitud, las especificaciones y orden correctos deben ser:

```
REAL*8     A,B,C(5,5)
INTEGER*2  M,N
LOGICAL*1  L
COMMON A,B,C(5,5),D,E,I,J,K,M,N,L
```

Si no se conserva el orden indicado en cuanto a longitudes, deben especificarse variables artificiales que permitan lograr el alineamiento que corresponde a cada variable.

Ejemplo 81:

Si las variables del ejemplo 80 se especifican en el orden siguiente: D,A,B,I,C(5,5),M,J,K,L,N,E, las proposiciones tendrán que ser:

```
INTEGER*2 AUX3
COMMON  D,AUX1,A,B,I,AUX2,C(5,5),M,AUX3,J,K,L
COMMON  AUX4,N,E
LOGICAL*1 AUX4
```

La primera variable del COMMON parte siempre con el alineamiento de ocho bytes. En este caso, D queda en esas condiciones, pero dado que tiene cuatro bytes, es necesario AUX1 para que queden con alineamiento correcto A y B. En igual forma, como I tiene cuatro bytes, se necesita AUX2 para dar alineamiento correcto a C(5,5). A continuación, como M ocupa dos bytes, se necesita AUX3 (longitud dos bytes) para que queden bien ubicados J y K. Finalmente, como L tiene un byte de longitud, se necesita AUX4 de igual largo para que la variable N quede en forma correcta. Al sacar la cuenta de los bytes ocupados, se observará que la variable E está con el alineamiento adecuado.

Ejemplo 82:

```
C  EJEMPLØ 82.
C  USØ  DE PRØPOSICIØN CØMMØN
       DIMENSIØN Ø(5),P(5)
       CØMMØN A,B/G1/C(5),D(5)//E,M/G1/G
       READ (1,50)N,(C(I),D(I),I=1,N),M,Ø,A,B,E
       L = 1
       G = 0.
       CALL SUB1(N,L)
       WRITE(3,51)C,D,G,N,L
       CALL SUB2(Ø,P)
       WRITE(3,52)A,B,E,M,Ø,P
       STØP
  50   FØRMAT(I2,10F2.0/I2,5F2.0,3F3.1)
  51   FØRMAT(' ',10F4.0,F6.1,2I3/)
  52   FØRMAT(' ',3F5.1,I4/' ',10F5.1)
       END
       SUBRØUTINE SUB1(N,L)
       DIMENSIØN U(5),V(5)
       CØMMØN /G1/U,V,C
       DØ 10 I=L,N
  10   C = C + U(I)*V(I)
       RETURN
       END
       SUBRØUTINE SUB2 (/X/,/Z/)
       DIMENSIØN X(N),Z(N)
       CØMMØN A,B,C,N
       DØ 10 I=1,N
  10   Z(I) = (X(I)+A*B)/C
       RETURN
       END
```

El programa y los subprogramas anteriores procesan los datos siguientes:

```
5 1 2 3 4 5 6 7 8 910
515202530352.54.0100
```

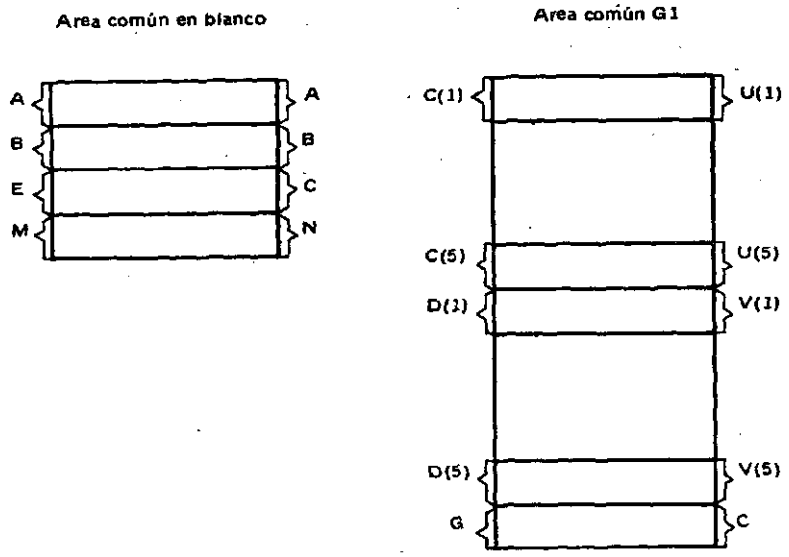
y entregan los resultados que figuran a continuación:

```
ØØ1.ØØ3.ØØ5.ØØ7.ØØ9.ØØ2.ØØ4.ØØ6.ØØ8.Ø10.Ø190.0ØØ5ØØ1

ØØ2.5ØØ4.0Ø10.0ØØØ5
Ø15.0Ø20.0Ø25.0Ø30.0Ø35.0ØØ2.5ØØ3.0ØØ3.5ØØ4.0ØØ4.5
```

En el ejemplo anterior la proposición COMMON del programa principal crea un área común en blanco que contiene las variables A,B,E y M, y un área común de nombre G1 que contiene los arreglos C(5),D(5) y la variable G.

El área común en blanco es compartida con el subprograma SUB2 y el área común G1 con el subprograma SUB1.

El esquema de distribución de dichas áreas, que aparece a continuación, muestra las áreas compartidas y sus respectivos nombres.



b) *Proposición* EQUIVALENCE

i) Estructura de la proposición

EQUIVALENCE $(a_{11},a_{12},a_{13},...),(a_{21},a_{22},a_{23},...),...$

donde:

los a_{ij}: pueden ser variables con o sin subíndices. Estos se pueden expresar considerando el arreglo como unidimensional, en cuyo caso la posición indicada es relativa al primer elemento o considerando las dimensiones reales, esto es, en cada dimensión la posición es relativa al primer elemento de dicha dimensión. En ambos casos los subíndices deben ser constantes enteras sin signo.

ii) Función. Se declaran, encerradas entre paréntesis, todas las variables que comparten memoria dentro del módulo de programa. Todas las variables deben ser del mismo tipo y longitud.

La equivalencia entre dos elementos de arreglos distintos implica la equivalencia de otros elementos de esos mismos arreglos, a causa del orden de almacenamiento preestablecido para ellos.

No pueden hacerse equivalentes variables que estén dentro de un área común o que pertenezcan a diferentes áreas comunes.

Una variable que esté en un área común se puede hacer equivalente a una variable que nó lo esté. Si esta última es un elemento de arreglo, se puede conseguir aumentar el tamaño del área común, que es válido cuando el límite superior del área se desplaza hacia adelante y no cuando el límite inferior o comienzo del área común se desplaza hacia atrás.

Ejemplo 83:

```
DIMENSION    B(5),C(10,10),D(5,10,15)
EQUIVALENCE (A,B(1),C(5,3)),(D(5,10,2),E)
```

La proposición EQUIVALENCE indica que las variables A,B(1) y C(5,3) comparten la misma posición de memoria. A partir de las variables B(1) y C(5,3), el resto de los elementos de los arreglos B y C comparten memoria de acuerdo con su ubicación en el respectivo arreglo. La proposición especifica también que el elemento D(5,10,2) comparte memoria con la variable E.

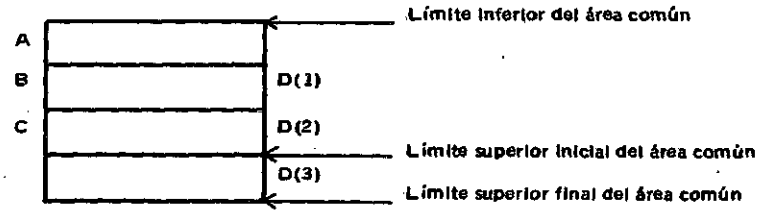
Se obtiene el mismo efecto al especificar:

```
EQUIVALENCE    (A,B(1),C(25)),(D(100),E)
```

Ejemplo 84:

```
COMMON      A,B,C
DIMENSION   D(3)
EQUIVALENCE (B,D(1))
```

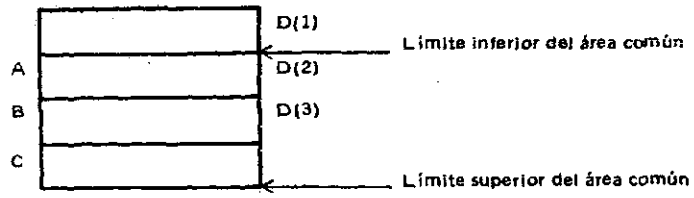
La proposición COMMON genera un área común para las variables A, B y C. La proposición EQUIVALENCE causará que la variable D(1) comparta memoria con B, D(2) con C y D(3) extienda el límite del área común como se indica a continuación:



Si se especifica :

```
EQUIVALENCE (B,D(3))
```

se produciría un error debido a que se fuerza a D(1) a ocupar posiciones anteriores al límite inferior del área común.




```
C  EJEMPLØ 85.
C  USØ  DE PRØPØSICIØN EQUIVALENCE
      DIMENSIØN A(9)
      COMMØN B(3,3)
      EQUIVALENCE (A(6),B(9))
      READ (1,51) B,A
      CALL DIAG(3,3)
      WRITE(3,52)
      WRITE(3,53)
      STØP
 51   FØRMAT(9F2.0)
 52   FØRMAT(' PRØBLEMA DE PRUEBA')
 53   FØRMAT(T10,'MATRIZ RESULTADØ'////9F4.1)
      END
      SUBRØUTINE DIAG(M,N)
      CØMMØN X(3,3)
      DØ 10 I = 1,M
      DØ 10 J = 1,N
      IF(J-I) 20,30,20
 30   X(I,J) = 0.
      GØ TØ 10
 20   X(I,J) = X(J,I)
 10   CØNTINUE
      RETURN
      END
```

El programa y subprograma anteriores procesan los datos siguientes:

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

y entregan los resultados que figuran a continuación:

```
PROBLEMA DE PRUEBA
bbbbbbbbMATRIZ RESULTADØ

0.0bb9.0bb6.0bb9.0bb0.0bb5.0bb6.0bb5.0bb0.0
```

J. *Dimensionamiento en tiempo de ejecución*

Las dimensiones absolutas de arreglos utilizados en subprogramas no necesitan especificarse mediante constantes enteras. Pueden indicarse en proposiciones DIMENSION o en proposiciones de especificación de tipo explícitas, mediante variables enteras de cuatro bytes de longitud.

En el momento de ser llamado el subprograma, las variables enteras que representan dimensiones son reemplazadas por los parámetros reales transferidos al subprograma. La transferencia se realiza a través de parámetros formales o mediante la proposición COMMON.

Deben considerarse cuatro elementos en total para efectuar un dimensionamiento correcto. Estos elementos son:

conjunto actual en el programa llamador
dimensiones absolutas del conjunto actual
conjunto formal en el subprograma
dimensiones actuales transferidas al subprograma.

El nombre del conjunto actual reemplazará al nombre del conjunto formal en el subprograma. Las dimensiones actuales transferidas deben ser iguales a las dimensiones absolutas del conjunto actual, se exceptúa el caso de arreglos unidimensionales,para los cuales pueden ser inferiores o iguales.

El nombre de un arreglo con dimensiones de tiempo de ejecución no puede figurar en una proposición COMMON.

El tamaño de la dimensión variable puede ser transferido a través de más de un nivel de subprogramas.

K. *Problemas propuestos*

a) Escribir una función de proposición para calcular:

i) raíz cúbica de x
ii) sen(2x) = 2 sen x cos x
iii) $D = ((x_1 - x_2)^2 + (y_1 - y_2)^2)^{1/2}$

b) Escribir un subprograma función que calcule la raíz máxima de una ecuación de segundo grado.

c) Hacer una subrutina que calcule el producto de dos matrices A y B de acuerdo con la fórmula general:

C(L,N) = A(L,M)*B(M,N)

El nombre de la subrutina y sus parámetros formales deben ser:

PMAT(A,B,C,L,M,N)

d) Hacer una subrutina que determine el triángulo de área mayor, dada una lista que contiene los lados de N triángulos consecutivos.

e) Hacer un subprograma que calcule los ángulos internos de un triángulo, por el teorema general de Pitágoras ($c^2 = a^2 + b^2 - 2ab(a,b)$).

Se dan las coordenadas de los tres vértices. Los valores de los ángulos deben ser devueltos en grados sexagesimales al programa principal.

f) Hacer un subprograma que ajuste una recta: $y=a+bx$, a una serie de puntos

$$a = \frac{\sum x_i^2 \sum y_i - \sum x_i y_i \sum x_i}{N \sum x_i^2 - (\sum x_i)^2}$$

$$b = \frac{N \sum y_i x_i - \sum y_i \sum x_i}{N \sum x_i^2 - (\sum x_i)^2}$$

g) Hacer una subrutina que integre una función por el método de los trapecios, entre los límites *a* y *b*. El valor de f(x) debe ser calculado con una función de proposición en el programa principal.

$$\text{AREA} = \int_a^b f(x)dx = \frac{b'}{2}(y_0 + 2y_1 + 2y_2 + \ldots + 2y_{n-1} + y_n)$$

$$\text{donde:} \qquad b' = \frac{b-a}{n}$$

h) Hacer un subprograma para calcular:

$$e^x = 1 + \frac{x}{\left(-\frac{x}{2} - \frac{k_0 + k_1 x^2}{1 + k_2 x^2}\right)}$$

para $-\log\sqrt{2} \leqslant x \leqslant \log\sqrt{2}$

$k_0 = 1.0000000020967$
$k_1 = 0.0999743507186$
$k_2 = 0.0166411490538$

i) Escribir un programa que utilice un subprograma SBP1 para calcular

$$h = \sqrt{a^2 - x^2}\ \text{Ln}(b) + \frac{\text{Ln}(c)}{\sqrt{a^2 + x^2}} - \frac{1}{b^2}\ \text{arctg}\frac{x}{a}$$

Todos los datos están perforados en una tarjeta:

a ocupa las columnas 1 a 6 (4 díg. enteros, 2 decimales)
b ocupa las columnas 7 a 12 (4 díg. enteros, 2 decimales)
c ocupa las columnas 21 a 25 (3 díg. enteros, 2 decimales)
x ocupa las columnas 26 a 31 (4 díg. enteros, 2 decimales)

Los resultados deben salir impresos en la siguiente forma:

Primera línea posición 15 RESULTADOS DEL PROGRAMA
Quinta línea posición 10 H = posición 15 el valor de H con F6.3

j) Una tarjeta perforada contiene lo siguiente:

ßxß=ß0.3723ßßßIAß=ßß3ßßßYß=ß0.3117E+01

A KA B

Programar una subrutina que lea una serie de N tarjetas de este tipo y deje los datos en las listas A,KA y B respectivamente. El nombre de la subrutina y sus parámetros formales deben ser:

LECT(A,KA,B,N)

Utilizar esta subrutina para leer N grupos de datos de este tipo, para definir las listas X,IA,Y ($N < 1000$) y formar una nueva lista Z de la siguiente manera:

si $IA_i < 0$ entonces $Z_i = Y_i / X_i^2$

$IA_i = 0$ entonces $Z_i = Y_i + \sqrt{5X_i}$

$IA_i > 0$ entonces $Z_i = \sqrt{\cos(X_i / Y_i)}$

Imprimir pares de IA y Z

k) Programar la transformación de un arreglo bidimensional A(n,n) en un arreglo unidimensional B(m), donde $m=n^2$. Entregar el arreglo unidimensional B a un subprograma que verifica a base de este arreglo, si el original A es simétrico. Si lo es, hacer SIM=0 e imprimir, y si no lo es, hacer SIM=1 e imprimir dicho valor.

l) Hacer una subrutina ELEM (M,N) que elimine elementos repetidos de una lista que se pasa a través de una proposición COMMON.

M = largo de la lista original
N = largo de la lista resultado

m) Se tiene una lista de edades (1000 datos), 40 por tarjeta. Se pide ordenarlos de mayor a menor y formar dos archivos en cinta magnética de acuerdo con el criterio siguiente:

archivo 1 : menores o iguales a 40 años
archivo 2 : mayores de 40 años

El ordenamiento de los datos debe hacerse en un subprograma al cual se le pasarán los datos a través de una proposición COMMON.

n) Analizar el siguiente programa e indicar qué resultados entrega:

```
      DIMENSIØN B(10)
      CØMMØN A(10),TMC
      EQUIVALENCE (A(5),B(5))
      READ (1,51) A
      WRITE(8) A
      END FILE 8
      CALL CUADR(10)
      DØ 10 J = 1,10
   10 B(J) = A(J) - TMC
      WRITE(3,52) A
      REWIND 8
      READ (8) B
      WRITE(3,52)A(K),K=1,10
      STØP
   51 FØRMAT(10F3.0)
   52 FØRMAT('0',10(F3.0,2X))
      END
      SUBRØUTINE CUADR(N)
      CØMMØN C(10),TMC
      REWIND 8
      READ (8) C
      SUM = 0.
      DØ 10 J = 1,N
   10 SUM = SUM + C(I)**2
      TMC = SQRT(SUM)
      RETURN
      END
```

Datos:

1. 4. 4. 5. 2. 1. 2. 1. 7. 2.

6. *Proposiciones para depuración de programas*

Se entiende por depuración de programas la "limpieza" que se hace de ellos para eliminarles todos los errores que se puedan producir durante el proceso y que son difíciles de localizar a simple vista por la complejidad del proceso, cantidad de subprogramas, variedad de datos procesados, etc.

Las proposiciones para depuración forman un paquete que debe ser colocado entre el programa de solución del problema y la proposición END.

A. *Proposición* DEBUG

Debe haber una proposición DEBUG por cada programa o subprograma que va a ser depurado y debe estar inmediatamente antes del paquete de depuración.

i) Estructura de la proposición

DEBUG opción 1, opción 2,...,opción n

donde:

opción i : puede ser cualquiera de las siguientes:

1) UNIT (a)

donde:

a: es una constante entera sin signo, que representa un número de referencia de conjunto de datos. Todos los resultados producidos por el paquete de depuración salen a través del dispositivo asimilado al número *a*.

2) SUBCHK (a_1,a_2,...,a_n)

donde:

los a_i: representan nombres de arreglos.

Al especificar esta opción se verifica la validez de los subíndices usados con los nombres de arreglos indicados. Para ello se utilizan como referencia las dimensiones declaradas para cada arreglo. Si se omite la lista de nombres y sólo se especifica SUBCHK se verifican los subíndices de *todos* los arreglos. Si algún subíndice se sale de rango, se emite un mensaje y el proceso continúa con el subíndice incorrecto.

3) TRACE

Si se especifica, se obtiene el trazado o recorrido que efectúa el proceso dentro de un programa. El trazado queda indicado por la especificación de los números de identificación de las proposiciones que se ejecutan, la cual se obtiene a través del dispositivo señalado en la opción UNIT.

Además de indicar esta opción, debe utilizarse conjuntamente la proposición TRACE ON en el paquete de depuración.

4) IN IT (n_1,n_2,...,n_n)

donde:

los n_i: representan nombres de variables o arreglos.

Especificando esta opción, se emite el nombre de una variable o del elemento de arreglo, y el valor respectivo, cada vez que éste cambia. Si se omite la lista, se entrega el nombre de cualquier variable o elemento de arreglo que cambie de valor.

5) SUBTRACE

Al colocar esta opción en la depuración de un subprograma, se

emite el nombre de éste, cada vez que es llamado, y al salir del subprograma se entrega el mensaje RETURN.

ii) Función. Permite especificar las operaciones de depuración que se ejecutarán en el programa principal o en los subprogramas.

B. *Proposición* AT

i) Estructura de la proposición

AT n

donde:

n: es un número de identificación de proposición ejecutable.

ii) Función. Identifica el comienzo de un paquete de depuración e indica el punto en el programa donde se iniciará dicha depuración.

Las operaciones de depuración especificadas en el paquete son realizadas antes de la ejecución de la proposición identificada con n.

C. *Proposición* TRACE ON

i) Estructura de la proposición

TRACE ON

ii) Función. Permite obtener el trazado o recorrido que efectúa el proceso, dentro de un programa. Debe especificarse, sin embargo, la opción TRACE en la proposición DEBUG.

D. *Proposición* TRACE OFF

i) Estructura de la proposición

TRACE OFF

ii) Función. Pone término al trazado iniciado con la proposición TRACE ON. El punto donde se termina el trazado se especifica con la proposición AT ubicada inmediatamente antes de la proposición TRACE OFF.

E. *Proposición* DISPLAY

i) Estructura de la proposición

DISPLAY lista

donde:

lista: es un conjunto de nombres de variables y arreglos, separados entre sí por coma.

ii) Función. Se emite el nombre de la variable y su valor. Si se

trata de arreglos, se entrega el nombre del arreglo y los valores de sus elementos.

El efecto producido por una proposición DISPLAY lista, es el mismo que se obtiene con:

NAMELIST /nombre/lista
WRITE (n,nombre)

No se pueden emitir valores de variables o arreglos que figuren como parámetros formales.

Ejemplo 86:

```
      C EJEMPLØ 86.
      C USØ DE PRØPØSICIØNES DE DEPURACIØN
            DIMENSIØN A(10),B(10)
            N = 10
            CALL LEA (A,N)
            CALL CALC(A,B,N,P)
            CALL IMP (A,B,N,P)
            STØP
            END
            SUBRØUTINE LEA (X,M)
            DIMENSIØN X(M)
            READ (1,51) X
            RETURN
     51     FØRMAT(10F3.1)
            DEBUG UNIT(3),SUBTRACE
            END
            SUBRØUTINE CALC(X,Y,M,PR)
            DIMENSIØN X(M),Y(M)
      1     PR = PRØM (X,M)
            CALL ØRDEN(X,M)
            DØ 10 I=1,M
     10     Y(I) = X(I)*X(I)
      2     RETURN
            DEBUG UNIT(3),SUBTRACE
            AT 1
            TRACE ØN
            AT 2
            TRACE ØFF
            END
            FUNCTIØN PRØM (X,M)
            DIMENSIØN X(M)
     21     PRØM = 0.
            DØ 10 I=1,M
     10     PRØM = PRØM + X(I)
            PRØM = PRØM/M
     22     RETURN
            DEBUG UNIT(3),SUBTRACE,TRACE,INIT(I,PRØM)
            AT 21
            TRACE ØN
```

```
      AT 22
      TRACE ØFF
      END
      SUBRØUTINE ØRDEN(Z,N)
      DIMENSIØN Z(N)
31    M = N - 1
      DØ 15 I=1,M
      K = I + 1
      DØ 10 J=K,N
      IF(Z(I).LE.Z(J)) GØ TØ 10
      AUX= Z(I)
      Z(I) = Z(J)
      Z(J) = AUX
10    CØNTINUE
15    CØNTINUE
32    RETURN
      DEBUG UNIT(3),SUBTRACE,TRACE,INIT(Z)
      AT 31
      TRACE ØN
      DISPLAY AUX
      AT 32
      TRACE ØFF
      END
      SUBRØUTINE IMP (X,Y,M,PR)
      DIMENSIØN X(M),Y(M)
41    WRITE (3,52)
      WRITE (3,53)X
      WRITE (3,54)
      WRITE (3,53)Y
      WRITE (3,55)PR
      RETURN
52    FØRMAT(' MATRIZ DATØ'//)
53    FØRMAT(' ',5F5.1)
54    FØRMAT(' MATRIZ RESULTADØ'//)
55    FØRMAT(' PRØMEDIØ',F7.2)
      DEBUG UNIT(3),SUBTRACE
      AT 41
      DIMENSIØN W2(10),W3(10)
      W4 = PR
      DØ 200 IW2=1,10
      W2(IW2) = X(IW2)
200   W3(IW2) = Y(IW2)
      DISPLAY W2,W3,W4
      END
```

El programa y subprogramas anteriores leen los datos siguientes:

7 5 3 9 4 1 8 2 6 10

y entregan los resultados que figuran a continuación. Por razones de

espacio, éstos se han agrupado en dos columnas sobre las cuales se especifica un número que indica el orden en que deben ser leídas.

(1)

```
SUBTRACE LEA
SUBTRACE *RETURN*
SUBTRACE CALC
SUBTRACE PRØM
TRACE 21
PRØM = 0.0
I = 1
TRACE 10
PRØM =  7.000000
I = 2
TRACE 10
PRØM = 12.00000
I = 3
TRACE 10
PRØM = 15.00000
I = 4
TRACE 10
PRØM = 24.00000
I = 5
TRACE 10
PRØM = 28.00000
I = 6
TRACE 10
PRØM = 29.00000
I = 7
TRACE 10
PRØM = 37.00000
I = 8
TRACE 10
PRØM = 39.00000
I = 9
TRACE 10
PRØM = 45.00000
I = 10
TRACE 10
PRØM = 55.00000
PRØM = 5.500000
SUBTRACE *RETURN*
SUBTRACE ØRDEN
&DBG00#
AUX=-0.17296600
&END
TRACE 31
Z(1) =   5.000000
```

(2)

```
Z(2) = 7.000000
TRACE 10
Z(1) = 3.000000
Z(3) = 5.000000
TRACE 10
TRACE 10
TRACE 10
Z(1) = 1.000000
Z(6) = 3.000000
TRACE 10
TRACE 10
TRACE 10
TRACE 10
TRACE 10
TRACE 15
Z(2) = 5.000000
Z(3) = 7.000000
TRACE 10
TRACE 10
Z(2) = 4.000000
Z(5) = 5.000000
TRACE 10
Z(2) = 3.000000
Z(6) = 4.000000
TRACE 10
TRACE 10
Z(2) = 2.000000
Z(8) = 3.000000
TRACE 10
TRACE 10
TRACE 10
TRACE 15
TRACE 10
Z(3) = 5.000000
Z(5) = 7.000000
TRACE 10
TRACE 10
TRACE 10
TRACE 15
Z(4) = 7.000000
Z(5) = 9.000000
TRACE 10
Z(4) = 5.000000
Z(6) = 7.000000
```

(3)

```
TRACE 10
TRACE 10
Z(4) = 4.000000
Z(8) = 5.000000
TRACE 10
TRACE 10
TRACE 10
TRACE 15
Z(5) = 7.000000
Z(6) = 9.000000
TRACE 10
TRACE 10
Z(5) = 5.000000
Z(8) = 7.000000
TRACE 10
TRACE 10
TRACE 10
TRACE 15
Z(6) = 8.000000
Z(7) = 9.000000
TRACE 10
Z(6) = 7.000000
Z(8) = 8.000000
TRACE 10
Z(6) = 6.000000
TRACE 10
Z(3) = 4.000000
Z(6) = 5.000000
TRACE 10
TRACE 10
Z(3) = 3.000000
Z(8) = 4.000000
Z(9) = 7.000000
TRACE 10
TRACE 10
TRACE 15
Z(7) = 8.000000
Z(8) = 9.000000
TRACE 10
Z(7) = 7.000000
Z(9) = 8.000000
TRACE 10
TRACE 10
TRACE 15
Z(8) = 8.000000
Z(9) = 9.000000
TRACE 10
```

(4)

```
TRACE 10
TRACE 15
TRACE 10
TRACE 15
SUBTRACE *RETURN*
SUBTRACE *RETURN*
SUBTRACE IMP
&DBG00#
W2= 1.0000000  ,  2.0000000
   3.0000000  ,  4.0000000  ,
   5.0000000  ,  6.0000000  ,
   7.0000000  ,  8.0000000  ,
   9.0000000  ,  10.000000  ,W3=
   1.0000000  ,  4.0000000  ,
   9.0000000  ,  16.000000  ,
   25.000000  ,  36.000000  ,
   49.000000  ,  64.000000  ,
   81.000000  ,  100.00000  ,W4=
   5.5000000
&END
MATRIZ DATØ

   1.0   2.0   3.0   4.0  5.0
   6.0   7.0   8.0   9.0 10.0

MATRIZ RESULTADØ

   1.0   4.0   9.0  16.0 25.0
  36.0  49.0  64.0  81.0100.0

PROMEDIØ  5.50

SUBTRACE*RETURN*
```

APENDICE A

FUNCIONES ESTANDAR

Función-general						
Nombre	Definición	Argumento(s)			Resultado	
		Nº	Tipo	Rango	Tipo	Rango
Logaritmo natural						
ALØG DLØG	$y = \log_e x$ ó $y = \ln x$	1 1	REAL*4 REAL*8	$x > 0$	REAL*4 REAL*8	$a \leq y \leq b$
Logaritmo decimal						
ALØG10 DLØG10	$y = \log_{10} x$ ó $y = \log x$	1 1	REAL*4 REAL*8	$x > 0$	REAL*4 REAL*8	$c \leq y \leq d$
Exponencial						
EXP DEXP	$y = e^x$	1 1	REAL*4 REAL*8	$a \leq x \leq b$	REAL*4 REAL*8	$0 \leq y \leq e^b$
Raíz cuadrada						
SQRT DSQRT	$y = \sqrt{x}$ ó $y = x^{\frac{1}{2}}$	1 1	REAL*4 REAL*8	$x \geq 0$	REAL*4 REAL*8	$0 \leq y \leq e^{\frac{b}{2}}$
Arco seno						
ARSIN DARSIN	$y = \arcsen x$	1 1	REAL*4 REAL*8	$\lvert x \rvert \leq 1$	REAL*4[-] REAL*8[-]	$-\frac{\pi}{2} \leq y \leq \frac{\pi}{2}$
Arco coseno						
ARCØS DARCØS	$y = \arccos x$	1 1	REAL*4 REAL*8	$\lvert x \rvert \leq 1$	REAL*4[-] REAL*8[-]	$0 \leq y \leq \pi$
Arco tangente						
ATAN DATAN ATAN2 DATAN2	$y = \arctg x$ $y = \arctg \frac{x1}{x2}$	1 1 2 2	REAL*4 REAL*8 REAL*4 REAL*8	Cualq. real id. excepto (0,0)	REAL*4[-] REAL*8[-] REAL*4[-] REAL*8[-]	$-\frac{\pi}{2} \leq y \leq \frac{\pi}{2}$ $-\pi \leq y \leq \pi$
Seno						
SIN DSIN	$y = \sen x$	1 1	REAL*4[-] REAL*8[-]	$\lvert x \rvert < 2^{18} \cdot \pi$ $\lvert x \rvert < 2^{50} \cdot \pi$	REAL*4 REAL*8	$-1 \leq y \leq 1$

FUNCIONES ESTANDAR (Continuación)

Función general						
Nombre	Definición	Argumento(s)			Resultado	
		Nº	Tipo	Rango	Tipo	Rango
Coseno						
COS DCOS	$y = \cos x$	1 1	REAL*4[–] REAL*8[–]	$\|x\| < 2^{20} \cdot \pi$ $\|x\| < 2^{50} \cdot \pi$	REAL*4 REAL*8	$-1 \leq y \leq 1$
Tangente						
TAN DTAN	$y = \text{tg}x$	1 1	REAL*4[–] REAL*8[–]	$\|x\| < 2^{18} \cdot \pi$ $\|x\| < 2^{50} \cdot \pi$	REAL*4 REAL*8	$-e \leq y \leq e$
Cotangente						
COTAN DCOTAN	$y = \text{cotg}x$	1 1	REAL*4[–] REAL*8[–]	$\|x\| < 2^{18} \cdot \pi$ $\|x\| < 2^{50} \cdot \pi$	REAL*4 REAL*8	$-e \leq y \leq e$
Seno hiperbólico						
SINH DSINH	$y = \frac{e^x - e^{-x}}{2}$	1 1	REAL*4 REAL*8	$\|x\| < 175.366$	REAL*4 REAL*8	$-e \leq y \leq e$
Coseno hiperbólico						
COSH DCOSH	$y = \frac{e^x + e^{-x}}{2}$	1 1	REAL*4 REAL*8	$\|x\| < 175.366$	REAL*4 REAL*8	$1 \leq y \leq e$
Tangente hiperbólica						
TANH DTANH	$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	1 1	REAL*4 REAL*8	Cualq. real	REAL*4 REAL*8	$-1 \leq y \leq 1$
Valor absoluto						
IABS ABS DABS	$y = \|x\|$	1 1 1	INTEGER*4 REAL*4 REAL*8	entero Cualq. real	INTEGER*4 REAL*4 REAL*8	

FUNCIONES ESTANDAR (Continuación)

Función general						
Nombre	Definición	Argumento (s)			Resultado	
		Nº	Tipo	Rango	Tipo	Rango
Función error						
ERF DERF	$y = \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du$	1 1	REAL*4 REAL*8	Cualq. real	REAL*4 REAL*8	$-1 \leq y \leq 1$
ERFC DERFC	$y = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-u^2} du$	1 1	REAL*4 REAL*8	Cualq. real	REAL*4 REAL*8	$0 \leq y \leq 2$
Función gama y logaritmo de gama						
GAMMA DGAMMA	$y = \int_0^\infty u^{x-1} e^{-u} du$	1 1	REAL*4 REAL*8	$2^{-252} < x < 1$	REAL*4 REAL*8	$0 \leq y \leq 8$
ALGAMA DLGAMA	$y = \log_e \int_0^\infty u^{x-1} e^{-u} du$	1 1	REAL*4 REAL*8	$0 < x < h$	REAL*4 REAL*8	$i \leq y \leq e$
Valores máximo y mínimo						
MAX0 AMAX0 MAX1 AMAX1 DMAX	$y = \max(x_1, \ldots, x_n)$	≥2 ≥2 ≥2 ≥2 ≥2	INTEGER*4 INTEGER*4 REAL*4 REAL*4 REAL*8	Cualq. entero Cualq. real	INTEGER*4 REAL*4 INTEGER*4 REAL*4 REAL*8	
MIN0 AMIN0 MIN1 AMIN1 DMIN1	$y = \min(x_1, \ldots, x_n)$	≥2 ≥2 ≥2 ≥2 ≥2	INTEGER*4 INTEGER*4 REAL*4 REAL*4 REAL*8	Cualq. entero Cualq. real		
Truncación						
AINT DINT INT IDINT	y=(signo de x)* n donde n es el mayor entero ≤ |x|	1 1 1 1	REAL*4 REAL*8 REAL*4 REAL*8	Cualq. real	REAL*4 REAL*8 INTEGER*4 INTEGER*4	

FUNCIONES ESTANDAR (Continuación)

Función general						
Nombre	Definición	Argumento(s)			Resultado	
		Nº	Tipo	Rango	Tipo	Rango
Conversión a punto flotante						
FLOAT DFLOAT	Convierte de entero a real	1 1	INTEGER*4 INTEGER*4	Cualq. entero	REAL*4 REAL*8	
Conversión a punto fijo						
IFIX HFIX	Convierte de real a entero	1 1	REAL*4 REAL*4	Cualq. real	INTEGER*4 INTEGER*2	
Transferencia de signo						
ISIGN SIGN DSIGN	$y = (\text{signo } x_2) * x_1$ $x_1 \neq 0$	2 2 2	INTEGER*4 REAL*4 REAL*8	entero cualq. real	INTEGER*4 REAL*4 REAL*8	
Diferencia positiva						
IDIM DIM DDIM	$y = X_1 - min (X_1, X_2)$	2 2 2	INTEGER*4 REAL*4 REAL*8	entero cualq. real	INTEGER*4 REAL*4 REAL*8	
Parte más significativa de un argumento real						
SNGL		1	REAL*8	real	REAL*4	

$a = -180.218$ $b = 174.673$ $c = -78.268$ $d = 75.859$

$e = 16^{63} * (1-16^{-6})$ para REAL*4 y $16^{63} * (1-16^{-14})$ para REAL*8

$f = 257.5744$ $g = 0.88560$ $h = 4.2913 * 10^{73}$

$i = -0.12149$

APENDICE B

PROPOSICIONES Y CARACTERISTICAS DE FORTRAN IV COMPLETO QUE NO SON ACEPTADAS POR FORTRAN IV BASICO

ASSIGN
Arreglos con códigos de formato
*BLOCK DATA**
COMPLEX
Constantes *complejas*, lógicas, literales y hexadecimales
Códigos de formato L, G y Z
COMMON con nombre
DATA
Dimensionamiento en tiempo de ejecución
ENTRY
Especificación de longitud en proposiciones de especificación de tipo
Facilidades de depuración
GO TO asignado
IMPLICIT
Inicialización de variables en proposiciones de especificación explícitas
IF lógico
Llamada por nombre
Literales como parámetros actuales
LOGICAL
Más de tres dimensiones en un arreglo
NAMELIST
Parámetros ERR y END en la proposición READ
PAUSE con literal
PRINT b, lista
PUNCH b, lista
READ b, lista
RETURN *i* con *i* distinto de blanco
Subíndices generalizados

* Las proposiciones y características destacadas en letra cursiva no han sido tratadas en este manual.

BIBLIOGRAFIA

1. Anderson, Decima M., *Computer Programming Fortran IV*, Appleton–Century–Crofts/Meredith Corporation, Estados Unidos, 1966, 435 págs.

2. Dimitry, Donald y Mott, Thomas Jr., *Introduction to Fortran IV Programming*, Holt Rinehart and Winston, Inc., Estados Unidos, 1966, 334 págs.

3. Hull, F.E. y Day, D.D., *Computer and Problem Solving*, Addison–Wesley, Estados Unidos, 1970, 276 págs.

4. IBM System Products Division, *IBM System/360 and System/370 Fortran IV Language*, Nueva York, 1971, 159 págs.

5. IBM System Products Division, *IBM System/360 Disk Operating System Fortran IV Programmer's Guide*, Nueva York, 1968, 96 págs.

6. IBM System Products Division, *IBM System/360 Operating System Fortran IV (G and H) Programmer's Guide*, Nueva York, 1968.

7. Nydegger, Adolph C., *An Introduction to Computer Programming with an Emphasis on Fortran IV*, Addison–Wesley, Estados Unidos, 1968, 268 págs.

8. Sánchez, C., Víctor, *Apuntes de Fortran*, Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, 1967, 77 págs.

LENGUAJE DE ENSAMBLE

I. INTRODUCCION

1. *Lenguaje de máquina y lenguaje de ensamble*

Para programar en el lenguaje de máquina de un computador es necesario conocer: características internas del computador como: capacidad de memoria, representación de los datos, sistema de direccionamiento, etc., formato de las instrucciones, significado y uso de cada uno de los operandos, códigos de error, etc. Todo lo anterior hace que dicha programación sea bastante lenta, con muchas posibilidades de error y difícil detección de los mismos.

A pesar de las desventajas enunciadas, se verá a continuación una máquina simplificada en la que se hará uso del lenguaje de máquina o lenguaje absoluto, dado que su utilización permite comprender de modo más claro lo que ocurre internamente cuando se está programando en un lenguaje de alto nivel, al mismo tiempo que sirve como introducción a los lenguajes de ensamble que constituyen el nivel de lenguaje inmediatamente superior al de máquina.

Respecto a los lenguajes de ensamble, si bien es cierto que presentan en programación un grado de dificultad menor que el lenguaje de máquina, distan bastante de ser la solución ideal para el programador, sin embargo, en toda instalación de computador o grupo de procesamiento de datos es necesario que hayan uno o dos especialistas en este tipo de lenguajes, pues cualquiera que sea el lenguaje de alto nivel que se utilice, en algún momento será necesario entrar a analizar en forma más detallada lo que ha ocurrido internamente en un proceso para ver qué causas han originado detenciones anormales del mismo.

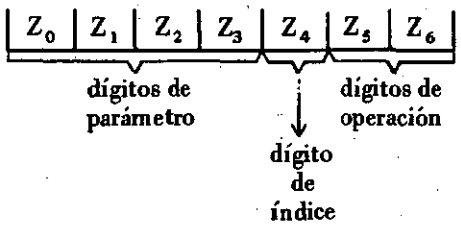
A. *Una máquina simplificada*

La máquina que se utilizará será un computador de segunda generación, el ER-56 de la Standard Elektrik Lorenz. Para el objetivo que se persigue se podría haber inventado un computador con un cierto número de instrucciones que diera la posibilidad de ejemplificar las características principales del funcionamiento o forma de operación de un computador al realizar un proceso. Se ha elegido el ER-56, pues es un computador muy simple y muy fácil de programar, aun en lenguaje de máquina.

a) *Características principales del computador*

Memoria de trabajo:	3000 palabras (celdas)
Palabra:	7 dígitos decimales
Dirección de cada palabra:	1000 a 3999
Unidad aritmética	
Acumulador:	14 dígitos decimales

Aritmética de punto fijo y de punto flotante
Registros: 10 (capacidad para 4 dígitos decimales)
Instrucción: 7 dígitos decimales (una palabra)
Formato:



Los dígitos de parámetro indican:

i) La dirección de la celda de memoria en la que está contenido el dato, o la dirección donde se almacenará un dato ($Z_0 \ldots Z_3 = n$).

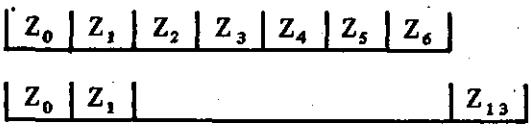
ii) Cualquier variable desde 0000 hasta 9999 que actuará como uno de los operandos en una operación ($Z_0 \ldots Z_3 = p$).

El dígito de índice representa uno de los diez registros de índice ($j_i = 0, \ldots, 9$) que tiene el computador.

Los dígitos de operación o código de operación indican la función que se va a realizar.

b) *Representación de la información*

i) Punto fijo (entera)



Z_0 = Signo (1 = +, 2 = −)
$Z_1 \ldots Z_6$ = Mantisa, palabra simple
$Z_1 \ldots Z_{13}$ = Mantisa, palabra doble.

Desde el punto de vista aritmético, el computador interpreta el número como menor que uno, presumiendo la coma decimal ubicada siempre inmediatamente después del dígito Z_0. Si se utiliza doble palabra, se trabaja con dos celdas contiguas.

ii) Punto flotante (real)

Z_0 = signo
$Z_1 \ldots Z_{11}$ = Mantisa
$Z_{12}\, Z_{13}$ = Característica

Para obtener la característica se normaliza el dato, esto es, se transforma en el producto de una fracción decimal, en que los dígitos significativos están inmediatamente a continuación de la coma decimal, por una potencia de 10. El exponente de 10 se suma a la constante 50 y el resultado es la característica.

Ejemplo 1.

Dato original		Dato normalizado	Representación interna
17,385	=	$0{,}17385*10^{2}$	11738500000052
0,017385	=	$0{,}17385*10^{-1}$	11738500000049
-17385,0	=	$-0{,}17385*10^{5}$	21738500000055

c) *El lenguaje de máquina (absoluto)*

Primer conjunto de instrucciones:

7200000 indica que se trabajará en punto flotante
7900000 pone fin al proceso

Son dos instrucciones que difieren del formato general.

Código de operación	Función
31	Llevar al acumulador desde la celda *n*
32	Llevar desde el acumulador a la celda *n*
35	Sumar al acumulador contenido de celda *n*
36	Restar del acumulador contenido de celda *n*
37	Multiplicar el acumulador por contenido de celda *n*.
38	Dividir el acumulador por contenido de celda *n*

Ejemplo 2.

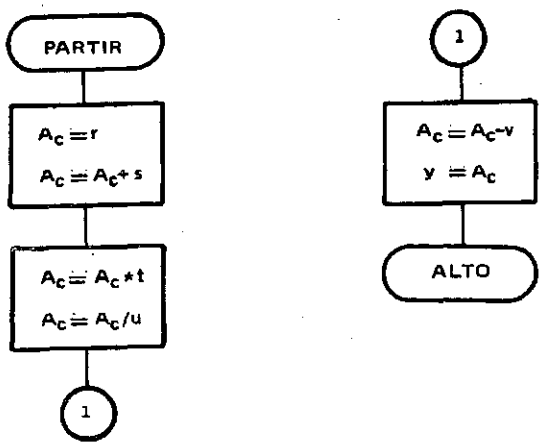
Programar el cálculo de:

$$y = \frac{(r+s)t}{u} - v$$

Para simplificar la descripción se utilizará la notación siguiente:

Ac: significa acumulador
( ): significa "contenido de"
el signo = significa "definido por"

El diagrama de flujo se hará orientado al ER-56.



Para codificar el programa se supondrá la siguiente distribución de memoria:

(1500) = r
(1502) = s
(1504) = t
(1506) = u
(1508) = v
(1510) = y

El programa se cargará en memoria a partir de la celda 3000.
Programa codificado:

(3000) = 7200000	Partir, modo punto flotante		
(3001) = 1500031	A_c=(1500)	;	A_c=r
(3002) = 1502035	A_c=A_c+(1502)	;	A_c=A_c+s
(3003) = 1504037	A_c=A_c*(1504)	;	A_c=A_c*t
(3004) = 1506038	A_c=A_c/(1506)	;	A_c=A_c/u
(3005) = 1508036	A_c=A_c-(1508)	;	A_c=A_c-v
(3006) = 1510032	(1510)=A_c	;	y =A_c
(3007) = 7900000	ALTO		
(3008) = 9999999	ω		
(3009) = 9999999	ω		

Observación: El dígito 0, en el lugar que corresponde a dígito de índice, no tiene ningún efecto.

El programa y los datos se perforan en cinta de papel. Para indicar el término de uno u otro se perfora al final de ellos la doble palabra omega, la que junto con ser almacenada pone fin a la lectura. Se utiliza también en salida para poner término a dicha operación.

Los códigos correspondientes a esas funciones son:

67 Leer y almacenar en memoria a partir de la celda *n*
68 Perforar desde memoria el contenido de la celda *n*
69 Perforar desde memoria a partir de la celda *n*

Ejemplo 3.

Programar el cálculo de

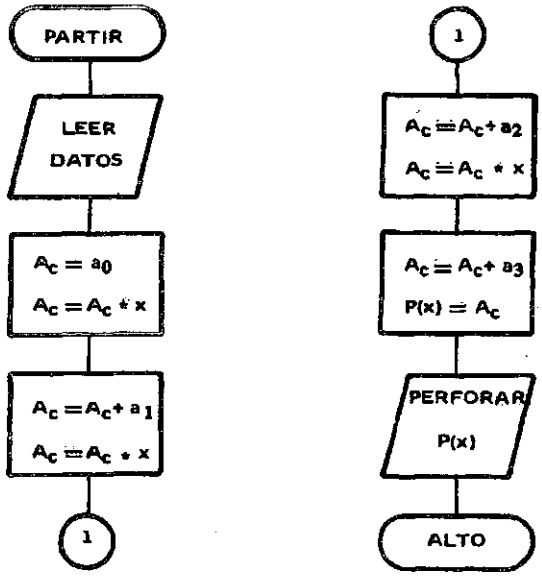
$$P(x) = a_0x^3 + a_1x^2 + a_2x + a_3$$

o lo que es lo mismo $P(x) = ((a_0x+a_1)x+a_2)x+a_3$

Distribución de memoria:

(1000) = x
(1002) = a_0
(1004) = a_1
(1006) = a_2
(1008) = a_3
(1010) = P(x)

El programa se cargará en memoria a partir de la celda 3000.



(3008)	3004016	Salto a la dirección 3004 si el indicador < está conectado	
(3009)	1010032	(1010) = Ac ;	P(x) = Ac
(3010)	1010068	Perforar (1010)	
(3011)	1011068	Perforar (1011)	
(3012)	7900000	Alto	
(3013)	9999999	ω	
(3014)	9999999	ω	

Aun cuando el programa tiene igual número de instrucciones que en la solución anterior, puede ahora servir para calcular el valor de un polinomio de cualquier grado; para ello lo único que se necesita cambiar es el valor de comparación en la instrucción almacenada en la dirección 3007.

Queda todavía por resolver una situación incómoda y es la de tener que cargar siempre el programa a partir de la celda 3000 (programación absoluta). Si no se hiciera así, al llegar a la instrucción de bifurcación, se producirá el salto a la dirección 3004, que puede contener cualquier cosa que no tenga ninguna relación con el proceso.

Lo contrario a la "programación absoluta" es la "programación relativa", que es aquella que permite cargar los programas en cualquier parte de la memoria sin que se produzca ningún problema. Se hace uso en este caso de un registro especial, el REGISTRO DE INDICE 9 que contiene siempre la dirección de la instrucción que se ejecuta, más 1. Para referirse a direcciones que están más adelante en el programa, se utiliza la fórmula:

$$p = n_2 \quad n_1 - 1$$

donde:

p = parámetro
n_1 = dirección de la instrucción origen
n_2 = dirección de la instrucción meta de salto

Ejemplo 4.

Se tiene en la dirección 1022 una instrucción de salto a la dirección 1025.

n_1 (1022) 1025 0 16
n_2 (1025) ---- - --
$p = 1025 - 1022 - 1$
$p = 0002$

luego, la instrucción se puede representar como:

(1022) 0002 9 16

dirección efectiva = (REG9) + 0002
dirección efectiva = 1023 + 0002
dirección efectiva = 1025

Para referirse a direcciones que están más atrás en el programa se utiliza la fórmula:

$$p = 9999 - (n_1 - n_2)$$

Ejemplo 5.

Se tiene en la dirección 3008 una instrucción de salto a la dirección 3004

n_2 (3004) ---- - --
---- ---- - --
---- ---- - --
(3008) 3004 0 16

p = 9999 – (3008–3004)
p = 9999 – 4
p = 9995

luego, la instrucción se puede representar como:

(3008) 9995 9 16
dirección efectiva =(REG9) + 9995
dirección efectiva = 3009 + 9995
dirección efectiva = 13004

El primer dígito se pierde, pues el registro tiene capacidad para cuatro dígitos solamente; queda entonces la dirección efectiva igual a 3004 que es lo que se estaba buscando.

II. UN ENSAMBLADOR

1. *Definición*

Los ensambladores son los traductores de los lenguajes orientados a la máquina. Se denominan también lenguajes uno a uno, porque existe una correspondencia, instrucción por instrucción, entre éstos y los lenguajes de máquina. Otro término, poco utilizado, para referirse a los ensambladores es el de autocódigo.

La ventaja que tienen los lenguajes orientados a la máquina es la de poder codificar las instrucciones a base de iniciales de palabras, abreviaturas o signos que indican la función a realizar y que son fáciles de recordar por el programador. Además, éste se despreocupa del lugar o dirección donde se almacenarán los datos y programa, pues trabaja con direcciones simbólicas. Para esto existen pseudo-instrucciones que permiten definir zonas de datos, constantes, etc., de tal manera que el ensamblador se preocupa de traducir las expresiones nemotécnicas y substituir las direcciones simbólicas por direcciones reales.

Para poder hacer la substitución de direcciones simbólicas por reales, normalmente el ensamblador necesita de dos pasadas de análisis del programa fuente. En la primera pasada asigna direcciones reales a cada

dirección simbólica definida en el programa. Estas direcciones reales las asigna tomando como punto de referencia una dirección que le ha sido entregada a través de una pseudo-instrucción o de constantes propias del ensamblador. Utiliza además, un contador de direcciones cuyo contenido inicial es la dirección de referencia. Este contenido será incrementado de acuerdo al espacio de memoria ocupado por cada instrucción, constante o área de resultados generada, de tal manera que siempre el contador de direcciones contendrá la próxima ubicación disponible. Se efectúa al mismo tiempo la construcción de una tabla de símbolos que contiene los nombres simbólicos creados por el programador y las direcciones reales que le ha asignado el ensamblador. Además se realiza la revisión sintáctica de las instrucciones y en algunos casos la traducción inmediata de los códigos de operación, literales incluidos en la instrucción y todos aquellos elementos que no participan en la descripción de los operandos simbólicos.

En la segunda pasada, el ensamblador traduce los operandos simbólicos de acuerdo a la tabla de símbolos construida en la primera pasada y se hace un análisis de error global, esto es, se determinan errores que involucren la interrelación de proposiciones.

Existen casos particulares en que la tabla de símbolos puede suprimirse y corresponden a problemas que tratan los ensambladores de una pasada. Este tipo de traductores se necesitan en los sistemas de tiempo compartido (time-sharing), en los cuales varios usuarios hacen uso simultáneo del computador desde consolas individuales. La restricción que existe para que puedan funcionar los ensambladores de una pasada es que las áreas de datos y de almacenamiento deben estar definidas antes de que se haga referencia a ellas. Subsiste, sin embargo, el problema de las bifurcaciones, el cual se resuelve transfiriendo el programa generado con las referencias simbólicas sin traducir, al programa cargador o a un sistema intérprete que termine la traducción en el momento de ejecutar el programa objeto.

Hay algunos ensambladores que reconocen y traducen macroinstrucciones, es decir, instrucciones simbólicas que provocan la inclusión, en el programa generado, de subprogramas escritos en lenguaje orientado a la máquina o subprogramas que han tenido la primera pasada del ensamblador. Este hecho permite la traducción conjunta del programa principal (monitor) y de los subprogramas (subrutinas). Se realiza así el ENSAMBLADO de programas que tienen origen distinto, y esta función es la que le ha dado el nombre a este tipo de traductores.

2. *El lenguaje de ensamble del Sistema IBM/360/370*

Llamado comúnmente ASSEMBLER, está formado por un conjunto de símbolos nemotécnicos que representan:

a) Códigos de operación del lenguaje de máquina

b) Operaciones que van a ser realizadas por el ensamblador.

El programador puede crear también macroinstrucciones.

A. *Hoja de codificación*

La hoja de codificación tiene líneas con capacidad para ochenta caracteres, lo que significa que cada línea puede ser vaciada en su totalidad en una tarjeta.

Cada línea queda dividida en dos sectores: el que corresponde a la proposición (columnas 1 a la 71) y el que corresponde a identificación o secuencia (columnas 73 a 80). Si la proposición ocupa más de 71 posiciones, se puede continuar en la línea siguiente. Para ello se especifica cualquier carácter distinto de blanco en la columna 72 y se empieza en la columna 16 de la línea siguiente. Se acepta sólo una línea de continuación y las primeras quince columnas de ella deben estar en blanco.

a) *Proposición*

Hay dos tipos de proposiciones y son: las instrucciones y los comentarios. Las primeras pueden consistir de uno a cuatro campos que son de izquierda a derecha:

NOMBRE
OPERACION
OPERANDOS
COMENTARIO

i) En el campo de NOMBRE puede figurar un símbolo que se utiliza para identificar la instrucción y debe cumplir las siguientes normas:

Debe estar compuesto de 8 caracteres o menos
Debe empezar con carácter alfabético
Debe empezar en columna 1
Debe aparecer una sola vez identificando a una instrucción
No debe contener caracteres especiales ni blancos.

ii) En el campo OPERACION figura un código de operación que indica la función que se va a realizar.

Los códigos de operación válidos tienen cinco caracteres o menos
Si no existe nombre, debe empezar, al menos, una posición a la derecha de la columna 1
Si existe nombre, debe ir separado de éste, al menos, por un blanco
No debe contener caracteres especiales ni blancos.

iii) En el campo OPERANDOS figuran códigos que suplementan la operación a realizarse. Pueden aparecer registros, longitudes, símbolos, etc.

Si hay más de un operando, deben separarse entre sí por coma

Debe separarse del código de operación, al menos, por un blanco

No deben haber blancos entre caracteres ni entre éstos y las comas.

iv) En el campo COMENTARIO figura información descriptiva de lo que realiza la instrucción o un conjunto de ellas.

Debe ir separada de los operandos, al menos, por un blanco

Puede aparecer cualquier carácter como comentario

Si no hay operandos, antes del comentario debe figurar una coma precedida y seguida, al menos, por un blanco.

La *proposición* comentario permite colocar información descriptiva más extensa. No tiene efecto en el programa compilado, dado que sólo se imprime en el listado.

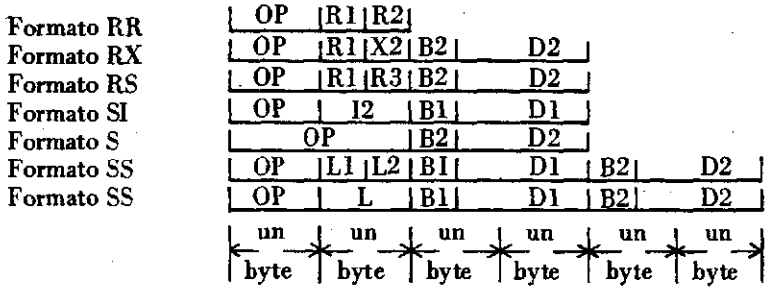
Puede aparecer en cualquier parte

Debe iniciarse con asterisco en columna 1.

b) *Identificación o Secuencia*

Ocupa las columnas 73 a 80 y se puede utilizar opcionalmente para identificar las tarjetas de un programa y/o para verificar la secuencia de dichas tarjetas.

B. *Formatos de las instrucciones del Sistema/360 y /370*



Formato RR: Registro a Registro (*R*egister to *R*egister)

Formato RX: Registro a Memoria Indexada (*R*egister to Indexed Storage)

Formato RS: Registro a Memoria (*R*egister to *S*torage)

Formato SI: Memoria y operando inmediato (*S*torage and *I*mmediate)

Formato S: Memoria (*S*torage)

Formato SS: Memoria a Memoria (*S*torage to *S*torage)

C. *Convenciones y símbolos que se adoptarán*

a) [ ] lo que está encerrado entre los paréntesis[ ]es

		optativo de ser colocado
b)	< >	indica "contenido de"
c)	$< >_s$	indica "contenido supuesto de"
d)	=	símbolo de definición, significa "definido por"
e)	RUG	registro de uso general
f)	RC	registro de control
g)	RPF	registro de punto flotante
h)	R1,R2,R3	registros de uso general, de control o de punto flotante que actúan como primer, segundo o tercer operando respectivamente
i)	X2,B2,B1	registros de uso general, índice y base respectivamente, que forman parte de la dirección del segundo (primer) operando o del segundo (primer) operando mismo
j)	D2,D1	valor decimal correspondiente a desplazamiento. Este forma parte de la dirección del segundo (primer) operando o del segundo (primer) operando mismo
k)	L	longitud del primer y segundo operando
l)	L1,L2	longitud del primer y segundo operando respectivamente
m)	I2	segundo operando, inmediato, esto es, está en la misma instrucción
n)	M1,M3	grupo de cuatro bits utilizado como máscara. Puede ser primer o tercer operando respectivamente

ñ) Las letras mayúsculas y puntuación representan información que debe ser codificada tal como aparece.

o) Las letras minúsculas representan información que debe ser proporcionada por el programador.

D. *Alineamiento*

Se debe tener presente que las direcciones en memoria corresponden a la dirección de un solo byte. Algunas instrucciones que direccionan un byte operan siempre con ese byte y un número fijo de bytes que le sigue. Este número fijo puede ser uno, formando un operando de dos bytes, media palabra (half-word), tres, formando un operando de cuatro bytes, una palabra (full-word) o siete, formando un operando de ocho bytes, doble palabra (double-word). En otras instrucciones debe especificarse la longitud, en bytes, del operando.

El alineamiento (boundary) es una restricción que debe cumplirse en la programación de operandos de longitud fija. Es así como la dirección del byte del extremo izquierdo de un campo fijo de dos bytes (half-word), cuatro bytes (full-word) u ocho bytes (double-word) debe

ser múltiplo de dos, cuatro u ocho respectivamente.

E. *Definición de constantes y áreas*

a) *Formato de la pseudo-instrucción DC (Define Constant)*

[nombre] DC dtLn'c'

donde:

nombre: identifica al byte del extremo izquierdo del campo ocupado por la constante
DC : código de operación
d : factor de repetición o número de veces que se define la constante. Puede ser omitido
t : tipo de constante. Debe aparecer
Ln : modificador de longitud donde *n* representa el número de bytes que tendrá la constante, y puede ser un valor decimal sin signo, o una expresión absoluta positiva encerrada entre paréntesis. Puede omitirse
'c' : constante que se desea generar en memoria.

b) *Constante de punto fijo F y H*

Estas constantes si son definidas sin modificador de longitud quedan con alineamiento de palabra y de media palabra respectivamente. En caso contrario, no se tiene seguridad de que la constante generada quede en el alineamiento correspondiente.

Ejemplo 1.

```
JUAN    DC    H'14'
PEDRØ   DC    F'46'
```

quedan en memoria como sigue (se supone que JUAN se inicia en una palabra):

los dos bytes achurados se saltaron para cumplir con el alineamiento que le corresponde a PEDRO (el primer bit en cada constante representada en binario, indica el signo, 0 si es positivo, 1 si es negativo).

Ejemplo 2.

```
JUAN    DC    H'14'
PEDRØ   DC    FL4'46'
```

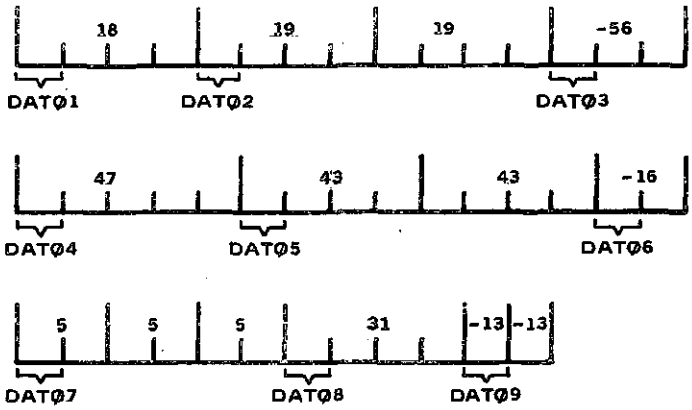
quedan en memoria en la siguiente forma:

Se pierde el alineamiento por existir el modificador de longitud L4.

Ejemplo 3.

```
                1ª columna
                ↓
DATØ1    DC    F'18'
DATØ2    DC    2F'19'
DATØ3    DC    FL3'-56'
DATØ4    DC    FL5'47'
DATØ5    DC    2FL4'+43'
DATØ6    DC    H'-16'
DATØ7    DC    3H'5'
DATØ8    DC    HL4'+31'
DATØ9    DC    2HL1'-13'
```

Estas constantes como parte de un programa, quedan definidas en memoria, una a continuación de la otra.



c) *Formato de la pseudo-instrucción DS (Define Storage)*

[nombre] DS dtLn

Como se observa en el formato, difiere de la proposición DC sólo en que no tiene el operando 'c', dado que DS permite reservar un área de memoria incluso sin borrar lo que existía en ella anteriormente.

F. *Direccionamiento y registros de uso general*

El sistema IBM/360/370 tiene un juego de 16 registros (0-15) que sirven

como:

Registros índice
Registros para control de programas
Acumuladores para aritmética de punto fijo
Acumuladores para aritmética de direcciones

por este motivo se denominan "registros de uso general" (RUG).

La capacidad de cada registro es una palabra (full-word) o lo que es lo mismo, 32 bits (0-31 de izquierda a derecha). Cada bit representa un coeficiente de una potencia de dos aumentando ésta de valor, de derecha a izquierda. El bit 0 o bit de orden superior, si se considera el valor de la potencia de dos, representa el signo en las operaciones algebraicas (0 indica signo positivo, 1 indica signo negativo).

En la aritmética de direcciones, se utilizan sólo los 24 bits de orden inferior. Esto permite direccionar hasta 16 777 216 bytes.

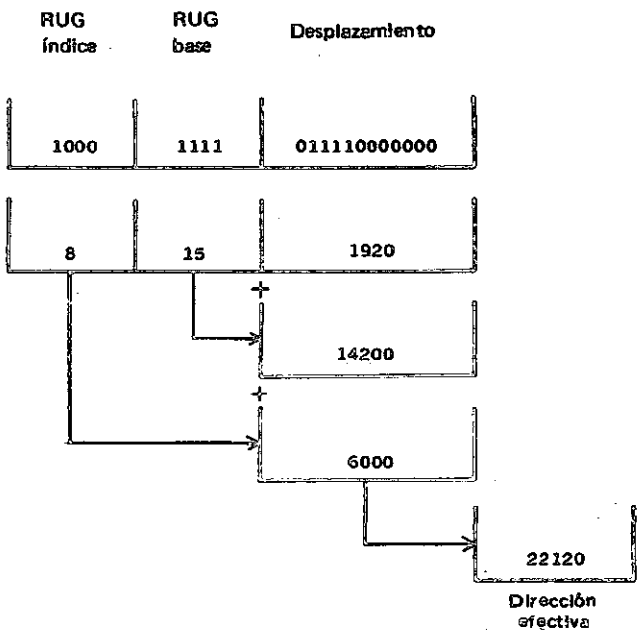
Un campo de cuatro bits en la instrucción permite especificar uno de los 16 registros. Los 24 bits de orden inferior de ese registro contienen una dirección, conocida como dirección base (B). Además, la instrucción contiene un campo de 12 bits cuyo contenido se conoce como desplazamiento (D). Este desplazamiento, al ser sumado a la dirección base, permite direccionar posiciones en memoria de hasta 4095 bytes, más allá de la dirección base.

Muchas instrucciones tienen otro campo de cuatro bits para designar un registro de uso general, denominado registro índice (X). En estas instrucciones la dirección efectiva se obtiene sumando los contenidos de los registros B y X y el desplazamiento D.

Dirección efectiva = <RUG B> + <RUG X> + D

Ejemplo 4.

Sea	RUG X el RUG 8 y su contenido	6000
	RUG B el RUG 15 y su contenido	14200
	Desplazamiento D	1920
	la dirección efectiva será	22120



Observación: El RUG 0 no se puede utilizar como registro base o índice. La aparición del 0 en esos campos es interpretado por el sistema como ausencia de registro.

3. *Aritmética de punto fijo*

En todas las operaciones que se verán en adelante, es necesario tener presente que la información obtenida de un almacenamiento magnético permanece en él sin alteración. Por el contrario, la información que se carga en un almacenamiento magnético borra lo que había antes en él.

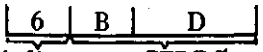
Las instrucciones de este capítulo corresponden al grupo de Aritmética de Punto Fijo, sin embargo, en algunos casos será necesario ver instrucciones de otros grupos, lo cual se indicará en la instrucción.

Las direcciones de almacenamiento son representadas generalmente en assembler por símbolos. La dirección efectiva que corresponde a ese símbolo es desglosada por el compilador en: un contenido de registro base y un desplazamiento. En aquellas instrucciones en que puede especificarse registro índice, si se indica sólo el símbolo, el compilador supone que no se hará uso de registro índice, y coloca 0 en la instrucción de máquina. Si se desea indicar registro índice, se especifica a continuación del símbolo, encerrado entre paréntesis ( ), el número del registro utilizado.

Ejemplo 5.

a) JUAN

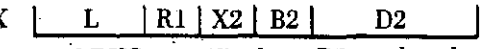
b) PEDRØ(6)



3.1. *Instrucción* LOAD

a) Instrucción: L R1,D2(X2,B2)

b) Formato : RX



c) Función : Se carga el RUG especificado en R1, con la palabra contenida en la dirección calculada con D2,X2 y B2.

Ejemplo 6.

Instrucciones válidas y no válidas
Válidas:

i) L 5,40(7,15)

función, en símbolos: <RUG5> = <40 + <RUG7> + <RUG15>>

Explicación: el contenido del RUG 5 se define por el contenido de la dirección calculada como 40, más contenido de RUG 7, más contenido de RUG15.

ii) L 7, JUAN

Se carga el RUG 7 con la palabra contenida en la dirección JUAN

iii) L 2,PEDRØ+4

Se carga el RUG 2 con la palabra contenida en la dirección PEDRO+4

iv) L 1,DIEGØ+2(3)

Se carga el RUG 1 con la palabra contenida en la dirección DIEGO+2+ <RUG 3>

No válidas:

v) L 15,4096(8,14)

Error en el desplazamiento, que como máximo puede ser 4095

vi) L 8,AREA(2,15)

Error porque se obliga al compilador a considerar el símbolo AREA como desplazamiento. El compilador rechaza esta instrucción porque todo símbolo de dirección tiene un valor superior a 4095

vii) L 6,DATØ(,13) o L 6,DATØ(0,13)

Igual motivo al de vi).

viii) Se producirá error por especificación si la dirección no corresponde a un alineamiento de palabra.

3.2. *Instrucción* LOAD HALF

a) Instrucción: LH R1,D2(X2,B2)

b) Formato : RX

LH	R1	X2	B2	D2

c) Función : Se carga el RUG especificado como primer operando,con la media palabra ubicada en la dirección calculada con D2, X2 y B2.

El procedimiento es el que se indica. La media palabra se lleva a la unidad aritmética donde es expandida a palabra completa propagando el bit de signo a través de las 16 posiciones de orden superior. El resultado se carga en el RUG. (Ver ejemplo 7).

3.3. *Instrucción* LOAD REGISTER

a) Instrucción: LR R1,R2

b) Formato : RR

LR	R1	R2

c) Función : Se carga el RUG especificado como primer operando,con el contenido del RUG indicado como segundo operando, (ver ejemplo 7).

Ejemplo 7.

```
DATØ1   DC    F'-18'
ALFA    DC    H'14'
BETA    DC    H'-35'
        L     5,DATØ1
        LH    6,ALFA
        LH    7,BETA
        LR    1,5
        LR    2,6
```

Explicación: Una vez definidas las constantes DATØ1, ALFA y BETA se cargan en los RUG 5, 6 y 7 respectivamente. Para efectuar la operación de carga, se utilizan las instrucciones LOAD y LOAD HALF. A continuación se cargan los RUG 1 y 2 con los contenidos de los RUG 5 y 6 utilizando la instrucción LOAD REGISTER.

3.4. *Instrucción* LOAD ADDRESS (Instrucción Lógica)

a) Instrucción: LA R1,D2(X2,B2)

b) Formato : RX

LA	R1	X2	B2	D2

c) Función : Se carga en el RUG especificado en R1 la dirección dada por D2, X2 y B2. La dirección ocupa los bits 8-31 del RUG, los bits 0-7 se ponen en cero.

Ejemplo 8.

```
LA      3,PEDRØ
LA      4,15(0,4)
LA      5,JUAN(6)
LA      0,500
```

Explicación: Se carga el RUG 3 con la dirección PEDRO. Se carga el RUG4 con la dirección 15+ <RUG4>. Se carga el RUG5 con la dirección JUAN+ <RUG6>. Se carga el RUG0 con la dirección 500. Al ser traducida esta última instrucción queda el valor 500 como desplazamiento y 0 en registro índice y registro base.

3.5. *Instrucción* STORE

a) Instrucción : ST R1,D2(X2,B2)

b) Formato : RX

c) Función : El contenido del RUG especificado como primer operando se almacena en la dirección calculada con D2, X2 y B2. Esa dirección debe estar en alineamiento de palabra.

Ejemplo 9.

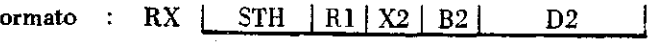
```
CTE1    DC      F'52'
AREA    DS      F
        L       7,CTE1
        ST      7,AREA
```

Explicación: Se carga el RUG7 con el valor 52 y luego se almacena ese contenido en la dirección AREA, ocupando una palabra.

3.6. *Instrucción* STORE HALF

a) Instrucción: STH R1,D2(X2,B2)

b) Formato : RX



c) Función : El contenido de la mitad inferior del RUG especificado como primer operando se almacena en la dirección calculada con D2, X2 y B2. Ocupa en memoria dos bytes. La mitad superior del RUG no interviene en la operación.

Ejemplo 10.

```
STH     5,AREA
STH     6,AREA+2
```

Explicación: Se almacena el contenido de la mitad inferior del RUG5 en la dirección AREA, y el del RUG6 en la dirección AREA+2.

3.7. *Instrucción* ADD

a) Instrucción: A R1,D2(X2,B2)

b) Formato : RX

A	R1	X2	B2	D2

c) Función : Al contenido del RUG especificado en R1, se le suma la palabra contenida en la dirección entregada por D2, X2 y B2. El resultado queda en el RUG especificado en R1.

Todas las instrucciones de tipo aritmético generan un Código de Condición (CC) de cuatro posibles, el cual queda registrado en la Palabra de Estado del Programa (Program Status Word-PSW) vigente, en los bits 34 y 35. Códigos de Condición generados por la instrucción ADD:

CC	*Resultado*	
00	cero	0
01	menor que cero	<0
10	mayor que cero	>0
11	desborde	ØF

El desborde (overflow) se produce cuando el resultado excede la capacidad del registro. Se produce interrupción (interrupt) de programa si el bit de desborde de punto fijo está en uno.

Ejemplo 11.

```
DATØA    DC     F'18'
DATØB    DC     F'52'
DATØC    DC     F'-31'
         L      6,DATØA
         A      6,DATØB
         A      6,DATØC
```

Explicación: Se carga el RUG6 con el contenido de DATØA que es la constante 18. A continuación se suma al contenido del RUG6 el contenido de DATØB; queda como resultado en RUG6 el valor 70. Finalmente se suma a ese resultado el contenido de DATØC. Queda en RUG6 el valor 39.

3.8. *Instrucción* ADD REGISTER

a) Instrucción: AR R1,R2

b) Formato : RR

AR	R1	R2

c) Función : Se suma al contenido del RUG especificado en R1 el contenido del RUG especificado en R2. El resultado queda en el RUG indicado en R1.

Se generan los mismos CC que genera la instrucción ADD.

3.9. *Instrucción* ADD HALF

a) Instrucción: AH R1,D2(X2,B2)

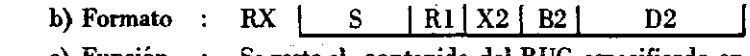
b) Formato :

c) Función : Se suma al contenido del RUG especificado en R1, la media palabra contenida en la dirección calculada con D2, X2 y B2. Para efectuar la suma, se expande la media palabra a palabra completa mediante la propagación del bit de signo a través de las 16 posiciones de orden superior. La expansión se realiza en la Unidad Aritmética.

3.10. *Instrucción* SUBTRACT

a) Instrucción: S R1,D2(X2,B2)

b) Formato :

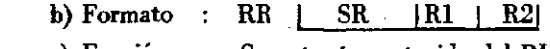


c) Función : Se resta al contenido del RUG especificado en R1 la palabra contenida en la dirección calculada con D2, X2 y B2. El resultado queda en el RUG especificado en R1.

3.11. *Instrucción* SUBTRACT REGISTER

a) Instrucción: SR R1,R2

b) Formato :

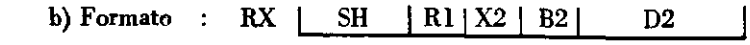


c) Función : Se resta al contenido del RUG especificado en R1 el contenido del RUG especificado en R2. El resultado queda en el RUG indicado en R1.

3.12. *Instrucción* SUBTRACT HALF

a) Instrucción: SH R1,D2(X2,B2)

b) Formato :



c) Función : Se resta al contenido del RUG especificado en R1 la media palabra contenida en la dirección calculada con D2, X2 y B2. Para efectuar la resta, se expande la media palabra a palabra completa mediante la propagación del bit de signo a través de las 16 posiciones de orden superior. La expansión se realiza en la Unidad Aritmética.

3.13. *Instrucción* ADD LOGICAL REGISTER

a) Instrucción: ALR R1,R2

b) Formato : RR | ALR | R1 | R2 |

c) Función : Se suma al contenido del RUG especificado en R1 el contenido del RUG indicado en R2. Se suman los 32 bits de ambos operandos sin que haya cambio posterior en el bit de signo del resultado. Si hay un desborde desde la posición del signo, se registra en el Código de Condición.

Código de Condición	*Resultado*
0	cero (sin desborde)
1	distinto de cero (sin desborde)
2	cero (con desborde)
3	distinto de cero (con desborde)

3.14. *Instrucción* ADD LOGICAL

a) Instrucción: AL R1,D2(X2,B2)

b) Formato : RX | AL | R1 | X2 | B2 | D2 |

c) Función : Se suma al contenido del RUG especificado en R1 la palabra contenida en la dirección calculada con D2, X2 y B2. Se generan los mismos códigos que con ALR.

3.15. *Instrucción* SUBTRACT LOGICAL REGISTER

a) Instrucción: SLR R1,R2

b) Formato : RR | SLR | R1 | R2 |

c) Función : Se resta al contenido del RUG especificado en R1 el contenido del RUG indicado en R2. Participan los 32 bits de ambos operandos, sin que haya cambio posterior en el bit de signo del resultado.

Código de Condición	*Resultado*
0	- - -
1	distinto de cero (sin desborde)
2	cero (con desborde)
3	distinto de cero (con desborde)

3.16. *Instrucción* SUBTRACT LOGICAL

a) Instrucción: SL R1,D2(X2,B2)

b) Formato : RX | SL | R1 | X2 | B2 | D2 |

c) Función : Se resta al contenido del RUG especificado en R1 la palabra contenida en la dirección calculada con D2, X2 y B2. Se generan los mismos códigos que con SLR.

Ejemplo 12.

```
A       DC      F'57'
B       DC      F'415'
C       DC      F'-35'
Z       DS      F
        L       5,A         <RUG5>=57
        L       7,B         <RUG7>=415
        ALR     5,5         <RUG5>=114
        AL      5,C         <RUG5>=79
        SLR     7,5         <RUG7>=336
        SL      7,A         <RUG7>=279
        ST      7,Z
```

El resultado sería el mismo si se hubieran utilizado las instrucciones AR, A, SR y S.

3.17. *Instrucción* MULTIPLY

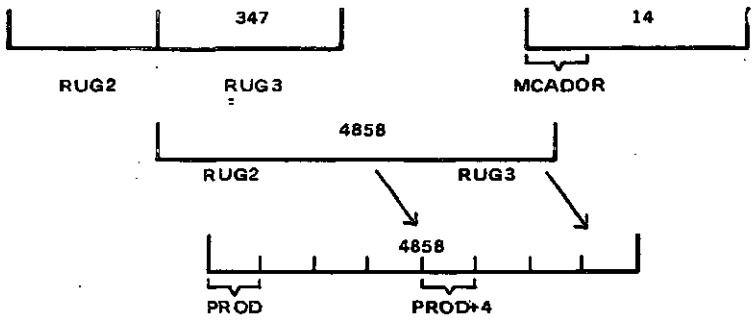
a) Instrucción: M R1,D2(X2,B2)

b) Formato : RX | M | R1 | X2 | B2 | D2 |

c) Función : El producto del multiplicando (1er operando) y el multiplicador (2º operando) reemplaza al primero. Como ambos operandos tienen 32 bits, el producto será un entero de 64 bits que ocupa un par de RUG, PAR e IMPAR siguiente. Debido a que el producto reemplaza al multiplicando, en el campo R1 se especifica el RUG PAR. El contenido de este RUG se ignora, a menos que contenga al multiplicador.

Ejemplo 13.

```
MCANDØ  DC      F'347'
MCADØR  DC      F'14'
PRØD    DS      2F
        L       3,MCANDØ
        M       2,MCADØR
        ST      2,PRØD
        ST      3,PRØD+4
```



3.18. *Instrucción* MULTIPLY REGISTER

a) Instrucción: MR R1,R2

b) Formato : RR | MR | R1 | R2 |

c) Función : Se realiza la misma función que efectúa la instrucción Multiply. El multiplicador (2º operando) está contenido en un RUG.

Ejemplo 14.

```
MCANDØ   DC    F'347'
MCADØR   DC    F'14'
PRØD     DS    2F
         L     3,MCANDØ
         L     7,MCADØR
         MR    2,7
         ST    2,PRØD
         ST    3,PRØD+4
```

Ejemplo 15.

```
MCANDØ   DC    F'347'
MCADØR   DC    F'14'
PRØD     DS    2F
         L     3,MCANDØ
         L     2,MCADØR
         MR    2,2
         ST    2,PRØD
         ST    3,PRØD+4
```

En este último ejemplo se utilizó el RUG2 para contener el multiplicador aprovechando que el contenido del RUG PAR se ignora "a menos que contenga el multiplicador".

3.19. MULTPLY HALF

a) Instrucción: MH R1,D2(X2,B2)

b) Formato :

c) Función : El producto del multiplicando (1er. operando) y el multiplicador (2º operando) reemplaza al primero. Antes de la multiplicación el multiplicador se expande de media palabra a palabra completa mediante la propagación del bit de signo a través de las 16 posiciones de orden superior. Una vez efectuada la multiplicación, el multiplicando es reemplazado por los 32 bits de orden inferior del producto. El RUG especificado en R1 puede ser par o impar.

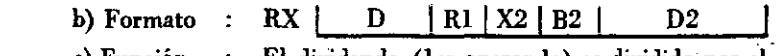
Ejemplo 16.

```
MCANDØ   DC    H'347'
MCADØR   DC    H'14'
PRØD     DS    F
         LH    5,MCANDØ
         MH    5,MCADØR
         ST    5,PRØD
```

3.20. *Instrucción* DIVIDE

a) Instrucción: D R1,D2(X2,B2)

b) Formato :



c) Función : El dividendo (1er operando) es dividido por el divisor (2º operando) y reemplazado por el residuo y cociente. El dividendo ocupa dos RUG, PAR e IMPAR siguiente. El divisor es una palabra contenida en la dirección calculada con D2, X2 y B2. El residuo ocupará el RUG PAR especificado en R1, y el cociente el RUG IMPAR siguiente, ambos con sus respectivos signos.

La división por cero produce una interrupción de programa. No se realiza la división y los operandos no cambian.

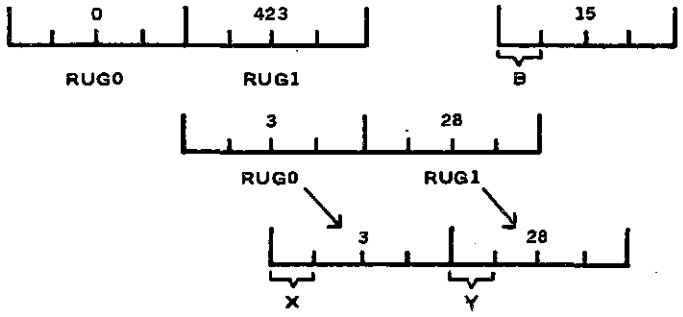
Ejemplo 17.

Se desea dividir 423:15

```
A        DC    F'423'
B        DC    F'15'
X        DS    F
```

```
Y       DS      F
        SR      0,0
        L       1,A
        D       0,B
        ST      0,X
        ST      1,Y
```

Las áreas X e ,Y se reservan para el residuo y cociente respectivamente. El RUG0 se deja en cero mediante la instrucción SR, con el objeto de borrar todo dato que hubiera de problemas anteriores. Esta operación debe hacerse porque la instrucción de división considera el dividendo como contenido en los dos RUG, PAR e IMPAR siguiente.



3.21. *Instrucción* DIVIDE REGISTER

a) Instrucción: DR R1,R2

b) Formato : RR | DR | R1 | R2 |

c) Función : Se realiza la misma función que efectúa la instrucción DIVIDE. El divisor (2º operando) está contenido en un RUG.

Ejemplo 18.

El mismo ejemplo 17 utilizando LA y DR.

```
X       DS      F
Y       DS      F
        LA      3,423
        LA      7,15
        SR      2,2
        DR      2,7
        ST      2,X
        ST      3,Y
```

Se cargan los RUG 3 y 7 con los valores 423 y 15 utilizando la instrucción Load Address. El RUG2 se deja en cero para borrar resultados anteriores. El residuo que queda en RUG2 se almacena en X, el cociente que queda en RUG3 se almacena en Y.

Ejemplo 19.

Cálculo de: VALOR=2489 + 3% de 2489; se simplifica si se coloca $VALOR = \frac{2489*103}{100}$

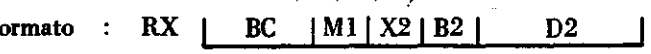
```
VALØR    DS    F
CIEN     DC    F'100'
CØNSTA   DC    F'50'
CØNSTB   DC    F'2489'
CØNSTC   DC    F'103'
         L     5,CØNSTB
         M     4,CØNSTC
         A     5,CØNSTA
         D     4,CIEN
         ST    5,VALØR
```

Para que se efectúe un redondeo al entero superior, en el resultado final, es necesario sumar; antes de realizar la división, la constante CONSTA de valor 50.

3.22. *Instrucción* BRANCH ON CONDITION (Instrucción de bifurcación)

a) Instrucción: BC M1,D2(X2,B2)

b) Formato : RX



c) Función : Si se cumple el CC correspondiente a la máscara M1, se ejecuta la instrucción que está en la dirección dada por D2, X2 y B2. En caso contrario, se sigue la secuencia normal de ejecución, esto es, se ejecuta la instrucción que viene a continuación.

El "branch" es una interrupción de la secuencia normal de ejecución de un programa, para continuarla en otro punto de él, anterior o posterior al punto de interrupción. También se denominan "saltos" en el programa, porque permiten saltarse una o más instrucciones hacia adelante o atrás.

El branch se realiza normalmente de acuerdo a una condición producida anteriormente, y de ahí el nombre de la instrucción, branch on condition. La mayoría de las instrucciones genera un Código de

Condición (CC) de cuatro posibles, el cual queda establecido en la Palabra de Estado del Programa vigente (PSW) en los bits 34 y 35. La instrucción, consulta si se ha producido un CC determinado, y si ello ha ocurrido, se efectúa el salto.

d) Relación existente entre las máscaras y los Códigos de Condición

M1 (decimal)	M1 (binario)	CC (decimal)	CC (binario)
8	1000	0	00
4	0100	1	01
2	0010	2	10
1	0001	3	11

las anteriores son las máscaras fundamentales que, combinadas, producen lo siguiente:

M1 (decimal)	M1 (binario)	CC (decimal)	CC (binario)
3	0011	2 ó 3	10 ó 11
5	0101	1 ó 3	01 ó 11
6	0110	1 ó 2	01 ó 10
7	0111	1 ó 2 ó 3	01 ó 10 ó 11
9	1001	0 ó 3	00 ó 11
10	1010	0 ó 2	00 ó 10
11	1011	0 ó 2 ó 3	00 ó 10 ó 11
12	1100	0 ó 1	00 ó 01
13	1101	0 ó 1 ó 3	00 ó 01 ó 11
14	1110	0 ó 1 ó 2	00 ó 01 ó 10
15	1111	0 ó 1 ó 2 ó 3	00 ó 01 ó 10 ó 11
0	0000	---	---

La máscara 15 corresponde a BRANCH (salto) incondicional, dado que cualquier Código de Condición que se produzca se efectúa siempre el salto.

3.23. *Instrucción* BRANCH ON CONDITION REGISTER (Instrucción de bifurcación)

a) Instrucción: BCR M1,R2

b) Formato : RR

BCR	M1	R2

c) Función : Si se cumple el CC correspondiente a la máscara M1, se ejecuta la instrucción que está en la dirección contenida en el RUG especificado en R2.

3.24. *Instrucciones de "Comparación Algebraica"*

a) Formato: nombre EQU operando
b) Función: Se asigna al símbolo especificado en nombre, el valor que tiene el operando.

Ejemplo 24.

```
MENOR   EQU   4
```

se asigna al símbolo MENOR el valor 4, esto significa que donde aparezca el símbolo MENOR se interpretará como 4. Así se puede tener:

i) AR MENOR,MENOR que equivale a
AR 4,4

ii) BC MENOR,OUT que equivale a
BC 4,OUT

iii) BC 8,NEXT+MENOR que equivale a
BC 8,NEXT+4

3.28. *Instrucción* LOAD NEGATIVE REGISTER

a) Instrucción: LNR R1,R2

b) Formato : RR | LNR | RI | R2 |

c) Función : El contenido del RUG especificado en R2 se carga con signo negativo en el RUG indicado en R1. Los números negativos no sufren alteración. El número cero queda siempre positivo. Se genera CC de acuerdo al resultado.

CC	*Resultado*
00	Cero
01	Menor que cero

3.29. *Instrucción* LOAD POSITIVE REGISTER

a) Instrucción: LPR R1,R2

b) Formato : RR | LPR | R1 | R2 |

c) Función: : El contenido del RUG especificado en R2 se carga con signo positivo en el RUG indicado en R1. Los números positivos no sufren alteración. Se genera CC de acuerdo al resultado.

CC	*Resultado*
00	cero
10	mayor que cero
11	desborde

3.30. *Instrucción* LOAD COMPLEMENT

a) Instrucción: LCR R1,R2

b) Formato : RR | LCR | R1 | R2 |

c) Función : El contenido del RUG especificado en R2 se carga con signo contrario en el RUG indicado en R1. El número cero queda siempre positivo. Se genera CC de acuerdo al resultado.

CC	*Resultado*
00	cero
01	menor que cero
10	mayor que cero
11	desborde

3.31. LOAD AND TEST

a) Instrucción: LTR R1,R2

b) Formato : RR | LTR | R1 | R2 |

c) Función : El contenido del RUG especificado en R2 se carga en el RUG indicado en R1. El signo y la magnitud determinan el CC. El segundo operando no sufre alteración.

CC	*Resultado*
00	cero
01	menor que cero
10	mayor que cero

Cuando se especifica el mismo registro en R1 y R2, la operación equivale a una prueba sin movimiento de datos.

3.32. *Instrucción* SHIFT LEFT SINGLE

a) Instrucción: SLA R1,D2(B2)

b) Formato : RS | SLA | R1 | B2 | D2 |

c) Función : El contenido del RUG especificado en R1 se desplaza a la izquierda, el número de posiciones indicado por los seis bits de orden inferior de la representación binaria de la dirección dada por D2 y B2. El resto de la dirección se ignora.
El signo de la información que se desplaza permanece sin cambio. Los lugares vacantes se llenan con ceros. Si se pierde algún bit distinto del signo se produce desborde.

Ejemplo 26.

```
CTE     DC      F'17'
RES     DS      2F
        L       1,CTE
        SLDA    0,31
        SRA     0,1
        SRDA    0,31
        ST      0,RES
        ST      1,RES+4
```

Explicación: Con la instrucción L 1,CTE se tiene:

		00.....010001	RUG1
SLDA	0,31	00.....01000	10.....0000
		RUG0	RUG1
SRA	0,1	00.....00100	10.....0000
		RUG0	RUG1
SRDA	0,31	00.....00000	00.....1001
		RUG0	RUG1

queda así, entonces en RUG0 el valor 0 y en RUG1 el valor 9

3.36. *Modificador de escala para constantes de punto fijo*

El modificador de escala especifica la potencia de dos por la cual la constante debe ser multiplicada después que ella ha sido convertida a su representación binaria. Esto significa que una porción fraccionaria puede ser desplazada a representación entera si el exponente es positivo. Por el contrario, si el exponente es negativo, la parte entera puede ser borrada total o parcialmente. El modificador de escala se representa como Sn, donde *n* es un valor decimal entero con o sin signo o una expresión absoluta encerrada entre paréntesis ( ).

Ejemplo 27.

```
DATØ1   DC      FS5'14.25'
```

Si no existiera el modificador de escala se representaría sólo el valor 14. En este caso la representación corresponderá a $14.25*2^5$ que es igual a 456. La representación binaria de 14.25 es:

0.....01110	01

punto binario

si esta información se desplaza 5 posiciones a la izquierda queda:

|0. . . . . .0111001000|

que corresponde al valor 456.

Ejemplo 28.

Cálculo de $2489+\frac{2489\cdot 3}{100}$, o lo que es lo mismo 2489*1.03

a)
```
AREA     DS    F
CIEN     DC    F'100'
CØNSTA   DC    F'50'
CØNSTB   DC    F'2489'
CØNSTC   DC    F'103'
         L     5,CØNSTB
         M     4,CØNSTC
         A     5,CØNSTA
         D     4,CIEN
         ST    5,AREA
```

b) Otra forma, usando modificador de escala Sn

```
AREA     DS    F
CØNSTA   DC    FS11'0,5'
CØNSTB   DC    F'2489'
CØNSTC   DC    FS11'1.03'
         L     5,CØNSTB
         M     4,CØNSTC
         A     5,CØNSTA
         SRA   5,11
         ST    5,AREA
```

3.37. *Otros tipos de constantes*

a) Constante tipo X.

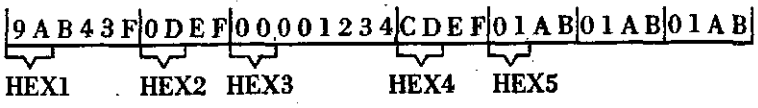
El tipo X permite definir constantes hexadecimales. Estas constantes consisten de uno o más dígitos hexadecimales. La longitud máxima es de 256 bytes, y el campo ocupado por la constante no se ajusta a alineamiento.

Cada dígito hexadecimal ocupa medio byte. Si la longitud del campo excede al número de dígitos que se genera, se completa el campo con ceros hexadecimales por la izquierda. Si el campo no puede contener a todos los dígitos, se trunca la constante por la izquierda.

Ejemplo 29.

```
HEX1     DC    X'9AB43F'
HEX2     DC    X'DEF'
HEX3     DC    XL4'1234'
HEX4     DC    XL2'ABCDEF'
HEX5     DC    3X'1AB'
```

Estas constantes quedarán generadas en la siguiente forma:



b) Constante tipo C.

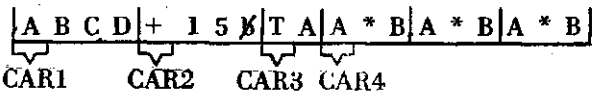
El tipo C permite definir caracteres en el código EBCDIC. Cada carácter por tanto ocupa un byte. La longitud máxima es de 256 bytes y el campo ocupado por la constante no se ajusta a alineamiento.

Si la longitud del campo excede al número de caracteres que se genera, se completa el campo con caracteres blancos, por la derecha. Si el campo no puede contener a todos los caracteres, se trunca la constante por la derecha.

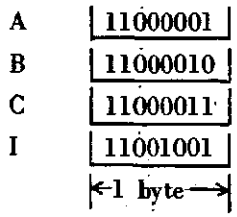
Ejemplo 30.

```
CAR1    DC    C'ABCD'
CAR2    DC    CL4'+15'
CAR3    DC    CL2'TABLA'
CAR4    DC    3C'A*B'
```

Estas constantes quedarán generadas en la siguiente forma:



En las tarjetas, la representación de las letras A,B,C,...,I está dada por una perforación en ZONA 12 y una perforación en el dígito 1,2,3,...ó 9 según sea la letra. En memoria, con el tipo C, quedan representadas esas mismas letras como sigue:



se puede observar que los cuatro bits de orden superior del byte tienen el valor binario 1100 = 12 decimal, y los cuatro bits de orden inferior tienen el valor 1, 2, 3 ó 9 decimal, según sea la letra. Aun cuando no existe la misma relación para el resto de los caracteres se acostumbra decir que la representación en memoria está en la forma ZONA–CARACTER o ZONA DIGITO o ZONA NUMERO, estas dos últimas aplicadas más bien al formato del tipo Z que se verá más adelante.

```
|Z,C|Z,C|Z,C|Z,C|Z,C|
```

De acuerdo a esta estructura, las constantes del ejemplo 30 se puede decir que quedan como sigue:

```
|ZAZBZCZD|Z+Z1Z5Z|ZTZA|ZAZ*ZB|ZAZ*ZB|ZAZ*ZB|
  CAR1      CAR2    CAR3   CAR4
```

c) Constante tipo Z.

El tipo Z permite definir valores numéricos en el formato ZONA–DIGITO. Difiere con el formato obtenido con el tipo C, en que el byte de orden inferior tiene la estructura SIGNO–DIGITO.

```
|Z.D|Z.D|Z.D|Z.D|S.D|
```

D = dígito S = signo

Si la longitud del campo excede al número de dígitos de la constante, se completa el campo con dígitos cero por la izquierda. Si el campo no puede contener a todos los dígitos, se trunca la constante por la izquierda.

Ejemplo 31.

```
ZØN1    DC    Z'+352'
ZØN2    DC    Z'-437'
ZØN3    DC    ZL5'4'
ZØN4    DC    ZL2'-3895'
ZØN5    DC    2ZL3'18'
```

Estas constantes quedarán generadas en la siguiente forma:

```
|Z3Z5+2|Z4Z3-7|Z0Z0Z0Z0+4|Z9-5|Z0Z1+8|Z0Z1+8|
  ZØN1    ZØN2    ZØN3        ZØN4   ZØN5
```

d) Constante tipo P.

El tipo P permite definir valores numéricos en el formato de dígitos empaquetados.

```
|D D|D D|D D|D D|D S|
```

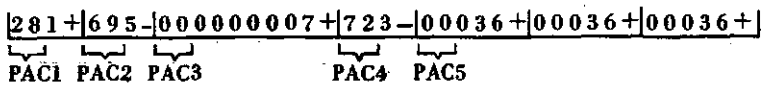
D = dígito S = signo

Si la longitud del campo excede al número de dígitos de la constante, se completa el campo con dígitos cero por la izquierda. Si el campo no puede contener a todos los dígitos, se trunca la constante por la izquierda.

Ejemplo 32.

```
PAC1    DC    P'+281'
PAC2    DC    P'-695'
PAC3    DC    PL5'7'
PAC4    DC    PL2'-856723'
PAC5    DC    3PL3'36'
```

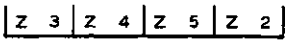
Estas constantes quedarán generadas en la siguiente forma:



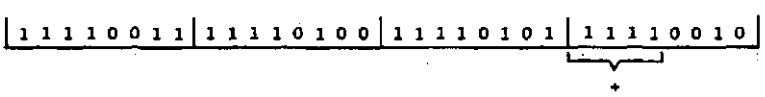
No existen instrucciones que permitan realizar operaciones aritméticas con constantes definidas con el tipo C, o el tipo Z. Por otra parte, la información al ser leída a memoria queda en el formato ZONA–CARACTER. Sin embargo, existen instrucciones que permiten convertir del formato ZONA–DIGITO (formato del tipo Z) a formato DIGITOS EMPAQUETADOS y de éste a BINARIO. Existen además, instrucciones que permiten realizar el proceso inverso, esto es, de formato BINARIO a formato de DIGITOS EMPAQUETADOS y de éste a ZONA–DIGITO. Para la información que se lee a memoria, es necesario considerar que la ZONA de los dígitos es 1111 y esta configuración se interpreta como signo +. Luego los números positivos leídos se pueden considerar como en formato ZONA-DIGITO puesto que el último byte tendrá la estructura 1111XXXX, es decir, + XXXX. En los números negativos será tarea del programador reemplazar con las instrucciones adecuadas la configuración 1111 por 1101 que corresponde al signo –. Se verá a continuación el siguiente ejemplo:

Ejemplo 33.

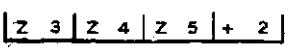
Caso a) Se lee a memoria el dato 3452. Su estructura será:



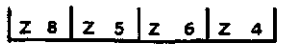
en binario:



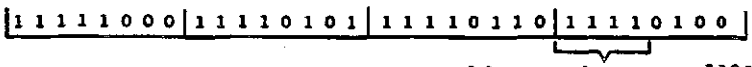
o lo que es lo mismo:



Caso b) Se lee a memoria el dato negativo 8564. Debe leerse como positivo. Su estructura será:

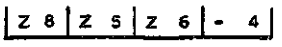


en binario:



debe reemplazarse por 1101

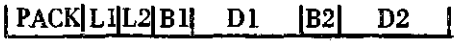
para que quede:



3.38. *Instrucción* PACK (Aritmética Decimal)

a) Instrucción: PACK D1(L1,B1),D2(L2,B2)

b) Formato : SS



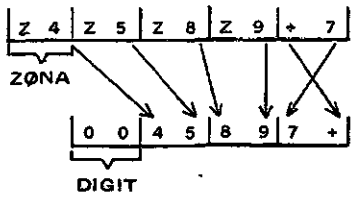
c) Función : El contenido del campo D2(B2) que se considera en formato de ZONA–DIGITO es convertido a formato PACKED y el resultado almacenado en el campo D1(B1). Para la conversión las zonas son ignoradas, excepto la del byte de orden inferior, pues se supone que ella representa el signo. El signo se coloca en los cuatro bits de la derecha del byte de orden inferior, y a la izquierda de él se agrupan los dígitos.

Los campos se procesan de derecha a izquierda y no hay verificación de validez de códigos. Si el campo del primer operando es demasiado corto como para contener todos los dígitos significativos del campo del segundo operando, los dígitos restantes se ignoran. Por el contrario, si el campo es más largo, se rellena con ceros decimales hasta completar el campo.

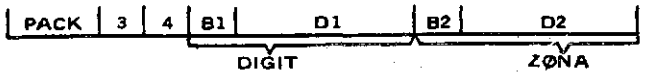
Nota: En todas las instrucciones en que se especifica longitud, el compilador al hacer la traducción resta 1 a las longitudes.

Ejemplo 34.

```
ZØNA    DC      Z'45897'
DIGIT   DS      PL4
        PACK    DIGIT(4),ZØNA(5)
```



La instrucción PACK DIGIT(4),ZONA(5) al ser generada quedará como:



Dado que al ser generada la constante o área, se asigna al símbolo la longitud en que se ha definido la constante o área, se puede escribir con el mismo resultado

```
   PACK     DIGIT,ZØNA
o  PACK     DIGIT(4),ZØNA
o  PACK     DIGIT,ZØNA(5)
```

3.39. *Instrucción* CONVERT TO BINARY

a) Instrucción: CVB R1,D2(X2,B2)

b) Formato : RX

c) Función : La doble palabra contenida en la dirección D2(X2,B2) y cuyo formato es PACKED, es convertida a binario puro y su resultado almacenado en el RUG especificado en R1. El mayor valor que puede convertirse corresponde a la capacidad de un RUG (2147483647 ó -2147483648). La dirección D2(X2,B2) debe corresponder a un alineamiento de doble palabra.

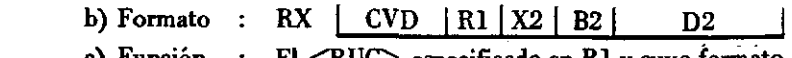
Ejemplo 35.

```
ZØNA      DC      Z'-48963'
BINARIØ   DS      D
RESULT    DS      F
          PACK    BINARIØ,ZØNA
          CVB     5,BINARIØ
          ST      5,RESULT
```

3.40. *Instrucción* CONVERT TO DECIMAL

a) Instrucción: CVD R1,D2(X2,B2)

b) Formato : RX



c) Función : El <RUG> especificado en R1 y cuyo formato es binario puro es convertido a PACKED y depositado en la doble palabra indicada con D2(X2,B2). La dirección debe corresponder a un alineamiento de doble palabra.

3.41. *Instrucción* UNPACKED (Aritmética Decimal)

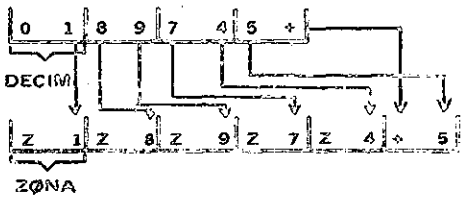
a) Instrucción: UNPK D1(L1,B1),D2(L2,B2)

b) Formato : SS | UNPK | L1 | L2 | B1 | D1 | B2 | D2

c) Función : El contenido del campo D2(B2) que se considera en formato PACKED es convertido a formato ZONA–DIGITO y el resultado almacenado en el campo D1(B1). Los campos se procesan procediendo de derecha a izquierda. Si el campo del primer operando es demasiado corto como para contener todos los dígitos significativos del segundo operando, los dígitos sobrantes se ignoran.

Ejemplo 36.

```
DECIM   DC    P'189745'
ZONA    DS    ZL6
        UNPK  ZONA,DECIM
```



3.42. *Problemas propuestos*

a) Definir:

i) un área de 57 bytes de longitud con alineamiento de palabra.

ii) una constante de nombre BYTE cuyo contenido sea 0111 1011.

b) Indicar los contenidos del RUG2 y AREA al terminar el siguiente proceso:

```
JUAN    DC    F'+ 35'
        DC    F'– 30'
        DC    H'10'
AREA    DS    2F
        L     4,JUAN
        LA    2,10(0,4)
        AR    4,2
        A     4,JUAN+4
        AH    2,JUAN+8
        AR    2,2
        ST    4,AREA
        STH   2,AREA+4
```

c) Ordenar en secuencia ascendente los tres datos que hay en el área TRESDAT (3 palabras completas).

d) Programar la siguiente expresión:

$$Z = 2(A+B-2(C-D))$$

e) Programar la siguiente expresión:
SUELDON = SUELDOA + 21% de SUELDOA

f) Analizar e indicar qué queda en AREA al término del siguiente programa:

```
UNØ     DC    F'1'
DØS     DC    F'2'
CTE1    DC    F'7'
CTE2    DC    F'-18'
AREA    DS    F
        L     1,CTE1
        L     2,CTE2
        SR    1,2
        BC    4,MENOR
        MR    0,2
        ST    1,AREA
        BC    15,ØUT
MENØR   M     0,UNØ
        D     0,DØS
        ST    0,AREA
ØUT
```

g) Analizar e indicar qué queda en RES al término del siguiente programa:

```
ALFA    DC    F'5'
BETA    DC    F'-4'
RES     DS    F
        LH    1,ALFA
        LA    2,10(0,1)
        AR    2,2
        SH    2,BETA
        ST    2,RES
```

h) Programar el siguiente cálculo:

$$Z = 2\,|\,X + 2|X - |Y||- |\ Y\ ||$$

suponer que siempre los dígitos significativos caben en un RUG.

i) Analizar e indicar qué queda en AREA al término del siguiente programa:

```
AREA    DS    F
DATØ1   DC    XL4'1'
DATØ2   DC    PL4'0'
        LM    1,2,DATØ1
        CR    1,2
```

```
         BC    4,MM
         AR    2,1
         BC    15,STØRE
MM       SR    2,1
STØRE    ST    2,AREA
```

j) Indicar el contenido de AREA, después del siguiente proceso:

```
AREA     DS    F
A        DC    F'15'
B        DC    F'18'
         LM    1,2,A
         SRA   2,1
         AR    1,2
         SLDA  0,1
         AR    1,2
         ST    1,AREA
```

k) Indicar el contenido de DOSB al terminar el siguiente proceso:

```
DØSB     DS    H
CUATRØ   DC    F'4'
ØCHØ     DC    F'8'
C256     DC    F'256'
CERØ     DC    H'0'
C15      DC    H'15'
         L     1,CUATRØ
         L     15,C256
         LA    2,40(1,15)
         L     3,CERØ
         AH    3,C15
         SLR   2,3
         STH   2,DØSB
```

l) Indicar el contenido de Z después del proceso siguiente:

```
A        DC    F'37'
B        DC    F'-42'
C        DC    H'-14'
Z        DS    F
         L     1,A
         LH    2,C
         LH    3,B
         MR    0,2
         DR    0,3
         AL    1,B
         SL    1,C
         ST    1,Z
```

ll) Indicar qué queda en Z después del siguiente proceso:

```
A        DC    F'-18'
         DC    H'0'
```

```
        DC    H'35'
B       DC    F'42'
Z       DS    F
        LH    1,B
        AH    1,A
        L     2,A+4
        AR    2,1
        S     2,B
        LH    3,A+4
        STH   3,Z
        STH   2,Z+2
```

m) Programar la expresión:

$$Z = |X| - 2|X - |Y||$$

n) Indicar lo que queda en Z después del siguiente proceso:

```
A       DC    F'57'
        DC    F'-38'
B       DC    H'32'
        DC    H'-25'
C       DC    F'0'
Z       DS    F
        LM    1,2,A
        LH    3,B
        CH    1,C
        BC    2,MAYØR
        ALR   1,3
        SR    1,2
MAYØR   STH   3,C+2
        A     2,C
        M     0,A+4
        D     0,C
        SH    1,B+2
        ST    1,Z
```

ñ) Programar el problema siguiente:

$$Z = \frac{A+B}{A-B} * 18 + X \qquad \text{si } A \leq 15$$

$$Z = \frac{A-B}{A+B} * 81 - X \qquad \text{si } 15 < A \leq 35$$

$$Z = \frac{(A+B)(A-B)}{2A - B} + X \qquad \text{si } A > 35$$

o) Indicar el contenido de RUG1 después del proceso siguiente:

```
A       DC    F'53'
B       DC    F'-48'
C       DC    F'15'
UNØ     EQU   4
```

```
         LM    1,3,A
         SR    0,0
         SLDA  0,32
         SRA   0,UNØ
         SLDA  0,32
         AR    2,3
         LPR   2,2
         SRA   2,UNØ
         AR    1,2
```

p) Indicar el contenido de A después del siguiente proceso:

```
A        DC    F'15'
B        DC    H'0'
C        DC    H'18'
         LM    1,2,A
         AH    1,B
         LTR   1,1
         BC    8,ALFA
         SLA   1,1
ALFA     ST    1,A
```

q) Programar el problema siguiente:

SL = SB*1.40-DL

Si SL <505 desplazar el <RUG 5>= 10 un lugar a la izquierda, en caso contrario, desplazarlo un lugar a la derecha y el resultado almacenarlo en CONTROL.

r) El descuento por Seguro Social en los EE. UU. es el 3 1/8 por ciento de los ingresos hasta US$ 4800 en un año. Dados los ingresos acumulados hasta la fecha, calculados con anterioridad (IAF) y sus entradas de la presente semana, calcular el descuento sobre sus ingresos de esta semana y los nuevos ingresos acumulados hasta el momento (NIA). Deben considerarse las siguientes posibilidades:

i) La persona ha ganado US$ 4800 o más, antes de esta semana, en cuyo caso el descuento es cero.

ii) La persona no ha ganado US$ 4800 incluso con su ingreso de esta semana, en cuyo caso el descuento es de 3 1/8 por ciento de las ganancias de esta semana.

iii) Antes de esta semana, la persona no ha ganado US$ 4800, pero sí, al incluir lo que ha ganado en ella. En este caso, el descuento es de 3 1/8 por ciento de la diferencia entre US$ 4800 y sus ingresos previos acumulados.

s) Se tienen tres datos A, B y C, ocupando ocho bytes cada uno, en el formato ZONA–DIGITO. Se pide calcular:

R = A + B – C

4. *Ensamblado y pseudo instrucciones*

4.1. *Términos y expresiones*

Cada término representa un valor. Este valor puede ser asignado por el ensamblador o puede estar implícito en el término (términos autodefinidos o literales).

Ejemplo 37.

15,4092,X'AB4',X'FF' son términos autodefinidos.

Expresiones: Hay dos tipos de expresiones:

- ABSOLUTA
- REUBICABLE

Una expresión estará formada por un término o un conjunto de términos relacionados entre sí por operadores aritméticos.

Reglas para construir expresiones:

a) No puede empezar con operador aritmético

b) Los términos deben separarse por operadores o paréntesis

c) No puede haber dos o más operadores seguidos

d) No puede tener más de 16 términos

e) No puede haber más de 5 niveles de paréntesis

f) Una expresión de varios términos no puede contener un literal

g) La evaluación de las expresiones se realiza de izquierda a derecha, con prioridad de multiplicaciones y divisiones sobre sumas y restas.

h) El resultado de la división es entero. Si se divide por cero, el resultado es cero.

A. *Expresión absoluta*

Una expresión es absoluta si su valor no cambia al ser reubicado el programa.

Una expresión absoluta puede tener términos reubicables, solos o en combinación con términos absolutos, con las siguientes restricciones:

a) Debe haber un número par de términos reubicables (R) en la expresión.

b) Los términos reubicables deben estar pareados, esto es, debe existir un término reubicable con signo + y un término reubicable con signo —. Los términos pareados no tienen que estar contiguos obligadamente.

c) Los términos pareados deben entrar en esa forma, en multiplicaciones o divisiones. Así, R-R*10 no es válido, en cambio, (R-R)*10 es válido.

B. *Expresión reubicable*

Una expresión es reubicable si su valor cambia en *n* al ser desplazado el programa en memoria (reubicado) *n* bytes desde su punto de origen.

Una expresión reubicable puede tener términos reubicables, solos o en combinación con términos absolutos, con las siguientes restricciones:

a) Debe haber un número impar de términos reubicables (R)

b) Todos los términos reubicables, salvo uno, deben estar pareados

c) El término no pareado no puede estar precedido por signo menos

d) No puede haber términos reubicables en una multiplicación o división.

4.2. *Seccionamiento de programas*

Un programa grande puede ser subdividido en secciones llamadas secciones de control (Control Section). Las secciones pueden ser traducidas en forma independiente y después combinadas formando un solo programa objeto. Una sección de control es un conjunto de instrucciones que puede ser reubicada independientemente de otra sección de control. Normalmente la identificación de una sección de control es realizada a través de la instrucción CSECT. La primera sección de control puede ser identificada también con la instrucción START.

La combinación de las secciones de control se puede realizar debido a que el ensamblador cuenta con un Contador de Direcciones (Location Counter) para cada sección de control. Aquéllas son asignadas a ubicaciones de partida consecutivas, de acuerdo al orden en que se van sucediendo en el programa. Cada sección de control posterior a la primera, empieza en la siguiente doble palabra disponible.

El Contador de Direcciones (CD) es inicializado con la instrucción START. Antes de generarse una instrucción, una constante o un área, el CD se ajusta al alineamiento apropiado para el ítem, si es que el ajuste es necesario. Después de traducido el ítem se incrementa el valor contenido en el CD en la longitud del ítem. Así, siempre indica al ensamblador la próxima posición disponible. Si la instrucción es nombrada por un símbolo, el valor atribuido del símbolo es el valor contenido en el CD después del ajuste al alineamiento, pero antes de sumar la longitud. Tales símbolos son siempre reubicables.

La primera sección de control de un programa tiene las siguientes propiedades:

a) El valor inicial de su CD puede ser especificado como un valor absoluto.

b) Normalmente contiene los literales (ver literales) requeridos en el programa.

Un programa no seccionado es tratado como una sección de control única.

A. *Pseudo Instrucción* CONTROL SECTION

a) Código : CSECT

b) Formato: [símbolo] CSECT sin operandos

c) Función : Permite identificar el comienzo o la continuación de una sección de control. No genera instrucción de máquina. El símbolo es establecido como el nombre de la sección de control; en caso contrario la sección es considerada sin nombre. Todas las instrucciones que siguen a la CSECT son traducidas como parte de esa sección de control, hasta que se encuentre otra CSECT o una pseudo instrucción DSECT.

Si aparecen varias pseudo instrucciones CSECT con el mismo nombre, se considera que la primera identifica el comienzo de la sección y las demás identifican reanudaciones de ella.

B. *Pseudo Instrucción* START

a) Código : START

b) Formato: [símbolo] START término autodefinido o blanco

c) Función : Permite identificar la primera o única sección de control. Además inicializa el Contador de Direcciones. No genera instrucción de máquina.

El símbolo es establecido como el nombre de la sección de control; en caso contrario la sección es considerada sin nombre. El valor del término autodefinido es cargado en el CD. Debe ser múltiplo de 8 para partir con almacenamiento de doble palabra. Si no se coloca operando autodefinido se carga 0 en el CD.

Ninguna instrucción que dependa del contenido del CD puede estar antes de START.

C. *Secciones de Control sin nombre*

Puede haber sólo una sección de control sin nombre en un programa. Si aparece una sección de control sin nombre y es seguida por secciones de control con nombre, cualquier otra sin nombre se considera reanudación de la primera.

D. *Pseudo Instrucción* USING

Permite indicar al ensamblador: el o los registros de uso general que puede utilizar como Registros Base y además cuáles son sus contenidos supuestos.

a) Código : USING

b) Formato: blanco USING expresión, r1,r2,...,rn

c) Función : Indica al ensamblador que puede utilizar como registros base aquellos indicados en los operandos r1,r2,...,rn. Especifica además cuáles serán los contenidos supuestos de esos registros.

El RUG indicado en r1 tiene como contenido supuesto el valor de la expresión. El que se indica en r2 tendrá el valor de la expresión + 4096. El que se indica en *rn* tendrá el valor de la expresión + (n−1)* 4096.

Ejemplo 38.

```
USING .          FIRST,4
```

Indica al ensamblador que puede utilizar el RUG4 como registro base y el contenido que debe suponer en él es el valor de la expresión FIRST.

Como expresión se puede utilizar el signo * (asterisco) que representa el valor contenido en el CD en el momento de analizarse el signo.

Ejemplo 39.

```
USING            *,15
```

Indica al ensamblador que puede utilizar el RUG15 como registro base y el contenido que debe suponer en él es a su vez el contenido del CD.

E. *Pseudo Instrucción* DROP

a) Código : DROP

b) Formato : blanco DROP r1,r2,...,rn

c) Función : Indica al ensamblador que no puede utilizar como registros base aquellos indicados en los operandos r1,r2,...,rn. Si el RUG indicado no se estaba utilizando, la instrucción no tiene efecto.

F. *Sección Formal (Dummy Section)*

Una sección formal representa una sección de control que es ensamblada pero que no es parte del programa objeto. Se puede describir así la estructura de un área de almacenamiento sin que se reserve dicha área (se supone que se reserva área de almacenamiento por alguna parte del mismo o de otro ensamblado).

Explicación. Se carga el RUG5 con la dirección SALTO. Se carga en el RUG7 la segunda mitad de la PSW; en la parte de dirección queda 10602. Se efectúa el salto a la dirección indicada en RUG5.

Nota: La pseudo instrucción USING se vió que permite indicar al ensamblador que registros puede utilizar como BASE y cuáles son sus contenidos supuestos. En la etapa de ejecución, esos contenidos deben ser cargados en los respectivos RUG. La instrucción BALR es una de las que permite cumplir ese objetivo.

Ejemplo 42.

```
                START   2000
                BALR    12,0
                USING   *,12
                BC      15,SIGA
JUAN            DC      F'0'
SIGA            ——
                ——
                ——
```

Explicación. En la etapa de traducción en una primera pasada de análisis del programa, se asignan valores a los símbolos de acuerdo con el operando de la pseudo instrucción START.

```
                START   2000
2000            BALR    12,0
                USING   *,12
2002            BC      15,SIGA
2008 JUAN       DC      F'0'
2012 SIGA       ——
                ——
                ——
```

Al asignar el valor al símbolo JUAN, el CD se ajusta al alineamiento de palabra necesario para generar la constante.

En la segunda pasada de análisis del programa se terminan de generar instrucciones constantes y áreas:

a) Se genera la instrucción BALR 12,0

b) La pseudo instrucción USING indica al ensamblador que puede utilizar como registro BASE el 12, y como contenido supuesto el valor que tiene el CD en ese momento, es decir, 2002

c) Se genera la instrucción BC 15,SIGA

i) La máscara 15 figura en el campo M1.

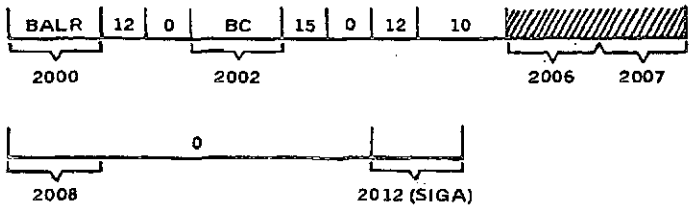
ii) Se coloca 0 en el campo X2, porque no se indica en la instrucción uso de registro INDICE.

iii) El valor del símbolo SIGA (2012) se distribuye entre <RUG B> y D.

Valor de SIGA = <RUG 12> + D2

2012 = 2002 + D2
D2 = 10

iv) Se tiene así:



H. *Instrucción* BRANCH AND LINK (Instrucción de bifurcación)

a) Instrucción: BAL R1,D2(X2,B2)

b) Formato : RX | BAL | R1 | X2 | B2 | D2 |

c) Función : Se realiza la misma función que con BALR. El salto en este caso lo efectúa a la dirección dada por D2(X2,B2).

I. *Normas que sigue el ensamblador para asignar Registro Base*

a) El contenido supuesto del registro que va a utilizar debe ser siempre menor o igual que la dirección que se va a conformar.

b) Si hay dos o más registros, utiliza aquel que permite menor desplazamiento en la instrucción compilada.

c) Si hay dos o más registros que tienen igual contenido supuesto, elige el mayor de ellos.

4.3. *Constantes de Dirección*

Una constante de dirección corresponde a una dirección de memoria almacenada como una constante. Normalmente se utilizan para inicializar registros base.

La constante de dirección es encerrada entre paréntesis. Si hay dos o más constantes, se separan entre sí por coma y el conjunto completo se encierra entre paréntesis.

Hay cuatro tipos de constantes de dirección: A, Y, S y V.

a) Tipo A: El valor máximo puede ser $2^{31}-1$, ocupa 4 bytes y se almacena en alineamiento de palabra completa. La constante puede ser especificada como una expresión absoluta, o una expresión reubicable. Lo normal es lo segundo, luego, al ser reubicado el programa, las constantes de dirección cambian de valor en base al nuevo punto de carga del programa.

b) Tipo Y: Similar al tipo A, difiere sólo en que su valor máximo puede ser $2^{15}-1$, o sea, ocupa dos bytes.

c) Tipo S: se utiliza para almacenar una dirección en la forma BASE–DESPLAZAMIENTO. Ocupa dos bytes con alineamiento de media palabra.

Se puede especificar en dos formas:

i) Como una expresión absoluta o reubicable. Ej. S(BETA)

ii) Como dos expresiones absolutas, siendo la primera el desplazamiento y la segunda el registro base. Ej. S(400(13)).

d) Tipo V: Se utiliza para reservar área para la dirección de un símbolo externo que será meta de salto. El símbolo debe ser especificado como símbolo reubicable y necesita ser declarado explícitamente como externo.

(Ver pseudo instrucción EXTRN) Ocupa 4 bytes con alineamiento de palabra completa.

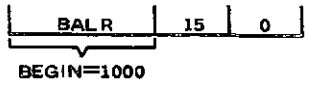
Ejemplo 41.

```
        PRØGA   START   1000
1000    BEGIN   BALR    15,0
                USING   FIRST,15
1002    FIRST   BC      15,SKP
1008    DATA    DC      F'3472'
1012    BASE1   DC      A(FIRST+4096)
1016    BASE2   DC      A(FIRST+2*4096)
                --
                --
1102    SKP     L       14,BASE1
                USING   FIRST+4096,14
                L       13,BASE2
                --
                --
                --
                USING   FIRST+2*4096,13
2000            BC      15,CK8
                --
                --
                --
3000    LØØP    A       4,DATA
                --
                --
                --
4000    LØØPB   S       5,DATA
                --
                --
                --
10000           BC      8,LØØP
                --
                --
```

```
11000   CK8     BC      8,LØØPB
                END     BEGIN
```

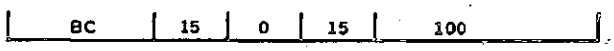
Los números que aparecen a la izquierda del programa corresponden a los valores tentativos asignados por el ensamblador en la primera pasada de análisis. Los pasos de la segunda pasada son los siguientes:

a) Se genera la instrucción BALR 15,0



b) La pseudo-instrucción USING indica al ensamblador que puede utilizar como registro BASE el 15 y como contenido supuesto, el valor de FIRST, esto es, 1002.

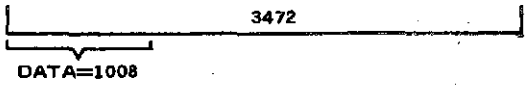
c) Se genera la instrucción BC 15,SKP



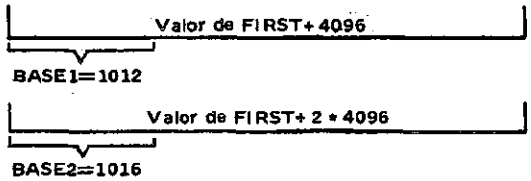
el desplazamiento se obtiene de:

$$\begin{aligned} \text{Valor de SKP} &= \langle \text{RUG15} \rangle_8 + D2 \\ 1102 &= 1002 + D2 \\ D2 &= 100 \end{aligned}$$

d) Se genera la constante 3472. Se saltan dos bytes para ajustarse al alineamiento adecuado.

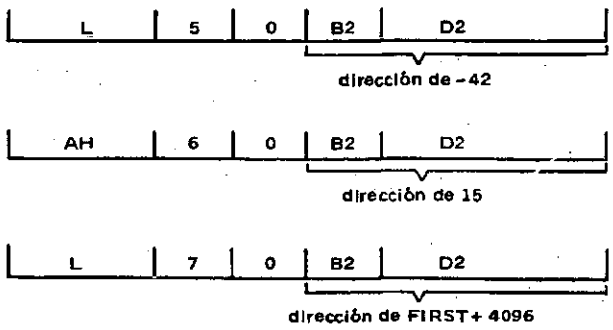


e) Se generan las constantes de dirección



f) Para generar la instrucción de nombre SKP se dispone sólo del RUG15 como BASE con contenido supuesto igual a 1002.

$$\begin{aligned} \text{Valor de BASE1} &= \langle \text{RUG 15} \rangle_8 + D2 \\ 1012 &= 1002 + D2 \\ D2 &= 10 \end{aligned}$$



Se puede observar que las direcciones de los literales no son conocidas por el programador, en otras palabras, no es conocida la dirección de origen del área de literales. Mediante la pseudo instrucción Literal Origen el programador puede controlar la primera posición del área de literales.

A. *Pseudo instrucción Literal Origen*

a) Código : LTORG

b) Formato: [símbolo] LTORG sin operandos

c) Función : Agrupa todos los literales, definidos desde la instrucción LTORG anterior o desde el comienzo del programa si no existe otra, inmediatamente después de su aparición. Si se especifica símbolo, él corresponde a la dirección del primer byte del área de literales. No genera instrucción de máquina.

Ejemplo 44.

```
START  0
 ---
 ---
 ---
L      7,=F'900'
 ---
 ---
 ---
SH     7,=H'52'
 ---
 ---
 ---
AR     5,2
LTØRG
M      4,DATØS
```

```
      —
      —
      —
      END
```

Al ser ensamblado el programa anterior, quedará en memoria la siguiente información:

```
AR        5,2
          =F'900'
          =H'52'
M         4,DATØS
```

Esto produce error en la ejecución, dado que después de la instrucción AR 5,2 aparecen constantes. La forma de solucionar el problema será:

```
          AR      5,2
          BC      15,SIGA
          LTØRG
SIGA      M       4,DATØS
```

con lo cual se obtiene:

```
          AR      5,2
          BC      15,SIGA
                  =F'900'
                  =H'52'
SIGA      M       4,DATØS
```

Si en el programa se desea hacer uso de la dirección de la constante 900, se puede utilizar como punto de referencia la instrucción BC 15,SIGA en la forma siguiente:

```
REF       BC      15,SIGA
                  LTØRG
SIGA      M       4,DATØS
          —
          —
          A       9,REF+4
```

Existe, sin embargo, la posibilidad del problema siguiente: la instrucción BC 15,SIGA se genera a partir de una dirección múltiplo de dos, pero no de cuatro. Para generarse la constante 900 se deben saltar dos bytes. Luego al especificar A 9,REF+4 se indicará error de DIRECCIONAMIENTO dado que REF+4 no es múltiplo de cuatro.

Para evitar situaciones de este tipo, se hace uso de la pseudo instrucción Conditional No Operation (CNOP) que permite ajustar el contenido del Contador de Direcciones (Location Counter) a una posición determinada.

B. *Pseudo instrucción Conditional no Operation*

a) Código : CNOP

b) Formato: [símbolo] CNOP b,p

c) Función : Se ajusta el contenido del Contador de Direcciones al alineamiento especificado por los operandos *b* y *p* donde:

b: indica el byte de un conjunto *p* de bytes

p: indica una palabra (4 bytes) o una doble palabra (8 bytes)

Los valores posibles son:

b,p	Indicación
0,4	comienzo de una palabra
2,4	comienzo de la 2a. mitad de la palabra
0,8	comienzo de una doble palabra
2,8	comienzo de la 2a. mitad de la 1a. palabra
4,8	comienzo de la 2a. palabra
6,8	comienzo de la 2a. mitad de la 2a. palabra

Si el Contador de Direcciones está ajustado al alineamiento que se indica con CNOP, la pseudo instrucción no tiene efecto. En caso contrario, se generan tantas instrucciones de "no operación" como sea necesario (BCR 0,0).

Ejemplo 45

14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1
6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5

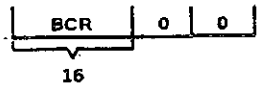
Se puede observar que los bytes 0,4 corresponden a direcciones múltiplo de 4, y los bytes 0,8 a direcciones múltiplo de 8.

a) <CD> = 16
CNOP 0,4

la pseudo instrucción no tiene efecto

b) <CD> = 16
CNOP 2,4

se genera, a partir de 16, una instrucción de no operación

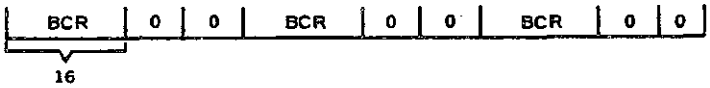


c) <CD> = 16
CNOP 0,8

la pseudo instrucción no tiene efecto

d) <CD> = 16
CNOP 6,8

se generan, a partir de 16, tres instrucciones de no operación



La solución al problema planteado al tratar de direccionar literales será:

```
         CNØP   0,4
REF      BC     15,SIGA
         LTØRG
SIGA     M      4,DATØS
         --
         --
         --
         A      9,REF+4
```

4.5. *Pseudo Instrucciones de control de listado*

Son utilizadas para colocar título a los listados de programas fuente, para insertar líneas o páginas en blanco, para imprimir las macro-instrucciones en detalle o sólo su nombre, etc. No generan instrucción de máquina, no se incluyen en el listado, excepto PRINT que aparece impresa.

A. *Pseudo instrucción* TITLE

a) Código : TITLE

b) Formato : [símbolo] TITLE 'uno a cien caracteres'

c) Función : El símbolo está formado por 1 a 4 caracteres alfanuméricos. Dicho símbolo se perfora en las columnas 73-76 de todas las tarjetas de resultado de en-

samble. Puede haber varias pseudo instrucciones TITLE en un programa, pero sólo la primera de ellas puede tener especificado el símbolo en el campo de nombre.

Los caracteres que aparecen entre apóstrofos se imprimen como títulos en la primera línea de cada página del listado del programa fuente. Si el conjunto de caracteres contiene apóstrofos o ampersands (&) ellos deben colocarse como un par, cada vez. Cada pseudo instrucción TITLE proporciona el encabezamiento para las páginas que le siguen. Exceptuando la primera pseudo instrucción TITLE, las restantes implican el salto a una nueva página antes de continuar el listado.

Si el título excede la columna 71, se perfora carácter de continuación en la 72, y se reinicia el título en la columna 16 de la tarjeta siguiente.

B. *Pseudo instrucción* EJECT

a) Código : EJECT

b) Formato : blanco EJECT sin operandos

c) Función : La pseudo instrucción EJECT causa que la siguiente línea del listado sea impresa como primera línea de la página siguiente. Si la siguiente línea es la primera de la página siguiente, la pseudo instrucción no tiene efecto. Para saltar n páginas debe especificarse $n+1$ EJECT seguidos.

C. *Pseudo instrucción* SPACE

a) Código : SPACE

b) Formato : blanco SPACE blanco o valor decimal

c) Función : Se utiliza para insertar en el listado un número de líneas en blanco especificado por el valor decimal. Si no se especifica valor, se inserta una línea en blanco. Si el valor especificado excede el número de líneas restantes en la página, tiene el efecto de una pseudo instrucción EJECT.

D. *Pseudo instrucción* PRINT

a) Código : PRINT

b) Formato : blanco PRINT $\left[\left\{\begin{matrix}\underline{\text{ON}}\\ \text{OFF}\end{matrix}\right\}\left\{\begin{matrix}\underline{\text{GEN}}\\ \text{NOGEN}\end{matrix}\right\}\left\{\begin{matrix}\underline{\text{DATA}}\\ \text{NODATA}\end{matrix}\right\}\right]$

Observación: El paréntesis $\left\{ \quad \right\}$ indica que hay que elegir uno de los operandos encerrados. Aquellos que aparecen subrayados, si no se colocan se suponen.

c) Función : De acuerdo a los operandos especificados se obtiene lo siguiente:

i)	ON	Se imprime el listado del programa fuente
ii)	OFF	No se imprime el listado
iii)	GEN	Se imprimen las macro-instrucciones en detalle, esto es, con todas las instrucciones que la componen
iv)	NOGEN	Se imprime sólo la llamada de la macro en la forma siguiente: [nombre] operación operandos
v)	DATA	Se imprimen las constantes definidas en el programa en toda su longitud
vi)	NODATA	Se imprimen solamente los 8 bytes de orden superior de las constantes.

4.6. *Pseudo instrucciones de control del programa*

Las pseudo instrucciones de control de programa permiten indicar final de ensamble, ajustar el contador de direcciones a un valor determinado, etc. Se han visto ya de este grupo, las pseudo instrucciones LTORG y CNOP. Otras pseudo instrucciones de importancia son END, ICTL, ISEQ entre las más usadas.

A. *Pseudo instrucción* END (Final de Ensamble)

a) Código : END

b) Formato : blanco END blanco o expresión reubicable

c) Función : Indica el término de las instrucciones que van a ser ensambladas. También puede designar a través del operando, un punto en el programa traducido, o en otro, al cual se transferirá el control después que el programa sea cargado a memoria para su ejecución.

B. *Pseudo instrucción* ICTL. *Control del formato de entrada* (Input Format Control).

a) Código : ICTL

b) Formato : blanco ICTL uno a tres valores decimales en la forma c,t,r

c) Función : La pseudo instrucción permite al programador, alterar el formato normal de las instrucciones de assembler, esto es, comienzo en columna 1, tér-

c) Suponiendo que el programa anterior se ejecute a partir de la dirección 56, ¿qué queda en los RUG 2 y 3?

d) Compilar el programa siguiente:

```
         START   400
         BALR    15,0
         USING   *,15
         BC      15,DIEGØ
DIR1     DC      A(DIEGØ+4)
DIR2     DC      A(DIEGØ)
DIEGØ    LM      0,1,DIR1
         BALR    2,0
         SR      2,1
         ST      2,PEDRØ
         LA      1,4(2,0)
         BC      15,ØUT
PEDRØ    DS      F
ØUT      EØJ
         END
```

e) Al procesar el programa anterior, indicar qué queda en RUG1 y en PEDRO, al finalizar el proceso, suponiendo que el programa se carga a partir de la dirección 400.

f) ¿Qué queda en los RUG 1 y 2 al terminar el proceso siguiente?

```
         START   256
         BALR    15,0
         USING   *,15
         SPACE   3
JUAN     EQU     1
         LA      1,*+2
         LA      2,*+2
         SR      2,1
         SLA     1,JUAN
```

g) Indicar qué queda en los bytes DATOS+3, DATOS+5 y DATOS+7 al término del proceso siguiente:

```
         ICTL    1
PRØG     TITLE   'PRUEBA'
         START   X'2800'
         BALR    15,0
         ISEQ    73,75
         USING   *,15
         BALR    10,0
         BALR    11,0
         STM     10,11,DATØS
         BC      15,ØUT
DATØS    DS      2F
ØUT      EØJ
         END
```

h) Suponiendo que el programa que figura a continuación pudiera ser procesado a partir de la dirección indicada en START, ¿cuál sería el contenido de los RUG 1, 2, 3 y 4 al finalizar el proceso?

```
         START  400
         BALR   15,0
         USING  *,15
         BC     15,GØ
JUAN     DS     0H
PEDRØ    DC     X'002'
DIEGØ    DC     HL2'+3'
GØ       LH     1,DIEGØ
         LH     2,DIEGØ-2
         LH     3,PEDRØ
         LH     4,JUAN
         AR     4,3
         AR     4,2
         A      4,=A(JUAN)
         SR     4,1
         EØJ
         END
```

i) En las mismas condiciones del problema anterior, indicar qué queda en AREA después del siguiente proceso:

```
         START  400
         BALR   15,0
         USING  *,15,14
FIRST    BC     15,BEGIN
AD1      DC     A(FIRST+4096)
AREA     DS     2F
BEGIN    L      14,AD1
         L      1,=F'5'
         L      2,=A(*)
         LR     3,1
         BC     15,SIGUE
         LTØRG
SIGUE    AR     1,14
ØCHØ     EQU    6
         SRL    1,9
         SR     2,1
         SRL    2,ØCHØ
         SRDL   2,96
         STM    2,3,AREA
         EØJ
         END
```

5. *Instrucciones lógicas*

5.1. *Instrucción* MOVE CHARACTER

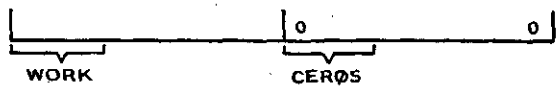
a) Instrucción : MVC D1(L,B1),D2(B2)

b) Formato : SS | MVC | L | B1 | D1 | B2 | D2 |

c) Función : La información contenida a partir de la dirección D2(B2) es transferida al campo indicado por D1(B1). Se transfieren L bytes (máximo 256). La operación se realiza de izquierda a derecha a través de cada campo, byte por byte, los cuales no se cambian ni inspeccionan.

Ejemplo 46

```
WØRK    DS      CL80
CERØS   DC      80C'0'
        MVC     WØRK(80),CERØS
```



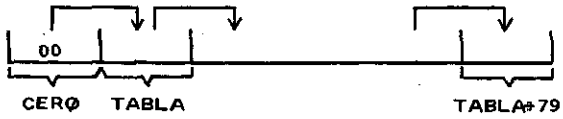
Observación: El símbolo WØRK tiene implícita, en su definición la longitud 80. Luego puede especificarse:

```
        MVC     WØRK,CERØS
```

con idéntico resultado.

Ejemplo 47.

```
CERØ    DC      X'0'
TABLA   DS      CL80
        MVC     TABLA,CERØ
```



En este caso, se propaga el byte CERØ (su contenido) a lo largo de todo el campo TABLA, dado que la instrucción opera de izquierda a derecha, byte por byte. Primero se transfiere el contenido del byte CERØ al byte TABLA, a continuación el contenido de TABLA, que ahora tiene ceros, al byte TABLA+1, y así sucesivamente.

5.2. *Instrucción* MOVE IMMEDIATE

a) Instrucción : MVI D1(B1),I2

b) Formato : SI | MVI | I2 | B1 | D1 |

c) Función : El contenido del segundo byte de la instrucción (operando inmediato), se transfiere a la dirección D1(B1).

Ejemplo 48.

```
MVI     PEDRØ,C'A'
MVI     JUAN,X'C1'
MVI     DIEGØ,193
```

La primera instrucción transfiere a la dirección PEDRØ, el carácter A, que en binario es 11000001. La segunda instrucción transfiere a la dirección JUAN, el valor hexadecimal C1, que en binario es 11000001. La última instrucción transfiere a la dirección DIEGØ, el valor decimal 193, que en binario es 11000001. Esto es, tres formas distintas para obtener igual resultado.

Ejemplo 49.

Se desea dejar el campo BLANCØS con el carácter ß. BLANCØS tiene longitud 10 bytes.

```
MVI     BLANCØS,C' '
MVC     BLANCØS+1(9),BLANCØS
```

5.3. *Instrucción* MOVE NUMERICS

a) Instrucción : MVN D1(L,B1),D2(B2)

b) Formato : SS

MVN	L	B1	D1	B2	D2

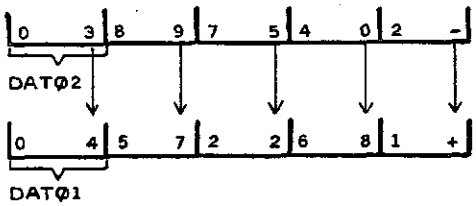
c) Función : Los cuatro bits de orden inferior de cada byte del campo indicado por D2(B2) se transfieren a las posiciones de los cuatro bits de orden inferior de los correspondientes bytes del campo D1(B1). Se operan L bytes.

La transferencia es de izquierda a derecha a través de cada campo, byte por byte. Los campos pueden superponerse en cualquier forma que se desee.

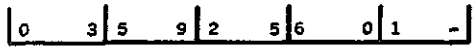
La parte que se transfiere, parte numérica, no sufre cambio ni es sometida a verificación de validez. Los cuatro bits de orden superior de cada byte, parte zona, no sufren cambio.

Ejemplo 50.

```
DATØ1   DC      P'45722681'
DATØ2   DC      P'-38975402'
        MVN     DATØ1,DATØ2
```



resultado



5.4. *Instrucción* MOVE ZONES

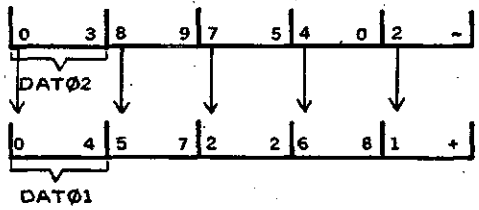
a) Instrucción : MVZ D1(L,B1),D2(B2)

b) Formato : SS | MVZ | L | B1 | D1 | B2 | D2 |

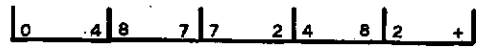
c) Función : Similar a la instrucción anterior, transfiere los bits de orden superior, parte zona. Los cuatro bits de orden inferior permanecen invariables en ambos operandos.

Ejemplo 51.

```
DATØ1  DC    P'45722681'
DATØ2  DC    P'-38975402'
       MVZ   DATØ1,DATØ2
```



resultado



5.5. *Instrucción* MOVE WITH OFFSET (aritmética decimal)

a) Instrucción: MVO D1(L1,B1),D2(L2,B2)

b) Formato : SS | MVO | L1 | L2 | B1 | D1 | B2 | D2 |

c) Función : El segundo operando se transfiere al campo que está a la izquierda y contiguo a los cuatro bits de orden inferior del primer operando.

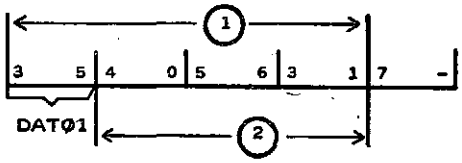
El procesamiento de los campos se realiza de derecha a izquierda, byte por byte. Si el

campo del primer operando no contiene al segundo operando, la información excedente se ignora. El caso contrario, esto es, campo del primer operando más largo, produce la extensión del segundo operando mediante ceros de orden superior. Puede producirse superposición de campos, y su procesamiento se realiza almacenando un byte de resultado tan pronto como hayan sido extraídos los bytes necesarios, de los operandos.

Ejemplo 52.

```
         START  0
         BALR   15,0
         USING  *,15
         B      BEGIN
AREA     DS     2PL10
DATØ1    DC     P'-354056317'
DATØ3    DC     P'182345903'
BEGIN    MVØ    DATØ1(4),DATØ1+1(3)
         MVC    AREA(5),DATØ1
         MVØ    DATØ1+1(4),DATØ1(3)
         MVC    AREA+5(5),DATØ1
         MVØ    DATØ3(2),DATØ3+1(3)
         MVC    AREA+10(5),DATØ3
         MVØ    DATØ3(3),DATØ3+2(2)
         MVC    AREA+15(5),DATØ3
         SVC    14
         END
```

Desarrollo de las instrucciones MVØ



① campo receptor

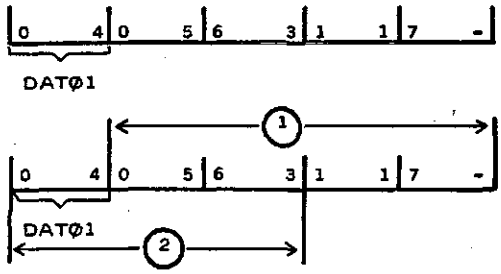
② campo que se transfiere

i) el byte del extremo derecho del campo receptor es DATO1+3, y su contenido es 31.

ii) los 4 bits de orden inferior de DATO1+3 contienen el 1 que permanecerá inalterable.

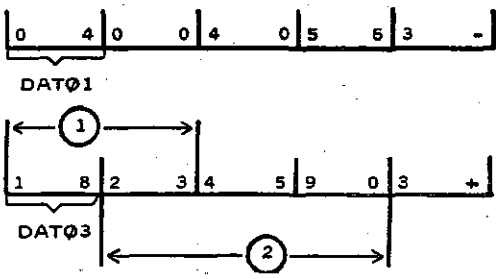
iii) a la izquierda y adyacente al 1 queda el campo que se transfiere.

iv) resultado:



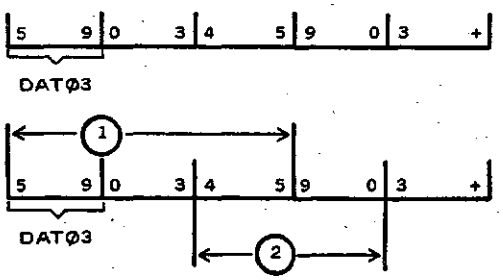
i) Todo el campo que se transfiere se coloca a la izquierda y adyacente al signo menos (byte del extremo derecho del campo receptor).

ii) Resultado:



i) El campo receptor puede recibir sólo los dígitos 590, el resto se pierde.

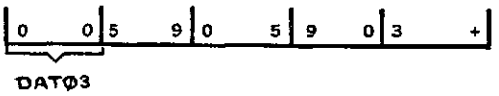
i) Resultado:



i) El primer byte que se transfiere es DATO3+3 que se coloca a la izquierda y adyacente al dígito 5.

ii) El segundo byte que se transfiere es DATO3+2 cuya información original ha sido modificada en el paso anterior.

iii) Resultado:



5.6. *Instrucción* INSERT CHARACTER

a) Instrucción : IC R1,D2(X2,B2)

b) Formato : RX

IC	R1	X2	B2	D2

c) Función : El contenido del byte direccionado con D2(X2,B2) se inserta en el RUG especificado en R1, en los bits 24 a 31. Los bits restantes del registro no sufren cambio.

Ejemplo 53.

```
C255    DC    X'FF'
        SR    5,5
        IC    5,C255
```

Análisis:

i) SR 5,5 deja el RUG en cero

ii) IC 5,C255 inserta en el RUG5, en los bits 24 a 31, un byte con unos.

iii) resultado

0	00111111111
0	24 31

<RUG5> = 255

5.7. *Instrucción* STORE CHARACTER

a) Instrucción : STC R1,D2(X2,B2)

b) Formato : RX

STC	R1	X2	B2	D2

c) Función : El contenido de los bits 24 a 31 del RUG especificado en R1, se almacena en la dirección D2(X2,B2).

Ejemplo 54.

```
DATØS   DS    CL3
        —
        —
        --
        LA    5,150
        LA    6,45
        L     8,=F'247'
        STC   5,DATØS
        STC   6,DATØS+1
        STC   8,DATØS+2
```

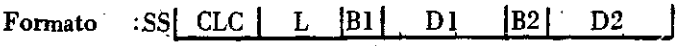
Análisis:

i) Las instrucciones LA y L cargan en los RUG 5, 6 y 8 los valores 150, 45 y 247 respectivamente. Todos contenidos en un byte.

5.12. *Instrucción* COMPARE LOGICAL CHARACTER

a) Instrucción : CLC D1(L,B1),D2(B2)

b) Formato :



c) Función : Compara el primer operando, ubicado en la dirección D1(B1), con el segundo operando, almacenado en D2(B2). La operación se realiza de izquierda a derecha a través de L bytes. La comparación es binaria y todos los códigos son válidos. Al detectarse desigualdad o término del campo, la operación se detiene y se genera Código de Condición.

Código de Condición	*Resultado*
0	Operando 1 = Operando 2
1	Operando 1 < Operando 2
2	Operando 1 > Operando 2

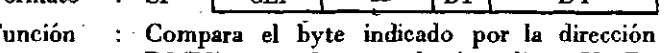
Se puede comparar todo tipo de caracteres y el resultado será de acuerdo al siguiente valor relativo de menor a mayor:

b <caract.especiales <caract.alfabéticos <caract.numéricos

5.13. *Instrucción* COMPARE LOGICAL IMMEDIATE

a) Instrucción : CLI D1(B1),I2

b) Formato :

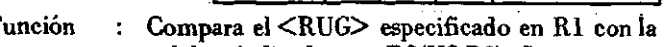


c) Función : Compara el byte indicado por la dirección D1(B1) con el operando inmediato I2. De acuerdo al resultado se genera Código de Condición.*

5.14. *Instrucción* COMPARE LOGICAL

a) Instrucción : CL R1,D2(X2,B2)

b) Formato :



c) Función : Compara el <RUG> especificado en R1 con la palabra indicada con D2(X2,B2). La comparación es binaria. La operación procede de izquierda a derecha y finaliza al detectar desigualdad o término de los operandos. De acuerdo al resultado se genera Código de Condición.*

5.15. *Instrucción* COMPARE LOGICAL REGISTER

a) Instrucción : CLR R1,R2

b) Formato : RR | CLR | R1 | R2 |

c) Función : Se compara el<RUG> especificado en el campo R1 con el<RUG> indicado en R2. No se considera el signo como tal. De acuerdo al resultado se genera Código de Condición.

* Los Códigos de Condición generados son los indicados en la instrucción Compare Logical Character.

Ejemplo 56. Ejemplos de instrucciones.

```
CLC     MASTER1(20),TRANS
CLI     LARGØ,16
CLI     BYTE,X'5'
CLI     ALFA,C'$'
CL      0,MASK
CLR     1,2
```

5.16. *Instrucción* TEST UNDER MASK

a) Instrucción : TM D1(B1),I2

b) Formato : SI | TM | I2 | B1 | D1 |

c) Función : El estado de los bits del byte cuya dirección es B1(D1) se investiga de acuerdo al contenido del operando inmediato I2 (máscara). La máscara se construye colocando "unos" en las posiciones que se desea investigar y "ceros" en las que no interesan. El contenido del byte analizado no se altera. De acuerdo al resultado se genera Código de Condición.

Código de Condición		*Resultado*
Decimal	Binario	
0	00	Todos ceros (OFF)
1	01	Mixto
3	11	Todos unos (ON)

Ejemplo 57.

```
BYTE    DC      X'54'
```

b) Formato : SS | OC | L | B1 | D1 | B2 | D2 |

L *Instrucción* EXCLUSIVE OR REGISTER

a) Instrucción : XR R1,R2

b) Formato : RR | XR | R1 | R2 |

J. *Instrucción* EXCLUSIVE OR

a) Instrucción : X R1,D2(X2,B2)

b) Formato : RX | X | R1 | X2 | B2 | D2 |

K. *Instrucción* EXCLUSIVE OR IMMEDIATE

a) Instrucción : XI D1(B1),I2

b) Formato : SI | XI | I2 | B1 | D1 |

L. *Instrucción* EXCLUSIVE OR CHARACTER

a) Instrucción : XC D1(L,B1),D2(B2)

b) Formato : SS | XC | L | B1 | D1 | B2 | D2 |

Del análisis de las instrucciones se puede concluir lo siguiente:

Instrucciones AND permiten:

i) Mantener los contenidos de los bits del primer operando, colocando unos en los bits respectivos del segundo operando.

ii) Dejar en cero (OFF) los bits del primer operando, colocando ceros en los bits respectivos del segundo operando.

Instrucciones OR permiten:

i) Mantener los contenidos de los bits del primer operando, colocando ceros en los bits respectivos del segundo operando.

Instrucciones Exclusive OR permiten:

i) Mantener los contenidos de los bits del primer operando colocando ceros en los bits respectivos del segundo operando.

ii) Cambiar los contenidos de los bits del primer operando colocando unos en los bits respectivos del segundo operando.

Ejemplo 59.

```
BYTE    DC      X'F0'
```

a) Dejar los bits 1 y 4 en OFF

```
NI      BYTE,X'B7'
```

b) Dejar los bits 1 y 4 en ON

```
ØI      BYTE,X'48'
```

c) Cambiar de estado los bits 1 y 4

```
XI      BYTE,X'48'
```

```
a)  11110000      b)  11110000      c)  11110000
    10110111          01001000          01001000
    --------          --------          --------
    10110000          11111000          10111000
```

5.18. *Instrucción* TRANSLATE AND REPLACE

a) Instrucción:: TR D1(L,B1),D2(B2)

b) Formato : SS | TR | L | B1 | D1 | B2 | D2 |

c) Función : Se traduce la lista de longitud L bytes ubicada a partir de la dirección D1(B1), de acuerdo a una tabla de traducción cuya dirección inicial es D2(B2). Los bytes del primer operando reciben el nombre de bytes del *argumento* y los bytes de la tabla se denominan bytes de *función*.

Los bytes del primer operando se seleccionan uno a uno para su traducción, procediendo de izquierda a derecha. Interpretados como número binario se suman a la dirección D2(B2) y el resultado obtenido se usa como dirección del byte de función el cual reemplaza al byte de argumento.

Ejemplo 60.

Convertir todos los caracteres alfabéticos y especiales del código EBCDIC en ceros y los caracteres numéricos en su complemento.

```
CØNVERT  START  0
         BALR   15,0
         USING  °,15
         --
         —
         TR     DATØS,TABLA
         —
         —
         EØJ
DATØS    DS     CL120
TABLA    DC     240X'FO'
         DC     X'FAF9F8F7F6F5F4F3F2F1'
         DC     6X'F0'
         END
```

El valor máximo que puede contener un byte es 255, de tal

D2(B2) que están en formato empaquetado, se imprimen de acuerdo a los caracteres contenidos en el campo D1(B1) denominado PATRON. En la impresión es posible suprimir ceros no significativos, insertar comas y puntos decimales, insertar el signo menos o el símbolo de crédito, etc. *El resultado reemplaza* al patrón. Los caracteres fundamentales de éste son:

Carácter	*Significado*	*Representación*	
		Binaria	*Hexadecimal*
d	Seleccionador de dígitos	00100000	X'20'
(	Iniciador de significación	00100001	X'21'
)	Separador de campos	00100010	X'22'

i) El carácter seleccionador de dígitos determina que un dígito del dato o un carácter de relleno sea insertado en el campo de resultado.

ii) El carácter iniciador de significación cumple la misma función del anterior, pero deja indicado que los dígitos que le siguen son significativos.

iii) El carácter separador de campos identifica cada uno de los campos, debiendo construirse el patrón como para un campo nuevo. Se reemplaza por carácter de relleno.

iv) El carácter de relleno puede ser cualquiera y es el primer carácter del área patrón. Normalmente se utiliza:

b = blanco, cuya representación en binario es 01000000 y en hexadecimal es X'40'. Otro carácter posible es:

* = asterisco, cuya representación en binario es 01011100 y en hexadecimal es X'5C'.

v) El proceso se realiza de izquierda a derecha, un carácter por vez. El carácter a almacenar depende de tres factores:

El dígito de datos
El carácter del área patrón
El estado de un interruptor (switch) llamado trigger S

De acuerdo a ellos se puede obtener:

Expansión del dígito a formato zona
Dejar sin cambio el carácter del área patrón
Almacenar un carácter de relleno.

vi) El trigger S se pone en cero al comienzo de la operación y luego cambia según los dos primeros factores indicados en el punto v).

vii) Cualquier código de signo positivo pone el trigger S en 0. Los códigos negativos lo dejan sin cambio.

La tabla que figura a continuación resume todos los aspectos anteriores:

AREA PATRON	d	d	d	d	(	(	(	(	ø	ø	)	)	OTRO	OTRO
DATO FUENTE	0	0	≠ 0	≠ 0	0	0	≠ 0	≠ 0	NA	NA	NA	NA	NA	NA
TRIGGER S	0	1	0	1	0	1	0	1	0	1	0	1	0	1
ACCION														
ALMACENA DIGITO FUENTE EN PATRON		X	X	X		X	X	X						
DEJA CARACTER PATRON SIN CAMBIO										X				X
ALMACENA CARACTER DE RELLENO	X				X				X		X	X	X	
ESTADO DEL TRIGGER DESPUES DE LA OP	0	1	1	1	1	1	1	1	0	1	0	0	0	1

Ejemplo 62.

Se tienen tres datos en formato empaquetado, cada uno ocupa cuatro bytes. Se pide eliminar los ceros no significativos, editar cada dato con punto decimal separando dos dígitos de la parte fraccionaria, de la parte entera, además, si el dato es negativo editar a continuación de él la palabra NEG.

Usar el carácter blanco como carácter de relleno.

```
EDIC     START  0
         BALR   15,0
         USING  *,15
         ---
         ---
         ED     PATRØN(42), DATØS
         ---
         ---
         EØJ
DATØS    DC     PL4'-10541'
         DC     PL4'48539'
         DC     PL4'-3276814'
```

B, *Instrucciones después de operaciones aritméticas.* (Se analiza el resultado)

	Instrucción	*Significado*	*Código ampliado*	
BC	13,D2(X2,B2)	Salto si no es >0	BNP	D2(X2,B2)
BC	11,D2(X2,B2)	Salto si no es <0	BNM	D2(X2,B2)
BC	8,D2(X2,B2)	Salto si es =0	BZ	D2(X2,B2)
BC	7,D2(X2,B2)	Salto si no es =0	BNZ	D2(X2,B2)
BC	4,D2(X2,B2)	Salto si es <0	BM	D2(X2,B2)
BC	2,D2(X2,B2)	Salto si es >0	BP	D2(X2,B2)
BC	1,D2(X2,B2)	Salto si es overflow	BO	D2(X2,B2)
BCR	13,R2	Salto si no es >0	BNPR	R2
BCR	11,R2	Salto si no es <0	BNMR	R2
BCR	8,R2	Salto si es =0	BZR	R2
BCR	7,R2	Salto si no es =0	BNZR	R2
BCR	4,R2	Salto si es <0	BMR	R2
BCR	2,R2	Salto si es >0	BPR	R2
BCR	1,R2	Salto si es overflow	BOR	R2

C. *Instrucciones después de comparación.* (A comparado con B)

	Instrucción	*Significado*	*Código ampliado*	
BC	13,D2(X2,B2)	Salto si A no mayor que B	BNH	D2(X2,B2)
BC	11,D2(X2,B2)	Salto si A no menor que B	BNL	D2(X2,B2)
BC	8,D2(X2,B2)	Salto si A es igual a B	BE	D2(X2,B2)
BC	7,D2(X2,B2)	Salto si A no es igual a B	BNE	D2(X2,B2)
BC	4,D2(X2,B2)	Salto si A es menor que B	BL	D2(X2,B2)
BC	2,D2(X2,B2)	Salto si A es mayor que B	BH	D2(X2,B2)
BCR	13,R2	Salto si A no mayor que B	BNHR	R2
BCR	11,R2	Salto si A no menor que B	BNLR	R2
BCR	8,R2	Salto si A es igual a B	BER	R2
BCR	7,R2	Salto si A no es igual a B	BNER	R2
BCR	4,R2	Salto si A es menor que B	BLR	R2
BCR	2,R2	Salto si A es mayor que B	BHR	R2

D. *Instrucciones después de* TEST UNDER MASK

	Instrucción	*Significado*	*Código ampliado*	
BC	14,D2(X2,B2)	Salto si no hay unos	BNO	D2(X2,B2)
BC	8,D2(X2,B2)	Salto si hay ceros	BZ	D2(X2,B2)
BC	4,D2(X2,B2)	Salto si resultado mixto	BM	D2(X2,B2)
BC	1,D2(X2,B2)	Salto si hay unos	BO	D2(X2,B2)
BCR	14,R2	Salto si no hay unos	BNOR	R2
BCR	8,R2	Salto si hay ceros	BZR	R2
BCR	4,R2	Salto si resultado mixto	BMR	R2
BCR	1,R2	Salto si hay unos	BOR	R2

7. *Aritmética Decimal*

La aritmética decimal opera con datos que están en el formato "empaquetado" (PACKED) en el cual cada byte contiene dos dígitos decimales, excepto el byte del extremo derecho del campo que contiene un dígito y el signo. De acuerdo al formato, cada dato se interpreta como si fuera entero, de tal manera que, si el programador desea resultados reales, esto es, con parte entera y parte fraccionaria, es de su responsabilidad ubicar correctamente el punto decimal, imaginario, y operar de acuerdo a dicha ubicación.

7.1. *Instrucción* ZERO AND ADD

a) Instrucción : ZAP D1(L1,B1),D2(L2,B2)

b) Formato : SS

ZAP	L1	L2	B1	D1	B2	D2

c) Función : El segundo operando, que se encuentra en la dirección D2(B2) y tiene una longitud de L2 bytes, se ubica en la dirección D1(B1). La operación es equivalente a sumar una cantidad a cero.

Si el campo del primer operando es demasiado corto como para contener todos los dígitos *significativos* del segundo operando, se produce un desborde (overflow) decimal y como consecuencia de esto una interrupción de programa. Si el campo del primer operando es más largo que el necesario para contener el segundo operando, se rellena con ceros por la izquierda.

La operación se realiza de derecha a izquierda, byte por byte, de tal manera que puede haber superposición de campos siempre que los respectivos bytes del extremo derecho coincidan o que el del primer operando quede a la derecha del byte extremo del segundo operando.

Ejemplo 64.

```
AREA1   DS    PL4
DATØ    DC    PL5'-4897'
        ---
        ZAP   AREA1,DATØ
        ---
        EØJ
        END
```

contiene al multiplicando da la seguridad de que no se producirá desborde.

La longitud máxima del producto es de dieciséis bytes (quince en la instrucción de máquina) y corresponde a treinta y un dígitos y signo.

Puede haber superposición de campos, siempre que los respectivos bytes del extremo derecho coincidan, lo que permite multiplicar un dato consigo mismo.

Ejemplo 66.

Se tienen tres variables:

COSTO cuya configuración es DDD,DDS	5 dígitos y signo
DESCUENTO cuya configuración es DDS	2 dígitos y signo
CANTIDAD cuya configuración es DDDS	3 dígitos y signo

Calcular:

TOTAL = COSTO*DESCUENTO*CANTIDAD

El resultado se pide ajustado a un dígito de parte fraccionaria (la coma que aparece en las configuraciones de los datos no se almacena)

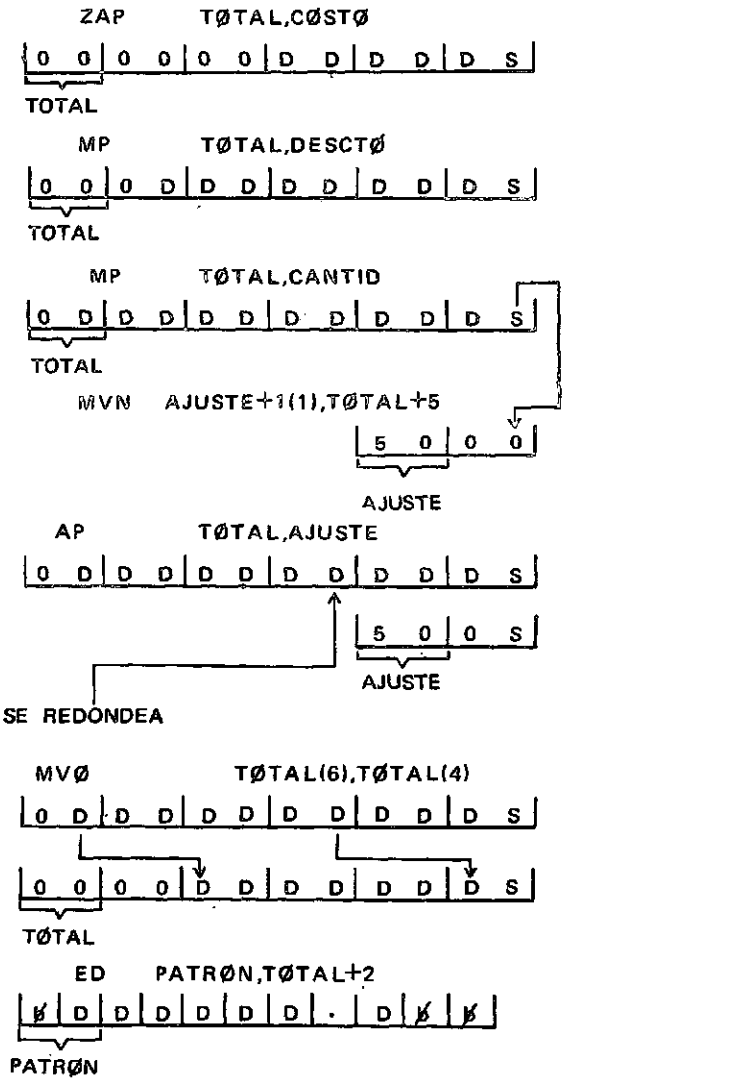
i) La cantidad máxima de dígitos que puede tener el resultado es 10, luego el campo TOTAL debe definirse con longitud de 6 bytes.

ii) La cantidad de dígitos que tendrá la parte fraccionaria será igual a 4, luego, si se quiere redondear a un dígito debe sumarse el valor 5 al dígito de orden inmediatamente inferior. En este caso se define una constante de valor 5000 en la que el dígito cero del extremo derecho es reemplazado por el signo del resultado total.

```
MULTI    START  0
         BALR   15,0
         USING  *,15
         --
         --
         ZAP    TØTAL,CØSTØ
         MP     TØTAL,DESCTØ
         MP     TØTAL,CANTID
         MVN    AJUSTE+1(1),TØTAL+5
         AP     TØTAL,AJUSTE
         MVØ    TØTAL(6),TØTAL(4)
         ED     PATRØN,TØTAL+2
         --
         --
         EØJ
AJUSTE   DC     X'5000'
```

```
TØTAL     DS      CL6
CØSTØ     DS      CL3
DESCTØ    DS      CL2
CANTID    DS      CL2
PATRØN    DC      X'40202020202021 4B204022'
          END
```

A continuación se describe el efecto producido por cada instrucción:



7.5. *Instrucción* DIVIDE DECIMAL

a) Instrucción : DP D1(L1,B1),D2(L2,B2)

b) Formato : SS

DP	L1	L2	B1	D1	B2	D2

c) Función : El dividendo (primer operando) es dividido por el divisor (segundo operando)y reemplazado por el cociente y el residuo. El campo del cociente va a la izquierda y el campo del residuo va a la derecha, en el área del primer operando. La longitud del residuo es igual a la del divisor y su signo es el mismo del dividendo.

La longitud máxima del divisor puede ser de 8 bytes (7 en la instrucción de máquina) y corresponde a 15 dígitos y signo, además, debe ser inferior a la del dividendo, en caso contrario se produce una interrupción de programa.

Los campos de dividendo y divisor pueden superponerse únicamente si sus bytes de orden inferior coinciden.

Para evitar que se produzca desborde, como consecuencia de un cociente mayor que el campo que lo puede contener, que significaría interrupción de programa, es conveniente generar el campo del dividendo con una longitud igual a la suma de los bytes que ocuparía el dividendo definido en forma independiente, más L2.

Ejemplo 67.

Se tienen las variables.

ALFA	que ocupa 5 bytes
BETA	que ocupa 3 bytes

Calcular:

$$\text{GAMMA} = \frac{\text{ALFA}}{\text{BETA}}$$

i) Suponiendo que ALFA y BETA son enteras y se desea GAMMA sin decimales.

ii) Suponiendo que ALFA y BETA son enteras y se desea GAMMA redondeado a dos decimales.

iii) Suponiendo que ALFA tiene dos decimales, BETA tiene un decimal y se desea GAMMA redondeado a dos decimales.

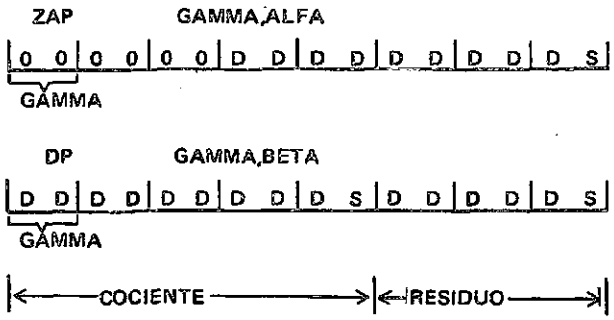
iv) Suponiendo que ALFA tiene 5 decimales, BETA un decimal y se desea GAMMA redondeado a un decimal.

v) Suponiendo que ALFA tiene un decimal, BETA dos decimales y se desea GAMMA redondeado a dos decimales.

Solución de i):

```
                START   0
                BALR    15,0
                USING   *,15
                ---
                ---
                ZAP     GAMMA,ALFA
                DP      GAMMA,BETA
                ED      PATRØN,GAMMA
                ---
                ---
                EØJ
GAMMA           DS      PL8
ALFA            DS      PL5
BETA            DS      PL3
PATRØN          DC      X'402020202020202020214B4022'
                END
```

Efecto de las instrucciones ZAP y DP:



Para solucionar los casos ii), iii), iv) y v) es conveniente hacer un análisis previo para determinar un procedimiento general. Deben considerarse los tres siguientes elementos:

DR = Dígitos de la parte fraccionaria que es necesario obtener en el resultado, para efectuar el redondeo adecuado.
DN = Dígitos de la parte fraccionaria del numerador (dividendo).
DD = Dígitos de la parte fraccionaria del denominador (divisor).

Se presentan tres casos posibles:

Primer caso: Numerador y denominador tienen igual número de dígitos en la parte fraccionaria (o no tiene ninguno de los dos) y se desea un resultado con parte fraccionaria.

La solución es amplificar el numerador por 10^{DR}.

El almacenamiento para el resultado debe contemplar: total de dígitos del numerador (TDN), DR y bytes que ocupa el denominador (L2).

$$L1 = \frac{TDN+DR+SIGNO}{2} + L2$$ si el resultado de la fracción no es entero exacto, se aproxima al entero superior inmediato.

Segundo caso: El numerador tiene mayor cantidad de dígitos en la parte fraccionaria que el denominador y se desea un resultado con parte fraccionaria.

Primero será necesario amplificar la fracción completa por 10^{DN} para eliminar las partes fraccionarias y luego amplificar el numerador por 10^{DR}. Pero se sabe que el punto decimal no está representado en el dato, luego, no es necesario amplificar el numerador por 10^{DN}; al mismo tiempo y por la misma causa, el denominador bastará con amplificarlo por 10^{DN-DD}.

Se tendrá así que el factor de amplificación está dado por:

$$\frac{10^{DR}}{10^{DN-DD}}$$

el cual se puede simplificar aún más, cuando se conozcan los datos DR, DN y DD.

Tercer caso: El numerador tiene menor cantidad de dígitos en la parte fraccionaria que el denominador, y se desea un resultado con parte fraccionaria.

Igual que en el segundo caso, primero será necesario amplificar la fracción completa, pero ahora por 10^{DD}, y luego amplificar el numerador por 10^{DR}. Sin embargo, el denominador no es necesario amplificarlo por 10^{DD} y el numerador se amplifica sólo por 10^{DD-DN} y luego por 10^{DR}.

Se tendrá así que el factor de amplificación está dado por:

$10^{DD-DN} * 10^{DR}$ o lo que es lo mismo $\frac{10^{DR}}{10^{DN-DD}}$

y se determina así que esta fórmula es aplicable en todos los casos, incluso en el primero en que la diferencia DN–DD=0, y se tiene por lo tanto:

$$\frac{10^{DR}}{10^{DN-DD}} = 10^{DR}$$

Solución de ii):

```
          START  0
          BALR   15,0
          USING  *,15
          ---
          ---
          ZAP    GAMMA,ALFA
          MP     GAMMA,MIL
          DP     GAMMA,BETA
          MVN    AJUSTE(1),GAMMA+6
          AP     GAMMA(7),AJUSTE
          MVØ    GAMMA(7),GAMMA(6)
          ED     PATRØN,GAMMA+1
          ---
          ---
          EØJ
AJUSTE    DC     X'50'
MIL       DC     P'1000'
PATRØN    DC     X'40202020202020202021420204022'
ALFA      DS     PL5
BETA      DS     PL3
GAMMA     DS     PL10
          END
```

Solución de iii):

$$DN = 2 \qquad DD = 1 \qquad DR = 3$$

$$\frac{10^3}{10^{2-1}} = 10^2$$

La solución es similar a la de ii), cambia solamente el factor de multiplicación que es CIEN en vez de MIL y las modificaciones que de esto se deriven para las instrucciones siguientes.

Solución de iv):

$$DN = 5 \qquad DD = 1 \qquad DR = 2$$

$$\frac{10^2}{10^{5-1}} = 10^{-2}$$

Observación : 5B es el código de operación correspondiente a SUBTRACT.

c) Indicar cuál es el contenido de los RUG 2 y 3 al finalizar el proceso siguiente:

```
         ICTL    1
         START   256
         BALR    8,0
         USING   *,8
         EJECT
         B       BEGIN
JØSE     DC      2F'+15'
BEGIN    LM      2,3,JØSE
         SRDL    2,64
         EØJ
         END
```

d) ¿Qué queda en los campo MAK2 y JUAN después de ejecu el programa siguiente?

```
         START   256
         BALR    7,0
         USING   *,7
         B       GØ
MAK1     DC      X'F6'
MAK2     DC      X'06'
JUAN     DC      2X'77'
GØ       NC      *-1(1),MAK2
         NC      JUAN,MAK1
         EØJ
         END
```

e) ¿Qué queda en el área PEPE después del proceso siguiente?

```
         START   400
         BALR    8,0
         USING   *,8
         B       BEGIN
PEPE     DC      PL4'567'
BEGIN    MVØ     PEPE(3),PEPE
         NI      PEPE+2,X'F0'
         NI      PEPE+3,X'0F'
         EØJ
         END
```

f) ¿Cuál es el contenido de las áreas M1, M2 y PEPE al término del proceso siguiente?

```
         START   0
         BALR    7,0
         USING   *,7
```

```
         B       GØ
M1       DC      X'F6'
M2       DC      X'16'
PEPE     DC      2X'30'
GØ       NC      PEPE+1(1),M2
         XC      M1(2),PEPE
         EØJ
         END
```

g) ¿Cuál es el contenido del área BEGIN después del proceso siguiente?

```
         START   0
         BALR    15,0
         USING   *,15
         B       SIGUE
BEGIN    DC      3F'2'
TABLA    DC      3X'0F'
SIGUE    TR      BEGIN(3),TABLA
         EØJ
         END
```

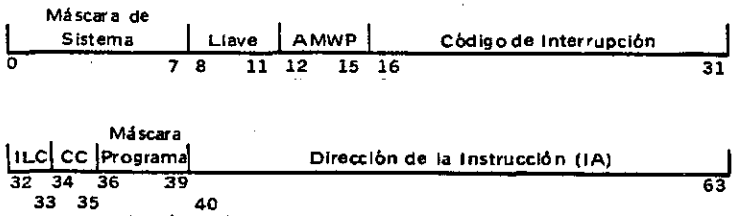
h) Indicar qué queda en DIGIT después del siguiente proceso:

```
         START   800
         BALR    14,0
         USING   *,14
         B       BEGIN
ZØNE     DC      ZL8'-18954312'
DIGIT    DS      PL6
BEGIN    PACK    DIGIT(6),ZØNE(8)
         MVC     DIGIT(1),DIGIT+5
         MVC     DIGIT+1(5),DIGIT
         EØJ
         END
```

i) ¿Qué queda en el campo A al finalizar el siguiente proceso?

```
         START   X'2800'
         BALR    2,0
         USING   *,2
         B       GØ
A        DC      PL6'4378000'
B        DC      P'26'
C        DC      X'50'
GØ       DP      A,B
         MVN     C(1),A+3
         AP      A(4),C
         MVØ     A(4),A(3)
         EØJ
         END
```

ejecuta. Esta dirección es incrementada en el número de bytes de la instrucción, obteniéndose así la dirección de la instrucción siguiente en secuencia, llamada dirección actualizada. La dirección actualizada reemplaza a la dirección anterior, lo que permite ejecutar la instrucción siguiente.



ILC = Código de longitud de la instrucción (*I*nstruction *L*ength *C*ode)
CC = Código de condición (*C*ondition *C*ode)
IA = Dirección de la instrucción (*I*nstruction *A*ddress)

En la bifurcación se opera fundamentalmente a base de los tres campos mencionados ILC, CC e IA. El primero da la longitud de la instrucción que se ejecuta, longitud que al ser sumada al valor del tercer campo permite obtener la dirección actualizada. El segundo campo indicará si la próxima instrucción se extrae de la dirección actualizada o de la dirección de bifurcación.

8.1. *Instrucción* BRANCH ON COUNT REGISTER

a) Instrucción : BCTR R1,R2

b) Formato : RR | BCTR | R1 | R2 |

c) Función : Al contenido del RUG especificado en R1 se le resta uno algebraicamente. Si el resultado obtenido es cero se ejecuta la siguiente instrucción en secuencia. Si el resultado es distinto de cero, la instrucción que se va a ejecutar se extrae de la dirección de bifurcación (se efectúa el salto a la dirección contenida en el RUG indicado en R2). Si el campo R2 es cero, se realiza la resta, pero, no se ejecuta salto.

8.2. *Instrucción* BRANCH ON COUNT

a) Instrucción : BCT R1,D2(X2,B2)

b) Formato : RX | BCT | R1 | X2 | B2 | D2 |

c) Función : Cumple igual función que BRANCH ON COUNT REGISTER. La dirección de bifurcación se calcula antes de ejecutar la resta.

Ejemplo 68.

Se tiene un registro de cien caracteres. Se pide contabilizar los caracteres blanco.

```
CUENTABL  START  0
          BALR   2,0
          USING  *,2
          SR     5,5
          L      6,DIRTABLA
          LA     6,99(6)
TEST      CLI    0(6),C'Ø'
          BE     IGUAL
          BCT    6,TEST
          B      STØRE
IGUAL     LA     5,1(5)
          BCT    6,TEST
STØRE     ST     5,CUENTA
          EØJ
CUENTA    DS     F
TABLA     DS     100CL1
BLANCØ    DC     C'Ø'
DIRTABLA  DC     A(TABLA)
          END
```

8.3. *Instrucción* BRANCH ON INDEX HIGH

a) Instrucción : BXH R1,R3,D2(B2)

b) Formato : RS

BXH	R1	R3	B2	D2

c) Función : Se suma un *incremento* a un *valor inicial* y la suma resultante se compara algebraicamente con un *valor final* o término de comparación. Si el resultado de la suma es mayor (HIGH) que el término de comparación, se efectúa el salto a la dirección dada por D2(B2), en caso contrario, se ejecuta la siguiente instrucción en secuencia.

El valor inicial debe estar cargado en el RUG especificado en R1, el incremento en el RUG dado en R3 que debe ser PAR y el valor de comparación en el RUG IMPAR inmediatamente posterior al indicado en R3.

Si R3 es un RUG IMPAR se considera el mismo incremento, como valor de comparación. Si R1 es igual al RUG que contiene el valor de comparación se considera como valor

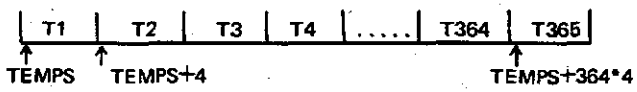
final el contenido original del registro, esto es, antes de que sea incrementado.

La dirección de bifurcación se calcula antes de las operaciones de suma y comparación.

Ejemplo 69.

Se tiene una tabla en la que están registradas las temperaturas máximas ocurridas en cada día de un año. Se pide obtener el promedio anual de temperaturas.

La información está en formato "empaquetado" ocupando cuatro bytes cada temperatura y con un dígito en la parte fraccionaria.




```
CØNTEMP   START  0
          BALR   6,0
          USING  *,6
          LA     7,TEMPS VALØR INICIAL
          LA     8,4           INCREMENTØ
          LA     9,TEMPS+364*4 VALØR FINAL
SUMA      AP     CØNTA,0(4,7)
          BXH    7,8,ØUT
          B      SUMA
ØUT       DP     CONTA,C365
          ED     PATRØ,CØNTA
          EØJ
CØNTA     DC     PL4'0'
PATRØ     DC     X'4020214B204022'
C365      DC     P'365'
TEMPS     DS     365PL4
          END
```

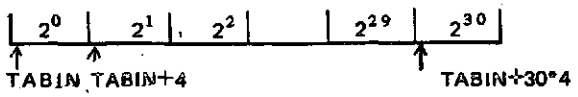
Se carga en el RUG7 la dirección de comienzo de la tabla que será el valor inicial. En el RUG8 se carga el valor 4 como incremento y en el RUG9 se carga la dirección del último elemento de la tabla que será el valor final o término de comparación.

Se suman en CONTA las temperaturas, utilizando el RUG7 que tiene la dirección de cada elemento.

En la instrucción BRANCH ON INDEX HIGH se suma al <RUG7> el <RUG8> y el resultado se compara con el <RUG9>. Si el resultado de la comparación es mayor se efectúa un salto a OUT, en caso contrario, continúa la secuencia normal, pero esta secuencia normal significa, en este caso, volver a la dirección SUMA.

Ejemplo 70.

Generar una tabla de potencias de 2, en binario, desde 2^0 hasta 2^{30}.




```
                START   0
                BALR    15,0
                USING   *,15
                L       0,UNØ
                SR      1,1
                LA      2,4
                LA      3,31*4
                LA      4,TABIN-4
LØØP            BXH     1,2,FIN
                ST      0,0(1,4)
                SLA     0,1
                B       LØØP
FIN             EØJ
UNØ             DC      F'1'
TABIN           DS      31F
                END
```

8.4. *Instrucción* BRANCH ON INDEX LOW OR EQUAL

a) Instrucción : BXLE R1,R3,D2(B2)

b) Formato : RS

BXLE	R1	R3	B2	D2

c) Función : La función es similar a la de la instrucción BRANCH ON INDEX HIGH. En esta instrucción la bifurcación se ejecuta cuando la suma del contenido del RUG especificado en R1 y el incremento, resulta menor o igual que el valor final.

Ejemplo 71.

Se tienen cinco tablas diferentes, cada tabla tiene cien elementos y cada elemento tiene nueve dígitos y signo, en formato empaquetado. Se pide calcular la suma de cada quintupla de elementos.

TAB1	TAB2	TAB3	TAB4	TAB5	TSUM
a_{11}	a_{21}	a_{31}	a_{41}	a_{51}	$\sum a_{i1}$
a_{12}	a_{22}	a_{32}	a_{42}	a_{52}	$\sum a_{i2}$
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
a_{1100}	a_{2100}	a_{3100}	a_{4100}	a_{5100}	$\sum a_{i100}$

```
         START  0
         BALR   15,0
         USING  *,15
         LM     6,11,REG
LØØP     ZAP    0(6,6),ZERØ
         AP     0(6,6),0(5,9)
         AP     0(6,6),500(5,9)
         AP     0(6,6),1000(5,9)
         AP     0(6,6),1500(5,9)
         AP     0(6,6),2000(5,9)
         BXLE   6,7,*+4
         BXLE   9,10,LØØP
         EØJ
REG      DC     A(TSUM)
         DC     F'6'
         DC     F'0'
         DC     A(TAB1)
         DC     F'5'
         DC     A(TAB1+99*5)
ZERØ     DC     P'0'
TAB1     DS     500CL5
TSUM     DS     100CL6
         END
```

La instrucción BXLE 6,7,*+4 se utiliza con el objeto de incrementar el contenido del RUG6 que ha sido cargado anteriormente con la dirección TSUM. Cualquiera que sea el resultado de la comparación siempre se ejecutará la instrucción siguiente, dado que la dirección de bifurcación corresponde a la instrucción que sigue en secuencia.

La instrucción BXLE 9,10,LOOP incrementa el contenido del RUG9 con el contenido del RUG10 y el resultado lo compara con el contenido del RUG11. Mientras dicho resultado sea menor o igual, la ejecución se reiniciará en la dirección LOOP.

8.5. *Instrucción:* EXECUTE

a) Instrucción : EX R1,D2(X2,B2)

b) Formato : RX | EX | R1 | X2 | B2 | D2 |

c) Función : Se modifica la instrucción ubicada en la dirección D2(X2,B2) por el contenido del RUG especificado en R1, y la instrucción resultante es ejecutada.

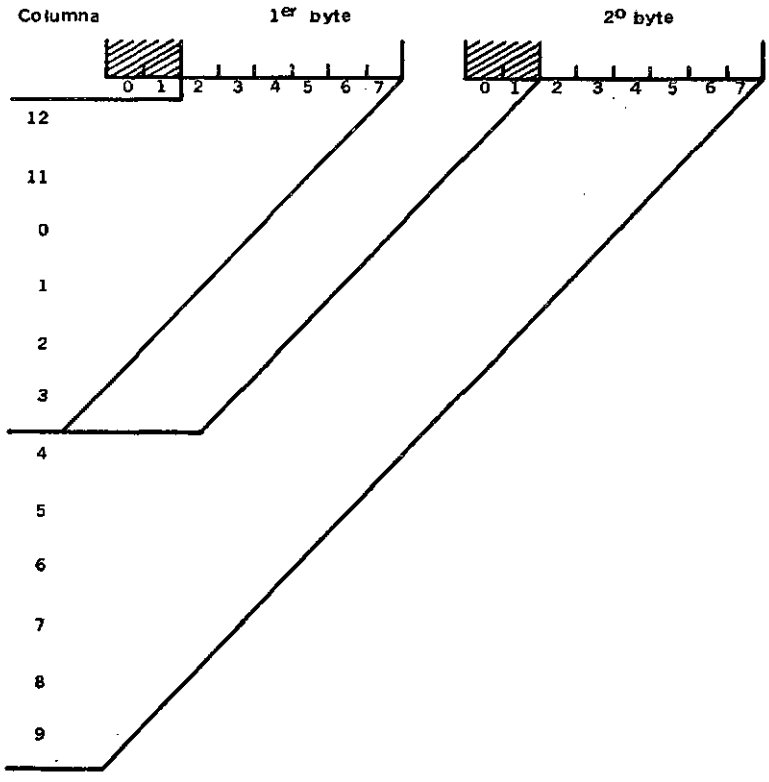
La modificación se realiza en los bits 8 al 15 de la instrucción, los que son puestos en conexión lógica OR con los bits 24 al 31 del registro. Si se especifica el registro 0 en R1 no hay modificación. La conexión lógica OR no

altera ni el contenido del registro, ni la instrucción original.

Una vez que la instrucción resultante ha sido ejecutada se retorna a la instrucción siguiente a EXECUTE, salvo que la instrucción sujeta a modificación sea de bifurcación, en cuyo caso la dirección de salto reemplaza en la PSW a la dirección actualizada. Si la instrucción que se modifica es otra instrucción EXECUTE se produce una excepción de ejecución y como consecuencia una interrupción de programa.

Ejemplo 72.

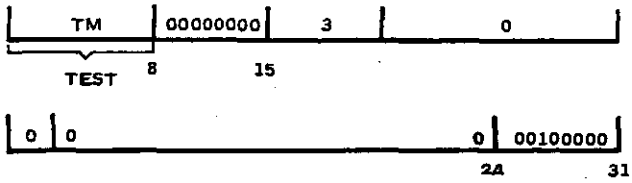
Existe la posibilidad de leer tarjetas multiperforadas, esto es, que pueden tener hasta las doce posiciones perforadas, dentro de cada columna. La información se almacena en dos bytes seguidos, por cada columna leída, como se indica a continuación:



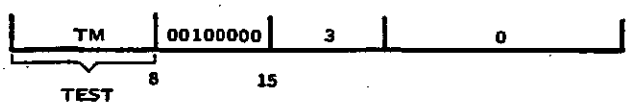
Los dos primeros bits de cada byte no se utilizan. En el problema que figura a continuación se ha simulado una tarjeta leída y almacenada en TARJBIN. Se pide analizar los bits de información de tal manera que si es 1 se guarda un carácter uno en MATRIZ, en caso contrario se guarda un carácter cero.

```
PRØG2     START  0
          BALR   2,0
          USING  *,2
          LM     3,6,REG
BEGIN     LM     7,9,ØTRØCER
          LA     10,32
CØMPA     EX     10,TEST
          BØ     UNØS
          MVC    0(1,6),CERØCAR
          B      MØDIF
UNØS      MVC    0(1,6),UNØCAR
MØDIF     A      6,UNØBIN
          SRL    10,1
          BXLE   7,8,CØMPA
          BXLE   3,4,BEGIN
          EØJ
REG       DC     A(TARJBIN)
UNØBIN    DC     F'1'
          DC     A(TARJBIN+159)
          DC     A(MATRIZ)
ØTRØCER   DC     F'0,1,5'
UNØCAR    DC     C'1'
CERØCAR   DC     C'0'
TARJBIN   DC     20X'0123456789ABCDEF'
TEST      TM     0(3),X'00'
MATRIZ    DS     12CL80
          END
```

La instrucción EX 10, TEST pone en conexión lógica OR los bits 8 a 15 de la instrucción que hay en TEST con los bits 24 a 31 del RUG 10.



La instrucción resultante será:



la que permitirá analizar el bit 2 del byte cuya dirección está dada por el contenido del RUG3, esto es, TARJBIN. Después de efectuado el análisis y hechas las modificaciones necesarias para almacenar el siguiente carácter cero o uno, se desplaza el contenido del RUG10 un lugar a la derecha, de tal manera que, al empezar de nuevo el ciclo se pueda investigar el bit 3 del byte que está en la dirección que contiene el RUG3, y así sucesivamente.

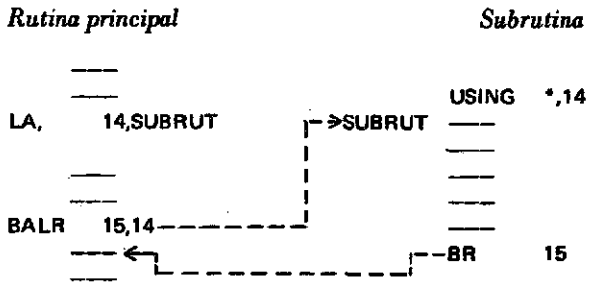
9. *Subrutinas y Subprogramas*

Se define como subrutina al segmento de programa al cual se hace referencia en distintos puntos de éste y se compagina (ensambla) junto con él.

Se define como subprograma al programa independiente que es llamado por un programa principal y que se compagina aparte de él.

En el primer caso surgen las preguntas siguientes: ¿Cómo sabe la subrutina donde debe volver? y ¿de dónde toma los datos y dónde deja los resultados? El problema que plantea la primera pregunta se resuelve con la instrucción BRANCH AND LINK REGISTER, que se ha utilizado anteriormente con el segundo operando igual a cero, antes de la pseudo-instrucción USING. La instrucción carga en el RUG especificado en R1 la dirección de la próxima instrucción y salta a la dirección contenida en el RUG indicado en R2. Esta última dirección será la de la subrutina, la cual tendrá como instrucción de retorno un salto incondicional a la dirección contenida en el RUG especificado en R2, que debe ser el mismo que figura en R1 en la instrucción BALR.

Ejemplo 73.



Los problemas que plantea la segunda pregunta los resuelve el propio programador estableciendo el o los registros donde entregará los resultados y en los que recibirá resultados. Es posible que en vez de registros utilice un área de memoria con acceso por parte de la rutina principal y de la subrutina.

Ejemplo 74.

```
BEGIN    START  0
         BALR   2,0
         USING  *,2
         L      3,UNØ        <RUG 3>  = 1
         L      14,DIRSR     <RUG 14> = SR1
         BALR   13,14        <RUG 13> = ST1
ST1      ST     3,RESP1      <RESP1>  = <RUG 3>
         L      3,CUATRØ     <RUG 3>  = 4
         L      14,DIRSR     <RUG 14> = SR1
         BALR   13,14        <RUG 13> = ST2
ST2      ST     3,RESP2      <RESP2>  = <RUG 3>
         EØJ
DIRSR    DC     A(SR1)
UNØ      DC     F'1'
CUATRØ   DC     F'4'
RESP1    DS     F
RESP2    DS     F
*SUBRUTINA
         USING  *,14
SR1      SLA    3,1
         BR     13
         END    BEGIN
```

La subrutina desplaza el contenido del RUG3 un lugar a la izquierda y retorna a la rutina principal.

Ejemplo 75.

```
PRØGA    START  0
         BALR   2,0
         USING  *,2
         L      14,DIREC
         CNØP   2,4
         BALR   13,14
CØNST    DC     A(LISTA)
         DC     F'4'
         DC     A(PRØM1)
         L      14,A
         A      14,A+4
         ST     14,C
         L      14,DIREC
         CNØP   2,4
         BALR   13,14
         DC     A(LIST8)
         DC     F'6'
         DC     A(PRØM2)
         EØJ
A        DC     F'95,81'
```

```
C          DS      F
LISTA      DC      F'31,42,27,18'
LISTB      DC      F'11,24,-5,-91,57,77'
PRØM1      DS      F
PRØM2      DS      F
DIREC      DC      A(PRØM)
*SUBRUTINA PRØMEDIØ
           USING   *,14
PRØM       STM     2,7,SALVA
           L       5,0(13)
           LA      6,4
           L       4,4(13)
           LR      7,4
           S       7,UNØ
           SLA     7,2
           AR      7,5
           SR      2,2
           SR      3,3
SUMA       A       3,0(5)
           BXLE    5,6,SUMA
           DR      2,4
           L       5,8(13)
           ST      3,0(5)
           LM      2,7,SALVA
           B       12(13)
UNØ        DC      F'1'
SALVA      DS      6F
           END     BEGIN
```

En el caso de subprogramas, el problema que es necesario resolver es el "enganche" entre la rutina principal y el subprograma, dado que ambos constituyen módulos independientes que pueden ser compaginados en momentos distintos y que en el momento de ser cargados en memoria para su ejecución no tienen por qué serlo en las mismas ubicaciones que ocuparon inicialmente. En otras palabras, cuando se compaginó la rutina principal, la referencia al subprograma, que corresponde a la dirección donde debe estar cargado el mismo, quedó sin resolver pues no era conocida dicha dirección. Sólo será resuelto ese problema cuando se carguen ambos módulos para ser ejecutados. El sistema debe saber cuál o cuáles símbolos son externos, o lo que es lo mismo, debe conocer cuáles símbolos están definidos en otro módulo y eso se consigue con la pseudo-instrucción EXTERNAL que tiene como operandos los símbolos externos. Además, el sistema debe saber dónde buscar la definición de dichos símbolos y eso se logra con la pseudo-instrucción ENTRY que tiene como operandos los símbolos utilizados en otros módulos y que están definidos en el módulo en que ella se encuentra.

9.1. *Pseudo-Instrucción* EXTERNAL

a) Código : EXTRN

b) Formato : blanco EXTRN operando

c) Función : Declara aquellos símbolos que son usados en el módulo, pero que están definidos fuera de él. Si hay más de un símbolo, se separa del resto por coma. Los símbolos declarados no se pueden utilizar como identificadores de proposición dentro del mismo módulo.

9.2. *Pseudo-Instrucción* ENTRY

a) Código : ENTRY

b) Formato : blanco ENTRY operando

c) Función : Declara aquellos símbolos que son definidos en el módulo y que son utilizados por otros módulos. Si hay más de un símbolo se separa del resto por coma. Los símbolos utilizados aparecen como identificadores de proposición dentro del mismo módulo. Si el símbolo es el nombre de la sección de control no necesita ser declarado con la pseudo-instrucción ENTRY.

Otra forma de declarar los símbolos externos es definiéndolos en la rutina llamadora, con constantes de dirección tipo V y cargando las constantes en registros de uso general antes de hacer un BRANCH AND LINK REGISTER que produciría el salto al subprograma respectivo.

Ejemplo 76.

El mismo problema del ejemplo 75, resuelto ahora, con un subprograma.

```
PRØGA    START  0
         EXTRN  PRØM
         BALR   2,0
         USING  *,2
         L      14,DIREC
         CNØP   2,4
         BALR   13,14
CØNST    DC     A(LISTA)
         DC     F'4'
         DC     A(PRØM1)
         L      14,A
```

```
                A       14,A+4
                ST      14,C
                L       14,DIREC
                CNØP    2,4
                BALR    13,14
                DC      A(LISTB)
                DC      F'6'
                DC      A(PRØM2)
                EØJ
A               DC      F'95,81'
C               DS      F
LISTA           DC      F'31,42,27,18'
LISTB           DC      F'11,24,-5,-91,57,77'
PRØM1           DS      F
PRØM2           DS      F
DIREC           DC      A(PRØM)
                END     PRØGA

PRØM            START   0
                ENTRY   PRØM
                USING   *,14
                STM     2,7,SALVA
                L       5,0(13)
                LA      6,4
                L       4,4(13)
                LR      7,4
                S       7,UNØ
                SLA     7,2
                AR      7,5
                SR      2,2
                SR      3,3
SUMA            A       3,0(5)
                BXLE    5,6,SUMA
                DR      2,4
                L       5,8(13)
                ST      3,0(5)
                LM      2,7,SALVA
                B       12(13)
UNØ             DC      F'1'
SALVA           DS      6F
                END
```

Observaciones:

a) No es necesario colocar ENTRY PROM dado que PROM es punto de entrada por derecho propio, por ser el nombre de la sección de control.

b) Si se especifica DIREC DC V(PROM) no es necesario colocar EXTERN PROM.

10. *Instrucciones nuevas de Assembler para el Sistema/370*

10.1. *Instrucción* COMPARE AND SWAP

a) Instrucción : CS R1,R3,D2(B2)

b) Formato : RS

c) Función : Se comparan el primer y el segundo operandos. Si son iguales, el tercer operando es almacenado en la ubicación del segundo. Si son distintos, el segundo operando es cargado en la ubicación del primero.

Todos los operandos tienen una palabra de longitud. El primero y el tercero están en los RUG especificados en R1 y R3 respectivamente, y el segundo está en la dirección D2(B2) que debe cumplir con alineamiento de palabra.

Si el resultado de la comparación es distinto no se efectúa almacenamiento en memoria y por lo tanto no se toman acciones de protección de memoria y cambio de bit.

Cuando el resultado de la comparación es igual, no se permite el acceso de otra Unidad Central de Proceso (UCP) a la ubicación del segundo operando. El acceso es impedido prácticamente desde el momento en que el segundo operando es cargado para comparación hasta el momento en que el tercer operando es almacenado en la ubicación del segundo.

Una función de tipo serial es realizada antes de que el operando esté cargado y lo mismo ocurre si aparece el código de condición 0 después que el resultado es almacenado. La operación de la UCP es demorada hasta que todos los accesos previos de ella a memoria principal han sido terminados, lo mismo es observado para canales y otras UCP y después que eso ocurre, el segundo operando es cargado.

A ninguna instrucción posterior o a sus operandos tiene acceso la UCP, hasta que la ejecución de la instrucción COMPARE AND SWAP está terminada, incluida la colocación del valor del resultado, si lo hay, en la memoria principal, lo mismo es observado para canales y otras UCP.

Código de Condición	*Resultado*
0	Primer y segundo operandos son iguales
1	Primer y segundo operandos son distintos

Observación: La instrucción COMPARE AND SWAP puede ser usada por programas que comparten áreas de almacenamiento común, ya sea en multiprogramación o en multiproceso. Por ejemplo, un programa puede modificar el contenido de una ubicación aun cuando exista la posibilidad de que otra UCP pueda actualizar simultáneamente la ubica-

ción. En este caso, primero se carga la palabra que se va a actualizar, en un RUG. En seguida, el valor actualizado es computado y colocado en otro RUG. Después es ejecutada la instrucción COMPARE AND SWAP con el valor original en el RUG especificado en R1 y con el valor actualizado en el RUG indicado en R3. Si se produce el código de condición 0, la actualización se ha producido (los valores son iguales), en caso contrario, la ubicación de memoria ya no contiene el valor original, o sea, no se ha producido la actualización deseada, el RUG indicado en R1 tiene un nuevo valor obtenido por la intervención de otro programa u otra UCP. Luego se puede repetir el procedimiento con los mismos valores.

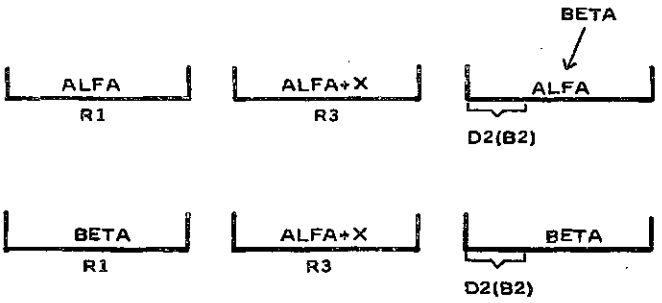
Gráficamente el problema se puede representar en la siguiente forma:

1º Se cargan los valores original y actualizado en los RUG . Sea ALFA el valor original y ALFA+X el valor actualizado.

2º Se ejecuta la instrucción COMPARE AND SWAP.

Si el primer y el segundo operandos son iguales queda en D2(B2) el valor ALFA+X.

Si el contenido de D2(B2) fue modificado por otro programa u otra UCP, el resultado de la comparación será distinto y en el RUG indicado en R1 y en D2(B2) quedará el nuevo valor, por ejemplo BETA.



3º Si el código de condición producido es 1, lo que significa que primer y segundo operandos eran distintos, se repite la instrucción COMPARE AND SWAP con los mismos valores que se tienen en D2(B2) y en el RUG indicado en R1.

Suponiendo que no hay una nueva modificación del contenido de la dirección D2(B2) se tendrá finalmente:

10.2. *Instrucción* COMPARE DOUBLE AND SWAP

a) Instrucción : CDS R1,R3,D2(B2)

b) Formato : RS

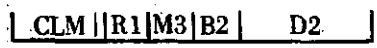
c) Función : Cumple la misma función que la instrucción COMPARE AND SWAP.

Todos los operandos tienen una doble palabra de longitud. En los campos R1 y R3 se especifican RUG pares, dado que el primer y el tercer operandos ocupan dos RUG, PAR e IMPAR siguiente, cada uno. El segundo operando está en la dirección D2(B2) que debe cumplir con alineamiento de doble palabra.

10.3. *Instrucción* COMPARE LOGICAL CHARACTERS UNDER MASK

a) Instrucción : CLM R1,M3,D2(B2)

b) Formato : RS



c) Función : Se compara el segundo operando con el primero en función de una máscara. Se genera código de condición de acuerdo al resultado.

Se utiliza el campo M3 como máscara, haciendo corresponder cada bit del campo con cada byte del RUG especificado en R1, partiendo de izquierda a derecha en ambos casos. Los bytes del RUG que corresponden a bits uno de la máscara se consideran contiguos y se comparan con igual número de bytes a partir de la dirección D2(B2). Los bytes que corresponden a bits cero no participan en la operación.

La comparación es realizada considerando los operandos como cantidades binarias sin signo. Ningún operando es cambiado.

Código de Condición	*Resultado*
0	Los bytes seleccionados son iguales o la máscara es cero.
1	El campo del primer operando es menor que el segundo operando.
2	El campo del primer operando es mayor que el segundo operando.

10.4. *Instrucción* COMPARE LOGICAL LONG

a) Instrucción : CLCL R1,R2

b) Formato : RR | CLCL | R1 | R2 |

c) Función : Se compara el primer operando con el segundo. Se genera código de condición de acuerdo al resultado.

En los campos R1 y R2 se especifican RUG pares, dado que ambos operandos ocupan dos RUG , PAR e IMPAR siguiente, cada uno. El primer byte o byte de orden superior de cada operando, es designado por el contenido de los bits 8–31 del RUG IMPAR respectivo. Los bits 0–7 del RUG IMPAR que corresponde al segundo operando contienen un carácter de relleno que se ocupa para extender el operando más corto hasta completar la longitud del operando más largo. Los bits 0–7 de los RUG pares y del impar que corresponde al primer operando, se ignoran.

El contenido de cada RUG se describe a continuación en forma gráfica:

RUG PAR indicado en R1

RUG IMPAR siguiente

RUG PAR indicado en R2

RUG IMPAR siguiente

La comparación se realiza de izquierda a derecha, byte por byte. La operación termina cuando se detecta una desigualdad o cuando se detecta el final de los campos. Ningún operando es cambiado. Si se especifica longitud cero para ambos operandos, se consideran iguales.

En el caso de encontrar bytes desiguales durante la comparación, el campo de longitud (contador) y el de dirección al término de la operación identifican al byte de la desigualdad, para ello, el contenido de los bits 8 a 31 de los RUG impares es disminuido en el número de bytes en que hubo igualdad, a menos que la desigualdad haya ocurrido con el carácter de relleno, en cuyo caso, el campo de longitud para el operando más corto es puesto en cero. El contenido de los bits 8 a 31 de los RUG pares es incrementado en el número de bytes en que hubo igualdad. Si los dos operandos son iguales, incluido el carácter de relleno si es necesario, los dos campos de longitud son puestos en cero y las direcciones son incrementadas en los valores de longitud correspondientes. El contenido de los bits 0 a 7 de los RUG pares es puesto en cero y el de los RUG impares permanece sin cambio.

El control que se tiene sobre la cantidad de bytes comparados permite que la instrucción sea interrumpida por un evento externo y reiniciada a partir del punto de interrupción. En este caso la dirección de la instrucción en la PSW aparece como si la instrucción no hubiera sido aún ejecutada.

Código de Condición	*Resultado*
0	Los operandos son iguales o ambos campos tienen longitud cero.
1	El primer operando es menor.
2	El primer operando es mayor.

10.5. *Instrucción* INSERT CHARACTERS UNDER MASK

a) Instrucción : ICM R1,M3,D2(B2)

b) Formato : RS

ICM	R1	M3	B2	D2

c) Función : Se almacenan en el RUG especificado en R1, bytes tomados a partir de la dirección D2(B2).

La cantidad de bytes que se almacena corresponde a los unos que contiene la máscara M3.

Los bytes que se llenan del RUG son los que corresponden a los unos de la máscara, comenzando de izquierda a derecha. Los bytes que corresponden a ceros de la máscara permanecen sin cambio.

El código de condición resultante depen-

de de la máscara y de los bits almacenados. Si la máscara es cero o si todos los bits almacenados son iguales a cero, el código es cero. Si no todos los bits son iguales a cero se considera el bit almacenado de orden superior (bit del extremo izquierdo del campo D2(B2)). Si el bit es uno, el código es uno, si el bit es cero, el código es dos.

Código de Condición	*Resultado*
0	Máscara cero o bits insertados son todos ceros.
1	Bit insertado de orden superior es uno.
2	Bit insertado de orden superior es cero.

10.6. *Instrucción* MOVE LONG

a) Instrucción : MVCL R1,R2

b) Formato : RR

MVCL	R1	R2

c) Función : El segundo operando es movido a la ubicación del primer operando, siempre que no haya traslapo de direcciones de operando que afecten al contenido final del resultado. Si quedan bytes de orden inferior de la ubicación del primer operando que no han sido llenados, se transfieren a ellas caracteres de relleno.

En los campos R1 y R2 se especifican RUG pares, dado que ambos operandos ocupan dos RUG, PAR e IMPAR siguiente, cada uno. El primer byte o byte de orden superior de cada operando es designado por el contenido de los bits 8-31 del RUG PAR respectivo. La longitud de cada operando es especificada por el contenido de los bits 8-31 del RUG IMPAR respectivo. Los bits 0-7 del RUG IMPAR que corresponde al segundo operando contienen un caracter de relleno. Los bits 0-7 de los RUG pares y del impar que corresponde al primer operando, se ignoran.

El movimiento parte en el extremo de orden superior de ambos campos y sigue hacia la derecha. No hay cambio ni inspección de los operandos. La operación es finalizada cuando el número de bytes especificados en el RUG impar del primer operando ha sido movido a la dirección del primer operando. A medida que se realiza la transferencia, el contenido de los RUG impares va siendo disminuido. Si el RUG impar que corresponde al segundo operando llega a cero primero, se continúan transfiriendo caracteres de relleno.

Como parte de la ejecución de la instrucción, los contenidos de los RUG impares (contadores) son comparados para establecer el código de condición, además se hace un chequeo de las direcciones de los operandos para determinar si hay traslapo destructivo. Se entiende por traslapo destructivo de los operandos cuando la ubicación del primero de ellos es utilizada como fuente (segundo operando), después de que un dato ha sido movido a ella. Cuando hay traslapo destructivo, no se produce movimiento y el código de condición es puesto en 3.

Dependiendo de si el segundo operando abarca desde la posición 16.777.215 a la posición 0, el movimiento tiene lugar en los siguientes casos:

a) Cuando el segundo operando no abarca esas posiciones, el movimiento es realizado cuando el byte de orden superior del primer operando coincide con o está a la izquierda del byte de orden superior del segundo operando, *o* si el byte de orden superior del primer operando está a la derecha del byte de orden inferior del segundo operando que está participando en la operación.

b) Cuando el segundo operando abarca esas posiciones, el movimiento es realizado de acuerdo a las mismas condiciones expresadas en a) cambiando solamente la *o* por *y*.

El byte de orden inferior del segundo operando se determina por la menor de las longitudes (contadores) de los operandos.

Cuando la longitud especificada en los bits 8 a 31 del RUG IMPAR que corresponde al primer operando es cero, no se realiza movimiento, pero se genera código de condición para indicar los valores relativos de las longitudes.

El control que se tiene sobre la cantidad de bytes transferidos permite que la instrucción sea interrumpida por un evento externo y reiniciada a partir del punto de interrupción. En este caso, la dirección de la instrucción en la PSW aparece como si la instrucción no hubiera sido aún ejecutada.

Código de Condición	*Resultado*
0	Las longitudes de los operandos son iguales
1	La longitud del primer operando es menor
2	La longitud del primer operando es mayor
3	No hay transferencia a causa de traslapo destructivo.

10.7. *Instrucción* SHIFT AND ROUND DECIMAL

a) Instrucción : SRP D1(L1,B1),D2(B2),I3

b) Formato : SS

SRP	L1	I3	B1	D1	B2	D2

c) Función : El primer operando ubicado en la dirección D1(B1) es desplazado de acuerdo a los seis bits de orden inferior de la representación binaria de la dirección D2(B2). Cuando se especifica un desplazamiento a la derecha, el resultado es redondeado con el factor I3.

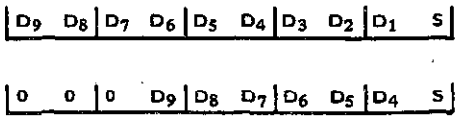
Los valores que figuran a continuación indican la interpretación de los seis bits que especifican el desplazamiento.

Contenido de los seis bits	*Interpretación*
011111	31 dígitos se desplazan a la izquierda
000001	1 dígito se desplaza a la izquierda
000000	No hay desplazamiento.
111111	1 dígito se desplaza a la derecha
100000	32 dígitos se desplazan a la derecha.

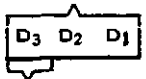
Se considera que el primer operando está en formato decimal empaquetado y se verifica la validez de los dígitos decimales y del código de signo, este último no participa en el desplazamiento. Se introducen ceros en las posiciones de dígitos que quedan desocupadas. Un resultado cero es considerado positivo.

Si un dígito significativo es desplazado fuera de la posición del dígito de orden superior durante un desplazamiento a la izquierda, se produce desborde (overflow) decimal. La operación se termina ignorando el desborde.

Durante el desplazamiento a la derecha, el contenido del campo I3 se utiliza como un factor de redondeo. Este factor se suma al último dígito que salió fuera del campo por efecto del desplazamiento y propagando el dígito de desborde si lo hay, hacia la izquierda. Tanto el primer operando como el factor de redondeo son considerados positivos sólo para efectuar la suma.



Dígitos Desplazados



último dígito desplazado

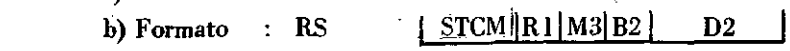
La validez del primer operando es verificada y se establece codigo de condición aun cuando se especifique desplazamiento cero.

Código de Condición	*Resultado*
0	Cero
1	Menor que cero
2	Mayor que cero
3	Desborde (overflow)

10.8. *Instrucción* STORE CHARACTERS UNDER MASK

a) Instrucción : STCM R1,M3,D2(B2)

b) Formato : RS



c) Función : Se seleccionan bytes del primer operando de acuerdo a una máscara y se almacenan en la dirección dada por el segundo operando.

Se utiliza el campo M3 como máscara, haciendo corresponder cada bit del campo, con cada byte del RUG especificado en R1, partiendo de izquierda a derecha en ambos casos. Los bytes del RUG que corresponden a bits uno de la máscara se almacenan uno a continuación del otro conservando el orden original, a partir de la dirección D2(B2).

El número de bytes almacenados es igual al número de unos en la máscara. El contenido del RUG no se altera. No se genera código de condición.

11. *Entrada/salida de información (Input/Output)*

Las operaciones de entrada/salida se ejecutan a través de dispositivos llamados canales, los cuales simplemente conectan unidades de entrada/salida a la unidad de procesamiento.

Puede considerarse el canal como un pequeño computador independiente para manejar operaciones de entrada/salida. Tiene un conjunto limitado de instrucciones llamados comandos. Un conjunto de comandos forma un "programa de canal". Tiene además sus propios registros internos para operaciones y por lo tanto no requiere del uso de los registros de la UCP, aún cuando comparte memoria con ella.

Pocas veces se realizan programas de canales, fundamentalmente porque existen macro-instrucciones que liberan al programador de aquella tarea. Sin embargo, con el objeto de dar una visión más completa se

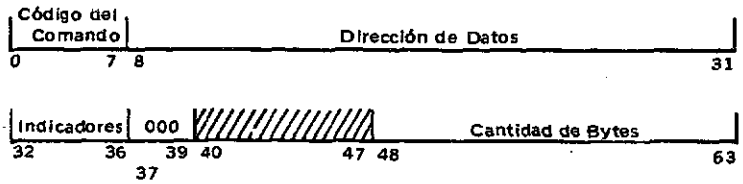
analizarán algunos aspectos y elementos de programación relacionados con los programas de canales aun cuando se utilicen siempre algunas macro-instrucciones que permitirán darle más claridad a los ejemplos.

Una entrada típica de datos es la lectura de una tarjeta como asimismo una salida típica es la impresión de una línea. Al aparecer una instrucción de entrada o salida en el programa del usuario y ser analizada esa instrucción por la UCP, ésta notifica al canal que corresponde que debe iniciar su programa. La comunicación la realiza a través de una instrucción privilegiada START INPUT OUTPUT (SIO) cuya ejecución genera una cadena de hechos, el primero de los cuales es transferir la *palabra de dirección del canal* (Channel Address Word–CAW) desde los bytes 72 a 75 de la memoria principal al canal designado por SIO. El formato de la palabra CAW es el siguiente:

Existe la posibilidad de "proteger" la memoria principal de posibles destrucciones de información inadvertidas. Con este objeto, se divide en bloques de 2048 bytes cada uno y a cada bloque se le asigna un registro de cuatro bits. Las combinaciones de cuatro bits pueden considerarse "llaves" de almacenamiento.

El almacenamiento se efectúa solamente si las combinaciones de la llave de protección proporcionada por la CAW (o la PSW) y la llave del almacenamiento coinciden o cuando la llave de la CAW (PSW) tiene un valor cero.

La dirección del comando es la dirección efectiva del primer comando en el programa de canal. El formato de la *palabra de comando de canal* (Channel Command Word – CCW) es el siguiente:



Los comandos son seis:

READ
READ BACKWARD
SENSE
WRITE
CONTROL y
TRANSFER IN CHANNEL

Read, causa la transferencia de información desde un dispositivo de entrada, a memoria. *Read Backward*, permite leer información desde una cinta magnética que se mueve en dirección contraria a la que se utilizó para grabarla. *Sense*, transfiere información de estados de un dispositivo a la memoria principal. *Write*, causa la transferencia de información desde memoria a un dispositivo de salida. *Control*, se utiliza para acciones como: rebobinar cinta magnética, saltarse archivos o registros físicos, etc. *Transfer in Channel*, es una instrucción de bifurcación.

Los comandos se definen en forma similar a las instrucciones de Assembler:

[nombre] CCW operando1,operando2,operando3,operando4

donde:

nombre : es optativo y permite identificar el comando

operando1: código del comando, especifica qué función se realizará. Ocupa los bits 0-7.

operando2: expresión reubicable que identifica el área I/0. Ocupa los bits 8-31.

operando3: indicadores (flags), permiten establecer encadenamiento de datos o comandos, saltarse áreas, etc. Ocupa los bits 32-36.

operando4: cantidad de bytes (Count), expresión absoluta que indica la cantidad de bytes que se transfieren o saltan. Ocupa los bits 48-63.

Significado de los bits indicadores (FLAGS)

BIT32. Encadenamiento de datos (Chain Data–CD). Si está en ON (1) indica que el área designada por el próximo comando utiliza la operación indicada en el primer comando del último grupo con encadenamiento de datos.

BIT33. Encadenamiento de comandos (Chain Command–CC). Si está en ON (1) indica que quedan comandos por procesar. Si está en OFF (0) indica que ese es el último comando que se ejecuta.

BIT34. Suprime error de longitud (Suppress Length Information–SLI). Si está en ON (1) y el bit CD está en OFF (0), en la última CCW usada queda suprimida la indicación de longitud incorrecta. Si están en ON los bits CC y SLI, tiene lugar el encadenamiento de comandos.

En la tabla que sigue se indican los efectos y acciones que produ-

ce SLI en combinación con CD y CC. La entrada "longitud incorrecta (LI)" significa que la indicación está disponible para el programa en la CSW, un doble guión significa que la indicación se suprime, parada, que se detiene la operación en el subcanal.

Bits Indicadores			*Acción e Indicación*	
CD	CC	SLI	OP. NORMAL	OP. INMEDIATA
0	0	0	Parada, LI	Parada, —
0	0	1	Parada, —	Parada, —
0	1	0	Parada, LI	Encad. comandos
0	1	1	Encad. comandos	Encad. comandos
1	0	0	Parada, LI	Parada, —
1	0	1	Parada, LI	Parada, —
1	1	0	Parada, LI	Parada, —
1	1	1	Parada, LI	Parada, —

BIT 35. Salta (Skip–SK). Si está en ON (1), especifica la supresión de transferencia de información al almacenamiento principal durante una operación de lectura, lectura hacia atrás, o consulta. Si está en OFF (0), tiene lugar la transferencia normal de datos.

BIT 36. Interrupción controlada por programa (Program Controlled Interruption–PCI). Si está en ON (1), determina que el canal genera una interrupción una vez extraída la CCW. El comando no se ejecuta, y la condición PCI queda establecida en el bit 40 de la CSW.

Se consideran las operaciones fundamentales READ y WRITE cuyos códigos de operación correspondientes son:

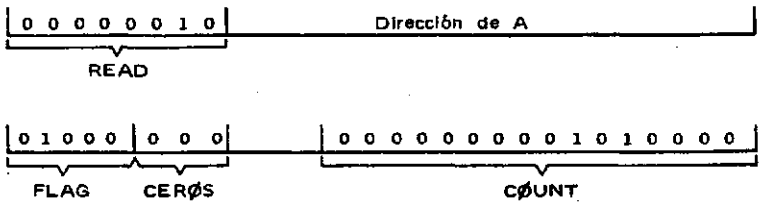
	Binario	*Hexadecimal*
READ	00000010	X'02'
WRITE	00000001	X'01' no produce espaciado X'09' un espacio después de imprimir.

Ejemplo 77.

Leer 3 tarjetas y ubicarlas en las zonas A, B, y C de memoria:

```
R       EQU     X'02'
A       DS      CL80
B       DS      CL80
C       DS      CL80

        CCW     R,A,X'40',80
        CCW     R,B,X'40',80
        CCW     R,C,0,80
```

Los distintos operandos ocuparán los lugares respectivos en la CCW como se indica en el siguiente ejemplo:



Ejemplo 78.

Se desea leer un registro de 80 caracteres de tal forma que:

Los primeros 20 vayan a A
los siguientes 50 vayan a B
los siguientes 5 vayan a C
los siguientes 5 vayan a D.

```
R       EQU     X'02'
CD      EQU     X'80'
A       DS      CL20        1 0 0 0 0 0 0 0
B       DS      CL50        CD
C       DS      CL5
D       DS      CL5
```

```
         CCW     R,A,CD,20
         CCW     0,B,CD,60
         CCW     0,C,CD,5
         CCW     0,D,X'00',5
```

Si hay encadenamiento de datos, el código que aparece en el lugar de operando 1 es ignorado en los comandos que siguen al primero del grupo con CD.

Combinaciones de CD y CC:

CD	*CC*	*Acción*
0	0	No hay encadenamiento. La CCW corriente es la última.
0	1	Encadenamiento de comandos
1	0	Encadenamiento de datos
1	1	Encadenamiento de datos.

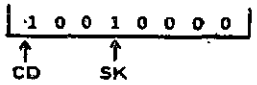
Ejemplo 79.

Se quiere leer un registro de 80 caracteres de tal forma que:

Los primeros 20 vayan a A

los últimos 15 vayan a A+20

```
R        EQU     X'02'
CDS      EQU     X'90'
A        DS      CL35
```




```
         CCW     R,A,CD,20
         CCW     0,0,CDS,45
         CCW     0,A+20,0,15
```

Ejemplo 80.

Leer un registro de 100 caracteres de tal forma que:

Los primeros 10 se salten

los siguientes 15 vayan a A

los siguientes 20 se salten

los siguientes 25 vayan a A+15

los últimos se salten

```
R       EQU     X'02'
CD      EQU     X'80'
S       EQU     X'10'
CDS     EQU     X'90'
A       DS      CL40

        CCW     R,0,CDS,10
        CCW     0,A,CD,15
        CCW     0,0,CDS,20
        CCW     0,A+15,CD,25
        CCW     0,0,S,30
```

Variante. Definiendo SLI EQU X'A0' |1 0,1 0,0 0 0 0| se puede colocar en lugar del penúltimo comando y siguiente

```
CCW     0,A+15,SLI,25
```

el error que se produzca por longitud incorrecta se suprime.

Ejemplo 81.

Grabar 132 caracteres que están en EDIT:

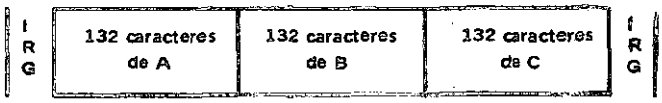
```
W       EQU     X'01'
EDIT    DS      CL132

        CCW     W,EDIT,0,132
```

Ejemplo 82.

Grabar tres registros A, B y C de 132 caracteres cada uno, como un solo registro de cinta.

```
CCW     W,A,CD,132
CCW     0,B,CD,132
CCW     0,C,0,132
```

Resultado:

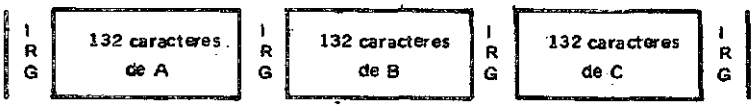


Ejemplo 83.

Grabar tres registros A, B y C de 132 caracteres cada uno, como tres registros físicos de cinta.

```
CCW     W,A,CC,132
CCW     W,B,CC,132
CCW     W,C,0,132
```

Resultado:



Las macro instrucciones que se utilizarán son:

Macro CCB (Command Control Block)

Formato:

nombre del bloque CCB SYSnnn, nombre del programa de canal

donde:

nombre del bloque: es el nombre que identifica la CCB
SYSnnn : nombre simbólico de la unidad a la que está asociada la CCB
nombre del programa del canal: nombre que identifica la primera CCW.

Macro EXCP (Execute Channel Program)

La operación de lectura se realiza sin que el proceso continúe debido a la macro-instrucción WAIT LEET1. Al finalizar la lectura se ejecuta la instrucción de assembler CLC A(2),=C'/* que permite detectar el término de los datos, esto es, que se ha leído la tarjeta que tiene /* en las columnas 1 y 2 respectivamente. Si eso no ha ocurrido, el proceso continúa imprimiendo la información de la tarjeta leída e iniciando nuevamente el ciclo.

12. *Definiciones de macros*

Una definición de macro consiste de:

a) Una proposición de encabezamiento de la definición de la macro, cuyo formato es:

NOMBRE	OPERACION	OPERANDOS
Blanco	MACRO	Blanco

b) Una proposición prototipo de la macro-instrucción cuyo formato es:

NOMBRE	OPERACION	OPERANDOS
Parámetro simbólico o blanco	*Símbolo*	Ninguno, uno o varios parámetros simbólicos

El objeto de la proposición prototipo es especificar el código de operación mnemotécnico y el formato de todas las macro-instrucciones que se refieren a la definición de la macro. Los parámetros simbólicos se utilizan en la definición de la macro-instrucción para representar el campo de nombre y los operandos en la macro-instrucción correspondiente. En el campo operando pueden existir desde 0 hasta 200 parámetros simbólicos separados entre sí por coma.

c) Ninguna, una o varias proposiciones modelo, instrucciones de ensamble condicional, etc. Una proposición modelo consta de uno a cuatro campos que son de izquierda a derecha: nombre, operación, operando y comentario. De las proposiciones modelo que están en la definición de la macro se generan las secuencias deseadas de proposiciones de lenguaje de ensamble.

d) Una proposición de salida de la definición de la macro, cuyo formato es:

NOMBRE	OPERACION	OPERANDOS
Blanco o símbolo de secuencia	MEND	Blanco

Ejemplo 85.

Se tiene la siguiente definición de macro:

```
        MACRØ                       Encabezamiento
&SUM    SUMAR  &ØP1,&ØP2            Prototipo
&SUM    ST     5,AREA               Modelo
        L      5,&ØP1               Modelo
        A      5,&ØP2               Modelo
        LR     6,5                  Modelo
        L      5,AREA               Modelo
        MEND                        Salida
```

y la macro-instrucción

```
AMASB   SUMAR  A,B
```

de acuerdo a ella se generarán las siguientes proposiciones de lenguaje de ensamble:

```
AMASB   ST     5,AREA               Generada
        L      5,A                  Generada
        A      5,B                  Generada
        LR     6,5                  Generada
        L      5,AREA               Generada
```

Ejemplo 86.

```
        MACRØ
&CAM    CAMB   &M,&DAT,&ØP1,&ØP2
&CAM    ST&M   5,AREA               Definición
        L&M    5,&DAT&ØP1             de la
        ST&M   5,&DAT&OP2             Macro
        L&M    5,AREA
        MEND

CAMBIØ  CAMB   M,DATØS,A,B          Macro-instrucción

CAMBIØ  STM    5,AREA
        LM     5,DATØSA             Instrucciones
        STM    5,DATØSB               generadas
        LM     5,AREA
```

A. *Instrucciones de ensamble condicional*

Las instrucciones de ensamble condicional le dan las siguientes posibilidades al programador:

Ejemplo 87.

A constante de dirección tipo A
C constante carácter
F constante de punto fijo, palabra completa
I instrucción de máquina
M macro-instrucción
W comando CCW
etc.

2) Atributos de longitud (L'), escala (S') y entero (I').

Los atributos de longitud, escala y entero son *valores numéricos*. Pueden ser de un símbolo o de operandos de macro-instrucciones.

Ejemplo 88.

```
A       DS      CL10
B       DC      CL5'/*/+Ø'
C       MVC     A(L'B),B
D       MVC     A+L'B-1(L'B),B
```

El atributo longitud de A es 10 y el de B es 5. La proposición C mueve 5 bytes desde B a la dirección A. La proposición C mueve 5 bytes desde B a la dirección A+5–1, esto es, a la dirección A+4.

El atributo longitud del símbolo * es igual a la longitud de la instrucción donde aparece, excepto en la instrucción

```
EQU   *
```

donde se considera el valor 1.

3) Los atributos de escala y entero se asignan a aquellos símbolos que aparecen en el campo nombre de proposiciones DC o DS de punto fijo, punto flotante y decimal. En los dos primeros casos el atributo de escala está dado por el modificador de escala y el atributo entero es una función de S' y L'. Para decimal, en cambio, el atributo escala es el número de dígitos decimales que aparecen a la derecha del punto decimal (empaquetado y zona-dígito) y el atributo entero es el número de dígitos decimales que aparecen a la izquierda del punto decimal (zona-dígito).

a) Punto fijo I' = 8*L'–S'–1

Ejemplo 89.

```
A       DC      HS9'-42.77'
B       DC      FS11'52.5E-1'
```

L' de A = 2 S' de A = 9
I' = 8*2–9–1=6

L' de B = 4 S' de B = 11
I' = 8*4–11–1=20

b) Punto flotante I'=2*(L'−1)−S'

c) Decimal I'=L'−S' (zona-dígito)
I'=2*L'−S'−1 (empaquetado)

Ejemplo 90.

```
        A       DC      P'−5.28'
        B       DC      P'77.09'
        C       DC      Z'15.29'
        D       DC      Z'−87.61'
```

A	L' = 2	S' = 2	I' = 1
B	L' = 3	S' = 2	I' = 3
C	L' = 4	S' = 2	I' = 2
D	L' = 4	S' = 2	I' = 2

4) Atributo cuenta (K')

El atributo cuenta está dado por el número de caracteres que tiene el operando en la macro-instrucción (se excluye la coma precedente y la siguiente). Si el operando es una sublista, el atributo cuenta está dado por todos los caracteres, incluidos los paréntesis inicial y final y las comas que separan los operandos de la sublista.

5) Atributo número (N')

El atributo número está dado por el número de operandos que contiene una sublista. Se puede calcular el número de operandos como igual al número de comas en la sublista, más uno.

Ejemplo 91.

```
(A,B,C,D,E)      5 operandos
(A,B,C, ,E)      5 operandos
(A,B, ,D, ,)     6 operandos
```

D. *Símbolos de secuencia*

El símbolo de secuencia está formado por un punto seguido de 1 a 7 caracteres alfanuméricos, el primero de los cuales debe ser alfabético.

Ejemplo 92.

```
. ALFA1          . SAG73
. BETA2          . XYW
. Z              . $56
```

El campo nombre de una proposición puede contener un símbolo de secuencia, el cual sirve como meta de salto para instrucciones que permiten alterar la secuencia de proceso de las proposiciones o, lo que es lo mismo, de generación de instrucciones.

```
&A          SETA    2
            A       5,&ØP2&A
            LR      6,5
            L       5,AREA
            MEND

AMASB       SUMAR   A,B

AMASB       ST      5,AREA
            L       5,A1
            A       5,B2
            LR      6,5
            L       5,AREA
```

Un símbolo SETA se puede utilizar como subíndice de un parámetro simbólico, esto es, encerrado entre paréntesis a continuación de él. En este caso indica un operando de una sublista de operandos que reemplaza al parámetro simbólico.

Ejemplo 98.

```
            MACRØ
&SUM        SUMAR   &RUG,&SUBLIS
            LCLA    &INDEX
&INDEX      SETA    N'&SUBLIS
&SUM        ST      &RUG,AREA
            L       &RUG,&SUBLIS(&INDEX)
            A       &RUG,&SUBLIS(1)
            LR      6,&RUG
            L       &RUG,AREA
            MEND

AMASB       SUMAR   5,(A,B,C,D)

AMASB       ST      5,AREA
            L       5,D
            A       5,A
            LR      6,5
            L       5,AREA
```

F. SETC

La instrucción SETC permite asignar un valor carácter a un símbolo SETC.

Formato de la instrucción:

Un símbolo SETC	SETC	Un operando

Como operando se puede tener: un attributo tipo, una notación de subcadena, una expresión carácter o una concatenación de estas dos últimas.

En el operando puede aparecer un símbolo SETA. En este caso el valor del símbolo se representa como un valor decimal, sin signo y sin ceros no significativos.

1) Atributo tipo

Se asigna la letra que corresponde al operando que reemplaza al parámetro simbólico, al símbolo SETC.

Ejemplo 99.

```
&SIMB       SETC     T'&PARAM
```

2) Expresión carácter

Una expresión carácter está formada por una cadena de caracteres encerrada entre apóstrofos. De la cadena de caracteres se asignan los ocho caracteres de la izquierda al símbolo SETC.

Se pueden concatenar expresiones carácter a través de la colocación de un punto entre el último apóstrofo de una expresión y el primer apóstrofo de la que sigue.

Ejemplo 100.

```
&CØNCAT     SETC     'ABCDEFGH'
```

Se puede escribir como:

```
&CØNCAT     SETC     'ABCD'.'EFGH'
```

o también,

```
&CØNCAT     SETC     'ABC'.'DEF'.'GH'
```

Los símbolos variables se concatenan de acuerdo a las reglas vistas anteriormente.

Ejemplo 101.

```
&SYMB1      SETC     'ABCD'
&SYMB2      SETC     '&SYMB1.EFGH'
&SYMB3      SETC     '&SYMB1'.'EF'
```

Las instrucciones anteriores asignan a :

```
&SYMB1      el valor  ABCD          a
&SYMB2      el valor  ABCDEFGH   y a
&SYMB3      el valor  ABCDEF
```

Si se trata de representar un & exceptuando el que forma parte de

un símbolo variable, se deben especificar dos &, los que forman parte a su vez de la cadena de caracteres asignada al símbolo SETC.

Ejemplo 102.

```
&SYMB4      SETC      'DØS&&'
```

La instrucción permite asignar a &SYMB4 el valor carácter DOS&&.

3) Notación de subcadenas

Mediante la notación de subcadenas se puede asignar parte de un valor carácter a un símbolo SETC. El formato utilizado es el siguiente:

expresión carácter (expresión aritmética, expresión aritmética)

donde, la primera expresión aritmética indica el primer carácter de la expresión carácter que será asignado al símbolo SETC y la segunda expresión aritmética, el número de caracteres consecutivos que van a ser asignados.

Las dos expresiones aritméticas constituyen la subcadena de valor carácter.

Ejemplo 103.

Se tienen los siguientes símbolos SETA:

```
&ARIT1  =  5
&ARIT2  =  3
```

y los símbolos SETC :

```
&CARA1  =  ABCDEFG
&CARA2  =  XUWX
&CARA3  =  $º/oA1
```

De acuerdo a los valores anteriores, las subcadenas que figuran a continuación entregarán los resultados que se indican:

'&CARA1'(3,2)	valor	CD
'&CARA1'(&ARIT2,3)	valor	EFG
'&CARA2'(1,&ARIT2+1)	valor	XUWX
'&CARA3'(&ARIT2+1,1)	valor	1

4) Concatenación de notaciones subcadena y expresiones carácter.

Una expresión carácter puede ser concatenada con una notación subcadena colocando un punto entre el último apóstrofo de la expresión carácter y el primer apóstrofo de la notación subcadena.

Ejemplo 104.

Utilizando los mismo símbolos SETA y SETC del ejempo 103, las concatenaciones siguientes dan los valores que se indican:

```
'&CARA2'.'&CARA3'(&ARIT1,&ARIT2)      valor      $º/oA
'&CARA1'.'&CARA3'(1,&ARIT1+4)         valor      ABCDEFG$º/o
```

Se pueden concatenar notaciones subcadena o notaciones subcadena con expresiones carácter sin necesidad de colocar punto, esto es, el punto es opcional.

Ejemplo 105.

```
'&CARA1'(3,2)'&CARA2'(1,&ARIT2+1)
'&CARA1'(&ARIT2,3)'&SYMB1'
```

Ejemplo 106.

```
          MACRØ
&SUM      SUMAR     &ØP1,&ØP2
          LCLC      &A
&A        SETC      'AREA'
&SUM      ST        5,&A
          L         5,&ØP1
          A         5,&ØP2
          LR        6,5
          L         5,&A
          MEND

AMASB     SUMAR     A,B

AMASB     ST        5,AREA
          L         5,A
          A         5,B
          LR        6,5
          L         5,AREA
```

Ejemplo 107.

```
          MACRØ
&SUM      SUMAR     &RUG,&DAT,&ØP1,&ØP2
          LCLC      &A,&B,&C
&A        SETC      '&DAT'
&B        SETC      'ALFABETA'
&C        SETC      '&B'(&ØP1,4)
&SUM      ST        &RUG,&A
          L         &RUG,&C
&C        SETC      '&B'(&ØP2,4)
          A         &RUG,&C
          LR        6,&RUG
```

```
         L        &RUG,&A
         MEND

AMASB    SUMAR    5,AREA,1,5

AMASB    ST       5,AREA
         L        5,ALFA
         A        5,BETA
         LR       6,5
         L        5,AREA
```

Ejemplo 108.

```
         MACRØ
&SUM     SUMAR    &RUG1,&RUG2,&AR1,&AR2
         LCLC     &A,&AREA1,&AREA2
&A       SETC     '&AR2'
&AREA1   SETC     '&AR1'(1,4)
&AREA2   SETC     '&AR1'(5,4)
&SUM     ST       &RUG1,&A
         L        &RUG1,&AREA1
         A        &RUG1,&AREA2
         LR       &RUG2,&RUG1
         L        &RUG1,&A
         MEND

AMASB    SUMAR    5,6,ALFABETA,AREA

AMASB    ST       5,AREA
         L        5,ALFA
         A        5,BETA
         LR       6,5
         L        5,A
```

G. SETB

La instrucción SETB permite asignar a un símbolo SETB el valor bi nario 0 ó 1.

Formato de la instrucción:

Un símbolo	SETB	0 ó 1, (0) ó (1) o una expresión lógica encerrada entre paréntesis.

Si se especifica una expresión lógica, ésta se evalúa para determinar si es "verdad" o es "falso". En el primer caso se asigna al símbolo SETB el valor 1 y en el segundo el valor 0.

La expresión lógica se define como un término o una combinación de términos y operadores lógicos. Los operadores lógicos son:

AND, OR y NOT (cualquiera de los dos primeros puede estar seguido de NOT).

Un término se define como una relación aritmética, una relación de carácter o un símbolo SETB. La relación aritmética es una expresión aritmética conectada con otra a través de un operador de relación. La relación de carácter es una expresión carácter conectada con otra a través de un operador de relación. Los operadores de relación son: EQ, NE, LT, GT, LE, GE.

Tanto los operadores lógicos como los de relación deben estar precedidos y seguidos, al menos, por un blanco.

Ejemplo 109.

```
&SYMB1    SETB    (&EXA1+X'FF')  LT   &EXA2
&SYMB2    SETB    'ABCDE'   GT   '&EXC1'
&SYMB3    SETB    &EXA1 LT &EXA2 AND &EXA1 GT &EXA3
&SYMB4    SETB    T'&ALFA EQ T'&BETA
```

1) Evaluación de expresiones lógicas.

a) Se evalúa cada término y se le asigna su valor lógico (verdadero o falso).

b) Se ejecutan las operaciones lógicas de acuerdo a la siguiente prioridad:

1º NOT
2º AND
3º OR

Si la prioridad es igual se procede de izquierda a derecha.

c) Se asigna el resultado obtenido al símbolo SETB.

2) Aplicación de los símbolos SETB

Los símbolos SETB pueden aparecer en el operando de una instrucción SETA o de una instrucción SETC, en una relación aritmética o de carácter en instrucciones AIF y SETB. De acuerdo a la instrucción o tipo de relación el valor binario del símbolo SETB se convertirá a un valor aritmético o a carácter.

Ejemplo 110.

```
          MACRØ
&SUM      SUMAR   &ØP1,&ØP2,&RUG
          LCLA    &A
          LCLB    &B,&BB
          LCLC    &C
&B        SETB    (N'&ØP1   EQ   5)
&BB       SETB    (K'&ØP1   GT   4)
&A        SETA    &B
&C        SETC    '&BB'
```

```
&SUM    ST      &RUG,&ØP2
        L       &RUG,&ØP2&A
        A       &RUG,&ØP2&C
        LR      6,&RUG
        L       &RUG,&ØP2
        MEND

AMASB   SUMAR   (A,B,C),AREA,5

AMASB   ST      5,AREA
        L       5,AREA0
        A       5,AREA1
        LR      6,5
        L       5,AREA
```

H. AIF *(Salto Condicional)*

La instrucción AIF permite alterar, de acuerdo a una condición, la secuencia en la cual se procesarán las proposiciones de la definición de macro en que ella se encuentre.

Formato de la instrucción:

Símbolo de secuencia o blanco	AIF	Una expresión lógica encerrada entre paréntesis, seguida por un símbolo de secuencia

Si la expresión lógica es 'verdadera' se efectúa un salto a la proposición identificada por el símbolo de secuencia. En caso contrario continúa la secuencia normal.

Ejemplo 111.

```
        MACRØ
        CARGA   &N,&DIR1,&DIR2
        LCLA    &A
&A      SETA    &N
        LA      3,&DIR1
        ST      3,INST+8
        AIF     (&A LE 1).END
        LA      4,&DIR2
        ST      4,INST+12
.END    MEND
```

I. AGO *(Salto Incondicional)*

Similar a la instrucción AIF, difiere sólo en que el salto se efectúa sin que sea necesaria una condición.

Formato de la instrucción:

Símbolo de secuencia o blanco	AGO	Símbolo de secuencia

Ejemplo 112.

```
         MACRØ
         SUMAR   &REG,&AREAS,&RES
         LCLA    &N
&N       SETA    1
         SR      &REG,&REG
.SUM     A       &REG,&AREAS(&N)
&N       SETA    &N+1
         AIF     (&N GT N'&AREAS).FIN
         AGØ     .SUM
.FIN     ST      &REG,&RES
         MEND

         SUMAR   5,(ALF,BET,GAM),RESUL

         SR      5,5
         A       5,ALF
         A       5,BET
         A       5,GAM
         ST      5,RESUL
```

J. ACTR *(Contador de Ciclos)*

La instrucción ACTR permite controlar el número de saltos AIF y AGO que se ejecuten en una definición de macro.

Formato de la instrucción:

Blanco	ACTR	Una expresión SETA

El valor de la expresión SETA inicializa un contador. La instrucción debe ubicarse inmediatamente a continuación de declaraciones globables o locales. Cada vez que se ejecuta un salto AIF o AGO, el contador es disminuido en 1 en su contenido. Al tener un valor cero antes de efectuar la resta se termina el procesamiento de la definición de macro y se continúa con la proposición siguiente del programa principal.

Ejemplo 113.

```
         MACRØ
         CØNVER  &A,&LØNG,&DIR1,&DIR2
         LCLA    &A1,&LØNG1
         ACTR    4
&A1      SETA    &A
.FIRST   PACK    &DIR1,&DIR2.+&LØNG1.(&LØNG)
```

```
         CVB     &A1,&DIR1
&A1      SETA    &A1+1
&LØNG1   SETA    &LØNG1+6
         AGØ     .FIRST
         MEND

         CØNVER  5,6,D,A

         PACK    D,A+0(6)
         CVB     5,D
         PACK    D,A+6(6)
         CVB     6,D
         PACK    D,A+12(6)
         CVB     7,D
         PACK    D,A+18(6)
         CVB     8,D
         PACK    D,A+24(6)
         CVB     9,D
```

K. ANOP *(No operación de ensamble)*

La instrucción ANOP permite realizar saltos a instrucciones que están identificadas por símbolos o símbolos variables.

Formato de la instrucción:

Símbolo de secuencia ANOP Blanco

Ejemplo 114.

```
         MACRØ
&SUMA    SUMAR   &REG,&AREAS,&RES
         LCLA    &N
&N       SETA    1
.SUM     ANØP
&SUMA    A       &REG,&AREAS(&N)
&N       SETA    &N+1
         AIF     (&N LE N'&AREAS).SUM
         ST      &REG,&RES
         MEND
```

L. *Otras facilidades para el programador*

1) MEXIT (Salida de una definición de macro)

La instrucción MEXIT le indica al ensamblador que debe terminar el procesamiento de la definición de macro en que ella aparece.

Formato de la instrucción:

Símbolo de secuencia MEXIT Blanco

Ejemplo 115.

```
            MACRØ
&SUM        SUMAR   &ØP1,&ØP2
            AIF     (T'&ØP1  EQ  'F').SIG1
            MEXIT
.SIG1       AIF     (T'&ØP2  EQ  'F').SIG2
            MEXIT
.SIG2       ANØP
&SUM        ST      5,AREA
            L       5,&ØP1
            A       5,&ØP2
            LR      6,5
            L       5,AREA
            MEND
```

2) MNOTE

La instrucción MNOTE permite generar un mensaje de error, se puede indicar junto con el mensaje un código de gravedad del error. Formato de la instrucción:

Símbolo de secuencia MNOTE Operando

El operando puede ser:

a) código de gravedad del error, 'mensaje'
b) ,'mensaje'
c) 'mensaje'

En b) y c) se supone código de gravedad igual a uno. El código de gravedad puede variar desde 0 hasta 255. Si se especifica asterisco, el contenido del campo operando se imprime como comentario.

Ejemplo 116.

```
            MACRØ
&SUM        SUMAR   &ØP1,&ØP2
            MNØTE   *,'RESULTADØ MACRØ SUMAR'
            AIF     (T'&ØP1  NE  'F').SIG1
            AIF     (T'&ØP2  NE  'F').SIG2
&SUM        ST      5,AREA
            L       5,&ØP1
            A       5,&ØP2
            LR      6,5
            L       5,AREA
            MEXIT
.SIG1       MNØTE   'ØP1 NØ ES DE TIPØ F'
            MEXIT
.SIG2       MNØTE   'ØP2 NØ ES DE TIPØ F'
            MEND
```

```
.FIN1    SETA    &A+1
         LA      &R1,&INC.(&R1)
.FIN2    MEND

BEG      CØMPAR  5,6,SIGA,4
SIGA     CØMPAR  5,6,GØ,8

BEG      CR      5,6
         BE      SIGA
         LA      5,4(5)
SIGA     CR      5,6
         BE      GØ
         LA      5,8(5)
```

BIBLIOGRAFIA

1. Computer Usage Company, *Programación del Sistema IBM/360,* México, Editorial Limusa-Wiley, S.A., 1968, 365 pp.

2. IBM System Products Division, *Introducción para Programadores a la Arquitectura, las Instrucciones y el Lenguaje Compaginador del Sistema IBM/360,* Argentina, 1971, 274 pp.

3. IBM System Products Division, *Assembler Language,* USA, 1967, 155 pp.

4. IBM System Products Division, *IBM System/370 Principles of Operation,* USA, 1974, 326 pp.

5. Kardonsky de F., Adriana y Sánchez C., Víctor, *Curso de Programación,* Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, 1967, 167 pp.

6. Sánchez C., Víctor, *Apuntes de Assembler,* Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, 1967, 170 pp.

7. Sánchez C., Víctor, *Manual de Assembler, Tomo I,* Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, 1970, 102 pp.

8. Steinhart, Robert F. y Pollack, Seymour V., *Programming the IBM System/360,* USA, Holt Rinehart and Winston, 1970, 576 pp.

9. Struble, George, *Assembler Language Programming: The IBM System/360,* USA, Addison-Wesley, 1969, 434 pp.

10. Thomas Alex, Jr., *System 360 Programming,* USA, Rinehart Press San Francisco, 1971, 273 pp.

11. Vickers, Frank, D., *Introduction to Machine and Assembly Language: System/360/370,* USA, Holt Rinehart and Winston, 1971, 303 pp.

I. INTRODUCCION

Se define como Sistema de Operación a un conjunto integrado de programas, diseñado para mejorar la efectividad de la operación total de un Sistema Electrónico de Procesamiento de Datos (computador).

En general está constituido por:

a) Sistemas de programación (traductores de lenguajes) para ayudar a la formulación de problemas, que serán resueltos mediante el computador.

b) Programas de control para ayudar a la operación, mantención y administración del sistema.

c) Programas de chequeo para recuperación de errores.

d) Rutinas de servicio de bibliotecas (residentes en el computador) para almacenar información para procesos posteriores y para contener al propio Sistema de Operación.

e) Rutinas de mantención de bibliotecas, que permiten tenerlas actualizadas.

Varios hechos han ido señalizando las distintas etapas que ha tenido el desarrollo de los Sistemas de Operación. Inicialmente la programación se efectuaba en lenguaje de máquina, lenguaje árido de difícil manejo y fuente continua de errores, más aún, complicado y complejo para ubicar esos mismos errores. Esto trajo como consecuencia la creación de programas traductores que se diseñaron para convertir programas escritos en lenguajes simbólicos a los lenguajes de máquina utilizados hasta ese momento. Esos nuevos lenguajes se denominaron "de ensamble" y estaban orientados a la máquina. Les siguieron los lenguajes simbólicos generales, que permitían a los usuarios escribir sus programas en forma parecida a la utilizada en matemáticas (ALGOL, FORTRAN) o en el planteamiento de problemas de tipo comercial (COBOL).

En esta forma los usuarios de computadores tuvieron la posibilidad de escribir ellos mismos sus programas, en muchos casos sin necesitar la ayuda o asistencia de programadores profesionales, quienes, al mismo tiempo, estuvieron en condiciones de desarrollar un trabajo más creativo.

Junto con los programas traductores de lenguajes se crearon los llamados sistemas de control de entrada/salida de información, cuyo objetivo fundamental era disminuir el tiempo ocioso de la Unidad Central de Proceso, debido a que las operaciones de entrada/salida eran relativamente lentas en comparación con la velocidad de ejecución de la UCP. Para lograr la reducción del tiempo ocioso, en primer lugar, se desarrollaron los sistemas de computación, esto es, los equipos, que permitieron realizar en forma paralela operaciones de entrada/salida y de procesamiento de datos. En segundo lugar, se generaron los sistemas de control de entrada/salida de tal manera que permitieron sincronizar

cenaron en dispositivos de memoria auxiliar y constituyeron la biblioteca de programas.

En la mayoría de los sistemas de operación el programa de control se subdividió en dos partes: una, llamada "núcleo" porque permanecía siempre en la memoria principal y la otra "transitoria", que se cargaba en memoria principal sólo durante el intervalo entre un trabajo y otro o entre etapas de ellos (job steps). El núcleo estaba compuesto fundamentalmente por rutinas de uso frecuente cuya función era procesar las interrupciones, cargar programas desde la biblioteca a memoria principal, transmitir mensajes al operador, etc. Contenía, además, información sobre los dispositivos de entrada/salida y la fecha de cada día. También formaban parte del núcleo rutinas de supervisión necesarias para controlar las operaciones de entrada/salida. En cuanto a la parte transitoria, su función primordial era interpretar y ejecutar las proposiciones de control leídas. La ejecución de las proposiciones de control constituirá el paso de un trabajo a otro o de una etapa de un trabajo a otra. Una vez realizada dicha transición, el espacio de memoria principal ocupado por la parte transitoria quedaba disponible para ser ocupado en la ejecución del trabajo o etapa de éste (job o job step).

Un gran impulso al desarrollo de los sistemas de operación le dieron algunas aplicaciones en las que se necesitaba una respuesta rápida a consultas hechas desde dispositivos de entrada/salida locales o remotos. En estas aplicaciones se utilizaron sistemas de operación que se comunicaban directamente con la fuente y destino de los datos que se procesaban. Estos eran enviados y recibidos desde distintos lugares que podían estar cercanos o remotos al computador, en este último caso conectados mediante líneas de telecomunicación. Estos sistemas para poder dar respuestas rápidas hicieron uso, en primer lugar, de los dispositivos de acceso directo que permiten llegar a los registros sin que éstos tengan que ser clasificados en un orden determinado con anterioridad a su utilización. En segundo lugar, se aplicaron nuevas técnicas que permitieron procesar varias transacciones en forma paralela. Para esto, el programa de control mantenía canales de recursos de información y de equipo y los asignaba a medida que fueran requeridos. Si se suspendía el proceso de una transacción, los recursos que dejaba disponibles se asignaban para iniciar el proceso de otra o continuar alguno que hubiera sido interrumpido previamente. Los sistemas que se obtuvieron se conocen como sistemas de tiempo compartido (time-sharing) y se refieren en general a aquéllos en los cuales los usuarios son independientes entre sí respecto al uso del computador o más claramente cada usuario entra información, la procesa y recibe resultados tal como si todo el computador estuviera a su disposición. Como ejemplo típico de estos sistemas está el desarrollado para resolver el problema de venta de pasajes de líneas aéreas desde distintas agencias. En este caso se requerían respuestas rápidas del sistema, de tal manera que en una agencia no se

vendiera un pasaje para un asiento que acababa de ser usado por otra agencia.

Otro tipo de aplicación es la de tiempo real (real-time) que corresponde a procesos que generan información que es elaborada por el computador y cuyos resultados permiten controlar o alterar los mismos procesos. En esta aplicación el computador está conectado directamente con la fuente emisora y receptora de información; la velocidad de la respuesta varía de acuerdo con el tipo de proceso, que puede ser el control de una máquina herramienta, de una fábrica de papel, de una industria petroquímica o de las luces de tráfico de una ciudad.

Es necesario definir el concepto de multiprogramación y éste corresponde a la ejecución en forma paralela de más de un programa. Para ello hay intercalación de ejecución de un programa con otro de acuerdo con las prioridades de ellos. Es fácil confundir multiprogramación con tiempo compartido, considerando que en ambos casos se trata de efectuar procesos paralelos; sin embargo, en el caso de tiempo compartido hay interacción entre el usuario y su programa, esto es, el usuario está en todo momento al tanto del progreso de su programa, de los errores que ocurren, que puede corregir inmediatamente, y de los resultados. En tiempo compartido hay una conversación entre el usuario y el computador. Se puede decir, sin temor a cometer un error, que tiempo compartido es una opción que puede funcionar bajo multiprogramación. Otro concepto es el de multiprocesamiento que corresponde a una técnica mediante la cual el procesamiento de datos es realizado entre dos o más UCP interconectadas de tal manera que la comunicación entre ellas, directa o indirecta, es realizada por el sistema de operación sin interrupción humana.

Finalmente, se han puesto en ejecución sistemas en los cuales se utiliza el concepto de memoria virtual, esto es, que permiten la ejecución de programas que exceden la capacidad de la memoria principal real disponible. Para ello se hace uso de la técnica de paginación, en la cual el programa se divide físicamente en páginas de las que sólo algunas necesitan residir en determinados momentos en la memoria principal, o la técnica de segmentación, en que el programa se divide en segmentos en vez de páginas; o de ambas técnicas combinadas. Algunos fabricantes establecen una diferencia entre los conceptos de página y segmento. La primera constituiría una unidad de tamaño fijo. En cambio, el segmento representaría partes de programa, relacionadas con la idea que tiene el usuario de la construcción lógica de su programa. Además, el usuario puede referirse a dichos segmentos mediante nombres que los identifican.

Fundamental en la aplicación del concepto de memoria virtual fue la "traducción dinámica de direcciones" (direcciones de memoria virtual a ubicaciones físicas) mediante el mismo equipo (hardware).

II. FUNCIONES DEL SISTEMA DE OPERACION

1. *Función de planificación*

El objetivo de la función de planificación es seleccionar un trabajo (job) de aquéllos que se encuentren disponibles para ser procesados y dejarlo en condiciones de ser ejecutado. El criterio aplicado para planificar depende de la modalidad de operación: procesamiento loteado (batch), tiempo real o tiempo compartido. Lo normal es asignar prioridades a cada tipo de programa o suceso que ocurra.

En el procesamiento batch la prioridad de ejecución puede estar dada por el orden que tenga el trabajo en la cola de entrada, lo que significa que son procesados a medida que van entrando. Para alterar ese orden, la prioridad de ejecución puede hacerse a través de un parámetro; en este caso, el lote de trabajos se lee y almacena en memoria auxiliar y a continuación son ejecutados de acuerdo con la prioridad que tienen.

Otras posibilidades de planificación importantes son la condicional y la algorítmica. En la primera la ejecución de una etapa del trabajo (job step) puede estar condicionada a la presencia o ausencia de errores en una etapa previa o a la modificación de indicadores internos realizada por esas etapas. En la planificación algorítmica se selecciona el programa de acuerdo con la relación que exista entre determinados factores, relación que es el *algoritmo de planificación* donde los factores pueden ser, por ejemplo, tiempo estimado de proceso, tiempo que el trabajo ha estado en la cola, etc.

En el procesamiento en tiempo real lo normal es que la función de planificación se obtenga mediante el mismo equipo (hardware). Sin embargo, existen también sistemas en que la planificación se realiza mediante programas (software). Si la función de planificación es asignación de prioridades, se ejecuta siempre el suceso que tenga la más alta, si se presenta uno con prioridad mayor se suspende el que se estaba ejecutando y los recursos del sistema quedan disponibles para el nuevo suceso.

En el procesamiento en tiempo compartido se tiene una rutina de planificación que determina en primer lugar si el sistema está saturado en el momento en que se presenta un nuevo usuario, para permitirle proceder o no. En algunos sistemas, si se autoriza la entrada al usuario, éste debe dar a conocer sus requerimientos de recursos y no lo planifican mientras esos recursos no estén disponibles. En otros sistemas el usuario queda colocado inmediatamente en una modalidad de ejecución en espera de que los recursos que necesita queden disponibles. Normalmente, en los sistemas que soportan aplicaciones que son de tiempo compartido y aplicaciones que no lo son, tienen prioridad los requerimientos de los usuarios del primer tipo debido a su modalidad de conversación.

La mayoría de los sistemas que trabajan en multiprogramación manejan más de un tipo de modalidad de operación, por ejemplo, procesamiento batch con tiempo compartido. En esos casos, si una modalidad de operación no tiene carga de trabajo, otra de las modalidades que están siendo manejadas utiliza los recursos que han quedado disponibles.

2. *Función de administración de recursos*

En general, los recursos que deben ser administrados por el sistema de operación son: memoria principal, tiempo de Unidad Central de Proceso (UCP), dispositivos de entrada/salida (input/output-I/O)y archivos de información.

A. *Administración de memoria*

El problema fundamental que se presenta en la administración de memoria es el de tratarla como un recurso que debe ser desocupado antes de asignarla a un programa distinto del que la está ocupando. El problema, evidentemente, no se presenta en un sistema de proceso serial, en el que todos los recursos están disponibles para el problema que se está ejecutando, sino en los sistemas de multiprogramacion, de tiempo compartido o de tiempo real, en los cuales hay más de un programa o suceso que está compitiendo por recursos.

La dificultad que se puede presentar es la de entrar en un círculo vicioso, que se origina, por ejemplo, al tener dos programas en memoria, procesándose en forma simultánea. Si ocurre que en un momento determinado los dos programas requieren más memoria, ninguno de los dos puede terminarse hasta que no haya obtenido la memoria que le hace falta y que se la puede proporcionar el otro.

Otro problema que es necesario resolver es el tropiezo que implica la necesidad de volver a utilizar las mismas posiciones de memoria que se tenían antes de desocuparla. Si esta necesidad no se elimina, no es posible atender en forma paralela dos programas que están simultáneamente en memoria y requieren más almacenamiento, pues uno de ellos, el de prioridad más baja, tendrá que ceder su espacio al de prioridad mayor y no podrá ser cargado nuevamente hasta que no sea desocupada "su" área de memoria. Este problema se conoce como "estancamiento" (deadlock).

Existen cinco métodos básicos que permiten asignar memoria. En el primer método se asignan, en forma estática, áreas fijas de memoria para proceso y esta asignación implica al mismo tiempo prioridad de ejecución. Las áreas se denominan "de primer término" (foreground) y "de último término" (background). Las áreas de primer término están destinadas, en un sistema complejo de multiprogramación, a procesos de tiempo real o de tiempo compartido y el área de último término a

proceso batch. En sistemas más sencillos de multiprogramación, las áreas de primer término están destinadas a procesos que tienen un alto porcentaje de uso de la UCP y bajo porcentaje de operaciones entrada/salida; en cambio, el área de último término se destina a procesos de características inversas.

En el segundo método se tiene la memoria dividida en áreas fijas (particiones) de tamaño fijo, en algunos sistemas, y de tamaño variable en otros. Cada partición puede tener un flujo de entrada aparte, o los programas se reciben de un flujo único y se asignan a las distintas particiones de acuerdo con parámetros proporcionados junto con el trabajo o éstos se asignan a las particiones disponibles más pequeñas que puedan contenerlos.

Se puede decir que ambos métodos presentan los defectos siguientes:

a) Cada trabajo debe diseñarse de tal manera que no exceda en su requerimiento de memoria, la máxima que le corresponde a su partición.
b) Durante el tiempo ocioso del trabajo se continúa reservando la memoria de la partición a la que está asignado.
c) En el caso de particiones fijas de tamaño fijo, debe distribuirse la memoria de modo que haya pocas particiones grandes y bastantes particiones pequeñas, pero esto trae como consecuencia que si hay muchos trabajos chicos el sistema no puede atender a una cantidad mayor debido a las particiones grandes y, por el contrario, si hay trabajos grandes el sistema no puede atender a una cantidad mayor a causa de las particiones pequeñas.

En el tercer método se tiene la memoria libre como un solo grupo de almacenamiento. De este grupo se le asigna a cada trabajo la cantidad exacta de memoria que necesita y cuando el trabajo termina devuelve al grupo la memoria que le ha sido asignada.

A pesar de ser este método más dinámico que los anteriores, se sigue presentando un problema que se denomina de fragmentación. Dado que una vez que se asignó espacio de memoria éste debe quedar reservado hasta que el trabajo sea concluido, existirá una serie de pequeñas áreas disponibles (fragmentos) que en conjunto podrían permitir atender otro trabajo pero que por estar reservadas no pueden ocuparse y además no son contiguas.

En el cuarto método se dividen, tanto la memoria principal como el programa y los datos, en páginas, que son áreas de memoria de tamaño relativamente pequeño. Con este método se soluciona en gran parte el problema de fragmentación, por cuanto al ser dividido un programa en páginas se necesitará bastante menos memoria que en el caso de estar agrupado el programa completo; luego, todo el espacio que no se usa queda disponible para otros programas.

Hablar de paginación es hacerlo de memoria virtual y de los conceptos que ella implica. En primer lugar se establece un "espacio de direcciones" (memoria virtual) que empieza en cero y se extiende hasta el máximo permitido por el método de dirección del sistema. Todas las referencias a la memoria principal deben ser hechas en términos de direcciones de memoria virtual. Considerando que la memoria virtual es mayor que la memoria real (física), se debe efectuar o establecer una correspondencia entre la dirección de memoria virtual y la dirección de memoria física. De ahí que se tiene que la definición de dirección de memoria virtual dice que "es un identificador de un pedazo de información requerido, pero no una descripción del punto de memoria principal en que está ese pedazo". La correspondencia es obtenida a través del mismo equipo (hardware), lo que se conoce como "traducción dinámica de direcciones". La parte de memoria virtual que excede a la memoria física se almacena en memoria auxiliar, de tal manera que si se solicita una dirección que no esté en la memoria principal se produce una interrupción que permite cargar la página que contiene la información solicitada.

En el quinto método se divide tanto la memoria principal como el programa y los datos en segmentos, los cuales pueden ser identificados mediante nombres por el usuario. La ventaja de este método, aparte de la posibilidad de identificar las particiones, es la de efectuar una división más ligada a la estructura modular que le da el usuario a su programa.

En los sistemas de operación desarrollados para sistemas de computación grande se ha aplicado una combinación de los dos últimos métodos, esto es, una división en segmentos y dentro de ellos una subdivisión en páginas.

B. *Administración de tiempo de la* UCP

Se puede decir que la administración de tiempo de la UCP es la función medular en un sistema de multiprogramación, tiempo real o tiempo compartido. Se trata de distribuir o compartir el tiempo entre programas que se están ejecutando en forma simultánea. La acción de asignar tiempo a esos programas se denomina "despacho" (dispatch).

Para poder asignar tiempo de UCP se utilizan distintos sistemas de despacho de "colas" de trabajos, los cuales están ordenados dentro de ellas de acuerdo con su prioridad, desde la más alta hasta la más baja. Además de la prioridad en la cola, se tiene la prioridad que le corresponde a la partición y en algunos casos dentro de la partición la prioridad que le ha sido asignada a la clase de trabajo. Así, la cola de trabajos correspondería a los de la misma clase en una partición.

El programa es ejecutado hasta que se produce una interrupción generada por un programa de mayor prioridad (de otra clase o de otra partición) o por el mismo programa debido a necesidad de servicios del

correspondientes. Conjuntamente con los resultados parciales o totales obtenidos, en algunos casos se proporciona un vaciado (dump) de las zonas de memoria ocupadas, contenidos de registros, indicadores, etc.

5. *Función de comunicación*

La función de comunicación comprende todo el intercambio de información entre el programador o el operador y el sistema de operación. La información puede ser para controlar la ejecución de los trabajos, para configurar los recursos que se necesitarán o para dar cuenta de distintos estados o sucesos del trabajo.

Se tiene comunicación no interactiva e interactiva. La primera corresponde a aquellas modalidades de procesamiento en las cuales la información se entrega al sistema de operación a través de tarjetas de control o de comandos proporcionados mediante una operación de teclado manejado por el operador en su dispositivo de entrada/salida y se recibe del sistema por medio del mismo dispositivo, por una impresora rápida o por otro dispositivo de salida. La segunda se refiere a la comunicación existente en una modalidad de tiempo compartido.

III. PUESTA EN MARCHA DEL SISTEMA

El operador efectúa la puesta en marcha para iniciar el sistema de operación para proceso normal. El proceso generalmente se realiza cada día una vez que el computador ha sido conectado, principalmente porque la carga de trabajo no es suficiente como para tener el computador funcionando las veinticuatro horas del día o porque es necesario inicializar algún tipo especial de rutinas. En los sistemas de tiempo real, en cambio, se pone en marcha el sistema de operación, que continúa funcionando día tras día. En las dos modalidades, sin embargo, se debe realizar la puesta en marcha cada vez que hay una caída del sistema (detención total por algún tipo de anormalidad).

IV. GENERACION DEL SISTEMA

La generación del sistema consiste en construir un sistema de operación de acuerdo con la configuración del equipo con que se cuenta y con las necesidades específicas de los usuarios de dicho equipo.

La firma vendedora proporciona un sistema de operación maestro a partir del cual se construye el que se utilizará en la instalación. Ese sistema de operación maestro contiene todas las rutinas requeridas para atender cualquier dispositivo de la configuración, como asimismo rutinas opcionales desarrolladas por la firma vendedora. Contiene, además, un conjunto de programas que procesarán la información proporcionada por el usuario para generar el sistema que desea. Esa información estará compuesta por características físicas del equipo (límites de

memoria, tamaños de las particiones, tipo de dispositivo, etc.), nombres simbólicos asignados a dispositivos o a clases de dispositivos, traductores que serán incorporados, programas utilitarios, etc.

V. UN SISTEMA DE OPERACION: DISK OPERATING SYSTEM / VIRTUAL STORAGE DE IBM (DOS/VS)

1. *Programas componentes del sistema* DOS/VS

A. *Programas de control*

a) Cargador de programa inicial (Initial program loader – IPL) que se utiliza para la puesta en marcha del sistema. Carga el programa Supervisor desde un dispositivo de acceso directo a memoria.

b) Supervisor, controla la operación total del sistema y proporciona funciones generales requeridas por el programa de control de trabajos (jobs) y todos los programas de procesamiento. Reside en el área más baja de memoria denominada área del supervisor.

c) Programa de control de trabajos (job control programs) es cargado por el supervisor para iniciar la ejecución de cada programa y para establecer cuáles facilidades del sistema van a ser invocadas mientras el programa está corriendo.

B. *Programas de procesamiento*

Se pueden dividir en tres categorías:

a) Traductores de lenguajes, que son los que traducen los programas fuente escritos en algún lenguaje de programación a lenguaje de máquina (programa objeto).

b) Programas de servicio. Entre los más importantes están:

i) El Linkage Editor, que convierte los programas objeto a programas objeto ejecutables.

ii) El Librarian, que realiza funciones de mantención y de servicio de las bibliotecas del sistema. Se tienen tres bibliotecas: la biblioteca imagen de memoria (Core Image Library – CIL) donde están todos los programas objeto ejecutables, la biblioteca reubicable (Relocatable Library – RL) donde están los programas objeto reubicables, esto es, los resultados de procesos de compilación, y la biblioteca de proposiciones fuente (Source Statement Library) donde están programas en lenguaje fuente.

iii) El Power (Priority Output Writers, Execution Pro-

cessors and Input Readers) que proporciona la posibilidad de leer y grabar flujos de entrada y salida en dispositivos de memoria auxiliar en forma paralela con la ejecución de trabajos. Esto se conoce como "spooling".

iv) Emuladores que permiten ejecutar en el Sistema/370 programas escritos para otros sistemas de computación.

v) Programas de aplicación, escritos por usuarios y en algunos casos proporcionados por IBM, para resolver problemas de tipo científico o comercial.

C. *Rutinas de administración de datos*

Permiten liberar al programador de escribir programas para tareas rutinarias de transferencia de datos entre memoria auxiliar y programas.

2. *Funciones del sistema de operación*

A. *Funciones de control*

Después que el sistema se pone en marcha por medio del IPL, está listo para aceptar información para ser procesada.

La unidad de trabajo que el usuario entrega para que sea procesada se conoce como "job" y el conjunto de jobs se identifica como "flujo de jobs" (job stream).

Cada job y el ambiente en el cual va a ser procesado se define por medio de "proposiciones de control de jobs" con los cuales se obtiene:

a) Transición de job a job con intervención mínima por parte del operador. El comienzo de un job se indica por medio de una proposición de control

```
// JOB
```

Cada job debe estar subdividido en pasos (job step) cada uno de los cuales corresponde a un programa. El job step se identifica con una proposición de control

```
// EXEC
```

b) Asignación de dispositivos de entrada/salida a nombres simbólicos de dispositivos especificados en los programas.

c) Carga de programas ejecutables desde bibliotecas a memoria.

d) Manejo de término de programas.

B. *Utilización de recursos*

a) Administración de memoria. Existen dos modalidades de procesa-

miento y la memoria es organizada de acuerdo con esas modalidades.

i) Procesamiento batch. La memoria se subdivide en dos áreas. La de orden más bajo se destina al Supervisor y el área restante (background) a los programas que van a ser procesados. En memoria principal puede estar sólo un programa a ejecutar. Se proporciona un sistema de protección de memoria que impide que por efectos del proceso del programa pueda ser destruido el Supervisor.

ii) Procesamiento en multiprogramación. En esta modalidad el área que no es ocupada por el Supervisor puede subdividirse en un máximo de cinco particiones. El área de orden más bajo es el área de último término (background) o de menor prioridad de ejecución; las cuatro áreas restantes o áreas de primer término (foreground) se identifican con los nombres FOREGROUND–4 hasta FOREGROUND–1 y siguen en ese orden al área de último término con su respectiva prioridad de ejecución en el orden inverso.

Cada partición puede contener un programa separado, lo que significa que pueden ser procesados en forma paralela hasta cinco programas.

Como una ampliación de la multiprogramación se puede considerar la modalidad de multitareas (multitasking), en la que se tiene el proceso paralelo de programas o partes de programas en una sola partición. Los programas o partes de programas reciben el nombre de tareas y se hace una diferencia entre los que se inician por el programa control de jobs y los que lo hacen con la macro instrucción ATTACH, que reciben el nombre de subtareas (subtasks). Las subtareas pueden estar relacionadas entre sí en forma lógica o pueden ser totalmente independientes.

El número total de tareas depende del número de particiones; a su vez la cantidad de subtareas no puede exceder de quince.

iii) Memoria virtual. Los programas para computador, en cualquier lenguaje en que ellos estén escritos, contienen instrucciones, descripciones de datos y operaciones de entrada/salida. En esos programas se tienen nombres de datos, de archivos de registros y de instrucciones. Se puede decir, entonces, que los programas fuente residen en un espacio creado por el programador. Ese espacio se denomina "espacio de nombres simbólicos" y normalmente se almacena en tarjetas que sirven de datos al compilador.

El compilador convierte los elementos simbólicos del espacio de nombres en instrucciones, datos, áreas y bloques de control. Después de este proceso llamado traducción, el espacio comprendido entre la dirección más baja y la dirección más alta correspondiente al programa recibe el nombre de "espacio de direcciones". Normalmente este espacio empieza con la dirección cero.

Se pueden combinar espacios de direcciones que han sido resultados de distintos procesos de traducción, operación que realiza, general-

mente, un programa de nombre Linkage Editor. El espacio de direcciones del programa se puede almacenar en tarjetas perforadas, en cinta magnética o en dispositivos de acceso directo, desde donde debe ser cargado en la memoria principal para ser ejecutado.

La operación de traducir direcciones del espacio de direcciones en ubicaciones específicas de la memoria real se define como "reubicación" y el tipo de reubicación dependerá del instante en que se efectúe la traducción. Si la traducción se realiza cuando se carga el programa para su ejecución, recibe el nombre de "reubicación estática" y si se realiza durante la ejecución se llama "reubicación dinámica". En el primer caso, el programa está limitado a la capacidad de la memoria real, pues para ser ejecutado debe estar cargado en su totalidad y si no puede ser cargado completo, porque excede la capacidad de memoria real, el programador debe recurrir a técnicas de traslapo (overlays) de secciones del programa. En la reubicación dinámica en cambio, el programador se despreocupa de los límites de la memoria real.

El programa Linkage Editor ofrece la posibilidad de modificar el punto de carga del programa, esto es, cambia la dirección cero, que era el límite inferior del espacio de direcciones del programa, lo cual a su vez permite la modalidad de multiprogramación. A pesar de esta característica, el tipo de reubicación continúa siendo estática pues se efectúa una sola vez, antes de la ejecución del programa.

Si se trata de reubicación dinámica, el espacio de direcciones del programa se divide en secciones (segmentos, páginas o ambos) que se cargan en memoria real, en el área que esté disponible, con sus direcciones relativas a la dirección cero del espacio de direcciones, y que sólo se traducen a medida que se está ejecutando la sección y de ahí el nombre de reubicación dinámica, la que puede lograrse con un dispositivo especial conocido como mecanismo de "traducción dinámica de direcciones". El programador, se decía anteriormente, que no se preocupa de los límites de la memoria real porque si en alguna de las secciones se hace referencia a una dirección que está en otra sección que no se encuentra en ese momento en la memoria real, la sección se busca en el dispositivo de acceso directo y se carga en la memoria real.

1) *Segmentación*

En cualquier sistema de segmentación cada segmento de un programa debe tener un identificador. De esta manera, la dirección de una instrucción, de un área de resultados o de un dato tendrá dos partes:

- la parte *ns* que representa el nombre del segmento, y
- la parte *d* que representa la dirección, dentro del segmento.

En el Sistema/370 la estructura de dirección tiene cuatro bytes, de los cuales los dos bytes de orden superior contienen el nombre o

miento y la memoria es organizada de acuerdo con esas modalidades.

i) Procesamiento batch. La memoria se subdivide en dos áreas. La de orden más bajo se destina al Supervisor y el área restante (background) a los programas que van a ser procesados. En memoria principal puede estar sólo un programa a ejecutar. Se proporciona un sistema de protección de memoria que impide que por efectos del proceso del programa pueda ser destruido el Supervisor.

ii) Procesamiento en multiprogramación. En esta modalidad el área que no es ocupada por el Supervisor puede subdividirse en un máximo de cinco particiones. El área de orden más bajo es el área de último término (background) o de menor prioridad de ejecución; las cuatro áreas restantes o áreas de primer término (foreground) se identifican con los nombres FOREGROUND–4 hasta FOREGROUND–1 y siguen en ese orden al área de último término con su respectiva prioridad de ejecución en el orden inverso.

Cada partición puede contener un programa separado, lo que significa que pueden ser procesados en forma paralela hasta cinco programas.

Como una ampliación de la multiprogramación se puede considerar la modalidad de multitareas (multitasking), en la que se tiene el proceso paralelo de programas o partes de programas en una sola partición. Los programas o partes de programas reciben el nombre de tareas y se hace una diferencia entre los que se inician por el programa control de jobs y los que lo hacen con la macro instrucción ATTACH, que reciben el nombre de subtareas (subtasks). Las subtareas pueden estar relacionadas entre sí en forma lógica o pueden ser totalmente independientes.

El número total de tareas depende del número de particiones; a su vez la cantidad de subtareas no puede exceder de quince.

iii) Memoria virtual. Los programas para computador, en cualquier lenguaje en que ellos estén escritos, contienen instrucciones, descripciones de datos y operaciones de entrada/salida. En esos programas se tienen nombres de datos, de archivos de registros y de instrucciones. Se puede decir, entonces, que los programas fuente residen en un espacio creado por el programador. Ese espacio se denomina "espacio de nombres simbólicos" y normalmente se almacena en tarjetas que sirven de datos al compilador.

El compilador convierte los elementos simbólicos del espacio de nombres en instrucciones, datos, áreas y bloques de control. Después de este proceso llamado traducción, el espacio comprendido entre la dirección más baja y la dirección más alta correspondiente al programa recibe el nombre de "espacio de direcciones". Normalmente este espacio empieza con la dirección cero.

Se pueden combinar espacios de direcciones que han sido resultados de distintos procesos de traducción, operación que realiza, general-

mente, un programa de nombre Linkage Editor. El espacio de direcciones del programa se puede almacenar en tarjetas perforadas, en cinta magnética o en dispositivos de acceso directo, desde donde debe ser cargado en la memoria principal para ser ejecutado.

La operación de traducir direcciones del espacio de direcciones en ubicaciones específicas de la memoria real se define como "reubicación" y el tipo de reubicación dependerá del instante en que se efectúe la traducción. Si la traducción se realiza cuando se carga el programa para su ejecución, recibe el nombre de "reubicación estática" y si se realiza durante la ejecución se llama "reubicación dinámica". En el primer caso, el programa está limitado a la capacidad de la memoria real, pues para ser ejecutado debe estar cargado en su totalidad y si no puede ser cargado completo, porque excede la capacidad de memoria real, el programador debe recurrir a técnicas de traslapo (overlays) de secciones del programa. En la reubicación dinámica en cambio, el programador se despreocupa de los límites de la memoria real.

El programa Linkage Editor ofrece la posibilidad de modificar el punto de carga del programa, esto es, cambia la dirección cero, que era el límite inferior del espacio de direcciones del programa, lo cual a su vez permite la modalidad de multiprogramación. A pesar de esta característica, el tipo de reubicación continúa siendo estática pues se efectúa una sola vez, antes de la ejecución del programa.

Si se trata de reubicación dinámica, el espacio de direcciones del programa se divide en secciones (segmentos, páginas o ambos) que se cargan en memoria real, en el área que esté disponible, con sus direcciones relativas a la dirección cero del espacio de direcciones, y que sólo se traducen a medida que se está ejecutando la sección y de ahí el nombre de reubicación dinámica, la que puede lograrse con un dispositivo especial conocido como mecanismo de "traducción dinámica de direcciones". El programador, se decía anteriormente, que no se preocupa de los límites de la memoria real porque si en alguna de las secciones se hace referencia a una dirección que está en otra sección que no se encuentra en ese momento en la memoria real, la sección se busca en el dispositivo de acceso directo y se carga en la memoria real.

1) *Segmentación*

En cualquier sistema de segmentación cada segmento de un programa debe tener un identificador. De esta manera, la dirección de una instrucción, de un área de resultados o de un dato tendrá dos partes:

- la parte *ns* que representa el nombre del segmento, y
- la parte *d* que representa la dirección, dentro del segmento.

En el Sistema/370 la estructura de dirección tiene cuatro bytes, de los cuales los dos bytes de orden superior contienen el nombre o

"número del segmento" y los restantes la dirección o "desplazamiento".

A medida que los segmentos de un espacio de direcciones se cargan en la memoria real, el sistema de operación construye una tabla de segmentos. Cada entrada en esta tabla contiene la identificación del segmento y el punto de origen de ese segmento en la memoria real. La tabla también se construye en la memoria real y a esa operación se la denomina mapping. Además de la tabla de segmentos se utiliza un registro de control que contiene el punto de origen de la tabla (Segment Table Origin Register-STOR).

Suponiendo que se ejecuta un programa que tiene cuatro segmentos y en el segmento 1 se hace referencia a una dirección relativa 8000, que está dentro del segmento, los pasos que se realizan son los siguientes:

- El registro STOR apunta a la dirección donde está la tabla de segmentos del programa.
- Se busca en la tabla la entrada que corresponde al segmento, en este ejemplo la primera entrada, y se obtiene ahí la dirección donde está cargado el segmento en memoria real.
- La dirección relativa 8000 se suma a la dirección de carga del segmento.

2) *Paginación*

La paginación es otro método que permite al programador construir sus programas despreocupándose de los límites de la memoria real. Consiste en subdivir el programa en partes de tamaño fijo llamadas "páginas". La memoria real también es subdividida en partes de igual tamaño que las páginas y estas partes reciben el nombre de "marcos de página".

El sistema de traducción de direcciones es similar al de la segmentación, esto es, se tiene una tabla de páginas cuyas entradas son el número de la página y el punto de carga de ella, además, se utiliza un registro que contiene la dirección de la tabla de páginas. Si bien es cierto que se pierde la posibilidad de subdividir el programa en secciones cuyo tamaño está en relación con la lógica de estructuración del mismo programa, se tiene en cambio la ventaja de poder cargar las páginas en cualquier marco, pues todos son de igual porte. Al mismo tiempo se pueden hacer subdivisiones más pequeñas.

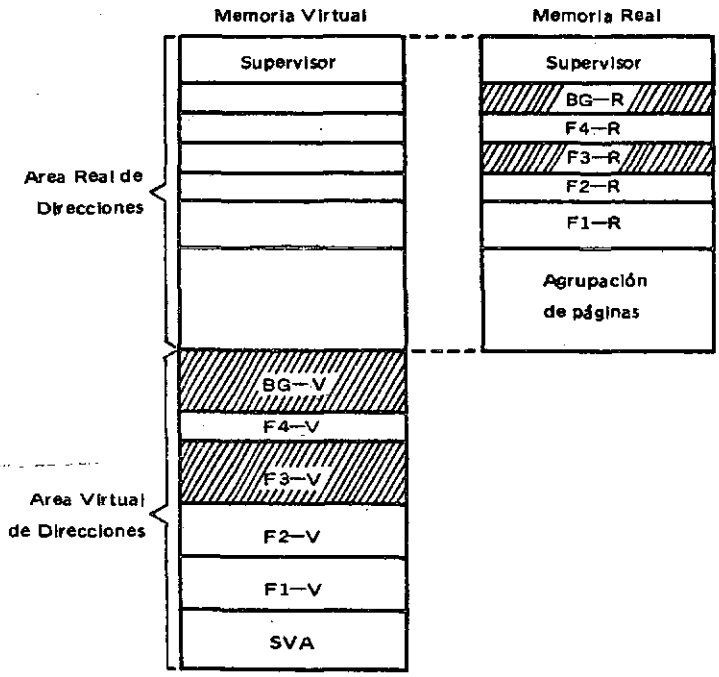
3) *Segmentación y paginación*

Combinando ambos métodos se logran las ventajas de los dos. La estructura de las direcciones relativas, dentro de cada página debe ser dada por:

- la parte ns que representa el nombre del segmento,

dientes y en ella se busca la página y la dirección en memoria real donde ha sido cargada. La dirección de carga de la página se suma con la dirección relativa para obtener la dirección absoluta.

Es posible ejecutar programas en modo real en cuyo caso no se efectúa paginación. Sin embargo, la partición real tiene siempre la correspondiente partición virtual, aun cuando no la usa. En la práctica hay programas que tienen que correr en modo real, como es el caso del Supervisor u otros programas cuya efectividad depende del tiempo.



Por ejemplo, en la figura anterior los programas que corren en las particiones background (BG-R) y foreground 3 (F3-R) lo harán en modo real, luego, las áreas virtuales correspondientes (BG-V y F3-V) no serán utilizadas.

iv) POWER. Una manera de disminuir la gran diferencia que existe entre la alta velocidad de la UCP y las velocidades de los dispositivos de entrada/salida que son relativamente bajos, es haciendo uso del programa POWER que efectúa una operación denominada SPOOL (Simultaneous Peripheral Operation on Line) que consiste en leer y grabar

flujos de entrada y salida en dispositivos de memoria auxiliar, en forma paralela con la ejecución de los trabajos.

POWER permite la multiprogramación, dado que puede atender en forma simultánea hasta cuatro particiones. Los pasos que se realizan con POWER son los siguientes:

- Lee los flujos de trabajos para cada partición y los almacena en una cola en disco.
- Transfiere los trabajos desde disco a las particiones y los ejecuta.
- Almacena en disco los resultados de los trabajos antes de que sean perforados, impresos o ambas cosas.

v) Bibliotecas. DOS/VS acepta cuatro tipos de bibliotecas: la de imágenes de memoria (core image), la de módulos reubicables (relocatable), la de proposiciones fuente (source statement) y la de procedimientos (procedure). Las tres primeras pueden ser del sistema y privadas, la última puede existir sólo como biblioteca del sistema. Estas están almacenadas en el archivo de residencia del sistema en discos, cuyo nombre simbólico es SYSRES y a ellas pueden tener acceso todas las particiones. Las bibliotecas privadas pueden estar almacenadas en paquetes de discos separados y a ellas pueden tener acceso sólo aquellos programas que están en particiones a las cuales han sido asignadas las bibliotecas.

El resultado de una traducción es un módulo objeto y éste puede ser obtenido en tarjetas perforadas a través de un dispositivo que tiene el nombre simbólico SYSPCH, puede ser catalogado en la biblioteca de módulos reubicables o puede ser almacenado en un área de discos de nombre simbólico SYSLNK.

El programa Linkage-editor que toma la información desde SYSLNK y opcionalmente desde la biblioteca de módulos reubicables realiza el proceso siguiente. El resultado es el programa objeto ejecutable formado por fases, cada una de las cuales tiene una dirección de carga en memoria real y se almacenan en la biblioteca de imágenes de memoria en forma permanente o temporal.

3. *Uso del sistema de operación*

A. *Cargador de programa inicial (Initial Program Loader*-IPL)

La operación del DOS/VS se inicia mediante el procedimiento de carga del programa inicial desde el paquete de discos donde reside el sistema (SYSRES). El operador monta el paquete de discos SYSRES en una unidad y monta además el paquete que contiene el conjunto de datos de página en la unidad asignada al nombre simbólico SYSVIS. Pone la dirección de la unidad donde está SYSRES utilizando las perillas de carga de unidad (load unit switches) presionando a continuación la tecla

LOAD en el panel de control del sistema. Esto causa que el primer registro en la pista cero del paquete de discos sea leido a memoria principal a los bytes 0-23. La información transferida consiste de una PSW de IPL y dos comandos de canal (CCW), los que a su vez producen la lectura y carga del IPL en las posiciones más altas de memoria.

Cuando se enciende la luz de WAIT en el panel de control del sistema, se presiona la tecla REQUEST en el dispositivo de comunicación operador-sistema (máquina 3210 ó 3215) a través del cual el sistema envía el mensaje

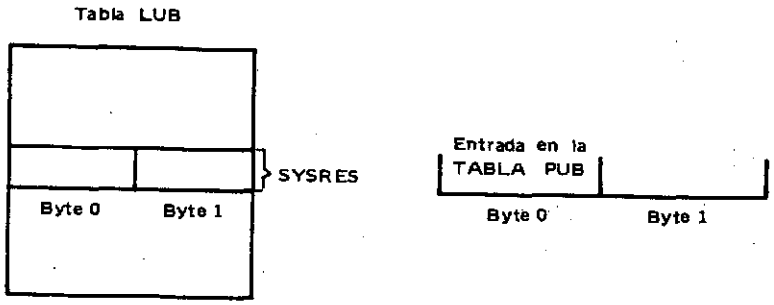
```
0103A SPECIFY SUPERVISOR NAME
```

el operador entra el nombre del supervisor y después presiona END. Si no se especifica nombre de supervisor y se presiona END asume el sistema el nombre $$A$SUP1.

IPL lee el núcleo del supervisor a las posiciones más bajas de memoria. Si se detecta un error mientras se lee el supervisor, se entra en estado de espera y se pone un código de error en la primera palabra de memoria. Si esto ocurre, se debe reiniciar IPL.

Si no ocurre nada anormal en la transferencia a memoria del núcleo del supervisor, IPL realiza las siguientes operaciones:

i) En la tabla "Bloque de unidades lógicas (Logical Unit Block-LUB)", en los dos bytes que corresponden al nombre simbólico SYSRES, coloca el punto de entrada de ese mismo nombre en otra tabla llamada "bloque de unidades físicas (Physical Unit Block-PUB)" en la que aparecen las direcciones reales o físicas de los distintos dispositivos que configuran el sistema. Esa dirección real está formada por el número del canal al cual está acoplada la unidad, y el número de esa unidad.



El objeto de esta primera operación de IPL es establecer la dirección real del sistema residente, ya que el paquete de discos puede ser cargado en distintas unidades. Obtiene la dirección desde las perillas de carga de unidad donde la estableció el operador

ii) Coloca la unidad de proceso en estado de espera con todos los códigos de interrupción enmascarados, excepto el de entrada/salida. Se enciende la luz de WAIT y debe presionarse nuevamente REQUEST.

El sistema responderá identificando el archivo SYSRES y la UCP, indicará si se requieren la fecha y la hora y si es necesario un operando relacionado con la ubicación geográfica del sistema, además entregará el mensaje.

```
0110A GIVE IPL COMMANDS
```

A continuación se entregan los comandos que sean necesarios, se presiona la tecla END y el sistema responderá

```
DOS/VS IPL COMPLETE
```

o el mensaje

```
DOS/VS IPL COMPLETE
1T00A  WARM START COPY OF SVA FOUND
```

después de lo cual el operador entregará los comandos que sean necesarios dependiendo de si se desea mantener o no la copia existente del Area Virtual Compartida (Shared Virtual Area - SVA) y de si se quiere o no hacer uso de las listas de directorio del sistema proporcionadas por IBM.

B. *Comandos de* IPL

La estructura de los comandos es: código de operación, escrito a partir de la primera posición y seguido al menos por un blanco, y a continuación operandos separados entre sí por coma.

i) Comando ADD. Permite agregar dispositivos a la tabla PUB. Estructura:

$$\text{ADD} \quad \text{X'cuu'}[(k)],\text{tipo de dispositivo}\left[\begin{matrix},\text{X'ss'} \\ ,\text{X'ssss'} \\ ,\text{X'ssssss'}\end{matrix}\right]$$

donde:

X' cuu': son los números de canal (c) y unidad (uu)

k: operando optativo, es igual a S si el dispositivo va a ser conmutable, esto es, que está asignado a dos canales adyacentes. El canal designado es el más bajo de los dos. Si el dispositivo no es conmutable, *k* varía de 0 a 255, siendo 0 la prioridad de atención más alta. Si no se especifica *k*, se supone prioridad 255.

tipo de dispositivo: código de dispositivo actual. Ejemplo: 2400T9, 3340R, etc.

X'ss' : operando optativo. Se utiliza para dar especificaciones de
X'ssss' dispositivo, por ejemplo, para indicar el modo de trabajo en
X'ssssss' cintas magnéticas de siete o de nueve canales, etc.

ii) Comando DEL. Se utiliza para borrar un dispositivo de la tabla PUB.

Estructura:

DEL X'cuu'

donde:

X'cuu' : son los números de canal (c) y unidad (uu).

iii) Comando CAT. DOS/VS proporciona un nuevo método de acceso de archivos, el "Método de acceso de almacenamiento virtual (Virtual Storage Access Method-VSAM)" el cual tiene archivos ordenados secuencialmente por llave, de tal modo que a ellos se puede tener acceso en forma secuencial (SAM) o directamente (DAM). Si se va a utilizar VSAM el comando CAT permite asignar el catálogo de VSAM al archivo de nombre simbólico SYSCAT. El comando CAT debe ser proporcionado después del comando SET y antes de DPD.

Estructura:

CAT UNIT = X'cuu'

donde:

X'cuu' : números de canal (c) y unidad (uu) del disco que será asignado a SYSCAT.

iv) Comando SET.

Estructura:

SET [DATE=|valor1,CLOCK=valor2][,ZONE={EAST|WEST}/hh/mm]

donde:

valor 1 : especifica día, mes y año con uno de los formatos siguientes:
mm/dd/aa
dd/mm/aa
en que:
mm = mes, dd = día, aa = año

valor 2 : especifica la hora con el formato hh/mm/ss
en que:

hh = hora, mm = minuto, ss = segundo

EAST: indica que el equipo está ubicado al este de Greenwich
WEST: indica que el equipo está ubicado al oeste de Greenwich
hh/mm: valor decimal que indica la diferencia en horas y minutos entre la hora local y la hora de Greenwich. El operando hh debe estar en el rango 0 – 12 y mm en el rango 0 – 59

v) Comando DPD. Permite definir el conjunto de datos de página. Se debe especificar siempre durante IPL y debe ser el último comando entregado.

Estructura:

$$\text{DPD}\ \left[\text{TYPE}=\left\{\begin{matrix}\text{N}\\ \text{F}\end{matrix}\right\}\right]\left[\begin{matrix},\text{UNIT=X'cuu'},\text{CYL=xxx}\\ ,\text{VOLID=xxxxxx}\end{matrix}\right]$$

donde:

TYPE = N indica que el conjunto de datos de página no necesita ser formateado y los límites del área no han sido cambiados. Cualquiera de las dos condiciones que no se cumpla, TYPE = N es ignorado. En este caso los operando UNIT y CYL deben ser especificados, si es que no lo fueron en el momento de generación del sistema
= F indica que el conjunto de datos de página va a ser formateado durante IPL. Esto es necesario si el conjunto va a ser ampliado o reubicado

UNIT = X'cuu' especifica el número de canal y de unidad del dispositivo en el cual va a estar el conjunto de datos de página. Si se especifica UNIT se debe indicar CYL

CYL = xxx especifica el número de secuencia del cilindro, relativo a cero, donde el conjunto de datos de página va a empezar, el sistema calcula el tamaño del área de discos que se va a utilizar a base de la fórmula:

$$\frac{\text{VSIZE}}{2} = \text{número de páginas (bloques de 2 K bytes)}$$

VSIZE es un parámetro con el que se define el tamaño del área virtual en el momento de generar el sistema.

VOLID = xxxxxx especifica el número de serie de volumen del paquete de discos que contiene al conjunto de datos de página.

Después de completar el procedimiento de IPL durante el cual se carga el programa de control de trabajos (JOB CONTROL), el sistema está listo para aceptar jobs para ser procesados en el área background. Si se desea procesar en el área foreground, el programa de control de trabajos se carga en la partición deseada en respuesta a los comandos

blanco en la columna 72 y en la tarjeta de continuación se empieza en la columna 16.

iii) Lista de las proposiciones

ASSGN	JOB	RSTRT	MTC
CLOSE	LBLTYP	TLBL	/&
DLAB	LISTIO	TPLAB	/+
DLBL	OPTION	UPSI	/*
DATE	OVEND	VOL	*
EXTENT	PAUSE	XTENT	
EXEC	RESET	ZONE	

iv) Descripción de algunas proposiciones

1) Proposición JOB. Identifica el comienzo de un trabajo.

Estructura:

```
//    JOB      nombre del trabajo
```

donde:

nombre del trabajo: Debe ser de uno a ocho caracteres alfanuméricos. Cualquier comentario del usuario puede aparecer en la proposición JOB después del nombre del trabajo. Si se tiene el medidor de tiempo (timer), aparece la hora en las posiciones 73 a 100 de SYSLST y en las posiciones 1 a 28 de SYSLOG. En ambos casos el formato utilizado es:

```
DATE    mm/dd/aa,CLOCK hh/mm/ss
```

2) Proposicion EXEC. Permite iniciar una "etapa de trabajo (job step)". Precediendo a EXEC puede haber cualquier proposición necesaria para preparar la ejecución del job step. La única limitación es la secuencia para las proposiciones de información de rótulos. Deben estar en el orden que se indica:

```
//    VOL        o        //    VOL
//    TPLAB                //    DLAB
                           //    XTENT
```

Estructura:

```
// EXEC  [[[PGM=]nombre de programa][,REAL][,SIZE=tamaño]]
         [PROC=nombre de procedimiento[,OV]              ]
```

donde:

nombre de programa: Es el programa, que está en la CIL, que va a ser ejecutado. Puede ser de uno a ocho caracteres alfanuméricos, el primero alfabético. Si el programa que va a ser ejecutado acaba de ser proce-

sado por el programa Linkage Editor, el operando de EXEC debe ser blanco.

REAL: Indica que el job step será ejecutado en modo real.
Si el operando se omite, el job step se ejecuta en modo virtual.

tamaño: Define de qué porte es necesaria una partición para el programa que se va a ejecutar. Si se ha especificado REAL, indica el porte de la sección de la parte real que será necesaria para procesar el job step. Si se omite SIZE, la partición real completa se destina al job step. Si no se ha especificado REAL, indica el porte de la sección de la parte virtual que estará disponible para el job step. Si se omite SIZE, la partición virtual completa se destina al job step.
El parámetro SIZE puede especificarse en una de las formas siguientes:

SIZE = nK
SIZE = AUTO
SIZE = (AUTO,nK)

donde:

n: debe ser múltiplo de dos y distinto de cero

AUTO: indica que el tamaño del programa será calculado por el sistema a base de la información contenida en el directorio de la CIL

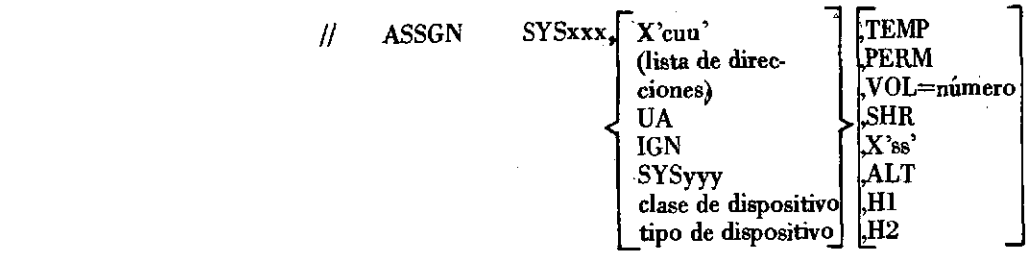
nombre de procedimiento: es el procedimiento, que está en la biblioteca de procedimientos, que va a ser ejecutado. Debe ser de uno a ocho caracteres alfanuméricos

OV: indica que las proposiciones que siguen a EXEC servirán para modificar el procedimiento catalogado. Las proposiciones que modifican deben tener el mismo identificador que las proposiciones a las que se refieren. El identificador debe aparecer en las columnas 73 a 79. En la columna 80 se puede tener el carácter siguiente:

A para insertar después de la proposición de referencia
B para insertar antes de la proposición de referencia
D para borrar la proposición de referencia
otro para reemplazar la p oposición de referencia.

3) Proposición ASSGN. Cuando se compilan los programas, usan nombres simbólicos para referirse a dispositivos de entrada/salida. En el momento de ejecución se hace uso de esta proposición para asignar la dirección de un dispositivo específico a los nombres simbólicos usados.

Estructura:



donde:

SYSxxx: es el nombre simbólico de la unidad (unidades lógicas)

X'cuu': son los números de canal (c) y unidad (uu)

c = 0 a 6 uu = 00 a FE

(lista de direcciones): Se pueden especificar hasta siete direcciones de dispositivo en la forma X'cuu' y separadas por coma. El sistema busca en la tabla PUB solamente aquellas unidades indicadas en la lista. Al encontrar una unidad libre la asigna a SYSxxx. En el caso de discos si se especifica SHR se asigna la primera unidad indicada en la lista, aun cuando esté ocupada

UA: indica que la unidad lógica va a ser desasignada. Cualquier operación que se refiera a este dispositivo causará la cancelación del job

IGN: indica que la unidad lógica va a ser desasignada y que todas las referencias del programa hacia el dispositivo lógico van a ser ignoradas. La opción no es válida para SYSRDR, SYSIPT,SYSIN y SYSCLB ni para programa en PL/I

SYSyyy: puede ser cualquier unidad lógica. Si se especifica, SYSxxx se asigna al mismo dispositivo al que está SYSyyy

Clase de dispositivo: se especifica una de las siguientes palabras claves: READER, PRINTER, PUNCH, TAPE, DISK o DISKETTE. El sistema busca en la tabla PUB el primer dispositivo desasignado de la clase indicada y lo asigna a SYSxxx

Tipo de dispositivo: se especifica un código de dispositivo, tal como 2400T9, 3420T7, etc. El sistema busca en la tabla PUB y asigna a SYSxxx el primer dispositivo libre del tipo especificado. En el caso de discos, si se especifica SHR se asigna el primer dispositivo del tipo indicado, aun cuando esté ocupado

TEMP: se indica si la asignación va ser temporal (TEMP) o

PERM: permanente (PERM)

número: se especifica el número de serie de volumen, del dispositivo requerido. Este parámetro se puede especificar sólo si se utilizan cintas o discos. El sistema efectúa un chequeo del rótulo del volumen montado para verificar si es el que se ha pedido con el mensaje

1T50A MOUNT nnnnnn ON X'cuu'

SHR: se especifica sólo para dispositivos de discos y se utiliza en combinación con: lista de direcciones, clase de dispositivo y tipo de dispositivo. Se asigna a SYSxxx la primera unidad del tipo, clase o dirección aun cuando esté ocupada. Si el parámetro no se indica, el sistema asigna la unidad a un dispositivo de discos privado

X'ss': número hexadecimal que permite indicar características de cinta magnética. Si no se especifica, el sistema asume las características definidas en el momento de generación del mismo

ALT: permite especificar una unidad de cinta magnética alternativa de aquélla asignada a SYSxxx. La cinta alternativa se utiliza cuando la capacidad de la cinta original se ha completado. El parámetro no es válido para SYSRDR, SYSIPT, SYSIN, SYSLNK, SYSCLB y SYSLOG

H1: se indica que en las máquinas multifuncionales 2560 ó 5425, se utilizará

H2: para entrada, el depósito uno (H1) o el depósito dos (H2). Si no se especifica ninguno de los dos parámetros, el sistema supone H1. Si el programa POWER sustenta la partición, no se puede especificar H2.

4) Proposición OPTION. Permite especificar una o más opciones de control de trabajos. Las opciones tienen vigencia sólo durante el job.

Estructura:

// OPTION opción1[,opción2...]

Las opciones pueden ser:

LOG: para listar todas las proposiciones y comandos de control en SYSLST

NOLOG: no se imprimen las proposiciones y comandos de control válidos, sólo se imprimen los no válidos

DUMP: en el caso de un término anormal de programa, se vacían en SYSLST los contenidos de los registros y de las particiones virtual y real provisoria, si fueron asignadas

NODUMP: suprime el efecto de la opción DUMP

LINK: si se especifica, el resultado de procesos de traducción se graba en SYSLNK, de donde lo tomará posteriormente el

programa Linkage Editor. Esta opción siempre debe preceder a la proposición EXEC LNKEDT en el flujo de entrada. La opción se cancela después de EXEC sin operando

NOLINK: suprime el efecto de la opción LINK

DECK: el resultado de procesos de traducción se obtiene en SYSPCH. Si se ha especificado LINK, la opción es aceptada por PL/I, FORTRAN IV, COBOL ANS y DOS/VS y el lenguaje de ensamble

NODECK; suprime el efecto de la opción DECK

EDECK; permite que el lenguaje de ensamble perfore todas las definiciones de macro fuentes en formato de edición en SYSPCH. El resultado se puede catalogar en la sub-biblioteca E de la biblioteca de proposiciones fuente

NOEDECK: suprime el efecto de la opción EDECK

ALIGN: permite que el lenguaje de ensamble deje constantes y áreas con el alineamiento adecuado y verifique, además, el alineamiento de direcciones utilizadas en instrucciones de máquina

NOALIGN: suprime el efecto de la opción ALIGN

LIST: si se especifica, los traductores de lenguajes entregan los módulos fuente en SYSLST. El lenguaje de ensamble entrega, además, el listado del módulo objeto en hexadecimal y junto con FORTRAN proporciona un resumen de todos los errores encontrados en el programa fuente

NOLIST: suprime el efecto de la opción LIST. Anula también la impresión del diccionario de símbolos externos, del diccionario de lista de reubicación y de la lista de referencias cruzadas (XREF)

LISTX: si se especifica, los traductores COBOL ANS y DOS/VS entregan el mapa de la PROCEDURE DIVISION en SYSLST. Los traductores PL/I y FORTRAN entregan los módulos objeto en SYSLST

NOLISTX: suprime el efecto de la opción LISTX

SYM: si se especifica, los traductores COBOL ANS y DOS/VS y COBOL entregan el mapa de la DATA DIVISION en SYSLST. El traductor PL/I entrega la tabla de símbolos en SYSLST

NOSYM: suprime el efecto de la opción SYM

XREF: si se especifica, el lenguaje de ensamble entrega la lista de referencias cruzadas simbólicas en SYSLST

NOXREF: suprime el efecto de la opción XREF

ERRS: si se especifica, los traductores FORTRAN, COBOL ANS y DOS/VS y PL/I entregan un resumen de los errores contenidos en el programa fuente, en SYSLST

NOERRS: suprime el efecto de la opción ERRS

ACANCEL: indica que el job debe ser cancelado si se intenta asignar un

dispositivo en forma infructuosa. Esto puede ser debido a dispositivo no definido, a estado no válido de dispositivo, etc.

NOACANCEL: suprime el efecto de la opción ACANCEL

CATAL: indica que una fase o programa va a ser catalogado en la CIL al término de un proceso de Linkage Editor, CATAL implica la opción LINK. La opción se cancela después de EXEC sin operando

STDLABEL: indica que todos los rótulos de cinta o de dispositivos de almacenamiento de acceso directo (DASD), proporcionados después de este punto, se graban al comienzo de la pista de rótulos estándar. Se vuelve a la opción USRLABEL (rótulos del usuario) al término del job o job step.

USRLABEL: indica que todos los rótulos de cinta o DASD, proporcionados después de este punto, se graban al comienzo de la pista de rótulos del usuario

PARSTD: indica que todos los rótulos de cinta o DASD, proporcionados después de este punto, se graban al comienzo de la pista de rótulos estándar de la partición. Se vuelve a la opción USRLABEL al término del job o job step

48C: se utiliza con PL/I y especifica que se tiene el conjunto de 48 caracteres en SYSIPT

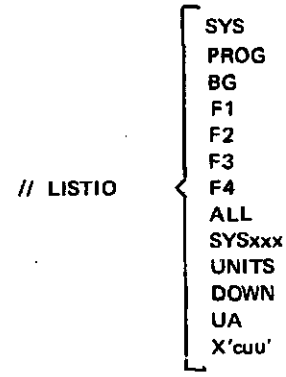
60C: se utiliza con PL/I y especifica que se tiene el conjunto de 60 caracteres en SYSIPT

SYSPARM= 'cadena de caracteres'
especifica un valor para el símbolo variable del lenguaje de ensamble, $SYSPARM. La cadena puede contener hasta ocho caracteres EBCDIC encerrados entre comillas

SUBLIB=DF indica que el lenguaje de ensamble DOS/VS y el programa ESERV recuperan macros no editadas y copian "libros" desde la sub-biblioteca D de la biblioteca de proposiciones fuente (SSL) y recuperan macros editadas desde la sub-biblioteca F en vez de las sub bibliotecas A y E respectivamente. La opción permanece vigente hasta el término del job o hasta encontrar la opción SUBLIB=AE

SUBLIB=AE indica que el lenguaje de ensamble y el programa ESERV recuperan las macros no editadas y copian "libros" desde la sub-biblioteca A y recuperan macros editadas desde la sub-biblioteca E.

5) Proposición LISTIO. Permite obtener un listado de todas las asignaciones hechas de entrada/salida. La proposición se ignora si no se ha asignado SYSLST.



donde:

SYS: se listan las unidades físicas asignadas a todas las unidades lógicas del sistema en background

PROG: se listan las unidades físicas asignadas a todas las unidades lógicas del programador en background

BG: se listan las unidades físicas asignadas a todas las unidades lógicas en background

F1–F4: se listan las unidades físicas asignadas a todas las unidades lógicas de la respectiva partición foreground

ALL: se listan las unidades físicas asignadas a todas las unidades lógicas

SYSxxx: se listan las unidades físicas asignadas a la unidad lógica especificada. No es válido el parámetro para SYSOUT y SYSIN

UNITS: se listan las unidades lógicas asignadas a todas las unidades físicas

DOWN: se listan todas las unidades físicas indicadas como no operativas

UA: se listan todas las unidades físicas que no están asignadas a una unidad lógica

X'cuu': se listan las unidades lógicas asignadas a la unidad física especificada.

6) Proposición PAUSE. Permite producir una interrupción temporal (pausa) del proceso. La proposición aparece en SYSLOG. El proceso continúa al presionar la tecla END o la ENTER en SYSLOG.

Estructura:

// PAUSE [cualquier comentario del programador]

7) Proposición RESET. Permite volver las asignaciones hechas en la partición donde se entrega RESET, a las estándares, que son aquéllas que se especifican en el momento de generar el sistema, más cualquier modificación hecha por el operador mediante un comando ASSGN sin la opción TEMP.

Estructura:

```
// RESET  [SYS   ]
          [PROG  ]
          [ALL   ]
          [SYSxxx]
```

donde:

SYS: vuelve todas las unidades lógicas del sistema a sus asignaciones estándares

PROG: vuelve todas las unidades lógicas del programador a sus asignaciones estándares

ALL: vuelve todas las unidades lógicas a sus asignaciones estándares

SYSxxx: vuelve la unidad lógica indicada a su asignación estándar. No se puede especificar SYSIN o SYSOUT.

8) Proposición CLOSE. Se utiliza para cerrar una unidad lógica asignada a cinta magnética. La operación consiste en grabar una "marca de cinta" (tape mark) un registro de salida EOV (end of volume), dos marcas de cinta, rebobinar y descargar la cinta.

Estructura:

```
// CLOSE  SYSxxx[,X'cuu',X'ss']
                [,UA         ]
                [,IGN        ]
                [,ALT        ]
```

donde:

SYSxxx: puede ser: SYSPCH, SYSLST, SYSOUT o SYS000 a SYSnnn

X'cuu': indica que la unidad lógica, después de haber sido cerrada se asignará al canal (0–6) y unidad (0–254, en hexadecimal 00–FE) especificados

X'ss': ver proposición ASSGN

UA: indica que la unidad lógica va a ser cerrada y desasignada

IGN: indica que la unidad lógica va a ser cerrada y quedar ignorada. El parámetro no es válido para SYSRDR, SYSIPT o SYSIN

ALT indica que la unidad lógica va a ser cerrada y se va a abrir como unidad alternativa. El parámetro es válido sólo para SYSPCH, SYSLST o SYSOUT asignadas a cinta magnética.

9) Proposición DATE. Permite colocar una fecha en el área de comunicaciones del Supervisor. La fecha se aplica sólo al job que se está ejecutando. Tiene dos formatos posibles.

Estructura:

```
// DATE   mm/dd/aa
// DATE   dd/mm/aa
```

5) + desplazamiento
6) F+dirección

} direcciones absolutas

en que:

símbolo: puede ser el nombre de una fase definida previamente, el de una sección de control o un rótulo externo que es el operando de una proposición ENTRY en un programa fuente

(fase): si el parámetro anterior (símbolo) es un nombre de sección de control o de rótulo externo que aparece en más de una fase, el nombre de la fase específica que debe haber sido definida anteriormente se indica entre paréntesis

reubicación: indica que el origen estará desplazado con respecto al símbolo, hacia atrás o hacia adelante, en el valor de la constante hexadecimal (uno a seis dígitos X'hhhhhh') o de la decimal (uno a ocho dígitos ddddddd) o de nK

*: se asigna como origen de la fase la siguiente ubicación de memoria en la partición virtual, alineada en una dirección múltiplo de doble palabra.
Si se trata de la primera proposición PHASE en el área background, * indica que el origen estará en la primera doble palabra después del área de salvación de rótulos, si los hay, y del área común asignada, si la hay.

S: si se especifica, el origen se determina en la misma forma que con el subparámetro * en una primera proposición PHASE

ROOT: indica que la fase será fase raíz, esto es, ninguna fase posterior podrá traslaparla, de lo contrario se especificará un diagnóstico de desastre. El origen se determina en la misma forma que con el subparámetro S. Sólo la primera fase puede ser raíz

desplazamiento: se especifica el punto de origen como una dirección absoluta, relativa a cero. El desplazamiento se da como una constante hexadecimal, como una constante decimal o como nK en igual forma que el subparámetro reubicación

F + dirección: permite que el origen del programa que se está procesando por linkage editor en una partición sea ubicado al comienzo de otra partición que no está asignada. La dirección se puede especificar como una constante hexadecimal (cuatro a seis dígitos), como una constante decimal (cinco a ocho dígitos), o como nK (*n*, de dos a cuatro dígitos)

NOAUTO: indica que la búsqueda automática en las bibliotecas

reubicables se suprime. La búsqueda se efectúa para resolver las referencias externas sin solucionar

SVA: indica que la fase será procesada en el área SVA (shared virtual area)

PBDY: se especifica que la fase será procesada por linkage editor en alineamiento de página.

ii) Proposición INCLUDE. Permite incluir módulos objeto reubicables para que sean procesados por linkage editor. Tiene dos parámetros como operandos, que pueden ser omitidos. Si se omiten, le indican al programa job control que copie en el área SYSLNK lo que hay en SYSIPT. La operación termina al detectarse fin de datos. Si está solamente el primer operando, se incluye un módulo desde la biblioteca reubicable. Si están los dos operandos, el módulo se lee desde la biblioteca reubicable y de él se extraen las secciones de control especificadas en el segundo operando. Si se indica solamente el segundo operando, debe estar precedido por una coma. En este caso se extraen las secciones de control del módulo que figura a continuación de la proposición INCLUDE en el área SYSLNK, previamente el programa job control debe haber copiado todo desde SYSIPT a causa de un INCLUDE sin operandos.

Estructura:

INCLUDE [nombre de módulo][,(lista de nombres)]

donde:

nombre de módulo: es el nombre con el cual se recuperará el módulo desde la biblioteca reubicable

(lista de nombres): nombres de las secciones de control que serán extraídas del módulo reubicable. La cantidad total de secciones de control especificadas en la lista no puede exceder de cinco, sin embargo, puede indicarse cualquier número de proposiciones INCLUDE. Una fase puede contener un módulo completo y además algunas secciones de control del mismo módulo.

iii) Proposición ENTRY. Todo conjunto de datos de entrada para el programa linkage editor debe terminar en la proposición ENTRY. Si el programador no la incluye, se utiliza la que genera el programa job control cuando es leida la proposición EXEC LNKEDT.

Estructura:

ENTRY [punto de entrada]

donde:

punto de entrada: nombre simbólico del punto al cual se le entregará el control (punto de entrada) en el momento de la ejecución. Debe ser el nombre de una sección de

control o de una definición de rótulo que aparezcan en la primera fase. Si el parámetro no se especifica, el linkage editor utiliza como punto de entrada la dirección proporcionada en la tarjeta END. Si END no tiene la dirección buscada, el punto de entrada será la dirección de carga de la primera fase.

iv) Proposición ACTION. Se utiliza para indicar distintas opciones a linkage editor. Debe ser la primera proposición en el conjunto de datos de entrada para el linkage editor. Si es necesario, se pueden especificar varias proposiciones ACTION seguidas.

Estructura:

ACTION $\left\{\begin{matrix}\underline{\text{REL}}\\ \underline{\text{NOREL}}\end{matrix}\right\}$ [,CLEAR] $\left[\begin{matrix},\text{MAP}\\ ,\text{NOMAP}\end{matrix}\right]$ [,NOAUTO][,CANCEL] $\left[\begin{matrix},\underline{\text{BG}}\\ ,\underline{\text{FN}}\end{matrix}\right]$

donde:

REL: indica que la fase que se va a producir será reubicable, lo cual depende del origen especificado en la proposición PHASE. Si en el sistema se tiene el programa que permite cargar en forma reubicable (relocating loader), se supone la opción REL

NOREL: indica que la fase que se va a producir no será reubicable. Si en el sistema no se tiene el relocating loader, se supone la opción NOREL

CLEAR: indica que el área no utilizada de la CIL se llenará con ceros binarios antes del proceso del linkage editor. La opción produce gran consumo de tiempo, por lo cual, en lo posible, debe evitarse su uso

MAP: permite que los mensajes de diagnóstico y un mapa de memoria virtual se obtengan a través de SYSLST. El mapa contiene los nombres de las secciones de control que componen las fases y los nombres de los puntos de entrada en cada sección de control

NOMAP: suprime el efecto de la opción MAP. Los mensajes de error se obtienen a través de SYSLOG

NOAUTO: suprime la característica de AUTOLINK para el programa completo

CANCEL: se cancela automáticamente el job si se produce algún error del tipo 2100I al 2170I. Si el parámetro no se especifica y ocurre alguno de los tipos de errores indicados, el job continúa. Algunos errores del tipo 2100I al 2170I son, por ejemplo, operación no válida en la proposición, la proposición se extiende más allá de la columna 71, nombre de fase duplicado, la biblioteca reubicable no está presente, etc.

BG: se coloca la dirección de término del supervisor al comienzo de la

F1 – F4 partición virtual especificada más los bytes ocupados por el área de salvación y los ocupados por el área de rótulos. Los programas que tienen un origen de fase igual a S o a* se cargan en la partición virtual indicada.

E. *Librarian*

Se denomina así a un conjunto de programas que atienden las bibliotecas del sistema realizando funciones de mantención, servicio y copia.

i) El programa MAINT, que es llamado mediante la proposición

```
// EXEC MAINT
```

ejecuta las funciones de mantención.

Esas funciones son:

- catalogar (catalog)
- borrar (delete)
- renombrar (rename)
- condensar (condense)
- reubicar (reallocate)
- actualizar (update)

Algunas proposiciones de uso frecuente:

1) Proposición CATALR. Permite catalogar un módulo en la biblioteca de módulos reubicables.

Estructura:

CATALR nombre del módulo[,v.m]

donde:

nombre del módulo: es el identificador del módulo y tiene de uno a ocho caracteres y no debe tener asterisco como primer carácter

v.m: indica número de versión (v = 0-127) y número de modificación (m = 0 – 255). Si el parámetro no se especifica, el sistema supone 0.0.

2) Proposición CATALS. Permite catalogar un libro en una sub-biblioteca de la biblioteca de proposiciones fuente.

Estructura:

CATALS sub-biblioteca.nombre del libro[,v.m[,C]]

donde:

sub-biblioteca: puede ser cualquier carácter alfanumérico que represente una sub-biblioteca

nombre del libro: es el identificador del libro y tiene de uno a ocho caracteres alfanuméricos, el primero de los cuales debe ser alfabético

v.m:	indica número de versión y de modificación
C:	indica que los números especificados en v.m deben ser verificados.

Si se van a catalogar definiciones de macros en la sub-biblioteca de lenguaje de ensamble, deben estar precedidas por la proposición MACRO y seguidas por la proposición MEND. Si se trata de otro tipo de libros, deben estar precedidos y seguidos por la proposición BKEND, cuya estructura es la siguiente:

BKEND [sub-biblioteca,nombre del libro],[verifica secuencia],[cantidad], [CMPRSD]

donde:

sub-biblioteca, nombre del libro:	es el mismo parámetro de la proposición CATALS
verifica secuencia:	puede ser SEQNFS para verificar la secuencia en las columnas 73 a 78 o SEQNCE, para verificar en las columnas 77 a 80
cantidad:	indica el número de tarjetas imagen en el libro, incluidas las dos proposiciones BKEND
CMPRSD:	indica que el libro está en formato comprimido, esto es, que se han eliminado todos los caracteres blanco.

Todos los parámetros son opcionales y deben estar en el orden indicado. Se utilizan sólo en la proposición que encabeza el libro.

3) Proposición DELETR. Permite borrar módulos de la biblioteca reubicable.

Estructura: Tiene tres formatos posibles:

DELETR nombre de módulo[,nombre de módulo,...]
DELETR prog1.ALL[,prog2.ALL,...]
DELETR ALL

La primera permite borrar uno o más módulos, la segunda uno o más programas completos y la tercera toda la biblioteca.

4) Proposición DELETS. Permite borrar libros de la biblioteca de proposiciones fuente.

Estructura: Tiene tres formatos posibles:

DELETS sub-biblioteca.libro1[,sub-biblioteca.libro2,...]
DELETS sub-biblioteca. ALL
DELETS ALL

La primera permite borrar uno o más libros, la segunda una sub-biblioteca completa y la tercera toda la biblioteca.

5) Proposiciones RENAMC y RENAMR. Permiten cambiarle el nombre a una fase en la CIL y a un módulo en la RL respectivamente.

Estructura:

RENAMC nombre antiguo, nombre nuevo[,nombre antiguo,nombre nuevo,...]

RENAMR nombre antiguo, nombre nuevo[,nombre antiguo,nombre nuevo,...]

6) Proposición RENAMS. Permite cambiarle el nombre a un libro en la SSL.

Estructura:

RENAMS sub-biblioteca.nombre antiguo,sub-biblioteca.nombre nuevo[,sub-biblioteca.nombre antiguo,sub-biblioteca.nombre nuevo,...]

7) Proposiciones CONDS. Permite condensar, esto es, eliminar los espacios desocupados producto de la función borrar (delete), de una o más bibliotecas.

Estructura:

CONDS biblioteca[,biblioteca,...]

ii) Seis programas realizan la función de servicio. Ellos son:

DSERV: para desplegar los directorios de cada una de las bibliotecas

CSERV: para desplegar o perforar fases desde la CIL o para ambas cosas a la vez

RSERV: para desplegar o perforar módulos desde la RL o para ambas cosas a la vez

SSERV: para desplegar o perforar libros desde la SSL o para ambas cosas a la vez

ESERV: para desplegar o perforar, verificar, etc. macros en lenguaje de ensamble editadas desde la SSL, o para ambas cosas a la vez

PSERV: para desplegar o perforar procedimientos desde la biblioteca de procedimientos (PL), o para ambas cosas al mismo tiempo

todos los programas son llamados con // EXEC

Algunas proposiciones de uso frecuente:

1) Proposición DSPLY. Permite desplegar directorios o contenidos de bibliotecas.

Ejemplo. Despliegue de contenido de la CIL.

Se utiliza la proposición DSPLY con uno de los formatos siguientes:

```
DSPLY    fase1[,fase2,...]
DSPLY    prog1.ALL[,prog2.ALL,...]
DSPLY    ALL
```

La primera permite desplegar una o más fases, la segunda uno o más programas completos y la tercera toda la biblioteca.

2) Proposición PUNCH. Permite perforar contenidos de bibliotecas.

Ejemplo. Perforación de contenidos de la RL.

Se utiliza la proposición PUNCH con uno de los formatos siguientes:

```
PUNCH    módulo1[,módulo2,...]
PUNCH    prog1.ALL[,prog2.ALL,...]
PUNCH    ALL
```

La primera permite perforar uno o más módulos, la segunda uno o más programas completos y la tercera toda la biblioteca.

3) Proposición DSPCH. Permite desplegar y perforar contenidos de bibliotecas.

Ejemplo. Despliegue y perforación de contenidos de la SSL.

Se utiliza la proposición DSPCH con uno de los formatos siguientes:

```
DSPCH    sub-biblioteca.libro1[,sub-biblioteca,libro2,...][,CMPRSD]
DSPCH    sub-biblioteca.ALL[,sub-biblioteca.ALL,...][,CMPRSD]
DSPCH    ALL[,CMPRSD]
```

La primera despliega y perfora uno o más libros, la segunda una o más sub-bibliotecas completas y la tercera toda la biblioteca. En los tres casos se puede perforar en formato comprimido utilizando el parámetro CMPRSD.

iii) El programa CORGZ ejecuta la función de copia. Realiza las siguientes operaciones:

- Definir o crear un paquete de sistema nuevo
- Definir o crear bibliotecas privadas
- Transferir fases, módulos o libros entre bibliotecas asignadas.

El programa se llama mediante la proposición // EXEC. Junto con ella se deben proporcionar otras como // DLBL y // EXTENT de job control, que no han sido tratadas en el texto.

F. *Problemas resueltos*

Se utilizan las siguientes asignaciones: X'001' lectora de tarjetas (SYSRDR y SYSIPT), X'00A' lecto-perforadora de tarjetas (SYSPCH), X'180' y X'181' cintas magnéticas.

1) Obtener el deck de la compilación de un programa en FORTRAN y de otro en ASSEMBLER. Catalogar en CIL el módulo de assembler más un módulo MOD1 de la RL. Ejecutar el módulo de fortran más un módulo MOD2 de la RL. Ejecutar el programa catalogado en CIL.

```
      SYSRDR,SYSIPT
//    JOB EJ1
//    OPTIØN DECK
//    EXEC FFØRTRAN
      } programa en fortran
/*
//    EXEC ASSEMBLY
      } programa en assembler
/*
*     PØNER DECK DE ASS.
//    PAUSE EN X'00A'
//    ASSGN SYSIPT,X'00A'
//    ØPTIØN CATAL
      PHASE PRØG1,S
      INCLUDE
      INCLUDE MØD1
//    EXEC LNKEDT
*     PØNER DECK DE FØRT.
//    PAUSE EN X'00A'
//    ØPTIØN LINK
      INCLUDE
      INCLUDE MØD2
//    EXEC LNKEDT
//    RESET SYS
//    EXEC
      } datos
/*
//    EXEC PRØG1
      } datos
/*
/&
```

2) En el programa que figura a continuación indicar: ¿Qué módulos quedan en la X'180'? ¿Qué módulos quedan en la X'181'? ¿Cuál es el conjunto de datos de entrada para el segundo // EXEC LNKEDT? ¿Cuál es la estructura del programa FASE1? ¿Qué lenguajes componen el programa FASE2?

3) Analizar el siguiente programa e indicar qué resultados entrega:
SYSRDR,SYSIPT

```
// JØB EJ3
// ØPTIØN LINK,LIST
// ASSGN SYSIPT,X'00A'
   PHASE FASE1,S
   INCLUDE MØD1,(CSA,CSC)
   PHASE FASE2,FASE1
   INCLUDE
// EXEC ASSEMBLY
   INCLUDE
   PHASE FASE3,FASE(FASE2)
   INCLUDE,(CSE,CSG,CSH)
   INCLUDE
// EXEC LNKEDT
// RESET ALL
// EXEC
   } datos
/*
/&
```

En la X'00A' (lecto-perforadora) se tiene:

```
   INCLUDE ,(CSR,CST,CSW)
   ESD
   TXT CSR
   TXT CSS
   TXT CST
   TXT CSU
   TXT CSW
   RLD
   END
   PHASE FASE4,*
/*
   } programa en assembler
     CSK,CSL
/*
   INCLUDE MØD2
/*
   ESD
   TXT CSE
   TXT CSF
   TXT CSG
   TXT CSH
   TXT CSI
   RLD
   END
/*
```

En la biblioteca de módulos reubicables se tiene:

MOD1	MOD2	MOD3
ESD	INCLUDE MØD3	ESD
TXT CSA		TXT CSX
TXT CSB		TXT CSY
TXT CSC		TXT CSZ
TXT CSD		RLD
RLD		END
END		

Al ser ejecutado el programa, se asigna a SYSIPT la unidad física X'00A' después de abrir el área SYSLNK con la opción LINK.

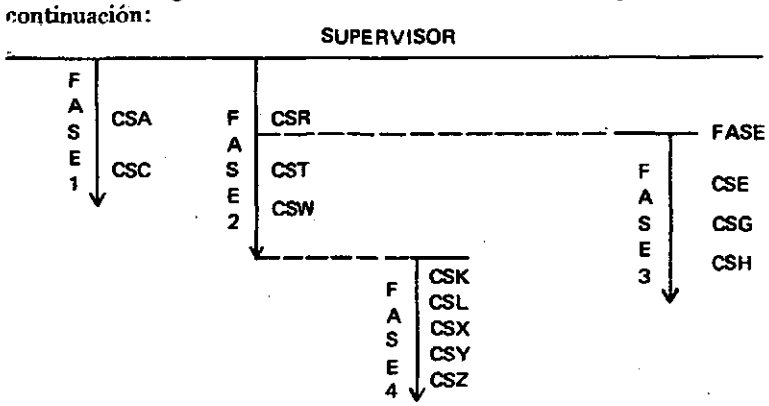
Se tiene como resultado en SYSLNK:

```
PHASE FASE1,S                    ┐
INCLUDE MØD1,(CSA,CSC)           ├─ copiadas por el job control
PHASE FASE2,FASE1                ┘  desde SYSRDR
INCLUDE ,(CSR,CST,CSW)           ┐
ESD                              │
TXT CSR                          │
TXT CSS                          │
TXT CST                          ├─ copiadas por job control
TXT CSU                          │  desde SYSIPT
TXT CSW                          │
RLD                              │
END                              │
PHASE FASE4,*                    ┘
ESD                              ┐
TXT CSK                          │
TXT CSL                          ├─ resultado del proceso
RLD                              │  de ensamble
END                              ┘
INCLUDE MØD2                     ─ copiado desde SYSIPT
PHASE FASE3,FASE(FASE2)          ┐  copiado por job control
INCLUDE ,(CSE,CSG,CSH)           ┘  desde SYSRDR
ESD                              ┐
TXT CSE                          │
TXT CSF                          │
TXT CSG                          ├─ copiado por job control
TXT CSH                          │  desde SYSIPT
TXT CSI                          │
RLD                              │
END                              ┘
ENTRY                            ┐
                                 ├─ generado por job control al
                                 ┘  aparecer // EXEC LNKEDT
```

El programa linkage editor genera las fases en forma temporal en la CIL. La estructura de las fases es la siguiente:

FASE1	FASE2	FASE4	FASE3
CSA	CSR	CSK	CSE
CSC	CST	CSL	CSG
	CSW	CSX	CSH
		CSY	
		CSZ	

Al ser cargadas en memoria formarán la estructura que se indica a continuación:



Se ha supuesto la existencia de un rótulo FASE dentro de FASE2 y que FASE4 se ejecuta a continuación de FASE2.

4) Analizar el siguiente programa e indicar qué resultado entrega:

```
SYSRDR,SYSIPT
// JØB EJ4
// ASSGN SYSIPT,X'00A'
// ØPTIØN LINK,LIST,SYM,NØDUMP
   ACTIØN CANCEL
   PHASE FASE1,RØØT
   INCLUDE
// RESET ALL
// EXEC FFØRTRAN
   } programa en fortran
        CSC,CSD
/*
   INCLUDE
   PHASE FASE2,*
   INCLUDE MØD1,(CSF,CSH)
/*
// ASSGN SYSIPT,X'00A'
// EXEC ASSEMBLY
// ASSGN SYS008,X'180'
```

```
// ASSGN SYS008,X'181',ALT
* MØNTAR CARRETES EN
// PAUSE X'180' Y X'181'
// RESET SYS
   INCLUDE
   PHASE FASE3,*+1024
   INCLUDE MØD3
/*
// EXEC LNKEDT
// EXEC
   } datos
/*
/&
```

En la X'00A' se tiene:

```
   ESD
   TXT CSA
   TXT CSB
   RLD
   END
/*
   } programa en assembler
      CSJ,CSK
/*
```

En la biblioteca de módulos reubicables se tiene:

MOD1	MOD2	MOD3
INCLUDE MØD2	ESD	ESD
ESD	TXT CSE	TXT CSL
TXT CSH	TXT CSF	RLD
TXT CSI	TXT CSG	END
TXT CSJ	RLD	
RLD	END	
END		

Con la ejecución se tiene primero en SYSLNK:

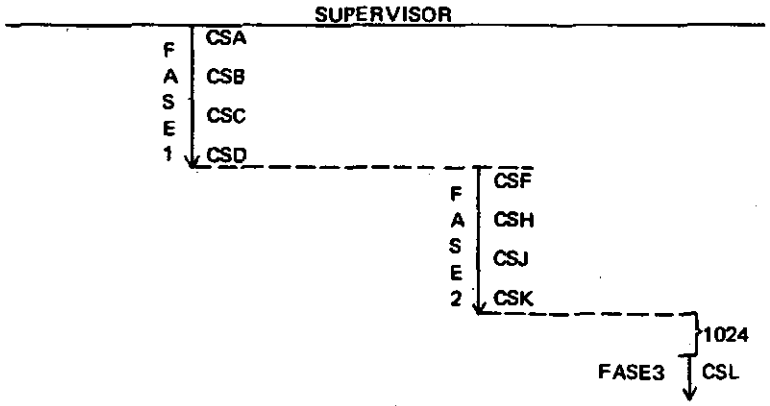
```
ACTIØN CANCEL
PHASE FASE1,RØØT
ESD
TXT CSA
TXT CSB
RLD
END
ESD
TXT CSC
TXT CSD
RLD
END
```

```
PHASE FASE2,*
INCLUDE MØD1,(CSF,CSH)
ESD
TXT CSJ
TXT CSK
RLD
END
PHASE FASE3,*+1024
INCLUDE MØD3
ENTRY
```

El programa linkage editor genera las fases en forma temporal en la CIL. La estructura de las fases es la siguiente:

FASE1	FASE2	FASE3
CSA	CSF	CSL
CSB	CSH	
CSC	CSJ	
CSD	CSK	

Al ser cargadas en memoria formarán la estructura que se indica a continuación:



5) Se tiene un programa en fortran y un programa en cobol. Se pide:

- compilar el programa en fortran y catalogarlo en la RL con el nombre de MØDFØR
- catalogar el programa en cobol en la sub-biblioteca C de la SSL con el nombre de LIBCOB
- cambiarle el nombre al módulo ALFA por el de BETA
- cambiarle el nombre al libro CINCO de la sub-biblioteca A por el de FIVE
- listar los directorios de las bibliotecas CIL,RL y SSL

```
// JØB EJ5
// ØPTIØN DECK
// EXEC FFØRTRAN
   }     programa en fortran
/*
*  PØNER DECK DE FØRTRAN
*  EN LA UNIDAD X'00A'
*  EN LA SIGUIENTE FØRMA:
*     CATALR MØDFØR
*     DECK
*  /*
// PAUSE LEER CØMENTARIØ
// ASSGN SYSIPT,X'00A'
// EXEC MAINT
// RESET SYS
// EXEC MAINT
   CATALS C.LIBCØB
   }     programa en cobol
/*
// EXEC MAINT
   RENAMR ALFA,BETA
/*
// EXEC MAINT
   RENAMS A.CINCØ,A.FIVE
/*
// EXEC DSERV
   DSPLY CD,RD,SD
/*
/&
```

4. *Tipos de macro instrucciones en* DOS/VS

Una macro instrucción es un conjunto de instrucciones que se identifica con un nombre, de tal manera que, si se desea realizar el proceso que efectúa dicho conjunto, basta con referirse a él por su nombre proporcionándole al mismo tiempo los parámetros que sean necesarios.

El sistema de macro instrucciones de DOS/VS está compuesto de dos partes básicas:

- Definiciones de macro instrucciones (en lo sucesivo macros) que son rutinas generales escritas en lenguaje simbólico y almacenadas en la sub-biblioteca de macros de la SSL. El conjunto de proposiciones define el nombre, formato y condiciones para generar instrucciones en lenguaje de ensamble.
- Macros del programa fuente que son las que el programador especifica en su programa para indicar al ensamblador cuáles son las definiciones de macros que se llaman desde la SSL.

Nombre de archivo es el identificador del archivo asociado, de impresión, perforación o lectura.

Por ejemplo, si FUNC = PW, en el operando ASOCFLE de la DTFCD se especifica el nombre de la DTFPR y en ésta el de la DTFCD.

BLKSIZE=n

Se especifica la longitud del área de entrada/salida (I0AREA1). Si el formato del registro es variable o indefinido debe indicarse la longitud del registro más grande.

Si el operando se omite se supone longitud 80 bytes con las excepciones siguientes:

160 para modo columna binaria en la 2560, 3505 ó 3525
96 para la 2536 ó 5425
900 para la 3881

CONTROL=YES

Se especifica si se va a utilizar la macro CNTRL en el proceso del archivo. En este caso debe omitirse CTLCHR.

CRDERR=RETRY

Se aplica para salida de tarjetas en la 2520 ó 2540. El sistema envía un mensaje al operador y entra en estado de espera. El operador puede cancelar el job, ignorar el error o pedirle al sistema que reperfore la tarjeta.

Si el operando no se especifica, el error se ignora.

CTLCHR=$\left\{\begin{matrix}\text{ASA}\\ \text{YES}\end{matrix}\right\}$

Se especifica si se va a utilizar el primer carácter como carácter de control. ASA significa American National Standards Institute Inc. y YES corresponde al conjunto de caracteres del Sistema/370.
El operando CONTROL se debe omitir.

DEVADDR=$\left\{\begin{matrix}\text{SYSIPT} \mid \text{SYSPCH} \mid \text{SYSRDR} \mid \text{SYSnnn}\end{matrix}\right\}$

Se especifica el nombre simbólico del dispositivo. El mismo nombre debe aparecer en la proposición ASSGN para asignar la dirección del dispositivo actual de entrada/salida al archivo.

DEVICE=$\left\{\begin{matrix}\underline{2540} \mid 1442 \mid 2501 \mid 2520 \mid 2560\text{P} \mid 2560\text{S} \mid 2596 \mid \\ 3504 \mid 3505 \mid 3525 \mid 5425\text{P} \mid 5425\text{S} \mid 3881\end{matrix}\right\}$

Se especifica el dispositivo de entrada/salida asociado con el archivo. Los parámetros que tienen P o S son dispositivos que tienen depósitos de entrada primario y secundario.

EOFADDR=nombre

Se especifica el nombre de la rutina a la cual se le entrega el

control cuando el sistema detecta fin de archivo.

ERROPT={IGNORE | SKIP | nombre}

Se especifica la acción a seguir al detectarse un error en un archivo de entrada o salida en una 2560, 3504, 3505, 3525 ó 5425. Las tres opciones se pueden especificar para archivos de entrada, en cambio, sólo IGNORE se puede indicar para archivos de salida.

IGNORE: indica que el error debe ser ignorado
SKIP: indica que el registro con error se debe saltar
nombre: es una rutina que tratará el error y a la cual se le entrega el control al detectar uno.

FUNC={R | P | I | RP | RW | RPW | PW}

Se especifica el tipo de archivo que va a ser procesado en una 2560, 3525 ó 5425, R indica lectura, P perforación y W impresión. Si se indica I significa que el archivo será perforado y además interpretado.

Los parámetros restantes se utilizan en conjunto con el operando ASOCFLE para especificar archivos asociados.

IOAREA1=nombre

Se especifica el nombre del área de entrada o salida utilizada para el archivo.

IOAREA2=nombre

Se especifica el nombre de una segunda área de entrada o salida.

IOREG=(r)

Si se utilizan dos áreas de entrada o salida y no se especifica área de trabajo, se debe indicar este operando; el parámetro corresponde a un RUG 2-12.

MODE={E | C | O | R | EO | ER | CO | CR}

El operando indica el modo utilizado para procesar un archivo de entrada o salida en una 2560, 3504, 3505 ó 3525. E corresponde a EBCDIC, C columna binaria, O marca óptica, R eliminación de columna leída. En la 3504 y 3505 se pueden especificar combinaciones de los parámetros.

MODNAME=nombre

Se especifica el nombre del módulo lógico que será usado con la DTF para procesar el archivo. Si el módulo lógico es ensamblado junto con el programa el nombre debe ser el mismo de la macro CDMOD. Si el operando se omite el sistema genera un nombre estándar para llamar el módulo.

OUBLKSZ=n

Se especifica el número máximo de caracteres que se transferirán cada vez. El operando se utiliza en conjunto con IOAREA2, pero sólo para archivos combinados. Se define como tal un archivo en la 1442, 2520 ó 2540 en el cual se lee y perfora el mismo conjunto de tarjetas.

Si no se especifica IOAREA2 se supone la longitud indicada en BLKSIZE.

RDONLY=YES

Se especifica si la DTF es usada con un módulo de lectura solamente. El RUG 13 debe tener la dirección de un área de 72 bytes con alineamiento de doble palabra.

RECFORM={FIXUNB | VARUNB | UNDEF}

Se especifica el formato del registro del archivo: fijo desbloqueado, variable desbloqueado o indefinido.

RECSIZE=(r)

Si se tienen registros indefinidos, este operando especifica un RUG 2-12 que contiene la longitud del registro de salida, esto es, debe cargarse el RUG antes de cada llamada de la macro PUT en el programa.

SEPASMB=YES

Se especifica si la DTF va a ser ensamblada aparte. En este caso, una tarjeta CATALR con el nombre del archivo se perforará encabezando el módulo, además, se define el nombre del archivo como un punto de entrada en el ensamblado.

SSELECT=n

Se especifica el carácter seleccionador de depósito que es válido para el archivo. Si no se indica se supone NR (normal read) o NP (normal punch).

TYPEFLE={INPUT | OUTPUT | CMBND}

Se especifica el tipo de archivo: entrada, salida o combinado.

WORKA=YES

Se especifica si los registros de entrada o salida van a ser procesados en un área de trabajo.

2) Macro CDMOD

Operandos de CDMOD:

CONTROL=YES

Se debe incluir si se utiliza la macro CNTRL. El módulo generado también procesa archivos para los cuales no se hace uso de la macro CNTRL.

CRDERR=RETRY

Se especifica si se incluyó en la DTF.

CTLCHR={ASA | YES}

Se especifica con el mismo parámetro que en la DTF.

DEVICE={2540 | 1442 | 2501 | 2520 | 2560P | 2560S | 2596 | 3504 | 3505 | 3525 | 5425P | 5425S | 3881}

Se especifica con el mismo parámetro que en la DTF.

FUNC={R | P | I | RP | RW | RPW | PW}

Se especifica con el mismo parámetro que en la DTF.

IOAREA2=YES

Se especifica si en la DTF se indicó IOAREA2.

RDONLY=YES

Se especifica si se incluyó en la DTF.

RECFORM={FIXUNB | VARUNB | UNDEF}

Se especifica con el mismo parámetro que en la DTF.

SEPASMB=YES

Se especifica si el módulo va ser ensamblado aparte. Se perfora el módulo con una tarjeta CATALR con el nombre del módulo al comienzo y además, el nombre como punto de entrada en el ensamblado.

TYPEFLE={INPUT | OUTPUT | CMBND}

Se especifica con el mismo parámetro que en la DTF.

WORKA=YES

Se especifica si se incluyó en la DTF.

3) Macro DTFCN. Se utiliza para describir un archivo de entrada o salida que es procesado en una impresora de teclado de la consola, 3210 ó 3215, o una consola de despliegue del operador.
Operandos de DTFCN:

BLKSIZE=n

Se especifica la longitud del área de entrada/salida.

DEVADDR={SYSLOG | SYSnnn}

Se especifica el nombre simbólico del dispositivo:

INPSIZE=n

Se especifica la longitud de la parte de entrada del área de entrada/salida, para uso de la macro PUTR.

IOAREA1=nombre

Véase igual operando de la DTFCD.

SEPASMB=YES

Véase igual operando de la DTFCD.

STLIST=LIST

Se especifica si se va a utilizar la característica de listado de cinta selectiva en la 1403. El operando RECFORM debe tener el parámetro FIXUNB.

UCS={OFF | ON}

Se indica una acción a tomar si se detecta error de datos en una 1403, 5203, 3203 ó 3211. ON significa que los errores de datos serán procesados con un mensaje al operador, OFF significa que los errores serán ignorados y será dejado en blanco el lugar del carácter no imprimible.

WORKA=YES

Véase igual operando de la DTFCD.

5) Macro PRMOD

Operandos de PRMOD.

CONTROL=YES

Véase igual operando de la CDMOD.

CTLCHR={YES | ASA}

Se especifica con el mismo parámetro que en la DTF.

DEVICE={1403 | 1443 | 2245 | 2560P | 2560S | 3203
3211 | 3525 | 5203 | 5425P | 5425S}

Se especifica con el mismo parámetro que en la DTF.

ERROPT=YES

Se especifica si ERROPT=nombre en la DTF, se omite en cualquier otro caso.

FUNC={W[T] | RW[T] | RPW[T] | PW[T]}

Se especifica con el mismo parámetro que en la DTF.

IOAREA2=YES

Se especifica si en la DTF se indicó IOAREA2.

PRINTOV=YES

Se especifica si la macro PRTOV va a ser utilizada en el proceso del archivo.

RDONLY=YES

Se especifica si se incluyó en la DTF.

RECFORM={FIXUNB | VARUNB | UNDEF}

Se especifica con el mismo parámetro que en la DTF.

SEPASMB=YES

Véase igual operando de la CDMOD.

STLIST=YES

Se especifica si se incluyó en la DTF.

WORKA=YES

Se especifica si se incluyó en la DTF.

6) Macro DTFMT. Se utiliza para cada archivo de entrada o salida, en cinta magnética en código EBCDIC o ASCII, que va a ser procesado.

Operandos de DTFMT:

ASCII=YES

Se especifica si la cinta está en código ASCII. El operando no se permite para archivos de trabajo.

BLKSIZE=n

Se especifica la longitud del área de entrada/salida. Si el formato del registro es variable o indefinido, debe indicarse la longitud del registro más grande.

El tamaño máximo del bloque es 32 767 bytes, el mínimo es de 12 bytes. En el caso de registros de salida variables, el tamaño mínimo es de 18 bytes.

BUFOFF={0 | n}

Se especifica si ASCII=YES. En él se indica la longitud de un campo de 0-99 bytes que precede cada registro y que puede contener datos o la longitud física del registro en el caso de longitud variable.

CKPTREC=YES

Se especifica si la cinta de entrada tiene registros "checkpoint" entre los registros de datos. El operando se omite si ASCII=YES (registro "checkpoint" contiene la información necesaria para reiniciar un job sin tener que volver al comienzo de él).

DEVADDR={SYSRDR | SYSIPT | SYSPCH | SYSLST | SYSnnn}

Véase igual operando de la DTFCD. Si es una cinta en código ASCII se debe especificar SYSnnn.

EOFADDR=nombre

Véase igual operando de la DTFCD.

ERREXT=YES

Se especifica para permitir que las rutinas indicadas en ERROPT o WLRERR retornen a MTMOD con la macro ERET (retorno de error).

ERROPT={IGNORE | SKIP | nombre}

Se especifica la acción a seguir al detectarse un bloque erróneo (error de paridad). El significado de los parámetros es similar al indicado en el operando de la DTFCD.

FILABL={NO | STD | NSTD}

Se especifica si el archivo tiene rótulos estándar, no estándar o si no tiene rótulos. ASCII=YES es incompatible con NSTD.

HDRINFO=YES

Se especifica para obtener el contenido de los campos 3-10 del rótulo de encabezamiento estándar en SYSLOG.

IOAREA1=nombre

Se especifica el nombre del área de entrada/salida. Si se procesan registros de longitud variable deben considerarse cuatro bytes para indicar el tamaño del bloque. El operando no se aplica a archivos de trabajo.

IOAREA2=nombre

Se especifica el nombre de una segunda área de entrada/salida.

IOREG=(r)

Se especifica un RUG 2-12 cuando:

- se tienen dos áreas de entrada/salida y no se tiene área de trabajo
- se tienen registros bloqueados y se procesan en las áreas de entrada/salida
- se leen registros de longitud variable desbloqueados
- se leen hacia atrás registros indefinidos
- no se especifica BUFOFF=0 ni WORKA=YES para cinta en código ASCII.

LABADDR=nombre

Se especifica el nombre de la rutina que procesará rótulos no estándar o rótulos estándar del usuario.

LENCHK=YES

Se especifica si la longitud del bloque en una cinta de entrada, en código ASCII, será comparada con la longitud del registro físico. Debe haberse indicado BUFOFF=4 y RECFORM=VARUNB o VARBLK.

MODNAME=nombre

Véase igual operando de la DTFCD.

NOTEPNT={POINTS | YES}

Se aplica sólo a archivos de trabajo. YES indica que las macros NOTE, POINTW, POINTR o POINTS van a ser utilizadas. POINTS indica que sólo esa macro será especificada.

RDONLY=YES

Véase igual operando de la DTFCD,

READ={FORWARD | BACK}

Especifica la dirección en la cual es leída la cinta.

RECFORM={FIXUNB | FIXBLK | VARUNB | VARBLK | SPNBLK | SPNUNB | UNDEF}

Se especifica la organización de los datos en la cinta:

FIXUNB	registros desbloqueados de longitud fija
FIXBLK	registros bloqueados de longitud fija
VARUNB	registros desbloqueados de longitud variable
VARBLK	registros bloqueados de longitud variable
SPNBLK	registros bloqueados de longitud variable "spanned" (registros que pueden ser más largos que el tamaño del bloque y ocupan, por lo tanto, uno o más bloques continuos)
SPNUNB	registros desbloqueados de longitud variable "spanned"
UNDEF	registros de longitud indefinida

Los archivos de trabajo pueden ser FIXUNB o UNDEF.

RECSIZE={n | (r)}

En el caso de registros bloqueados de longitud fija, se especifica el número de caracteres del registro.

En registros indefinidos o "spanned" se especifica un RUG 2-12 que contiene la longitud del registro.

REWIND={UNLOAD | NORWD}

La operación normal con una macro OPEN o CLOSE o condición de fin de volumen (EOV) es rebobinar la cinta al punto de carga pero no descargarla. Con el parámetro UNLOAD se descarga y con NORWD no se rebobina.

SEPASMB=YES

Véase igual operando de la DTFCD.

TPMARK=NO

En cintas de salida con rótulos no estándar, normalmente se graba

una "marca de cinta" (tape mark) al comienzo. Este operando evita esa operación. En cintas sin rótulos se supone este operando.

TYPEFLE={INPUT | OUTPUT | WORK}

Véase igual operando de la DTFCD.

VARBLD=(r)

Se especifica cuando se van a construir registros bloqueados de longitud variable en el área de salida. Se indica el RUG 2-12 que contiene la longitud del espacio que queda disponible en el área.

WLRERR=nombre

Se especifica el nombre de una rutina a la que se le entrega el control si se detecta un registro de longitud errónea en una cinta de entrada.

WORKA=YES

Véase igual operando de la DTFCD.

7) Macro MTMOD
Operandos de MTMOD:

ASCII=YES

Se especifica si se incluyó en la DTF.

CKPTREC=YES

Se especifica si se incluyó en la DTF. El módulo también procesa archivos cuyas DTF no tienen este operando.

ERREXT=YES

Se especifica si se incluyó en la DTF.

ERROPT=YES

Se especifica si se incluyó en la DTF. El módulo también procesa archivos cuyas DTF no tienen este operando.

NOTEPNT={YES | POINTS}

Se especifica si se incluyó en la DTF. El módulo también procesa archivos cuyas DTF no tienen este operando. Si el parámetro es YES, también procesa archivos en cuyas DTF se especificó solamente POINTS.

RDONLY=YES

Se especifica si se incluyó en la DTF.

READ={FORWARD | BACK}

Si se especifica FORWARD, el módulo maneja cintas cuya DTF tiene el mismo parámetro. Con BACK se manejan cintas cuyas DTF tienen cualquiera de los dos parámetros.

RECFORM=$\left\{\begin{matrix}\text{FIXUNB} \mid \text{FIXBLK} \mid \text{VARUNB} \mid \text{VARBLK} \mid \\ \text{SPNBLK} \mid \text{SPNUNB} \mid \text{UNDEF}\end{matrix}\right\}$

Si el operando se omite o si se especifica cualquiera de los dos primeros parámetros, el módulo generado permite procesar registros bloqueados o desbloqueados de longitud fija. Cualquiera de los dos parámetros que siguen produce un módulo que permite procesar registros bloqueados o desbloqueados de longitud variable o "spanned". Si se especifica UNDEF, el módulo generado permite procesar registros indefinidos.

SEPASMB=YES

Véase igual operando de la CDMOD.

TYPEFLE=$\left\{\text{OUTPUT} \mid \text{INPUT} \mid \text{WORK}\right\}$

Si se especifica el parámetro WORK, se genera un módulo para procesar archivos de trabajo. Cualquiera de los otros dos parámetros genera el mismo módulo lógico y formato de tabla.

WORKA=YES

Se especifica si se incluyó en la DTF. El módulo también procesa archivos cuyas DTF no tienen este operando.

B. *Macros imperativas*

Se dividen en macros de iniciación, procesamiento y término.

1) Macro de iniciación OPEN. Deja disponible el archivo para que sea procesado.

Estructura:

$$[\text{nombre}]\ \text{OPEN}\ \left\{\begin{matrix}\text{nombre de archivo 1} \\ (\text{r1})\end{matrix}\right\}\left[,\left\{\begin{matrix}\text{nombre de archivo 2} \\ (\text{r2})\end{matrix}\right\}\ldots\right]$$

donde:

nombre: es el identificador de la macro OPEN
nombre de archivo i: es el identificador de la DTF respectiva
(ri): RUG en el cual se especifica el nombre del archivo. Se recomienda usar registros 2-12. Se puede abrir un máximo de dieciséis archivos.

2) Macro de procesamiento GET. Deja el siguiente registro lógico en secuencia, disponible para que sea procesado en un área de entrada o en un área de trabajo.

Estructura:

$$[\text{nombre}]\ \text{GET}\ \left\{\begin{matrix}\text{nombre de archivo} \\ (1)\end{matrix}\right\}\left[,\left\{\begin{matrix}\text{nombre de área de trabajo} \\ (0)\end{matrix}\right\}\right]$$

donde:

nombre: es el identificador de la macro GET
nombre de archivo: es el identificador de la DTF. El parámetro se puede especificar en el RUG 1
nombre de área de trabajo : es el identificador del área de trabajo utilizada en el proceso del archivo. El parámetro se puede especificar en el RUG 0.

3) Macro de procesamiento PUT. Graba, perfora o imprime registros lógicos que han sido construidos en un área de salida o en un área de trabajo.

[nombre] PUT {nombre de archivo / (1)} [,nombre de área de trabajo / (0)]

[,{STLSP = campo de control / (r)} / ,{STLSK = campo de control / (r)}]

donde:

nombre, nombre de archivo y nombre de área de trabajo tienen igual significado que en la macro GET.

STLSP = campo de control, se especifica un byte de control que permite el espaciado mientras se hace uso de la característica de listado de cinta selectiva en la 1403. El operando STLST = YES debe indicarse en la DTF

STLSK = campo de control, se especifica un byte de control que permite saltar, o lo que es lo mismo, avanzar el carro después de imprimir mientras se hace uso de la característica de listado de cinta selectiva en la 1403. El operando STLST = YES debe indicarse en la DTF.

Ambos operandos se pueden especificar en un RUG 2-12.

4) Macro de procesamiento CNTRL. Proporciona comandos que realizan funciones tales como: rebobinado de cinta, selección de bolsillo receptor de tarjetas, espaciado de líneas, etc.

Estructura:

[nombre] CNTRL {nombre de archivo / (1)} ,código [,n1][,n2]

donde:

código, n1 y n2 se deben indicar de acuerdo con la tabla siguiente:

Unidad	Códigos	n1	n2	comando
Cinta magnética 3420 y serie 2400	los mismos códigos indicados en la proposición MTC (n1 y n2 no se permiten)			
Lecto-perforadora de tarjetas 1442 y 2520	SS		1	selecciona bolsillo 1
			2	selecciona bolsillo 2
	E			salto a bolsillo 1 (sólo la 1442)
Lecto-perforadora 2540	PS	1		selecciona bolsillo
Lectoras 3504 y 3505		2		1, 2 ó 3
Perforadora 3525		3		
Máquina de tarjetas multi-funcional 2560	SS	1		selecciona bolsillo
		2		1, 2, 3, 4 ó 5
		3		
		4		
		5		
Lecto-perforadora 2596	SS	1		selecciona bolsillo 1R ó 3P
		2		selecciona bolsillo 2R ó 4P (R = lectura, P = perforación)
Unidad de tarjetas multi-funcional 5425	SS	1		selecciona bolsillo
		2		1, 2, 3 ó 4
		3		
		4		
Impresoras 1403, 1443, 3203, 3211 y 5203 Perforadora de tarjetas 3525 con impresión	SP	C	d	salta 1, 2 ó 3 líneas
	SK	C	d	salta a canal *C* y/o *d* (*C* es un entero que indica antes de imprimir y *d* es un entero que indica después de imprimir)
Impresoras 1403 y 5203 con el conjunto universal de caracteres o las impresoras 3203 y 3211	UCS	ON		se procesan errores de datos con mensaje al operador
		OFF		errores de datos se ignoran y se dejan blancos

5) Macro de procesamiento PRTOV. Se utiliza con impresoras para especificar la operación que va a ser realizada cuando ocurre una condición de desborde (overflow) de formulario.

Estructura:

[nombre] PRTOV [nombre de archivo (1)] , [9 12] [, [nombre de rutina (0)]]

9 ó 12: indica el canal que controla el desborde de formulario. Al detectarse la condición, se produce un salto a canal 1, esto es, la primera línea de la página siguiente

nombre de rutina: es el identificador de la rutina a la cual se le entrega el control para que maneje la condición de desborde de formulario.

6) Macro de término CLOSE. Exceptuando los archivos de consola, los que fueron abiertos previamente deben ser desactivados con la macro CLOSE. En un archivo de cinta magnética, la operación consiste en grabar un registro de fin de archivo (EOF) y rótulos de salida si los hay. A continuación, la acción que realiza depende del operando REWIND de la DTF respectiva.

Estructura:

[nombre] CLOSE [nombre de archivo 1 (r1)] [, [nombre de archivo 2 (r2)] ...]

Los parámetros son los mismos de la macro OPEN.

C. *Problemas resueltos*

1) Se tiene un archivo en tarjetas. Se pide leer el archivo e imprimir una tarjeta por línea, dejando una línea en blanco entre cada línea impresa. Utilizar dos áreas de entrada y área de trabajo. Se lee desde una 2540 y se imprime en una 1403. Contemplar la posibilidad de desborde de formulario.

```
                                                              col. 72
// JØB PRØB1
// ØPTIØN LINK,LIST,NØDECK,XREF
// EXEC ASSEMBLY
        PRINT NØGEN
        START
        SPACE 3
LECTØ   DTFCD   BLKSIZE=80,DEVADDR=SYSIPT,                    x
                DEVICE=2540,EØFADDR=FINTAR,                   x
                IØAREA1=AREA1,IØAREA2=AREA2,                  x
                WØRKA=YES,MØDNAME=TAR
TAR     CDMØD   WØRKA=YES,IØAREA2=YES
IMPRE   DTFPR   BLKSIZE=80,DEVADDR=SYSLST,                    x
                DEVICE=1403,CØNTRØL=YES,                      x
                IØAREA1=AREA3,IØAREA2=AREA4,                  x
                MØDNAME=MØDPR,WØRKA=YES,                      x
                PRINTØV=YES
```

```
MØDPR  PRMØD  WØRKA=YES,CØNTRØL=YES,                    x
               PRINTØV=YES,IØAREA2=YES
       SPACE  2
AREA1  DS     CL80
AREA2  DS     CL80
AREA3  DS     CL80
AREA4  DS     CL80
       SPACE  3
INICIØ BALR   2,0
       USING  *,2
       ØPEN   LECTØ,IMPRE
       CNTRL  IMPRE,SK,1
READ   GET    LECTØ,A
       PUT    IMPRE,A
       CNTRL  IMPRE,SP,1
       PRTØV  IMPRE,12
       B      READ
FINTAR CLØSE  LECTØ,IMPRE
       EØJ
A      DS     CL80
       END    INICIØ
/*
//  EXEC LNKEDT
//  EXEC
       }datos
/*
/&
```

2) Se tiene un archivo en tarjetas. Se pide leer el archivo desde la unidad lógica SYS014 y grabarlo en la cinta magnética asignada a SYS009 bloqueado, diez registros de ochenta caracteres por bloque. Utilizar dos áreas de entrada y área de trabajo.

col. 72

```
//  JØB PRØB2
//  ØPTIØN LINK,LIST,XREF,NØDECK
//  EXEC ASSEMBLY
         START
         PRINT  NØGEN
INPUT    DTFCD  BLKSIZE=80,DEVADDR=SYS014,          x
                IØAREA1=T1,IØAREA2=T2,             x
                WØRKA=YES,EØFADDR=FINAL,           x
                MØDNAME=MØDCD
ØUTPUT   DTFMT  BLKSIZE=800,RECSIZE=80,            x
                RECFØRM=FIXBLK,TYPEFLE=ØUTPUT,     x
                WØRKA=YES,IØAREA1=C1,              x
                IØAREA2=C2,FILABL=NØ,              x
                MØDNAME=MØDMT,DEVADDR=SYS009
MØDCD    CDMØD  WØRKA=YES
MØDMT    MTMØD  WØRKA=YES
```

```
T1       DS       CL80
T2       DS       CL80
C1       DS       10CL80
C2       DS       10CL80
INICIØ   BALR     2,0
         USING    *,2
         ØPEN     INPUT,ØUTPUT
LEE      GET      INPUT,B
         PUT      ØUTPUT,B
         B        LEE
FINAL    CLØSE    INPUT,ØUTPUT
         EØJ
B        DS       CL80
         END      INICIØ
/*
// EXEC LNKEDT
// ASSGN SYS014,X'001'
// ASSGN SYS009,X'182'
*  MØNTAR CARRETE M-245 EN UNIDAD X'182'
// PAUSE Y LUEGØ DAR EØB
// EXEC
     } datos
/*
/&
```

BIBLIOGRAFIA

1. Auslander, M.A. y Jaffe, J.F., "Functional Structure of the IBM Virtual Storage Operating System", Parte I, en *IBM System Journal*, Vol. 12, Nº 4, 1973, págs. 368-380.
2. Bensoussan, A., Clingen, C.T. y Daley, R.C., "The Multics Virtual Memory: Concepts an Design", en *Communications of the ACM*, Vol. 15, Nº 5, mayo 1972, págs. 308-318.
3. Cohen, Leo J., *Operating System Analysis and Design*, USA, Hayden Book Company Inc., 1970, 182 págs.
4. Colin, A.J.T., *Introduction to Operating Systems*, Inglaterra, Redwood Press Limited, 1971, 120 págs.
5. Denning, Peter J., "Virtual Memory", en *Computing Surveys*, Vol. 2, Nº 3, septiembre de 1970, págs. 153-189.
6. Denning, Peter J., "Third Generation Computer Systems", en *Computing Surveys*, Vol. 3, Nº 4, diciembre de 1971, págs. 175-216.
7. IBM System Products Division, *Introduction to DOS/VS*, Nueva York, 1973.
8. IBM System Products Division, *DOS/VS Supervisor and I/O Macros*, Nueva York, 1973.
9. IBM System Products Division, *DOS/VS System Management Guide*, Nueva York, 1974.
10. IBM System Products Division, *DOS/VS System Control Statements*, Nueva York, 1973.
11. IBM System Products Division, *IBM System/360 Operating System, Introduction*, Nueva York, 1971.
12. Katzan, Harry Jr., *Advanced Programming*, USA, Van Nostrand Reinhold Company, 1970, 285 págs.
13. Martin, James, *Desingn of Real Time Computer Analysis*, USA, Prentice-Hall, Inc., 1967, 629 págs.
14. Mealy, G.H., Witt, B.Y. y Clark, W.A., "The functional structure of OS/360", en *IBM System Journal*, Vol. 5, Nº 1, 1966, págs. 2-51.
15. Parmelee, R.P., Peterson. T.Y., Tillman, C.C. y Hatfield, D.J., "Virtual storage and virtual machine concepts", en *IBM System Journal* Nº 2, 1972, págs. 99-130.
16. Rosen, Saul, "Programming Systems and Languages 1965-1975", en *Communications of the ACM*, Vol. 15, Nº 7, julio 1972, págs. 591-610.
17. Rosin, Robert F., "Supervisory and Monitor Systems," en *Computing Surveys*, Vol. 1, marzo, 1969, págs. 37-54.

Impreso en CEPCO Ltda.
Adriana Undurraga 223
Teléfono: 718856
Santiago

9284.-Esc. Tip. Sales.

El presente libro, que agrupa parte del material entregado en el primer curso de Procesamiento Electrónico de Datos aplicado a ciencias sociales y que corresponde a aquellos temas expuestos en él por el profesor Víctor Sánchez C., pretende ser una guía y manual de consulta para el estudiante de computación. Está compuesto por cinco capítulos que son: 1) Introducción a computación, donde se dan a conocer los principales conceptos básicos de computación; 2) Lenguajes y programación, en el que se hace especial hincapié en el uso del diagrama de flujo; 3) FORTRAN IV, en el que se exponen con numerosos ejemplos las distintas proposiciones de dicho lenguaje; 4) Lenguajes de ensamble, que permite conocer en detalle un lenguaje orientado a un computador, y 5) Sistemas de Operación, que da una visión general de los objetivos y estructura de ellos, como asimismo de los principales conceptos utilizados.