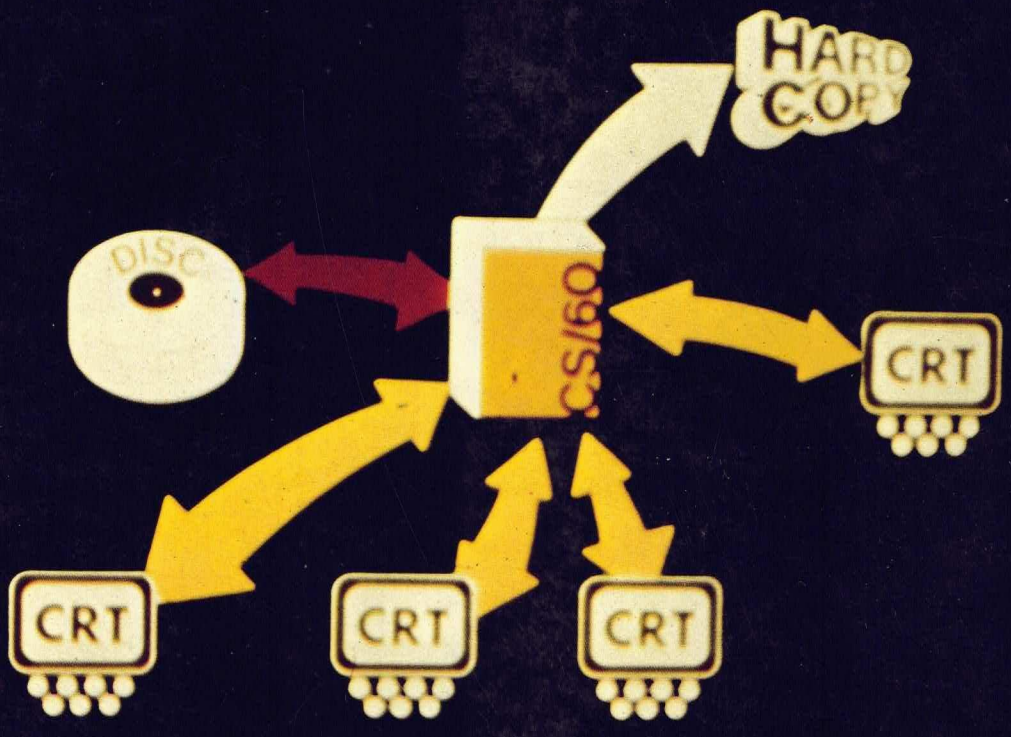
PROGRAMACION DE ORDENADORES

ESTRUCTURA BASICA DEL PROCESO DE DATOS



EDITORIAL tesys

TOMO 3

El presente tratado de **Programación de Ordenadores** está compuesto por un conjunto de temas que constituyen la estructura imprescindible para todo aquel que desee un amplio conocimiento del campo de la Programación.

Dichos temas vienen englobados en volúmenes diversos que a continuación se detallan; constituyéndose por separado como unidades monográficas susceptibles de un estudio particular, al margen del conocimiento conjunto y global de todos ellos, como compendio completo para el logro final de programar.

La agrupación por tomos se desarrolla de la siguiente forma:

Tomo 1. **Estructura Básica del Proceso de datos.**

Tomo 2. **Estructura Básica del Proceso de datos.**

Tomo 3. **Estructura Básica del Proceso de datos.**

Tomo 4. **Estructura Básica del Proceso de datos.**

Tomo 5. **Estructura Básica del Proceso de datos.**

Tomo 6. **Lenguaje Basic**

Tomo 7. **Lenguaje Cobol**

Tomo 8. **Lenguaje Cobol**

Tomo 9. **Leyes de Construcción de programas.**

Tomo 10. **Diccionario de Proceso de datos**

El tratamiento de los temas comprende tanto los aspectos teóricos como las aplicaciones al diseño de estructuras básicas de posible uso inmediato, reagrupándose en los siguientes conjuntos específicos que por sí mismos pasan a ser unidades independientes.

Son los siguientes:

Grupo I: Tomos 1, 2, 3, 4 y 5. **(Estructura Básica del Proceso de Datos).**

Grupo II: Tomo 6. **(Lenguaje Basic).**

Grupo III: Tomos 7 y 8. **(Lenguaje Cobol).**

Grupo IV: Tomo 9. **(Leyes de Construcción de Programas).**

PROGRAMACION DE ORDENADORES

ESTRUCTURA BASICA DEL PROCESO DE DATOS

PROGRAMACION DE ORDENADORES

ESTRUCTURA BASICA DEL PROCESO DE DATOS

AUTOR

JOSE LUIS MORALES PASCUAL
Licenciado en Informática

COLABORADORES

CARLOS ANTONIO MARTINEZ TREVIN
Ingeniero de Telecomunicación

ANDRES J. ERNESTO RUANA GIRONES
Ingeniero Industrial

BARCELONA (ESPAÑA)

© E.C.C. — Barcelona, 1980

Editado por:
EDITORIAL TESYS, Barcelona (España), 1980

Primera edición: Octubre, 1980

Depósito Legal: B.: 25869-1980 (III)
ISBN 84-85838-00-9 (Obra completa)
ISBN 84-85838-03-3 (Tomo III)

Impreso por:
GRAFOS, Arte sobre papel.
Paseo de Carlos I, 157 - Barcelona-13

Impreso en España - Printed in Spain

CONTENIDO

Introducción 9

1. Etapas en la resolución de un problema 11

2. Organigramas 37

3. Tablas de decisión 77

4. Lenguajes de programación 105

Soluciones de los ejercicios 139

Indice general 155

INTRODUCCION

Los ordenadores pueden realizar diversas tareas, resolver problemas científicos o comerciales, y posee la capacidad de tratar gran volumen de datos.

La aplicación de los ordenadores en cualquiera de estos campos para resolver un determinado problema, requiere un conjunto de medios y operaciones que sirvan para preparar el programa de forma que pueda ser tratado por el ordenador. En el primer capítulo de este texto se analizan las fases que debemos seguir desde el momento que nos presentan el problema hasta que se obtiene el resultado por ordenador.

En los capítulos dos y tres se explican los métodos para indicar al ordenador cuáles son los pasos que debe seguir, y en qué orden debe hacerlo.

Esta parte de la obra «Programación de Ordenadores» está destinada a preparar al lector a «dialogar» con el ordenador. De ahí que se dé gran importancia a la construcción de algoritmos, así como su representación en forma de organigramas o de tablas de decisión.

La tarea del programador consiste en especificar detalladamente, mediante esquemas (algoritmos), el orden en que el ordenador debe tratar los datos.

Para ello, el programador debe conocer algún lenguaje con el que pueda comunicarse con el ordenador. En el capítulo cuatro se estudian los diferentes tipos de lenguajes de programación sin entrar en detalle en uno concreto.

1.

ETAPAS EN LA RESOLUCIÓN DE UN PROBLEMA

CUADRO RESUMEN DE LA LECCIÓN

Etapas en la resolución de un problema

- Definición del problema
- Análisis del problema
- Programación
  - Organigrama
  - Codificación
  - Depuración
  - Documentación
- Ejecución del programa
  - Desarrollo lineal de un programa
  - Instrucciones de bifurcación hacia adelante
  - Instrucciones de bifurcación hacia atrás
  - Subprogramas
  - Macroinstrucciones
- Resultados

La aplicación de los ordenadores ha proporcionado mayor rapidez, seguridad, exactitud y rentabilidad a la resolución de problemas científicos, empresariales, de teleproceso (proceso de datos a distancia), de Medicina, etc.

En las aplicaciones comerciales así como en muchas aplicaciones científicas se procesa un gran volumen de datos. Existe poca diferencia, en general, del tratamiento que sufren los datos en una aplicación comercial o científica.

Las funciones que se realizan en una aplicación no las realiza en su totalidad el ordenador ya que existen funciones muy importantes que se deben realizar manualmente, funciones efectuadas por el analista, el programador o cualquier otra persona relacionada con el sistema. La preparación de algunos problemas para su posterior resolución mediante el ordenador puede resultar larga y costosa.

Una *aplicación* es el conjunto de operaciones necesarias para resolver un problema mediante un sistema de proceso de datos.

Por ejemplo, una aplicación puede ser la facturación en una empresa.

Para resolver un problema en un sistema electrónico de proceso de datos es necesario seguir un plan sistemático con las siguientes etapas:

1. Definición del problema.
2. Análisis del problema.
3. Realización del programa.
4. Ejecución del programa.
5. Evaluación de los resultados.

1.1. DEFINICIÓN DEL PROBLEMA

Todo lo que va a procesarse en un ordenador debe considerarse como un problema que exige una definición antes de procesarse.

La primera fase para la resolución de un problema en un ordenador consiste, por tanto, en identificar y definir el problema de la forma más exacta posible. Una mala definición del problema puede llegar a complicar notablemente su resolución.

En esta primera fase se estudia detalladamente el problema con el fin de entender el enunciado del mismo y para saber qué es lo que pide.

Todo enunciado de un problema debe identificar y definir el problema mediante una exposición clara y detallada de los procedimientos, de las acciones a seguir y del equipo que se va a utilizar.

Las operaciones a realizar en esta primera etapa son:

- descripción general del sistema de proceso de datos.
- Descripción de los datos que van a utilizarse en el proceso.
- Descripción del tratamiento que sufrirán los datos para la obtención de los resultados.

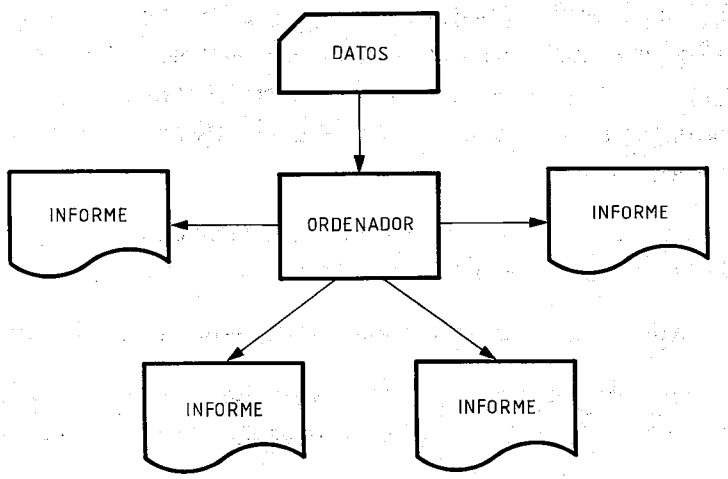


Fig. 1.1. Una aplicación *es el conjunto de operaciones que se realizan en el* ordenador *con unos* datos *de entrada para obtener el resultado* (informes) *de un problema.*

- Definición de los objetivos. Descripción de los resultados que se desean obtener.
- Obtener la información adicional necesaria para resolver el problema.

Es muy importante definir claramente el problema y escribir correctamente el enunciado, resolviendo todo tipo de ambigüedades del mismo, antes de comenzar el proceso. Asimismo, es necesario revisar una y otra vez el modo en que debe realizarse el enunciado del problema antes de procesarlo.

1.2. ANÁLISIS DEL PROBLEMA

En esta segunda etapa se efectúa un estudio del problema analizando la posibilidad de mecanizarlo, es decir, especificando si puede mecanizar el problema y si resulta conveniente esta mecanización.

Algunos problemas sencillos no requieren la aplicación del ordenador ya que con la utilización de un sistema mecánico o manual es suficiente para su resolución. Varios problemas, que en principio se consideran adecuados para resolverlos mediante un ordenador, tal vez resulte más apropiado utilizar métodos manuales o manuales-mecánicos.

Si se decide el uso de un sistema manual o mecánico deben organizarse y describirse los esquemas y procedimientos de trabajo, los impresos que van a utilizarse y sus características. Luego se efectúan las operaciones de forma manual o mecánica y los resultados obtenidos se envían a las personas interesadas. Después se almacenan los datos de entrada y de salida y los procedimientos de trabajo para usarlos posteriormente en otro trabajo.

Si la decisión es utilizar un sistema electrónico de proceso de datos para la resolución del problema en sustitución de un sistema manual o mecánico, ha de efectuarse una preparación previa para la puesta en marcha del nuevo sistema. Esta preparación consta de las siguientes operaciones:

- Análisis de la situación actual.
- Determinar las características del equipo que va a utilizarse. Determinar el tipo de ordenador, la capacidad de la memoria, las unidades periféricas y las planificaciones del ordenador.
- Determinar el coste y tiempo que supondrá la conversión de un sistema manual a uno mecanizado.

- Determinar la frecuencia con que va a utilizarse el sistema.
- Determinar la rentabilidad del nuevo sistema.

Estas operaciones las efectúa el analista de sistemas.

Cuando se planifica una aplicación que va a tratarse por ordenador se deben pasar los antiguos registros a los medios utilizados por el nuevo sistema. Los antiguos registros pueden estar en los libros de contabilidad, en fichas donde la información es legible por el hombre pero no por el ordenador. Estos registros se deben transformar a un formato legible por el ordenador y almacenarlos en soportes (tarjetas perforadas, cintas y discos magnéticos, etc.), constituyendo los nuevos ficheros.

El almacenamiento de la información en estos ficheros resulta más fácil y efectivo que en sistemas manuales, sin embargo debe procurarse que no se repita la información en distintos ficheros y utilizar el mínimo de ficheros necesarios.

En la definición del problema se describen los datos de entrada y los de salida, así como la información que se desea obtener. En la fase de análisis debe especificarse cuál es la naturaleza y la situación actual de los datos del problema. Por ejemplo, especificar cuáles son los documentos originales de los datos que van a utilizarse: albaranes, facturas, etc.

Debe tenerse en cuenta el volumen de datos que se manejan y sus características: ficheros de datos (como por ejemplo, los ficheros de clientes o de artículos de una determinada empresa) para consultas o para puesta al día (actualización).

En la fase de análisis deben efectuarse, además las siguientes operaciones:

- Definir los datos de E/S de una forma definitiva.
- Describir el tipo de datos: numéricos, alfabéticos o alfanuméricos.
- Describir el formato de los datos de E/S: tamaño o longitud.
- Describir los soportes de información en dónde se encuentran los datos de entrada y en dónde se van a obtener los datos de salida y resultados (ficheros en disco magnético, cinta magnética, etc.), así como el tratamiento que se les dará.
- Especificar la disposición de los datos en los soportes de información de entrada. Determinar la situación de los datos en los ficheros, el modo en que están agrupados los datos (organización de los datos) y la forma de tratarlos. Los datos deben prepararse en los soportes de forma que el tiempo de proceso sea el mínimo.

- describir los informes o documentos de salida y la situación de los datos en estos documentos con el fin de preparar las operaciones necesarias para obtener los resultados de la forma que se necesitan.
- describir los periféricos y las zonas de la memoria que van a utilizarse.

El análisis y evaluación de todos los datos disponibles nos llevará a decidir el tipo de acciones a realizar, como por ejemplo:

- Decidir si debe mecanizarse todo el trabajo o, en otro caso, qué operaciones se mecanizarán.
- Reducir los flujos de información al mínimo.
- Evitar redundancias o repeticiones en los documentos.
- etc.

El análisis del problema permite hallar la solución mejor y más rentable, con el máximo rendimiento del equipo que va a utilizarse.

1.3. PROGRAMACION

Después de efectuado el análisis, la siguiente etapa en la resolución de un problema es la realización del programa. Esta etapa, denominada *fase de programación,* consiste en establecer, con los resultados del análisis, un conjunto de programas aptos para su desarrollo en el ordenador.

En esta etapa debe decidirse el tipo de programa que va a utilizarse. Puede ocurrir que se utilice un programa ya desarrollado y procesado para la resolución de problemas anteriores, en cuyo caso, este programa se encuentra almacenado en una *biblioteca de programas* de donde ha de escogerse de entre los distintos programas que componen dicha biblioteca.

En el caso de tener que escribir un nuevo programa han de efectuarse las siguientes operaciones:

- Transcribir los datos en un medio de entrada.
- Asignar unas posiciones en la Memoria para los datos o para las instrucciones.
- Especificar las operaciones aritméticas/lógicas: sumas, restas, comparaciones, transferencias, etc.
- Especificar las secuencias (orden) de las operaciones y las posibles alteraciones de estas secuencias, es decir, del orden en que se realizan las operaciones.

Para realizar estas operaciones la fase de programación consta de las siguientes etapas:

- Construcción del organigrama.
- Codificación.
- Depuración.
- Documentación.

1.3.1. Construcción del organigrama

En el caso de tener que escribir un nuevo programa, el primer paso consiste en desarrollar un procedimiento de cálculo o un esquema de los pasos matemáticos y lógicos para la resolución del problema.

En esta etapa debe plantearse y organizarse la solución del problema de forma que pueda ser tratado por el ordenador. Puesto que el ordenador es una máquina *secuencial,* es decir, que realiza sus operaciones paso a paso, una tras otra, los problemas deben enunciarse como una secuencia lógica de operaciones o acciones a realizar por el ordenador.

Los procedimientos o instrucciones que constituyen el programa suelen expresarse en forma de *organigramas* en donde se representa gráficamente la estructura lógica del problema.

Por tanto, el *organigrama,* o *diagrama de flujo,* es la representación gráfica de la solución del problema en donde se utilizan símbolos o bloques (figuras geométricas) para representar los datos, las operaciones, el flujo de la información y las decisiones lógicas del programa.

Un ejemplo de organigrama es el de la figura 1.2 que se presenta en las dos páginas siguientes, en donde están representados los pasos para la resolución de un problema mediante un sistema de proceso de datos.

1.3.2. Codificación

El procedimiento desarrollado en la fase anterior (organigrama) debe expresarse de la forma más apropiada al problema y al ordenador en donde va a resolverse.

Los organigramas pueden ser entendidos por el hombre pero no por el ordenador. Para que el ordenador pueda entender estos procedimientos y

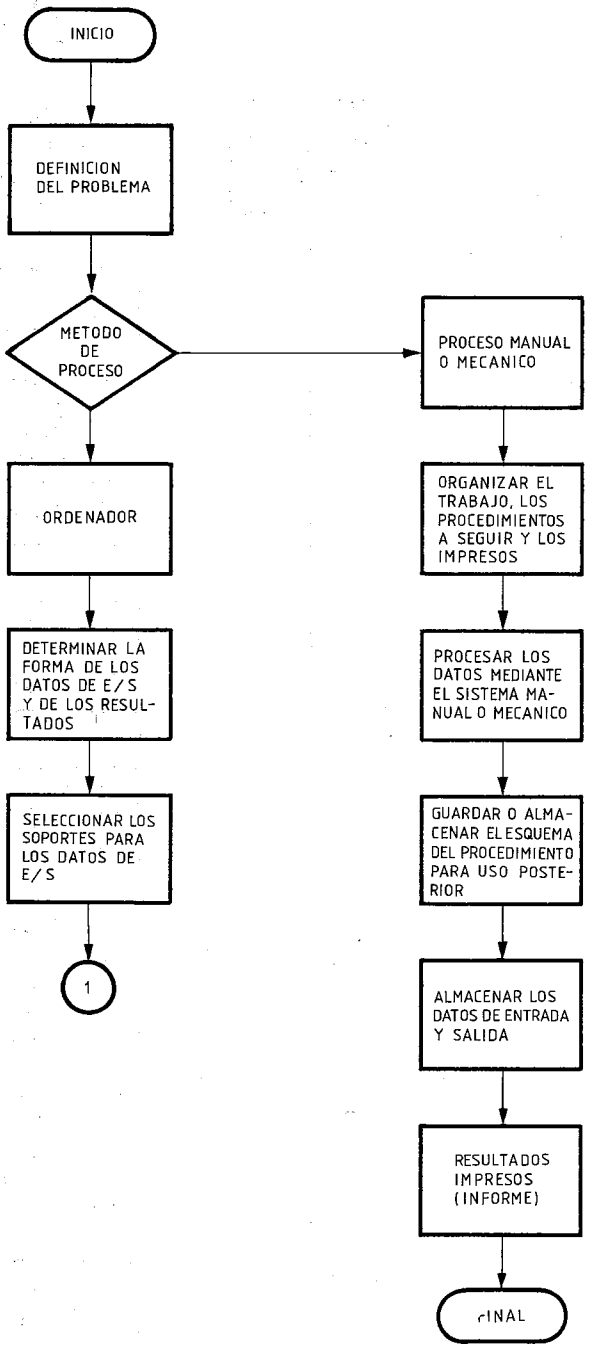
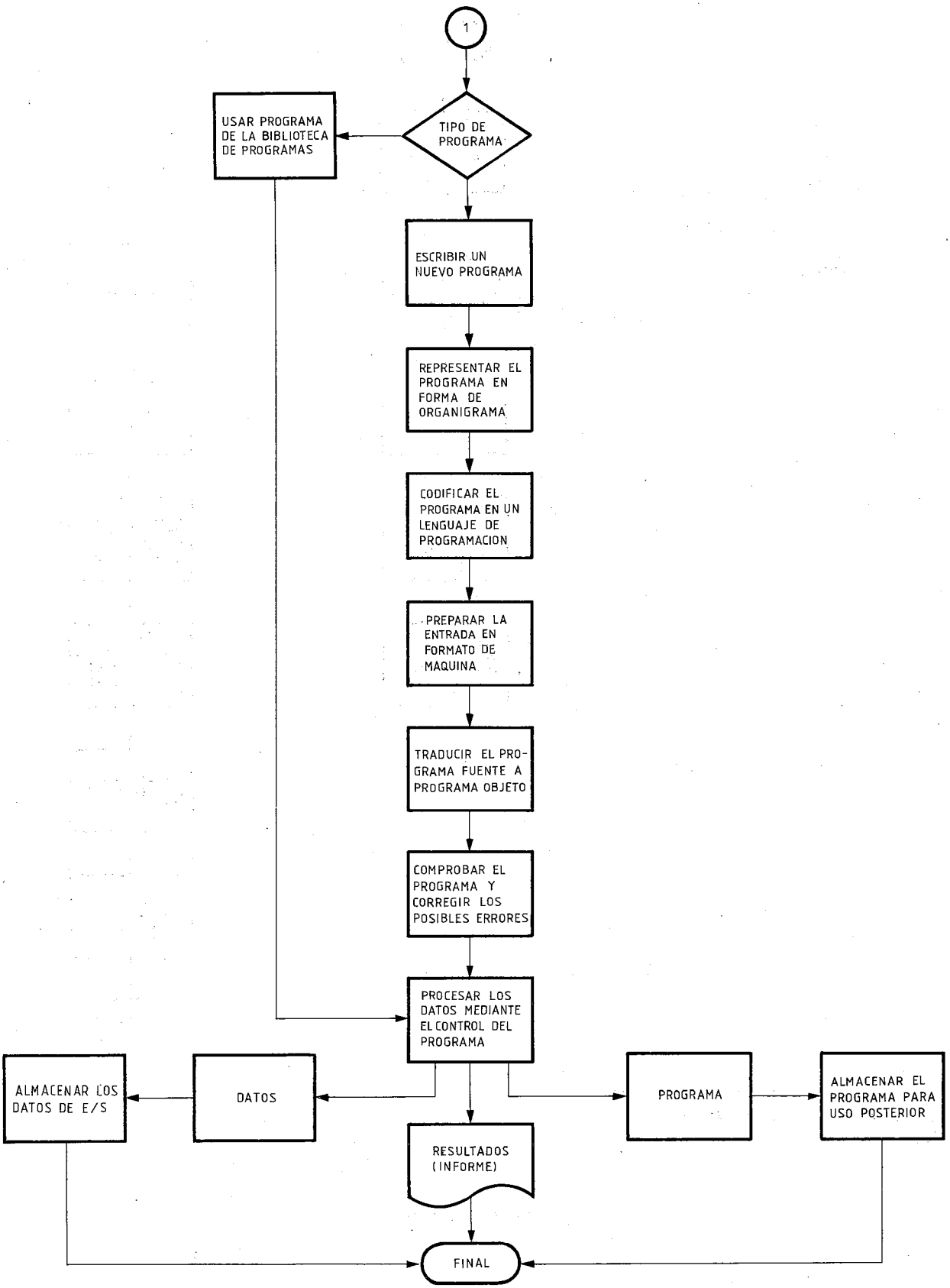


Fig. 1.2. Etapas para la resolución de un problema en un sistema de proceso de datos.


1
TIPO DE PROGRAMA
USAR PROGRAMA DE LA BIBLIOTECA DE PROGRAMAS
ESCRIBIR UN NUEVO PROGRAMA
REPRESENTAR EL PROGRAMA EN FORMA DE ORGANIGRAMA
CODIFICAR EL PROGRAMA EN UN LENGUAJE DE PROGRAMACION
PREPARAR LA ENTRADA EN FORMATO DE MAQUINA
TRADUCIR EL PROGRAMA FUENTE A PROGRAMA OBJETO
COMPROBAR EL PROGRAMA Y CORREGIR LOS POSIBLES ERRORES
PROCESAR LOS DATOS MEDIANTE EL CONTROL DEL PROGRAMA
ALMACENAR LOS DATOS DE E/S
DATOS
PROGRAMA
ALMACENAR EL PROGRAMA PARA USO POSTERIOR
RESULTADOS (INFORME)
FINAL

esquemas de las acciones a realizar es necesario utilizar un determinado *código,* inteligible por el ordenador. Estos códigos se denominan *lenguajes de programación.*

La escritura del programa en uno de estos lenguajes se denomina *codificación del programa.*

La persona que realiza el programa (programador) escribe o codifica el programa siguiendo el camino lógico indicado por el organigrama, y lo hace sobre un documento de papel llamado *hoja de codificación.* Este documento está diseñado para representar las instrucciones del programa en el lenguaje elegido.

Como veremos más adelante, existen diversos tipos de lenguajes de programación. Para obtener el máximo rendimiento del ordenador debe construirse el programa más conveniente con el lenguaje de programación más apropiado para el problema a resolver.

La mayor parte de las instalaciones de proceso de datos disponen de perforistas para perforar en tarjetas el programa codificado por el programador.

Para evitar posibles errores de perforación, el programador debe seguir algunas normas que suelen utilizarse normalmente a la hora de codificar el programa.

Por ejemplo: el número cero puede confundirse con la letra O, el 1 con la i, el 2 con la z, el 5 con la s, etc. Una de las normas más usuales para distinguir estos caracteres entre sí es la siguiente:

número	*letra*
0	Ø
1	I
2	Z
5	S

Fig. 1.3. Hoja de codificación para el lenguaje de programación COBOL.

IBM

Hoja de Codificación en COBOL

GX10-8028-1 (GX28-1464-5)
Impreso en España

Sistema		Instrucciones para Perforación		Hoja	De
Programa		Signo		Formato de Ficha	
Programado por	Fecha	Perforaciones			

Secuencia (Página) 1 3	Secuencia (Línea) 4 6	A 7 8	B 12	Declaración COBOL 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72	Identificación 76 80
	01				
	02				
	03				
	04				
	05				
	06				
	07				
	08				
	09				
	10				
	11				
	12				
	13				
	14				
	15				
	16				
	17				
	18				
	19				
	20				

1 3 4 6 7 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80

1.3.3. Depuración

La fase de *depuración del programa* consiste en verificar, localizar y suprimir los errores de un programa.

Durante la elaboración del programa pueden producirse errores de lógica o errores de codificación, siendo estos últimos más frecuentes.

Si se produce un error de codificación puede ocurrir que no se inicie la ejecución del programa o que se interrumpa en caso de haber comenzado, indicándose de alguna manera el error que ha motivado la interrupción.

Cuando se corrigen los errores se efectúa una nueva prueba o «pasada por ordenador» hasta que el programa se encuentre en condiciones de ser ejecutado.

Los errores de programación no pueden ser corregidos, aunque sí detectados, por el ordenador, debiendo ser el programador o usuario quien deba corregirlos. Por lo tanto la depuración de un programa corresponde al programador, quien debe realizar las siguientes operaciones:

- Diseñar un método de comprobación o verificación.
- Detectar e identificar los fallos en el programa que producen los errores.
- Corregir los errores.
- Comprobar de nuevo el programa corregido.

Cuando se produce un error lógico, el ordenador no da resultados o los da falsos. En este caso se efectúa una comprobación detallada de la lógica del programa hasta que se conozca con exactitud de dónde proviene el error.

Una vez corregida y revisada la lógica del programa se codifica y se vuelve a pasar el programa por el ordenador para asegurar un funcionamiento correcto.

1.3.4. Documentación

Una vez depurado el programa, el programador debe preparar una *documentación* adecuada que lo acompañe, para que el programa pueda ser utilizado por cualquier otra persona que no sea el propio programador.

La documentación está constituida por una serie de registros tales como:

- Nombre del programa.
- Autor.

- Fecha de terminación.
- Sistema de proceso de datos.
- Lenguaje de programación.
- Breve descripción del problema.
- Breve exposición de los objetivos del programa.
- Instrucciones para procesar el programa.
- Organigrama del programa.
- Listado del programa (codificado) anotado con exactitud y actualizado.

1.4. EJECUCIÓN DEL PROGRAMA

La siguiente etapa para resolver el problema consiste en introducir los datos y las instrucciones (programa) en el ordenador para su procesamiento.

Los datos pueden introducirse en la memoria del ordenador al inicio del proceso o durante la ejecución del programa.

Las instrucciones se almacenan en la memoria en el mismo orden en que han de ejecutarse posteriormente.

Las instrucciones de un programa se ejecutan, generalmente, en la misma secuencia en que han sido almacenadas en la memoria. Por lo que el programador debe codificar el programa en esta misma secuencia.

El ordenador utiliza un programa para convertir las instrucciones, codificadas en el lenguaje elegido, a instrucciones máquina (las instrucciones máquina son las que realmente entiende el ordenador). Este programa traductor recibe el nombre de *ensamblador* o *compilador.*

Aunque también es posible la codificación directa de instrucciones máquina, generalmente se codifican en un *lenguaje fuente* que el programador puede comprender más fácilmente y que puede ser interpretado por un programa compilador o ensamblador.

El programa resultante de esta traducción se llama *programa objeto* y el programa escrito por el programador se llama *programa fuente.* Ambos programas tienen la misma descripción pero están escritos con lenguajes diferentes: el programa fuente con un lenguaje fuente y el programa objeto con un lenguaje objeto *(lenguaje máquina).*

El programa proporciona al ordenador una descripción detallada del problema y el modo de resolverlo: un *programa* es un conjunto ordenado de instrucciones que dirigen y controlan la ejecución de las operaciones que debe efectuar el ordenador, puesto que estas instrucciones están basadas en el diseño de los circuitos del ordenador.

El ordenador realiza las operaciones en el orden en que le indican las instrucciones del programa, en donde se explica detalladamente el tratamiento al que deben ser sometidos los datos para la consecución de los resultados deseados.

Un programa consta de tres partes:

- *Principio de programa:* en esta parte del programa se describe el estado inicial de las posiciones de la Memoria Central que se van a utilizar en la resolución del problema.
  Por ejemplo, poner a cero algunas posiciones de la memoria que se van a utilizar como contadores.
  Las instrucciones del principio de programa sólo se realizan una vez.
- *Cuerpo de programa:* consta de una serie de instrucciones que se ejecutan repetidas veces, como por ejemplo, la lectura de los datos, las operaciones de cálculo y la escritura de resultados.
- *Fin de programa:* consta de las instrucciones de parada del funcionamiento del ordenador o de preparar el inicio de la ejecución del programa siguiente.

Las instrucciones se almacenan en la memoria del ordenador para ser ejecutadas en el mismo orden en que han sido codificadas y almacenadas *(programa lineal)* excepto cuando se ejecuta una instrucción de bifurcación.

Un programa puede abandonar la secuencia operacional por dos procedimientos diferentes:

a) Mediante una *bifurcación incondicional,* efectuando una transferencia sin tener que realizar comparación alguna. Ejemplo: «continuar en la instrucción 14».

b) Mediante una *bifurcación condicional,* es decir, a partir de una comparación o decisión.

Una bifurcación incondicional es lo mismo que continuar con una secuencia de operaciones sin desviarse pero se diferencia del desarrollo lineal de un programa porque distintas bifurcaciones pueden conducir al mismo

lugar del programa o porque una bifurcación incondicional puede dirigirse a una instrucción anterior.

En un programa pueden darse tres tipos de bifurcación:

- Bifurcación hacia adelante.
- Bifurcación hacia atrás.
- Bifurcación a un subprograma.

1.4.1. Desarrollo lineal de un programa

Cuando las instrucciones de un programa se ejecutan secuencialmente, sin bifurcaciones, decisiones ni comparaciones, se dice que este programa se ha escrito mediante *codificación rectilínea* o que su desarrollo es *lineal.*

Supongamos el siguiente programa:

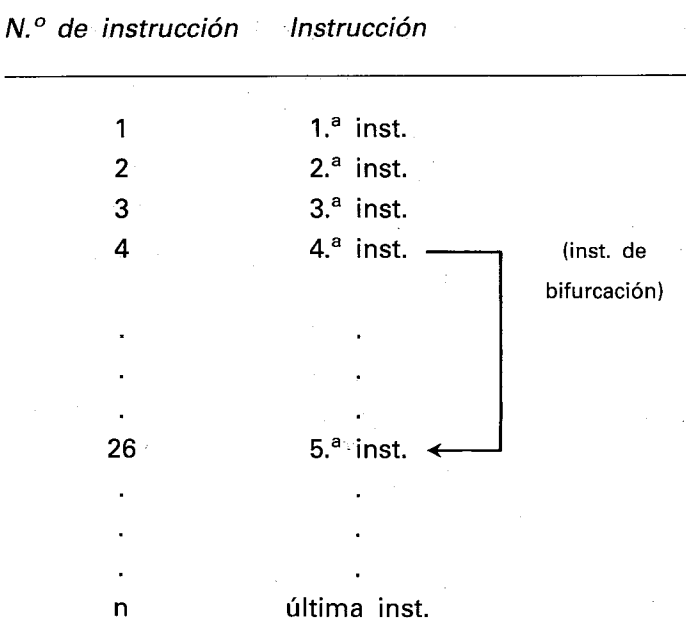
N.º de instrucción	*Instrucción*
1	1.ª inst.
2	2.ª inst.
3	3.ª inst.
4	4.ª inst.
.	.
.	.
.	.
n	última instrucción a ejecutar

Este sería el desarrollo lineal de un programa, en el cual no apareciesen instrucciones de bifurcación.

El desarrollo lineal de un programa se interrumpe cuando se ejecuta una instrucción de bifurcación

1.4.2. Instrucciones de bifurcación hacia adelante

Supongamos el siguiente programa con instrucción de bifurcación hacia adelante:



La instrucción n.º 4 del programa es una instrucción de bifurcación que produce la continuación del programa en la instrucción número 26. El programa continúa su desarrollo hasta que se ejecuta la última instrucción, n.

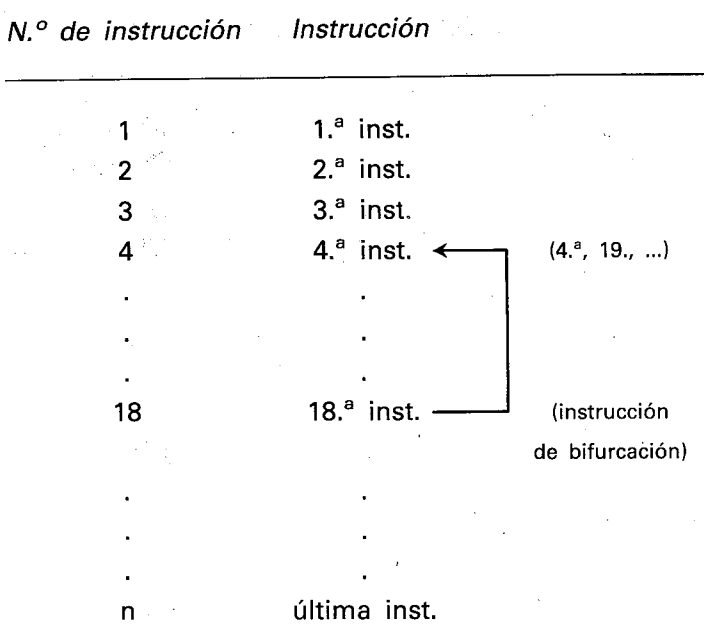
Una bifurcación hacia adelante se produce a causa de una instrucción condicional, es decir, la bifurcación se produce, o no, en función del resultado obtenido de una comparación anterior.

Por ejemplo, las instrucciones comprendidas entre los números 5 y 25, ambas inclusive, se ejecutan si se cumple una determinada condición. Por el contrario, si no se cumple dicha condición, estas instrucciones no se ejecutan y pasa a ejecutarse la instrucción 26 después de haberse ejecutado la instrucción número 4.

En la instrucción 4 se especifica la dirección de la instrucción 26 en la memoria.

1.4.3. Instrucciones de bifurcación hacia atrás

Supongamos el siguiente programa con instrucción de bifurcación hacia atrás:



Una instrucción de bifurcación hacia atrás produce la ejecución de una parte del programa tantas veces como sea necesario hasta que se cumpla una determinada condición.

Cuando se ejecuta la instrucción 18, la dirección de la siguiente instrucción a ejecutar es la dirección de la instrucción 4. Por lo tanto, la instrucción que se ejecuta en cuarto lugar se ejecuta también en décimo noveno y así sucesivamente. Las instrucciones repetidas constituyen un *bucle* y estos métodos se emplean para procesos iterativos.

La instrucción de bifurcación viene condicionada por algún resultado obtenido en el interior del bucle, para que el programa pueda salir del bucle.

1.4.4. Subprogramas

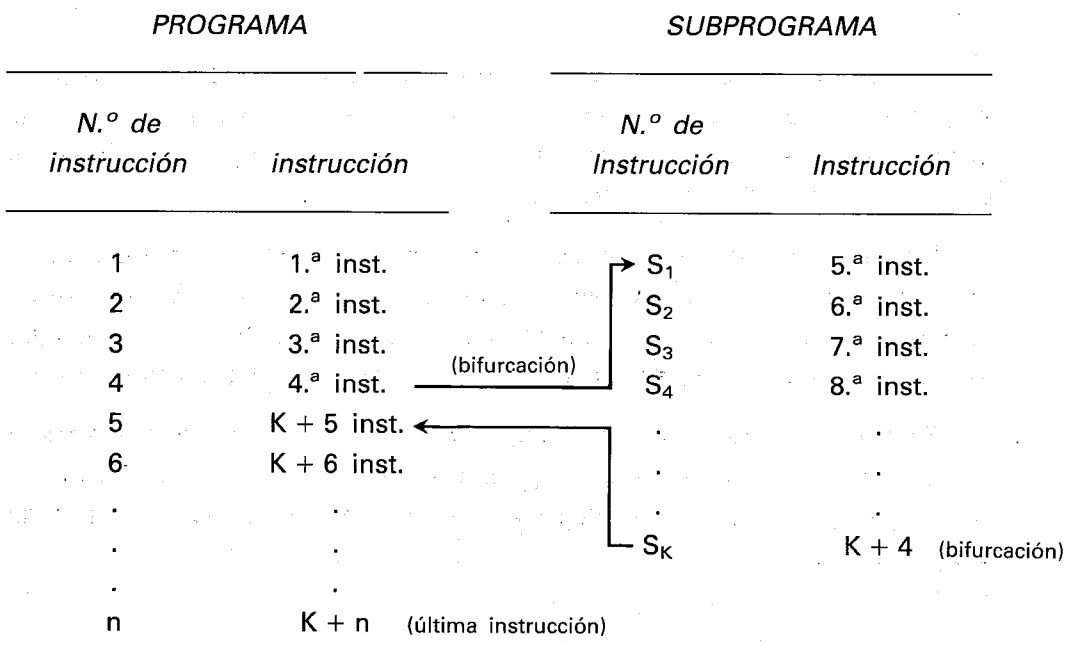
Un *subprograma* (llamado también *subrutina)* es un conjunto de instrucciones diseñado para que lo utilicen otros programas para realizar operaciones determinadas.

Un subprograma es un programa completo e independiente pero que puede ser referenciado por cualquier otro programa *(programa principal).*

Cuando un subprograma debe ser utilizado durante la ejecución del programa, existe una instrucción de bifurcación en un lugar adecuado del programa para bifurcar a la primera instrucción del subprograma.

Una bifurcación de un programa principal transfiere el control del programa al subprograma. Una vez ejecutadas las instrucciones del subprograma se devuelve el control al programa principal. La última instrucción del subprograma debe ser una instrucción de bifurcación al programa principal, concretamente, a la instrucción siguiente de la que originó la bifurcación al subprograma.

Supongamos el siguiente programa con subprograma:



Suponiendo que el subprograma está compuesto por K instrucciones (cuyos números son $S_1, S_2, ..., S_K$), éstas se ejecutan después de la instrucción 4 del programa principal.

Una vez finalizado el proceso del subprograma se devuelve el control al programa principal y se ejecuta la instrucción 5 (cuando el ordenador ejecuta la instrucción 5 habrá ejecutado K + 5 instrucciones).

Un subprograma puede tener varias entradas o salidas de forma que puede devolverse el control al programa principal antes de que se ejecute la última instrucción del subprograma (K + 4 en el ejemplo).

La llamada al subprograma se efectúa mediante una instrucción especial que sirve para conectar con los subprogramas. En la llamada se emplea una palabra clave que representa el nombre del subprograma.

Por ejemplo, supongamos que el ordenador para resolver un problema debe realizar varias raíces cuadradas. Las instrucciones correspondientes aparecerán en diversas ocasiones a lo largo del programa. Para evitar el tener que codificarlas muchas veces se puede utilizar el subprograma llamado SQRT que sirve para hallar la raíz cuadrada de un número. Entonces cada vez que se necesite realizar una raíz cuadrada en el programa principal se efectúa una llamada al subprograma SQRT.

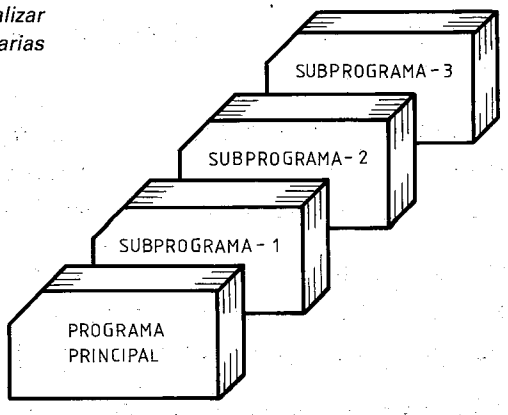
El conjunto de instrucciones de llamada al subprograma se denomina *secuencia de llamada.*

Una secuencia de llamada proporciona datos necesarios para el subprograma, tales como la posición de los datos con los que van a realizarse las operaciones y el lugar del programa principal donde hay que devolver el control de ejecución del programa.

Un subprograma puede almacenarse fuera del programa principal, el cual puede llamar al subprograma tantas veces como sea necesario. De esta forma las instrucciones del subprograma no estarán repetidas en la memoria aunque se utilicen varias veces, ahorrando, de este modo, espacio en la memoria.

El almacenamiento de los subprogramas puede realizarse mediante dispositivos como cinta, tambor o disco magnético, en una biblioteca de subprogramas, en donde pueden estar escritos en diferentes lenguajes de programación.

Fig. 1.4. Cada subprograma puede realizar distintas operaciones, todas ellas necesarias para el programa principal.



Un programa puede disponer de varios subprogramas, de modo que puede ocurrir que el programa principal sólo contenga instrucciones de bifurcación que conecten con los distintos subprogramas.

Un subprograma suele aparecer sólo una vez en cada programa, aunque existe un tipo de subprograma llamado *subrutina abierta,* la cual aparece varias veces dentro de la codificación del programa. Por lo tanto, estas subrutinas se almacenan dentro del programa principal, por lo que no se necesitan instrucciones de llamada.

Una subrutina abierta debe contener pocas instrucciones ya que pueden necesitarse varias copias.

Por ejemplo, si una secuencia determinada de operaciones debe realizarse siete veces a lo largo del programa, el programador sólo la escribirá una vez y se efectuarán seis copias más para introducirlas, en donde sea necesario.

1.4.5. Macroinstrucciones

Otra técnica para simplificar la codificación y evitar repeticiones consiste en utilizar las *macroinstrucciones.*

Una *macroinstrucción* (llamada también *macro* simplemente) es una instrucción de un determinado lenguaje de programación que equivale a una o más instrucciones del mismo lenguaje.

El programa traductor (compilador o ensamblador) convierte una macroinstrucción en varias instrucciones de máquina que son las que realmente se procesan. Es decir, antes de ejecutarse el programa que contiene la macroinstrucción el programa traductor la convierte en varias instrucciones más sencillas.

Las macroinstrucciones permiten al programador ampliar y modificar los lenguajes de programación.

El programador debe definir una macroinstrucción antes de utilizarla en el programa.

1.5. EVALUACIÓN DE LOS RESULTADOS

Después de la ejecución del programa se obtienen los resultados del proceso mediante los dispositivos de salida.

La recogida de datos en un sistema de proceso de datos debe efectuarse mediante su presentación en códigos y formatos correspondientes al lenguaje de programación del ordenador. Una vez obtenidos todos los datos, deben estar disponibles para su utilización en cualquier momento.

La recogida de datos consiste en representar la información producida en forma manuscrita o impresa. Luego se han de seleccionar los datos, eliminando los que no sirvan y deben corregirse los errores que se hayan producido.

Al seguir el proceso de las instrucciones que se le han suministrado, el ordenador produce las respuestas deseadas. Algunas de estas instrucciones dirigen al ordenador a imprimir la información necesaria para comprender la solución del problema.

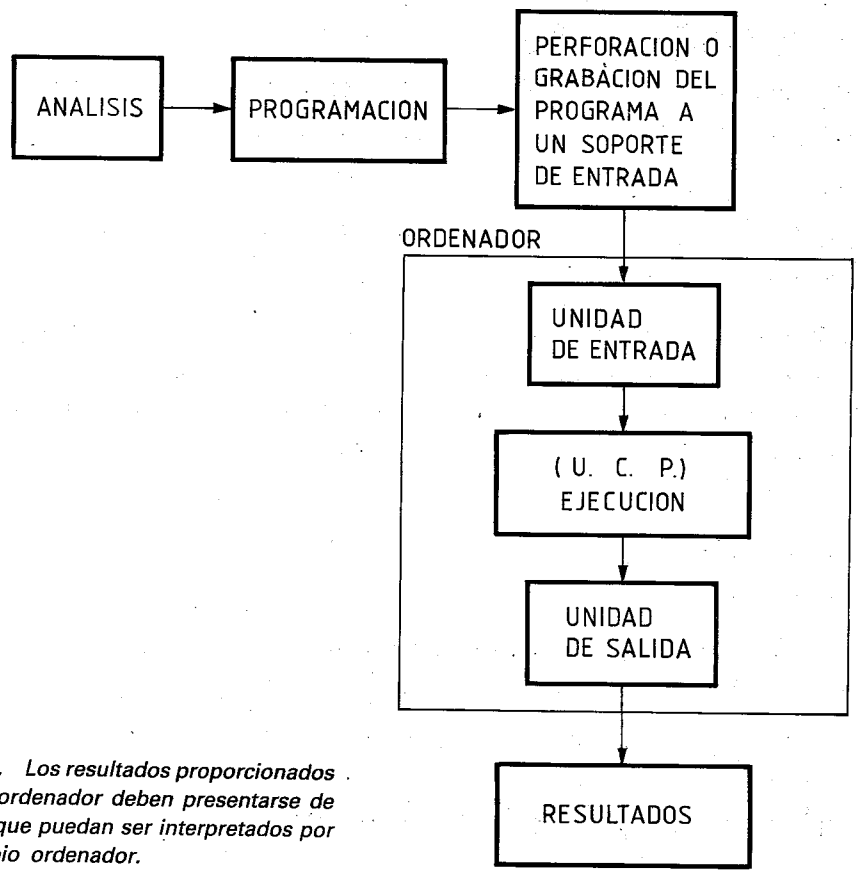


Fig. 1.5. Los resultados proporcionados por el ordenador deben presentarse de forma que puedan ser interpretados por el propio ordenador.

El ordenador prepara los informes correspondientes a cada resultado. Una vez obtenidos estos informes se envían a las personas que tengan que utilizarlos.

Los datos de entrada y salida se almacenan, así como el programa, de forma que puedan ser utilizados en procesos posteriores.

Puede suceder que los resultados obtenidos con el programa no sean satisfactorios, bien por errores en el análisis del problema (por ejemplo: definición incorrecta del problema), o bien por errores de codificación del programa, (errores de perforación de las tarjetas de entrada, análisis incompleto de los resultados necesarios, etc).

Para reducir los posibles errores es necesario tener en cuenta ciertas consideraciones, como por ejemplo evitar la repetición de pasos ya realizados, con lo cual se reducirían, además, los costos.

Una forma de obtener mejores resultados puede ser realizar la planificación comenzando el trabajo partiendo del resultado final, de modo que durante el período de la realización del plan se pueda ir comparando el trabajo con el resultado final. De esta manera puede saberse cuáles son los datos que se necesitan y que no habían sido previstos anteriormente y, por otra parte, pueden eliminarse datos innecesarios.

Deben preveerse todas las situaciones posibles que puedan darse y el modo de resolverlas de la forma más rápida y con el mínimo coste.

Cuando se ejecuta el programa por primera vez es muy frecuente que se produzcan errores no detectados en la fase de depuración, en cuyo caso deben corregirse y procesar de nuevo el programa hasta que los resultados sean correctos.

Una vez obtenidos los resultados, ha de efectuarse una revisión acerca de las posibles modificaciones que pueden realizarse para una mejor solución del problema.

Por ejemplo, pueden aparecer datos nuevos que contribuyan a obtener resultados más exactos, o también, pueden surgir problemas imprevistos que han de ser supervisados por el analista y el programador, para que el plan funcione de la forma deseada.

NOTA:

Al final de cada lección se le plantean unos ejercicios que usted deberá contestar.

La finalidad de estos ejercicios es la de que usted mismo compruebe si ha asimilado bien el contenido de la lección.

Verá que en algunos ejercicios le damos tres respuestas, pero solamente una es la válida, usted deberá señalar cuál de las tres es la correcta.

En otros ejercicios le dejamos unos espacios para que usted escriba la palabra o palabras que completen y hagan verdadera la frase.

Al final del libro le transcribimos las soluciones correctas de todos ellos.

EJERCICIOS

1. El conjunto de operaciones necesarias para resolver un problema en un sistema de proceso de datos recibe el nombre de .....

2. La escritura de un programa en un determinado lenguaje de programación se denomina .....

3. El documento utilizado para que el programador escriba las instrucciones de un programa se llama .....

4. Un programa que traduce el programa fuente a programa objeto recibe el nombre de .....

5. El conjunto de instrucciones de llamada a un subprograma se denomina .....

6. La descripción de los periféricos se realiza en la etapa de:

   a) definición del problema.
   b) análisis del problema.
   c) programación.

7. La documentación de un programa debe ser preparada por el programador de tal forma que pueda ser utilizada por:

   a) cualquier persona interesada.
   b) el programador exclusivamente.
   c) el ordenador exclusivamente.

8. El programador codifica el programa generalmente en un lenguaje que debe ser traducido para que el ordenador lo entienda. Este lenguaje utilizado por el programador se denomina:

   a) lenguaje fuente.
   b) lenguaje objeto.
   c) lenguaje máquina.

9. Un organigrama representa los pasos matemáticos y lógicos para la resolución de un problema y puede ser entendido por:

   a) el ordenador.
   b) el programador.
   c) el ordenador y el programador.

10. La fase de *depuración del programa* consiste en:

    a) codificar un programa en el lenguaje de programación adecuado.
    b) desarrollar un procedimiento de cálculo en la resolución de un problema.
    c) verificar, detectar y suprimir los errores de un programa.

2.

ORGANIGRAMAS

CUADRO RESUMEN DE LA LECCIÓN

- Organigramas
  - Algoritmos
    - Algoritmo no numérico
    - Algoritmo numérico
  - Símbolos de los organigramas
  - Bucles
    - Contadores
    - Bucles anidados
  - Organigrama del sistema
  - Consideraciones sobre los organigramas
  - Ejemplos de organigramas

Un *organigrama,* denominado también *diagrama de flujo* (flow chart), es la representación gráfica de los pasos lógicos y secuenciales que describen las operaciones y decisiones lógicas de un programa. Es decir, un organigrama describe la secuencia de ejecución del programa mediante el ordenador.

Las *secuencias* son los pasos sucesivos que el ordenador debe seguir para llegar a la solución de un problema.

Los ordenadores digitales son máquinas secuenciales, es decir, no realizan más de un paso a la vez. Los problemas que ha de resolver el ordenador deben ser representados como una secuencia de acciones que pueda realizar el ordenador.

2.1. ALGORITMOS

Los procedimientos que se siguen paso a paso (secuencialmente) para determinar la solución de un problema, numérico o no numérico, se denominan *algoritmos.*

Un diagrama de flujo (organigrama) es, por lo tanto, la representación gráfica de un proceso algorítmico orientado al ordenador.

2.1.1. Algoritmo no numérico

Veamos un ejemplo de algoritmo para resolver un problema no numérico:

Un cliente efectúa un pedido a una determinada empresa. La empresa examina la ficha del cliente, si el cliente es buen pagador entonces la empresa

aceptará el pedido, en caso contrario, si no es buen pagador, rechazará el pedido.

El algoritmo es el siguiente:

1. Principio.
2. Leer el pedido.
3. Examinar la ficha del cliente.
4. Si el cliente es buen pagador, aceptar el pedido. En caso contrario, rechazar el pedido.
5. Final.

En el procedimiento para la resolución de un problema se deben tener en cuenta todas las posibles situaciones que se puedan producir.

Como puede comprobarse en el ejemplo, un algoritmo es una lista de instrucciones para realizar, paso por paso, un determinado proceso.

2.1.2. Algoritmo numérico

Los algoritmos que se procesan en un ordenador pueden combinar en un problema de cálculo complicado, millones de operaciones elementales, como por ejemplo sumas, restas, multiplicaciones y divisiones.

Consideremos por ejemplo un algoritmo para saber si un número es primo (un número primo es el que sólo puede dividirse por sí mismo y por la unidad. Por ejemplo: 2, 3, 5, 7, 11, 13, 17, etc. son números primos, y los números 4, 6, 8, 9, 10 etc. no lo son ya que son divisibles por números distintos a ellos mismos y a la unidad. Así, por ejemplo, el 4 es divisible por 2, el 6 por 2 y por 3, el 8 por 2 y por 4, etc.).

Sea *N* un número entero cualquiera, por ejemplo, 14753. Determinar si *N* es un número primo, utilizando para ello la variable *A* como divisor de *N*.

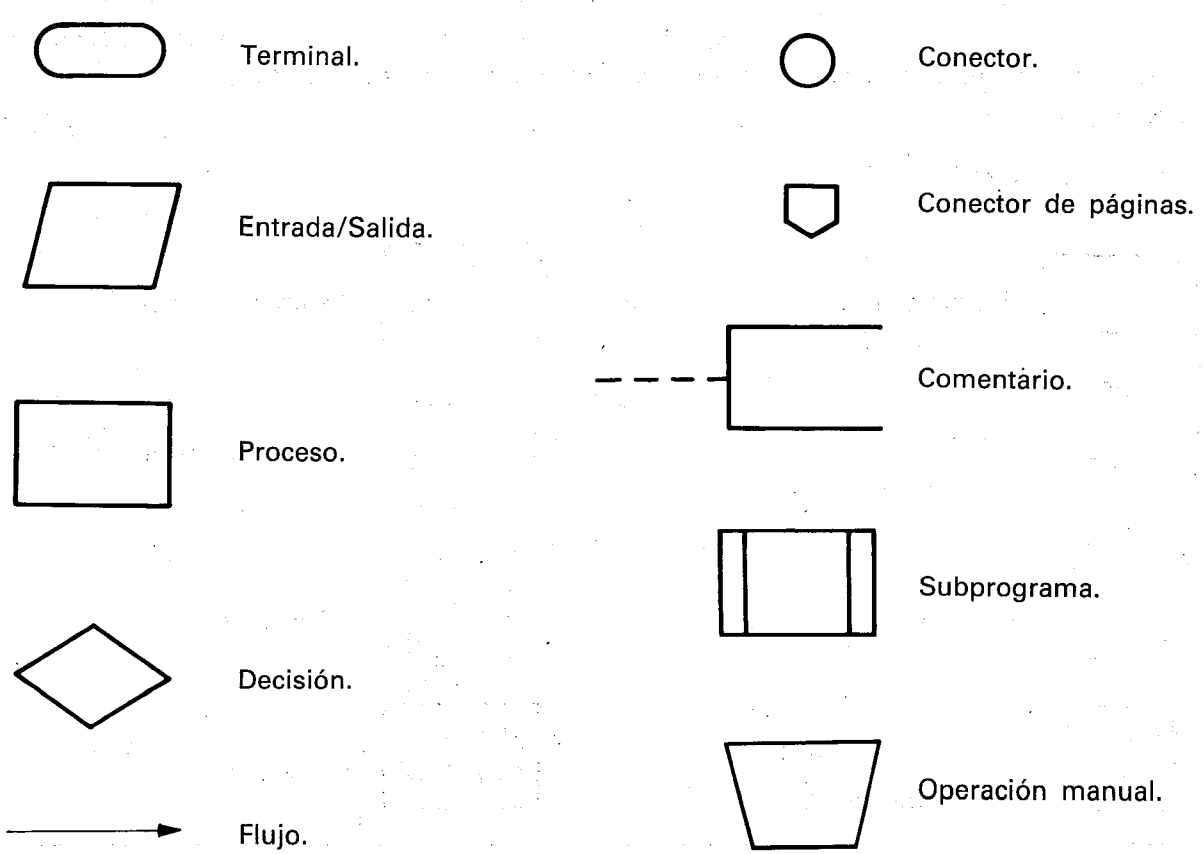
1. Poner *A* igual a 2 (*A*=2).
2. Dividir 14753 por *A* (14753/*A*).
3. Si el resultado es entero, entonces 14753 no es número primo, ir al punto 6. En caso contrario continuar con el punto 4.
4. Sumar 1 a *A* (*A* = *A* + 1).
5. Si *A* es igual a 14753, entonces 14753 es un número primo. En caso contrario, ir al punto 2.
6. Final.

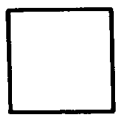
2.2. SÍMBOLOS DE LOS ORGANIGRAMAS

Los procedimientos o instrucciones que constituyen el programa suelen expresarse en forma de *diagramas por bloques* en donde se representa gráficamente la estructura lógica del problema. Se utilizan bloques (figuras geométricas) para representar los procesos enlazados entre sí mediante líneas de transferencia de control que indican el orden en que deben realizarse las operaciones. Cuando el ordenador realiza una operación representada en un determinado bloque se transfiere el control al siguiente bloque para efectuar la operación u operaciones representadas en él.

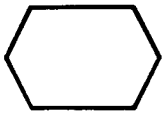
Un organigrama, por lo tanto, está compuesto por una serie de símbolos que representan, cada uno de ellos, una acción (una operación de cálculo, lógica o de E/S) que debe realizar el ordenador.

En la mayoría de los organigramas se usan los siguientes símbolos:

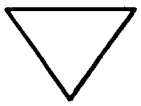




Operación auxiliar.



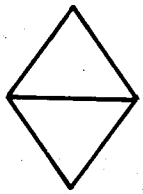
Modificación de programa.



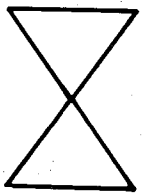
Fusión.



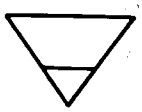
Extracción.



Clasificación.



Intercalación.



Almacenamiento off-line.



Operación por teclado.

Además se utiliza otro tipo de símbolos para operaciones de entrada y salida específicas:

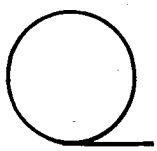
Símbolos de entrada/salida:



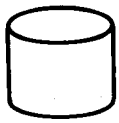
Tarjeta perforada.



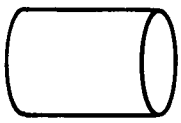
Cinta perforada.



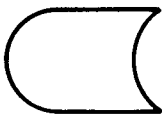
Cinta magnética.



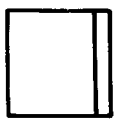
Disco magnético.



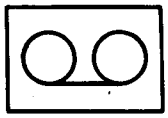
Tambor magnético.



Almacenamiento on-line.

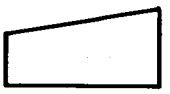


Ficha magnética.



Cassettes.

Símbolos de entrada:

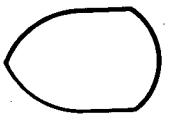


Entrada manual.

Símbolos de salida:



Documento.

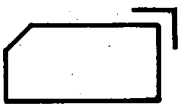


Representación visual.

Existen otros símbolos, como por ejemplo los referentes a lotes o ficheros de tarjetas perforadas:



Lote de fichas.



Fichero de fichas.

Terminal

El símbolo *terminal* representa un punto terminal en un sistema en donde los datos pueden entrar o salir.

Los terminales se utilizan para iniciar, parar, interrumpir o finalizar un programa o un subprograma.

El símbolo de comienzo de un programa indica que algunas o todas las instrucciones del programa se encuentran en la memoria del ordenador y que va a tomarse y ejecutarse la primera instrucción del programa, (Fig. 2.1).

El inicio de un subprograma debe indicarse con el símbolo «entrada al subprograma S» indicando S el nombre del subprograma, (Fig. 2.1).

En estos símbolos, en lugar de «inicio» suele ponerse también la palabra inglesa START.

Los símbolos de parada (STOP) y final (END) de un programa representan la ejecución de una instrucción de «parada» (Fig. 2.2), bien porque se encuentre una detención incondicional (STOP) o porque haya finalizado el proceso bajo el control del programa (END), (Fig. 2.2).

El símbolo «salida del subprograma S» indica el final del proceso de un subprograma.

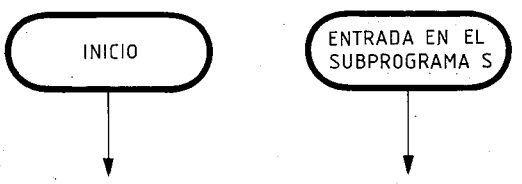


Fig. 2.1. Con la palabra «inicio» se indica el comienzo del organigrama (del programa principal). Cuando es el comienzo de un subprograma debe especificarse en el símbolo terminal *(«entrada al subprograma S»).*

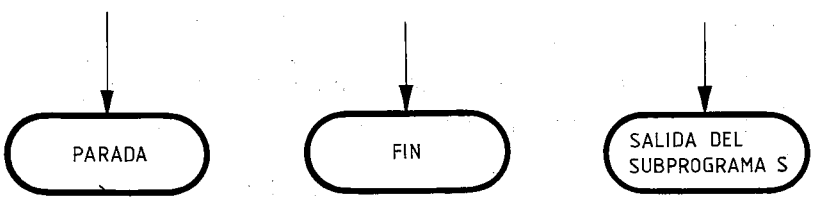


Fig. 2.2. El símbolo terminal *se utiliza, además para indicar una interrupción («parada») o el final del proceso («fin» y «salida del subprograma S»).*

Entrada/Salida

Este símbolo representa el movimiento de datos entre la memoria y un medio de almacenamiento externo. Es decir, la introducción de los datos a la memoria para su proceso (entrada), o el registro (perforación, gravación o impresión) de los datos procesados (salida).

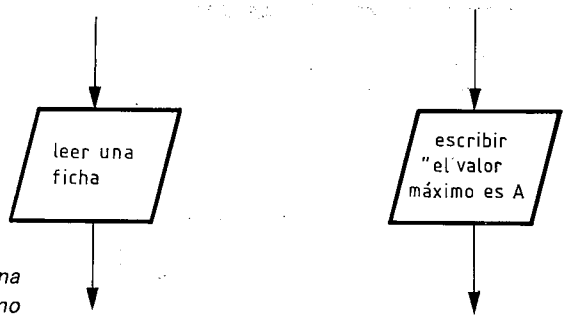


Fig. 2.3. Ejemplos de entrada (leer una ficha) y de salida (escribir «el valor máximo es A»).

Proceso

Representa la función de proceso, es decir, la ejecución de cierta operación o grupo de operaciones. El resultado de dichas operaciones puede ser el cambio de valor o de posición de la información.

Los movimientos de los datos de una zona (o campo) de la memoria a otra zona, o desde la memoria al área de salida, pueden expresarse utilizando una flecha (A → B) que indica el movimiento del contenido de un campo de memoria (A) a otro campo (B). Con lo cual el contenido de los dos campos de memoria (A y B) es el mismo. En este texto se utiliza en lugar de la flecha el signo igual (A = B).

Las operaciones aritméticas pueden expresarse de diversas formas. Por ejemplo, sean los campos de memoria A y B. El resultado de la suma del contenido de ambos campos puede situarse en otra zona de la memoria denominada campo C (fig. 2.5) o bien a uno de los campos A (fig. 2.6) o B (fig. 2.7).

Fig. 2.4.

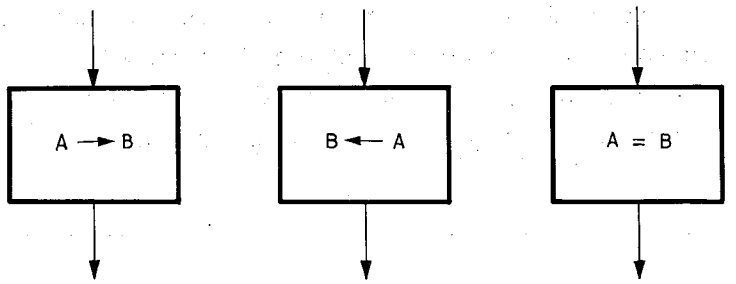


Fig. 2.5.

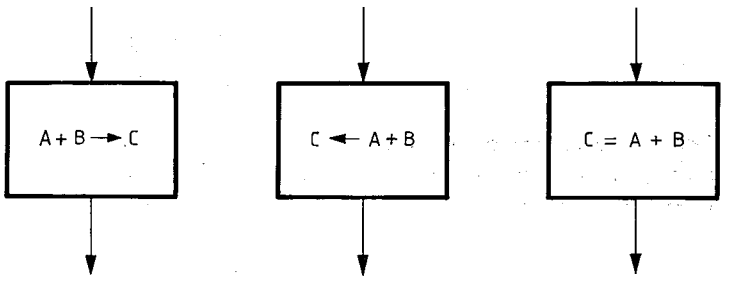


Fig. 2.6.

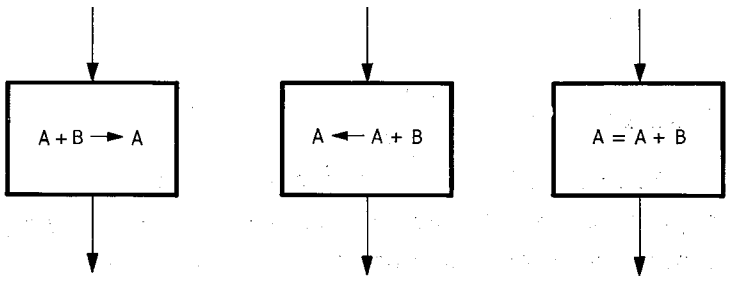
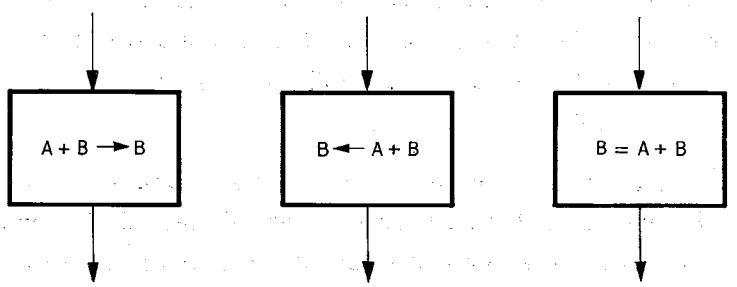


Fig. 2.7.



Bloques de decisión

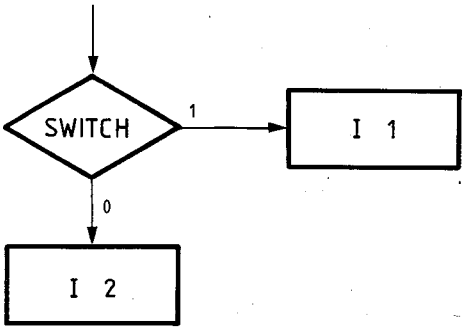
Frecuentemente, es necesario interrumpir la secuencia de un programa y saltar o bifurcar a otra parte del programa si se cumple una determinada condición.

El bloque de decisión representa una operación a partir de la cual se determina el camino a seguir entre un cierto número de alternativas dependiendo de ciertas condiciones.

Por ejemplo (Fig. 2.8), si el estado de un interruptor (switch) es 1 (conectado) el programa bifurca a la instrucción I1, y si, por el contrario, se cumple la condición SWITCH = 0 (desconectado) el programa bifurca a la instrucción I2.

El ordenador puede comparar dos valores para decidir si ha de bifurcar a un sitio u otro del programa según el resultado de dicha comparación.

Por ejemplo, sean los valores A y B. si se cumple la condición A = B, entonces la siguiente instrucción a ejecutar es I1. En caso contrario, es decir, si A ≠ B se ejecuta la instrucción I2 (Fig. 2.9).



◀ *Fig. 2.8. Si SWITCH = 1 se ejecuta la instrucción I1. En caso contrario, si SWITCH = 0, se ejecuta I2.*

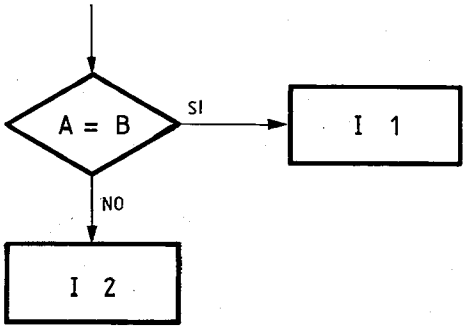


Fig. 2.9. Si se cumple la condición A = B entonces se ejecuta I1. Si no se cumple dicha condición, se ejecuta I2. ▶

Las comparaciones posibles entre A y B son las siguientes:

1.er Valor	Operador	2.º Valor	Comparación	Significado
A	=	B	A = B	A es igual a B.
A	≠	B	A ≠ B	A no es igual a B.
A	<	B	A < B	A es menor que B.
A	>	B	A > B	A es mayor que B.
A	≤	B	A ≤ B	A es menor o igual a B.
A	≥	B	A ≥ B	A es mayor o igual a B.

Estas comparaciones se expresan dentro de un símbolo de decisión, en un organigrama, de la forma que muestran las figuras siguientes, y en las que, el «SI» puede sustituirse por «VERDADERO» y el «NO» por «FALSO».

Fig. 2.10.

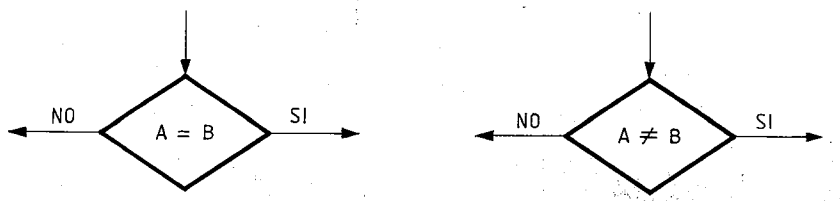


Fig. 2.11.

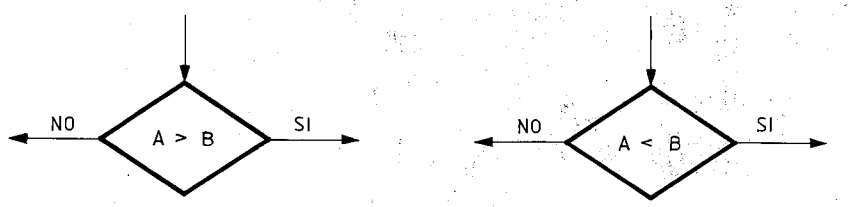
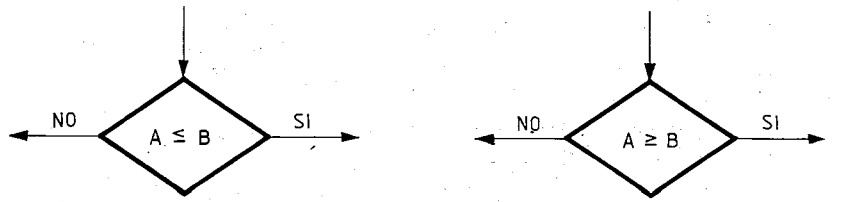


Fig. 2.12.



También se puede representar la comparación de dos valores utilizando otro operador (::).

Comparación	*Significado*	*Resultados posibles*
A :: B	A comparado con B	A = B, A < B, A> B

La figura 2.13 muestra cómo se expresarían estas comparaciones dentro del organigrama.

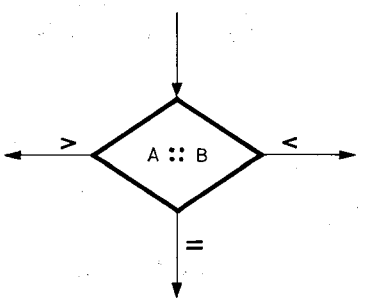


Fig. 2.13. Representación en el organigrama del operador de comparación.

Flujo

Representa la dirección del flujo o secuencia de las operaciones y las informaciones que van a ser procesadas.

La dirección del flujo se representa por medio de flechas dibujadas entre los símbolos del organigrama. Generalmente el sentido de la flecha es de arriba hacia abajo y de izquierda a derecha, aunque en muchas ocasiones es necesario dibujarlas de derecha a izquierda y de abajo hacia arriba (como por ejemplo cuando se representa una bifurcación hacia atrás).

Conectores

Cuando las líneas de flujo se interrumpan a causa de la terminación de la página o por limitaciones de otro tipo, bien sea por necesidad o para presentar

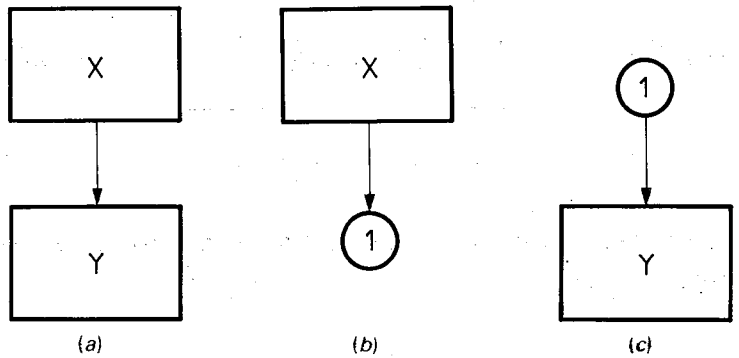


Fig. 2.14. Utilización del conector. La parte (a) de la figura representa la conexión del símbolo X con el símbolo Y mediante una línea de flujo. Las partes (b) y (c) de la figura muestran el mismo esquema con la utilización de conectores.

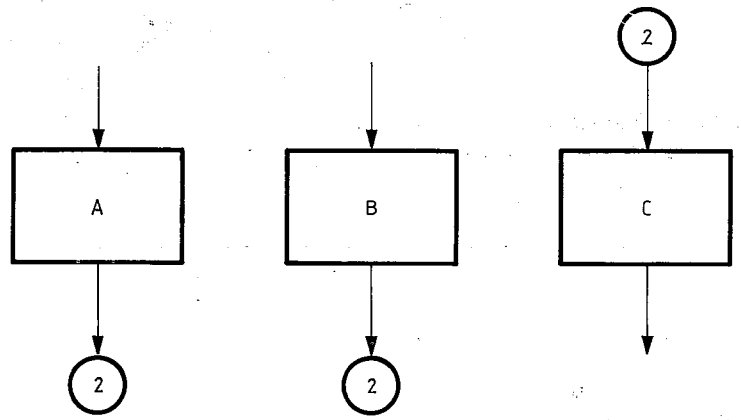


Fig. 2.15. Si A y B conectan con C deben utilizar el mismo símbolo conector.

mayor claridad, se utilizan los símbolos conectores que indican la interrupción y el salto de un punto a otro del organigrama.

Los conectores enlazan dos puntos de un programa sin tener que trazar entre ellos las líneas que los unen.

Si distintos puntos del programa conectan con el mismo punto todos ellos deben utilizar un conector idéntico.

Conector de páginas

Cuando debe conectarse con un punto del organigrama que se encuentra en otra página por falta de espacio en la anterior se utiliza el *conector de páginas.*

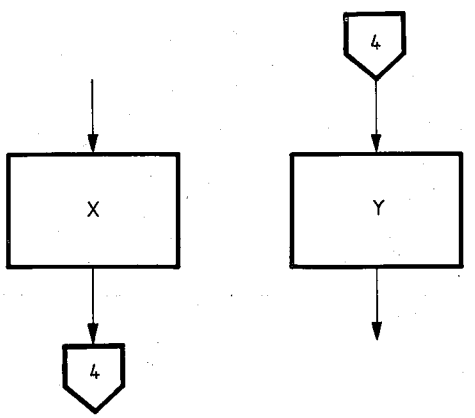


Fig. 2.16. El símbolo Y, que va a continuación de X en el organigrama, se encuentra representado en una página distinta al símbolo X.

Comentario

Este símbolo se utiliza para hacer algún tipo de anotación o aclaración. La línea de puntos une el comentario con los símbolos en donde quiere efectuarse este comentario.

Subprograma

Representa un grupo de operaciones a ejecutar y que están representadas en otro lugar.

Operación manual

Representa a los procesos *off-line* (fuera de línea), realizados o dirigidos manualmente por el operador.

Operación auxiliar

Representa una operación *off-line* realizada en equipos que se encuentran fuera del control directo de la unidad central de proceso.

Modificación de programa

Representa la modificación de instrucciones de los programas.

Fusión

Combinación de dos o más conjuntos de datos en uno sólo.

Extracción

Extracción de uno o más conjuntos de datos a partir de otro conjunto de datos.

Clasificación

El símbolo de clasificación (SORT) se utiliza para indicar la ordenación de un conjunto de datos basándose en ciertos criterios de clasificación.

Intercalación

El símbolo de intercalación (MERGE) se utiliza para designar una operación de fusión con extracción, es decir, combinación de dos o más subconjuntos de datos partiendo de dos o más conjuntos diferentes.

Almacenamiento off-line

Representa el almacenamiento off-line de la información sin tener en cuenta el medio (soporte) en donde se almacena.

Operación por teclado

Representa cualquier operación en la que se utiliza un dispositivo con teclado, como por ejemplo, perforación, verificación, mecanografiado, etc.

Tarjeta perforada

Representa una operación de entrada/salida en la que el soporte de información es cualquier tipo de ficha perforada.

Cinta perforada

Representa una operación de entrada/salida en la que el soporte es la cinta de papel perforada.

Cinta magnética

Representa una operación de entrada/salida cuyo soporte es la cinta magnética.

Disco magnético

Operación de entrada/salida cuyo soporte es el disco magnético.

Tambor magnético

Operación de entrada/salida cuyo soporte es el tambor magnético.

Almacenamiento on-line

Representa una operación de entrada/salida en la que se utiliza cualquier tipo de almacenamiento *on-line* (en línea), cómo por ejemplo: cinta, disco o tambor magnético.

Ficha magnética

Operación de entrada/salida cuyo soporte es la ficha de banda magnética.

Cassettes

Representa una operación de E/S que utiliza cassettes como soporte de información.

Entrada manual

Representa una operación de E/S en la que la información se introduce manualmente durante el proceso mediante teclados on-line, pulsadores, etc.

Documento

Representa una operación de E/S cuyo soporte es un documento. Se utiliza para la salida de información por cualquier dispositivo impresor.

Representación visual

Representación de la información visualmente, en el momento del proceso, por medio de indicadores on-line, dispositivos visualizadores, impresoras de consola, etc.

Lote de fichas

Colección de tarjetas perforadas.

Fichero de fichas

Conjunto de registros en tarjetas perforadas relacionadas entre sí (fichero).

2.3. BUCLES

Cuando la ejecución de una parte del organigrama se repite hasta que se cumple una determinada condición se dice que esa parte del organigrama comprende un *bucle.*

Un bucle consta de tres partes:

a) decisión.
b) sección de proceso.
c) salida del bucle.

El bucle comienza a partir de una decisión. Hasta que no se cumple una cierta condición se continúan haciendo las operaciones incluidas en el bucle, por lo que la salida del bucle también parte del mismo bloque de decisión.

Ejemplo:

Supongamos que se dispone de 5 tarjetas que contienen un número cada una (A, B, C, D y E). Calcular la suma de estos números utilizando para ello, un registro de la memoria llamado SUMA. Imprimir el resultado.

En primer lugar se pone el registro SUMA a cero. Después se lee la primera tarjeta. A continuación se suma el valor de la tarjeta (A) al contenido del registro SUMA:

$$SUMA = SUMA + A$$

o también se puede escribir:

$$SUMA \leftarrow SUMA + A$$

Estas dos notaciones significan lo mismo: «Sumar A a SUMA». El valor de SUMA era cero, y ahora, después de la operación, es SUMA + A, es decir, A.

$$SUMA = SUMA + A = 0 + A = A$$

La siguiente operación es leer la segunda tarjeta y después se suma el valor de la tarjeta (B) al contenido del registro SUMA (en estos momentos es A):

$$SUMA = SUMA + B = A + B$$

Ahora el contenido de SUMA es A + B. Después de sumar el valor de la última tarjeta quedará:

$$SUMA = A + B + C + D + E$$

La última operación será imprimir SUMA.

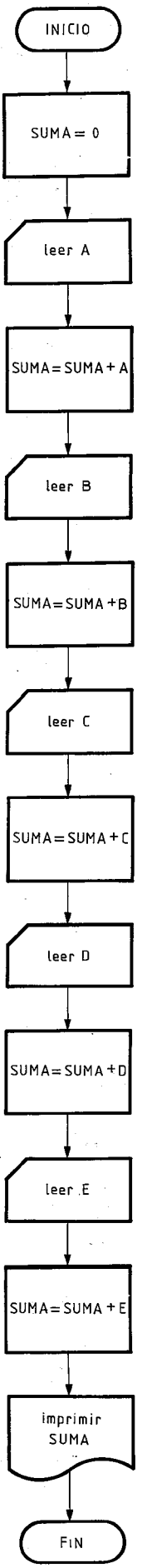


Fig. 2.17. ▶

En el proceso descrito se repiten varias veces las mismas operaciones: «leer una tarjeta» y «sumar el valor de la tarjeta a SUMA».

Por lo tanto, estas operaciones se podrían hacer utilizando un bucle:

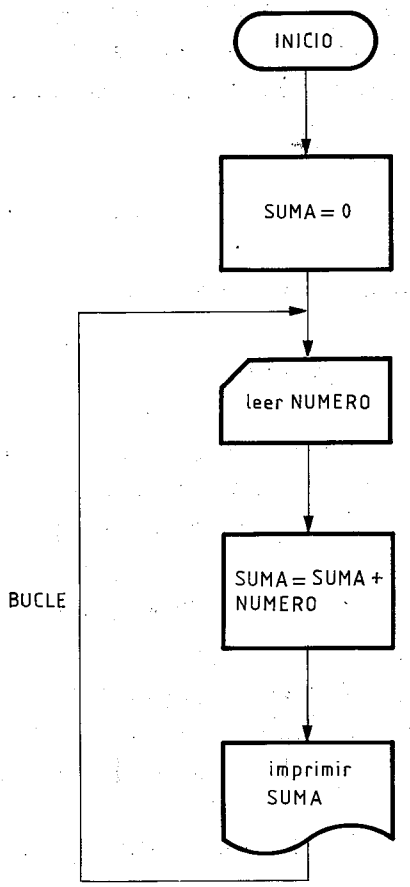


Fig. 2.18. Organigrama incompleto. Las operaciones incluidas en el bucle se efectúan un número indefinido de veces.

Al valor de cada tarjeta le llamaremos NUMERO. De manera que cuando se efectúa la primera lectura, NUMERO será igual a A; en la segunda lectura, NUMERO = B, y así sucesivamente.

Según este organigrama no sólo se imprime la suma total de los 5 números sino que se van imprimiendo las sumas parciales (A, A + B, A + B + C, A + B + C + D y A + B + C + D +E).

Una consideración importante al organigrama de la figura 2.18, es la de que no aparece el bloque de FIN.

Entonces el ordenador seguiría haciendo las operaciones mientras quedaran datos disponibles.

Por lo tanto este bucle debe tener una salida. Por ejemplo, supongamos que los 5 números son mayor que cero, entonces se puede añadir al final, una ficha con el valor cero, de forma que cuando el número leído sea cero se transfiera el control fuera del bucle. Este sería el *punto de salida* del bucle.

De este modo, además de obtener una salida del bucle, se imprime solamente la suma total de los cinco números que es lo que se pedía. Obteniéndose, además, el final del proceso (Fig. 2.19).

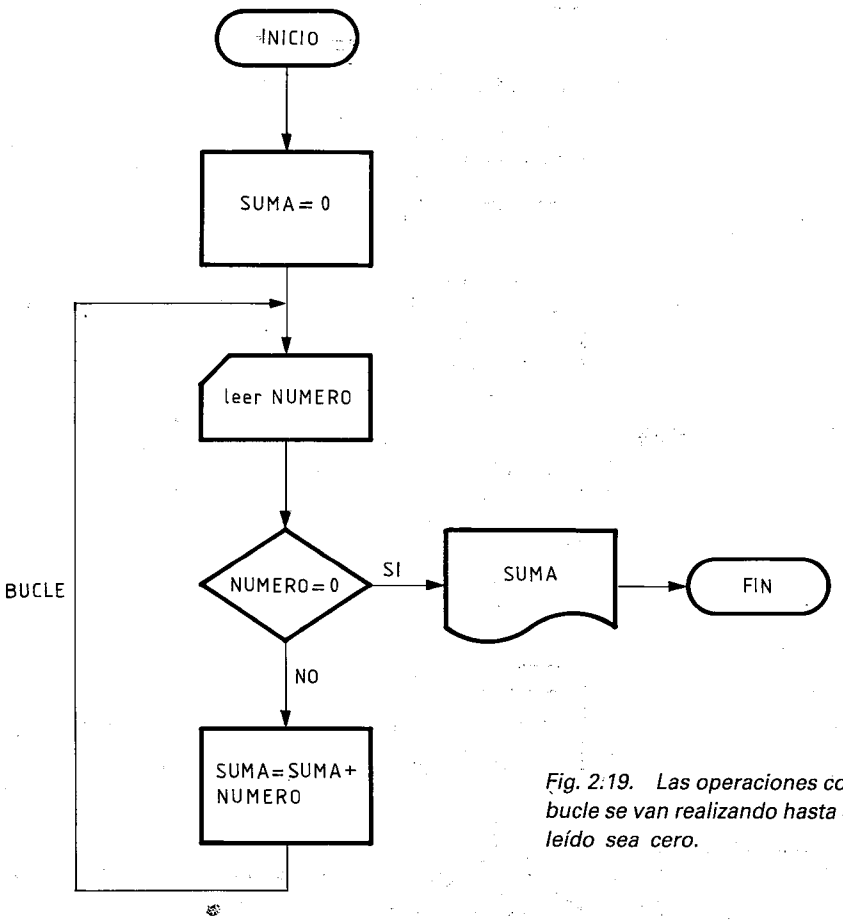


Fig. 2.19. Las operaciones contenidas en el bucle se van realizando hasta que el número leído sea cero.

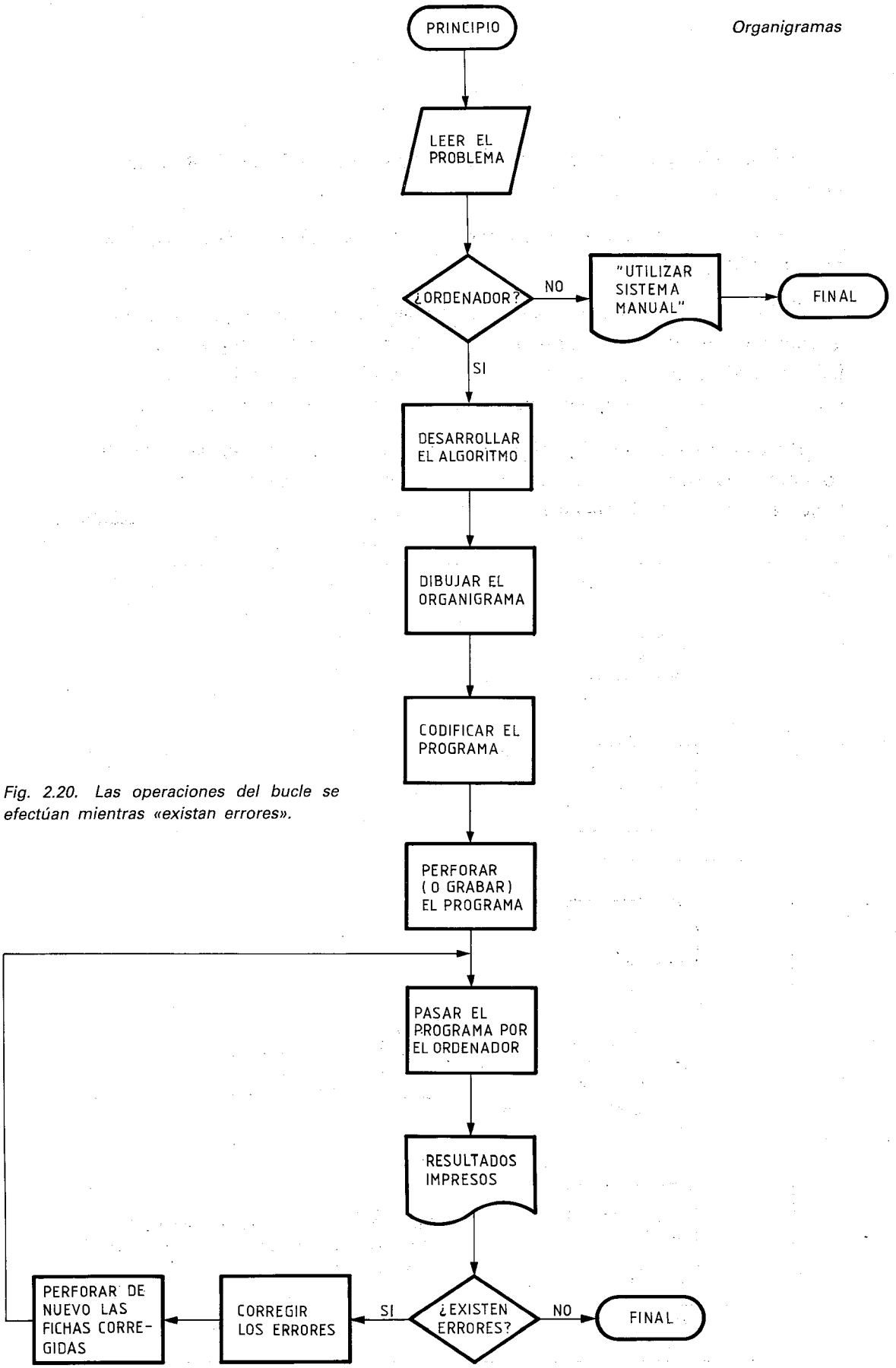


Fig. 2.20. Las operaciones del bucle se efectúan mientras «existan errores».

Ejemplos.

Supongamos el organigrama con los pasos a seguir para la resolución de un problema. (Véase dicho organigrama en la página anterior, figura 2.20).

El bucle comienza con la respuesta «SI» a la pregunta «¿existen errores?». En este caso, se corrigen los errores, se perforan las correcciones, se pasa el programa por el ordenador y se imprimen los resultados, entonces, si vuelven a aparecer errores, se repiten las mismas operaciones. Cuando se cumpla la condición de que no existen errores se obtendrá la salida del bucle.

Veamos ahora otro ejemplo: construir el organigrama que represente la suma S, de los cuadrados de los 100 primeros números naturales. Siendo I una variable que representa uno a uno a los 100 números. Los sucesivos resultados de I x I, es decir, los cuadrados de I, se pondrán en la variable I2.

En este organigrama habría que representar hasta cien veces las operaciones que contienen algunos de los símbolos del organigrama de no ser por la utilización del bucle.

Para evitar estas repeticiones se utiliza el bucle, mediante el cual se hace una *iteración* (se bifurca) hacia la primera operación que debe repetirse. En este ejemplo se harán 99 iteraciones o bifurcaciones hasta que se cumpla la condición y se salga del bucle. La figura 2.21 muestra cómo sería el organigrama.

Las variables I, I2 y S van tomando, respectivamente, los siguientes valores:

$$I = 0, 1, 2, 3, 4, 5, \ldots$$

$$I2 = 0, 1, 4, 9, 16, 25, \ldots$$

$$S = 0, 1, 5, 14, 30, 55, \ldots$$

La operación «poner S a cero» se puede escribir de otras formas más adecuadas en un organigrama:

$$S = 0$$

o también

$$S \leftarrow 0$$

Con esta notación, el organigrama sería el que presenta la figura 2.22.

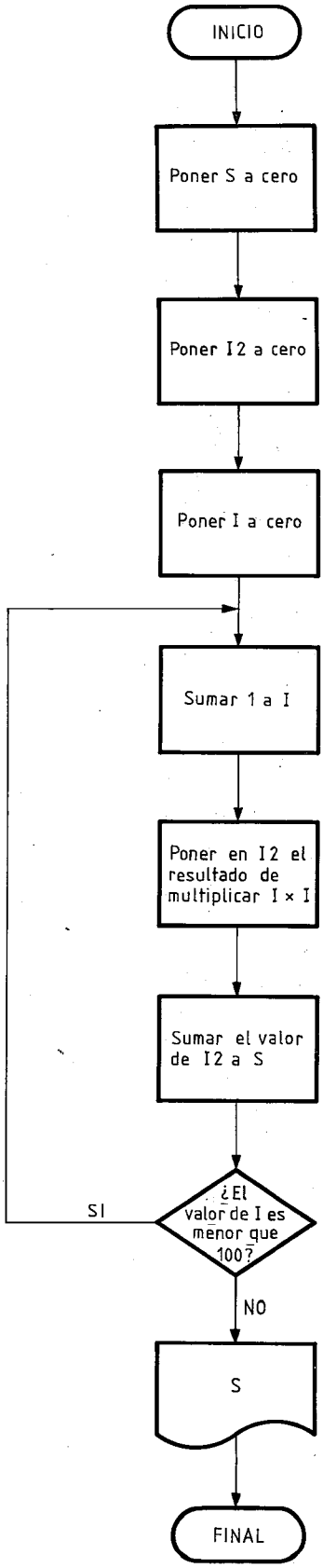


Fig. 2.21. Organigrama que calcula la suma de los cuadrados de los 100 primeros números naturales.

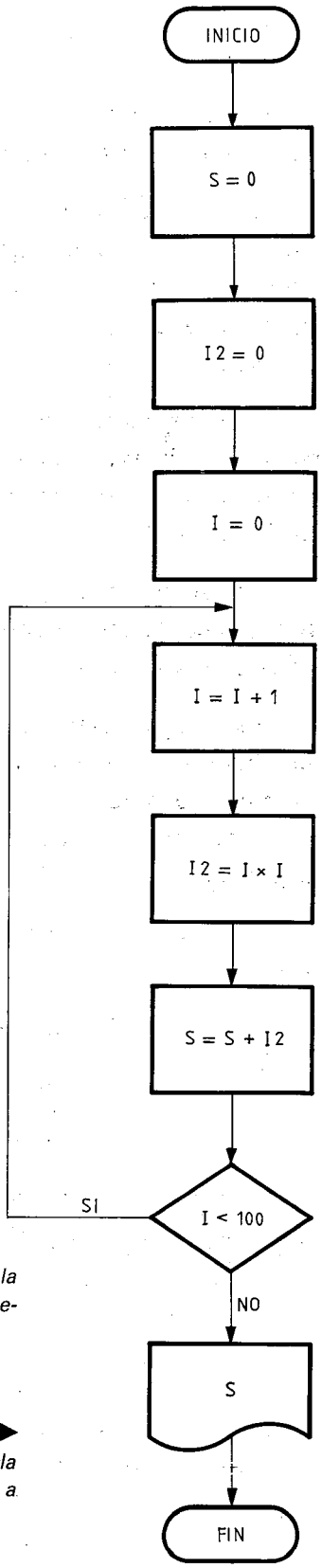


Fig. 2.22. El mismo organigrama de la figura 2.21 con la notación que se usará a lo largo del texto.

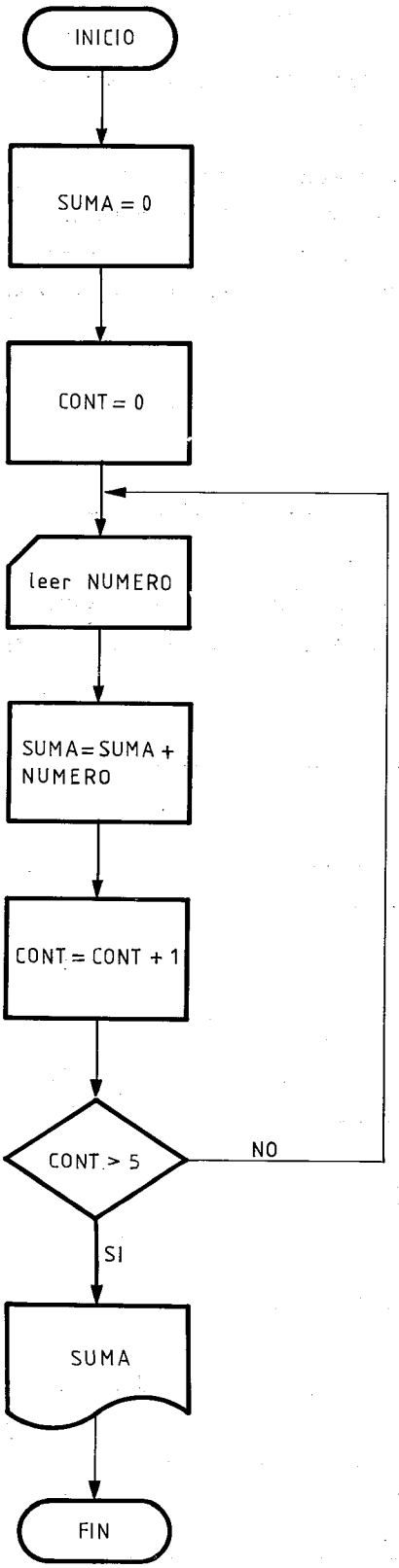


Fig. 2.23. Utilización de un contador (CONT) en un organigrama para sumar cinco números.

2.3.1. Contadores

Los contadores son registros que calculan el número de veces que se repiten las operaciones en un bucle.

En el organigrama del ejemplo presentado en la figura 2.19, para salir del bucle tuvo que añadirse una tarjeta al final con el valor cero. Otra forma de salir del bucle puede ser con la utilización de un contador que vaya «contando» las tarjetas leídas y cuando se lea la última se pueda devolver el control fuera del bucle (Fig. 2.23).

2.3.2. Bucles anidados

En un programa pueden haber varios bucles. Estos pueden estar anidados o independientes.

Los bucles anidados deben estar dispuestos de manera que un bucle contenga a uno o más bucles, o esté contenido totalmente en otro bucle.

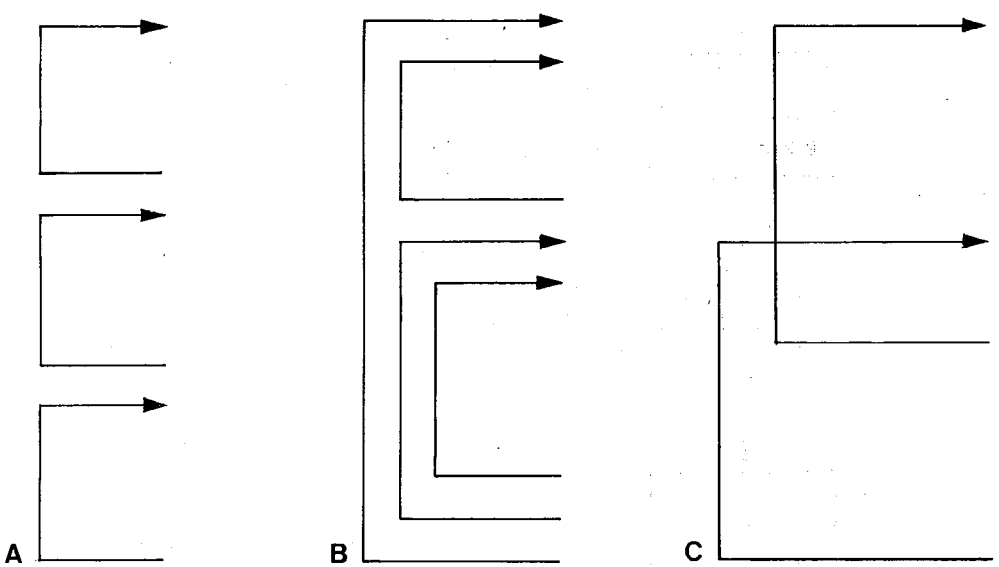


Fig. 2.24. A) *Bucles independientes.* B) *Bucles anidados correctamente.* C) *Bucles anidados incorrectamente.*

2.4. ORGANIGRAMAS DEL SISTEMA

Los organigramas pueden establecerse a distintos niveles de detalle, desde especificar cada una de las operaciones que debe seguir el ordenador, hasta realizar un esquema de las operaciones principales del proceso sin detallar las operaciones individuales.

En el diseño general de un sistema el organigrama debe representar las operaciones principales que componen el sistema.

Por ejemplo, supongamos que una determinada empresa debe poner al día (actualizar) el fichero de artículos *(fichero maestro)* mediante las transacciones o movimientos que se han producido durante un mes *(fichero movimiento)*. Es decir, el fichero movimiento contendrá todas las entradas (compras) y salidas (ventas) de artículos a lo largo del mes.

Mediante un programa de *actualización* el ordenador transformará el *fichero maestro* en un nuevo fichero con las modificaciones que se han producido *(fichero maestro actualizado)*.

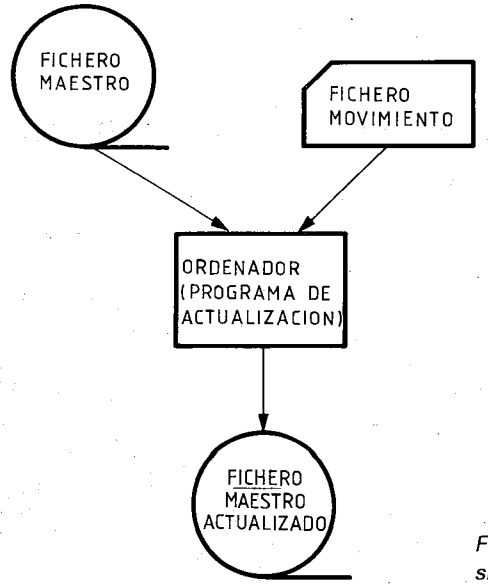


Fig. 2.25. Ejemplo de organigrama del sistema.

En la figura 2.25 puede verse la dirección principal del flujo, que se representa por flechas que entran y salen del símbolo.

En este ejemplo, el organigrama del sistema sólo tiene un programa (programa de actualización). En otros casos puede contener varios programas.

El organigrama del sistema representa la secuencia de operaciones realizadas con los datos a través del sistema.

Si se trata de un gran sistema en el que se van a realizar varias operaciones o grupos de operaciones, de forma que cada grupo resuelva una parte del problema, puede ocurrir que en un solo organigrama resulte imposible representar todas las operaciones por lo que pueden efectuarse diversos organigramas para los distintos grupos de operaciones que componen el sistema.

Asimismo, cada uno de estos organigramas puede corresponder a un determinado programa.

En los organigramas del sistema se representan los datos y los medios de entrada y salida para cada proceso.

Los organigramas del sistema son de gran ayuda para la planificación de un sistema de proceso de datos y para su revisión cuando sea necesario efectuar algún cambio.

2.5. CONSIDERACIONES SOBRE LOS ORGANIGRAMAS

Existen diversos motivos por los que debe construirse el organigrama antes de la codificación de un programa en un lenguaje determinado:

- El organigrama representa los pasos lógicos sucesivos que ha de dar el ordenador para llegar a la solución del problema.
- El organigrama es el paso intermedio entre el planteamiento inicial del problema y la codificación del programa y ofrece una clara definición del problema.
- El programador (persona que realiza el programa) puede determinar mediante el organigrama, si se han tenido en cuenta todas las posibilidades.
- Cuando el programador codifica el programa utiliza como guía el organigrama, que le ofrece la secuencia exacta de operaciones. En

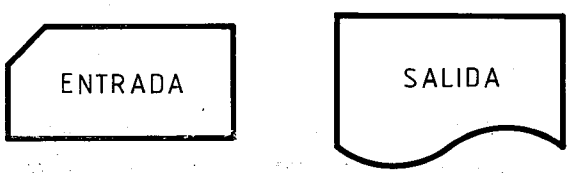
problemas algo complicados resultaría muy difícil la programación sin la ayuda del organigrama.

- Existen diversos caminos para llegar a la solución de un problema. El programador puede determinar mediante el organigrama cuál es el mejor de ellos antes de codificar el programa.
- En la documentación de un programa suele incluirse algún organigrama. Cuando ha de modificarse un programa puede hacerse más fácilmente viendo el organigrama antes de ver el programa codificado.

Cuando se traza un organigrama es conveniente seguir ciertas normas:

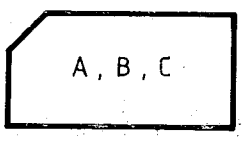
- Usar líneas rectas verticales y horizontales.
- Evitar que se crucen las líneas, utilizando los conectores si es necesario.
- Considerar todas las posibilidades.

Como símbolo de entrada emplearemos, generalmente, en los ejemplos de organigramas, el símbolo de la tarjeta perforada y para la salida el símbolo de salida por impresora.

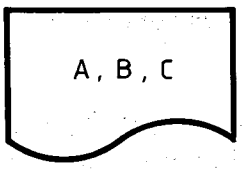


Estos símbolos se emplearán al dibujar un organigrama siempre que no se especifique que la entrada o salida corresponde a otro dispositivo determinado (discos magnéticos, cintas magnéticas, pantallas, etc.).

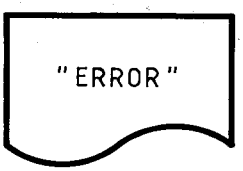
Los nombres de las variables que han de leerse deben aparecer en los símbolos de entrada. Si hay más de una variable se separan por comas, por ejemplo las variables A, B y C deben aparecer como muestra la figura:



Del mismo modo, si se quieren escribir una o más variables deben escribirse en el interior del símbolo de salida.



Para representar otro tipo de información de salida que no sea una variable definida en el proceso se representa entre comillas en el interior del símbolo de salida. Por ejemplo, si queremos que se imprima la palabra «ERROR» cuando se produce un determinado error en el proceso, en el organigrama se representa como muestra la figura:



Otras consideraciones a tener en cuenta al dibujar el organigrama y escribir el programa es la forma en que se deben representar las expresiones aritméticas y el orden de prioridad que deben tener las operaciones en una expresión aritmética.

Por ejemplo, sea la siguiente expresión aritmética:

$$A + B * C/D - ((E + F) * * G) + H$$

El ordenador debe interpretar esta expresión siguiendo un orden de prioridad en las operaciones. El orden de prioridad para cualquier expresión aritmética es el siguiente:

1.°) Efectuar las operaciones que están entre paréntesis.
2.°) Operaciones de exponenciación (* *).
3.°) Multiplicación (*) y división (/).
4.°) Suma (+) y resta (−).

Por lo tanto, para calcular la expresión anterior se debe seguir este orden de prioridad.

En primer lugar se efectúan las operaciones dentro del paréntesis, es decir,

$$((E + F) \ast\ast G)$$

Si hay paréntesis dentro de otros paréntesis, se hacen primero las operaciones de los paréntesis interiores, es decir,

$$(E + F)$$

La siguiente operación es elevar el resultado de la suma E + F a la potencia G.

$$(E + F) \ast\ast G$$

Ahora ya no quedan operaciones dentro algún paréntesis. La operación a realizar a continuación es

$$B \ast C \text{ o } C/D$$

ya que la multiplicación y la división tienen el mismo orden de prioridad. En estos casos se sigue la norma de efectuar las operaciones de izquierda a derecha, es decir, para calcular B × C/D, en primer lugar se hace la operación de la izquierda B × C y el resultado se divide por D.

$$B \ast C/D = \frac{B \ast C}{D}$$

en el caso de que fuera la expresión:

$$B/C \ast D = \frac{B}{C} \ast D$$

Volviendo a la expresión anterior, una vez realizadas las operaciones de los paréntesis, exponenciaciones, multiplicaciones y divisiones, se efectúan las sumas y restas sin que el orden de éstas altere el resultado.

2.6. EJEMPLOS DE ORGANIGRAMAS

Ejemplo 1

Consideremos el algoritmo visto anteriormente (algoritmo no numérico). Construir el organigrama teniendo en cuenta que se efectúan 50 pedidos ordenados por NUMERO-DE-PEDIDO (NP) del 1 al 50. Con este supuesto el algoritmo quedará como sigue:

1. Principio
2. Leer el pedido.
3. Si NUMERO-DE-PEDIDO es mayor que 50 ir a punto 9.
   En caso contrario continuar con el punto 6.
4. Examinar la ficha del cliente.
5. Si el cliente no es buen pagador ir al punto 8.
   En caso contrario continuar con el punto 6.
6. Servir el pedido.
7. Ir al punto 2
8. Rechazar el pedido.
9. Final.

El organigrama correspondiente es el que presenta la figura 2.26.

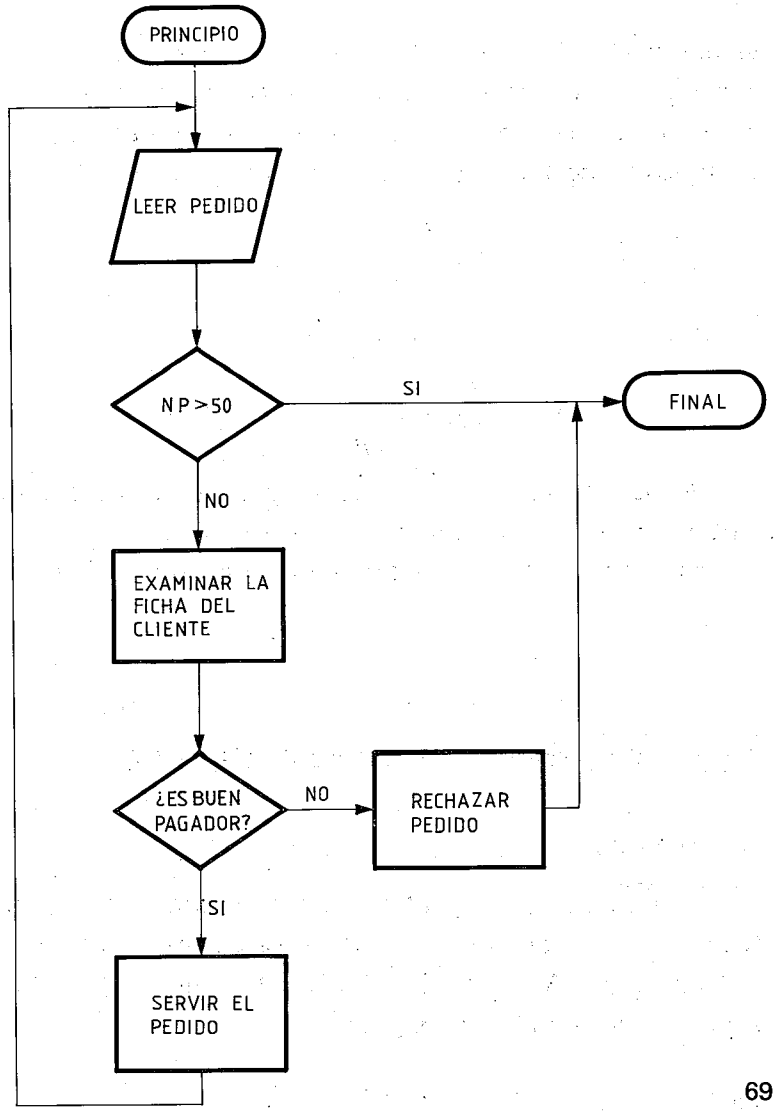


Fig. 2.26.

Ejemplo 2

Organigrama para calcular la media aritmética, (Fig. 2.27).

Supongamos un lote de 20 fichas, cada una contiene un número cualquiera. Calcular la media aritmética de estos 20 números, e imprimir el resultado.

Para representar la operación de lectura se utiliza el símbolo de la *tarjeta perforada* y para imprimir el resultado se utiliza el símbolo de *documento.*

La variable S se utiliza para representar las sumas parciales y la suma total de los números leídos.

La variable I se utiliza como contador. Cuando I sea mayor que 20 se habrán sumado todos los números y sólo queda dividir la suma total por el total de números, es decir, por 20.

La variable M contendrá al final del proceso la media aritmética de los 20 números.

Una vez construido el organigrama puede efectuarse una comprobación para asegurarnos de que funciona correctamente.

Por ejemplo, supongamos que sólo hay tres fichas en el lote: la primera contiene el número 10, la segunda el 12 y la tercera el 8. Es evidente que si el organigrama funciona para 3 números también funciona para 20 o para cualquier otra cantidad, sólo hay que modificar algunos símbolos.

Cuando comienza el proceso, las variables S e I valen 0 y 1 respectivamente. Estas variables van cambiando de valor durante el proceso. Después de leer el primer número (N = 10) los valores de S y de I cambian:

$$S = S + N = 0 + 10 = 10$$
$$I = I + 1 = 1 + 1 = 2$$

En este caso I debe ser mayor que 3 para salir del bucle y efectuar la división de S por 20.

Por lo tanto como I no es mayor que 3 (I = 2) se bifurca a la operación de lectura. Ahora el número leído es 12 (N = 12), y a continuación se efectúan las operaciones:

$$S = S + N = 10 + 12 = 22$$
$$I = I + 1 = 2 + 1 = 3$$

Como I no es mayor que 3, se bifurca de nuevo. Se lee el nuevo número (N = 8) repitiéndose el proceso:

$$S = S + N = 22 + 8 = 30$$

$$I = I + 1 = 3 + 1 = 4$$

Ahora se cumple la condición $I > 3$, por lo tanto la siguiente operación es:

$$M = S/3 = 30/3 = 10$$

La media aritmética es 10

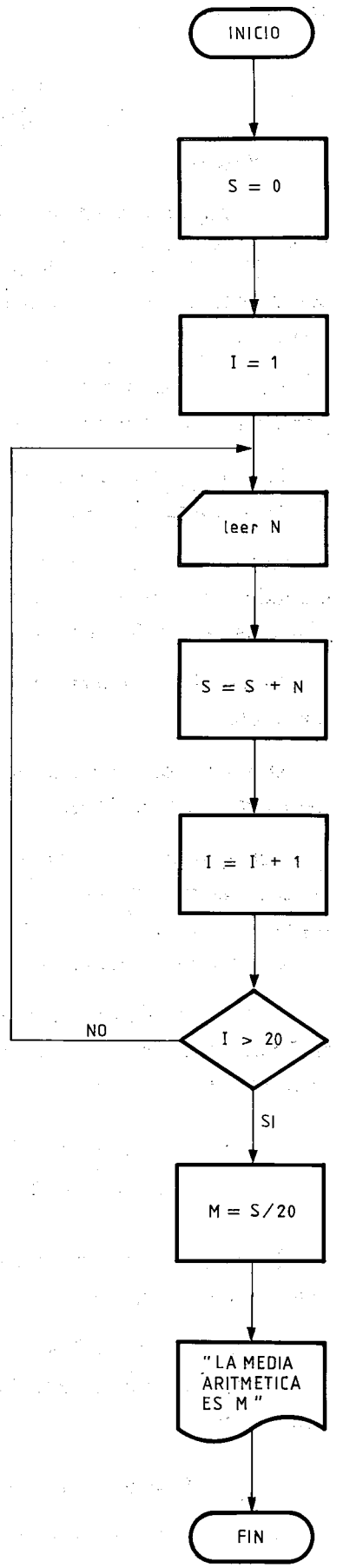


Fig. 2.27. Organigrama para el cálculo de la media aritmética de 20 números cualquiera.

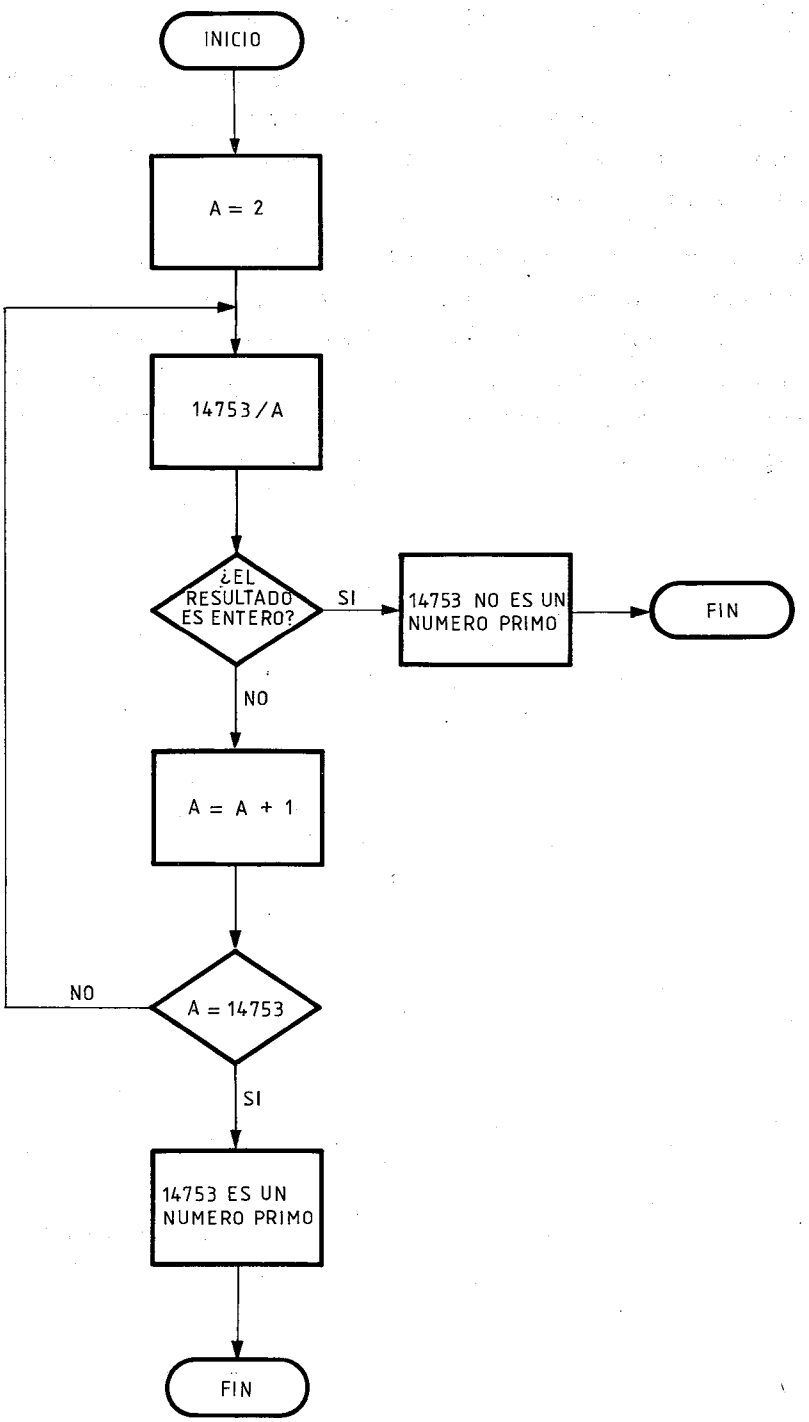
Ejemplo 3

Organigrama para determinar si 14753 es un número primo. Utilizaremos la variable A cuyo valor inicial lo pondremos igual a 2.

En primer lugar se pone la variable N igual a 14753, y la variable A igual a 2. A continuación se divide 14753 por A (es decir, por 2) y se pregunta si el resultado es entero, en caso afirmativo implica que 14753 no es un número primo y finaliza el proceso.

En caso de que el resultado de la división no sea entero se suma a continuación, 1 a la variable A (quedando A = 3) y se pregunta si A es igual a 14753. Si la respuesta es negativa se vuelve a repetir el proceso del bucle hasta encontrar una salida del bucle, es decir, si se cumple alguna de las dos condiciones: 1) que el resultado de una división efectuada sea entero, y 2) que A sea igual a 14753, en cuyo caso el número 14753 es primo.

Fig. 2.28. Organigrama para determinar si 14753 es un número primo.


INICIO
A = 2
14753 / A
¿EL RESULTADO ES ENTERO?
SI
14753 NO ES UN NUMERO PRIMO
FIN
NO
A = A + 1
NO
A = 14753
SI
14753 ES UN NUMERO PRIMO
FIN

Ejemplo 4:

Organigrama para clasificar un conjunto de valores (50) en orden ascendente. Estos valores se encuentran almacenados en la memoria del ordenador.

X_1	X_2	X_3	X_4	...
8	4	5	7	...

Se utiliza la variable N como contador. En primer lugar se hace N = O. Cuando N es igual a 49 finaliza el proceso, ya que para clasificar 50 valores se deben efectuar 49 comparaciones.

Al comienzo se pone el subíndice I igual a 1. La primera comparación se efectuará entre X_1 y X_2.

En el supuesto de los 4 números anteriores, $X_1 = 8$ y $X_2 = 4$. Siguiendo el organigrama, como $X_1 > X_2$, el contenido de X_2 pasa a X_1, y el contenido de X_1 pasa a X_2,

X_1	X_2	X_3	X_4	...
4	8	5	7	...

A continuación se hace I = 1 y N = O. Se pregunta si $X_1 > X_2$. Ahora la respuesta es negativa, entonces,

$$I = I + 1 = 2$$

$$N = N + 1 = 1$$

Ahora se compara X_2 con X_3. Como X_2 es mayor se intercambian los valores,

$$X_{I+1} = X_I$$

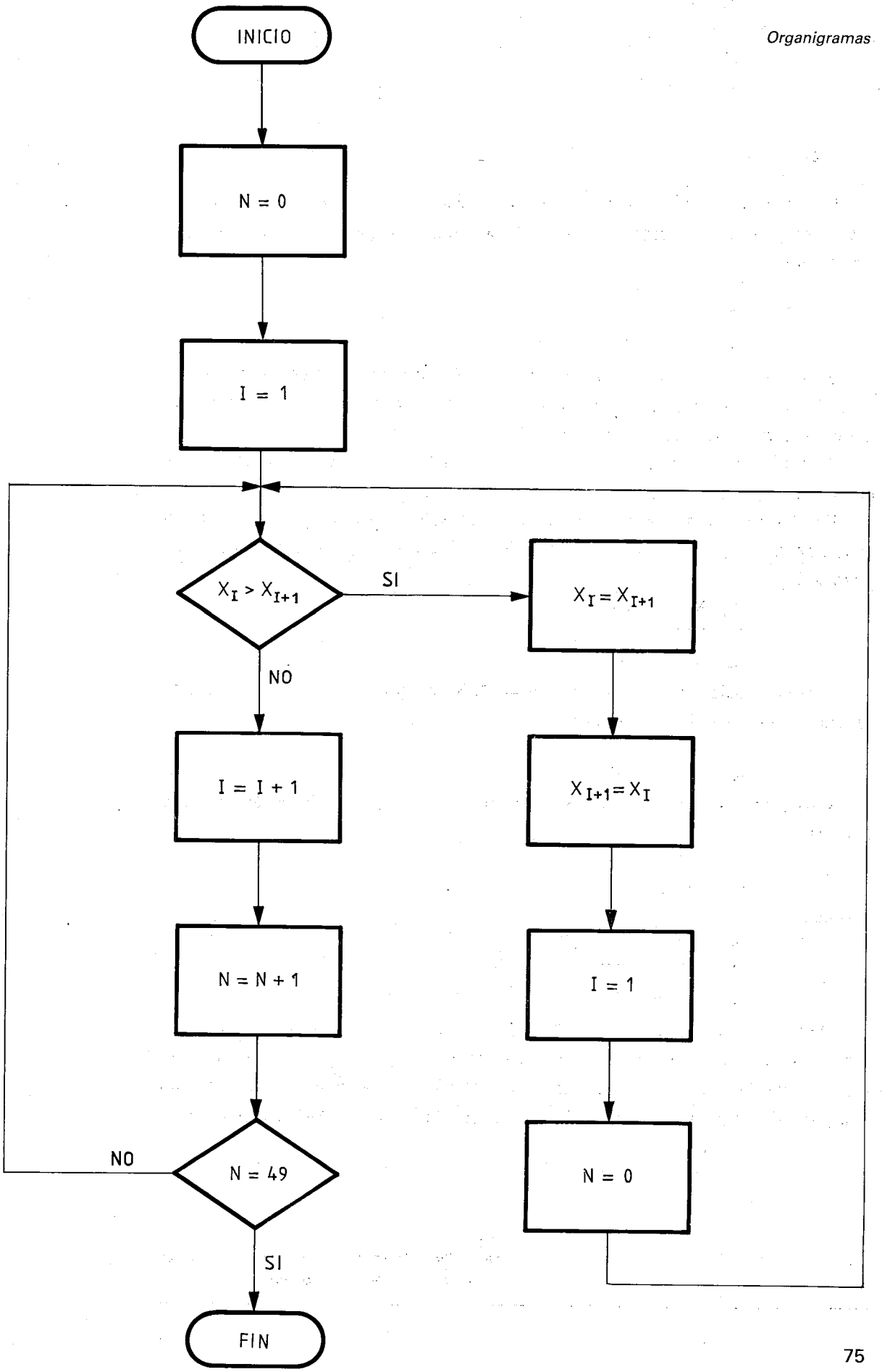
$$X_I = X_{I+1}$$

y queda:

X_1	X_2	X_3	X_4	...
4	5	8	7	...

Nuevamente se pone I = 1 y N = O, y se compara X_1 con X_2. Como X_1 es menor, se suma 1 al índice y 1 a N y se compara X_2 con X_3 y así se continúa sucesivamente hasta que N sea igual a 49, que serían las últimas 49 comparaciones que se deben efectuar para clasificar los 50 números.

Fig. 2.29. Organigrama para clasificar 50 números en orden ascendente. ▶


INICIO
N = 0
I = 1
$X_I > X_{I+1}$
SI
$X_I = X_{I+1}$
NO
I = I + 1
$X_{I+1} = X_I$
N = N + 1
I = 1
NO
N = 49
N = 0
SI
FIN

EJERCICIOS

1. Leer *x* y *n*. Calcular y = x^n e imprimir x e y.

2. Hacer el organigrama que efectúe el siguiente proceso 50 veces: leer cinco números, sumarlos e imprimir el resultado.

3. Construir el organigrama para calcular la suma de los 100 primeros números naturales.

4. Construir el organigrama para hallar el mayor de los números A, B y C.

5. Sumar los números pares del 1 al 100.

6. Organigrama que calcula la media aritmética de los 100 primeros números naturales.

7. Sea un lote de 50 fichas, cada una de ellas contiene un número. Construir un organigrama que calcule la suma de estos números de forma que se imprima la suma de los que sean pares y la suma de los impares por separado.

8. Construir el organigrama que calcule las raíces de una ecuación de segundo grado. (Para calcular la raíz cuadrada se emplea SQR).

9. Organigrama para calcular el máximo y el mínimo valor de un lote de 50 fichas desordenadas, cada una de ellas contiene un número.

10. Organigrama para calcular el factorial de un número cualquiera, N.

3.

TABLAS DE DECISION

CUADRO RESUMEN DE LA LECCION

Tablas de Decisión

- Estructura básica
- Condiciones
- Reglas de decisión
- Tratamientos (Acciones)
- Ejemplos
- Conversión de Tablas en Organigramas

Al igual que con los organigramas, una de las formas de llegar a la solución de un problema consiste en utilizar las *tablas de decisión.*

Una tabla de decisión contiene todas las posibilidades que deben tenerse en cuenta al definir un problema así como las acciones que han de tomarse.

En la documentación de un programa se utilizan en algunas ocasiones las tablas de decisión en lugar de los organigramas.

3.1. ESTRUCTURA BASICA

Las tablas de decisión son representaciones gráficas de las acciones o tratamientos a tomar en determinadas condiciones por el ordenador para la resolución de un problema.

Las tablas de decisión permiten decidir los tratamientos necesarios frente a distintas combinaciones de condiciones.

En la figura 3.1 se presenta la estructura básica de una tabla de decisión.

Una tabla de decisión suele representarse en forma de matriz tal como muestra la figura.

Las líneas horizontales dobles separan las condiciones de los tratamientos. A la derecha de las líneas verticales dobles se encuentran las *reglas de decisión.* Cada columna representa una regla de decisión.

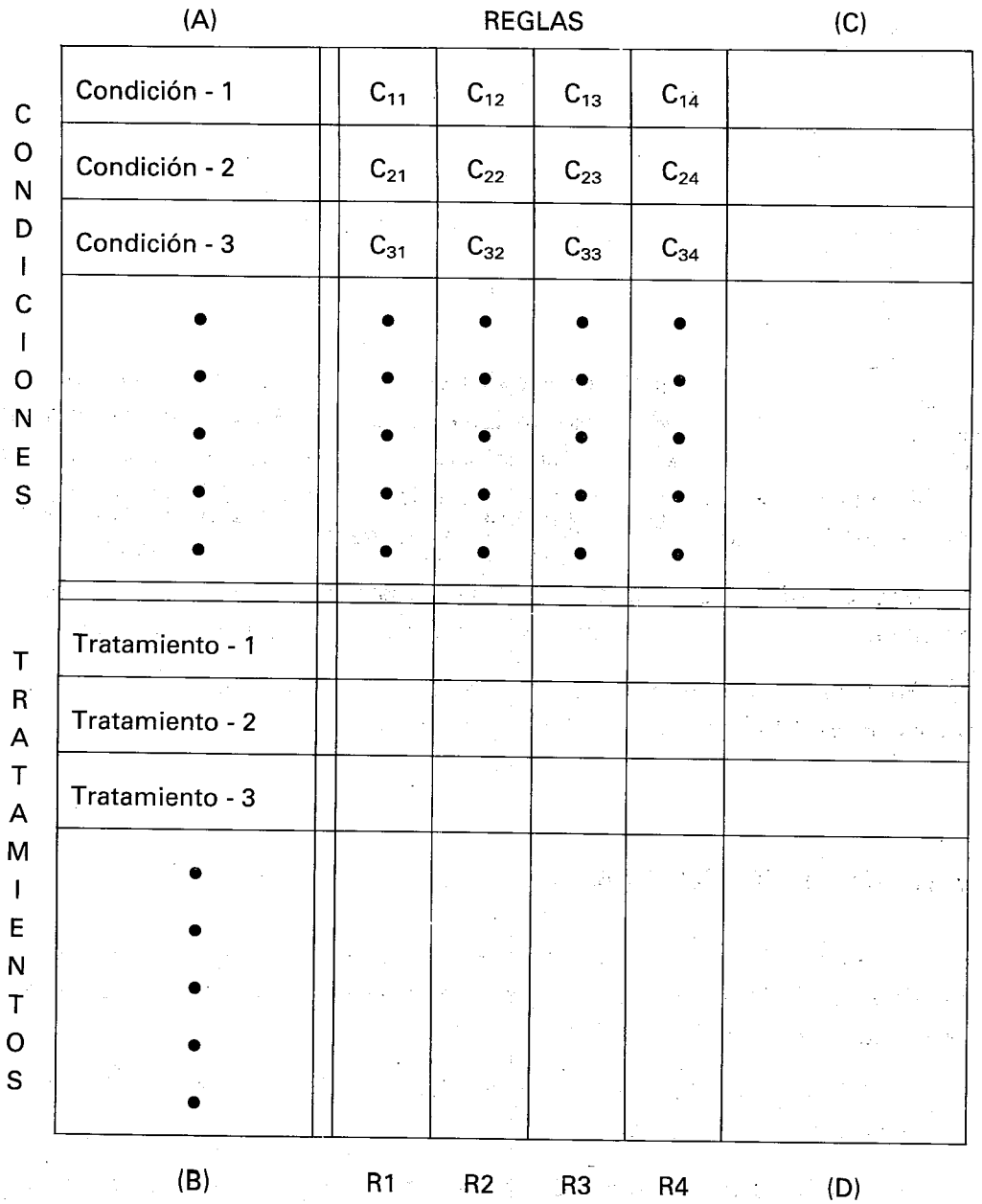


Fig. 3.1. Estructura de una tabla de decisión.

El cuadrante A de la tabla se llama *matriz de condiciones* y en él se escriben cada una de las condiciones que pueden existir en el problema. No es necesario que estas condiciones se escriban en el orden en que se van a tratar pero sí es conveniente que se escriban siguiendo una secuencia lógica.

El cuadrante superior derecho (C) de la tabla corresponde a la *entrada de condiciones*, en donde se escriben todas las situaciones combinadas posibles de las condiciones. El valor de cada situación C_{ij}, puede ser *S* (sí) o *N* (no).

A las condiciones que no interesa darle un valor (S o N) se dejan en blanco o se pone un guión (–) en el lugar correspondiente.

En el cuadrante B de la tabla, denominado *matriz de tratamientos,* se escriben los tratamientos o acciones que han de tomarse para resolver el problema.

Cada una de las acciones que llevará a cabo la máquina se basan en las condiciones escritas en la parte superior de la tabla. Estas acciones se colocan en la parte inferior izquierda (cuadrante B).

Los tratamientos (acciones) se acoplan con las condiciones adecuadas en las columnas donde se han escrito las S y N. En las columnas correspondientes se escribe una X.

Las acciones que deben aplicarse son las señaladas con una X, mientras que las acciones no señaladas deben ignorarse.

El cuadrante inferior derecho (D) de la tabla corresponde a la *entrada de tratamientos*, en donde se va colocando el signo X en los lugares que corresponda al tratamiento que se va a tomar, dicho tratamiento está determinado por la condición o combinación de condiciones de la misma columna, es decir, de la misma regla de decisión. Por lo tanto, los tratamientos no señalados con una X no se han de tener en cuenta.

Por ejemplo, la siguiente tabla de decisión contiene tres condiciones y dos posibles tratamientos a tomar. En el cuadrante superior derecho de la tabla aparecen todas las combinaciones posibles de esas tres condiciones binarias (S, N). Y en la parte inferior derecha aparecen señaladas con una X (escogidas al azar como ejemplo) los tratamientos a tomar.

En la primera columna de la misma se puede observar que si se cumplen las tres condiciones entonces la acción a realizar es TRATAMIENTO-2. En la segunda columna, si se cumplen las condiciones 1 y 2 y no se cumplen la 3, la

acción a tomar será TRATAMIENTO-1. En la tercera columna, si se cumplen las condiciones 1 y 3 y no se cumple la 2, las acciones a tomar serán el tratamiento 1 y 2, y así sucesivamente.

Condición - 1	S	S	S	S	N	N	N	N
Condición - 2	S	S	N	N	S	S	N	N
Condición - 3	S	N	S	N	S	N	S	N
Tratamiento - 1		X	X	X			X	
Tratamiento - 2	X		X		X	X	X	X

3.2. MATRIZ DE CONDICIONES

En las tablas de decisión deben analizarse cada una de las condiciones del problema así como las posibles alternativas que pueden tomarse.

En la matriz de condiciones deben escribirse todas las condiciones que determinan la decisión o el tratamiento a tomar.

Las condiciones en una tabla de decisión son, generalmente, binarias, es decir, de dos alternativas posibles (S, N). Estas tablas se denominan tablas de *entradas limitadas*.

Si las condiciones de una tabla poseen más de dos alternativas se llaman tablas de *entradas ampliadas*. Estas tablas pueden convertirse en tablas de entradas limitadas.

Supongamos una condición con cuatro alternativas:

Condición	Alternativas ampliadas			
Color	Rojo	Azul	Amarillo	Verde

Esta condición puede transformarse en otras condiciones binarias:

Condiciones	Alternativas limitadas			
Rojo	S	N	N	N
Azul	–	S	N	N
Amarillo	–	–	S	N
	Rojo	Azul	Amarillo	Verde

3.3. TRATAMIENTOS

La matriz de tratamientos está constituida por las acciones a ejecutar.

Según el tipo de acciones, los tratamientos pueden ser de dos tipos:

a) tratamientos elementales, como por ejemplo: «Ir a 1», «Parar», «A = 0», etc.

b) tratamientos complejos: como por ejemplo: «Efectuar la facturación», «Actualizar un fichero», etc.

3.4. REGLAS DE DECISION

Cada regla de decisión está constituida por las situaciones (S ó N) de las condiciones y por las acciones resultantes de una decisión.

Las reglas de decisión se consideran una a una, individualmente, de manera que sólo una de ellas se puede cumplir en un determinado momento.

Cada una de las columnas de la parte derecha de la tabla representa una regla de decisión. Para saber cuántas columnas o reglas se han de poner se hace la siguiente operación: si las condiciones son binarias, eso significa que

pueden estar en dos situaciones diferentes (S ó N), entonces si hay n condiciones, el número de situaciones o de reglas será 2^n.

Por ejemplo, si hay tres condiciones binarias el número de reglas será:

$$2 \times 2 \times 2 = 2^3 = 8.$$

Por otra parte, si todas las condiciones no son binarias entonces las combinaciones de n condiciones no será 2^n. Por ejemplo si hay tres condiciones:

C1: 3 alternativas
C2: 4 alternativas
C3: 2 alternativas

el número de reglas de decisión será:

$$3 \times 4 \times 2 = 24$$

es decir, se multiplican entre sí las alternativas de cada condición, puesto que las alternativas de una condición se combinan con las alternativas restantes.

Las reglas de decisión pueden ser de dos tipos:

a) simples: cuando todas las condiciones tienen un determinado valor en la regla de decisión.

b) complejas: cuando alguna de las entradas de condiciones (situaciones) están en blanco.

Por ejemplo:

	R1	R2	R3	R4	R5
C1	S	N	S	N	–
C2	N	S	N	–	–
C3	S	N	–	–	–

Las reglas R1 y R2 son reglas de decisión simples y las reglas R3, R4 y R5 son complejas.

En algunas aplicaciones no se admiten las entradas en blanco. En ese caso, la columna que contiene la entrada en blanco debe sustituirse por dos columnas, (es decir, por dos reglas de decisión), la primera corresponderá a la S y la segunda a la N.

Así la tabla anterior quedaría:

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
C1	S	N	S	S	N	N	N	S	N	S	N	S	N	S	N
C2	N	S	N	N	S	N	N	S	S	N	N	S	S	N	N
C3	S	N	S	N	S	S	N	S	S	S	S	N	N	N	N

Por lo tanto, una regla compleja equivale a dos o más reglas simples.

Esta tabla no puede tener las 15 reglas de decisión que aparecen aquí, ya que cuando son condiciones binarias (con dos alternativas), el número de reglas de decisión es 2^n, siendo n el número de condiciones (en este caso n = 3). Por lo que el número de reglas será $2^3 = 8$. Es decir, existen 7 reglas que se repiten.

Eliminando estas reglas, la tabla queda:

	R1	R2	R3	R4	R5	R6	R7	R8
C1	S	N	S	N	N	S	S	S
C2	N	S	N	S	N	S	S	N
C3	S	N	N	S	N	S	N	N

Para facilitar la tarea de completar una tabla con valores binarios puede seguirse el método siguiente:

- En la primera fila las situaciones S y N se escriben alternativamente, una a una (la repetición de cada S o N es $2^0 = 1$). Por ejemplo: S N S N...
- En la segunda fila la escritura de las situaciones S y N se efectúan de dos en dos ($2^1 = 2$). Por ejemplo: S S N N S S N N...
- En la tercera fila se escriben de cuatro en cuatro ($2^2 = 4$). Por ejemplo: S S S S N N N N S S S S N N N N...
- En la cuarta fila de ocho en ocho ($2^3 = 8$). Por ejemplo: S S S S S S S S N N N N N N N N...
- Etc.

De esta forma la tabla quedará como sigue:

	R1	R2	R3	R4	R5	R6	R7	R8
C1	S	N	S	N	S	N	S	N
C2	S	S	N	N	S	S	N	N
C3	S	S	S	S	N	N	N	N

Según esto veamos cómo se puede simplificar una tabla con reglas simples.

Supongamos la siguiente tabla:

	R1	R2	R3	R4	R5	R6	R7	R8
C1	S	N	S	N	S	N	S	N
C2	S	S	N	N	S	S	N	N
C3	S	S	S	S	N	N	N	N
T1	X	X	X	X				
T2					X	X		
T3							X	
T4								X

Esta tabla indica que el tratamiento T1 se realiza en las 4 primeras reglas de decisión.

C1	S	N	S	N
C2	S	S	N	N
C3	S	S	S	S

De esta tabla se deduce que el tratamiento T1 sólo depende de C3 ya que T1 se realiza en cualquier combinación binaria (S, N) de C1 y C2. Es decir, T1 es independiente de las condiciones C1 y C2. Sin embargo, C3 es siempre S. Por lo tanto estas cuatro reglas pueden simplificarse en una sola:

	R1
C1	–
C2	–
C3	S
T1	X

Cuando el tratamiento es independiente de la condición la entrada estará en blanco. De ahí que sólo se escriba la S en C3.

El tratamiento T2 se realiza en las reglas de decisión R5 y R6.

C1	S	N
C2	S	S
C3	N	N

De esta tabla se deduce que T2 no depende de C1. Por lo tanto, quedará de la forma siguiente:

	R2
C1	–
C2	S
C3	N
T2	X

Ya que en los dos casos en que se realiza T2, la condición C2 vale S y la condición C3 vale N.

Los tratamientos T3 y T4 se realizan en R7 y R8 respectivamente, por lo que no es posible simplificarlos más.

La tabla de decisiones queda por lo tanto de la forma siguiente:

	R1	R2	R3	R4
C1	–	–	S	N
C2	–	S	N	N
C3	S	N	N	N
T1	X			
T2		X		
T3			X	
T4				X

3.5. EJEMPLOS

Ejemplo 1

Supongamos que se efectúan una serie de pedidos a una determinada empresa y que ésta acepta o rechaza cada uno de los pedidos dependiendo de ciertas condiciones:

Condiciones:

- ¿Hay existencias del producto en el almacén?
- ¿El cliente es buen pagador?

Tratamientos (acciones):

- Aceptar la orden de pedido.
- Rechazar la orden de pedido.

La tabla de decisión será la siguiente:

¿Existencias?	S	S	N	N
¿Pagador?	S	N	S	N
Aceptar	X			
Rechazar		X	X	X

El número de reglas de decisión será $2^2 = 2 \times 2 = 4$. Estas cuatro reglas se pueden reducir a tres ya que si no hay existencias no importa que el cliente sea o no pagador por lo que esta condición es *independiente* de la anterior.

Entonces la tabla quedará del siguiente modo:

¿Existencias?	S	S	N
¿Pagador?	S	N	–
Aceptar	X		
Rechazar		X	X

Ejemplo 2

A los trabajadores de una determinada empresa se les aplica tres tipos diferentes de impuestos según la situación de cada trabajador.

Condiciones

¿Edad < 25?
¿Edad > 55?
¿Casado?
¿Tiene hijos?

Tratamientos

Impuesto-1
Impuesto-2
Impuesto-3

Edad < 25		S	S	S	S	S	S	S	S	N	N	N	N	N	N	N	N
Edad > 55		S	S	S	S	N	N	N	N	S	S	S	S	N	N	N	N
Casado		S	S	N	N	S	S	N	N	S	S	N	N	S	S	N	N
Tiene hijos		S	N	S	N	S	N	S	N	S	N	S	N	S	N	S	N
Impuesto - 1																	
Impuesto - 2																	
Impuesto - 3																	

El número de reglas de decisión será: $2^4 = 2 \times 2 \times 2 \times 2 = 16$. Ahora bien, muchas de estas reglas son contradictorias, por lo que se han de suprimir. Como, por ejemplo, un trabajador no puede tener menos de 25 años de edad y más de 50. Por otra parte, si tiene más de 50 ya se incluye la condición de que tiene más de 25 años.

Eliminando, por lo tanto, las condiciones redundantes o repetitivas y contradictorias, la tabla puede quedar de la siguiente manera:

Edad < 25	S	S	S	N	N	N	N	N	N
Edad > 55	–	–	–	S	S	S	N	N	N
Casado	S	S	N	S	S	N	S	S	N
Hijos	S	N	–	S	N	–	S	N	–
Impuesto - 1		X	X		X		X	X	
Impuesto - 2	X		X			X		X	X
Impuesto - 3		X	X	X	X	X			X

3.6. CONVERSION DE TABLAS EN ORGANIGRAMAS

El uso de tablas de decisión es muy conveniente para la resolución de algunos problemas ya que permiten representar todas las condiciones y acciones fundamentales que aparecen en los mismos. En el caso de que sea necesario añadir una condición o acción nueva simplemente se amplía la tabla, sin tener que reconstruir todo el planteamiento desarrollado para la resolución del problema.

Por lo tanto, en determinados problemas, es conveniente realizar una tabla de decisión antes de dibujar el organigrama.

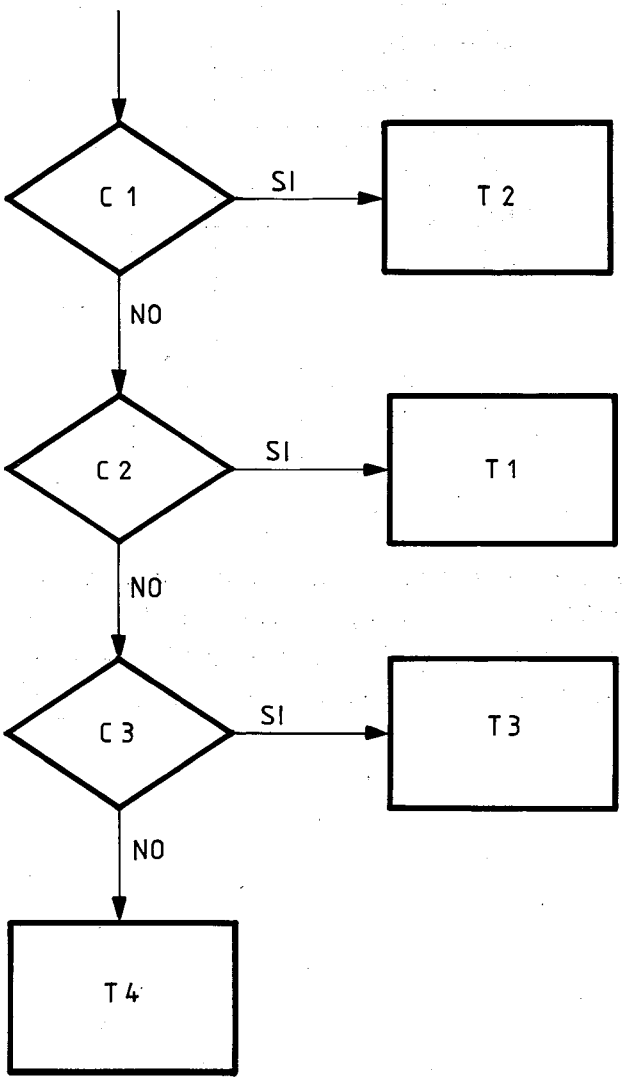
Supongamos la siguiente tabla:

C1	S	N	N	N
C2	–	S	N	N
C3	–	–	S	N
T1		X		
T2	X			
T3			X	
T4				X

Al hacer el organigrama hay que tener en cuenta el orden de importancia de cada condición. En este ejemplo, la condición C1 es la que se tratará primero en el organigrama ya que es la condición que interviene en un mayor número de reglas; en segundo lugar la condición C2, y en tercer lugar C3.



Fig. 3.2. La condición C1 es la de mayor importancia, puesto que C2 depende de C1, y C3 depende de C1 y de C2.

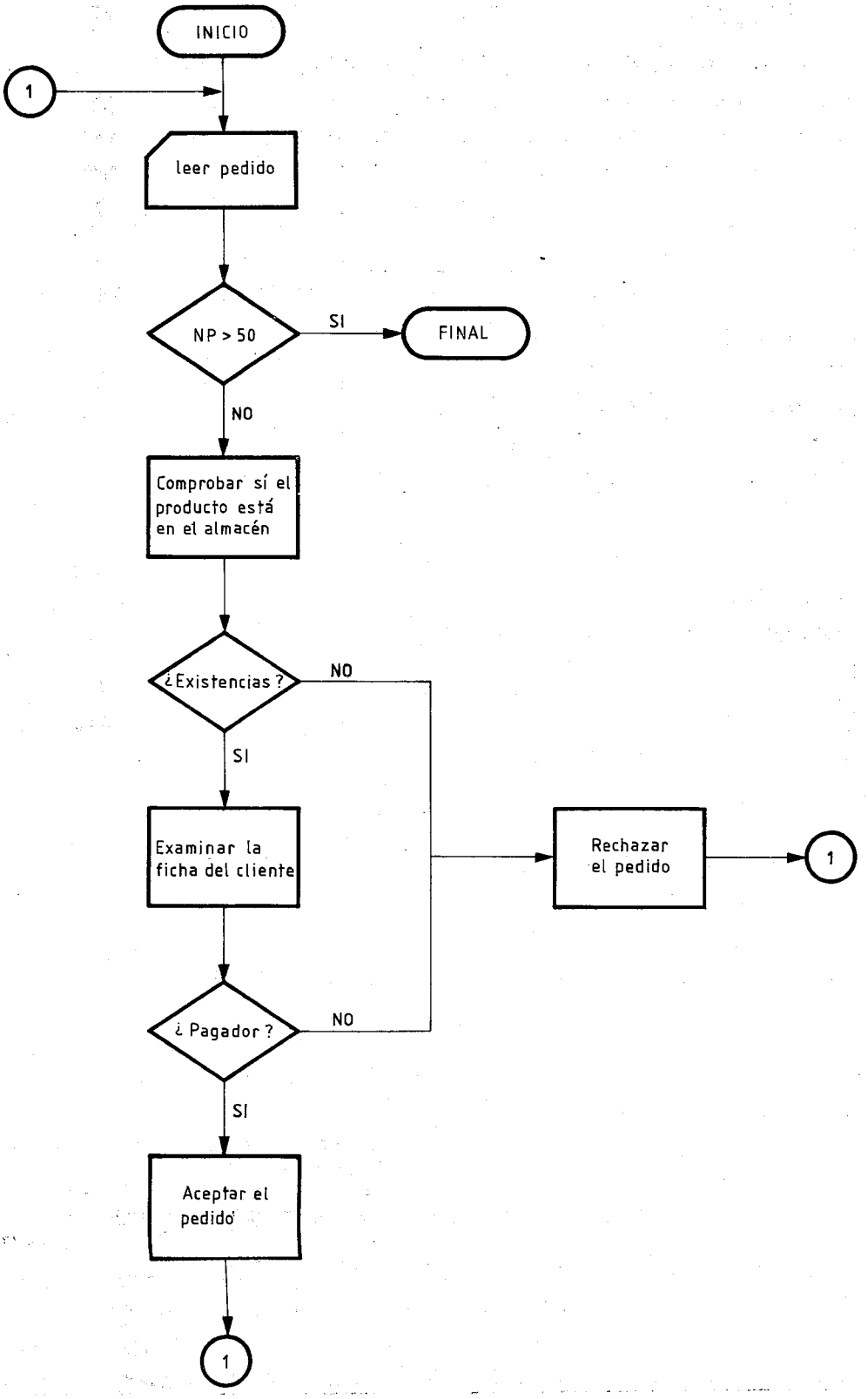

C 1
SI
T 2
NO
C 2
SI
T 1
NO
C 3
SI
T 3
NO
T 4

Ejemplo 1

Dibujar el organigrama de la tabla siguiente:

¿Existencias?	S	S	N
¿Pagador?	S	N	–
Aceptar	X		
Rechazar		X	X

Fig. 3.3. Si se cumplen ambas condiciones («existencias» y «pagador») se «acepta el pedido». Si una de ellas no se cumple se «rechaza el pedido».

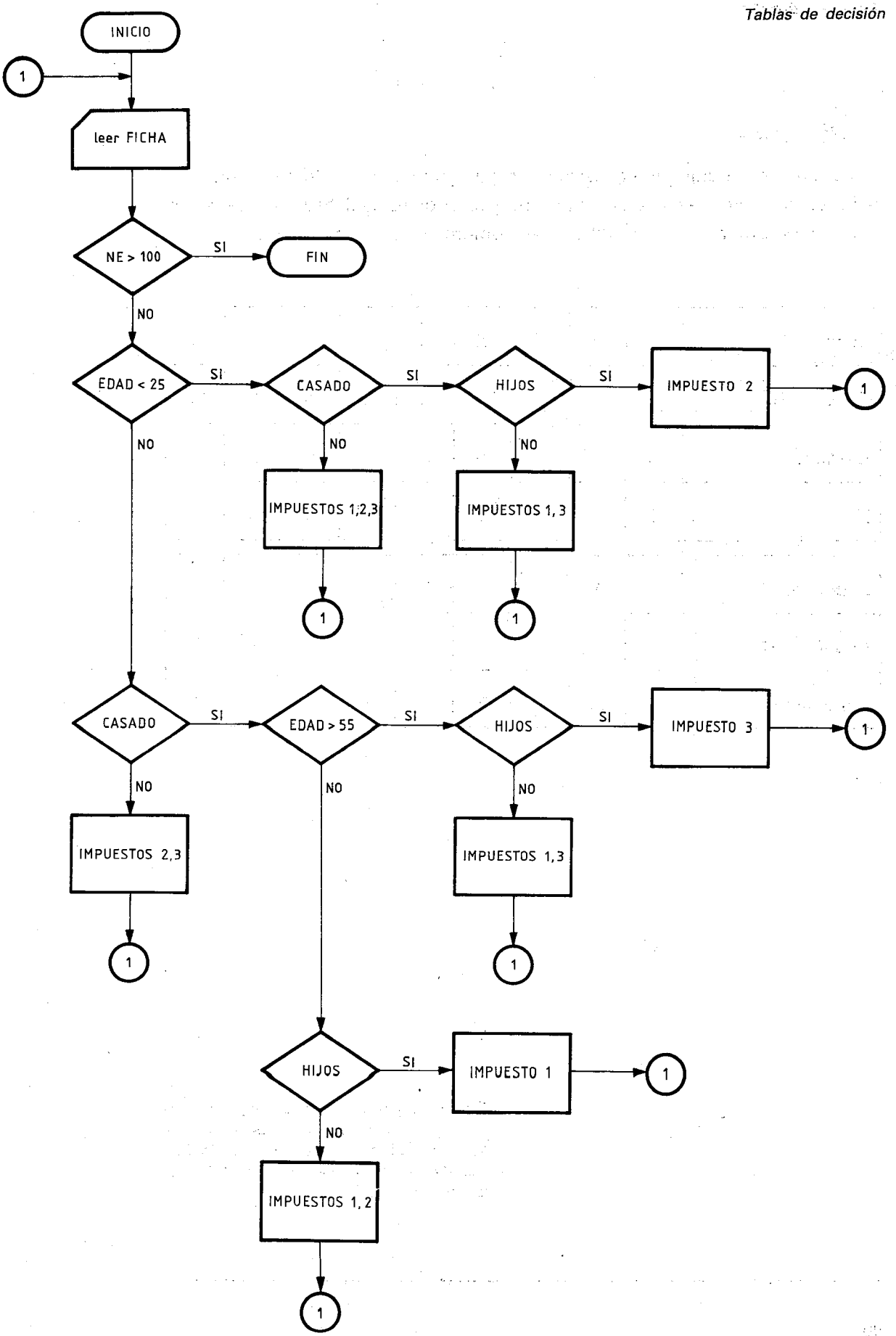

INICIO
1
leer pedido
NP > 50
SI
FINAL
NO
Comprobar sí el producto está en el almacén
¿Existencias ?
NO
SI
Examinar la ficha del cliente
Rechazar el pedido
1
¿ Pagador ?
NO
SI
Aceptar el pedido
1

Ejemplo 2

Dada la tabla siguiente construir el organigrama. Teniendo en cuenta que se lee cada vez una ficha de un empleado, y estando estas fichas clasificadas en orden ascendente del 1 al 100 por el número de empleado (NE).

Edad < 25	S	S	S	N	N	N	N	N	N
Edad > 55	–	–	–	S	S	S	N	N	N
Casado	S	S	N	S	S	N	S	S	N
Hijos	S	N	–	S	N	–	S	N	–
Impuesto - 1		X	X		X		X	X	
Impuesto - 2	X		X			X		X	X
Impuesto - 3		X	X	X	X	X			X

Fig. 3.4. En el organigrama, una vez aplicados los correspondientes «impuestos», se lee una nueva ficha. El proceso finaliza cuando el número del empledo (NE), es mayor que 100.


INICIO
1
leer FICHA
NE > 100
SI
FIN
NO
EDAD < 25
SI
CASADO
SI
HIJOS
SI
IMPUESTO 2
1
NO
NO
NO
IMPUESTOS 1;2;3
IMPUESTOS 1, 3
1
1
CASADO
SI
EDAD > 55
SI
HIJOS
SI
IMPUESTO 3
1
NO
NO
NO
IMPUESTOS 2,3
IMPUESTOS 1,3
1
1
HIJOS
SI
IMPUESTO 1
1
NO
IMPUESTOS 1, 2
1

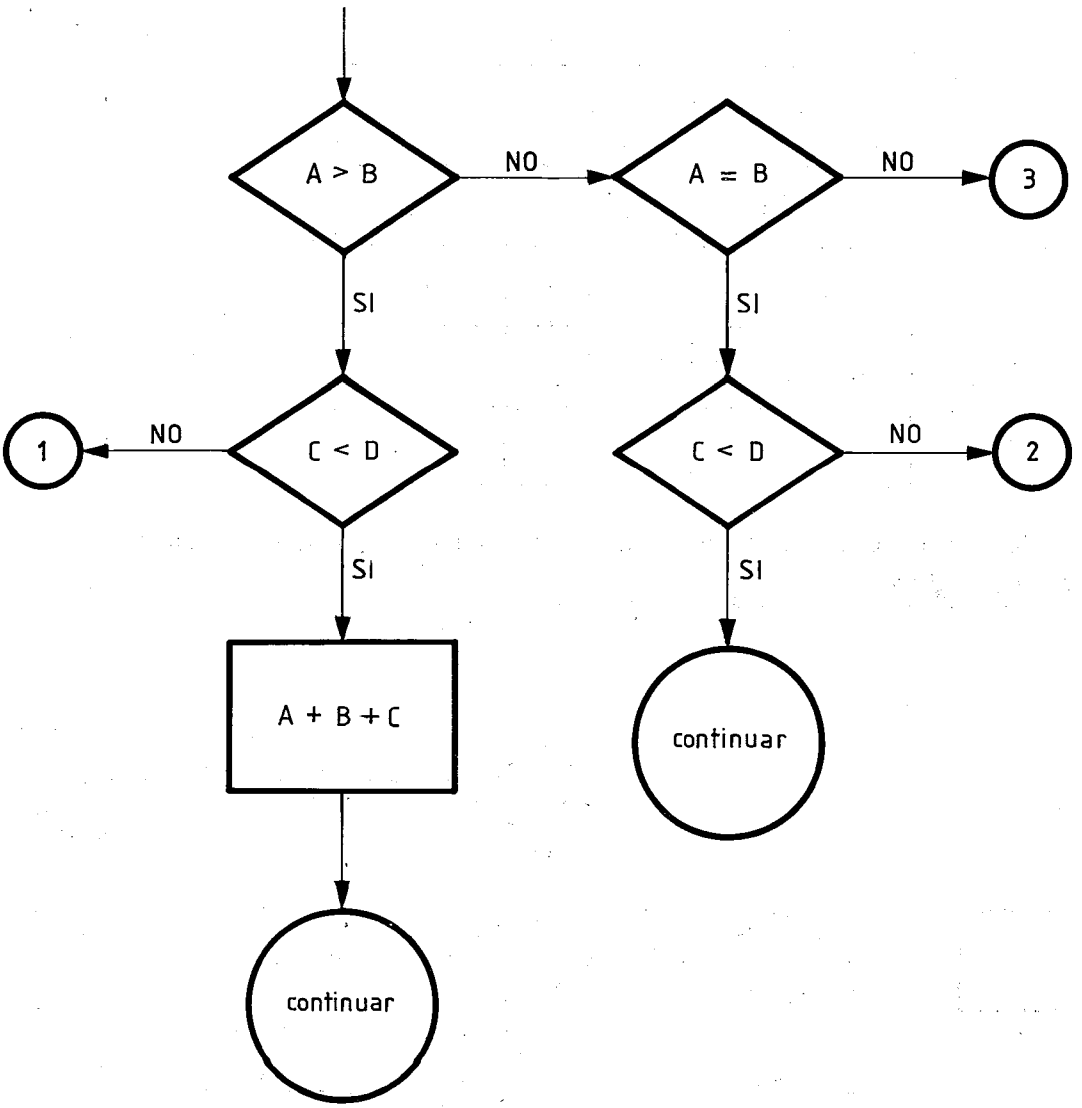
Ejemplo 3

Sea la siguiente tabla de decisión:

A > B	S	N	S	N
A = B	N	S	N	N
C < D	N	N	S	–
Ir a 1	X			
Ir a 2		X		
A + B + C			X	
Ir a 3				X

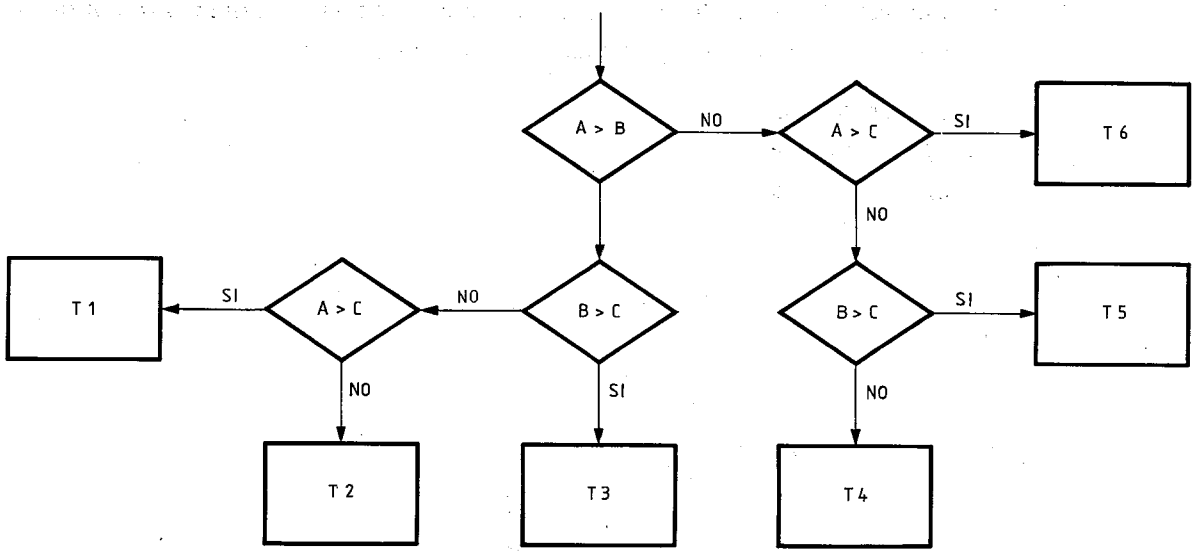
Dibujar el organigrama correspondiente.

Fig. 3.5. Parte del organigrama en donde se efectúan las comparaciones de la tabla de decisión anterior.


A > B
NO
A = B
NO
3
SI
SI
1
NO
C < D
C < D
NO
2
SI
SI
A + B + C
continuar
continuar

EJERCICIOS

1. Dado el organigrama de la figura siguiente construir la tabla de decisión correspondiente.



2. Una tabla de decisión de entradas ampliadas puede tener:

   a) Una alternativa posible.
   b) Dos alternativas.
   c) Más de dos alternativas.

3. Si en una tabla de decisión hay n condiciones binarias, el número máximo de reglas de decisión es:

   a) $2.^n$
   b) 2.
   c) 8.

4. Sean dos condiciones de una tabla de decisión, C1 y C2, con 2 y 4 alternativas respectivamente. El número de reglas de decisión es:

   a) 2.
   b) $2^2 = 4$.
   c) $2 \times 4 = 8$.

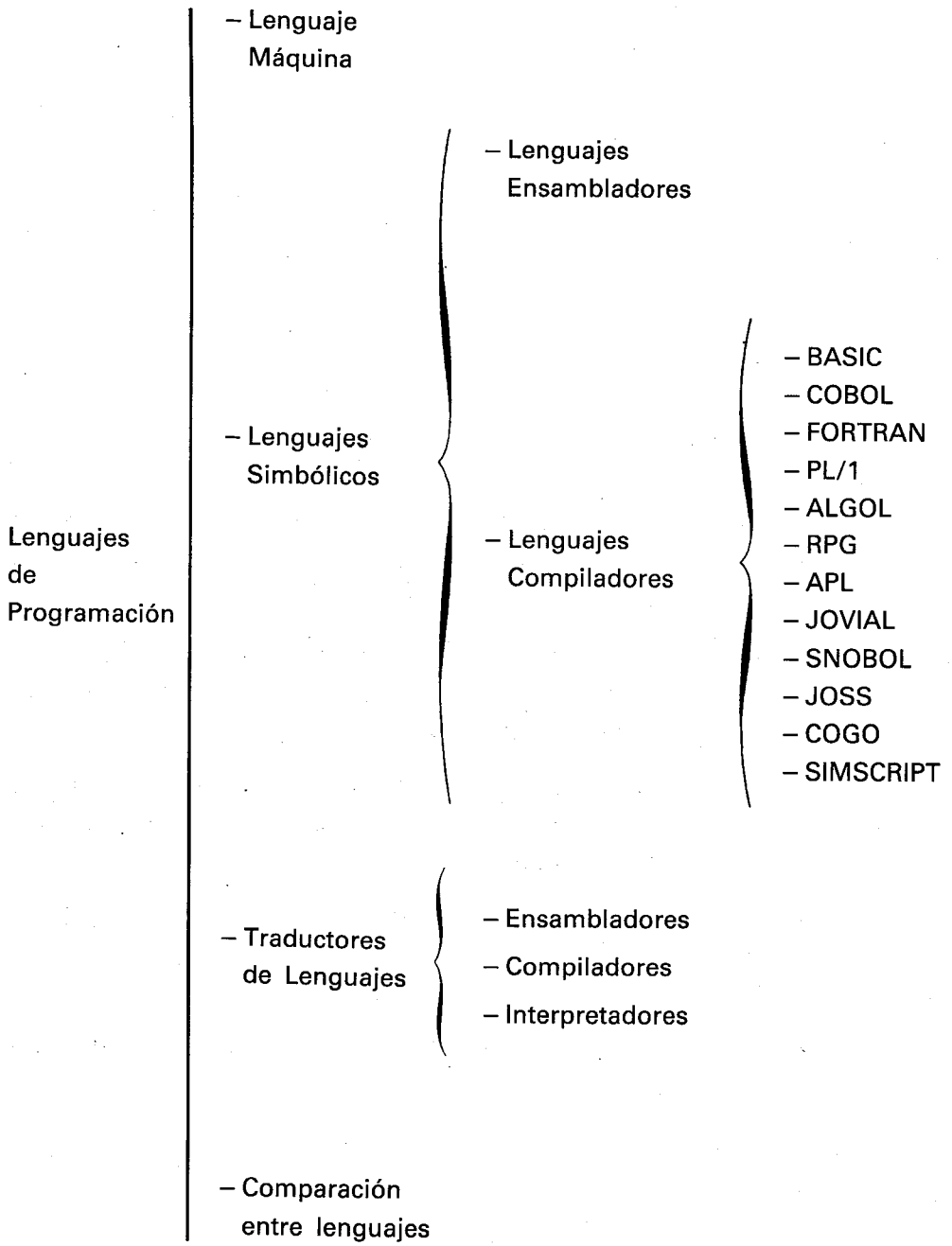
5. Cuando alguna de las entradas de condiciones está en blanco (o contiene un guión, –) se dice que las reglas de decisión son:

   a) elementales.
   b) simples.
   c) complejas.

4.

LENGUAJES DE PROGRAMACION

CUADRO RESUMEN DE LA LECCION



Una vez realizado el organigrama, el paso siguiente en la resolución de un problema es desarrollar un conjunto de instrucciones (programa) de forma comprensible por el ordenador.

Para que el ordenador pueda ejecutar las instrucciones de un programa, éste debe estar escrito en un lenguaje capaz de ser entendido por el ordenador. Estos lenguajes se llaman *lenguajes de programación*.

Al igual que los seres humanos se comunican entre sí mediante un determinado lenguaje, los lenguajes de programación sirven para la comunicación del hombre con el ordenador.

Como en todo lenguaje, en un lenguaje de programación deben seguirse una serie de reglas o normas para una utilización correcta. Estas *reglas gramaticales* difieren de un lenguaje a otro.

Los lenguajes de programación, como ocurre con los lenguajes naturales (castellano, inglés, francés, etc.), tienen cada uno de ellos su propio alfabeto.

Puesto que los lenguajes naturales resultan en muchas ocasiones ambiguos y poco precisos y que sus significados son a menudo diferentes para distintas personas, su aplicación en los ordenadores es muy compleja e ineficaz. De ahí que los lenguajes de programación son lenguajes artificiales creados para la comunicación con los ordenadores. Estos lenguajes artificiales carecen de redundancias y de diversas interpretaciones.

El programador debe escoger el lenguaje que mejor enuncie su problema y que resulte más adecuado al sistema de proceso de datos empleado, es decir, que ocupe el mínimo espacio en la memoria, que se procese en el menor tiempo posible, etc.

Existen dos tipos de lenguajes de programación:

- lenguajes máquina.
- lenguajes simbólicos.

4.1. LENGUAJE MAQUINA

La finalidad de los lenguajes de programación es la realización de programas para los ordenadores. Cada lenguaje de programación consta de una serie de instrucciones que utiliza el programador para codificar un programa *(programa fuente).* Estas instrucciones se convierten en códigos que pueden ser interpretados por la unidad de control del ordenador *(programa objeto).* Las instrucciones resultantes de esta conversión y que constituyen el programa objeto se denominan *instrucciones máquina.*

Los diseñadores de ordenadores especifican las características de los conmutadores electrónicos que posee el ordenador. Las combinaciones de estos conmutadores reciben el nombre de *lenguaje máquina.* Por lo tanto, las instrucciones que componen el lenguaje máquina (*instrucciones máquina*) se representan en código binario (por los dos estados que pueden adoptar dichos conmutadores).

El conjunto de instrucciones máquina de un determinado ordenador constituye un *lenguaje máquina.*

Las instrucciones escritas en lenguaje máquina son interpretadas directamente por el ordenador.

Cada ordenador trabaja con su propio lenguaje máquina, siendo notables las diferencias entre las diversas marcas de ordenadores.

Un lenguaje máquina está formado por instrucciones, principalmente de los siguientes tipos:

- Instrucciones que transfieren los datos desde un dispositivo de entrada a la memoria central. Estas instrucciones indican: desde qué unidad de entrada se efectúa la transferencia; el número de datos que se trans-

fieren y cuáles serán las posiciones de memoria que ocuparán dichos datos.
Estas instrucciones, además, controlan las unidades de entrada.

- Instrucciones que transfieren los datos desde la memoria central a un dispositivo de salida. Indican: la unidad o unidades de salida a la que van dirigida los datos; el número de datos y la zona de memoria desde donde se transfieren.
  Además controlan el equipo de salida.

- Instrucciones que controlan la secuencia que debe seguir el programa. Estas instrucciones son de dos tipos:
  a) Instrucciones de control incondicional (bifurcación y parada).
  b) Instrucciones de control-condicional ($A < B$, $A = B$, $A > B$, etc.).

- Instrucciones que realizan operaciones aritméticas (+, −, ×, /).

- Instrucciones que mueven los datos de un lugar a otro de la memoria. Indican las posiciones hacia dónde van dirigidos los datos y desde qué posiciones provienen, y el número de caracteres.

Programar en lenguaje máquina resulta complicado y difícil puesto que es necesario utilizar el modo en que procesa los datos el ordenador cómo método de programación.

Si se programa en lenguaje máquina es necesario especificar las direcciones de memoria en donde deben almacenarse los datos y las instrucciones del programa y las direcciones de memoria en donde deben reservarse campos de memoria para las operaciones de entrada/salida.

En la realidad no suele programarse en lenguaje máquina. Los programadores utilizan para codificar sus programas, los llamados *lenguajes simbólicos.*

4.2. LENGUAJES SIMBOLICOS

Un lenguaje simbólico de programación es un sistema que permite al programador codificar el programa de una forma distinta al lenguaje máquina.

Mediante los lenguajes simbólicos, a los códigos de operación y a las direcciones de memoria indicadas en las instrucciones máquina se les asignan nombres simbólicos y se utilizan directamente los números decimales en lugar de escribirlos en binario.

Aplicando este tipo de codificación a las instrucciones máquina se obtienen otras más fáciles de entender y de codificar.

Una instrucción máquina puede ser, por ejemplo:

0101	001100	010010

esta instrucción puede significar que se desea sumar los datos contenidos en las posiciones de memoria indicadas por las direcciones 001100 y 010010.

La misma instrucción codificada en un lenguaje simbólico podría ser:

SUMAR A, B

que significa que se desea sumar el contenido de la posición A de la memoria al contenido de la posición B de la memoria, dejando el resultado en la posición B.

Los lenguajes simbólicos permiten al programador utilizar códigos que se relacionen con la función exacta de cada código. Por ejemplo, si la instrucción que quiere representar el programador es «sumar» puede codificarla como «SUMAR», con lo que el traductor de programas se encargará de convertir esta instrucción en la instrucción real de la máquina.

Dado que el único lenguaje que entiende el ordenador es el lenguaje máquina, los programas escritos en otro tipo de lenguaje deben traducirse mediante programas traductores a lenguaje máquina.

Los programas fuente están codificados en un determinado lenguaje simbólico.

Los programas traductores convierten las instrucciones simbólicas en otras instrucciones de lenguaje máquina (instrucciones máquina) y las direcciones simbólicas en direcciones de la máquina.

Los traductores disponen de los códigos de instrucciones máquina y de los códigos de instrucciones del lenguaje simbólico que se utilice.

Los programas traductores que convierten el programa fuente a programa objeto se llaman *compiladores* o *ensambladores*.

El *compilador* se utiliza cuando el programa fuente está codificado en un lenguaje de programación orientado al problema. Si el programa fuente está

codificado en un lenguaje de programación orientado a la máquina se utiliza el *ensamblador.*

Para efectuar un trabajo en un ordenador en primer lugar se almacena en la memoria el programa traductor y a continuación se introduce el programa simbólico para realizar un cálculo determinado. Este programa simbólico se traduce (ensambla o compila) a lenguaje máquina y después se ejecuta en lenguaje máquina.

Los programas traductores se sitúan en la memoria del ordenador antes de que se lea el programa simbólico utilizado.

Los programas traductores liberan al programador de la tarea de tener que asignar direcciones a las instrucciones y datos o modificar las instrucciones. Además el programador no tiene que preocuparse de las características particulares del ordenador que utilice, de la disposición de la memoria o del formato de las instrucciones de la máquina.

La utilización de lenguajes simbólicos implica simplificar la programación lo cual supone reducir los errores y facilitar la corrección de los mismos, así como aprender a programar en menos tiempo.

Un programa escrito en un lenguaje simbólico puede reducirse a pocas instrucciones ya que una instrucción en lenguaje simbólico equivale a varias instrucciones máquina.

Existen dos tipos de lenguajes de programación simbólicos:

- lenguajes ensambladores.
- lenguajes compiladores.

Los lenguajes ensambladores, llamados también lenguajes de bajo nivel, son lenguajes de programación *orientados a la máquina.*

Los lenguajes compiladores o de alto nivel son lenguajes de programación *orientados a los problemas.*

4.2.1. Lenguajes ensambladores

Los lenguajes ensambladores son lenguajes simbólicos de *bajo nivel* (es decir, son el nivel más bajo de los lenguajes simbólicos). Se les llama de bajo nivel porque están orientados al lenguaje máquina, es decir, las instrucciones de un lenguaje ensamblador son semejantes a las de un lenguaje máquina

aunque se utilicen determinados códigos o símbolos. Los símbolos de un lenguaje ensamblador (*assembler*) son generalmente abreviaturas del idioma inglés, y por lo tanto más fáciles de recordar y de interpretar que las sucesiones de unos y ceros de un lenguaje máquina.

Los lenguajes ensambladores varían de un fabricante a otro.

Para programar en un lenguaje ensamblador, las instrucciones se representan mediante una escritura mnemotécnica (con símbolos) y mediante números. Así, la parte operativa de una instrucción se representa utilizando símbolos para designar el tipo de operación, y en la parte direccional pueden utilizarse números u otros símbolos.

Por ejemplo, sea la instrucción: «restar Y al campo de memoria X».

$$X = X - Y$$

La representación mnemotécnica de esta instrucción será:

SUB	N

Siendo SUB la parte operativa y N la parte direccional (N es la dirección de la posición de memoria que contiene el valor Y).

En lenguaje máquina será (suponiendo que N sea el número 7):

0101	0111

Las instrucciones mnemotécnicas se traducen al código binario por medio de los programas traductores adecuados.

Por ejemplo, una instrucción en lenguaje ensamblador puede representarse como sigue:

C	5,A

que significa, «cargar A en la posición número 5 de la memoria».

Otra instrucción del mismo programa podría ser:

S	5,B

que significa «sumar B a A».

Una instrucción genera un cálculo o un movimiento de datos.

Cada instrucción se escribe en un código simbólico que posee un determinado valor mnemotécnico para el programador.

Estas instrucciones simbólicas deben ser traducidas (*ensambladas*) para que puedan ser interpretadas y ejecutadas por el ordenador. Es el mismo ordenador quién se encarga de esta traducción siguiendo las instrucciones de un programa codificado en binario llamado *programa ensamblador.*

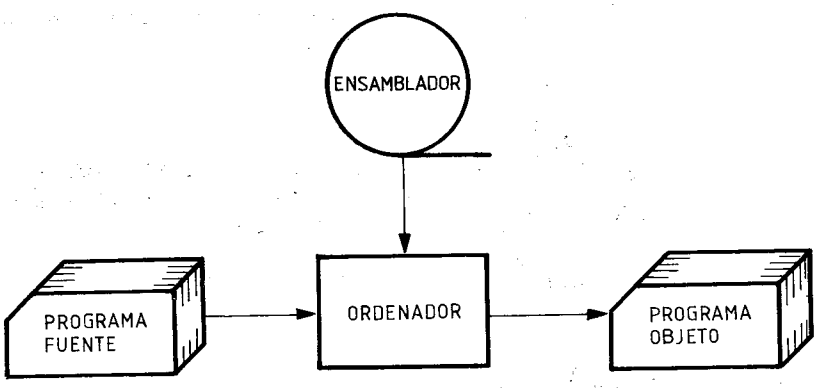


Fig. 4.1. El programa fuente *escrito en un lenguaje ensamblador es traducido mediante el* programa ensamblador *a otro programa ejecutable en forma binaria* (programa objeto).

4.2.2. Lenguajes compiladores

Los *lenguajes compiladores* son lenguajes simbólicos de *alto nivel* ya que están más orientados hacia los problemas que se han de resolver que hacia la máquina, por este motivo los lenguajes compiladores resultan independientes

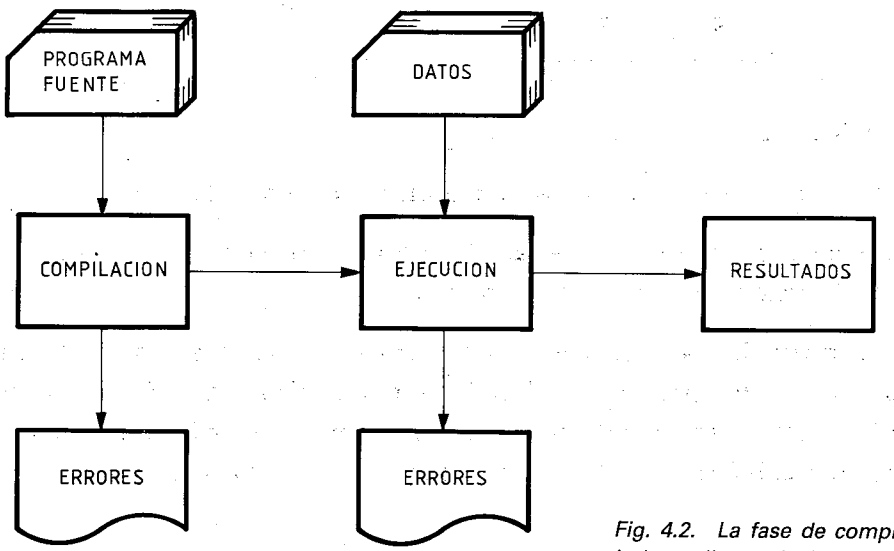


Fig. 4.2. La fase de compilación es independiente de la fase de ejecución. En ambas fases pueden producirse listados de errores de programación.

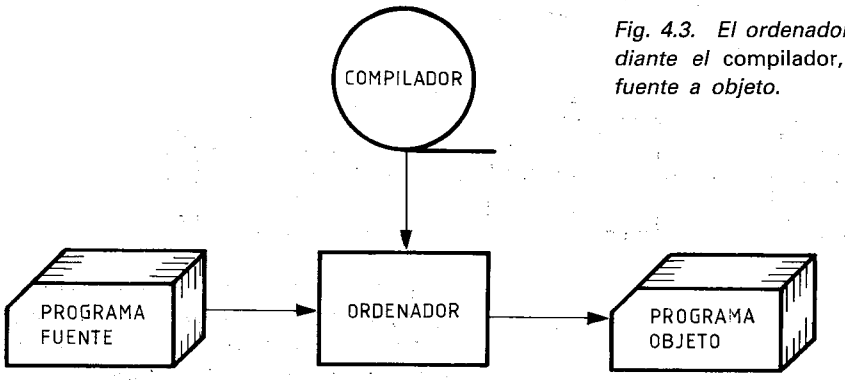


Fig. 4.3. El ordenador traduce, mediante el compilador, *el programa fuente a objeto.*

de la máquina, pudiéndose utilizar muchos de ellos en distintas marcas de ordenadores.

Los programas escritos en lenguajes compiladores son programas fuentes y deben traducirse al lenguaje máquina, con lo que pasan a ser programas objetos. En este caso, esta traducción se llama *compilación* y es efectuada por un programa llamado *programa compilador.*

Los lenguajes de alto nivel emplean símbolos fácilmente comprensibles por el ser humano (generalmente abreviaturas del inglés como en el caso de los lenguajes ensambladores).

Una instrucción de un lenguaje de alto nivel puede producir varias instrucciones en lenguaje máquina, sin embargo, con un lenguaje de bajo nivel cada instrucción genera una sola instrucción en lenguaje máquina.

Los lenguajes compiladores suelen ser independientes de la máquina en que se utilicen, ya que pueden utilizarse en distintas marcas de ordenadores.

Escribir un programa en uno de estos lenguajes orientados al problema resulta mucho más sencillo que codificar en un lenguaje orientado a la máquina.

Algunos de los lenguajes de alto nivel son los siguientes:

BASIC

El BASIC (Beginner's All-purpose Symbolic Instruction Code, que significa: «código de instrucciones simbólicas de uso general para principiantes») es un lenguaje de alto nivel apropiado para iniciarse en la programación y el cálculo.

El BASIC es un lenguaje de programación de alto nivel dirigido al programador y resulta muy adecuado para la comunicación mediante un terminal conectado a un ordenador a distancia.

En el Basic cada línea del programa empieza por un número (número de línea) que sirve para identificar la línea y para especificar el orden en que el ordenador debe realizar las operaciones.

Por ejemplo:

```
10 LET A = 4
20 LET B = B
30 LET X = A + B
40 PRINT X
50 END
```

Este programa calcula la suma A + B y pone el resultado en X y a continuación imprime X.

En Basic, los signos aritméticos se representan de la siguiente forma:

Operación	*Símbolo*
suma	+
resta	−
multiplicación	*
división	/
exponenciación	↑

Una ventaja del Basic, sobre otros lenguajes es que pueden escribirse expresiones mixtas. Así, por ejemplo:

```
40 PRINT 37 × 4 ↑ 2
```

Es decir, en una misma instrucción se representa la impresión, la multiplicación y la exponenciación.

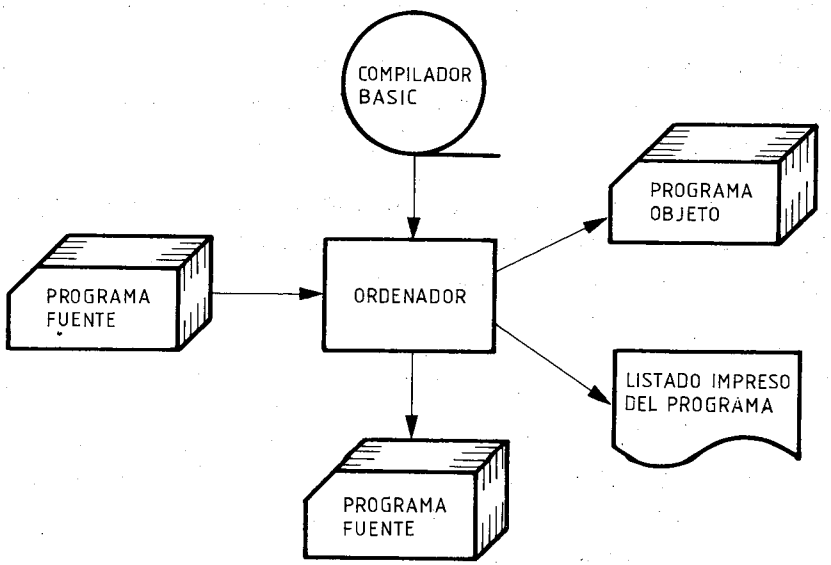


Fig. 4.4. En la fase de compilación se obtiene de nuevo el programa fuente, un listado del programa y el programa objeto.

COBOL

El COBOL (COmmon Business Oriented Language, «lenguaje orientado hacia los negocios») es un lenguaje que se usa para problemas comerciales.

Un programa en Cobol está dividido en cuatro partes llamadas *divisiones*:

1) IDENTIFICACION DIVISION, en donde se describe el programa.
2) ENVIRONMENT DIVISION, en donde se describe el ordenador que procesará el programa.
3) DATA DIVISION, en donde se describen los datos (formatos y ficheros de datos) que utiliza el programa.
4) PROCEDURE DIVISION, en donde se describen las operaciones que deben realizarse con los datos.

El Cobol es un lenguaje muy flexible a la hora de describir los datos o las operaciones, y es muy semejante al idioma inglés. Pero tiene el inconveniente de que es necesario escribir mucho a la hora de preparar un programa y el tiempo de compilación resulta más lento.

La principal aplicación del COBOL se encuentra en el tratamiento de ficheros.

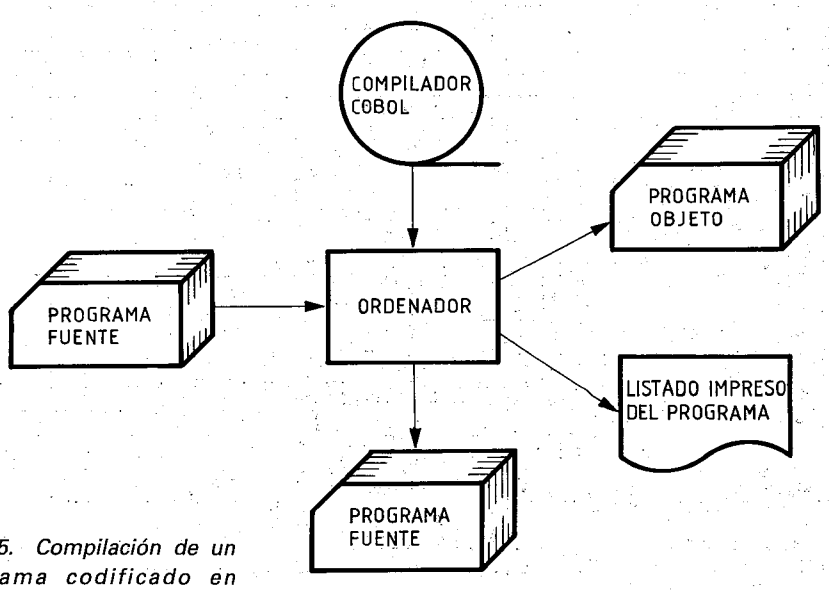


Fig. 4.5. Compilación de un programa codificado en COBOL.

IBM

Hoja de Codificación en COBOL

GX10-8028-1 (GX28-1464-5)
Impreso en España

Sistema		Instrucciones para Perforación		Hoja	De
Programa		Signo		Formato de Ficha	
Programado por	Fecha	Perforaciones			

Secuencia (Página)	Secuencia (Línea)	A	B	Declaración COBOL	Identificación
	01			IDENTIFICATION DIVISION.	
	02			PROGRAM-ID. EJEMPLO.	
	03			ENVIRONMENT DIVISION.	
	04			CONFIGURATION SECTION.	
	05			OBJECT-COMPUTER.	
	06			INPUT-OUTPUT SECTION.	
	07			SYSCHANØ1 IS OTRA-HOJA.	
	08			FILE-CONTROL.	
	09			SELECT FICHERO ASSIGN TO UNIVERSO.	
	10			DATA DIVISION.	
	11			FD FICHERO LABEL RECORD OMITTED DATA RECORD REGISTRO.	
	12			Ø1 REGISTRO.	
	13			Ø2 CAMPO-1 PIC 9.	
	14			WORKING-STORAGE SECTION.	
	15			77 CAMPO-2 PIC 9(3).	
	16			PROCEDURE DIVISION.	
	17			UNO. ACCEPT CAMPO-2.	
	18			MULTIPLY 3 BY CAMPO-2.	
	19			DISPLAY CAMPO-2.	
	20			STOP-RUN.	

Fig. 4.6. Hoja de codificación COBOL.

FORTRAN

el FORTRAN (FORmula TRANslation, «Traducción de fórmulas») se usa para problemas científicos.

El lenguaje FORTRAN original existe, aún en la actualidad, en algunas máquinas de capacidad limitada.

El lenguaje FORTRAN que actualmente se utiliza es el FORTRAN IV.

El FORTRAN IV admite subprogramas de llamada y de escritura. Para llamar a un subprograma utiliza la sentencia CALL (llamar).

Mientras que el COBOL utiliza el lenguaje de los negocios, el FORTRAN emplea el lenguaje matemático. Así, por ejemplo, una instrucción COBOL:

SUMAR *X* A *Y*

en FORTRAN se escribiría:

$$Y = Y + X$$

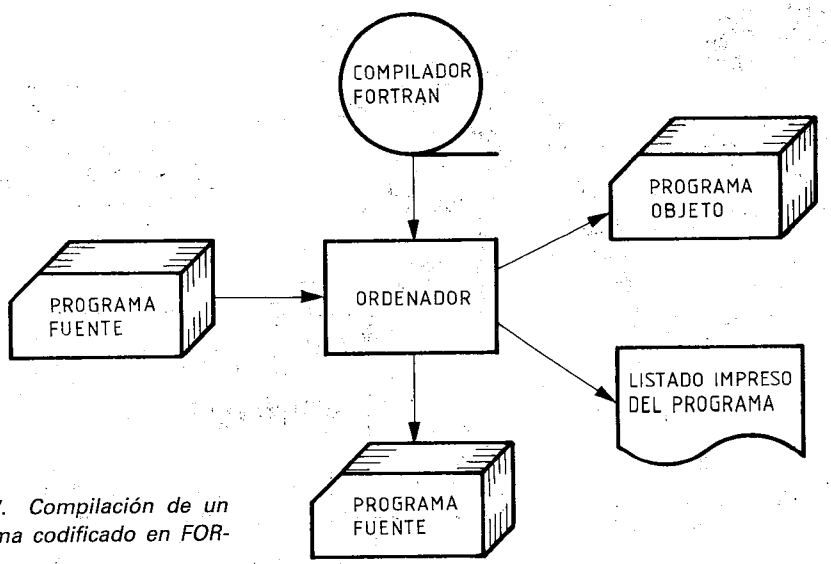


Fig. 4.7. Compilación de un programa codificado en FORTRAN.

Las ventajas del FORTRAN consisten en que se puede escribir y aprender con mucha rapidez y el tiempo de compilación es muy corto, produciendo en la compilación programas objeto muy eficientes.

Los programas en FORTRAN suele perforarse en fichas, de ahí que las hojas de codificación empleadas tengan 80 columnas. Cada línea codificada en el formulario (hoja de codificación FORTRAN) representa una ficha perforada.

PL/1

El lenguaje PL/1 (Programing Language One, «Lenguaje uno de programación») se utiliza para resolver problemas científicos y comerciales.

El PL/1 fue desarrollado por IBM para sus sistemas modelo 360. Puede ser utilizado en problemas sencillos o en problemas de cálculo muy complicados. Un programa PL/1 puede controlar las operaciones a cualquier nivel de complejidad o sencillez que se desee.

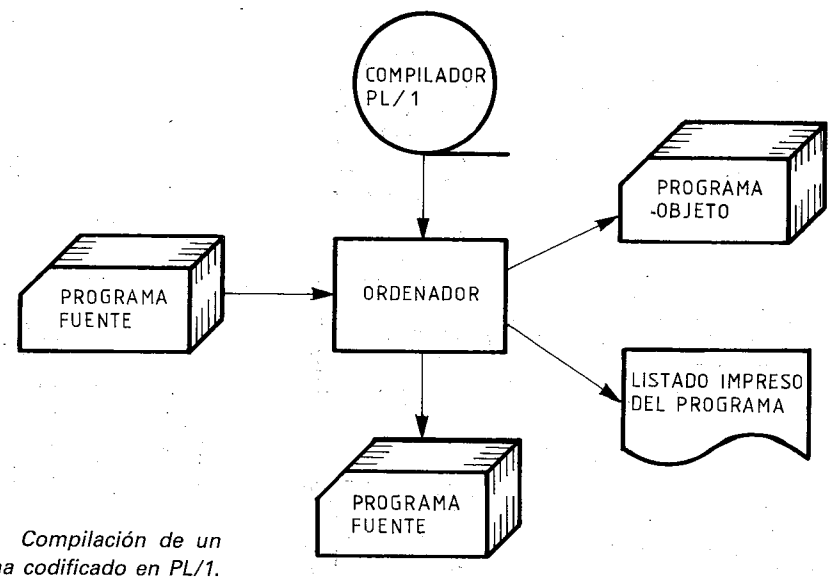


Fig. 4.8. Compilación de un programa codificado en PL/1.

El PL/1 es un lenguaje de sentencias fáciles de escribir y comprender por el programador, pero estas sentencias han de traducirse, como todos los lenguajes fuente, al lenguaje binario de la máquina. Esta traducción la efectúa el *compilador PL/1.*

ALGOL

El ALGOL (ALGOrithmic Language, «lenguaje algorítmico») es un lenguaje algorítmico orientado a los problemas, que se utiliza principalmente en la programación de problemas matemáticos.

En todos los lenguajes de programación existen dos tipos de palabras: *palabras reservadas* y *palabras optativas.*

Las palabras reservadas tienen una función específica definida por el lenguaje. Por ejemplo, en el lenguaje ALGOL son palabras reservadas: COMMENT (comentario), REAL (real) y END (final).

RPG

El lenguaje RPG (Report Program Generator, «programa generador de informes») se utiliza para producir informes o documentos destinados generalmente para la dirección de las empresas.

El RPG suele utilizarse para ejecutar cálculos e imprimir informes que podrían efectuarse mediante calculadoras mecánicas y máquinas de contabilidad. Con el RPG se automatizan todos estos procesos consiguiéndose mayor rapidez.

▶

Figs. 4.9. a 4.14. Ejemplo de un programa codificado en RPG. Para la codificación se necesitan 6 hojas, cada una de ellas con una determinada función. (En este ejemplo se barra la letra O).

IBM INTERNATIONAL BUSINESS MACHINES S.A.E.

Fórm.X21 – 9892 – 1
Impreso en España

GENERADOR AUTOMATICO DE PROGRAMAS (R. P. G.)

ESPECIFICACIONES DE LA FICHA DE CONTROL Y DE DESCRIPCION DE FICHEROS

Fecha _____

Programa _____

Programado por _____

INSTRUCCIONES DE PERFORACION: Signo | Perforaciones

Página 01

IDENTIFICACION DEL PROGRAMA (75 76 77 78 79 80)

ESPECIFICACIONES DE LA FICHA DE CONTROL

LINEA (3 4 5)	TIPO DE FORMULARIO (6)	DIMENSION DE MEMORIA PARA COMPILACION (7 8 9)	SALIDA OBJETO (10)	ORDENES DE LISTADO (11)	DIMENSION DE MEMORIA PARA EJECUCION (12 13 14)	DEBUG (Depuracion) (15)	SEC. DESCARGA NFCH (16)	ESTERLINAS ENTRADA CHELINES (17)	ESTERLINAS ENTRADA PENIQUES (18)	ESTERLINAS SALIDA CHELINES (19)	ESTERLINAS SALIDA PENIQUES (20)	PUNTUACION DECIMAL INVERTIDA (21)	360/20 ALM. INT. 2501 (22)	NUMERO DE POSICIONES DE IMPRESION (23 24 25)	SEC. INTER. ALTER. (26)	SOPORTE CONSULTA R.P.G. (37)	POSICIONAMIENTO FORMULARIOS (41)	ITABLAS] CONVERSION FICHEROS (42)	SUPRES. CEROS IZQUIERDA (43)	OPCION CAR. NO IMPRIMIBLE (44)	(45)	(46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74)
01	H											J										

ESPECIFICACIONES DE DESCRIPCION DE FICHEROS

LINEA (3 4 5)	TIPO DE FORMULARIO (6)	NOMBRE DEL FICHERO (7 8 9 10 11 12 13 14)	I/O/U/C/D (15)	TIPO DE FICHERO: P/S/C/R/T/D (16)	CLASE DE FICHERO: E (17)	FIN DE FICHERO: A/D (18)	SECUENCIA: F/V (19)	FORMATO (20)	LONGITUD DEL BLOQUE (20 21 22 23)	LONGITUD DEL REGISTRO (24 25 26 27)	TIPO DE PROCESO: L/R (28)	LONGITUD DEL CAMPO DE CLAVE O DE DIRECCION DEL REGISTRO: A/R/I (31)	TIPO DE DIRECCION: I/T/O 1-9 (32)	ORGANIZACION DEL FICHERO O AREA ADICIONAL DE E/S: INDICADOR OVERFLOW (33 34)	POSICION INICIAL DEL CAMPO DE LA CLAVE (35 36 37 38)	CODIGO EXTENSION E/L (39)	DISPOSITIVO (40 41 42 43 44 45 46)	NOMBRE SIMBOLICO DEL DISPOSITIVO (47 48 49 50 51 52)	ETIQUETAS (S. N. o E.) (53)	NOMBRE DE LA SALIDA PARA TRATAMIENTO DE ETIQUETAS (54 55 56 57 58 59)	SALIDA EXTENSION PARA DAM: POSICIONES DE MEMORIA DEL INDICE DE CILINDRO (60 61 62 63 64 65)	A/U (66)	ADICION AL FICHERO/ SIN ORDEN: NUMERO DE PISTAS PARA OVERFLOW DE CILINRO (67 68 69)	NUMERO DE EXTENSIONES (70)	N/U (71)	REBOBINADO DE CINTA (72)	COND.FICH U1 - U8 (73 74)
02	F	FFICHAB	I	P		A				96							MFCU1									
03	F	FFICHAT	C	S	E	A				96																
04	F	IMPRESØ	Ø							120				ØF			LPRINTER									
05	F																									
06	F																									
07	F																									

IBM

INTERNATIONAL BUSINESS MACHINES S.A.E.

GENERADOR AUTOMATICO DE PROGRAMAS (R.P.G.)

ESPECIFICACIONES ADICIONALES DE FICHEROS Y DEL CONTADOR DE LINEAS

Form. x21 - 9091 - 1
Impreso en España

Fecha _____

Programa _____

Programada por _____

INSTRUCCIONES DE PERFORACION	Signo							
	Perforaciones							

Página 02

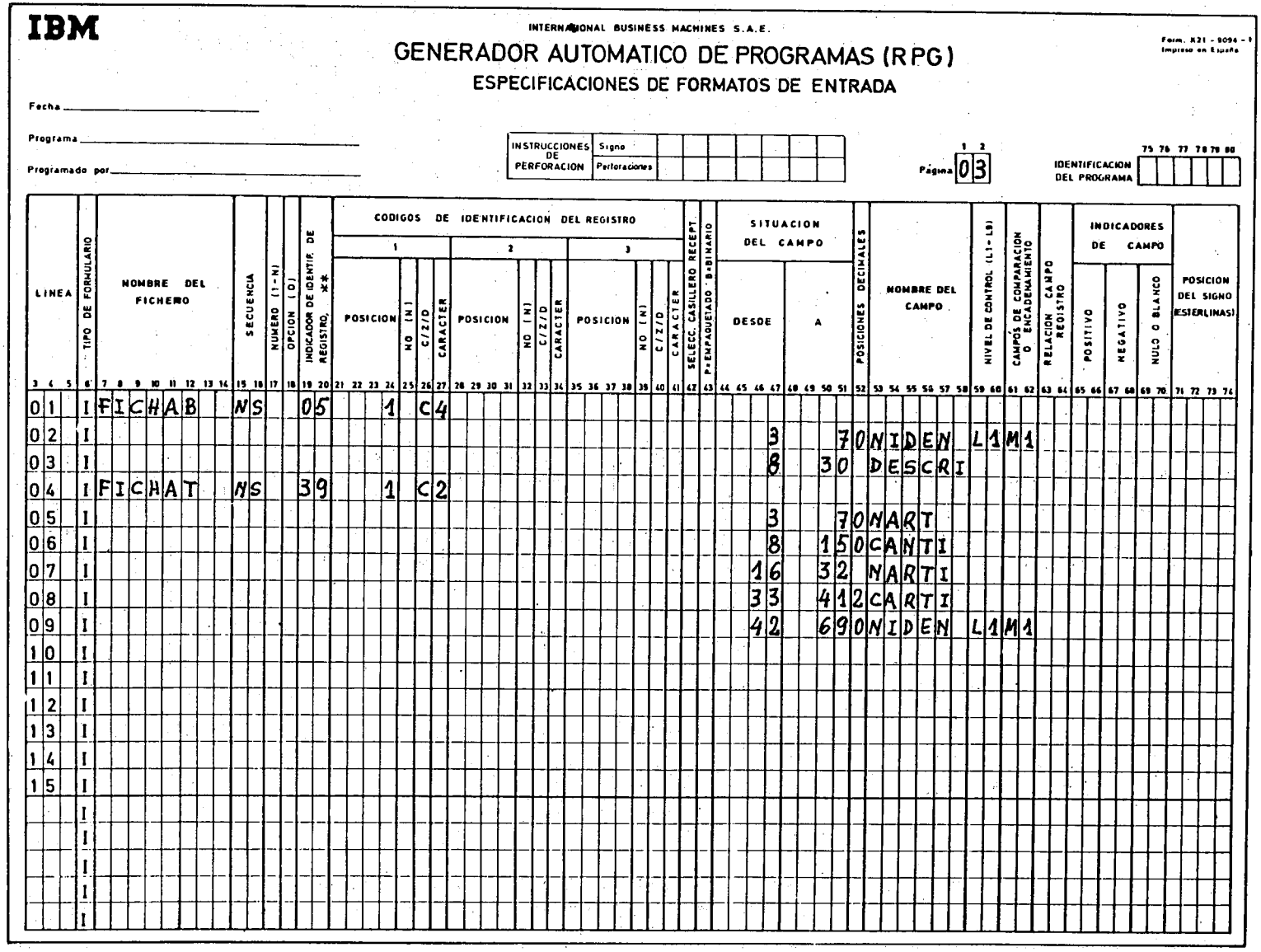
IDENTIFICACION DEL PROGRAMA

ESPECIFICACIONES ADICIONALES DE FICHEROS

LINEA	TIPO DE FORMULARIO	SECUENCIA DE REGISTROS DEL FICHERO DE ENCADENAMIENTO / NUMERO DEL CAMPO DE ENCADENAMIENTO / NOMBRE DEL FICHERO (DE)	NOMBRE DEL FICHERO (A)	NOMBRE DE LA TABLA O SERIE HOMOGENEA	NUMERO DE ENTRADAS POR REGISTRO	NUMERO DE ENTRADAS POR TABLA O SERIE HOMOGENEA	LONGITUD DE LA ENTRADA	P=EMPAQUETADO / B=BINARIO	POSICIONES DECIMALES	SECUENCIA DE LA TABLA (A/D)	NOMBRE DE LA TABLA O SERIE HOMOGENEA ALTERNATIVA	LONGITUD DE LA ENTRADA DE TABLA	P=EMPAQUETADO / B=BINARIO	POSICIONES DECIMALES	SECUENCIA DE LA TABLA (A/D)	COMENTARIOS
01	E															
02	E															
03	E															
04	E															
05	E															
06	E															
07	E															
08	E															
09	E															
10	E															

ESPECIFICACIONES DEL CONTADOR DE LINEAS

LINEA	TIPO DE FORMULARIO	NOMBRE DEL FICHERO	1 NUMERO DE LINEAS POR PAGINA	FL. O NUMERO DE CANAL	2 NUMERO DE LA LINEA DE "OVERFLOW"	OL O NUMERO DE CANAL	3 NUMERO DE LINEA	NUMERO DE CANAL	4 NUMERO DE LINEA	NUMERO DE CANAL	5 NUMERO DE LINEA	NUMERO DE CANAL	6 NUMERO DE LINEA	NUMERO DE CANAL	7 NUMERO DE LINEA	NUMERO DE CANAL	8 NUMERO DE LINEA	NUMERO DE CANAL	9 NUMERO DE LINEA	NUMERO DE CANAL	10 NUMERO DE LINEA	NUMERO DE CANAL	11 NUMERO DE LINEA	NUMERO DE CANAL	12 NUMERO DE LINEA	NUMERO DE CANAL
11	L	IMPRESØ	066	FL	056	ØL																				
12	L																									
13	L																									



IBM

INTERNATIONAL BUSINESS MACHINES S.A.E.

GENERADOR AUTOMATICO DE PROGRAMAS (RPG)

ESPECIFICACIONES DE FORMATOS DE ENTRADA

Form. X21 - 9096 - 1
Impreso en España

Fecha _____

Programa _____

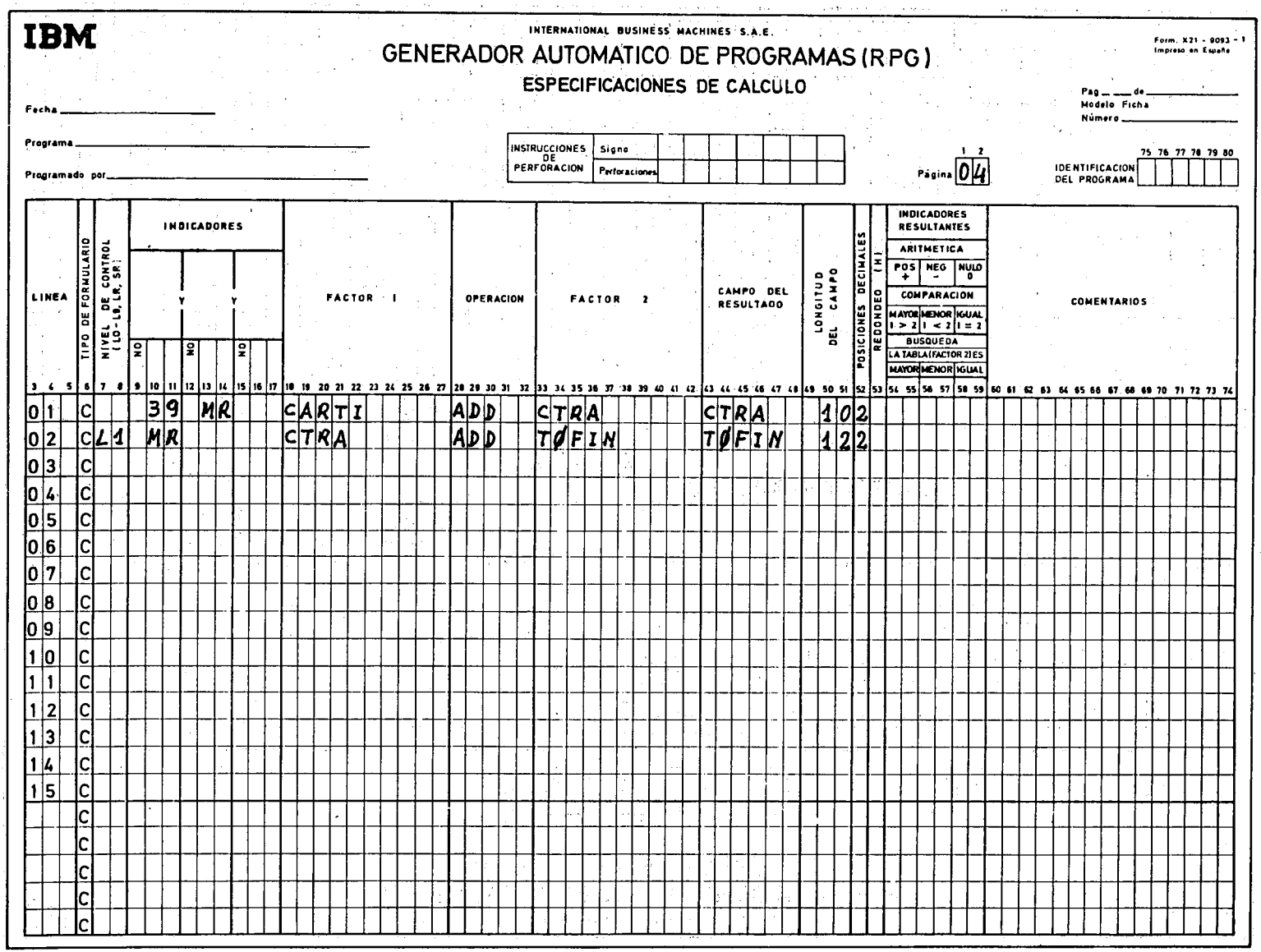
Programado por _____

INSTRUCCIONES DE PERFORACION: Signo | Perforaciones

Página 03

IDENTIFICACION DEL PROGRAMA

LINEA	TIPO DE FORMULARIO	NOMBRE DEL FICHERO	SECUENCIA	NUMERO (1-N)	OPCION (O)	INDICADOR DE IDENTIF. DE REGISTRO, **	CODIGOS DE IDENTIFICACION DEL REGISTRO 1: POSICION	1: NO (N)	1: C/Z/D	1: CARACTER	2: POSICION	2: NO (N)	2: C/Z/D	2: CARACTER	3: POSICION	3: NO (N)	3: C/Z/D	3: CARACTER	SELECC. CASILLERO RECEPT.	P=EMPAQUETADO B=BINARIO	SITUACION DEL CAMPO: DESDE	A	POSICIONES DECIMALES	NOMBRE DEL CAMPO	NIVEL DE CONTROL (L1-L9)	CAMPOS DE COMPARACION O ENCADENAMIENTO	RELACION CAMPO REGISTRO	INDICADORES DE CAMPO: POSITIVO	NEGATIVO	NULO O BLANCO	POSICION DEL SIGNO (ESTERLINAS)
01	I	FICHAB	NS			05	1		C	4																							
02	I																				3		70	NIDEN	L1	M1						
03	I																				8		30	DESCRI								
04	I	FICHAT	NS			39	1		C	2																							
05	I																				3		70	NART								
06	I																				8		150	CANTI								
07	I																				16		32	NARTI								
08	I																				33		412	CARTI								
09	I																				42		690	NIDEN	L1	M1						
10	I																																
11	I																																
12	I																																
13	I																																
14	I																																
15	I																																
	I																																
	I																																
	I																																
	I																																
	I																																



IBM

INTERNATIONAL BUSINESS MACHINES S.A.E.

GENERADOR AUTOMATICO DE PROGRAMAS (RPG)

ESPECIFICACIONES DE CALCULO

Form. X21 - 9093 - 1
Impreso en España

Fecha _____

Programa _____

Programado por _____

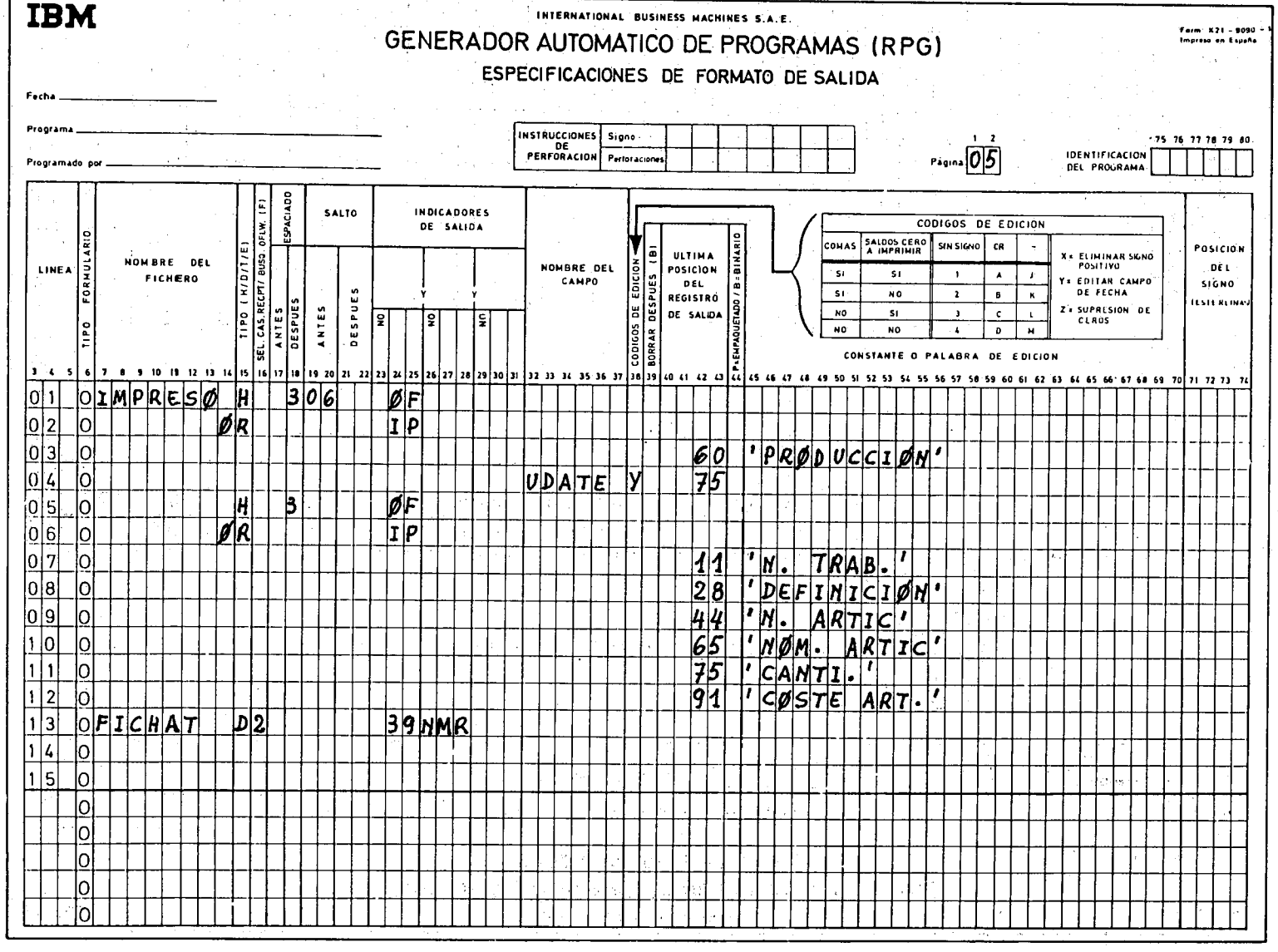
Pag ___ de _____
Modelo Ficha
Número _____

INSTRUCCIONES DE PERFORACION: Signo | Perforaciones

Página 04

IDENTIFICACION DEL PROGRAMA

LINEA	TIPO DE FORMULARIO	NIVEL DE CONTROL (LO-L9, LR, SR)	INDICADORES (NO / Y / Y / NO)	FACTOR 1	OPERACION	FACTOR 2	CAMPO DEL RESULTADO	LONGITUD DEL CAMPO	POSICIONES DECIMALES	REDONDEO (H)	INDICADORES RESULTANTES: ARITMETICA (POS +, NEG -, NULO 0); COMPARACION (MAYOR 1>2, MENOR 1<2, IGUAL 1=2); BUSQUEDA LA TABLA (FACTOR 2) ES (MAYOR, MENOR, IGUAL)	COMENTARIOS
01	C		39 MR	CARTI	ADD	CTRA	CTRA	102				
02	C	L1	MR	CTRA	ADD	TØFIN	TØFIN	122				
03	C											
04	C											
05	C											
06	C											
07	C											
08	C											
09	C											
10	C											
11	C											
12	C											
13	C											
14	C											
15	C											
	C											
	C											
	C											
	C											
	C											



IBM

INTERNATIONAL BUSINESS MACHINES S.A.E.

GENERADOR AUTOMATICO DE PROGRAMAS (RPG)

ESPECIFICACIONES DE FORMATO DE SALIDA

Form X21 - 9090 - 1
Impreso en España

Fecha _____

Programa _____

Programado por _____

INSTRUCCIONES DE PERFORACION: Signo | Perforaciones

Página 05

IDENTIFICACION DEL PROGRAMA

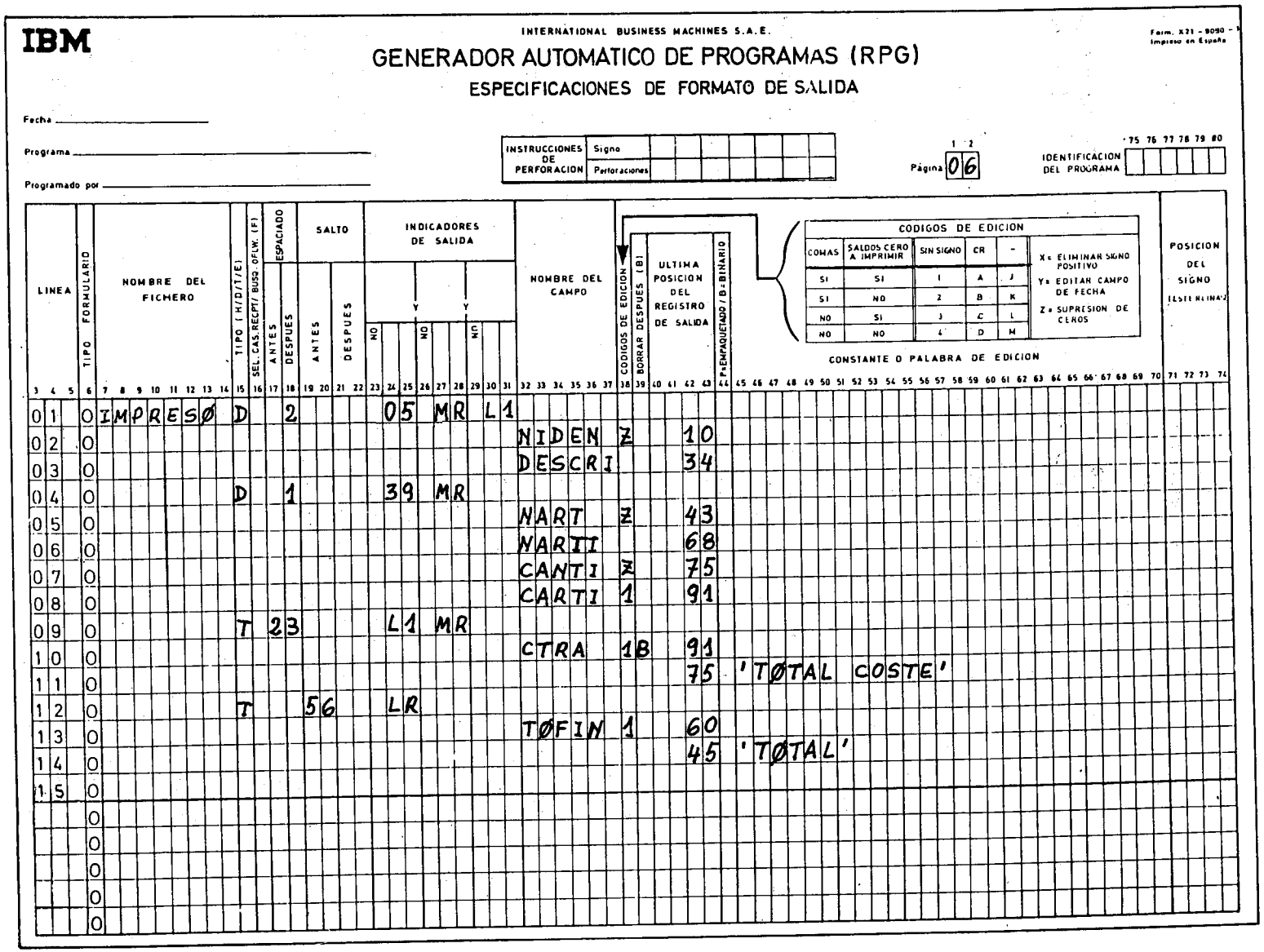
CODIGOS DE EDICION

COMAS	SALDOS CERO A IMPRIMIR	SIN SIGNO	CR	-
SI	SI	1	A	J
SI	NO	2	B	K
NO	SI	3	C	L
NO	NO	4	D	M

X = ELIMINAR SIGNO POSITIVO
Y = EDITAR CAMPO DE FECHA
Z = SUPRESION DE CEROS

CONSTANTE O PALABRA DE EDICION

LINEA	TIPO FORMULARIO	NOMBRE DEL FICHERO	TIPO (H/D/T/E)	SEL. CAS. RECPT/ BUSQ. OFLW. (F)	ESPACIADO ANTES	ESPACIADO DESPUES	SALTO ANTES	SALTO DESPUES	INDICADORES DE SALIDA	NOMBRE DEL CAMPO	CODIGOS DE EDICION	BORRAR DESPUES (B)	ULTIMA POSICION DEL REGISTRO DE SALIDA	P=EMPAQUETADO / B=BINARIO	POSICION DEL SIGNO (ESTE ULTIMO)
01	O	IMPRESØ	H			306			ØF						
02	O	Ø	R						IP						
03	O												60		'PRØDUCCIØN'
04	O									UDATE	Y		75		
05	O		H			3			ØF						
06	O	Ø	R						IP						
07	O												11		'N. TRAB.'
08	O												28		'DEFINICIØN'
09	O												44		'N. ARTIC'
10	O												65		'NØM. ARTIC'
11	O												75		'CANTI.'
12	O												91		'CØSTE ART.'
13	O	FICHAT	D	2					39NMR						
14	O														
15	O														
	O														
	O														
	O														
	O														
	O														



IBM

INTERNATIONAL BUSINESS MACHINES S.A.E.

Form. X21 - 8090 - 1
Impreso en España

GENERADOR AUTOMATICO DE PROGRAMAS (RPG)

ESPECIFICACIONES DE FORMATO DE SALIDA

Fecha _____

Programa _____

Programado por _____

INSTRUCCIONES DE PERFORACION: Signo | Perforaciones

Página 06

IDENTIFICACION DEL PROGRAMA

CODIGOS DE EDICION

COMAS	SALDOS CERO A IMPRIMIR	SIN SIGNO	CR	-
SI	SI	1	A	J
SI	NO	2	B	K
NO	SI	3	C	L
NO	NO	4	D	M

X = ELIMINAR SIGNO POSITIVO
Y = EDITAR CAMPO DE FECHA
Z = SUPRESION DE CEROS

CONSTANTE O PALABRA DE EDICION

LINEA	TIPO FORMULARIO	NOMBRE DEL FICHERO	TIPO (H/D/T/E)	SEL. CAS. RECPT/ BUSQ. OFLW. (F)	ESPACIADO ANTES	ESPACIADO DESPUES	SALTO ANTES	SALTO DESPUES	INDICADORES DE SALIDA	NOMBRE DEL CAMPO	CODIGOS DE EDICION	BORRAR DESPUES (B)	ULTIMA POSICION DEL REGISTRO DE SALIDA	P=EMPAQUETADO / B=BINARIO	CONSTANTE O PALABRA DE EDICION	POSICION DEL SIGNO (ESTE RELLENO)
01	O	IMPRESØ	D			2			05 MR L1							
02	O									NIDEN	Z		10			
03	O									DESCRI			34			
04	O		D			1			39 MR							
05	O									NART	Z		43			
06	O									NARTI			68			
07	O									CANTI	Z		75			
08	O									CARTI	1		91			
09	O		T		23				L1 MR							
10	O									CTRA	1B		91			
11	O												75		'TØTAL COSTE'	
12	O		T				56		LR							
13	O									TØFIN	1		60			
14	O												45		'TØTAL'	
15	O															
	O															
	O															
	O															
	O															
	O															

APL

El APL (A Programing Language, «Un Lenguaje de Programación») se introdujo en 1962 para resolver problemas científicos.

Es un lenguaje que proporciona la posibilidad de utilizar gran número de funciones y operadores simbólicos para escribir programas complicados de una forma muy sencilla y eficaz.

Existen otros lenguajes de programación menos utilizados que los anteriores. Ejemplo de tales lenguajes son:

JOVIAL

Jules Own Version of the International Algorithmic Language («Versión Jules del lenguaje algorítmico internacional»).

Es un lenguaje orientado hacia los problemas derivados del ALGOL.

SNOBOL

String-Oriented-Symbolic Language («lenguaje simbólico orientado hacia las series»). Este lenguaje tiene gran importancia en la generación de ecuaciones simbólicas y en la compilación de programas.

Una de las operaciones básicas del SNOBOL consiste en la formación de series.

Otros lenguajes parecidos al SNOBOL son el LISP, SLIP, COMIT e IPL-V, creados para el manejo de series secuenciales o proceso de listas. Estas series o listas de caracteres se procesan como símbolos y no como cantidades numéricas.

JOSS

El «Johnniac Open-Shop System» (sistema de libre acceso Johniac) es un lenguaje diseñado para ser utilizado por varias personas a la vez, que dispongan cada una de una máquina de escribir de consola (puede servir a más de 100 consolas).

Se desarrolló para realizar cálculos rápidos y complicados. Es un lenguaje algebraico semejante a otros lenguajes de este tipo.

COGO

COordinate GeOmetry = Geometría analítica.

Lenguaje orientado hacia problemas de ingeniería. Está basado en un vocabulario técnico.

SIMSCRIPT

Es un lenguaje de programación para la simulación (representación de un sistema operativo que suele hacerse mediante un *modelo matemático*).

El SIMSCRIPT es un lenguaje semejante al FORTRAN, que puede ser utilizado para reducir complicados modelos matemáticos a una serie de elementos más sencillos relacionados entre sí.

4.3. PROGRAMAS TRADUCTORES

Un ordenador no puede interpretar directamente los programas simbólicos.

Por lo tanto, un programa escrito en un lenguaje simbólico (*programa fuente*) es necesario traducirlo antes de que pueda ser ejecutado al lenguaje máquina (*programa objeto*). La figura 4.15 presenta, en esquema, todo este proceso.

Existen tres tipos de programas traductores de lenguajes:

- ensambladores.
- compiladores.
- interpretadores.

4.3.1. Ensambladores

Los símbolos utilizados por un lenguaje ensamblador no pueden ser «entendidos» por el ordenador, por consiguiente, cualquier programa escrito con un lenguaje de este tipo debe traducirse al lenguaje máquina mediante un programa traductor llamado *programa ensamblador* escrito también en lenguaje máquina.

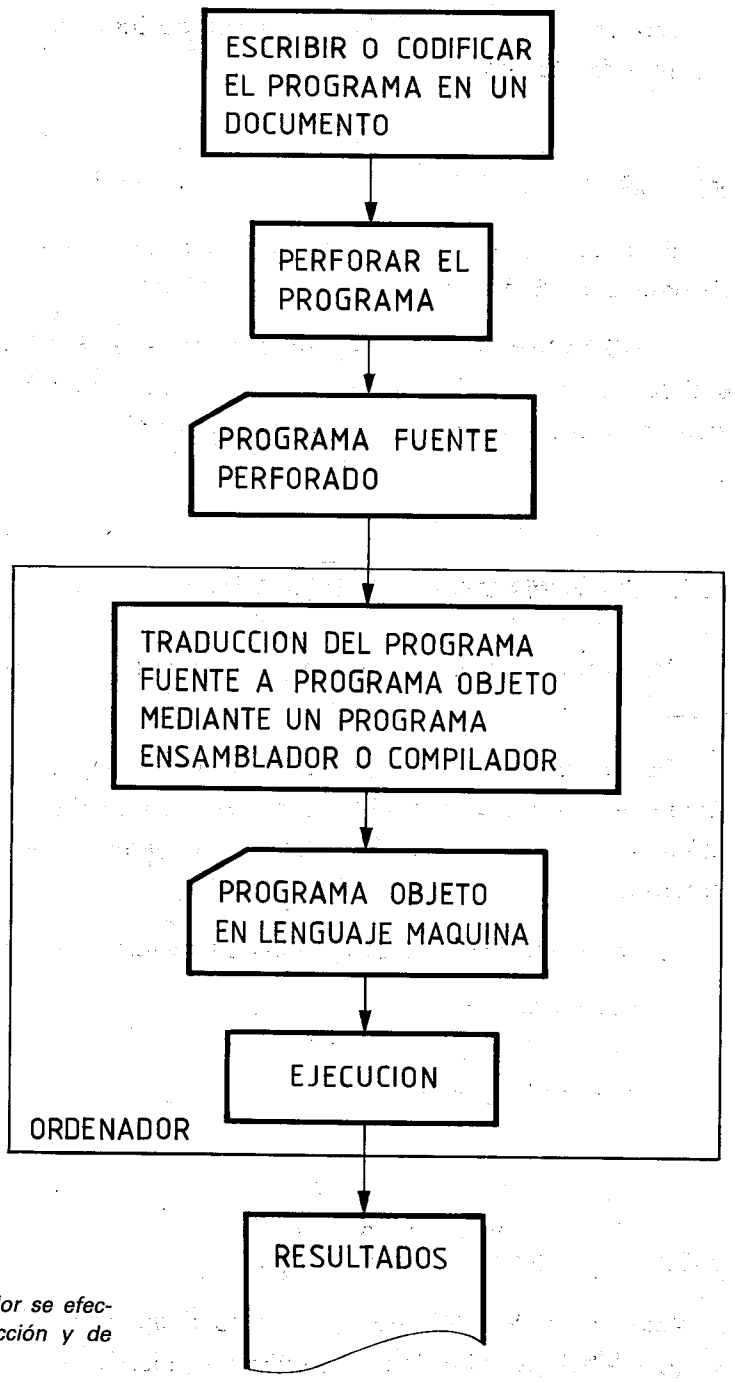


Fig. 4.15. En el ordenador se efectúan las fases de traducción y de ejecución del programa.

La operación de traducción de un lenguaje ensamblador a lenguaje máquina recibe el nombre de *ensamblaje.*

La operación de ensamblaje del programa fuente produce un programa ejecutable en binario (*programa objeto*) y además un listado impreso del programa.

Durante el proceso de ensamblaje el programa fuente constituye la entrada (*input*) del ensamblador, siendo la salida (*output*) el programa objeto.

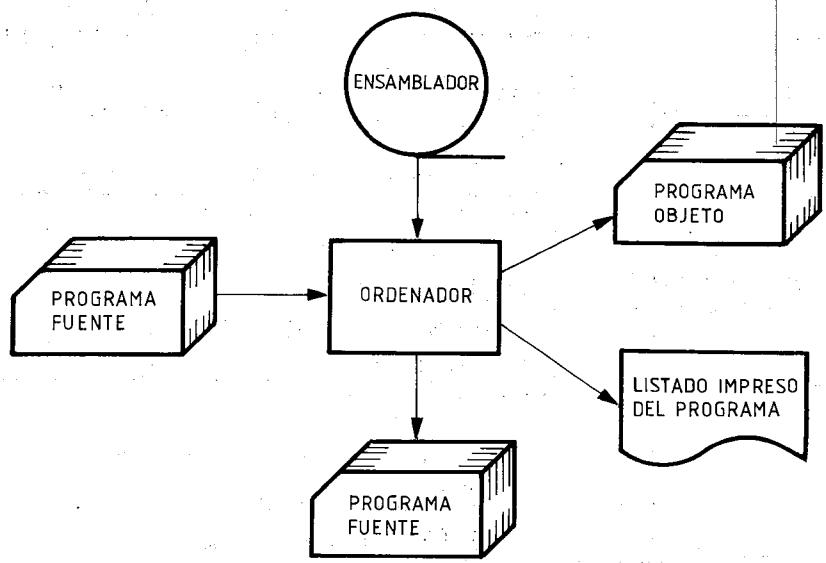


Fig. 4.16. Proceso de ensamblaje.

En un programa escrito en un lenguaje ensamblador, cada instrucción corresponde a una instrucción máquina. Entonces, un programa escrito en lenguaje ensamblador debería tener el mismo número de instrucciones que dicho programa escrito en lenguaje máquina, sin embargo, los lenguajes ensambladores añaden algunas instrucciones especiales llamadas, *seudoinstrucciones,* cuyo objetivo es simplemente transmitir información al ensamblador para comunicarle cómo debe realizar sus operaciones.

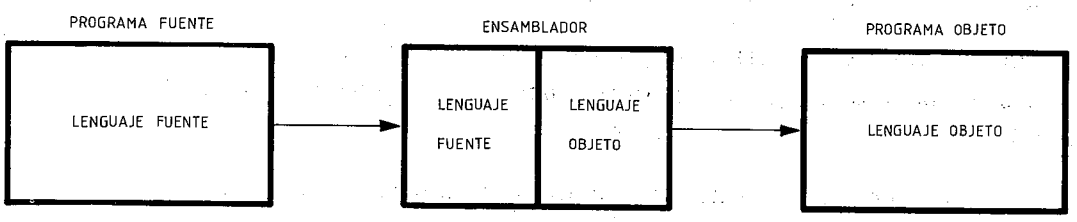


Fig. 4.17. El lenguaje fuente es el input *del ensamblador, siendo el* output *el lenguaje objeto.*

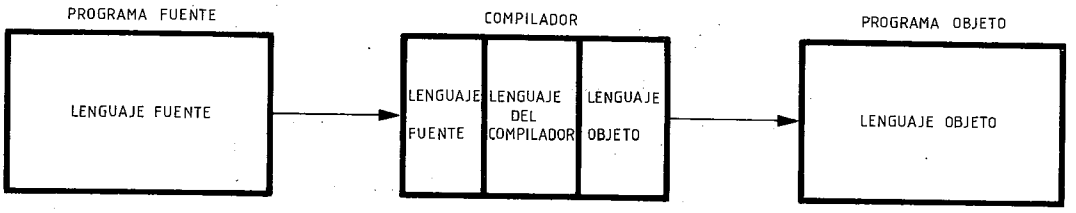


Fig. 4.18. En el proceso de compilación se traduce el lenguaje fuente al lenguaje del propio compilador y de éste al lenguaje objeto.

4.3.2. Compiladores

Para poderse ejecutar en el ordenador, los programas fuente escritos en un lenguaje compilador deben ser traducidos a lenguaje máquina, obteniéndose el programa objeto, mediante un programa traductor llamado *compilador.*

La operación de traducción mediante un compilador recibe el nombre de *compilación.*

Los programas fuente escritos en un determinado lenguaje compilador constituyen la entrada (input) del compilador. La salida (output) del compilador puede consistir en un programa escrito en un lenguaje de alto nivel o directamente en un lenguaje máquina.

Es evidente que si se desea obtener un programa para ser directamente ejecutado por el ordenador resulta más apropiado que el programa resultante

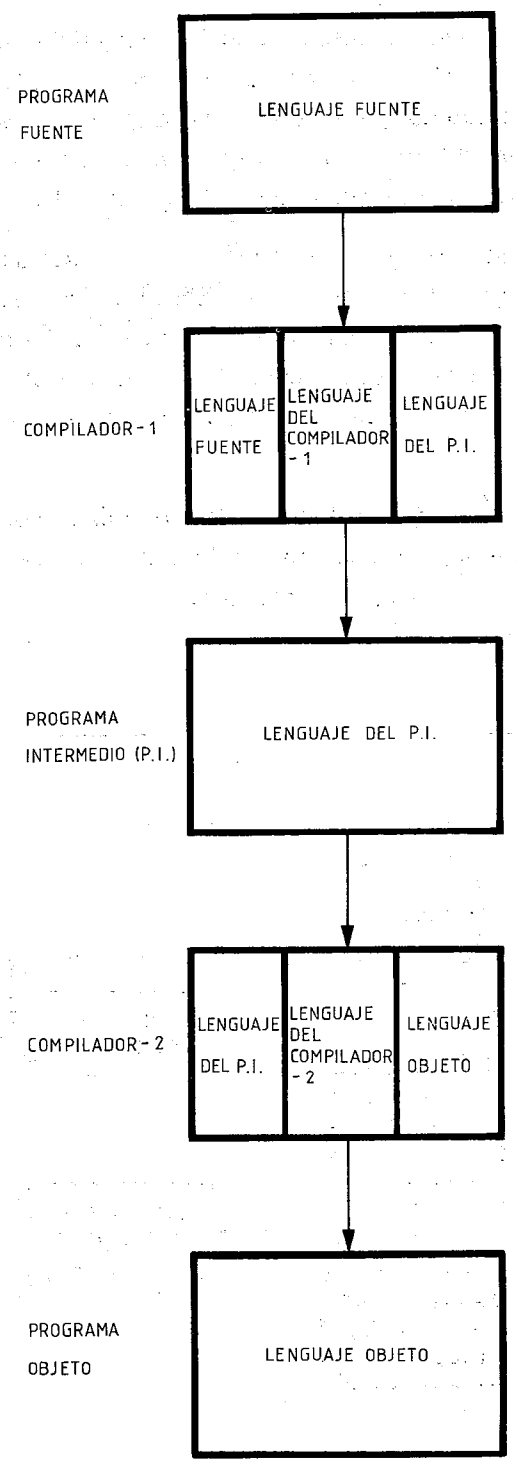


Fig. 4.19. La fase de compilación puede efectuarse en dos etapas utilizando dos compiladores.

de la compilación esté escrito en lenguaje máquina. Sin embargo, esta traducción puede efectuarse en dos etapas.

Por ejemplo, si el programa que se desea traducir está escrito en un determinado lenguaje de alto nivel, en lugar de realizar la traducción directamente (al lenguaje de la máquina que se utilice) porque no se dispone del compilador adecuado, puede utilizarse otro compilador para traducir el programa resultante de la primera compilación en el lenguaje máquina.

El lenguaje en que está escrito el programa compilador puede ser distinto al lenguaje fuente y al lenguaje objeto. Así como el ensamblador sólo se escribe en lenguaje máquina, el compilador puede escribirse en cualquier lenguaje, incluso de alto nivel. Sin embargo, suele escribirse en su lenguaje orientado a la máquina para obtener mayor aprovechamiento de la máquina y reducir redundancias.

Los lenguajes de alto nivel son independientes a las máquinas pero los programas objeto resultantes de la compilación no lo son, ya que cada ordenador suele diseñarse su propio lenguaje máquina.

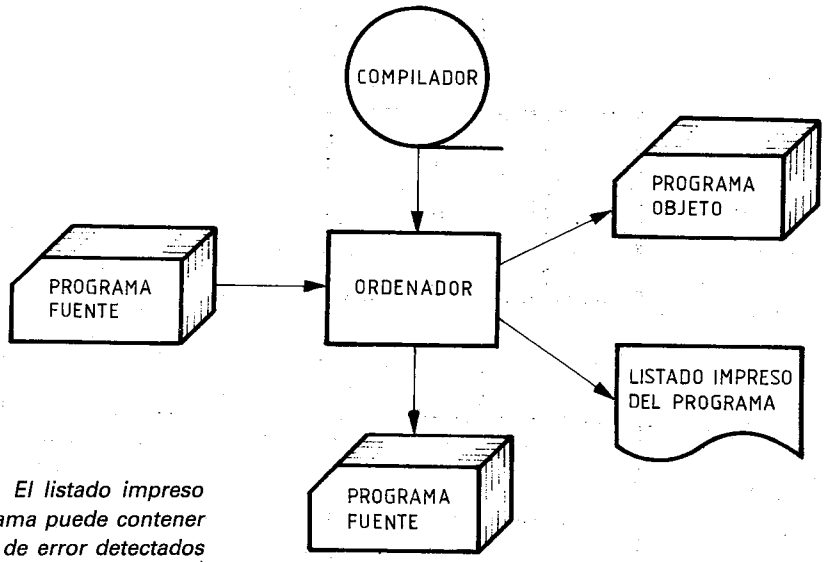


Fig. 4.20. El listado impreso del programa puede contener mensajes de error detectados por el compilador.

El compilador asigna las posiciones de memoria a los datos necesarios para el programa, interpreta las especificaciones del programa fuente y determina cuáles son las instrucciones de ordenador necesarias para la ejecución del programa.

Después de la fase de compilación, la salida del ordenador suele contener un programa en lenguaje máquina, llamado programa objeto, y un listado del programa fuente, junto con un esquema de la memoria en el que figuran los nombres y las posiciones de los campos de datos.

El listado impreso del lenguaje fuente puede contener mensajes indicando los errores de programación detectados por el compilador.

4.3.3. Interpretadores

Existen lenguajes de alto nivel, como el Basic y el APL, que pueden traducirse por medio de *interpretadores*.

Un *interpretador* es un programa que traduce y ejecuta cada instrucción del programa fuente antes de traducir y ejecutar la siguiente. Es decir, ejecuta cada instrucción a medida que las va traduciendo.

Cuando se está procesando un programa, el interpretador lee las instrucciones del programa y las conduce a la memoria.

El interpretador está escrito en lenguaje máquina.

4.4. COMPARACION ENTRE LENGUAJES

Los lenguajes de programación orientados a la resolución de problemas (lenguajes de alto nivel o lenguajes compiladores) poseen una serie de propiedades que los hacen superiores a los lenguajes orientados a la máquina (lenguajes máquina y lenguajes ensambladores).

Las ventajas de los lenguajes de programación orientados a los problemas son, entre otras, las siguientes:

- Aprendizaje en mucho menos tiempo. El programador no necesita efectuar una distribución de la memoria ni cambiar las direcciones de los datos en la memoria.

- Los lenguajes orientados a la máquina requieren unos conocimientos técnicos sobre los elementos que componen el sistema de proceso de datos. Estos conocimientos no son necesarios para programar en un lenguaje de alto nivel, ya que éstos están creados con independencia del sistema de proceso de datos. Es suficiente poseer ciertos conocimientos sobre el sistema de proceso de datos empleado.
- Los lenguajes orientados al problema ofrecen mayor claridad, ya que los programas fuente son mucho más cortos que una vez traducidos a programas objeto. Debido a la mayor claridad también se reduce el número de errores y la corrección de éstos resulta más sencilla, consiguiéndose menor número de pruebas del programa con lo que disminuye el tiempo de funcionamiento del sistema de proceso de datos.
- Los programas fuente codificados en lenguajes de alto nivel resultan más fáciles de leer por el propio programador y por las personas que no han realizado el programa.
- Un programa codificado en un lenguaje de alto nivel necesita pocas modificaciones al utilizarlo de un sistema de proceso de datos a otro. Suele ser compatible a las distintas marcas de sistemas de proceso de datos.

Los lenguajes de programación orientados al problema también tienen ciertos inconvenientes respecto a los lenguajes orientados a la máquina. Uno de estos inconvenientes consiste en que la elaboración de los programas traductores (compiladores) resulta muy costosa y complicada.

Sin embargo, hay que tener en cuenta que sólo se realizan una vez, por lo que a largo plazo los gastos no son tan considerables ya que pueden ser utilizados en cualquier sistema. En cuanto a la complejidad de su construcción, los usuarios no son quienes han de elaborarlos sino que vienen desarrollados de las casas constructoras.

Los programas objeto resultantes de la traducción de un programa fuente ocupan más posiciones de memoria que un programa escrito directamente al lenguaje máquina.

Los programas fuente escritos en un lenguaje orientado al problema no se adaptan de forma óptima a la máquina como puede hacerlo los programas escritos en un lenguaje orientado a la máquina, por lo tanto, los programas en lenguajes de alto nivel son más lentos.

Lenguaje de programación	*Instrucción*		*Programa fuente*	*Programa traductor*	*Programa objeto*
	Operando	*Dirección*			
– Lenguaje máquina	Binario	Binario	–	–	Instrucciones máquina
– Lenguaje ensamblador	mnemotécnica o simbólica	simbólica	Palabra-instrucción	Ensamblador	Instrucciones máquina
– Lenguaje compilador			Sentencias	Compilador	Instrucciones máquina

EJERCICIOS

1. ¿Cuál es la ventaja principal que presenta el lenguaje ensamblador respecto al lenguaje máquina?
   a) Total independencia de la máquina en que se utilice.
   b) Posee direcciones simbólicas lo cual permite al programador el no tener que efectuar la distribución de la memoria y realizar el cálculo de las direcciones.
   c) La representación en código binario, y que es el lenguaje que puede interpretar el ordenador.

2. Las instrucciones escritas en lenguaje máquina:
   a) deben ser compiladas a instrucciones ejecutables por el ordenador.
   b) deben ser ensambladas a instrucciones ejecutables por el ordenador.
   c) son interpretadas directamente por el ordenador.

3. El programa objeto está escrito en un lenguaje:
   a) máquina.
   b) ensamblador.
   c) compilador.

4. El programa traductor para pasar del programa fuente al programa objeto en los lenguajes de programación orientados a la máquina se llama:
   a) ensamblador.
   b) compilador.
   c) interpretador.

5. Para los problemas científicos son más adecuados los lenguajes de programación:
   a) FORTRAN, ALGOL y APL.
   b) PL/1, COBOL y BASIC.
   c) RPG, SNOBOL y LISP.

6. Los lenguajes simbólicos de alto nivel se llaman:
   a) lenguajes objeto.
   b) lenguajes ensambladores.
   c) lenguajes compiladores.

7. El lenguaje de alto nivel más apropiado para iniciarse en la programación y en el cálculo es el lenguaje:
   a) PL/1
   b) BASIC
   c) FORTRAN

8. Existe un lenguaje de programación cuyos programas están divididos en cuatro partes (*divisiones*):
   a) FORTRAN
   b) RPG
   c) COBOL

9. El lenguaje de programación más orientado a los problemas comerciales es:

   a) COBOL
   b) APL
   c) BASIC

10. Durante el proceso de traducción el programa que constituye el input del programa traductor se llama:

   a) Programa objeto.
   b) Programa fuente.
   c) Programa compilador.

11. Los programas interpretadores están escritos en:

   a) lenguaje máquina.
   b) lenguaje ensamblador.
   c) lenguaje compilador.

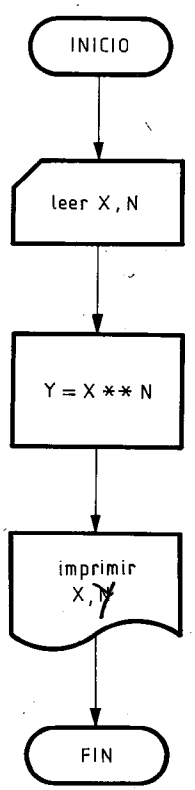
SOLUCIONES DE LOS EJERCICIOS

1. Etapas en la resolución de un problema

1. Aplicación; 2. Codificación del programa; 3. Hoja de codificación; 4. Compilador o ensamblador; 5. Secuencia de llamada; 6. *b*); 7. *a*); 8. *a*); 9. *b*); 10. *c*).

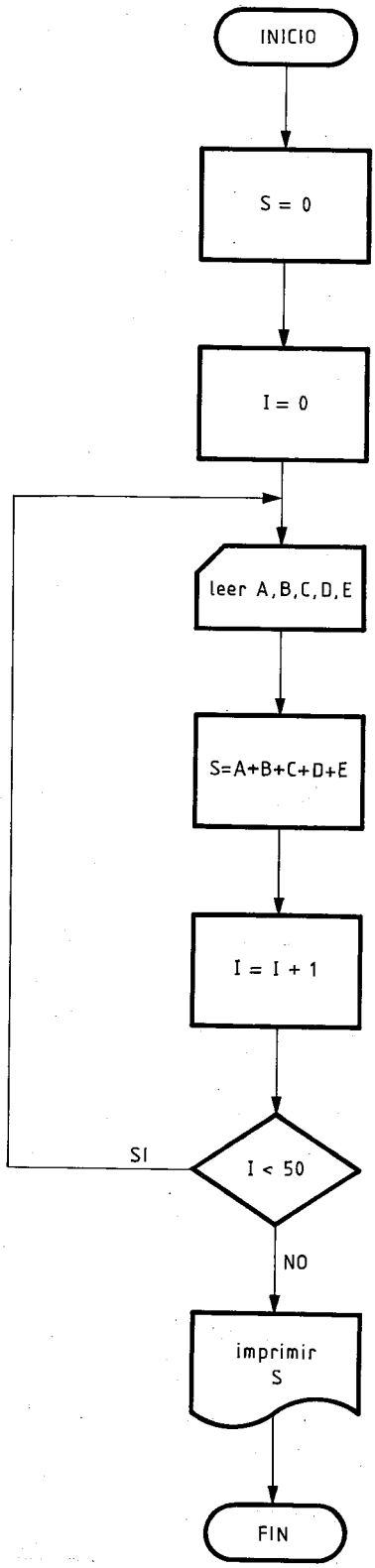
2. Organigramas

Solución al ejercicio 1



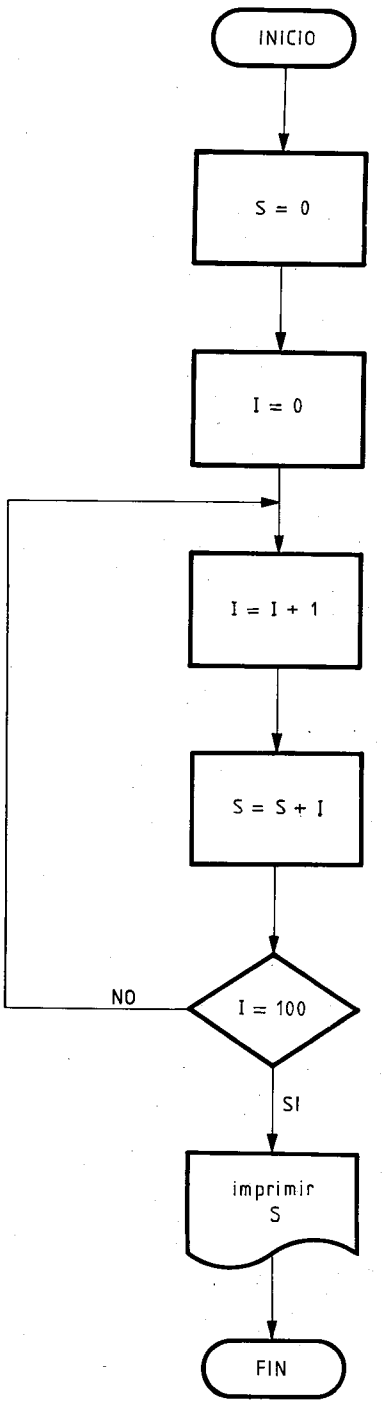
Organigramas

Solución al ejercicio 2



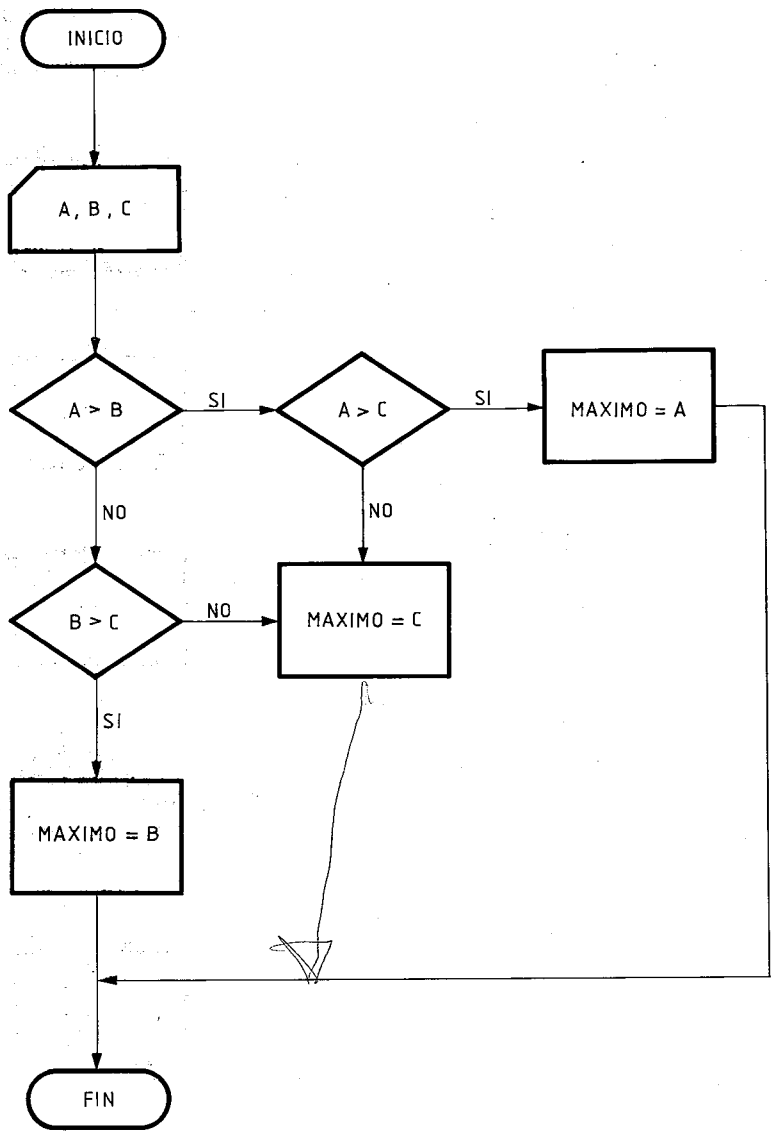
Organigramas

Solución al ejercicio 3



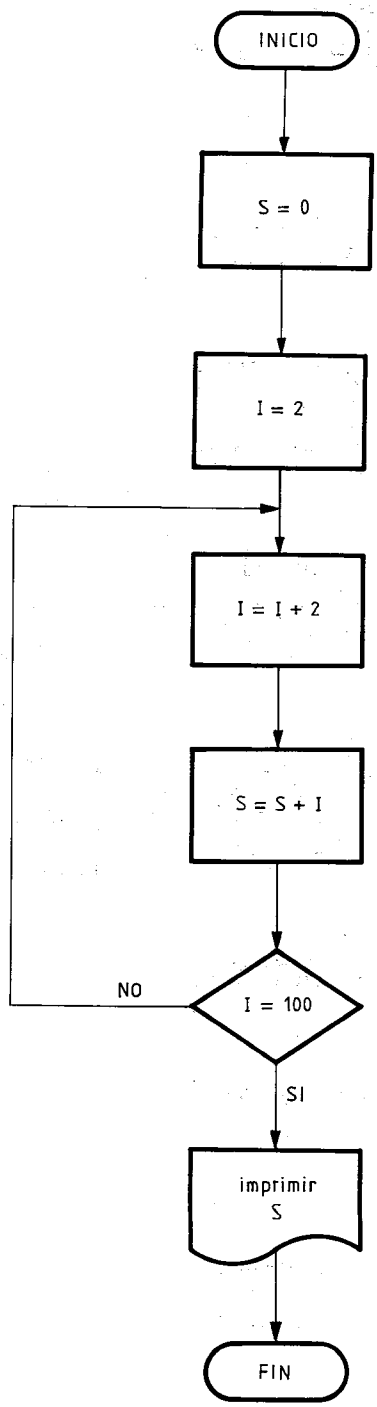
Organigramas

Solución al ejercicio 4



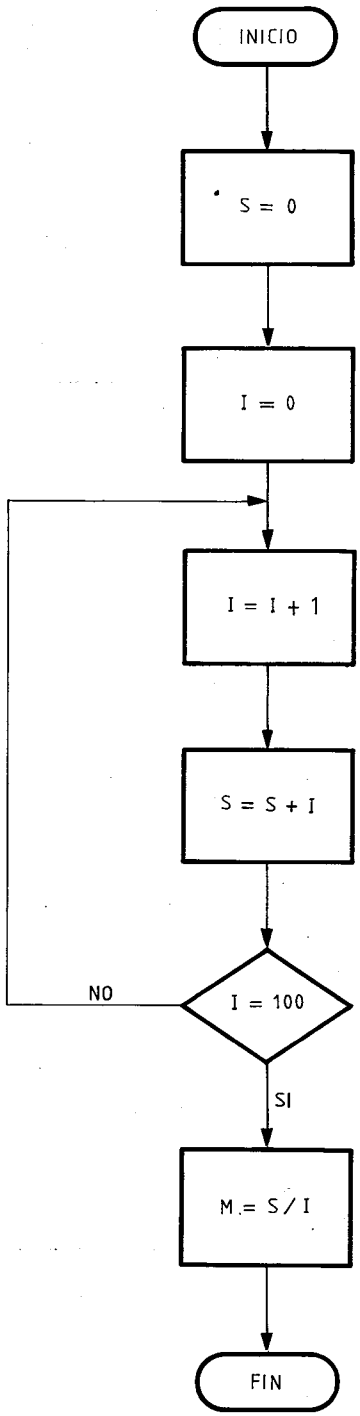
Organigramas

Solución al ejercicio 5



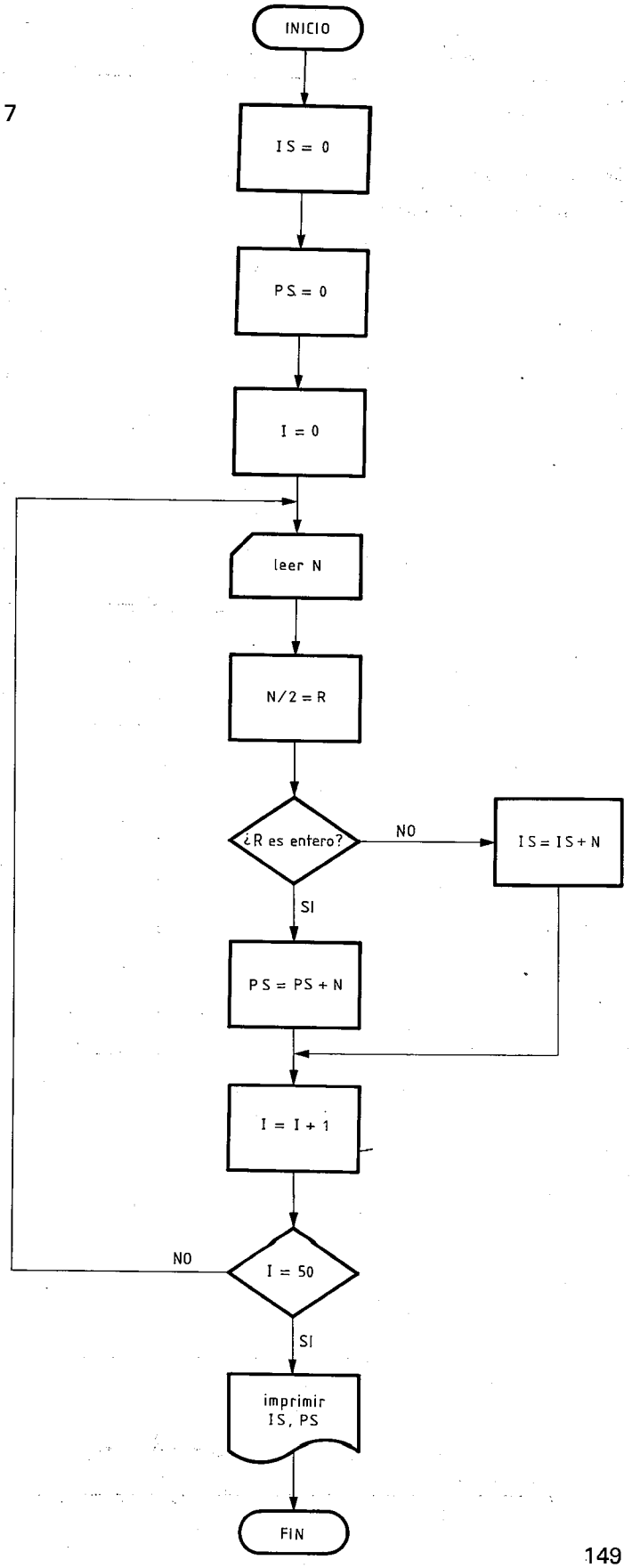
Organigramas

Solución al ejercicio 6



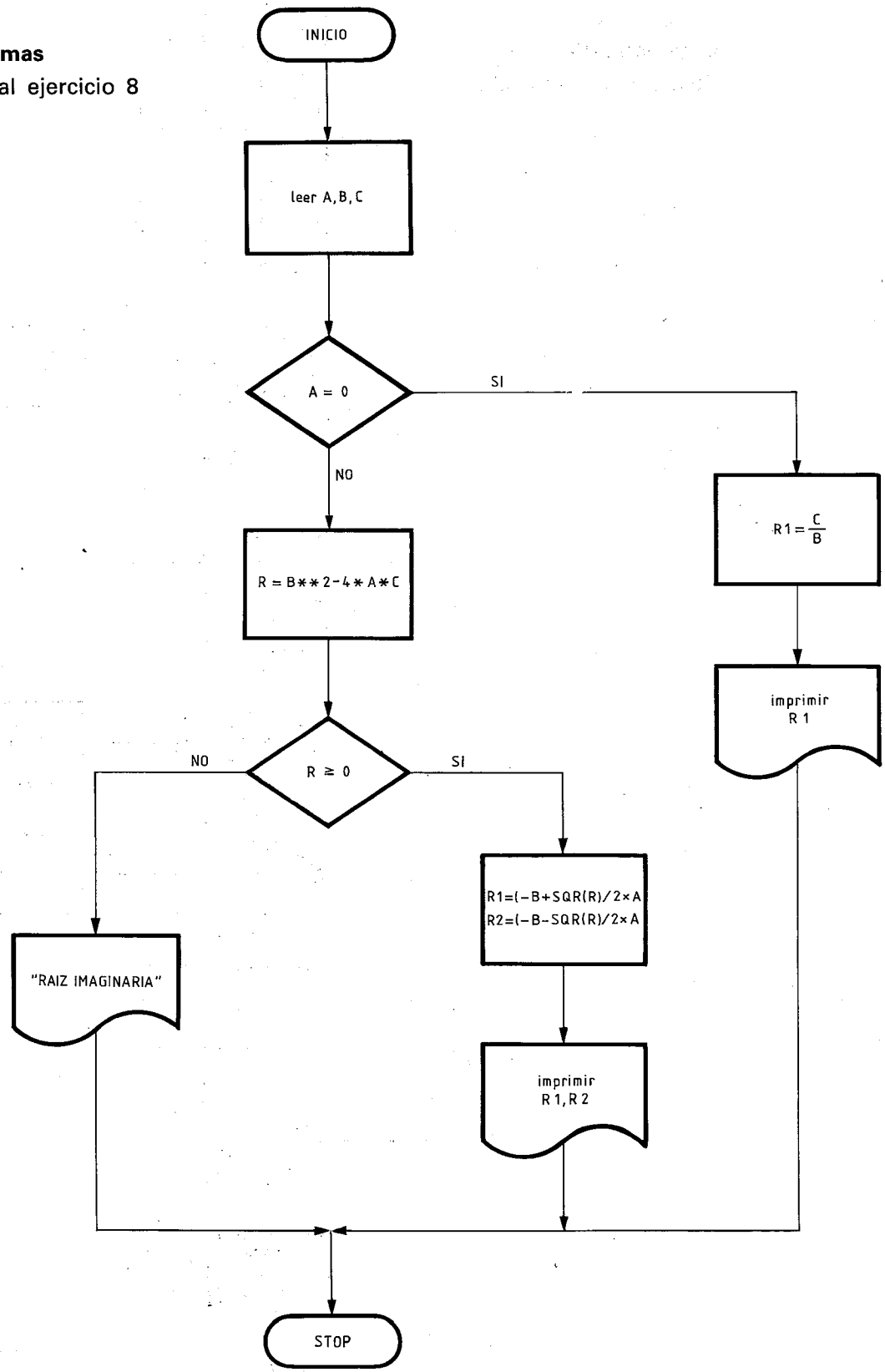
Organigramas

Solución al ejercicio 7



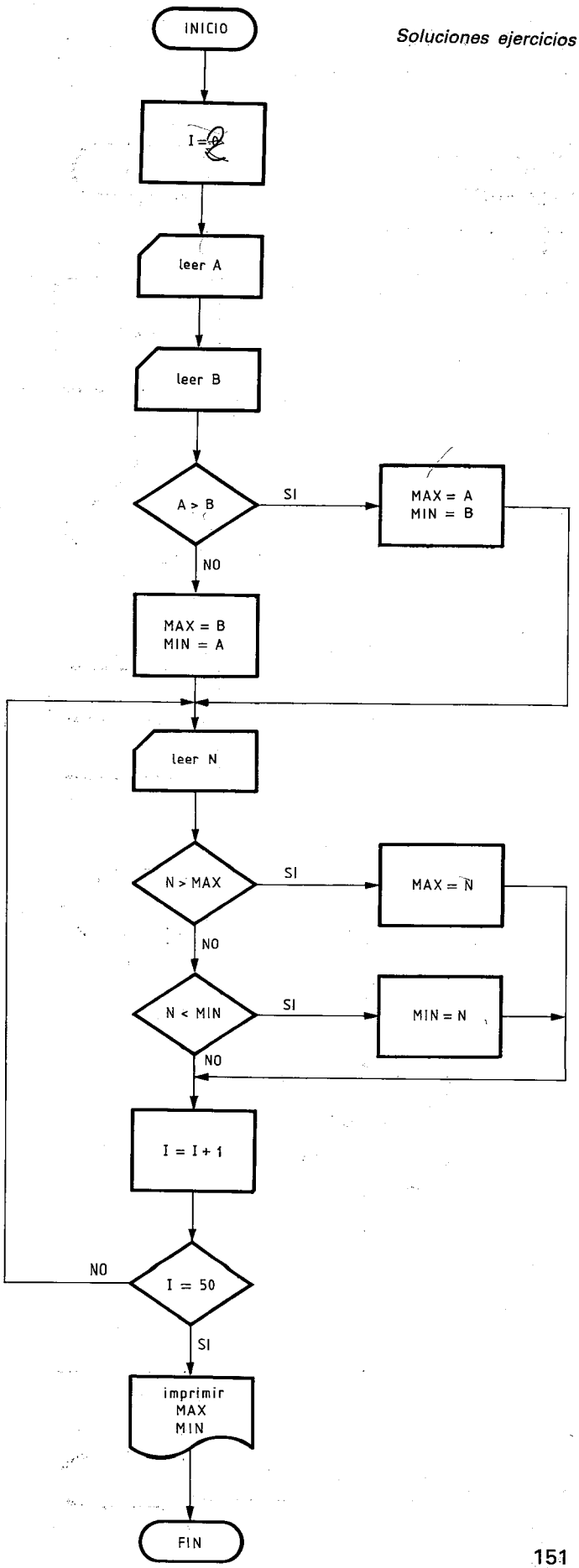
Organigramas

Solución al ejercicio 8



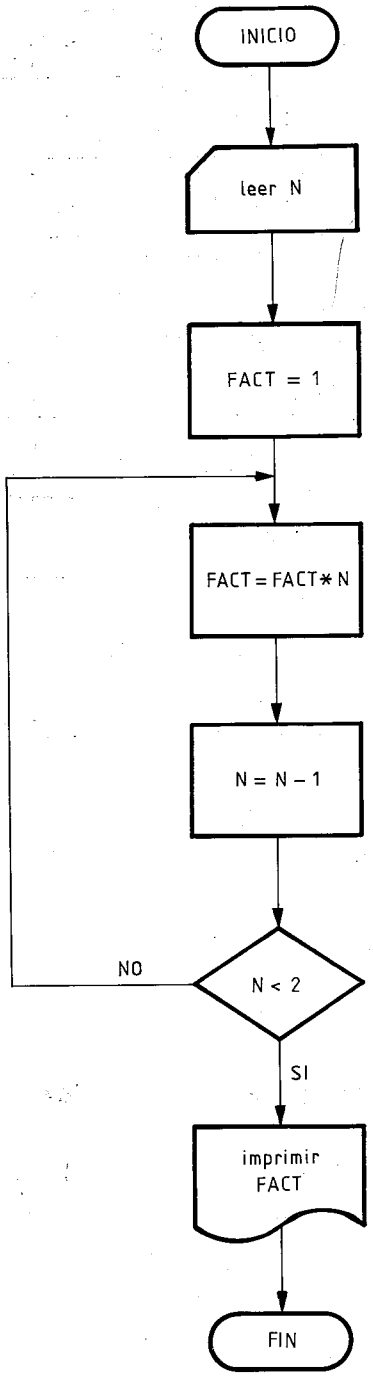
Organigramas

Solución al ejercicio 9



Organigramas

Solución al ejercicio 10



3. Tablas de decisión

1.

A > B	S	S	S	N	N	N
A > C	S	N	–	N	N	S
B > C	N	N	S	N	S	–
T1	X					
T2		X				
T3			X			
T4				X		
T5					X	
T6						X

2. *c*); 3. *a*); 4. *c*); 5. *c*).

4. Lenguajes de programación

1. *b*); 2. *c*); 3. *a*); 4. *a*); 5. *a*); 6. *c*); 7. *b*); 8. *c*); 9. *a*); 10. *b*); 11. *a*).

Indice General

Contenido 7
Introducción 9

1. ETAPAS EN LA RESOLUCION DE UN PROBLEMA

Cuadro de resumen 13
1.1. Definición del problema 15
1.2. Análisis del problema 16
1.3. Programación 18
1.3.1. Construcción del organigrama 19
1.3.2. Codificación 19
1.3.3. Depuración 24
1.3.4. Documentación 24
1.4. Ejecución del programa 25
1.4.1. Desarrollo lineal de un programa 27
1.4.2. Instrucciones de bifurcación hacia adelante 27
1.4.3. Instrucciones de bifurcación hacia atrás 28
1.4.4. Subprogramas 29
1.4.5. Macroinstrucciones 32
1.5. Evaluación de los resultados 32

Ejercicios 35

155

2. ORGANIGRAMAS

Cuadro resumen 39
2.1. Algoritmos 40
2.1.1. Algoritmo no numérico 40
2.1.2. Algoritmo numérico 41
2.2. Símbolos de los organigramas 42
2.3. Bucles 55
2.3.1. Contadores 63
2.3.2. Bucles anidados 63
2.4. Organigramas del sistema 64
2.5. Consideraciones sobre los organigramas 65
2.6. Ejemplos de organigramas 68

Ejercicios 76

3. TABLAS DE DECISION

Cuadro resumen 79
3.1. Estructura básica 80
3.2. Matriz de condiciones 83
3.3. Tratamientos 84
3.4. Reglas de decisión 84
3.5. Ejemplos 90
3.6. Conversión de tablas en organigramas 93

Ejercicios 102

4. LENGUAJES DE PROGRAMACION

Cuadro resumen 107
4.1. Lenguaje máquina 109
4.2. Lenguajes simbólicos 110
4.2.1. Lenguajes ensambladores 112
4.2.2. Lenguajes compiladores 114
— BASIC 116
— COBOL 118
— FORTRAN 120
— PL/1 121
— ALGOL 122
— RPG 122

— APL 129
— JOVIAL 129
— SNOBOL 129
— JOSS 129
— COGO 130
— SIMSCRIPT 130
4.3. Programas traductores 130
4.3.1. Ensambladores 130
4.3.2. Compiladores 133
4.3.3. Interpretadores 136
4.4. Comparación entre lenguajes 136

Ejercicios 139

Soluciones de los ejercicios 143

En este volumen se describen los fundamentos principales para la introducción de pleno en el campo de la Programación. Los conceptos expuestos aparecen de una forma progresiva que desemboca en la preparación del lector a un más profundo estudio de los campos específicos de la operativa informática.

El proceso de datos, instrumentos, partes, componentes, y relaciones entre todos ellos, vienen descritos de manera que su asimilación y entendimiento sea perfectamente asequible a toda persona que acompañe a la exposición con un mínimo de estudio, constancia e interés.

Al final de los capítulos se expone una serie de ejercicios que sirven para apreciar si la materia estudiada está perfectamente comprendida y el mismo lector puede juzgarlo mediante el solucionario que subsiguientemente se adjunta.

Una característica importante del libro es la subdivisión con que se presentan los temas más importantes que conduce a una exposición explícita y clara, siempre de gran ayuda para el lector.

El dibujo de la cubierta es cortesía de GENERAL DE COMPUTADORES, S.A.