

## COMPUTACION ELECTRONICA OCTUBRE-NOVIEMBRE 1982

1.     ING. LUIS CORDERO BORBOA  
      JEFE DEL DEPARTAMENTO DE COMPUTACION  
      EDIFICIO DE INGENIERIA MECANICA Y ELECTRICA  
      ANEXO DE INGENIERIA 1° PISO CUBICULO 11  
      U N A M  
      MEXICO, D. F.  
      550 52 15 EXT. 3746
  
2.     M. EN C. MARCIAL PORTILLA ROBERTSON (COORDINADOR)  
      PROGRAMA UNIVERSITARIO DE COMPUTACION  
      U N A M  
      EDIFICIO IIMASS  
      CUBICULO 13 P B  
      MEXICO, D.F.  
      550 52 15 EXT.4545
  
3.     ING. ROBERTO MANDUJANO WILD  
      INVESTIGADOR  
      FUNDACION ROSENBLUETH  
      HORAS NO. 738  
      COL. DEL VALLE  
      B. JUAREZ  
      03100 MEXICO, D.F.  
      524 90 05 y 534 24 87
  
4.     ING. HERIBERTO OLGUIN ROMO  
      JEFE DEL CENTRO DE CALCULO  
      FACULTAD DE INGENIERIA  
      UNAM  
      MEXICO, D.F.  
      550 52 15 EXT. 3734
  
5.     ING. RICARDO CIRIA MERCE  
      JEFE DE LA UNIDAD DE REGISTRO Y CONTROL ESCOLAR  
      UNAM  
      550 52 15 EXT.4150
  
6.     ING. ARMANDO REYES  
      550 52 15 Ext.4135

INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA 1982.

Fecha	Tema	Horario	Profesor
Octubre 8	INTRODUCCION A LA COMPUTADORA DIGITAL  Las unidades de entrada Las unidades de salida La memoria El control La unidad aritmética lógica	17 a 21 h	Ing.Armando Reyes
Octubre 9	LENGUAJE FORTRAN	9 a 14 h	Ing.Heriberto Olguín Romo e
Octubre 15			Ing. Ricardo Ciria Merce
Octubre 16	Constantes Variables y expresiones Proposiciones aritméticas Proporciones de entrada y salida Control de transferencia Variables con subíndice Proposiciones de iteración (DO) Formatos Subrutinas y funciones	9 a 14 h en la Fac. de Ing.	
Octubre 22	CASOS DE APLICACIONES MATEMATICOS	17 a 21 h	Ing. Luis Cordero Borboa
Octubre 23	Práctica, Facultad de Ingeniería, CU	9 a 14 h	Ings. Ricardo Ciria Merce Heriberto Olguín Romo
Octubre 29	CASOS DE APLICACIONES MATEMATICAS	17 a 21 h	Ing. Luis Cordero Borboa
Octubre 30	CASOS DE APLICACIONES NO MATEMATICAS	9 a 14 h	Ing. Roberto Mandujano Wild
Noviembre 5	UTILIZACION DE PAQUETES CLAUSURA	17 a 21 h	M. en I. Marcial Portilla Robertson
Noviembre 6	Práctica, Facultad de Ingeniería, CU.	9 a 14 h	Ing. Roberto Mandujano Wild

# EVALUACION DEL PERSONAL DOCENTE

①

**CURSO:** INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA

**FECHA:** Del 8 de Octubre al 6 de Noviembre de 1982.

		DOMINIO DEL TEMA	EFICIENCIA EN EL USO DE AYUDAS AUDIOVISUALES	MANTENIMIENTO DEL INTERES. (COMUNICACION CON LOS ASISTENTES, AMENIDAD, FACILIDAD DE EXPRESION).	PUNTUALIDAD	
<b>CONFERENCISTA</b>						
1.	ING. ARMANDO REYES					
2.	ING. HERIBERTO OLGUIN ROMO					
3.	ING. RICARDO CIRIA MERCE					
4.	ING. LUIS CORDERO BORBOA					
5.	ING. ROBERTO MANDUJANO WILD					
6.	M. EN I. MARCIAL PORTILLA ROBERTSON					
7.						
8.						
9.						
<b>ESCALA DE EVALUACION: 1 a 10</b>						

# EVALUACION DE LA ENSEÑANZA

②

SU EVALUACION SINCERA NOS AYUDARA A MEJORAR LOS PROGRAMAS POSTERIORES QUE DISEÑAREMOS PARA USTED.

TEMA	ORGANIZACION Y DESARROLLO DEL TEMA	GRADO DE PROFUNDIDAD LOGRADO EN EL TEMA	GRADO DE ACTUALIZACION LOGRADO EN EL TEMA	UTILIDAD PRACTICA DEL TEMA
INTRODUCCION A LA COMPUTADORA DIGITAL				
LENGUAJE FORTRAN				
CASOS DE APLICACIONES MATEMATICOS				
PRACTICA				
CASOS DE APLICACIONES MATEMATICOS				
CASOS DE APLICACIONES NO MATEMATICOS				
UTILIZACION DE PAQUETES				

ESCALA DE EVALUACION: 1 a 10

## EVALUACION DEL CURSO

CONCEPTO		EVALUACION
1.	APLICACION INMEDIATA DE LOS CONCEPTOS EXPUESTOS	
2.	CLARIDAD CON QUE SE EXPUSIERON LOS TEMAS	
	GRADO DE ACTUALIZACION LOGRADO CON EL CURSO	
4.	CUMPLIMIENTO DE LOS OBJETIVOS DEL CURSO	
5.	CONTINUIDAD EN LOS TEMAS DEL CURSO	
6.	CALIDAD DE LAS NOTAS DEL CURSO	
7.	GRADO DE MOTIVACION LOGRADO EN EL CURSO	

ESCALA DE EVALUACION DE 1 A 10

1. ¿Qué le pareció el ambiente en la División de Educación Continua?

MUY AGRADABLE	AGRADABLE	DESAGRADABLE

2. Medio de comunicación por el que se enteró del curso:

PERIODICO EXCELSIOR ANUNCIO TITULADO DI VISION DE EDUCACION CONTINUA	PERIODICO NOVEDADES ANUNCIO TITULADO DI VISION DE EDUCACION CONTINUA	FOLLETO DEL CURSO

CARTEL MENSUAL	RADIO UNIVERSIDAD	COMUNICACION CARTA, TELEFONO, VERBAL, ETC.

REVISTAS TECNICAS	FOLLETO ANUAL	CARTELERA UNAM "LOS UNIVERSITARIOS HOY"	GACETA UNAM

3. Medio de transporte utilizado para venir al Palacio de Minería:

AUTOMOVIL PARTICULAR	METRO	OTRO MEDIO

4. ¿Qué cambios haría usted en el programa para tratar de perfeccionar el curso?

---



---



---

5. ¿Recomendaría el curso a otras personas?

SI	NO

6. ¿Qué cursos le gustaría que ofreciera la División de Educación Continua?

---



---

7. La coordinación académica fue:

EXCELENTE	BUENA	REGULAR	MALA

8. Si está interesado en tomar algún curso intensivo ¿Cuál es el horario más conveniente para usted?

LUNES A VIERNES DE 9 A 13 H. Y DE 14 A 18 H. (CON COMIDAS)	LUNES A VIERNES DE 17 A 21 H.	LUNES, MIÉRCOLES Y VIERNES DE 18 A 21 H.	MARTES Y JUEVES DE 18 A 21 H.

VIERNES DE 17 A 21 H. SABADOS DE 9 A 14 H.	VIERNES DE 17 A 21 H. SABADOS DE 9 A 13 Y DE 14 a 18 H.	O T R O

9. ¿Qué servicios adicionales desearía que tuviese la División de Educación Continua, para los asistentes?

---



---

10. Otras sugerencias:

---



---



---



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

**INTRODUCCION A LA PROGRAMACION Y  
COMPUTACION ELECTRONICA**

**INTRODUCCION A LA COMPUTADORA DIGITAL**

**OCTUBRE, 1982**



## CONCEPTO DE COMPUTADORA

### OBJETO :

El objeto de esta breve reseña sobre los computadores electrónicos y sus múltiples aplicaciones al servicio del hombre, es transmitir al lector una completa visión de conjunto, mediante un lenguaje sencillo que permita comprender conceptualmente los temas tratados, sin necesidad de conocimientos previos en la materia.

Esperamos que estas páginas, muy simples en apariencia pero con profundo contenido, permitan, a quienes las lean, ingresar al maravilloso mundo de las máquinas automáticas.



Este señor se llama Control. Trabaja en una pequeña habitación. Tiene a su disposición una máquina de calcular que suma, resta, multiplica y divide. Tiene también el señor Control un archivo parecido al casillero que existe en los trenes para clasificación postal.

Hay, además, en la habitación, dos ventanillas identificadas con sendos carteles: "Entrada" y "Salida".

El señor Control tiene un manual que le indica cómo debe desenvolverse con estos elementos, si alguien le pide que haga un trabajo.



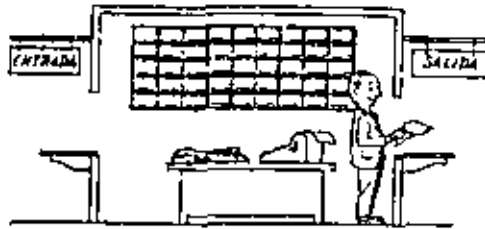
Una persona quiere saber el resultado de un complicado cálculo. Para ello, escribe ordenado, preciso y detalladamente, cada una de las operaciones que, en conjunto, integran ese cálculo, anota cada instrucción elemental en una hoja de papel y coloca todas las hojas en orden en la ventanilla "Entrada".

El señor Control, al ver las hojas, lee en su manual que debe tomar esas hojas con instrucciones, una por una, y colocarlas convenientemente en su archivo. Y así lo hace.

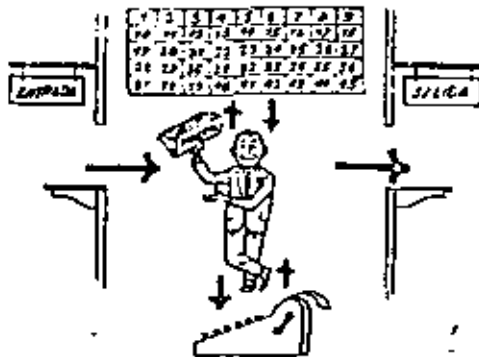


Una vez ubicadas todas las instrucciones en el archivo, el señor Control consulta nuevamente el manual. Allí se le indica que, a continuación, debe tomar la instrucción de la casilla 1 y ejecutarla luego, la de la casilla 2 y ejecutarla, y así sucesivamente hasta ejecutar la última instrucción. Algunas instrucciones indicarán que hay que sumar una cantidad a otra (instrucciones aritméticas); otras, que el señor Control debe ir a la ventanilla "Entrada" para buscar algún dato que intervenga en el cálculo — (instrucciones de "entrada/salida"), dato que la persona que le formuló el problema habrá colocado ya en dicha ventanilla, en otro hoja de papel.

Finalmente, otras instrucciones indicarán que debe elegirse una de entre dos alternativas (instrucciones lógicas); por ejemplo, — supongamos que una parte del cálculo — dada la instrucción que está en la casilla 5 del archivo hasta la que está en la casilla 9 debe ejecutarse 15 veces porque el cálculo así lo exige. En tal caso, la instrucción que está en la casilla 10 indicará que, si los pasos 5 a 9 se han ejecutado menos de 15 veces, se debe volver al paso 5. Cuando se hayan realizado los 15 repeticiones y no antes, el señor Control seguirá con la instrucción de la casilla 11.



Después de ejecutar todas las instrucciones del archivo, haciendo con la máquina de calcular las operaciones en ellas indicadas, el señor Control entrega, a través de la ventanilla "Salida", los resultados obtenidos . . . y se sienta a esperar un nuevo trabajo.



Obsérvese que la actuación del señor Control es puramente mecánica: sólo sigue las indicaciones de su . . . y cumple de acuerdo con ellas las instrucciones que recibe a través de la ventanilla "Entrada". Toma decisiones, pero solamente cuando se le señalan las alternativas que existen y con qué criterio debe elegir una de ellas.

El señor Control puede resolvernos cualquier problema, por complicado que éste sea. Pero para ello debemos indicarle paso a paso, en la forma más elemental y detallada, todo lo que debe hacer para resolverlo, sin olvidarnos absolutamente nada porque, en ese caso, el señor Control no sabría continuar por sí mismo.

Haga el . . . la prueba de formular un problema cualquiera de modo tal que una persona que no renata nada acerca de ese problema pueda resolverlo sin necesidad de hacer consultas. Vaya que es una experiencia interesante!

El esquema que acabamos de representar mediante el señor Control y sus elementos de trabajo, corresponde exactamente al esquema de funcionamiento de una computadora electrónica.

A continuación presentaremos una breve descripción de los elementos de la computadora que corresponden a los elementos de trabajo del señor Control.

Las unidades de entrada (representadas por la ventanilla "Entrada"):

Son en la computadora, dispositivos capaces de leer información (Instrucciones o Datos) con el objeto de procesarla. Existen una gran variedad de elementos de entrada, entre los cuales tenemos:

Tarjetas de Cartulina y Cintas de Papel: Que son perforadas de manera que cada perforación representa un número, una letra ó un símbolo especial de acuerdo con un código predeterminado.

Cintas magnéticas: Conocidas como "memorias externas" tienen la ventaja de permitir almacenar la información en formas más concentrada (a razón de 80 a 2400 caracteres por pulgada de longitud) y de ser más velozes, ya que pueden enviar o recibir información a la unidad de control y velocidades que van de 10,000 a 680,000 caracteres por segundo. Pueden llegar a tener hasta 730 m. de longitud.

Disco Magnético: También conocida como "Memoria externa" en general tienen un diámetro aproximado de 30 cms. y pueden grabar hasta 400,000 letras, números, y caracteres especiales, formando palabras, cifras, ó registros completos se pueden grabar o leer a razón de 77,000 a 312,000 caracteres por segundo y su tiempo de acceso a un registro alcanza un promedio de 60 milisegundos.

Una diferencia importante entre las cintas y los discos es la siguiente:

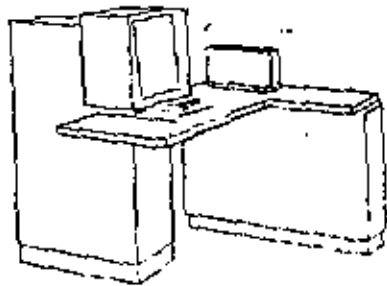
En las cintas los registros se graban o leen secuencialmente.

En los discos se tiene "Libre Acceso" a un registro cualquiera, en forma inmediata, pues cada registro se localiza por su posición física dentro del disco.

Lectora Óptica de Caracteres Impresos: Puede leer un documento impreso por una máquina de escribir, o por una máquina de contabilidad o por la impresora de una computadora a una velocidad de 30,000 caracteres por minuto.

Unidad de Representación Visual: Esta unidad de entrada/salida sirve para hacer consultas a la computadora, por medio de un teclado de máquina de escribir, y obtener la respuesta reflejada en una pequeña pantalla de televisión.

La imagen está formada por hasta 12 renglones de hasta 80 caracteres (letra, números, ó signos especiales) cada uno.



Vamos aquí una Unidad de Representación Visual, más evolucionada que la anterior, la comunicación hombre-máquina puede establecerse en ella por medio de gráficos, es decir que la entrada y la salida de datos se hacen por medio de imágenes.

Cuenta esta unidad para ello con un dispositivo con forma de lápiz, que tiene en su punta una célula fotoeléctrica. Un delgado haz de luz parte en determinado momento de un punto de la pantalla y la recorre en forma de zig-zag. Si se apoya el "lápiz" en cualquier posición de la pantalla, su célula fotoeléctrica detectará en algún momento el haz de luz. Por el tiempo transcurrido desde que el haz de luz comenzó su "barrido" hasta que fue detectado, la computadora determina en qué punto de la pantalla se encuentra apoyado el "lápiz".

Como el barrido dura una fracción de segundo y se realizan muchas barridas por segundo, se puede "escribir" con el "lápiz" sobre la pantalla y el dibujo "ingresa" en la memoria de la computadora como una sucesión de puntos codificados.

La pantalla está imaginariamente dividida en 1,040,576 puntos, de manera que los trazos que se obtienen son prácticamente continuos.

Pueden dibujarse así curvas, estructuras, letras, números y cualquier tipo de gráfico, y esa información ingresa automáticamente a la computadora.

Por otra parte, los resultados obtenidos por la computadora se capturan en la pantalla también como curvas, letras, etc., bajo control del programa almacenado en la memoria.

Lectura Óptica de Manuscritos: Salvo algunas pequeñas restricciones en cuanto al formato de los caracteres, esta unidad puede "leer" documentos escritos por cualquier persona y con cualquier ejemplo a una velocidad aproximada de 30,000 caracteres por minuto.

El registrador/analizador Fotográfico es una Unidad de Entrada/Salida de datos que realiza las siguientes funciones.

- 1) Registra los resultados de la computadora sobre microfotografías, mediante un tubo de rayos catódicos, que inciden sobre una película fotográfica, y cuyo haz electrónico actúa gobernado por el Programa Almacenado. La película se revela automáticamente dentro de la unidad y 48 segundos después está lista para ser proyectada.
- 2) Proyecta sobre una pantalla transilúcida las microfotografías registradas.
- 3) Analiza imágenes reproducidas en negativo sobre película transparente, las digitaliza y las transmite a la Unidad Central de Procesamiento.

La película utilizada tiene 30,5 milímetros de ancho y 120 metros de longitud. La Entrada o Salida de imágenes puede consistir en letras, números, símbolos, dibujos, gráficas, mapas, curvas, etc. En una microfotografía de 30,5 mm X 30,5 mm pueden registrarse hasta 30.600 letras y números, o hasta 16.777.216 puntos correspondientes a imágenes. La velocidad de Registración/Análisis es de 40.000 letras, números y símbolos por segundo, o su equivalente si se trata de imágenes.

Maquina de Escribir (Teletipo). Las unidades de almacenamiento o memorias (representadas por el archivo del señor Control) permiten registrar las instrucciones y los datos para resolver un problema; entre estas se tienen.

Las Anillas Magnetizadas: Estas pueden magnetizarse en un sentido ó en otro "Recordando" así un 1 ó un 0 respectivamente. Con 8 de éstas anillas se forma una posición de memoria, en la cual puede registrarse una letra, un dígito ó un carácter especial, según las distintas combinaciones de anillas "En 1" y "En 0", de acuerdo a un código predefinido.

Las Memorias de Flip - Flop

Las Cintas Magnéticas

Los Discos Magnéticos

El dispositivo aritmético (representado por la máquina de calcular)

realiza las cuatro operaciones aritméticas.

Las unidades de salida (representadas por la ventanilla "Salida"), que pueden ser:

Impresoras

Maquinas de Escribir (Teletipos)

Grabadoras de Cintas Magnéticas

Grabadoras de Discos Magnéticos

Unidad de Representación Visual

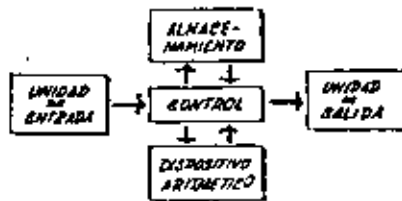
Registrador Analizador Fotográfico

Unidad de Respuesta Oral con la cual la Computadora puede hablar en todo el sentido de la palabra.

Contiene una Cinta magnetofónica en la cual un locutor ha grabado un diccionario de una gran variedad de palabras, en cualquier idioma.

Finalmente, un dispositivo electrónico de control (representado por el señor control) ayudado de un programa especial o sistema operativo (representado por el manual del señor Control), gobierna todas las operaciones de todas las unidades que componen la computadora.

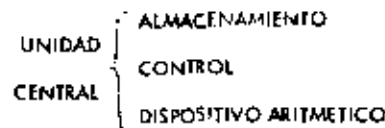
Habiendo descrito las partes que componen la computadora podemos mostrar el siguiente esquema que la representa:



O en forma más resumida:



Siendo:



Hemos hablado hasta este momento de la computadora electrónica desde el punto de vista conceptual. Durante las dos últimas décadas se han producido avances tecnológicos tan extraordinarios en materia de electrónica que la computadora ha sufrido enormes transformaciones. Veremos ahora cómo se ha ido modificando la idea original hasta llegar a los más modernos sistemas de procesamiento de datos.

Las primeras computadoras tenían circuitos con válvulas de vacío. Los tiempos de operación se medían en ellas en milisegundos (milésimas de segundo). Cuando aparecieron los transistores, el diseño de los circuitos se mejoró notablemente y la duración de las operaciones en las computadoras que utilizaban esta "Tecnología de Estado Sólido" se midió en microsegundos (millonésimas de segundo).

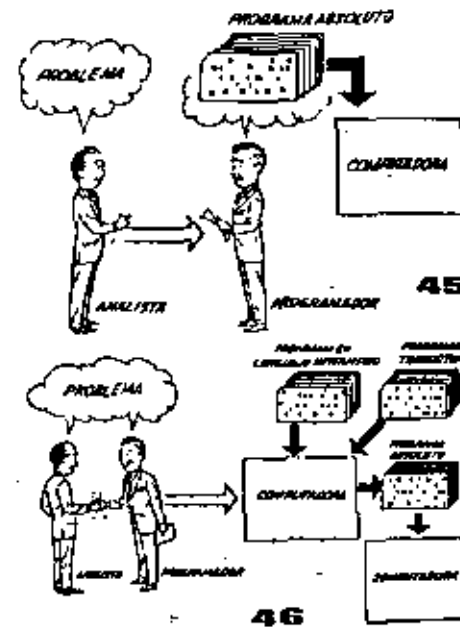
El hecho de que las nuevas máquinas fueran miles de veces más rápidas que las anteriores trajo aparejada la creación de unidades de entrada, salida y memoria externa mucho más veloces.

La invención de un nuevo tipo de transistor ("chip") provocó una verdadera revolución en los circuitos electrónicos y su proceso de fabricación el nuevo elemento es tan pequeño que en un dedo de costura caben más de 50,000 chips. Puede observarse en la figura, marcada con un círculo, un circuito completo basado en esta nueva "Tecnología de Lógica Sólida". Debido a su tamaño, se los denomina circuitos microminiaturizados o microcircuitos. Los tiempos de operación se miden ahora en nanosegundos (milmillonésimas de segundo). Ha nacido en esta forma la tercera generación de computadoras, y las altas velocidades alcanzadas permitieron un nuevo enfoque en el diseño de los sistemas de procesamiento de datos.

CS

Enunciamos brevemente los adelantos que esta tercera generación ha introducido con respecto a la tecnología anterior :

- La computadora se autogobierna y trabaja sin detenerse, pasando de un trabajo a otro sin demora alguna.
- El Operador interviene sólo cuando algún problema excepcional - ocurre. La comunicación entre hombre y máquina se realiza sólo sobre la base de "Informes por Excepción".
- Si ocurre una falla en los circuitos o en la parte electrónica la máquina realiza un autodiagnóstico e indica cuál es la anomalía.
- La velocidad de Entrada-Proceso-Salida se ha incrementado extraordinariamente.
- Todas las operaciones del sistema se realizan en forma simultánea.
- Los lenguajes de programación han evolucionado de manera notable.
- El autocontrol y la autoverificación de operaciones han alcanzado niveles insuspechados.
- Pueden realizarse, con máximo rendimiento, varios trabajos distintos simultáneamente.



Hasta ahora hemos visto muchas unidades que, en distintas combinaciones, configuran computadoras - electrónicas para las más variadas aplicaciones. Ahora nos detendremos para analizar el manejo de dichos sistemas.

El Programa de instrucciones almacenado en la Unidad Central de Procesamiento, consta de una secuencia de órdenes y comandos, expresados según una codificación especial denominada "Lenguaje Absoluto de Máquina". Las primeras computadoras se "programaban" en este complejo lenguaje. Había entonces una enorme diferencia entre nuestro idioma y aquél según el cuál debíamos comunicarnos con la máquina. Esto obligaba a un gran esfuerzo común - entre el analista que conocía el problema, y el programador que conocía la computadora, pues ambos hablaban del mismo proceso pero en distintos lenguajes.

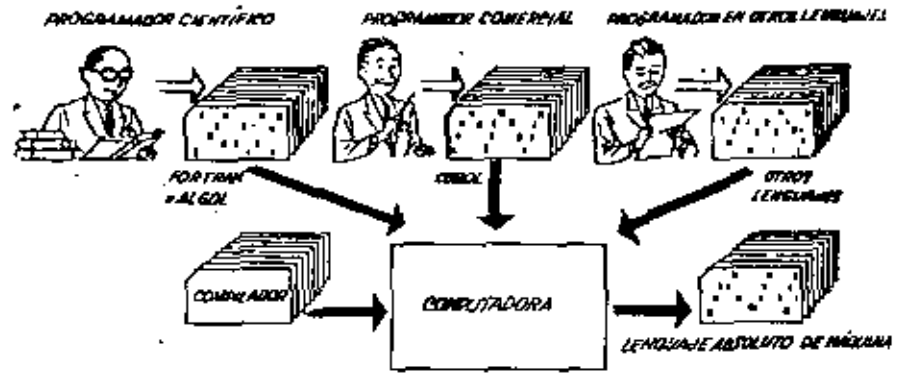
Se crearon, para solucionar el problema, lenguajes intermedios cada vez más parecidos a nuestro idioma. Es decir que cada nuevo lenguaje le temedío se acercaba más al problema y se alejaba más de la máquina. Para cada uno de estos lenguajes se creó un programa traductor llamado "Compaginador" o "Compilador", que tenía la misión de traducir el lenguaje intermedio al absoluto de máquina. Ahora, el analista y el programador "hablan un mismo idioma": ambas conocen el problema y la solución.

Pero la computadora seguía desarrollándose, y pronto los lenguajes intermedios fueron insuficientes para formular intrincados problemas científicos o comerciales. Nacieron, entonces, lenguajes especializados: dos de ellos, el FORTRAN y el ALGOL, permiten programar problemas científicos-Matemáticos utilizando una notación casi idéntica a la notación matemática común. El COBOL es un lenguaje comercial cuyos sentencias configuran oraciones y frases en forma tal que una persona que no sabe qué es una computadora, puede leer un programa y entender perfectamente qué es lo que hará la máquina cuando lo tenga almacenado.

Cada uno de estos lenguajes tiene un programa Compilador para cada tipo distinto de computadora capaz de procesarlo. Esto significa que un programador que sabe FORTRAN, por ejemplo, puede programar una computadora aún sin conocerla. Es decir que estos tres lenguajes constituyen un "aparato" de las máquinas.

La tercera generación de computadoras permitió abordar complejos problemas que incluían, entre otras, aspectos comerciales y científicos. No había un lenguaje que abarcara todas las especialidades. Entonces se reunieron todos los lenguajes conocidos en un superlenguaje llamado PL/I, cuyo compilador es tan poderoso que permite la estructuración de la programación en la forma que muestra el dibujo: varios programadores pueden programar distintas partes del proceso, incluso en diferentes lenguajes, y el programa compilador entregará como resultado las instrucciones del proceso completo, en Lenguaje Absoluto de Máquina.

Hemos llegado así a que la computadora nos "entienda", en lugar de que se leente a recibir órdenes en su idioma.



## 1-1 Algorithms and flowcharts

What is an algorithm? An *algorithm* is a list of instructions for carrying out some process step by step. A recipe in a cookbook is an excellent example of an algorithm. The preparation of a complicated dish is broken down into simple steps that every person experienced in cooking can understand. Another good example of an algorithm is the choreography for a classical ballet. An intricate dance is broken down into a succession of basic steps and positions of ballet. The number of these basic steps and positions is very small but, by putting them together in different ways, an endless variety of dances can be devised.

In the same way, algorithms executed by a computer can combine millions of elementary steps, such as additions and subtractions, into a complicated mathematical calculation. Also by means of algorithms, a computer can control a manufacturing process or coordinate the reservations of an airline as they are received from ticket offices all over the country. Algorithms for such large-scale processes are, of course, very complex, but they are built up from pieces, as in the example we will now consider.

If we can devise an algorithm for a process, we can usually do so in many different ways. Here is one algorithm for the everyday process of changing a flat tire.

1. Jack up the car.
2. Unscrew the lugs.
3. Remove the wheel.
4. Put on the spare.
5. Screw on the lugs.
6. Jack the car down.

We could add many more details to this algorithm. We could include getting the materials out of the trunk, positioning the jack, removing the hubcaps, and loosening the lugs before jacking up the car, for example. For algorithms describing mechanical processes, it is generally best to decide how much detail to include. Still, the steps we have listed will be adequate to convey the idea of an algorithm. When we get to mathematical algorithms, we will have to be much more precise.

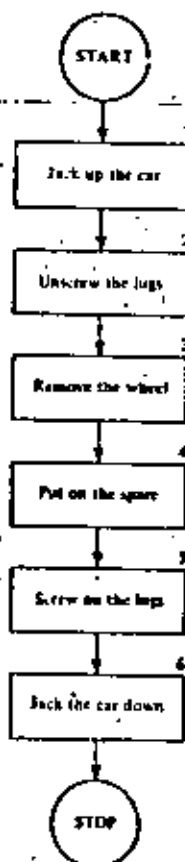


FIGURE 1-1  
First flat-tire flowchart.

## Chapter

## Algorithms and computers

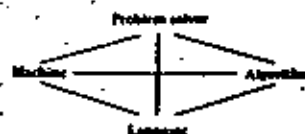
### 9

Computers arouse curiosity in most of us. Articles in popular magazines and newspapers, current books, and TV shows heighten this curiosity, but such sources cannot be expected to present information in the carefully ordered sequences that is possible in a book like this. Whether you are drawn by curiosity alone, or economic necessity, or both, conscientious study of this book will help you to break through to a new level of understanding about computers, their uses, and their consequences.

Computer science deals with people who have problems to solve and with algorithms, the solutions to these problems. The solutions are expressed in special languages that represent stored data and communicate to machines the manipulations that are to be carried out on that data.

Each of these four elements (problem solver, algorithm, language, and machine) affects the others in interesting ways. For example, depending on its richness, a language can either limit or extend our ability to express complex plans of action effectively. And, depending on its capabilities (i.e., its architecture), a machine can execute some plans of action on certain data representations more effectively than on others. The loop of interaction closes when the problem solver changes the plan of action, the language, or the machine architecture to suit his purpose.

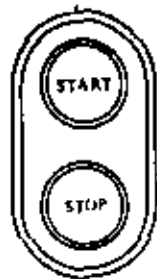
This book introduces all four components of this interaction. Every chapter takes you around this "four-cornered race track" and, with every circumnavigation, you gain a deeper and clearer understanding of the interplay among the four elements. You, of course, play the problem solver using a computer. To get the most out of this experience, laboratory practice is almost indispensable. But, even if you can't have actual computer experience, a careful reading of this book should illuminate the computer science scene far better and far beyond what you have previously perceived.



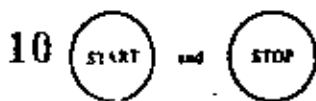
Elements of computer problem solving.



A *flowchart* is a diagram representing an algorithm. In Figure 1-1 we see a flowchart for the flat-tire algorithm.



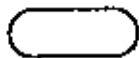
The



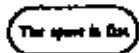
in the flowchart remind us of the buttons used to start and stop a piece of machinery. Each instruction in the flowchart is enclosed in a frame or "box." As we will soon see, the shape of the frame indicates the kind of instruction written inside. A rectangular frame indicates a command to take some action.

To carry out the task described by the flowchart, we begin at the start button and follow the arrows from box to box, executing the instructions as we come to them.

After drawing a flowchart, we always look to see whether we can improve it. For instance, in the flat-tire flowchart we neglected to check whether the spare was flat. If the spare is flat, we will not change the tire; we will call a garage instead. This calls for a decision between two courses of action. For this purpose we introduce a new shape of frame into our flowchart.

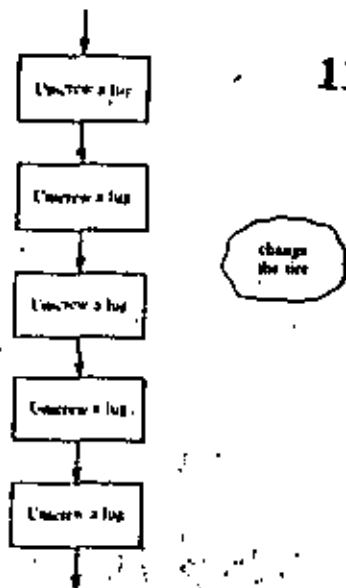
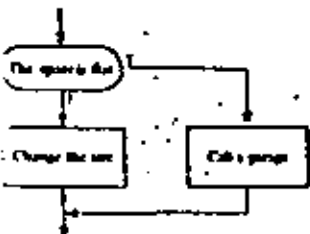


Inside this oval frame we will write an assertion instead of a command.



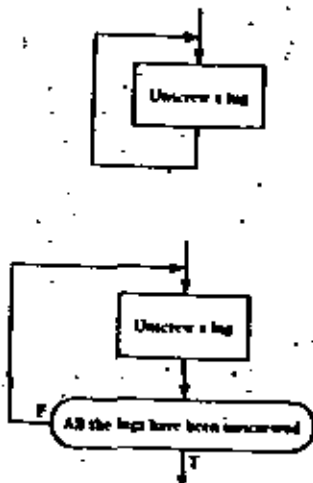
This is called a *decision box* and will have two exits, labeled T (for true) and F (for false). After checking the truth or falsity of the assertion, we choose the appropriate exit and proceed to the indicated activity. Incorporating the flowchart fragment on the left into Figure 1-1, we obtain the flowchart in Figure 1-2.

There is another instructive improvement possible. The instruction in box 2 of our flowchart actually stands for a number of repetitions of the same task. To show the additional detail we could replace box 2 by a step for each lug.



The awkwardness of this repeated instruction can be eliminated by introducing a *loop*.

As we leave the box, we find that the arrow leads us right back to repeat the task again. However, we are caught in an endless loop, since we have provided no way to get out and go on with the next task. To correct this situation, we require another decision box, as shown on the left.



Replacing box 2 of our flowchart with this mechanism and making a similar replacement for box 5, we get the final result shown in Figure 1-3.

Now that you have followed the development of the flat-tire flowchart, try to devise one of your own. In the algorithm of the following exercise, you will probably discover some decisions and loops. There are many different ways of flowcharting this algorithm, so many different-looking flowcharts will be created.

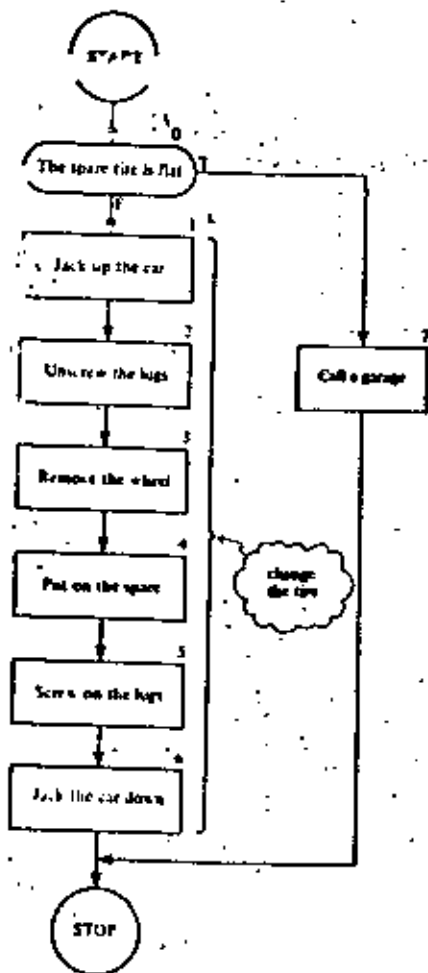


FIGURE 1-2  
Second flat-tire flowchart.

12

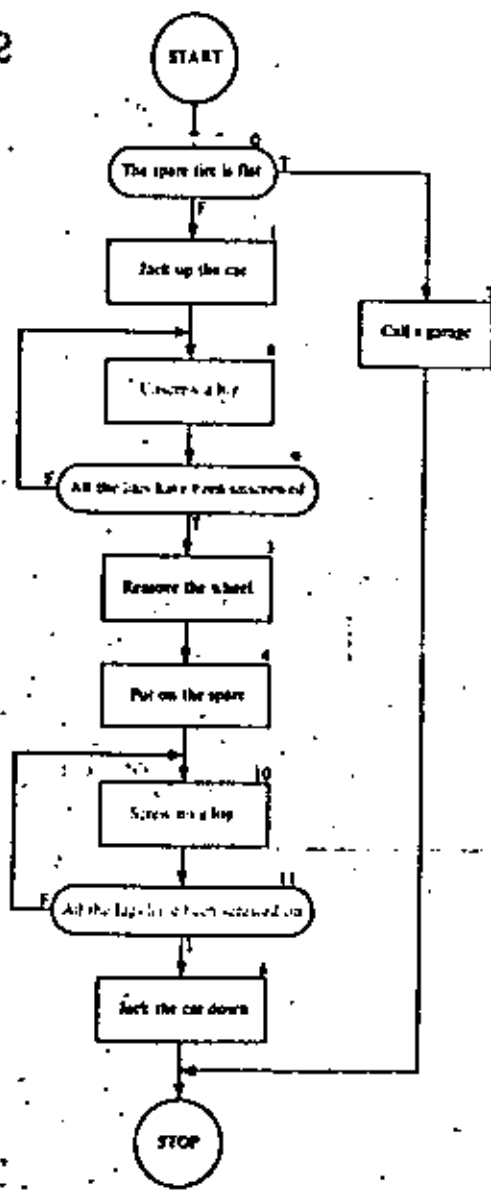


FIGURE 1-3  
First flat-tire flowchart.

EXERCISE 1-1

1. Prepare a flowchart representing the following recipe.

*Mrs. Good's Rocky Road*

Ingredients:

- |                                       |                       |
|---------------------------------------|-----------------------|
| 1 cup chopped walnuts                 | 1 cup evaporated milk |
| 1 pound block of baker's chocolate    | 1 cup corn syrup      |
| 1 pound of marshmallows cut in halves | 1 teaspoon of vanilla |
| 3 cups sugar                          | 1 pound of butter     |
|                                       | 1 teaspoon of salt    |

Place milk, corn syrup, sugar, chocolate, and salt in a four-quart pan, and cook over a high flame, stirring constantly until the mixture boils. Reduce to medium flame and continue boiling and stirring until a drop of syrup forms a soft ball in a glass of cold water. Remove from the flame and allow to cool for 10 minutes. Beat in butter and vanilla until thoroughly blended. Stir in walnuts. Distribute marshmallow halves over the bottom of a 10-inch square, buttered baking pan. Pour syrup over the marshmallows. Allow to cool for 10 minutes. Cut in squares and serve.

1-2  
A numerical algorithm

Now we are ready to examine an algorithm for a mathematical calculation. As a first example, we consider the problem of finding terms of the Fibonacci sequence:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots$$

In this sequence, or list of numbers, the first two terms given are 0 and 1. After that, the terms are constructed according to the rule that each number in the list is the sum of the two preceding ones. Check that this is the case. Thus, the next term after the last one listed above is

$$34 + 55 = 89$$

Clearly, we can keep on generating the terms of the sequence, one after another, for as long as we like. But, in order to write an algorithm for the process (so that a computer could execute it, for example), we have to be much more explicit in our instructions.

Before subjecting this process to closer scrutiny, let us review a little of the interesting history of this sequence. It was introduced in 1202 A.D. by the Italian mathematician, Fibonacci, to provide a model of population growth in rabbits.

His assumptions were: (1) it takes rabbits one month from birth to reach maturity, (2) one month after reaching maturity, and every month thereafter, each pair of mature rabbits will produce another pair of rabbits; and (3) rabbits never die.

One senses that this model is not completely realistic. But the essence of mathematical modeling is to start with a crude model that emphasizes the important aspects of the situation and suppresses less important information. A more refined model can be developed later, profiting from the experience with the crude model. Thus we might eventually improve the Fibonacci model by obtaining more accurate figures on the birth rate, taking mortality into account, considering the limitations of food supply, the effects of predators, disease, and overcrowding, and the like.

In spite of its frivolous origins, the Fibonacci sequence has many fascinating properties and plays a role in the solution of a number of seemingly unrelated mathematical problems. There is currently a published quarterly journal entirely devoted to the properties and applications of the Fibonacci sequence.

After this long digression, let's see how the rabbit-pair population model gives rise to the Fibonacci sequence. Fibonacci starts with one pair of newborn rabbits at the beginning of month *one*, and he then lets nature take its course. This is shown in Table 1-1, which we now explain.

TABLE 1-1  
Rabbit Population

Beginning of Month	1	2	3	4	5	6	7	8
Infant rabbit pairs	1	0	1	1	2	3	5	8
Mature rabbit pairs	0	1	1	2	3	5	8	13
Total rabbit pairs	1	1	2	3	5	8	13	21

Look at the arrows in the table. The number of pairs of infant rabbits in any month (after the first) is equal to the number of pairs of mature rabbits in the preceding month (condition 2 in the Fibonacci model). This explains the green arrows. In each month after the first, the number of pairs of mature rabbits will equal the total number of rabbit pairs in the preceding month (condition 1 in the Fibonacci model). This

explains the gray arrows. Following the arrows, we see that, from the third month onward, the total in any month is the sum of the totals in the two preceding months. Thus the rabbit population model generates the Fibonacci sequence except for the initial zero, which can be taken as the total number of rabbits in month zero.

Eliminating the reference to rabbits, we can tabulate the calculation of the terms of the Fibonacci sequence in Table 1-2.

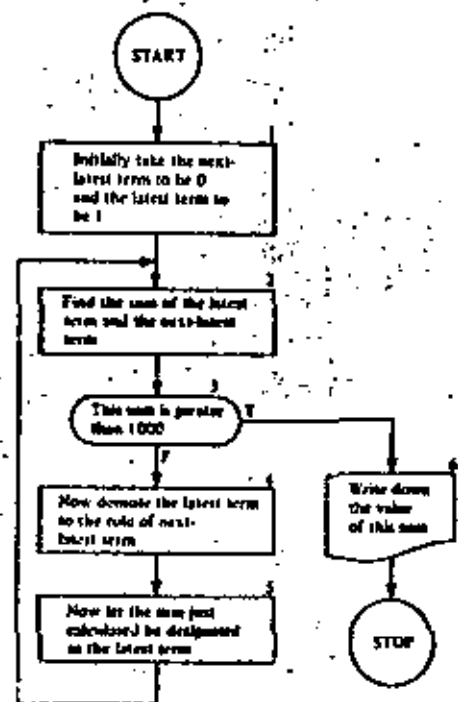


FIGURE 1-4  
Flowchart for Fibonacci sequence.

TABLE 1-2

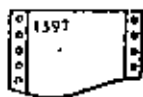
Next Latest Term	Latest Term	Sum
0	1	$0 + 1 = 1$
1	1	$1 + 1 = 2$
1	2	$1 + 2 = 3$
2	3	$2 + 3 = 5$
3	5	$3 + 5 = 8$
5	8	$5 + 8 = 13$
8	13	$8 + 13 = 21$

25

We can see that in each step the latest term gets "demoted" to the role of next latest term and the sum becomes the new latest term.

Let's construct a flowchart for finding the first term to exceed 1000 in the Fibonacci sequence (Figure 1-4).

After going through the loop of flowchart boxes numbered 2 to 5 enough times (it happens to be 15 times), we eventually emerge from box 3 at the T exit and proceed to box 6. This box is seen to have a different shape because it calls for a different kind of activity—that of writing down our answer. The shape is chosen so as to suggest a page torn off a line printer, once the most common of computer output devices.



1-3  
SIMPLOS, a conceptual model of a computer

Variables

17

rabbits in the current month to that in the preceding month. Express each ratio as a decimal and carry out the calculation to the nearest thousandth.

- Express in your own words what seems to be happening to these ratios.
  - Find the reciprocals of each of the ratios in Problem 3a.
  - What relationship between the ratio  $r$  and its reciprocal  $1/r$  seems to be becoming more and more true? Express this relationship as an equation.
  - If this relationship held exactly, what would be the exact value of  $r$ ? That is, solve the equation for  $r$ .
4. Repeat Problem 3 using:
- The table in Problem 1.
  - The table in Problem 2.

The algorithm of the preceding section can be expressed in much simpler notation that is, at the same time, more nearly acceptable by a computer as a set of instructions. To do this we must introduce a conceptual model of how a computer works. This conceptual model is so extraordinarily simple that we will call it the SIMPLOS computer. It is amazing, but true, that such a simple view of how a computer works is completely adequate for this entire course. We will present a more realistic picture of a computer in later sections of this chapter.

In computing work, a *variable* is a letter or a string of letters used to stand for something. For now, this "something" that a variable stands for will always be a number. (As we progress through this book, we will take an ever broadening view of the sort of thing a variable can stand for.) In the formula

$$A = L \times W$$

the letters  $A$ ,  $L$ , and  $W$  are variables. In the formula

$$\text{DIST} = \text{RATE} \times \text{TIME}$$

$\text{DIST}$ ,  $\text{RATE}$ , and  $\text{TIME}$  are variables.

At any particular time, a variable will stand for one particular number, called the *value* of the variable, which may change from time to time during a computing process. The value of a variable may change millions of times during the execution of a single algorithm.

In our conceptual model of a computer we associate with

- EXERCISES 1-2
- Suppose in the rabbit problem we had started in month one with one pair of infant rabbits and three pairs of mature rabbits. Make a table similar to Table 1-1 to show the state of the population over the first eight months.
    - How would you modify the flowchart of Figure 1-4 so as to generate the first term of this modified sequence greater than 1000?
  - Repeat Problem 1 with three pairs of infant rabbits and one pair of mature rabbits.
  - For the Fibonacci sequence in Table 1-1, calculate from month two through month twelve the ratio,  $r$ , of the total number of

each variable a *storage box*. On the top of each box there is a removable gummed sticker with the associated variable inscribed on it, and inside the box there is a strip of paper with the *present value* (or *current value*) of the variable written on it. The variable is a *name for the number* that currently appears inside.

Each box has a lid that may be removed when we wish to assign a new value to the variable. Each box has a window in the side so that we may read the value of a variable with no danger of altering its value. These boxes constitute the *storage* of our computer. In Figure 1-5 we see one stage in the execution of the Fibonacci sequence algorithm of the preceding section. Here NEXT stands for "next latest term" and LATEST stands for "latest term."

To summarize, the *data storage* of a computer is to be thought of as subdividable into a number of information containers or boxes. Each such storage box may be given a meaningful name (sticker), and each may be given (assigned) a value.

Some people view a computer as an electronic and mechanical system having a data storage similar to that just described, along with a number of other interconnected units or modules, each with a special set of functions that, when activated appropriately, carry out algorithms. Figure 1-6 is one way to depict the organization of such a computer system. If we were to pursue the explanation of this system according to the module view, it would be necessary to define the functions of each module and explain the significance of the arrowed lines into and out of each box. But it would also be

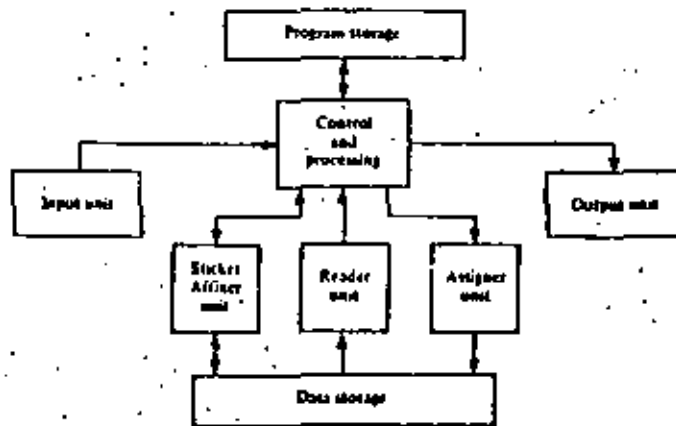


FIGURE 1-5 Storage.

FIGURE 1-6 View of a SIMPL OS system as a set of interconnected modules.

necessary to bring the diagram to life by explaining the action sequences that occur in which each module serves the needs of the others so that the overall effect is to process information (i.e., to compute) in the desired fashion.

A second way to view a computer is to picture the active modules as robots working as a team. The actions of each robot always follow a fixed pattern, according to a set of relatively simple rules. We shall take this view in our conceptual model, SIMPL OS.

The Model and How It Works

We visualize a computer as a number of storage boxes together with a staff of four robots—the *Master Computer* and three assistants, the *Assigner*, the *Reader*, and the *Sticker Affixer*. All these components are quartered in one room, isolated from those who will use the computer.

The *Master Computer* corresponds to the control and processing unit in Figure 1-6. He has a flowchart on his desk that sets forth the instructions according to which he delegates certain tasks to his assistants (Figure 1-7). "Note that the flowchart corresponds to the information kept in the *program storage* module of SIMPL OS."

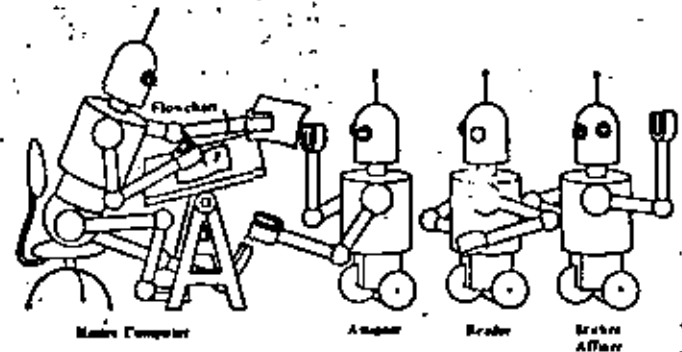


FIGURE 1-7 The Master Computer and his staff.

To see how this team operates, let us suppose the computer is in the midst of executing the Fibonacci sequence algorithm of Figure 1-4. One of the instructions in this algorithm was:

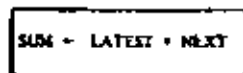


In a simplified flowchart notation, this instruction will take the form:



Inside this flowchart box we find an *assignment statement*. Reading this statement aloud, we would say, "Assign to SUM the value of LATEST plus NEXT," or more simply, "Assign LATEST + NEXT to SUM." The arrow pointing left is called the *assignment operator* and is to be thought of as an order or a command. Rectangular boxes in our flowchart language will always contain assignment steps and will therefore be called *assignment boxes*.

To see what takes place when the Master Computer comes to the above statement in the flowchart, let us assume that the variables LATEST and NEXT (but not SUM) have the values seen in Figure 1-5. The computation called for in the assignment statement is spelled out on the right-hand side of the arrow, so the Master Computer looks there first.



He realizes that he needs to know the values of the variables LATEST and NEXT, so he sends the Reader out to fetch copies of these values from storage.

The Reader then goes and finds the storage boxes labeled

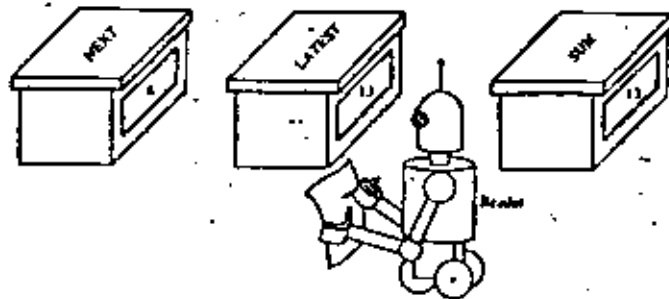


FIGURE 1-8 The Reader copying a value from storage.

LATEST and NEXT. He reads the values of these variables through the windows (Figure 1-8), jots down the values, and carries them back to the Master Computer (Figure 1-9).

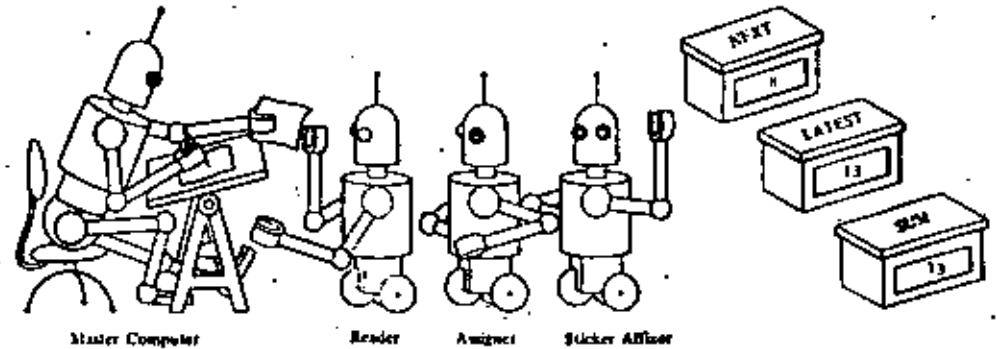


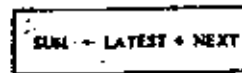
FIGURE 1-9 The Master Computer receives the copy.

The Master Computer computes the value of LATEST + NEXT using the values of these variables brought to him by the Reader.

$$8 + 13 = 21$$

What does he do with this value?

The Master Computer now looks to the left of the assignment arrow in his instruction.



He sees that he must assign the computed value of LATEST + NEXT, namely, 21, to SUM so he writes "21" on a slip of paper, calls the Assigner, and instructs him to assign this value to the variable SUM.

The Assigner goes to storage, finds the box labeled SUM, and *empties* out its contents (Figure 1-10). Then he places in the box the slip of paper containing the new value, closes the lid, and returns to the Master Computer for a new task.

In other words, assignment is the process of giving a value to a variable. We say that assignment is *destructive* because it displaces the former value of the variable. Reading is *non-destructive* because the process in no way alters the values of any of the variables in storage.

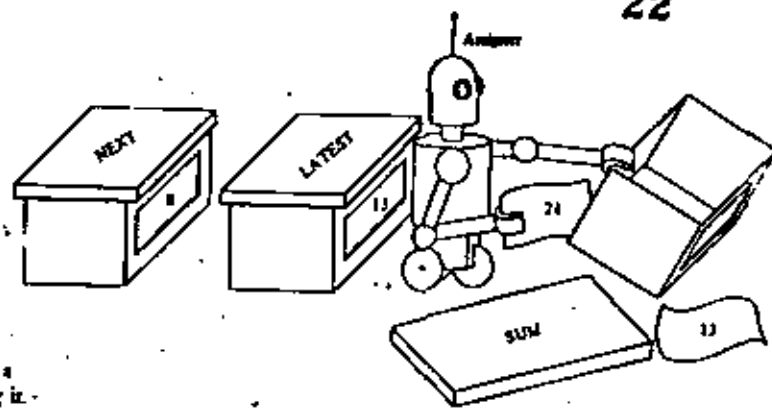


FIGURE 1-10  
The Assigner emptying a  
storage box and refilling it.

In Figure 1-11 we present the entire flowchart of Figure 1-4 in simplified flowchart language. The old and new flowcharts are placed side by side for easy comparison.

The translation requires very little explanation. It should be obvious that the statement in box 1 on the left is equivalent to the two statements in box 1 on the right. The new version of box 2 has been discussed in detail.

We see that the two statements in boxes 4 and 5 of the old flowchart are compressed into one box, box 4 of the new flowchart. This is permissible whenever we have a number of assignment statements with no other steps in between. However, it is very important to understand that these assignment statements must be executed in order from top to bottom, not in the opposite order and not simultaneously. The order in which things are done may be extremely important.

You can see that the statements in box 4 involve no computation but merely change the values in certain storage boxes. This sort of activity occurs frequently in flowcharts.

In box 6 of the flowchart we see only the word SUM. The shape of the box (called an *output box*) tells us that the value of the variable SUM is to be written down or displayed. If, in some other algorithm, we wished to write down the values of several variables, we would list these variables in an output box separated by commas, as illustrated on the left.

A, B, C, DEST

We will now describe the duties of the Sticker Affixer. We consider that the computation is begun by the transmission of a flowchart to the Master Computer. The first thing the Master Computer does is to scan the flowchart, making a list

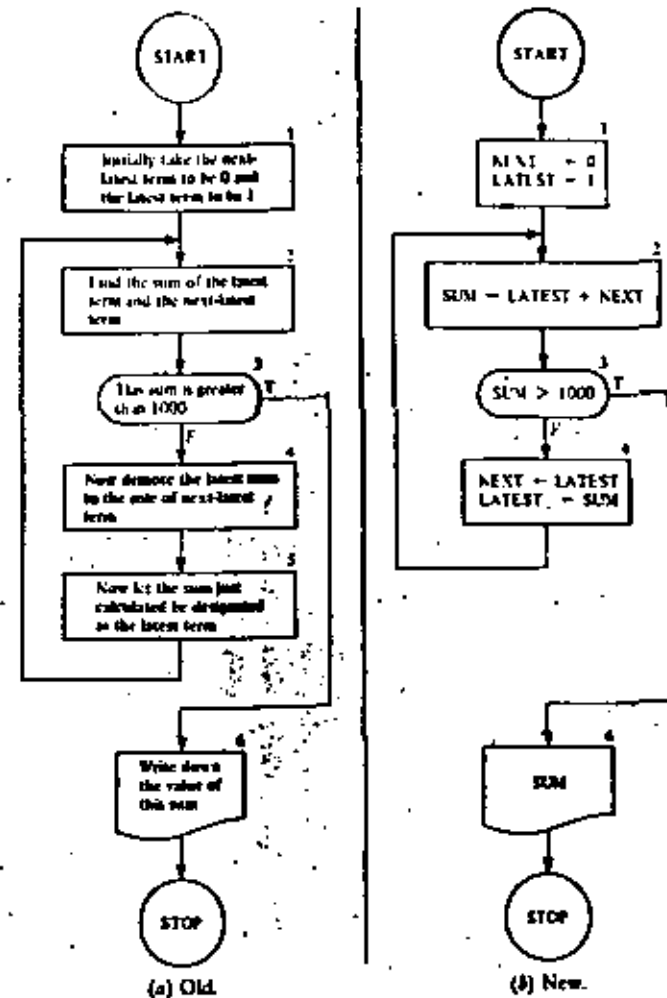


FIGURE 1-11  
Translation of Fibonacci  
sequence flowchart into formal  
flowchart language.

of all the variables used. In the case of the Fibonacci sequence flowchart of Figure 1-11b, this list would have the form

NEXT  
LATEST  
SUM

The Master Computer hands this list to the Sticker Affixer, who now springs into action. He inscribes each of these variables on a sticker, goes to a bin of unfabricated storage boxes, and slaps one of these stickers on each of three boxes (Figure 1-12).

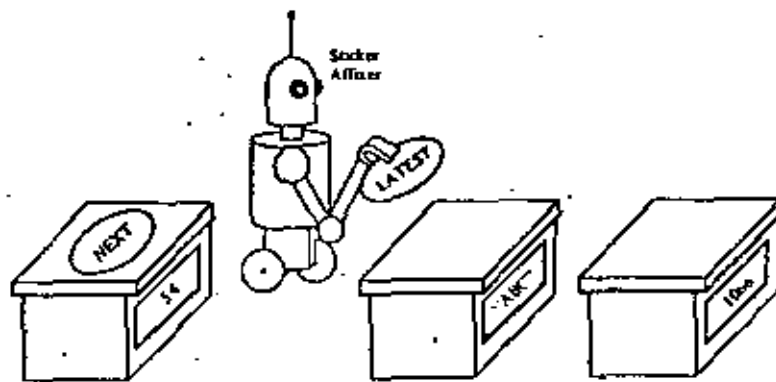


FIGURE 1-12  
Sticker Affixer at work.

Now the instructions in the flowchart are executed until the **stop** instruction is reached. At this juncture, the Master Computer directs the Affixer to unpeel all the labels and throw them into a recycle bin.

### Tracing the Flowchart

To understand better what our flowchart in Figure 1-11b does, let us trace through it, executing the steps as the Master Computer and his assistants do them (see Table 1-3).

In this trace, for ease of reading, the values of the variables are reproduced only when assignments are made to them. In between such steps, the values of the variables do not change and therefore have the most recently recorded values. For example, in step 33, where a test is performed, the values of the variables are

$$\text{NEXT} = 55, \quad \text{LATEST} = 89, \quad \text{SUM} = 144$$

In step 34 the values are

$$\text{NEXT} = 89, \quad \text{LATEST} = 144, \quad \text{SUM} = 144$$

You can see that in step 48 in the execution of our algorithm we finally leave box 3 by the *true* exit and pass on to box 6, where we output the answer, 1597, and stop.

The utter simplicity of our conceptual model avoids and removes certain pitfalls. There is an ever-present danger of thinking of assignment as equality or substitution. (We will say

TABLE 1-3  
Tracing of the Flowchart of Figure 1-11b

Step Number	Flowchart Box Number	Values of Variables			Test	True or False
		NEXT	LATEST	SUM		
1	1	0	1			
2	2			1		
3	3				$1 > 1000$	F
4	4	1	1			
5	2			2		
6	3				$2 > 1000$	F
7	4	1	2			
8	2			3		
9	3				$3 > 1000$	F
10	4	2	3			
11	2			5		
12	3				$5 > 1000$	F
13	4	3	5			
14	2			8		
15	3				$8 > 1000$	F
16	4	5	8			
17	2			13		
18	3				$13 > 1000$	F
19	4	8	13			
20	2			21		
21	3				$21 > 1000$	F
22	4	13	21			
23	2			34		
24	3				$34 > 1000$	F
25	4	21	34			
26	2			55		
27	3				$55 > 1000$	F
28	4	34	55			
29	2			89		
30	3				$89 > 1000$	F
31	4	55	89			
32	2			144		
33	3				$144 > 1000$	F
34	4	89	144			
35	2			233		
36	3				$233 > 1000$	F
37	4	144	233			
38	2			377		
39	3				$377 > 1000$	F
40	4	233	377			
41	2			610		
42	3				$610 > 1000$	F
43	4	377	610			
44	2			987		
45	3				$987 > 1000$	F
46	4	610	987			
47	2			1597		
48	3				$1597 > 1000$	T
49	6			1597		



more about this later.) This and other potential sources of confusion, such as the effect of a certain sequence of flowchart statements, can be cleared up by thinking in terms of the SIMPLoS model, which will always give the right answers.

In fact, an excellent way to understand these ideas of reading and assigning values to variables is to make some storage boxes and, with some friends, work through several algorithms as described in this section.

What is the effect of changing the order of the two assignment statements in box 4 of Figure 1-11b so as to appear as seen below?



Trace through the flowchart with this modification until you find the answer.

2. (a) To compare the effects of the assignment statements



find the missing numbers in the table below.

Values Before Execution of Assignment		Assignment To Be Executed	Values After Execution of Assignment	
A	B		A	B
7	13	A ← B	?	?
7	13	B ← A	?	?

- (b) In which of the two cases is it true that  $A = B$  after assignment?  
 (c) Are the effects of the two assignment statements the same or different?

2. Modify the flowchart in Figure 1-11b so as to carry out the algorithm of Problem 1, Exercises 1-2.

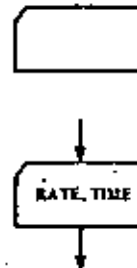
- Modify the flowchart of Figure 1-11b so as to output each term of the Fibonacci sequence starting with the third (i.e., omit the initial 0, and 1).
- Revise the flowchart of Problem 4 to calculate the ratio,  $r$ , of LATEST to NEXT (as calculated in Problem 3a, Exercises 1-2) and output this ratio (as well as LATEST) at each step.
- Revise the flowchart of Problem 5 to calculate at each step the reciprocal of  $r$ . Add this value to the output list.

1-4 Input-output

Imagine that you are a bookkeeper in a large factory. You have records of the hourly rate of pay and the number of hours worked for each employee, and you have to calculate the week's wages. Of course, this can be done by hand, but assume there are nearly 1000 workers in the plant, so that the job would be quite tedious. Naturally you prefer to have the computer execute this task for you, but you will have to devise a flowchart to convey the instructions to the computer.

How will the hourly wages and the hours worked come into our computation? Must each new value of RATE and TIME be represented by a separate assignment box? This is certainly a possibility, but it would require thousands of flowchart boxes—a most undesirable state of affairs. This unpleasant necessity can be eliminated by using the concept of *input*.

We now introduce a new shape of frame, the *input box*, into the flowchart language. The input box has this shape to suggest a "punch card" (a frequently used input medium, but not the only one). Inside the box will appear a single variable or a list of variables separated by commas.



What happens in our SIMPLoS model when the Master Computer encounters such an instruction? To answer this question, we must endow the SIMPLoS model with an additional feature not previously needed (Figure 1-13). SIMPLoS has a conveyor belt (called the *input belt*) that carries slips of paper from outside the room into the environment of the computing staff. On the outside end of the belt the "user" or "programmer" (who is not a member of the computer staff) places these slips of paper, with values written on them, on the conveyor belt in the order in which he wants them to be used.

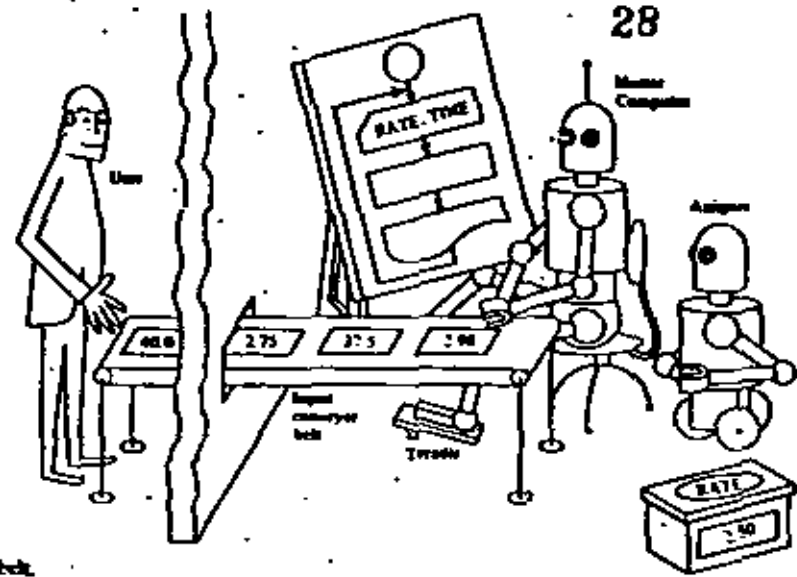


FIGURE 1-13 SIAIPLOS with input belt.

When the Master Computer comes to the input instruction he does the following:

1. Steps on a treadle running the conveyor belt until the next slip of paper comes within reach.
2. Remove his foot from the treadle, stopping the belt.
3. Takes a slip of paper from the belt and hands it to the Assigner with instructions to assign the value thereon to the variable, RATE.

When the Assigner returns from this task, the Master Computer repeats the above process, but this time tells the Assigner to assign the new value to TIME. When this is done, the Master Computer follows the arrow in his flowchart to the next instruction.

We see that an input box is a command to make assignments, but this command is essentially different from that in an assignment box. In an assignment box the values to be assigned are to be found in computer storage or are computed from values already stored, whereas with an input box the values to be assigned are obtained from outside the computer. No calculation is called for in an input box. Moreover, the

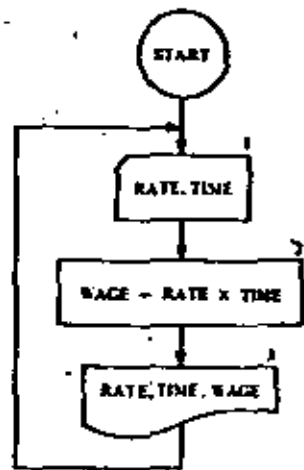


FIGURE 1-14 Payroll algorithm.

values to be input never appear in the flowchart itself. Only the variables to which these values are to be assigned appear in the input boxes of the flowchart.

In an actual computer (not our conceptual one) the distinction between the two kinds of assignment need not be so sharp. Assignments called for in an input box usually involve some mechanical motion such as transporting a punched card or other unit of recorded information past a reading station where the coded contents may be copied. But to gain speed the data often are transported into a special section of storage called an *input buffer*, well before the data are actually needed by the executing algorithm. In this case, when the input step is executed, what actually happens is that data values are simply copied at electronic speed from storage boxes of the input buffer to storage boxes of the variables that are specified in the input step of the algorithm.

Now let's see how to use the input box in our hourly rate and payroll problem. Should we input the data from all the cards before we start our calculations? If so, we would need a great many storage boxes in which to store all these data. Instead, we will calculate the wages after each data set is read. A description of our method is as follows.

1. Input one value of RATE and one value of TIME by the process described above.
2. Multiply the RATE by the TIME to get the WAGE.
3. Output the values of RATE, TIME, and WAGE.
4. Return to step 1.

In the flowchart of Figure 1-14 each of the first three steps of the above list appears in a similarly numbered box. Step 4 is represented by the arrow returning from box 3 to box 1.

You may wonder why the flowchart does not have a stop button. SIAIPLOS always terminates execution of an algorithm when an input step is being executed, and the input belt contains too few values to match the variables in the input box. Execution of the payroll algorithm will therefore always halt after the last *rate, time* pair of data values has been processed and control once again reaches box 1, where it is discovered that the input belt is empty.

It will also be useful to visualize output in a similar way.

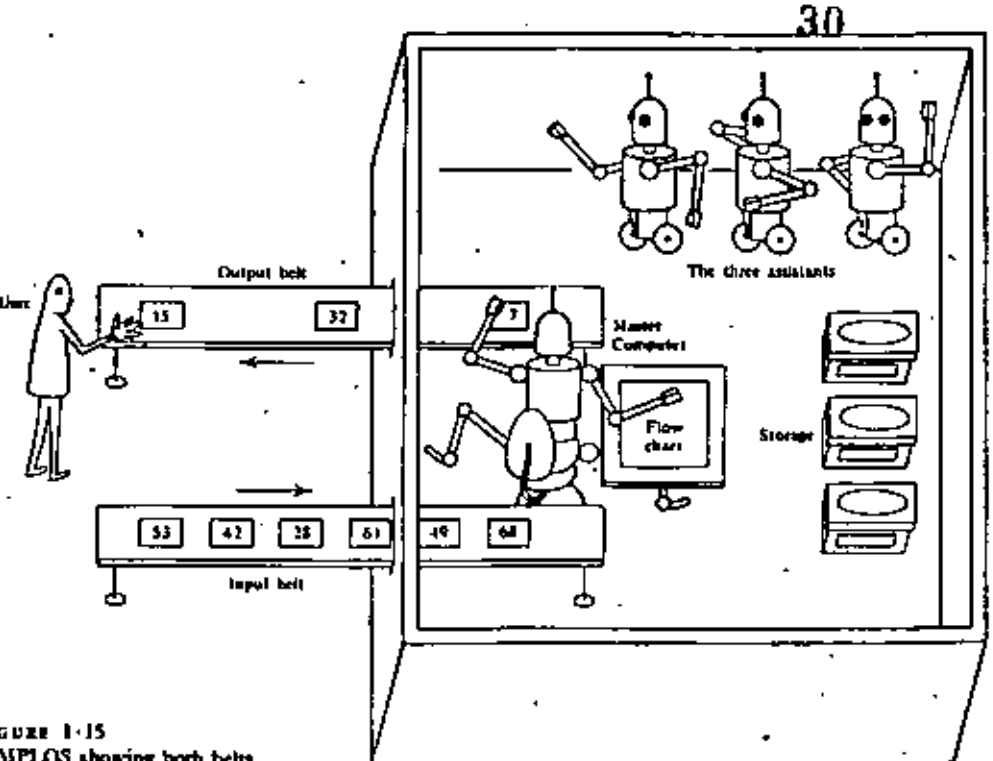


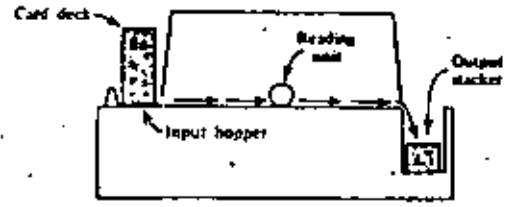
FIGURE 1-15  
SIMPLIOS showing both belts.

We endow SIMPLIOS with a second conveyer belt, the *output belt*. This belt runs *out* instead of in and runs continuously—it needs no treadle. Each time the flowchart calls for output, the Master Computer writes the proper value on a slip of paper and drops it on the output belt, which carries the slip through the wall to the outside environment of the user. A view of this situation from the top is seen in Figure 1-15.

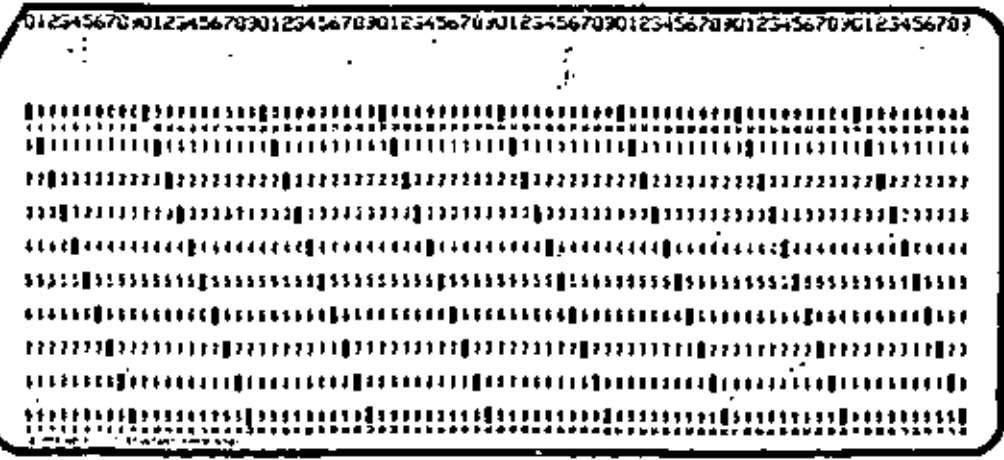
Records versus Streams

Our conveyer belt model of input-output suggests that data values move as a *stream* into and out of a computer system. Although in actual computers this is not always strictly the case, the analogy nevertheless is quite close. To pursue this idea let us consider the punched card reader, one of the most common input devices on actual computers. First, a sequence of data values is punched on cards. The cards are then placed in proper order in the *input hopper* of the card reading device.

Usually the card deck is placed face down so that the bottom (first) card in the deck is the first one to be read. Each time more input data are required, another card is drawn from the bottom of the deck and its contents are read, either electro-mechanically or photo-optically. Once read, the card is dropped into an *output stacker* and thus discarded.



The information contained on a single punched card need not in principle be limited to one value. For example, depending on what a program is designed to expect, an 80-column card may contain up to eight 10-digit integers or up to twenty 4-digit integers.



When preparing data cards one is always faced with the decision of whether to utilize their capacity fully or to punch on each only the values required for the execution of one input step in the algorithm. The latter choice, although somewhat wasteful of card space, makes the data cards easier to check.

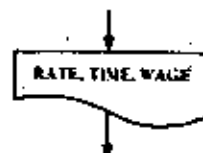
Our payroll problem provides us with a case in point. If,

for convenience in locating data preparation errors, we restrict the contents of each card to one *rate*, *time* pair, then each time box 1 is executed, one and only one data card will be drawn off the input deck, read, and discarded. We could be more wasteful and punch only one data value on each card. Then successive cards would contain first a *rate* value, then a *time* value, and so on. In this case, each execution of box 1 must cause two data cards to be drawn from the deck and read, and the analogy between the input conveyor belt and the card reading activity is very close indeed. That is, when the Master Computer hits the foot treadle to bring in one data value, the actual computer will signal the card reader to draw off one card and read it.

The analogy is less apparent if we allow the data cards to contain more than one *rate*, *time* pair and if we expect the pairs to be considered in turn during successive executions of box 1. In this case, box 1 can no longer mean "read a card" but, instead, "assign respectively to *rate* and *time* values from the next data pair in the deck. If the next pair cannot both come from the current data card, then draw off another card from the deck and read it. If, on the other hand, there is at least one more data pair yet to be processed from the most recently read card, then process that data pair." This interpretation assumes that an input buffer is filled (and refilled) with data from each newly read data card and that values are assigned to *rate* and *time* by simply copying information from this buffer into the respective program variables, always remembering for future use which items in the buffer have not yet been copied.

We see, therefore, that using an input buffer guarantees that each data pair in the sequence will be processed in turn, no matter how many pairs are punched on each data card. None will be missed or skipped over. It is in this sense that the stream analogy is preserved even though the sequence of data items is grouped into arbitrary-sized *card records*.

The SIMPLIOS model is a primitive machine. It has only stream-oriented input and output. After values are placed on the conveyor belt to be output and are carried to the outside environment of the user, how are they displayed? We certainly are aware that in actual computer systems all values are printed or displayed on a screen in some sort of "format" with a



Importance of the Conceptual Model

particular number of columns, but the only fact we are interested in with respect to SIMPLIOS is that the values are output.

When it comes to interpreting output boxes of a flowchart, the situation is somewhat different. The output box on the left is considered to be a command to print the current values of the three variables, RATE, TIME, and WAGE on one line. (If the list won't fit on one print line, more lines are used.) Furthermore, if the same output box is executed again, the next set of three values will appear on a new line below the first set. If the three variables appeared in three individual boxes instead of in one single box, then each would be printed on a separate line. Thus each execution of an output box is considered to begin printing a new line.

No doubt you have wondered why, at the very start of our study, so much attention has been given to a conceptual model of a computer and its details. Can any model, especially this one, which seems so simple and at the kindergarten level for some readers, be that important or that valuable to us? You may develop similar doubts about the value of flowcharts as you proceed further.

The model and the flowcharts we develop are *abstractions* of real machines and of real computer programs. Once we see the connection between an abstraction and the *concrete* or real thing, we can often gain more understanding of the real thing by studying and manipulating its abstract counterpart. So, high on our list of priorities should be an attempt to understand and appreciate the connections between the abstract and the concrete. For example, in the next sections of this chapter we examine how an actual computer is organized and how it works. Thereafter, it will be easier to see why the conceptual model, no matter how silly it may have first appeared, is a very useful, simplified view of a real computer. Likewise, just as soon as we try to write and test actual computer programs, we shall see that the flowchart gives us a simpler but more revealing way to think about computer programs for most purposes.

Experience has taught us that problem solving with computers is very effective if we can work first with a simplified model of a machine and a simple descriptive algorithmic lan-

guage in which to express our problem solutions. Then it is comparatively easy to map these solutions over to programs written in some convenient programming language such as BASIC, FORTRAN, ALGOL, or COBOL, so that the programs can be executed on some real, convenient computer.

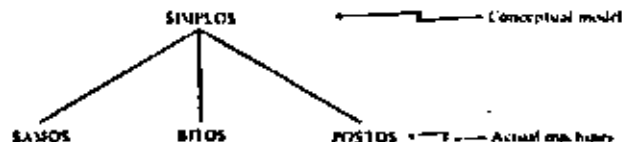
Modify the flowchart of Figure 1-14 to provide for an overtime feature. All hours in excess of 40 are to be paid at time and a half. You will have to place a decision box somewhere in the flowchart to determine whether the worker actually put in any overtime. The formula by which his wages are computed will depend on the outcome of this test.

## 1-5

## Actual computers

Now we are ready to examine how our conceptual model of a computer can be realized in an actual machine. For the first 25 years of modern computer history (1949 to 1974), nearly all actual machines were built following a more or less stereotyped pattern suggested by John Von Neumann (1903-1957). A prototype machine following this pattern is discussed in this and following sections. We will call it SAMOS. SAMOS is a very simple machine; that is, it is stripped down to the bare essentials. Some features of its operation are described in considerable detail, while others are glossed over. The programming of SAMOS is described briefly in Section 1-6 and in more detail in the Appendix, the purpose of which is to help the reader see a closer connection between language for expressing algorithms and machines that execute them.

It would be foolhardy to assume that SAMOS-like machines are the "be all and end all of computers," since the architecture of computers is still undergoing rapid change. For this reason, aspects of two other machines are discussed briefly in this book. One machine, called BITOS, appears later in this chapter; the other, called POSTOS, is considered in Chapter 8. Each of the three machines exhibits certain distinct characteristics for the implementation of our conceptual model, SIMPLOS.



In order to study this book it is useful, although not essential, to gain a good understanding of how an actual computer works. We suggest that you read once through the material of the next two sections without attempting to master it. As you work exercises that relate to SAMOS, or have occasion to study SAMOS in the Appendix, you will no doubt come back to the next two sections for a more careful study.

1-6  
SAMOS

How are all those storage boxes of SIMPLOS realized in actual practice? The storage of actual computers is built of electronic components in a variety of ways and with a variety of materials. Here we describe one way that a SAMOS storage can be built.

## Core Storage



FIGURE 1-16  
A magnetic core.

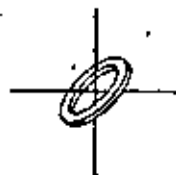


FIGURE 1-17  
Where two wires cross.

SAMOS storage, packaged in a rectangular box, is an arrangement of tiny magnetic doughnuts as small as 1/40 of an inch in diameter. These doughnuts are called *cores* (Figure 1-16). The cores are laid out in 61 horizontal layers or trays called *core planes*. On each of these layers, wires are strung evenly in two directions like the lines on a sheet of graph paper. There are 100 wires in each direction. At each point where two wires cross, the wires are threaded through a core, like the thread passing through the eye of a needle (Figure 1-17). (Still other wires are threaded diagonally through each core within each plane. Their function is not important to the discussion that follows, and they are therefore ignored.)

Figure 1-18 is a picture of a core plane from an actual computer built in the mid-1960s. Since there are  $100 \times 100$  crossings in each SAMOS core layer, we see that there are 10,000 cores in each core plane and hence  $61 \times 10,000 = 610,000$  cores in the entire SAMOS storage.

These cores are capable of being magnetized in either the clockwise or the counterclockwise sense (Figure 1-19). Because of this a core can store information. We could think of clockwise magnetization as meaning "yes" and counterclockwise as meaning "no." We will instead think of clockwise as standing for "0" and counterclockwise for "1." In any event, the information contained in the magnetization of a core is the smallest

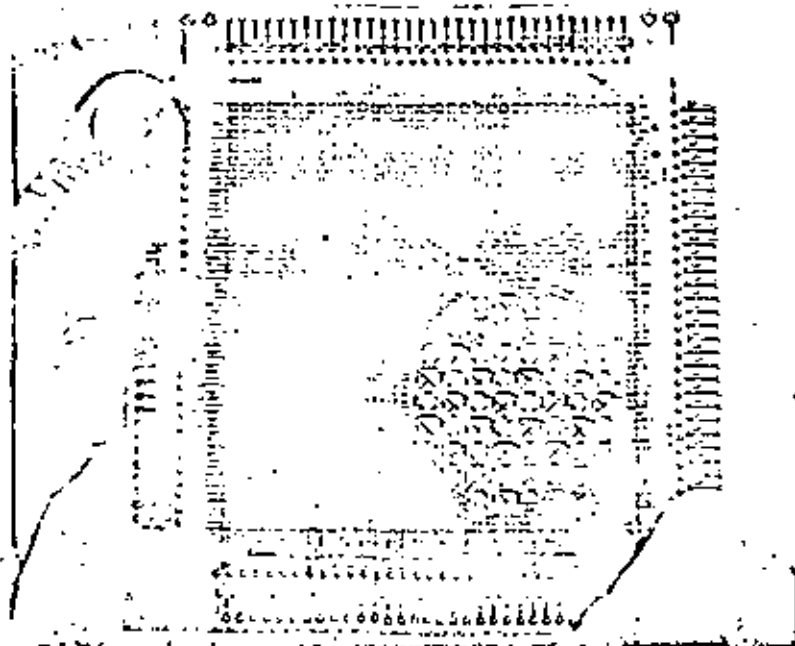


FIGURE 1-18 An actual core plane. (Courtesy of IBM).

unit of information and is called a bit of information. We see that one core can store one binary digit, 0 or 1, but a collection of cores can store a very large number of bits. We will discuss this idea later, after a digression on how the cores get their magnetism.

First, you must know that a pulse of electric current moving along a wire generates a magnetic field running around the wire, as shown in Figure 1-20. The strength of the magnetic field is strongest near the wire and dies away as we move further from the wire.

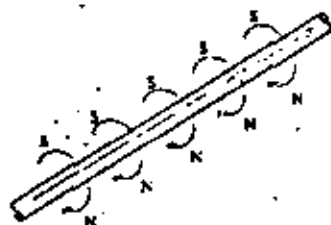


FIGURE 1-20 Magnetic field resulting from a pulse of electric current.

If the direction of the current is reversed, the direction of the magnetic field is also reversed (Figure 1-21).

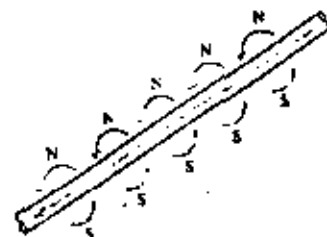


FIGURE 1-21 Reversing the direction of the magnetic field.

Thus, when a pulse of current passes through a core, the core will become magnetized in one direction or the other, depending on the direction of the current (Figure 1-22).



FIGURE 1-22 A core in a magnetic field.

But how can we manage to magnetize just one core instead of the whole string of cores (Figure 1-23) through which the pulse passes? The answer lies in the magnetic properties of the material from which the core is made. In this material,



FIGURE 1-23 A row of cores in a magnetic field.

if the pulse is too weak, the direction of magnetization of the core is only temporarily altered, and after the pulse of current has passed by, the core merely returns to its former magnetic condition, whatever that was.

On the other hand, if the current is strong enough, the core remains magnetized in the sense established by the direction of the current, regardless of the former magnetic condition of the core. The situation is analogous to trying to throw a ball from the ground to the flat roof of a building. If you have enough power in your throw, the ball will land on the roof; otherwise it will bounce against the wall and fall back to the ground.

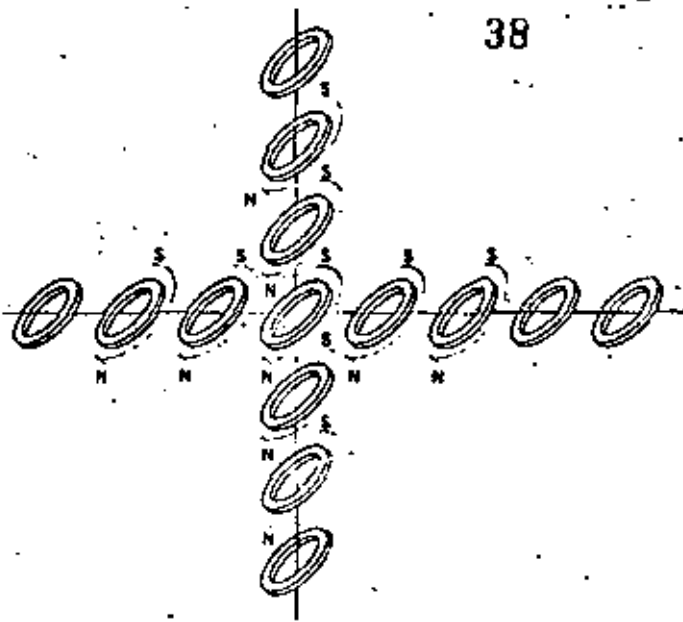


FIGURE 1-24  
Doubling the magnetic field at  
the wire crossing.

The strength of the pulses is carefully regulated so that one pulse is not sufficient to permanently magnetize a core, but two pulses acting simultaneously will exceed the threshold strength and result in permanent magnetization. Thus, pulses passing along two of the wires (Figure 1-24) will permanently magnetize just the one core that is located where the wires cross.

### SIMPLOS and SAMOS Stores Compared

Let's leave the individual core planes and consider the entire store of the SAMOS computer, composed of the 61 core planes (Figure 1-25). Each vertical column of 61 cores constitutes a *computer word*. Thus, the storage of the computer is composed of 10,000 words. These words have addresses that are four-digit numbers from 0000 to 9999 and, like house numbers, the addresses identify the words. Each of the 10,000 dots suggested on the top of the box is the top of a vertical column of 61 cores (or a word). The method of assigning the addresses is indicated in the figure.

Each of these words corresponds to a storage box in our conceptual SIMPLOS model. For each variable in the flow-

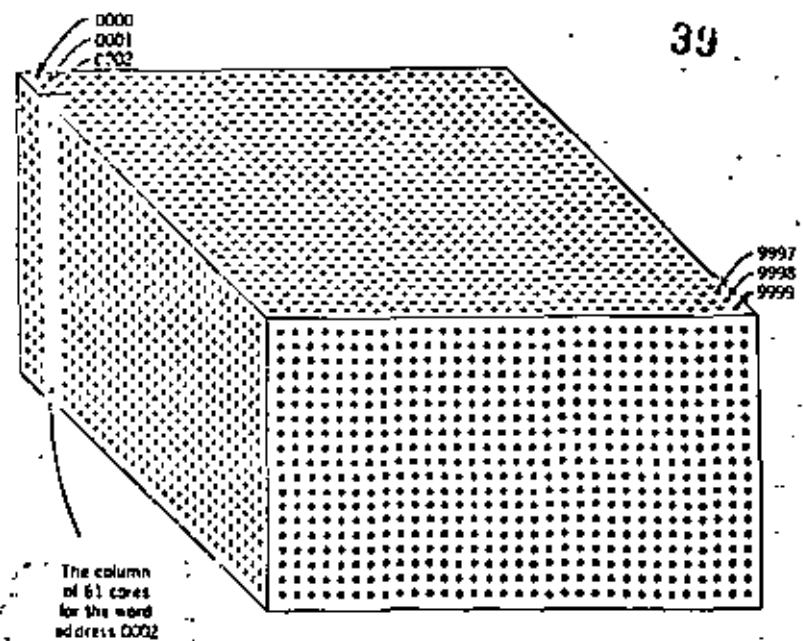


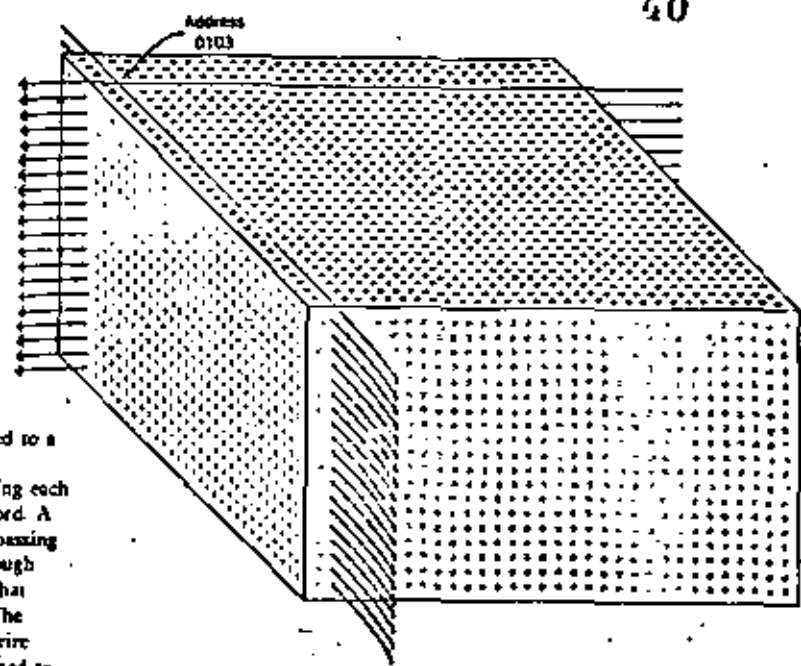
FIGURE 1-25  
Cores of one computer  
word

chart there is a SAMOS word with a definite address. The word contains a certain pattern of bits determined by the directions of magnetization of its cores and representing the value of that variable. "Assigning a value to a variable" is effected by putting a certain pattern of bits into a word (Figure 1-26).

When we say "the Master Computer tells the Assigner to assign the value 1597 to the variable SUM," what actually takes place is this. The variable SUM is represented inside the machine by means of its address; suppose it is 0103. Now all the  $61 \times 2 = 122$  wires passing through cores in the word addressed 0103 are energized with pulses of current in the proper directions so as to achieve the pattern of bits representing the number 1597. In a modern computer this assignment process can be performed in a fraction of a microsecond; a microsecond is a millionth of a second.

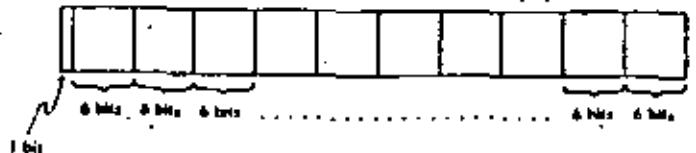
In the binary system of representation, a number such as 1597 is coded as a string of 1's and 0's, for example:

11000111101



**FIGURE 1-26**  
A bit pattern is assigned to a computer word by appropriately magnetizing each of the cores for that word. A core is magnetized by passing an electronic pulse through each of the two wires that intersect at that core. The diagram identifies the wire pairs that must be selected to assign a bit pattern for the word at address 0103.

For SAMOS any string would be preceded by a string of zeros to fill out all the bit positions of the word of storage. While numbers are coded in binary form in many computers, binary is certainly not the only choice. In a machine such as SAMOS, for instance, computation is carried out in the decimal system, which means that bit patterns in a word of storage must be coded to represent decimal digits instead of binary digits. Moreover, we want to store letters as well as decimal digits. For this reason, we subdivide our 61-bit SAMOS words into 11 character positions as shown below.



The first position (one bit only) is reserved for a code that designates the sign, + or -. Here 0 is sufficient to represent the + character and 1 signifies the - character. Each of the other positions consists of six bits and can be used to store

Character	Code	Character	Code	Character	Code	Character	Code
0	00 0000	A	01 0001	J	10 0001	0	11 0000
1	00 0001	B	01 0010	K	10 0010	S	11 0010
2	00 0010	C	01 0011	L	10 0011	T	11 0011
3	00 0011	D	01 0100	M	10 0100	U	11 0100
4	00 0100	E	01 0101	N	10 0101	V	11 0101
5	00 0101	F	01 0110	O	10 0110	W	11 0110
6	00 0110	G	01 0111	P	10 0111	X	11 0111
7	00 0111	H	01 1000	Q	10 1000	Y	11 1000
8	00 1000	I	01 1001	R	10 1001	Z	11 1001
9	00 1001						

**FIGURE 1-27**  
Character code

a digit or a letter, that is, *character*, according to the code shown in Figure 1-27.

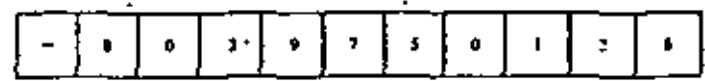
For each group of six bits,  $2^6$  or 64 distinct combinations of zeros and ones are possible. In Figure 1-27 we have used up only 37 of the 64 combinations possible with a six-bit code. This leaves 27 additional combinations for other special symbols such as +, ≥, and the like. One of the 37 combinations of special interest is the blank space, □, which is coded as

1 1 0 0 0 0

With this code you can see that the two 61-bit computer words displayed vertically in Figure 1-28 turn out to be

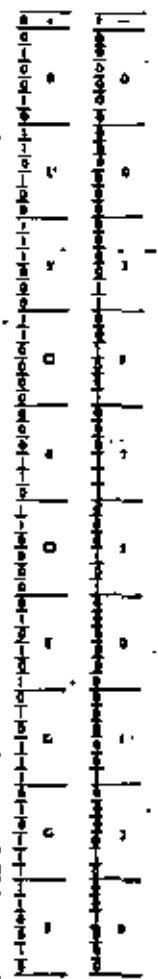


and



From now on we shall represent our SAMOS computer words as strings of 11 characters instead of strings of 61 bits. In a number of conventional computers of similar design eight instead of six bits are grouped to represent character codes, making it possible to distinguish among a considerably larger set of characters than is the case in SAMOS. This distinction, however, has absolutely no effect on the principles of character representation and manipulation that occur in ensuing chapters.

The construction of the main storage for any actual computer is of great interest mainly to computer engineers and designers. Storage components currently are built from various



**FIGURE 1-28**  
Detailed bit patterns for two computer words.



types of physical devices and materials, including magnetic cores, magnetic thin films, and transistor flipflops. There is considerable variety in the circuitry used to organize and utilize such components and in the methods of packaging and miniaturizing them. Their physical characteristics, such as size, speed of access for storing and retrieving information, energy requirements to operate them, and cost of fabrication, vary also. Nevertheless, schemes similar to that used in the word-organized core storage of SAMOS have been used to assemble and incorporate all of these types of storage units into conventional computer systems. You might be surprised at how much understanding of this subject you can gain with a relatively small investment of study time. (See, for example, one of the references on this topic in the reading list at the end of this book.)

SAMOS Processing Unit

Now that we have seen how the SAMOS storage is structured, we will consider how the storage is used in executing an algorithm.

Our computer has several other components besides the store. These are shown in Figure 1-29.

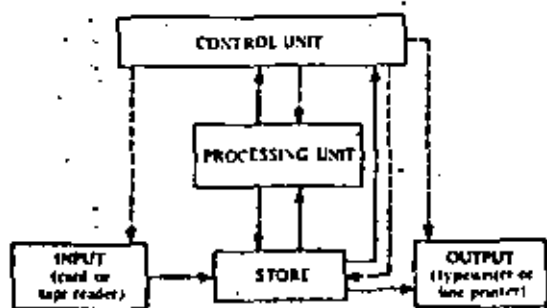
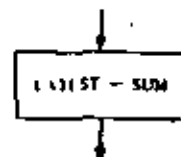


FIGURE 1-29  
The principal components of SAMOS.

The solid lines indicate the directions in which values or instructions may be transferred. The dashed lines indicate the exercise of control. The control unit and the processing unit perform the duties of the "Master Computer" and his helpers.

An important part of the processing unit is the *accumulator*. This special computer word holds the result of each arithmetic operation.

Furthermore, a simple assignment such as



is carried out by first obtaining a copy of the value of SUM, placing it in the accumulator, and then copying the value in the accumulator into the computer word belonging to the variable LATEST. The value of SUM is unchanged in this process. Notice, however, that values to be input or output do not pass through the SAMOS accumulator but go directly into and out of storage.

When the control unit receives and interprets an *order*, some computer operation is activated. The orders are in the form of coded instructions stored in the computer; we will see about them presently.

Machine Language

Getting an algorithm into a form a machine can execute involves several translations that we can represent as follows:



You have already had a little experience with the first translation step. The second translation step is the process of converting a flow chart into a procedural language such as FORTRAN, ALGOL, COBOL, or PL/I. You learn how to do this in your language manual. If approached properly, this translation step is quite mechanical and can be performed by a person (or by a machine) who has no idea what the algorithm is all about.

In many computers of advanced design, the third translation process can be omitted because the machine's language and the procedural language are effectively identical. When the third translation step is necessary, and it is for a computer such as SAMOS, the process is completely mechanical and is normally done by the computer itself. This process is called *compiling*.

It is not necessary right now to know how compiling is done, but it may be interesting to know the reason for doing it. Each make and style of computer has its own language—that is, its own set of instructions that it can understand. Use of a procedural language allows us to avoid a tower of Babel in which a programmer would have to learn a new language for each machine he wishes to use. A procedural language constitutes a kind of "Esperanto" that enables a programmer to communicate with many different machines in the same language. Moreover, a procedural language is generally much easier to learn to use than machine language. The programmer merely prepares, say, a FORTRAN program on punched cards and feeds it into the computer, which "compiles" a sequence of machine language instructions. This sequence, called a *machine language program*, is then placed in the computer storage. In many systems the programmer may transmit his program to the computer storage by typing it a line at a time, using a typewriter or other keyboard instrument to serve as the input device of the computer system.

### Sequencing of Computer Instructions

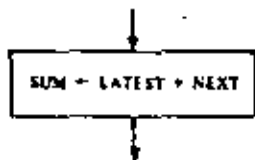


FIGURE 1-30  
A flowchart box.

Successive SAMOS instructions are placed in consecutively addressed storage locations starting with 0000. After the computer has executed an instruction, the control unit will always take the next instruction from the next address, unless there is a *branching* instruction providing a different address from which to take the next instruction.

To see how this works, consider the instruction taken from the Fibonacci sequence flowchart (Figure 1-11b), shown here in Figure 1-30.

The procedural language equivalent will not look much different. Thus, in FORTRAN this instruction would appear as

and in ALGOL as

```
SUM = LATEST + NEXT
```

and in APL as

```
SUM := LATEST + NEXT
```

and in BASIC as

```
LET SUM = LATEST + NEXT
```

and in COBOL as

```
COMPUTE SUM EQUALS LATEST  
PLUS NEXT
```

FIGURE 1-31  
SAMOS instructions for  
Figure 1-30.

```
+ LDA 000 0101
+ ADD 000 0100
+ STO 000 0102
```

In the SAMOS machine language, a variable cannot be referred to by *name* but only by the *address* in storage associated with the variable. Suppose that NENT, LATEST, and SUM have been given, respectively, locations 0100, 0101, and 0102. Then in the SAMOS language, the flowchart instruction translates to a sequence of three machine instructions, as shown in Figure 1-31.

These instructions have the form of 11-character words, although the first character is unimportant and the fifth, sixth, and seventh are of no interest to us here. The type of *operation* to be performed is coded using the three letters in positions two, three, and four, and the four-digit numeral at the right is the *address* associated with that operation.

The letters LDA stand for "Load the Accumulator." The whole instruction

```
+ LDA 000 0101
```

means, "Make a copy of the value stored in address 0101 without altering the original, and store the copy in the accumulator." Clearly, this is the function of the Reader in our conceptual model. We will not go into the details of the electronics involved in carrying out this instruction. It is sufficient to know that when a copy of that instruction is brought to the control unit, certain switches are set by the control unit that allow a pulse current to pass through the cores of the word 0101. The magnetized cores cause a change in the current that, in turn, allows a copy to be made.

The second instruction in Figure 1-31 means, "ADD the value in the word addressed 0100 to the value already in the accumulator and place the result in the accumulator." The third instruction means, "Copy (or STORE) the value in the accumulator into the word addressed 0102." Executing a STO instruction is analogous to the work of the Assigner in our conceptual model. Speeds vary from machine to machine, but in modern computers, the time required to carry out such instructions is usually on the order of a millionth of a second.

### A Complete SAMOS Program

We are almost able to translate the entire flowchart for the Fibonacci sequence algorithm (repeated here in Figure 1-32) into SAMOS language. First note, however, that constants never appear explicitly in SAMOS instructions. Instead, an

instruction to fetch a constant must refer to the storage address where the desired value may be found. Of course, this also applies to variables. Thus part of the compiling process involves providing storage addresses for the constants (as well as for the variables) appearing in the program. We allocate the locations 0017, 0018, and 0019 for the constants 0, 1, and 1000 appearing in the flowchart and specify the proper values for these words.

We assume that the storage locations 0100, 0101, and 0102 have been allocated for the variables NEXT, LATEST, and SUM, but that no values have been placed in these words. As execution of the SAMOS program for the Fibonacci algorithm starts, the state of storage is shown in Figure 1-33. This figure

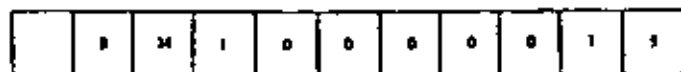
Discussion

also illustrates a "coding form" on which one might have written the SAMOS program (gray-colored information). You will notice several new SAMOS operations not previously seen. These are explained in the following discussion.

The instructions in storage addresses 0004, 0005, and 0006 have already been discussed. Before looking at the other instructions, remember that the variables are in storage locations 0100 through 0102.

From previous discussions you should see that the instruction found at 0000 will, when executed, copy the value in 0017 (i.e., the number 0) into the accumulator. Next, the instruction in 0001 copies the value in the accumulator into the word at address 0100. Together these steps are equivalent to assigning 0 to the variable NEXT. Similarly, the instructions in addresses 0002 and 0003 are equivalent to assigning the value 1 to the variable LATEST.

Remember that the control unit executes the instructions in order until it comes to a branching instruction. The first branching instruction is found in address 0009, reading



The code BMI stands for "Branch on a Minus." The whole instruction means, "If the value in the accumulator is negative, go to address 0015 for the next instruction; otherwise, go on as usual to the next numbered address (0010)." We will see shortly that the value in the accumulator at this time is just

$$1000 - \text{SUM}$$

so that the value in the accumulator will be negative only in the case that

$$\text{SUM} > 1000$$

is true. In this case, the branching instruction sends us to address 0015, where we see the instruction

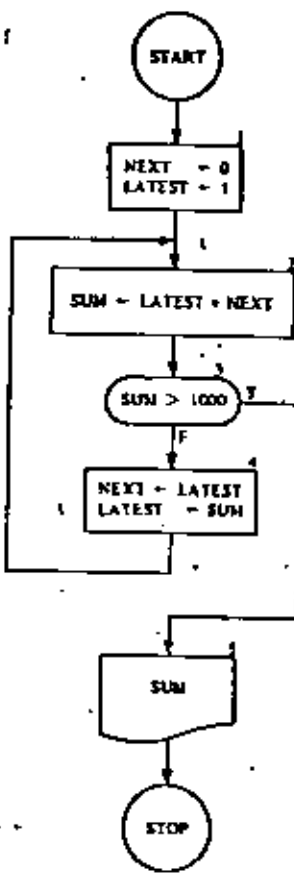
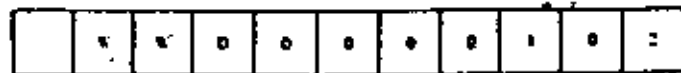


FIGURE 1-32 Fibonacci sequence algorithm.

Storage Location (Address)	Operation Code				Address	Flowchart Equivalent
	1	2	3	4		
0000	LDA	000	001	7	0017	NEXT = 0
0001	STO	000	010	0	0100	
0002	LDA	000	001	8	0018	LATEST = 1
0003	STO	100	010	1	0101	
0004	LDA	000	010	1	0101	SUM = LATEST + NEXT
0005	ADD	000	010	0	0100	
0006	STO	000	018	7	0187	
0007	LDA	000	001	9	0019	SUM > 1000
0008	SUB	000	010	2	0102	
0009	BMI	000	001	5	0015	
0010	LDA	000	010	1	0101	NEXT = LATEST
0011	STO	000	010	0	0100	
0012	LDA	000	010	2	0102	LATEST = SUM
0013	STO	000	010	1	0101	
0014	AKI	000	000	4	0004	Arrow from flowchart box 4 to box 2
0015	WMI	000	010	2	0102	SUM
0016	HLL	000	000	0	0000	STOP
0017	000	000	000	0	0000	The constant 0
0018	000	000	001	0	0001	The constant 1
0019	000	000	100	0	0000	The constant 1000
0100						The variable NEXT
0101						The variable LATEST
0102						The variable SUM

FIGURE 1-33 Fibonacci sequence algorithm. The 1 column and columns 5, 6, and 7 may be left empty in lines containing instructions.

which means, "Write the Word in address 0102." This amounts to printing out the value of SUM.

Now why is it that when the instruction in address 0009 is reached, the number in the accumulator is

1000 - SUM

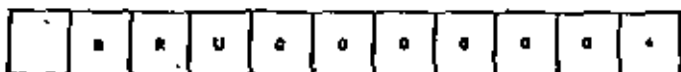
Well, on looking at the instruction in address 0007, we see that it instructs us to load the accumulator with the contents of address 0019, that is, to put the number 1000 in the accumulator. The next instruction, the one in 0008, tells us to "subtract the contents of address 0102 from the accumulator and put the result in the accumulator." Since the contents of 0102 are just the value of SUM, this amounts to the placing of

1000 - SUM

in the accumulator.

You should be able to verify for yourself that the instructions in addresses 0010 through 0013 accomplish the assignments indicated in the right-hand column of Figure 1-33.

The instruction in address 0014 needs to be described.



BRU stands for "BRanch Unconditionally." The meaning of the entire instruction is, "Go back to address 0004 for the next instruction and continue in order from there." You can see that this corresponds to the arrow from flowchart box 4 leading back to flowchart box 2, where we repeat the summing step.

The instruction in 0016, of course, stands for HALT and amounts to stopping the computing process.

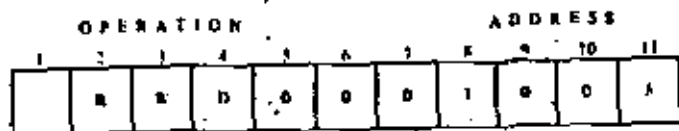
You can best understand all this by tracing through the SAMOS program by hand, keeping a record of the following details.

1. Which instruction is being executed.
2. The value in the accumulator.
3. The values in the addresses 0100, 0101, and 0102 (the values of NEXT, LATEST, and SUM).

Notice that the instructions in addresses 0000 through 0016 are never altered, nor are the contents of the locations 0017 through 0019 (the constants 0, 1, and 1000).

EXERCISES 1-6

1. Construct a list of SAMOS instructions for the flowchart in Figure 1-14. You will need two additional types of instructions. The first is



which is an instruction to read a value from a card into the computer word addressed 1005.

The second is



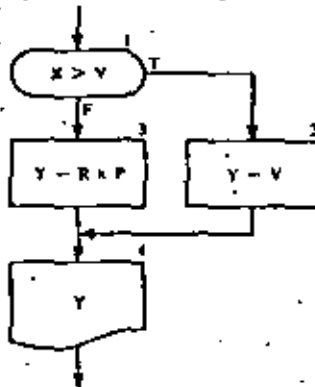
which is an instruction to multiply the value in the accumulator by the value in address 1023 and put the result in the accumulator. (Of course, in the address part of these instructions, we may put any address we wish.)

2. This question relates to the flowchart fragment and proposed SAMOS translation of it shown below. For each of your answers the assumed objective is to make the proposed SAMOS fragment consistent with the given flowchart fragment.

Loc	Opcode	Addr
0018	LDA	000 0500
0019	MPY	0351
0020	MPY	0351
0021	STO	0451
0022	WFO	0451
0023	LDA	0351
0024	SUB	0401
0025		003C
0026	LDA	0351
0027	SUB	0501
0028	STO	0351
0029	BRU	
0030		

- With what memory location must the variable X be associated?
- What operand address is needed for the BRU instruction that is shown at location 0029?
- What should be the operation code for the instruction at location 0025?
- What operation code is needed for the instruction located at 0030?

3. This question relates to the flowchart fragment and proposed SAMOS translation of it shown below. For each of your answers, assume the objective is to make the proposed SAMOS fragment consistent with the given flowchart fragment.



Loc	Opcode	Addr
0017	LDA	000 0100
0018	SUB	000 0200
0019	BMI	000 0024
0020	LDA	000 0105
0021	MPY	000 0107
0022	STD	000
0023	BRU	000
0024	LDA	000 0100
0025	STO	000 0201
0026	RWD	000 0201

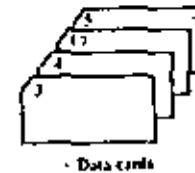
- With what location must the variable X be associated?
- What is the operand address that should be filled in for the BRU instruction shown at location 0023?
- What should be the value of the address field for the STO instruction at location 0022?
- If at location 0022 the STO were replaced by a BRU operation code, what then would be the appropriate value for the address field?

4. This question relates to the following SAMOS program and the four data cards displayed to the right of it; you are to assume that the given SAMOS program executes with the data cards shown. The DIV (divide) instruction produces an integer quotient (see Appendix A).

```

0000 RWD 000 0012
0001 RWD 000 0013
0002 LDA 000 0012
0003 DIV 000 0013
0004 MPY 000 0013
0005 STD 000 0014
0006 LDA 000 0012
0007 SUB 000 0014
0008 STO 000 0014
0009 RWD 000 0014
0010 BRU 000 0000
0011 HLT 000 0000
  
```

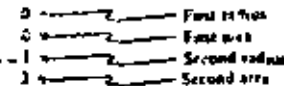
SAMOS program



- Which of the following is a *false* statement?
  - The instruction at 0011 will never be reached.
  - Only two values will be printed.
  - All four data cards will be read.
  - The first value printed will be 3.
  - Three values will be printed.
- Which of the following is a *true* statement?
  - The program will halt whenever the result of a division is zero.
  - The instruction at 0011 would be executed using the given set of data if the instruction at 0010 were revised to BMI 000 0000.
  - This program inputs two numbers, selects the larger, and prints its value.
  - All three of the above statements are false.

Problems 5 through 7. The following three problems involve programs to be written in SAMOS machine language and run on a computer using a SAMOS simulator. If you do not have a computer available, your final result will be a SAMOS coding form showing your program.

5. Draw a flowchart and write and run a SAMOS program to find the areas (to the nearest integer) of circles with each of the following radii: 0, 1, 2, 3, ..., 10, 11, 12. Use  $\pi = 22/7$ . The output is to consist of each radius value followed by the associated area, that is,



This SAMOS program should not execute any *input* steps. Storage locations will be required for instructions and for all constants, including 22, 7, the current radius, 1 (to increment the radius to the next value), and a number (11, 12, or 13, depending on your particular flow chart) to test against to determine when to branch to *halt*.

**Question** Will SAMOS give the same result when you compute  $(22/7) \times r^2$  as when you compute  $(22 \times r^2)/7$ ? If not, which gives a better result? Why?

Draw a flowchart and write and run a SAMOS program to do the following:

For the values from 1 to 10 inclusive (i.e.,  $1 \leq X \leq 10$ ), evaluate the following mathematical expression:

$$F = 5X^2 + 10X + 6$$

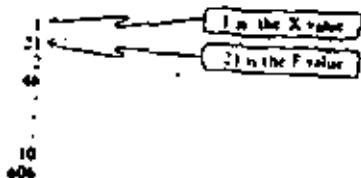
Print out the value for X and F after each evaluation.

**Example** The first value of X will be 1. For this value,

$$F = 5(1) + 10(1) + 6$$

$$F = 5 + 10 + 6 = 21$$

Thus the numbers 1 and 21 will be printed out, and F will then be evaluated for  $X = 2, 3, \dots, 10$ . The complete output will consist of 20 numbers:



**Note** Additional Information for Problem 6

1. No data cards will be needed for this program.
  2. The values from 1 to 10 need not all be stored at the beginning of the program.
  3. You must include some way to terminate your program after the final value has been processed and printed.
7. Draw a flowchart using variables C, X, TALLY, SUM, and AVG, and write and run a SAMOS program to do the following:
- (a) Read a value for the variable C.
  - (b) Read a value for the variable X.
  - (c) Check to see whether X equals 9999. If X does not equal 9999 then check whether X equals C. If X equals C, then return to

step (b). If X does not equal C, then add one to a counter called TALLY and add X to the variable SUM and then return to step (b).

If X equals 9999, then no more data cards are to be read. At this point print out the values of TALLY and SUM. Compute AVG, the quotient of SUM and TALLY ( $AVG = SUM/TALLY$ ). Print the value of AVG.

**Note** Additional Information for Problem 7

1. In your flowchart the average will be a variable AVG. The variable TALLY will hold a count of how many values of X are *not* equal to C. The assignment

$$TALLY \leftarrow TALLY + 1$$

will be needed. The sum of the values of X not equal to C will be called SUM. What should be the initial values of TALLY and SUM?

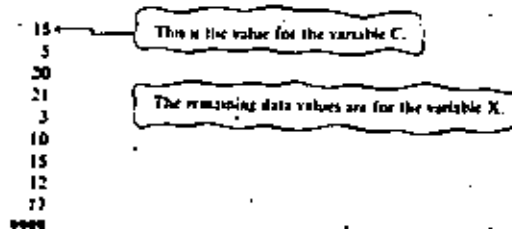
2. The value 9999 is called a *sentinel* value. Its purpose is to indicate that all the values of X have been read and processed. (In computer language a sentinel value is said to represent the end of file, i.e., the end of data.) Therefore, your data deck will consist of a value for C, the given values for X, and the value 9999. (See Section 2.2 for additional discussion of sentinels.)

3. In SAMOS the only conditional branch instruction is **BAII** (Branch On Minus). The programmer faces a problem when he needs to check whether the values of two variables are equal or whether the value of a variable is equal to some constant value. The following is one method of determining whether the values of the variables A and B are equal. First, subtract the value of B from the value of A and, if the result is not negative, subtract the value of A from the value of B. If this result is not negative, we can conclude that the values of A and B are equal.

**Examples**

- (a)  $A = 6$  and  $B = 9$   
 $A - B = 6 - 9 = -3$   
 Result:  $A \neq B$  since  $A - B$  is a negative number.
- (b)  $A = 5$  and  $B = 5$   
 $A - B = 5 - 5 = 0$   
 $F - A = 5 - 5 = 0$   
 Result:  $A = B$  since neither subtraction produced a negative number.
- (c)  $A = 4$  and  $B = 3$   
 $A - B = 4 - 3 = 1$   
 $B - A = 3 - 4 = -1$   
 Result:  $A \neq B$  since  $B - A$  is a negative number.

4. The data for Problem 7 are as follows:



54

Most of the storage capacity of a 61-bit SAMOS word goes unused when a small integer, for instance, 2, is represented. Conversely, even though a number is known to very great precision, a 61-bit word has a fixed capacity to represent digits. Character strings, such as names and addresses, for instance, vary greatly in length. In general, information comes in many sizes and lengths, and it would be exceedingly convenient to have computer storage responsive to this fact.

The SAMOS language is heavily influenced (i.e., constrained) by its *word-organized* storage system. We briefly mention here another kind of computer storage called BITOS (BIT-Organized SAMOS), whose storage is structured in a more flexible and natural way—natural for the processing of different types of information. The BITOS storage is best thought of as a single sequence of bits instead of a single sequence of words that are, in turn, sequences of bits. For example, a BITOS store roughly equivalent to the SAMOS store contains 610,000 bits whose addresses are 0 through 609,999 respectively. To fetch a unit of information of some known length, one must specify the "bit address" of the beginning of the desired information unit together with its "bit length." Thus,

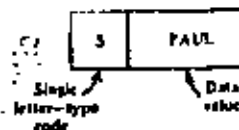
op code	address	length
LDA	24972	39

is the way one might write out a BITOS instruction to load the accumulator with a data value 39 bits long beginning at bit location 24972.

The information containers in a BITOS machine resemble the storage boxes of the conceptual model SIMPLoS in that the capacity of the containers is arbitrary.

55

There is a second important way in which the containers of SIMPLoS and BITOS resemble one another. In both cases, values stored in these containers are *self-describing*. Note that the SIMPLoS Reader is able to deduce that a storage box contains a character string value as opposed, say, to a numerical value, because he is able to see the quotation marks. The fetching mechanism of BITOS can convey the same type of information to its processing unit because the data object in each container consists of two parts: a code that describes the type or nature of the value and the data value itself. For example, suppose the container associated with X, is located at bit address 3901, and suppose character coding for the BITOS store employs the same 6-bit representation used in SAMOS. We might expect to see at that location:



where the type code S stands for string. Then, to represent a string of 4 characters would require 24 bits for the string itself and 6 more bits (for the letter S) to identify the 24 bits as a string. For a string of 91 characters, 546 bits are needed for the string and 6 more for the type code—or 552 bits in all. If a nonnegative integer variable, AGE, never requires more than three digits we can picture the corresponding BITOS container as



for a data value of 52. Here the type code I denotes integer.

To recapitulate, in BITOS one defines the size of the container to fit the need. That is, the store is divided up into containers that reflect their actual use. Every reference to a container consists of the bit address of the container and its length. The container itself holds as part of the data object a code that makes the remainder of the information in the container self-describing. Each time a new container is needed, a section of the store large enough to hold the required number of bits is "partitioned" for this purpose. When this container

is no longer needed, that section of store and others like it are repartitioned, that is, reused, typically in different container sizes, to suit new needs. To make an analogy with SIMPLoS, imagine that all storage boxes are constructed to fit dimensions of the data values they are to contain. (The Affixer who pastes the sticker on the storage box can also adjust the size of the box if necessary.)

Only a few of the ideas about SAMOS and BITOS need to be remembered. One of the important ideas is the sequential manner in which the computer works, that is, the step-by-step way in which the computer performs its tasks. The order in which the tasks are performed is just as important as what is accomplished.

Another property of computers that we must understand is the *finite word length*. We have seen that SAMOS words consist of 10 characters and a sign, so that the largest number representable in this coding system is

$$+9,999,999,999$$

a rather large number, but still finite. Although BITOS store may use very "long" containers, they are still finite, so the limitation on what can be represented, although less constricting, still exists in principle. From a practical viewpoint, integer containers, whether in a SAMOS-like or in a BITOS-like store, are sometimes very unsuitable. Consider a variable that, from time to time, has various values assigned to it, sometimes very small integers and at other times very large integers. The storage container for such a variable cannot always be used efficiently if it must be large enough for the largest possible integer value that will be assigned to it.

To cover this situation there are other ways of coding numbers that not only solve this problem but also allow us to work with real numbers as well as integers. One of the most common of these alternate codings is *floating-point form*, which is related to the so-called "scientific notation."

To see how this works, recall that any decimal numeral such as

$$-382.519$$

can be expressed as

$$-382519 \times 10^3$$

in which there is a decimal point (just after the sign, if any) followed by a string of digits (the first not zero) and multiplied by a suitable power of 10. We can code numbers in this way by reserving three character positions for the exponent. The result is shown in Figure 1-34 for  $-382.519$ .

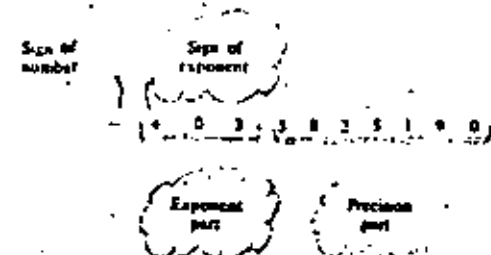


FIGURE 1-34  
Anatomy of a floating-point number for a fixed-word size store.

Some examples of how to code numbers including integers in this system are given in Figure 1-35. In this figure, we see that the 8-digit representation of  $\pi$ , as given in the first column of the third entry, has to be chopped to 7 digits of precision because of space requirements. The same holds true for  $1/3$  and  $11/7$  at the bottom of the table. Thus we see that in a computer even a simple fraction such as  $1/3$  cannot be represented exactly, but only to a close approximation. This characteristic of "finite word length" presents important prob-

Number	Floating-Point Form	Coding of Floating-Point Form
4	$4 \times 10^0$	+ + 014000000
-999999000	$-999999 \times 10^3$	- + 0999999990
3.1415926	$3.1415926 \times 10^0$	+ + 013141592
-273.14	$-27314 \times 10^2$	- + 032731400
.0008761	$.8761 \times 10^{-3}$	+ - 038761000
.73	$.73 \times 10^0$	+ + 007300000
1	$.33333333 \times 10^0$	+ + 003333333
4	$.157142857 \times 10^0$	+ + 041571428

FIGURE 1-35  
Floating-point coding of numbers in a fixed-word-sized store.



## EXERCISE 1-9

- Several million pocket-sized electronic calculators are now being produced annually. They are becoming relatively accessible to the average student. Many of these calculators, such as the one shown in Figure 1-36, use floating point arithmetic.

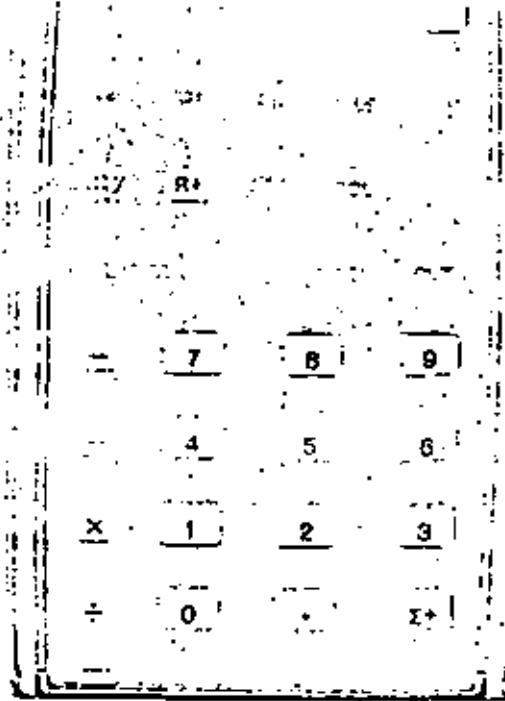


FIGURE 1-36  
A popular hand-held calculator.

- Locate an electronic calculator and compare the method it uses to represent floating-point numbers with the method used in SAMOS.

lems that will be discussed in various places in this text, especially Chapter 11.

In floating-point form we can represent large numbers, but for computers such as SAMOS with fixed-word size stores the price we pay is giving up three places of precision. Were SAMOS to use floating-point numbers, the largest number representable in floating-point form would be:

.	.	.	.	.	.	.	.	.	.	.
---	---	---	---	---	---	---	---	---	---	---

which represents the number

999,999,900,000,000,000,000,000,000  
 000,000,000,000,000,000,000,000,000  
 000,000,000,000,000,000,000,000,000  
 000,000,000,000,000,000,000,000,000

Similarly, there is a smallest positive number that could be represented:

.	.	.	.	.	.	.	.	.	.	.
---	---	---	---	---	---	---	---	---	---	---

.000 000 000 000 000 000 000 000 000  
 000 000 000 000 000 000 000 000 000  
 000 000 000 000 000 000 000 000 000  
 000 000 000 000 000 000 1

which is very small, indeed.

Coding in floating-point form for a BITOS-like machine could be quite similar to the scheme shown in Figure 1-35. On the other hand, since the size of the container may be chosen to fit the particular "needs" of a given variable, the size of the precision part could easily be permitted to vary as required. For that matter, the size of the exponent part could also be expanded or contracted to fit the need.

In summary, both SAMOS-like and BITOS-like machines are often built to operate on numbers coded in floating-point form. However, in our discussions of SAMOS, in particular in the description given in the Appendix, the machine is initially described as if it were not capable of dealing with numbers coded in floating-point form, but only with numbers coded as integers.



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

**INTRODUCCION A LA PROGRAMACION Y COMPUTACION ELECTRONICA**

**INTRODUCCION A LA COMPUTADORA DIGITAL**

**OCTUBRE, 1964**

specifying ADD WORD (addressed by) A AND WORD (addressed by) B; PUT RESULT INTO MEMORY LOCATION (addressed by) C. But specification of three separate addresses would make the instruction word too long (even for a large digital computer, not to speak of a minicomputer). We can, however, implement the above operation in terms of several simpler instructions each referencing only a single memory address:

- LOAD INTO ACCUMULATOR (the word addressed by) A
- ADD INTO ACCUMULATOR (the word addressed by) B
- STORE ACCUMULATOR (in memory location addressed by) C

The "basic" minicomputer discussed in this chapter, then, will be a single-address machine whose instructions move data between a single suitably addressed memory location and a specified processor register, or possibly between two such registers. There will also be some instructions which do not reference memory at all (e.g., COMPLEMENT ACCUMULATOR). We remark, however, that the possibility of using simplified two-address instructions and zero-address (stack) instructions in minicomputers is of the greatest interest and will be discussed in connection with more advanced designs in Chap. 6.

2-3. The "Basic" Minicomputer. Figure 2-1 illustrates the typical organization of a small digital computer. The machine has all the ingredients of Secs. 1-6 to 1-11, viz.,

1. A core or semiconductor memory, which will store instructions and data
  - (a) Memory buffer register (memory data register): contains the instruction or data word currently leaving or entering the memory
  - (b) Memory address register: contains the address of the currently addressed memory location
  - (c) Program counter: contains the address of the instruction to be executed
  - (d) Instruction register: contains the current instruction
  - (e) General-purpose register (accumulator, arithmetic register) or registers; and, possibly, an index register (Sec. 2-7)
  - (f) One-bit registers ("flags"): indicates overflow, carry, sign bit, etc., resulting from past or current operations
2. An arithmetic/logic unit: logic circuits to combine words from two registers by addition, subtraction, bit-by-bit ANDing, etc., and to complement, shift, etc., single words
3. Control logic: decodes the 0s and 1s of the instruction currently in the instruction register to generate logic levels and time pulses, which:
  - (a) Gate (steer) words between processor registers.
  - (b) Determine the function of the arithmetic/logic circuits

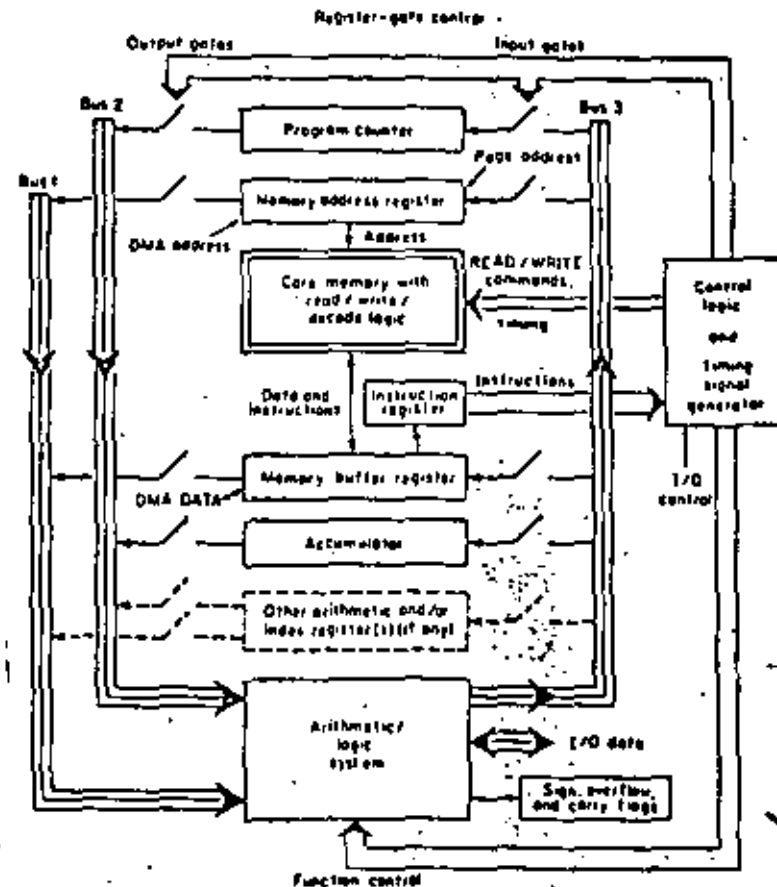


Fig. 2-1. Organization of a "basic" single-address minicomputer.

In Fig. 2-1, there are three buses for transfers between registers, always via the arithmetic/logic unit. This is a practical compromise: some extra register-to-register paths would permit more concurrent register transfers and speed computation, but we would pay for more complex interconnections and logic.

Finally, we must have input/output connections through the arithmetic/logic unit or through register gates.

2-4. Processor Operation. (a) Instruction Fetching. If we assume that a suitable program and data are in memory, processor operation proceeds

# COMPUTER STRUCTURES: HEADINGS AND EXAMPLES

Poll & Howell, Aldermaston 1971

Poll & Howell, Aldermaston 1971

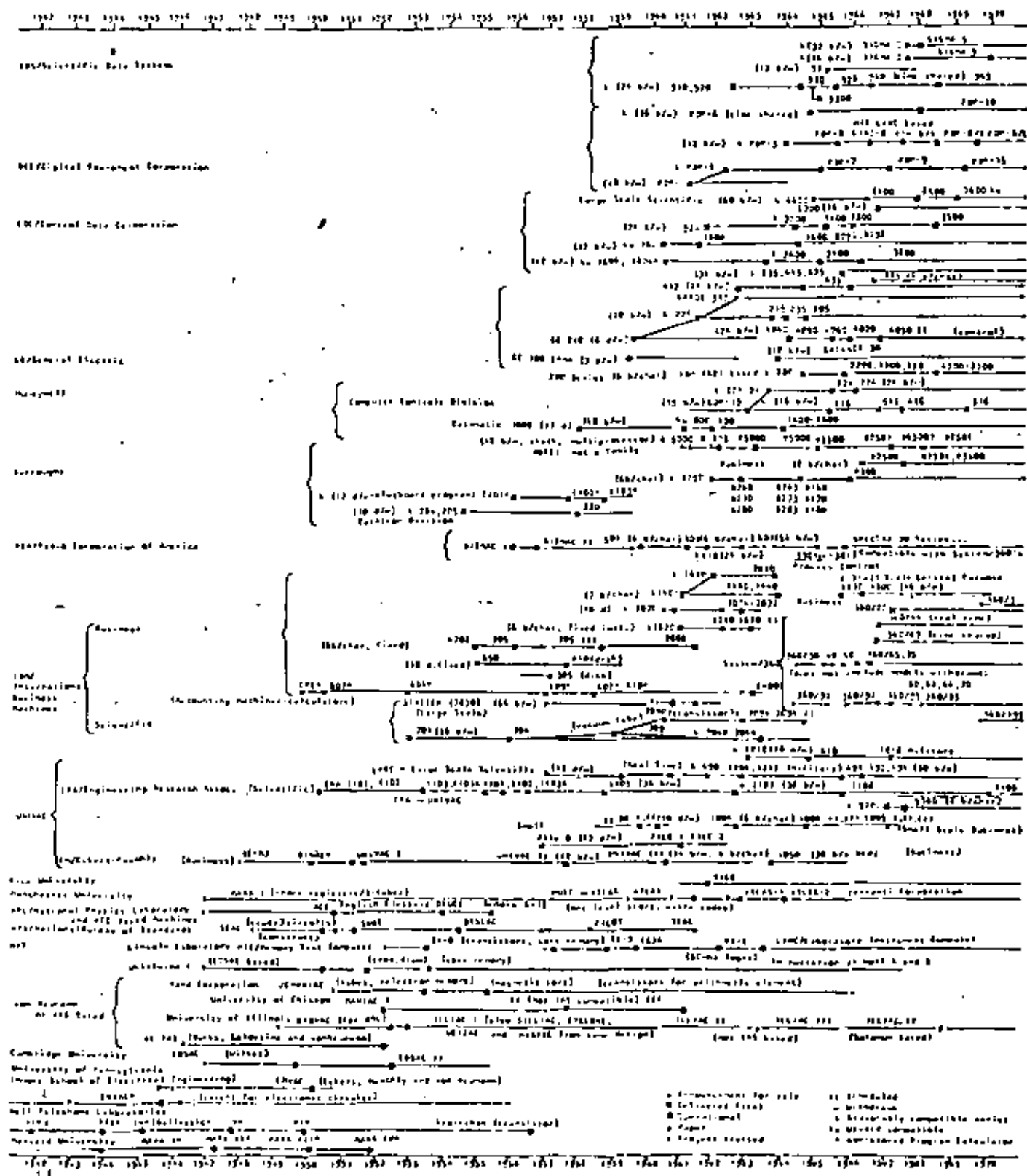


Fig. 2a. Time chart, computers by originator.

16 LENGUAJES DE PROGRAMACIÓN

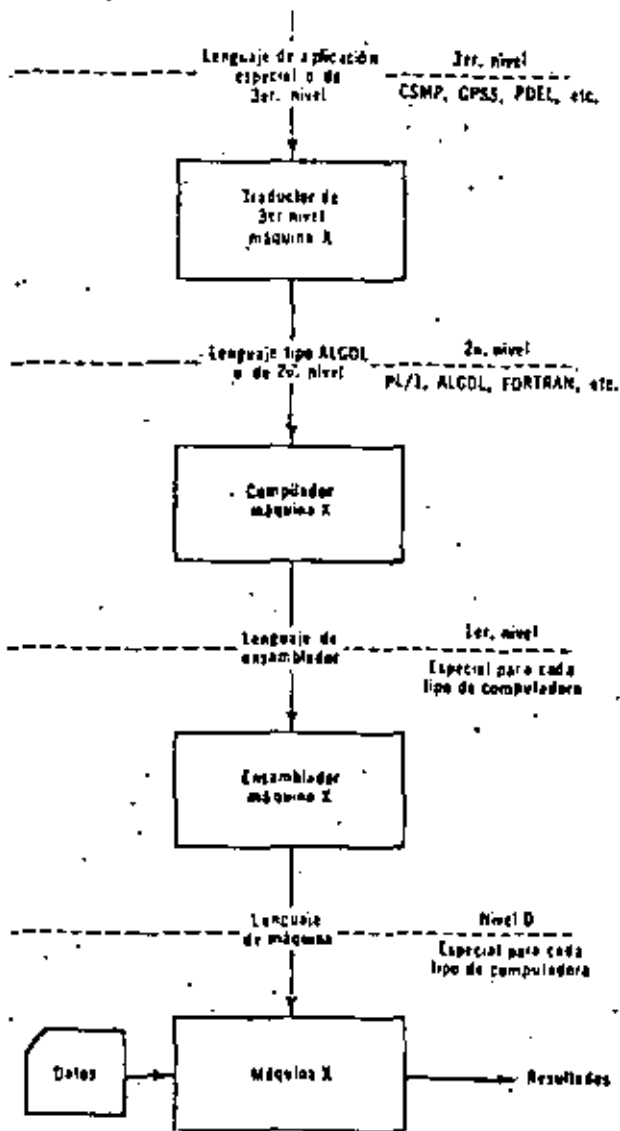


Fig. 1. El proceso de traducción común.

mente, reduciendo, muchas veces, aún más el tiempo total de solución de un problema en el área para la cual ha sido diseñado. Los llamados traductores de tercer nivel convierten tales lenguajes a lenguaje

Característica	I. Tubo de vacío	II. Transistores, diodos de estado sólido	III. Circuitos integrados	IV. Redes integradas a gran escala
Aparición en sistemas	1ª Generación 1935	2ª Generación 1945	3ª Generación 1953	4ª Generación 1969
Complejidad de la parte representada	1 elemento de circuito	1 elemento de circuito	4 a 100 elementos de circuito	más de 1,000 elementos de circuito
Volumen físico de un chip-chip (pulgadas cúbicas)	100	5	0.25	0.02
Velocidad operativa (Miles/seg)	10	1,000 (en 1947)	10,000 (en 1967)	25,000 (en 1969)
Consumo de fuerza (watts)	5	1	0.10	0.04
Costo de un chip-chip dólares	\$ 10.00	\$ 15.00	\$ 5.00	\$ 2.00
Porcentaje relativo de fallas	1	0.01	0.001	0.001 (proyectado para 1972)

Fig. 1 Características de cuatro generaciones de componentes electrónicos.



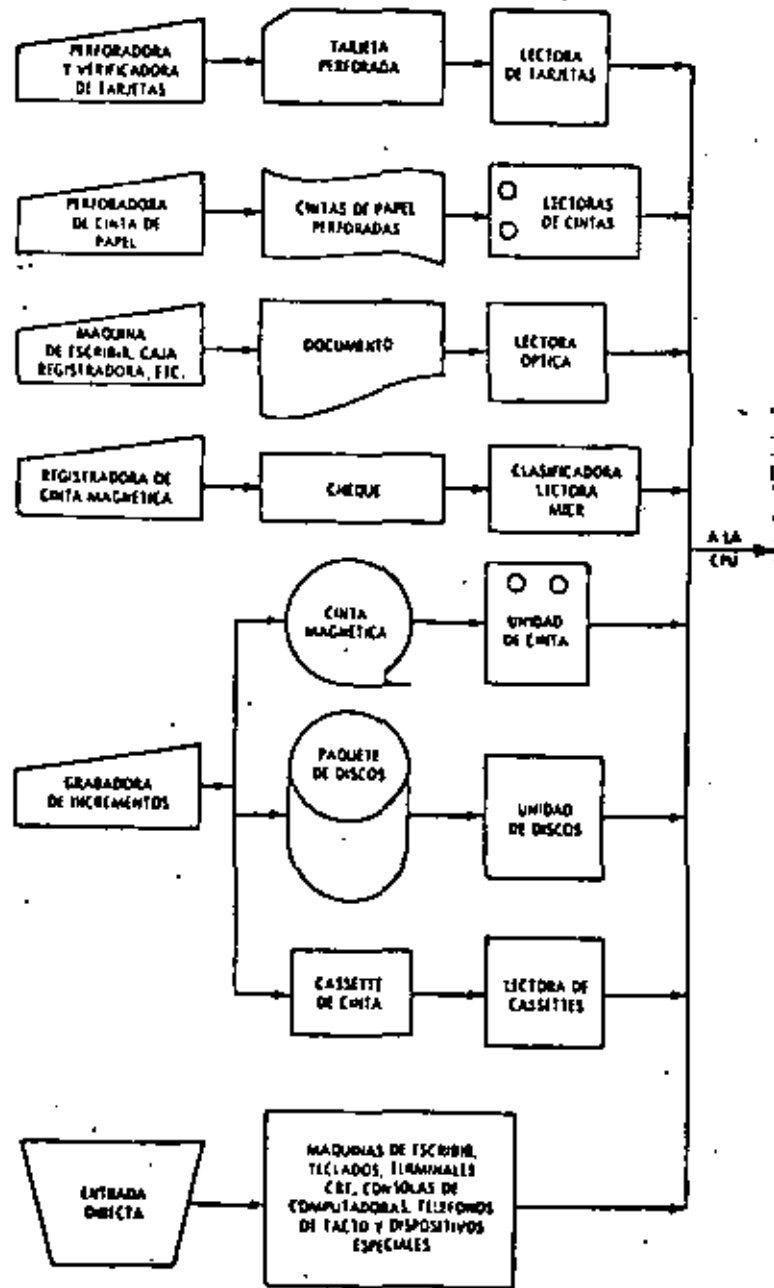


Fig. 8-1: Presentación de dispositivos y medios de entrada.

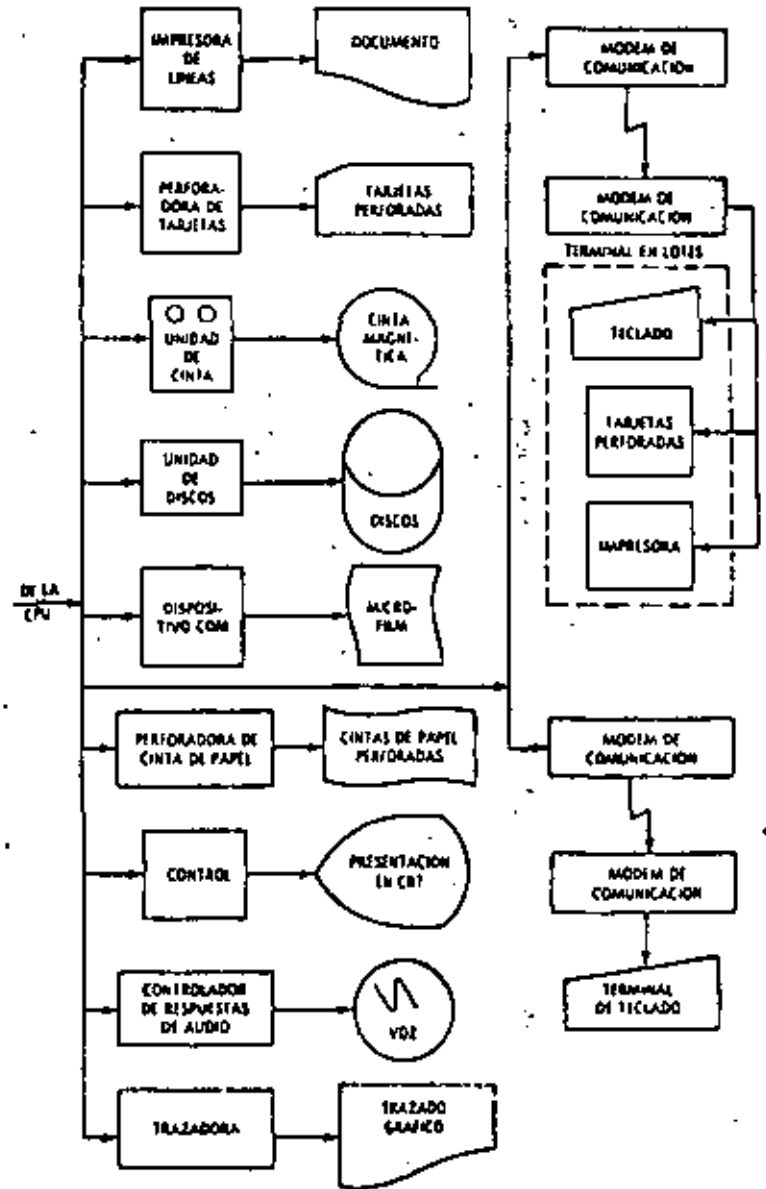


Fig. 8-2: Presentación de dispositivos y medios de salida.



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

**INTRODUCCION A LA PROGRAMACION Y  
COMPUTACION ELECTRONICA**

**L E N G U A J E F O R T R A N**

**ING. HERIBERTO OLGUIN ROMO  
ING. RICARDO CIRIA MERCÉ**

**OCTUBRE, 1982**

HERIBERTO OLGUIN ROMO

RICARDO CIRIA MERCE

## FORTRAN

## ELEMENTOS DE UN SUPERLENGUAJE DE PROGRAMACION: FORTRAN

- 1.- Introducción al lenguaje FORTRAN
  - 1.1 El alfabeto
- 2.- Números
  - 2.1 Constantes enteras
  - 2.2 Constantes reales
  - 2.3 Variables enteras
  - 2.4 Variables reales
- 3.- Operaciones aritméticas
- 4.- Expresiones aritméticas
  - 4.1 Reglas para las expresiones aritméticas
  - 4.2 Funciones predefinidas disponibles en lenguaje FORTRAN
- 5.- Enunciados
  - 5.1 Los enunciados aritméticos de asignación
  - 5.2 Los enunciados de control
    - 5.2.1 El enunciado GO TO

- 5.2.2 El enunciado IF
- 5.2.3 El enunciado DO
- 5.2.4 El enunciado STOP
- 5.3 Los enunciados de entrada y salida
  - 5.3.1 El enunciado READ
  - 5.3.2 El enunciado WRITE
- 5.4 Los enunciados de especificación
  - 5.4.1 El enunciado FORMAT
    - 5.4.1.1 La especificación I : Iw
    - 5.4.1.2 La especificación F : Fw.d
    - 5.4.1.3 La especificación E : Ew.d
    - 5.4.1.4 La especificación A : Aw
    - 5.4.1.5 La especificación T : Tw
    - 5.4.1.6 Las especificaciones L, ll y /
  - 5.4.2 El enunciado END
- 6.- Arreglos
  - 6.1 Variables con subíndices
    - 6.1.1 Reglas para los subíndices
  - 6.2 El enunciado DIMENSION
    - 6.2.1 Reglas para las variables con subíndices
  - 6.3 Arreglos de entrada y salida
- 7.- SUBPROGRAMAS
  - 7.1 Funciones
    - 7.1.1 Ejemplos
  - 7.2 Subrutinas
    - 7.2.1 COMMON



1.- Introducción al lenguaje FORTRAN

El lenguaje FORTRAN, cuyo nombre corresponde a las primeras letras de las palabras inglesas FORMula (fórmula) y TRANslation (traducción), es un lenguaje de programación orientado a problemas matemáticos y se emplea en casi todas las computadoras del mundo. Debido a su parecido con el lenguaje aritmético común, el FORTRAN simplifica la preparación de problemas que pueden resolverse mediante una computadora. Los datos e instrucciones se pueden organizar mediante una secuencia de enunciados fortran; estos constituyen el llamado Programa Fuente.

Todas las computadoras que "entienden" el lenguaje FORTRAN, tienen lo que se llama un Compilador Fortran, llamado también traductor o intérprete, el cual analiza los enunciados fortran y los traduce a un Programa Objeto, el cual queda en lenguaje de Máquina.

Un programa escrito en lenguaje FORTRAN se puede procesar en cualquier máquina que tenga un Compilador Fortran. Esto nos indica que el lenguaje es independiente para cada máquina, o sea que el compilador se debe preparar en cada caso teniendo en cuenta la máquina que ha de usarse en particular; puesto que las máquinas difieren en su organización interna, se ha desarrollado un número de "dialectos" del lenguaje FORTRAN, cada uno de los cuales es apropiado para una clase de máquinas. Las diferencias entre los

varios dialectos son mínimas y se ajustan el uno al otro fácilmente.

1.1 El alfabeto

El alfabeto FORTRAN está constituido de caracteres que son símbolos familiares de escritura y de teclados de máquinas de escribir, así como de dispositivos especiales de perforación; dichos caracteres son:

- Alfabéticos: A B C D E F G H  
I J K L M N  
\* P Q R S T U V W X Y Z
- Númericos: 0 1 2 3 4 5 6 7 8 9
- Símbolos: + - \* / = . , ( ) \* @

De este alfabeto se construyen todos nuestros símbolos, expresiones y enunciados que se utilizan en el lenguaje FORTRAN.

2.- Números

Los números pueden representarse en diferentes formas, las cuales se asemejan a los símbolos de la aritmética general; pero debido a la estructura interna de las computadoras se establecen las convenciones de: Punto Fijo y Punto Flotante que proporcionan facilidades para su manejo en FORTRAN. Los símbolos de punto fijo

\* La letra O la expresaremos como @ para diferenciarla del 0 cero.

se usarán solamente con números enteros y los cálculos asociados se denominarán aritmética de los enteros o modo entero; mientras que la aritmética de los números reales se hará en la forma de punto flotante y se llamará aritmética de los reales o modo real. Debido a que también es necesario distinguir las constantes (números que no cambian durante toda la ejecución de un programa) de las variables (números que pueden cambiar), surgen cuatro clases de símbolos para los números.

### 2.1 Constantes enteras

Dependiendo del tipo de computadora se podrán representar por un cierto número de dígitos, así para IBM-1130 se representan mediante cinco dígitos sin el punto decimal. Si el entero es negativo, los dígitos deberán ser precedidos del signo menos; si el entero es positivo el signo es opcional.

Ejem. Símbolos para constantes enteras pueden ser entre otras:

1976 +1 0 +1976 -1976

Símbolos que no se aceptan para constantes enteras:

7483282 (más de cinco dígitos)

1976. (el punto decimal no se permite)

### 2.2 Constantes reales

Dependiendo del tipo de computadora, las constantes reales se podrán representar por varios dígitos, pero en el caso de la

IBM-1130 sólo se admiten siete dígitos con punto decimal pudiéndose colocar al principio de los dígitos, al final o entre dos dígitos cualesquiera. Cuando aparece un punto en una constante su tratamiento será de punto flotante. Si la constante real es precedida de un signo menos, se indicará que es negativa, si es positiva el signo es opcional.

Ejem. Símbolos para constantes reales pueden ser entre otras:

1976. - .00001976 +12.345 -12.345

-.007 .007 5.348 0.3

Símbolos que no se aceptan para constantes reales:

123456789.32 (más de siete dígitos significativos)

5343 (falta el punto decimal)

Para representar las constantes reales existe también la llamada forma exponencial; esta la podemos representar mediante una letra E y una constante entera de uno o dos dígitos, positiva o negativa. Esta constante entera es un exponente del número diez; el signo menos es para los exponentes negativos y para los positivos, el signo es opcional. En FORTRAN, la presencia del exponente hace que el uso del punto decimal sea opcional.

Ejem. Forma exponencial Forma no exponencial

1.328E2 132.8

1.328E02 132.8

1.328E00	132.8
-4.724E-03	-.004724
+7.61E3	7610.
-6432E-3	-6.432

### 2.3 Variables enteras

Estas se representan por combinaciones de una a cinco letras y dígitos (IBM-1130), no se permiten otros caracteres y el primer caracter deberá ser una de las letras I, J, K, L, M ó N. El primer caracter de una variable es el que indica si es entera o real. Durante la ejecución de un programa, las variables enteras deberán restringirse a valores enteros.

Ejem. Símbolos para variables enteras pueden ser, entre otros:

NUMCT	KILO	N1	N2	M10	KONT
FIJALC	JCLAV	NARY	KONT1	L1976	

Símbolos no aceptables para variables enteras:

CUENT (el primer caracter debe ser I, J, K, L, M ó N)

KONTADOR (demasiados caracteres)

12.34 (sólo se aceptan letras y números)

### 2.4 Variables reales

Estas se representan por combinaciones de una a cinco letras y dígitos (IBM-1130), no se permiten otros caracteres y el primer

caracter tiene que ser necesariamente una letra diferente a I, J, K, L, M ó N. Durante la ejecución de un programa dichas variables se deben restringir a valores reales.

Ejem. Símbolos para variables reales pueden ser, entre otros:

FUENZ	VELOC	ACEL1	CUENT	A1	A2
ALFA	VIELA	RA42	X1	PROD	SUMA

Símbolos no aceptables para variables reales:

A3.8 (el punto no es letra o número)

CORRIEN (demasiados caracteres)

3 BASD (el primer caracter debe ser una letra)

MUNCT (el primer caracter no pueda ser M)

### 3.- Operaciones aritméticas

Las operaciones aritméticas y los símbolos que se utilizan en FORTRAN son:

	Ejem.	Algebra	FORTRAN
Adición	+	$a + b$	A + B
Sustracción	-	$a - b$	A - B
Multiplicación	*	$a * b$	A * B
División	/	$\frac{a}{b}$	A / B
Exponenciación	**	$a^b$	A ** B

### 4.- Expresiones aritméticas

En base a lo expuesto anteriormente podemos ahora formular

expresiones aritméticas en lenguaje FORTRAN y nos daremos cuenta que son muy similares a las expresiones aritméticas del álgebra común.

Expresiones FORTRAN	Expresiones Comunes
$A**2-B**2$	$a^2-b^2$
$B**2-4.*A*C$	$b^2-4ac$
$(A+B)/2.$	$\frac{1}{2}(a+b)$
$2*K-J+N$	$2k-j+n$
$C+B-3.*A$	$c+b-3a$

#### 4.1 Reglas para las expresiones aritméticas

Las reglas a las que debemos sujetar las expresiones aritméticas son necesarias debido a la estructura de las computadoras y al observarlas tendremos un ahorro en el tiempo de ejecución de un programa.

**Regla 1** Si nos fijamos en las expresiones FORTRAN anteriores nos damos cuenta que: Todas las constantes y variables en una expresión deben estar en el mismo modo, esto es, todas deben ser enteras o todas deben ser reales. (Como toda regla existe su excepción que mencionaremos más adelante). Es necesario consultar los manuales de cada máquina, ya que como hemos mencionado anteriormente dependerá esta regla del tipo de computadora. Por lo pronto la consideraremos como se ha indicado.

#### Regla 2

$A**i$ ,  $i**j$  y  $A**B$  son exponenciaciones permitidas. En el caso  $A**i$  se mezclan los modos y es la excepción a la Regla 1, pero sabemos que esta exponenciación significa multiplicaciones sucesivas (así  $B**3=B*B*B$ ), mientras que las potencias no enteras implican cálculos más sofisticados. Nos damos cuenta que  $i**A$ , no es forma de exponenciación permitida (en algunas máquinas sí se permite).

#### Regla 3

Deberá tenerse en cuenta que las operaciones se ejecutarán con las siguientes prioridades:

- 1) Las operaciones indicadas dentro de los paréntesis más internos se ejecutan en primer lugar.
- 2) Exponenciación.
- 3) Multiplicación y división.
- 4) Adición y sustracción.

Entre las operaciones de igual prioridad, el orden de ejecución es de izquierda a derecha.

Ejemplo. Si  $A=5$ ,  $B=8$ , y  $C=2$ .

$A+B-3.*C$  se calculará en el siguiente orden:

$$3.*2.=6. \quad 5.+8.=13. \quad 13.-6.=7.$$

$B**2-4.*A*C$  se calcula en el siguiente

orden:

$$8.**2.=64. \quad 4.*5.=20. \quad 20.*2.=40.$$

$$64.-40.=24.$$

Si  $A=5$ ,  $B=8$ ,  $C=2$ , y  $D=1.6$

Entonces  $(A+B)/C$  se calcula en el siguiente orden:

$$5+8=13 \quad 13/2=6.5$$

Mientras que  $A+B/C$  se calcula en el siguiente orden:

$$8/2=4 \quad 5+4=9$$

Ahora si deseamos calcular  $(A+C)**2$  conducirá a:

$$5+2=7 \quad 7**2=49$$

Mientras que  $A+C**2$  conducirá a:

$$2**2=4 \quad 5+4=9$$

Ahora si:  $(A*B)/(C*D)=40./3.2=12.5$

Entonces:  $A*B/C*D=40./C*D=20.*D=32.$

Finalmente si tenemos paréntesis dentro de otros paréntesis se tiene:

$$(A*(B+C))**2=(A*10)**2=50.**2=2500.$$

$B+C$  tiene la más alta prioridad por encontrarse en el paréntesis más interno.

$$(A*B+C)**2=(40.+2)**2=42.**2=1764.$$

$$A*(B+C)**2=A*10.**2=AA100.=500.$$

$$A*B+C**2=A*B+4.=40.+4.=44$$

Debemos tener cuidado en expresar lo que deseamos realizar.

#### Regla 4

No deberemos colocar un signo de operación antes de un signo más o menos, esto es, no deberemos poner dos signos de operación juntos.

Ejem.  $A*-B \quad I+(-J) \quad M-(-N) \quad A/(-B)$

Estas expresiones deberán sustituirse por:

$$A*(-B) \quad I+(-J) \quad M-(-N) \quad A/(-B)$$

#### 4.2 Funciones predefinidas en lenguaje FORTRAN

Estas funciones predefinidas que proporciona el lenguaje FORTRAN son de tipo de biblioteca. Para utilizarlas usaremos el nombre de la función seguido de un argumento que deberá estar entre paréntesis. Dichos argumentos pueden ser variables simples ó con subíndices, constantes, expresiones aritméticas u otras funciones predefinidas en FORTRAN.

Para IBM - 1130 tenemos:

NOMBRE	FUNCION EJECUTADA	NUM. DE ARGUMENTOS	TIPO DE ARGUMENTO(S)	TIPO DE FUNCION
SIN	Seno trigonométrico (argumento en radianes)	1	Real	Real
COS	Coseno trigonométrico (argumento en radianes)	1	Real	Real
ALOG	Logaritmo natural	1	Real	Real
EXP	Argumento de potencia del número e	1	Real	Real
SQRT	Raíz cuadrada	1	Real	Real
ATAN	Arco tangente	1	Real	Real

del número e

ABS	Valor absoluto	1	Real	Real
IABS	Valor absoluto	1	Entero	Entero
FLDAT	Convertir argumento de entero a real	1	Entero	Real
IFIX	Convertir argumento de real a entero	1	Real	Entero
SIGN	Transferencia de signo (Arg.1 recibe signo de Arg.2)	2	Real	Real
ISIGN	Transferencia de signo (Arg.1 recibe signo de Arg.2)	2	Entero	Entero
TANH	Tangente Hiperbólica	1	Real	Real

Ejem.  $SQRT(B^2 + A^2 + 4 * A * C)$  indica que a lo que se encuentra entre paréntesis se le sacará la raíz cuadrada.  
 $SIN(BETA)$  indica que se obtendrá el seno trigonométrico de el valor de la variable BETA.

## 5. Enunciados

Los enunciados son las unidades básicas con las cuales se construyen los programas FORTRAN. Podemos clasificarlos de acuerdo a su función en grupos como:

- 1.- Aritméticos de asignación
- 2.- De control
- 3.- De entrada y salida
- 4.- De especificación

### 5.1 Los enunciados aritméticos de asignación

Se forman con las expresiones presentadas anteriormente y

nos indican los cálculos particulares que deben hacerse. Su forma es:

$$\text{Variable} = \text{Expresión aritmética}$$

El significado del signo = es el de asignación, esto es, que deberá calcularse el valor de la expresión a la derecha del signo = y su valor se asignará a la variable que se encuentre a la izquierda del signo, la cual tiene una localidad en la memoria de la computadora.

Ejem.  $S1=A-5.$ ,  $B=8.$ ,  $C=2.$  y  $D=1.6$

$X=(A+B)/C$  se le asignará a la X el valor 6.5

$ALD=(A+B)*A^2$  se le asignará a ALD el valor 169.

$RAI=SQRT(B*C)$  se le asignará a RAI el valor 4.

Algo diferente al álgebra normal es el enunciado  $A=A+3.$  el cual no debe alarmarnos ya que indica que a la localidad de memoria con el nombre A se le asignará el nuevo valor A+3, esto es:

Si  $A=5.$  y  $A=A+3.$  entonces:

$A=5.+3.$   $A=8.$  o sea que la variable A se le asigna el valor de 8, y el valor anterior que fué 5, se pierde.

### 5.2 Los enunciados de control

Debido a que los enunciados de un programa FORTRAN se ejecuten en el orden que aparecen y que en muchas ocasiones queremos transferir la ejecución a otros enunciados si se satisface una

cierta condición, FORTRAN nos permite numerar dichos enunciados. Un número de enunciado debe ser una constante entera de uno a cinco caracteres, sin el signo más o menos; el número se coloca a la izquierda del enunciado.

Ejem. 3 CONT = CONT+1.

24 RAIZ = SQRT(A\*\*2+B\*\*2)

5.2.1 El enunciado GO TO

Este toma la forma GO TO N en donde N es un número de enunciado.

El GO TO produce un salto incondicional; así GO TO 3 envía la ejecución al enunciado número 3 que puede ser la instrucción de conteo del ejemplo anterior. GO TO 24 pasa el control al enunciado 24 que puede ser el del ejemplo anterior.

Ejem. Supongamos que unos de los enunciados de un programa son:

1 = 1            Esto nos representa la suma de  
 ISUM = 0        los números enteros, desde luego  
 1 ISUM = ISUM+1    es necesario ponerle otros enuncia-  
 1 = 1+1        dos pero por el momento nos aclara  
 GO TO 1        lo indicado.

5.2.2 El enunciado IF

Debido a que las computadoras están diseñadas a base de circuitos lógicos y el pensamiento del ser humano debe

ser de este tipo, nos concretaremos el IF lógico, además de que el mismo ya tiene elementos de algunos operadores de relación como OR, AND y NOT.

El IF lógico es de la forma:

IF (L) S

L = expresión lógica que puede tener dos valores: Verdadero o Falso.

S = cualquier enunciado FORTRAN diferente de: un GO, un enunciado de especificación o de otro IF lógico.

Si L es falso (.FALSE.) entonces se ignora S y la computación continúa al siguiente enunciado. Si L es verdadero (.TRUE.) el enunciado S se ejecuta en seguida.

Resulta interesante hacer notar que si L es relativamente complicada, éste IF puede ser el equivalente de varios IF aritméticos.

Para formar las expresiones lógicas (L) utilizaremos los operadores de comparación y los de relación:

Operadores de comparación:

<u>Símbolo Matemático</u>	<u>Significado</u>	<u>Símbolo FORTRAN</u>	<u>Significado Inglés</u>
<	Menor que	.LT.	Less than
>	Mayor que	.GT.	Greater than
≤	Menor o igual a	.LE.	Less or equal
≥	Mayor o igual a	.GE.	Greater or equal

=	Igual a	.EQ.	Equal
≠	Diferente a ó No Igual a	.NE.	Not equal
Operadores de relación:			
∪	Unión	.OR.	ó ('o inclusive)
∩	Intersección	.AND.	y ('al mismo tiempo)
~	Complemento	.NOT.	no

Para evaluar una expresión lógica se hará con las siguientes prioridades:

- 1.- Expresiones entre paréntesis
- 2.- Operadores aritméticos
- 3.- Operadores de comparación (.LT., .GT., .LE., .GE., .EQ. y .NE.)
- 4.- .NOT.
- 5.- .AND.
- 6.- .OR.

En caso de igual jerarquía la evaluación será de izquierda a derecha.

Ejem. (1) X=5. Y=0.5

IF (X.GT.3..AND. Y .LE.2.) Z=X\*\*3+X\*Y

Significa que si X>3. y (al mismo tiempo) y<=2. se asignará a Z el valor que se obtenga al calcular  $X^3+XY$ , esto es  $Z=125.+2.5=127.5$

(2) IF (A.LE.X.AND.B.GE. Y .OR.C.GT.2) GO TO 12

Significa que si A<=X y (al mismo tiempo) B>Y es

verdadero ó C>2 es verdadero ó ambos, entonces se transfiere el control al enunciado 12.

(3)

I = 1

ISUM = 0

Esto nos indica

1 ISUM = ISUM+I

que sólo sumare-

I = I+1

mos los números

IF (I.LE.100) GO TO 1

enteros del 1 al 100

STOP

5.2.3 El enunciado DO

Este toma la forma:

DO K I = L, M, N

GO K I = L, M

La segunda forma sólo se aplica cuando M=1, lo que es bastante frecuente.

K represente un número de enunciado

I represente una variable entera

L, M, N son variables enteras ó constantes sin

signo.

El DO produce la ejecución repetida de todos los enunciados que le siguen, hasta el enunciado número K. La primera vez que se ejecutan estos enunciados la variable I es igual a L, en cada paso subsiguiente I se incrementa en la cantidad M, hasta hacerse mayor ó igual a N en el paso final; en este momento se termina el llamado



lazo DØ y el control pasa al enunciado que está a continuación del enunciado K. Así, L es el valor inicial de la variable I y M su valor final. I se llama el índice del enunciado DØ y su valor corriente se puede usar en cálculos durante la ejecución del lazo. Todos los enunciados que le siguen al DØ hasta el número K inclusive constituyen el rango del DØ. También es posible que la variable I no se encuentre en ninguno de los enunciados del rango del DØ y esto nos indica que se realice la ejecución de todos los enunciados del rango del DØ M entre M veces (la parte entera de este cociente M/N). Debemos tomar en cuenta que: el índice I se incrementa sucesivamente y automáticamente durante la ejecución del lazo y que se puede, en estos momentos, tratar como cualquier variable entera; el índice I queda indefinido después de terminado el lazo del DØ y puede utilizarse para cualquier uso general. El enunciado K no debe ser un enunciado de especificación ni una transferencia de control esto incluye cosas como GØ TØ, IF y DØ, así como FORMAT, END y algunos otros. Debemos considerar que no se puede desde ningún punto del programa llegar a un enunciado dentro del rango de un DØ. Y que la entrada a un DØ deberá hacerse a través del enunciado DØ. Y por último es muy frecuente que un DØ esté completamente dentro de otro.

Ilustrando graficamente tenemos:

Correcto

Incorrecto

DØ DØ

DØ DØ

DØ DØ

DØ DØ

DØ

DØ

Ej. Utilizaremos un DØ para sumar los números enteros del 1 al 100, ejemplo que ya hemos visto anteriormente.

ISUM = 0

Nos damos cuenta que el DØ tiene la misma función que un IF,

DØ I I = 1,100

un GØ TØ y un contador; como podrá observarse con el ejemplo anterior.

ISUM = ISUM+1

STØP

STØP

5.2.4 El enunciado STØP

Este aparece simplemente como STØP y es el que nos indica que ha terminado la ejecución y en el caso de IBM - 7130 la computadora se detiene y el operador tendrá que hacer que continúe trabajando. Debido a ello se recomienda que se utilice el enunciado CALL EXIT, el cual pasa el control a un programa monitor que hace que la computadora continúe ejecutando los otros programas que siguen a continuación.

Tanto el STØP como el CALL EXIT podrán aparecer después de cualquier enunciado.

### 5.3 Los enunciados de entrada y salida

Estos, como su nombre lo indica, sirven para introducir y sacar información de la computadora.

#### 5.3.1 El enunciado READ

Este enunciado tiene la forma READ (I, N) LISTA I y N son enteros sin signo y LISTA represente una lista de nombres de variables para las cuales se leerán valores. I designa el tipo de periférico de entrada que se utilice (lectora de tarjetas, consola, etc.). N es el número de un enunciado FORMAT asociado al READ.

Ejem. El enunciado READ (2, 101) J, B, H

Producirá la lectura de tres números: un entero y dos reales y se almacenarán en las localidades de la memoria de la computadora designadas con las variables J, B y H en su orden. Las comas que separan éstos nombres de variables en el READ son indispensables, 2 es la unidad de entrada y 101 un FORMAT.

#### 5.3.2 El enunciado WRITE

Este tiene la forma WRITE (I, N) LISTA I y N son enteros sin signo y LISTA representa una lista de variables para las cuales se impriman valores. I designa el tipo de periférico de salida que se utilice (impresora, cinta, etc.). N es el número de un enunciado FORMAT aso-

ciado al WRITE.

Ejem. El enunciado WRITE (3, 108) L, X, Y

Producirá que se impriman los valores de las variables L, X y Y que se encuentren en las localidades de memoria con esos nombres, en el formato especificado por el enunciado número 108 y por la unidad de salida número 3; las comas que separan éstos nombres de variables en el WRITE son indispensables.

### 5.4 Los enunciados de especificación

Este tipo de enunciados no inician por sí mismos los cálculos, no producen transferencia de control ni estimulan el flujo de información, pero proveen al compilador FORTRAN de los detalles esenciales para la traducción del programa fuente en FORTRAN al programa objeto en lenguaje de máquina ó para la conversión de datos a la entrada o la salida.

Si queremos introducir datos a la computadora lo podemos hacer mediante un enunciado que esté dentro del programa, como  $A = 3.1416$ , esto es lo que podríamos llamar inicializar una variable; y el programa se compilaría cada vez que quisieramos darle un valor diferente a A, lo cual resulta muy costoso, ya que las compilaciones son laboriosas. Para evitar esto se usa el enunciado READ y los valores que se le den a A podrán estar en tarjetas de datos, los cuales son independientes del progra-

na fuente.

#### 5.4.1 El enunciado FORMAT

Este tiene la forma: `N FORMAT ( , , , ... )` en la cual `N` es el número del enunciado `FORMAT` y corresponde al `N` de los enunciados `READ` y `WRITE`. Los espacios entre las comas están disponibles para las especificaciones del tipo que se describen más adelante, siendo el número de espacios uno o más, de acuerdo a las necesidades del programador.

##### 5.4.1.1 La especificación `I:W`

Aquí `I` indica un valor entero y `W` es un entero que indica el número de columnas o ancho de campo, que ocupa ese valor en la tarjeta de entrada o en el papel de impresión. El número `w` deberá incluir un lugar para el signo de ese valor, siendo `+` opcional.

Ejem. Valor de los datos

de entrada o salida: 1130 +1620 -370 0 +14

Especificación:     14     15     14   11   13

##### 5.4.1.2 La especificación `F:Fw,d`

Aquí `F` indica un valor real, `w` indica el número de columnas que ocupará el valor en la tarjeta de entrada o en el papel de impresión; `d` indica el número de cifras que se encontrarán des-

pues del punto decimal. `w` deberá incluir un lugar para el signo y otro para el punto decimal.

Ejem. Valor de los datos de

entrada ó salida: 32787 -0.07 1130 +3.70

Especificación: F6.3 F5.3 F5.0 F5.2

##### 5.4.1.3 La especificación `E:Ew,d`

Aquí `E` indica un valor real en forma exponencial y `w` indica la anchura de campo para ese valor y debe de incluir el signo, si lo hay, el punto decimal, el lugar para la letra `E`, un lugar para el signo del exponente, si es negativo, y dos lugares para el exponente; `d` indica el número de dígitos a la derecha del punto decimal.

Ejem. Valor de los datos

de entrada o salida: .1403E04 -.7E-02 .1442E+04

Especificación: E8.4 E7.1 E9.4

Es conveniente que cuando deseemos sacar

información de la computadora, tomemos en cuenta

para el ancho del campo lo siguiente:

- 1.- El signo, aún cuando el `+` generalmente no se imprime.
- 2.- El punto decimal para las especificaciones `F` y `E`.
- 3.- Por lo menos un dígito a la izquierda

del punto decimal, puesto que muchas máquinas imprimirán allí un cero si otro dígito no ocurre.

- 4.- Suficientes lugares para todos los dígitos significativos deseados, debido a que para los dígitos que no se les deja espacio se truncan o redondean.
- 5.- Cuatro lugares para el exponente de la especificación E.
- 6.- El primer lugar se deja en blanco para el control de carro.

#### 5.4.2 El enunciado END

Este se lee simplemente END e informa al compilador que el programa fuente ha terminado y debe ser el último enunciado de cualquier programa FORTRAN.

### 6.- Arreglos

Frecuentemente tratamos con un grupo de variables que forman o pertenecen a una clase o colección. Cuando las variables forman un conjunto ordenado, pueden relacionarse unas con otras por la notación de subíndices; entonces designamos esa colección como arreglo y las variables que pertenecen a ésta serie son los elementos del arreglo. A veces se emplea como sinónimo de

arreglo el nombre de matriz y, en consecuencia, hablamos de elementos de la matriz.

#### 6.1 Variables con subíndices

Un conjunto de números que puede arreglarse en un renglón o columna se considera como un arreglo lineal o unidimensional, y éste serie puede llamarse vector. Identificamos los elementos de un vector renglón o columna por un sólo subíndice.

Ejem. La columna de números del vector llamado A, consiste de los elementos A<sub>1</sub> hasta A<sub>N</sub> inclusive y se representa como sigue:

Notación acostumbrada	Notación FORTRAN
A <sub>1</sub>	A (1)
A <sub>2</sub>	A (2)
A <sub>3</sub>	A (3)
A <sub>l</sub>	A (l)
A <sub>N</sub>	A (N)

Cada una de estas A(l), en donde l varía de 1 a N, son el nombre de una variable, el conjunto de todas ellas es lo que llamamos arreglo.

Si se usan dos subíndices para identificar los elementos de un arreglo se considera éste como un arreglo bidimensional. Los cuadros de un tablero de ajedrez, pueden considerarse como un arreglo bidimensional. Y si llamamos a cualquiera de los cuadros

dros con la variable CTAJ tendremos 64 variables; pero como el tablero tiene 8 renglones y 8 columnas, podemos referirnos al cuadro que se encuentre en el renglón 3 y la columna 5 con la variable CTAJ-(3,5).

Dependiendo del tipo de computadora será el número de subíndices que podremos asignarle a un arreglo; en IBM - 1130 sólo se admiten arreglos con un máximo de tres subíndices.

Las variables que se utilicen para designar arreglos deberán observar las reglas que se dieron anteriormente al hablar de variables enteras y reales considerando que para los cinco caracteres alfanuméricos son independientes de los índices que se encuentran entre paréntesis.

### 6.1.1 Reglas para los subíndices.

Regla 1 Un subíndice debe ser un entero, puede ser constante, variable ó una de las expresiones aritméticas siguientes:

$$A + V + b \quad A * V - b$$

en donde v es una variable entera y a y b son constantes enteras sin signo.

Ejem. Algunos subíndices pueden ser:

$$1972 \quad 10 * KONT \quad 2 * I \quad J$$

$$1976 * N - 8 \quad 2 * I - 4 \quad 2 * I + 3$$

No se pueden usar como subíndices:

$$1 * I \quad -1 \quad 2 - 10 * CONT \quad -1932 \quad -KIL0$$

Regla 2 Un subíndice sólo debe tomar valores positivos.

Regla 3 Un subíndice en sí no debe ser una variable con subíndices. Así X{(2)} no es permitido.

Regla 4 Un símbolo que representa un arreglo, una variable con subíndice, no debe usarse sin subíndices para representar otra variable diferente en el mismo programa. Esto es A(I) y A no deben referirse a variables diferentes. Como siempre hay una excepción que por ahora no tocaremos.

Ejem. Los símbolos para variables reales con subíndices podrían incluir:

$$X(I) \quad SUM(X+2) \quad A(I, 2 * J + 1) \quad B(INT) \quad C(I, J)$$

Para variables enteras con subíndices podemos tener:

$$INT(M, N) \quad I(J) \quad 1CTA(J, 2 * I)$$

### 6.2 El enunciado DIMENSION

Siempre que en un programa utilicemos variables con subíndices deberemos poner como primer enunciado el DIMENSION, el cual indica al compilador qué tanto espacio de memoria se debe reservar para las variables con subíndices. Su forma es:

$$DIMENSION u, v, w, \dots$$

Donde u, v, w, ... son nombres de variables, cada una de las cuales va seguida por el máximo número de elementos en el

arreglo correspondiente. Deberán observarse las siguientes reglas:

Regla 1 Cada variable con subíndices se debe mencionar en un enunciado DIMENSION antes de su primer uso en el programa.

Regla 2 Los símbolos representados anteriormente por u, v, w, ... deben tener la forma:

nombre de variable (máximo número de elementos)

el número entre paréntesis debe ser una constante entera sin signo.

Ejemplo: DIMENSION A(20), B(4,8), CARR(5,3,4)

Esto indica que el compilador reservará 20 loca-

lidades para el arreglo A, sus veinte variables serán A(1), A(2), ..., A(20) al mismo tiempo se reservarán 32 (4x8) localidades para las variables B(1,1), B(1,2), B(1,3), ..., B(1,8), B(2,1), B(2,2), ..., B(2,8), B(3,1), B(3,2), ..., B(3,8), B(4,1), B(4,2), ..., B(4,8) y por último se reservarán 60 (5x3x4) localidades para las variables del arreglo CARR, con tres subíndices cada una.

Regla 3 El arreglo que se use en particular, dentro del programa podrá tener menos elementos que los especificados en la magnitud del enunciado DIMENSION, pero no más.

Regla 4

La variable tal como aparece en el enunciado DIMENSION debe tener exactamente el mismo número de subíndices que en cualquier otra parte del programa.

## 7.- SUBPROGRAMAS.

Los subprogramas, también llamados subrutinas, son programas que pueden ser puestos en uso por otros programas cuando sea necesario.

Las funciones de biblioteca & funciones del sistema constituyen una variedad de subprogramas.

## 7.1- FUNCIONES

Cuando el valor de una variable depende de una ó más variables ó constantes y además de una serie de cálculos, y dicha variable ha de calcularse repetidamente y en diferentes puntos de un programa, es posible definirla como una función. En otras palabras, además de las funciones con que cuenta la biblioteca del sistema, el usuario puede escribir sus propias funciones para uso específico de su programa.

Tomemos un ejemplo para visualizar lo anterior:

Supongamos que para un programa es especial, en el cual trabajamos con grados en lugar de radianes, deseamos calcular continuamente  $\text{SENO}(X)$ , sin el uso de funciones sería necesario transformar el argumento deseado de grados a radianes y después llamar a la función del sistema  $\text{SIN}(X)$ . A continuación presentamos una función que calculará  $\text{SENO}(X)$ , ( $X$  en grados):

```
FUNCTION SENO ( X )
  X = X * 3.1415927/180.
  SENO = SIN ( X )
  RETURN
END
```

que es llamada desde el programa como:

```
GRAD= SENO (GRADOS)
```

En base a éste ejemplo podemos generalizar el uso de la proposición FUNCTION.

- Debe ser codificada en forma independiente del programa que la usará, es decir, no debe aparecer "dentro" del programa.
- Debe empezar con la palabra FUNCTION

FUNCTION nombre (parámetro )

- A continuación se escribe el nombre con que será llamada.
- Después, entre paréntesis y separados por comas, aparecen los argumentos.

## 7.1.1 EJEMPLOS.-

```
FUNCTION RAIZ1 (A,B,C)
  RAIZ1= (-B + SQRT (B**2 - 4.*A*C)) / (2.*A)
  RETURN
END
FUNCTION RAIZ2 ( A, B, C )
  RAIZ2 = ( -B - SQRT (B**2.*A*C)) / (2.*A)
  RETURN
END
```

```
C      (C. SEGUNDO GRADO
      READ ( 2,100) A, B, C
100    FORMAT ( 3F10.5)
      X1 = RAIZ1 (A,B,C)
      X2 = RAIZ2 (A,B,C)
      WRITE (3,200) A,B,C, X1,X2
200    FORMAT ( 5 (,F10.5)
      CALL EXIT
      END
```

Este ejemplo es solamente para mostrar el uso de la proposición FUNCTION y no contempla algunas situaciones como raíces complejas, etc.

## 7.2 SUBRUTINAS

Como es fácil notar, la proposición FUNCTION nos "regresa" un sólo valor y lo hace a través de su nombre. En muchos casos es conveniente & necesario que se nos regrese más de un valor, para éstos casos usamos la proposición o enunciado:

SUBROUTINE.

Una subrutina es un subprograma que puede "recibir" cualquier número de parámetros ( desde cero hasta un número determinado por el tipo de compilador) y puede "regresar" diferentes valores calculados.

Veamos algunos ejemplos:

Supongamos que al imprimir resultados de un cierto programa tenemos que escribir algún título usando los primeros renglones de la hoja. En tal caso podemos hacer uso de una subrutina como sigue:

```

SUBROUTINE ENCA
WRITE (3,200)
200  FORMAT(/,IX, 'REPORTE SEMANAL' , /)
RETURN
END

```

Como vemos no hemos pasado ningún parámetro ó valor a la subrutina. Para que se ejecute éste se debe hacer uso de la proposición CALL, de la siguiente forma:

```
CALL ENCA
```

dentro del programa y en el lugar donde deseamos que ocurra la impresión.

Disputemos ahora un ejemplo muy simple para ejemplificar el uso de parámetros. Hagamos una subrutina que "reciba" como entrada dos números, los suma y el resultado lo "regrese" en otra variable. Sean A y B los números a sumar, y C la variable en donde se pondrá el resultado.

```

SUBROUTINE SUMA (A,B,C)
C = A + B
RETURN
END

```

Es importante detenerse a ver el significado de los parámetros para las subrutinas:

La subrutina anterior SUMA puede ser llamada de diversas formas:

```

CALL SUMA (AA, BB, CC)
CALL SUMA (6, 7, X)
etc.

```

Como vemos, las variables A, B y C que aparecen en la subrutina son variables

locales ó dormidas y solo tienen sentido dentro de la subrutina. Veamos lo anterior:

Supongase el siguiente programa:

```

X1= 3.
X2= 4.
CALL SUMA ( X1,X2,X3)
SUM= X3
WRITE (3,200) X1,X2,X3, SUM
200  FORMAT(4 F10.5)
CALL EXIT
END

```

Se propone como ejercicio al lector que haga los veces de la máquina y escriba lo que ésta imprime.

La máquina imprimirá:

3.0 4.0 7.0 7.0

Una de las facilidades más útiles en subrutinas es la de arreglos como parámetros, ej:

```

SUBROUTINE MAXIM (A, MAX)
DIMENSION A ( 10)
-----
-----
-----
-----
RETURN
END

```

Supóngase que ésta subrutina encuentra el elemento del arreglo A (10) con mayor valor y lo regresa a través de la variable MAX. Es importante notar que si pasamos como parámetro uno ó más arreglos hay que dimensionarlos otra vez dentro de la subrutina, lo cual se puede hacer de al menos dos formas: 1) poniendo la dimensión que aparece en el programa que lo llama;

2) Poniéndole dimensión ) (una)

Ejemplo:



```
DIMENSION A (10), B (20)
```

```
CALL PRDEN (A)
```

```
CALL MAXIM (B)
```

```
CALL MAXIM (A)
```

```
CALL EXIT
```

```
END
```

Caso 1:

```
SUBROUTINE PRDEN (X)
```

```
DIMENSION X (10)
```

```
RETURN
```

```
END
```

Caso 2:

```
SUBROUTINE MAXIM (Y)
```

```
DIMENSION Y (1)
```

```
RETURN
```

```
END
```

### 7.2.1 COMMON.-

Como es posible visualizar en los párrafos anteriores, las variables usadas en las subrutinas, o mejor dicho, dentro de las subrutinas, son totalmente independientes a las variables usadas en el programa principal. Muchas veces es conveniente que tanto las subrutinas como el programa que las llama tengan variables en COMMON. Para lograr esto existe la declaración

```
COMMON
```

La forma general de esta proposición es:

```
COMMON lista de variables
```

donde "lista de variables" es un conjunto de variables y/o arreglos separados por comas a las cuales queremos adjudicarles la propiedad anterior, es decir, sean comunes a varios subprogramas.

Ej.

```
COMMON A, B, X (10), AB (30)
```

Esta declaración debe aparecer al principio de cualquier programa o subrutina en que se desea usar. Veamos un ejemplo:

```
C SUMA DE DOS NUMEROS
```

```
COMMON A, B, C
```

```
A = 3
```

```
B = 7
```

```
CALL SUMA
```

```
Z = C
```

```
WRITE (3,200) A, B, C, Z
```

```
200 FORMAT( A F10.5 )
```

```
CALL EXIT
```

```
END
```

```
SUBROUTINE SUMA
```

```
COMMON A, B, C
```

```
C = A + B
```

```
RETURN
```

```
END
```

Este programa debe imprimir :

```
3.0  7.0  10.0  10.0
```

Una propiedad importante del COMMON es que si un arreglo es especificado en COMMON que es automáticamente dimensionado; es decir, no hay que especificar dicho arreglo a través de la declaración DIMENSION .

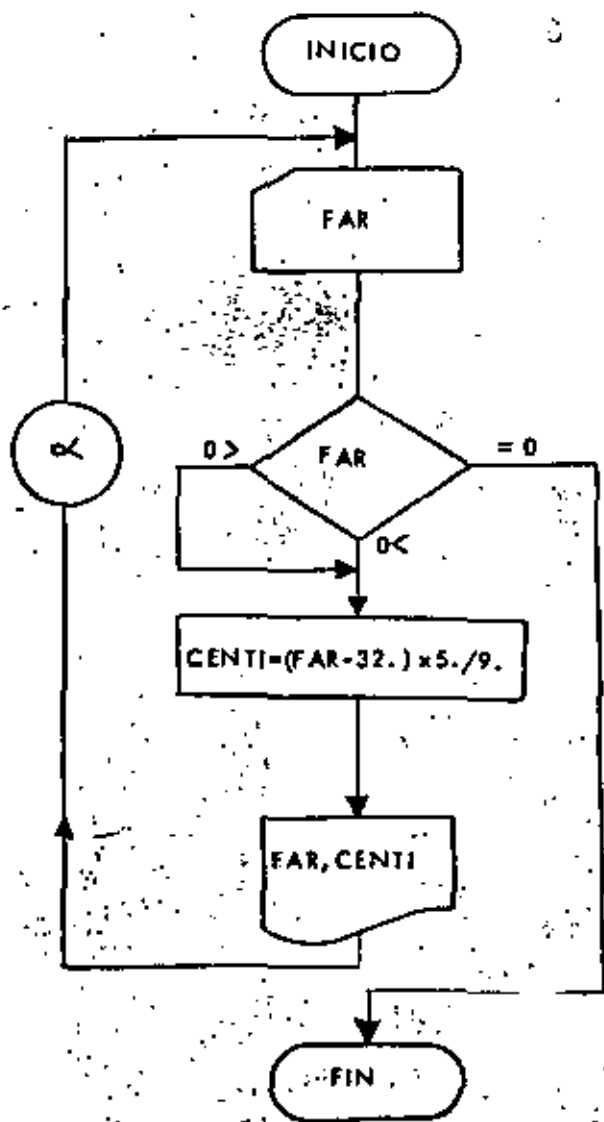
En las siguientes páginas se muestran veintidós programas, que incluyen sus diagramas de flujo, codificaciones, datos y resultados; el objeto es que el lector pueda complementar la parte teórica con la práctica, amén de que debe hacer los propios y procesarlos en una computadora a su alcance.

#### REFERENCIAS BIBLIOGRAFICAS:

- 1.- J.K. Hughes : Programación del Sistema (BM-1130)  
Limusa-Wiley-1969
- 2.- D.D. McCracken : Fortran IV  
Limusa-Wiley-1964
- 3.- E.I. Organick : Fortran IV  
Fondo Educativo Interamericano, S. A.  
1966
- 4.- Francis Scheld : Introducción a la Ciencia de  
las Computadoras.  
Serie de Compendios Schaum.  
McGraw-Hill-1970
- 5.- W. Schick y Ch. J. Merz, Jr. : Fortran para Ingeniería.  
McGraw-Hill - 1972
- 6.- R.E. Smith y D.E. Johnson : Fortran, Texto Programado.  
Limusa Wiley - 1971



"CONVERSION DE GRADOS FARENHEIT A GRADOS CENTIGRADOS"



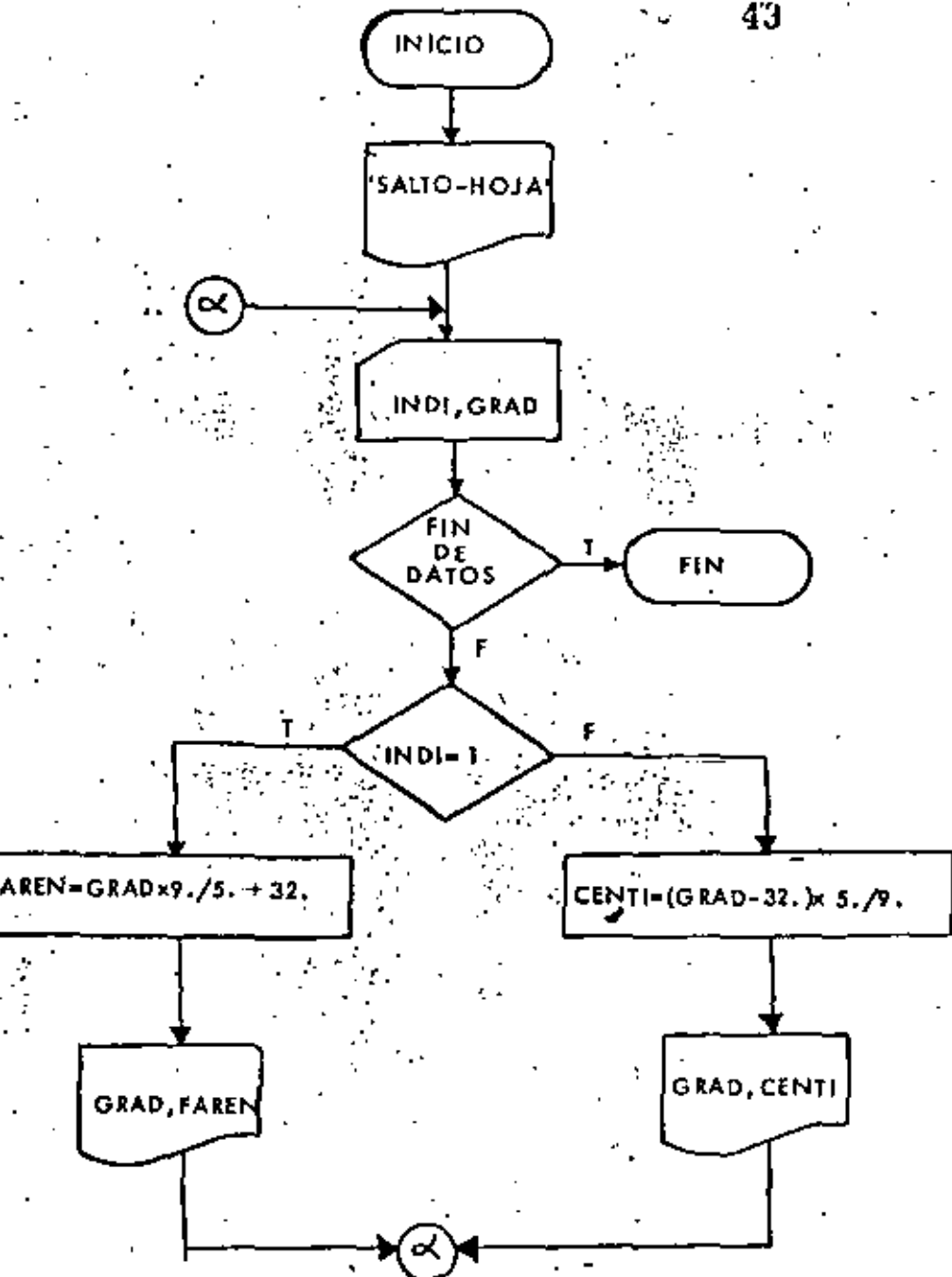
42

```

// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*10CS(CARD,1132 PRINTER)
C-----D 0 5-----
C      CONVERSION DE GHADOS FARENHEIT A
C      GRADOS CENTIGRADOS
100 FORMAT(F10.4)
101 FORMAT(F10.4,22M GRADOS FARENHEIT SON ,F10.4,20M GRADOS CENTIGRADO
15.1
LEE*2
IMP*3
200 READILEE,100IFAR
C      FAR IGUAL CERO INDICA TERMINO DE DATOS.
IF(FAR)210,220,210
210  CENTI=(FAR-32.)*5./9.
      WRITE(IMP,10)FAR,CENTI
      GO TO 200
220 CALL EXIT
END
// XEQ
126.0000
126.
-14.
18.26
0.0
/0
  
```

RESULTADOS

126.0000 GRADOS FARENHEIT SON	52.2222 GRADOS CENTIGRADOS.
126.0000 GRADOS FARENHEIT SON	52.2222 GRADOS CENTIGRADOS.
-14.0000 GRADOS FARENHEIT SON	-26.1111 GRADOS CENTIGRADOS.
18.2600 GRADOS FARENHEIT SON	7.1556 GRADOS CENTIGRADOS.

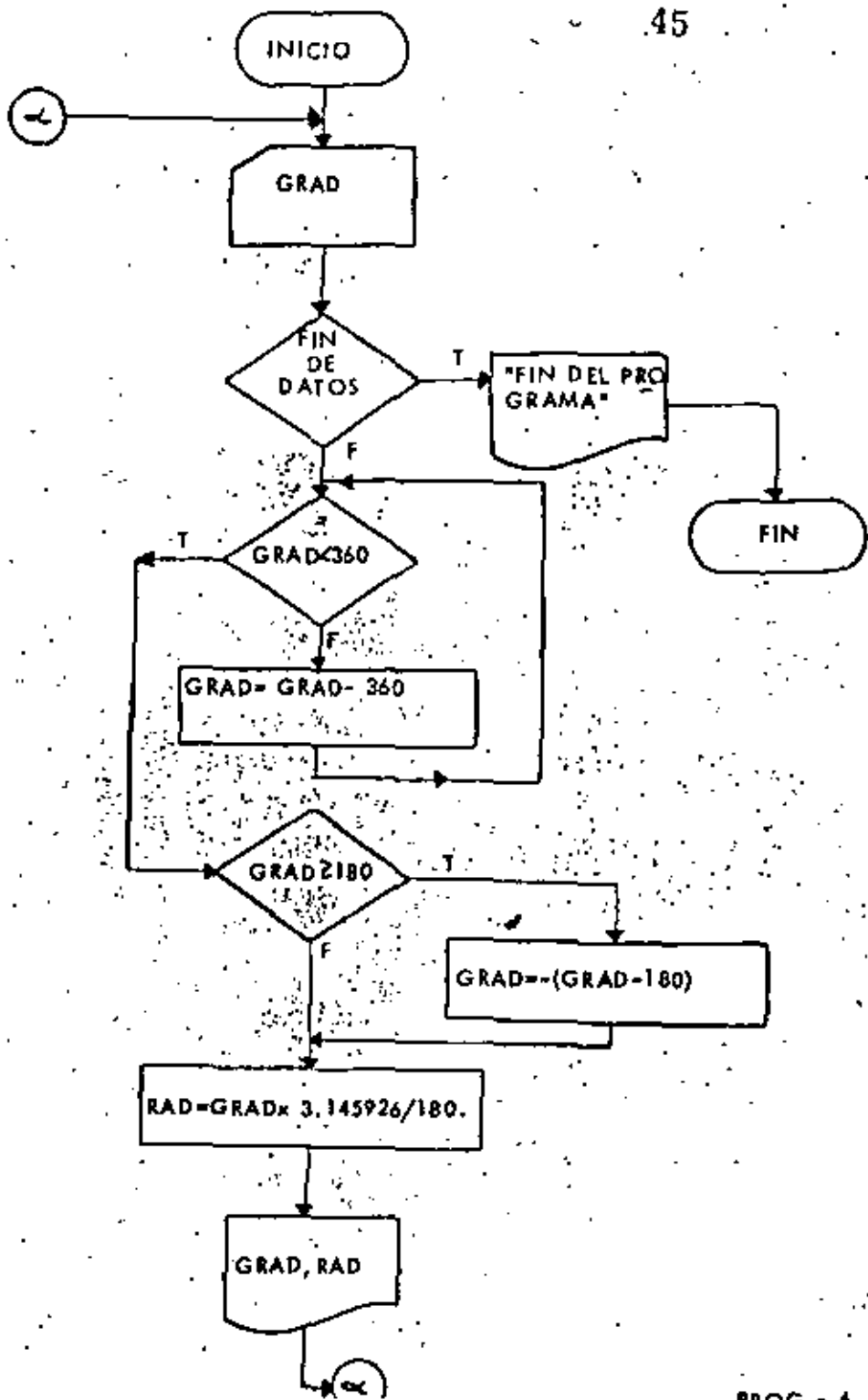


```

// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*10CS (CARD,1132 PRINTER)
-----T R E S-----
C      CONVERSION ENTRE GRADOS FARENHEIT
C      Y GRADOS CENTIGRADOS
.100 FORMAT(1H)
.101 FORMAT(11,F10.3)
.102 FORMAT(F10.2,15H FARENHEIT SON ,F11.3,13H CENTIGRADOS,)
.103 FORMAT(F10.2,17H CENTIGRADOS SON ,F9.3,11H FARENHEIT,)
LEE=2
IMP=3
WRITE(IMP,100)
200 HEAD(LEE,101,END=220)INDI,GRAD
IF(INDI.EQ.1)GO TO 210
C      INDI DIFERENTE DE 1 DAJO EN GRADOS FARENHEIT,
C      SE CONVIERTE A CENTIGRADOS.
CENTI=(GRAD-32.) * 5./9.
WRITE(IMP,102)GRAD,CENTI
GO TO 200
210 CONTINUE
C      EL DATO ES EN GRADO CENTIGRADO.
C      SE CONVIERTE A FARENHEIT.
FAREN=GRAD*9./5. + 32.
WRITE(IMP,103)GRAD,FAREN
GO TO 200
220 CALL EXIT
END
// XEQ
1 1200
D 11.40
1 0.
2 32.00
1 -16.
1 16.
/*
    
```

RESULTADOS

TEMPERATURA	UNIDAD	CONVERSION	RESULTADO
12:00	CENTIGRADOS	SUN	53.600 FARENHEIT
11:40	FARENHEIT	SUN	-11.800 CENTIGRADOS
0:00	CENTIGRADOS	SON	32.000 FARENHEIT
22:00	FARENHEIT	SUN	0.000 CENTIGRADOS
-10:00	CENTIGRADOS	SON	3.200 FARENHEIT
18:00	CENTIGRADOS	SON	6.400 FARENHEIT



```

*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*10CS(CARD,1132 PRINTER)
-----C U A T R O-----
C   CONVERSION DE GRADOS A RADIANTES
101 FORMAT(F8.3)
102 FORMAT(F9.3,12H GRADOS SON ,F6.3,10H RADIANTES.)
103 FORMAT(///,10X,16HFIN DEL PROGRAMA)
LEE=2
IMP=3
200 READ(L,101,END=230)GRAD
210 IF(GRAD,LT,360)GO TO 220
C   EL DATO ES IGUAL O SOBREPASA LOS 360 GRADOS SE AJUSTA.
GRAD=GRAD-360.
GO TO 210
220 CONTINUE
C   SE TRABAJA ENTRE +180 Y -180 GRADOS.
IF(GRAD,GE,180.)GRAD=-GRAD-180.
RAD=GRAD*3.1415926/180.
WRITE(IMP,102)GRAD,RAD
GO TO 200
230 WRITE(IMP,103)
CALL EXIT
END
    
```

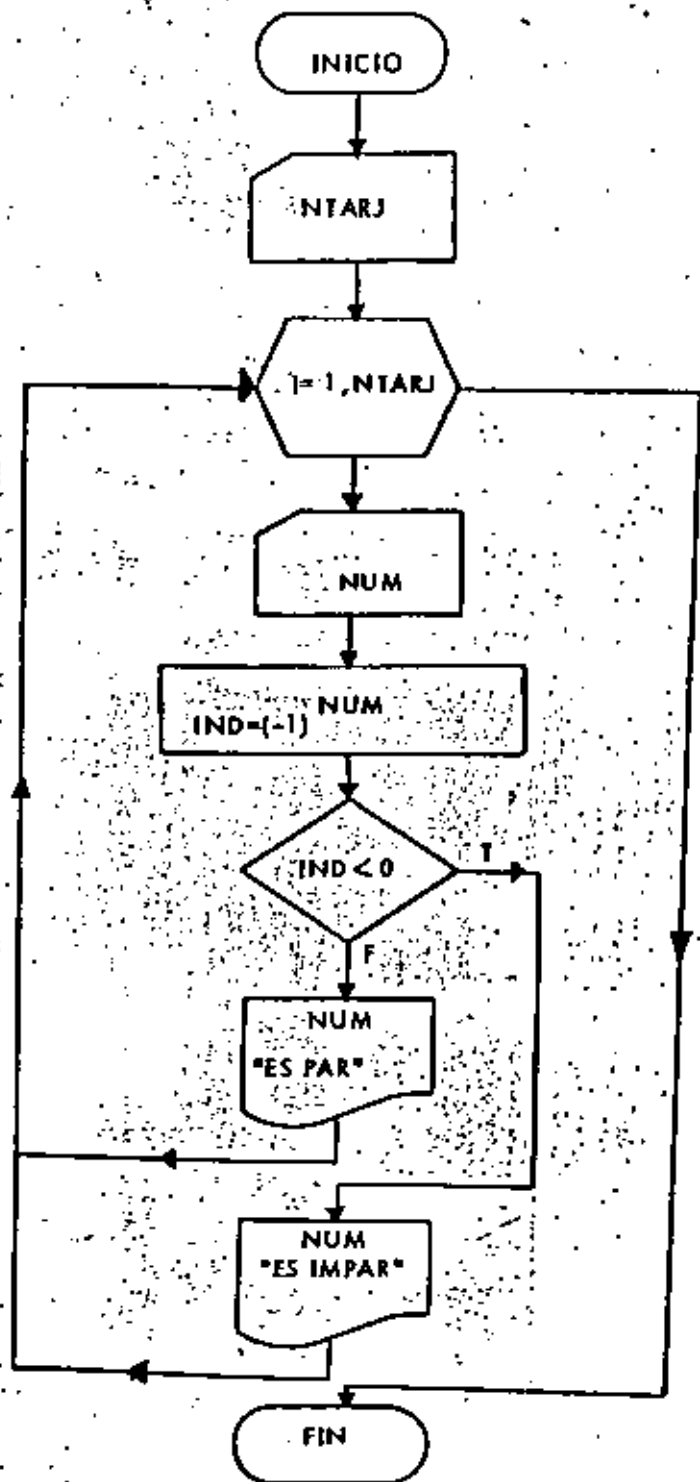
```

// XEQ
90000
-90.
3600.
-380.
0.0
185.27
132.4
-79.9
//
    
```

RESULTADOS

GRADOS	SON	RADIANTES
90.000	1.571	RADIANTES
-90.000	-1.571	RADIANTES
0.000	0.000	RADIANTES
180.000	3.142	RADIANTES
0.000	0.000	RADIANTES
185.270	3.241	RADIANTES
132.400	2.311	RADIANTES
-79.900	-1.395	RADIANTES

FIN DEL PROGRAMA



```

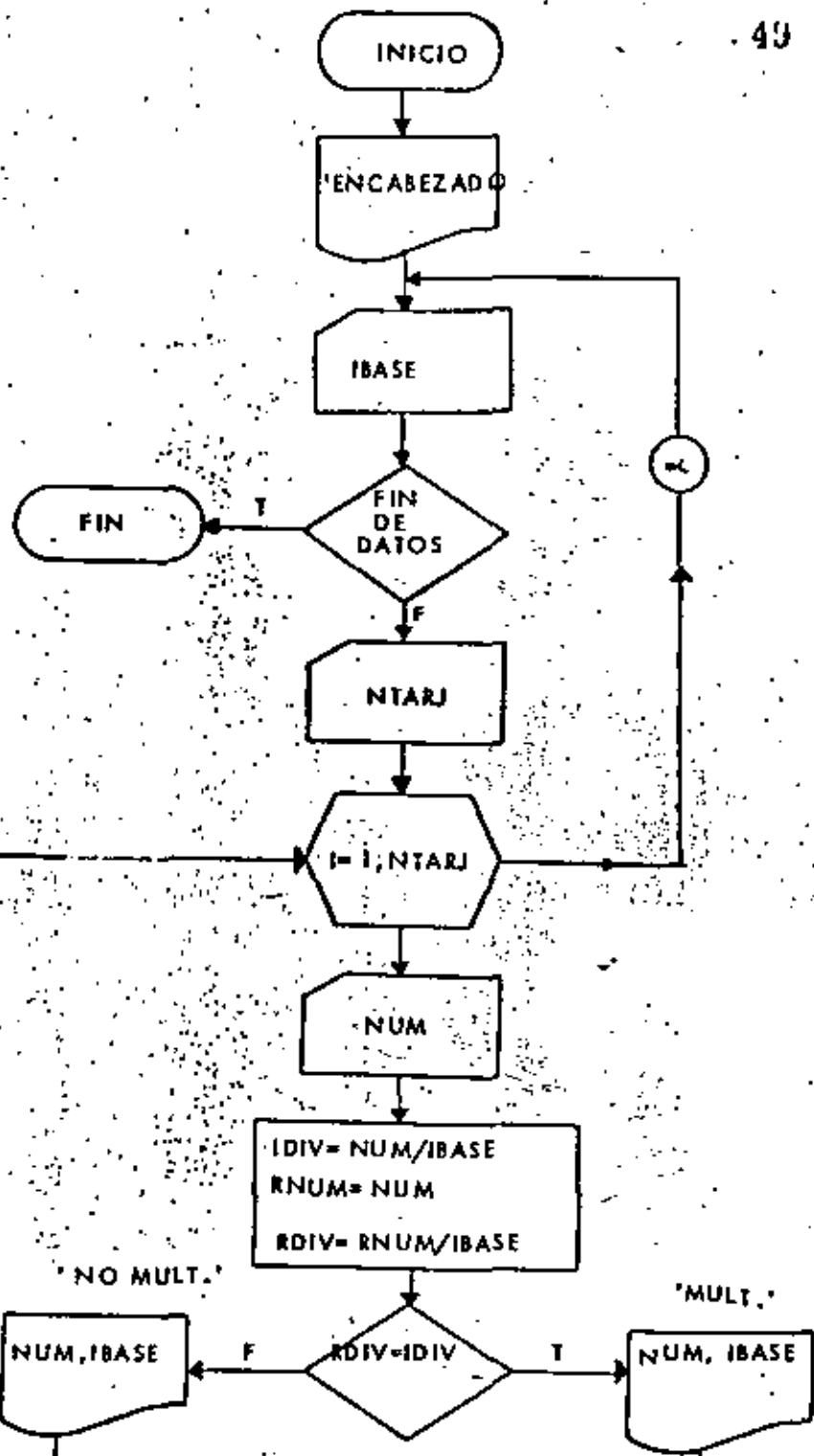
// JOB T
FOR
  1ST SOURCE PROGRAM
  *ONE WORD INTEGERS
  *10CS(CARD,1132 PRINTER)
  -----C I N C O-----
  C   DETERMINACION DE NUMEROS PARES
  C   E IMPARES
  100 FORMAT(13)
  101 FORMAT(14,8H ES PAR.)
  102 FORMAT(14,10H ES IMPAR.)
  LEE=2
  IMP=3
  READ(LEE,100)NTARJ
  C   NTARJ INDICA NO. DE TARJETAS CON DATOS.
  DO 202 I=1,NTARJ
    READ(LEE,100)NUM
    IND=(-1)*NUM
    IF(IND,LT,0)GO TO 200
  C   EL NUMERO ES PAR.
    WRITE(IMP,101)NUM
    GO TO 201
  200 CONTINUE
  C   EL NUMERO ES IMPAR.
    WRITE(IMP,102)NUM
  201 CONTINUE
  202 CONTINUE
  CALL EXIT
  END
  
```

```

// XEO
005
17
1
14
291
8
/*
  
```

RESULTADOS

17	ES IMPAR
1	ES IMPAR
14	ES PAR
291	ES IMPAR
8	ES PAR



```

// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*JOCSCARD,132 PRINTER)
-----S E I S-----
C DETERMINACION DE MULTIPLOS
C DE UN NUMERO
100 FORMAT(13)
101 FORMAT(34H)NUMERO=MULTIPLO DE=NO MULTIPLO DE)
102 FORMAT(2X,13,7X,13)
103 FORMAT(2X,13,21X,13)
LEE=2
IMP=3
WRITE(IMP,10)
200 READ(LEE,100,END=240)IBASE
READ(LEE,100)NTARJ
DO 230 I=1,NTARJ
READ(LEE,100)NUM
IDIV=NUM/IBASE
RNUM=NUM
RDIV=RNUM/IBASE
IF(RDIV.EQ.IDIV)GO TO 210
C NUM NO ES MULTIPLO DE IBASE.
WRITE(IMP,103)NUM,IBASE
GO TO 220
210 CONTINUE
C NUM SI ES MULTIPLO DE IBASE.
WRITE(IMP,102)NUM,IBASE
220 CONTINUE
230 CONTINUE
GO TO 200
240 CALL EXIT
END

```

```

// KEY
002
005
17
001
14
291
8
003
002
9
11
005
001
009
/*

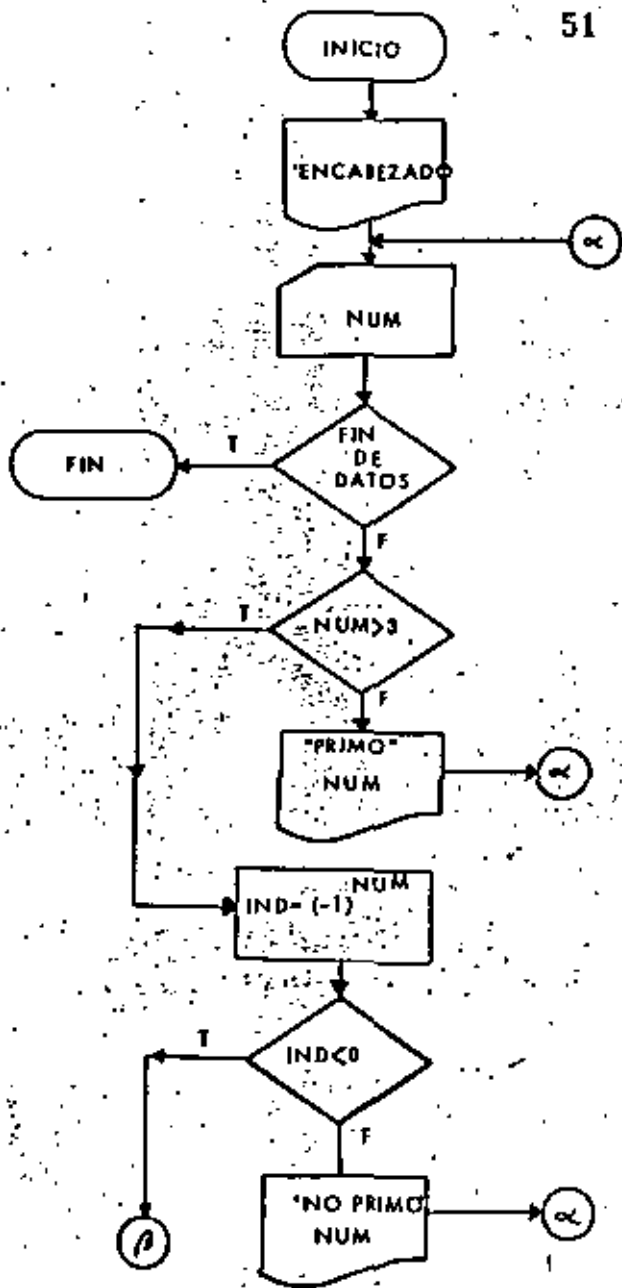
```

RESULTADOS

NUMERO	MULTIPLO DE	NO MULTIPLO DE
14	2	2
29	3	2
11	3	3
9	3	3

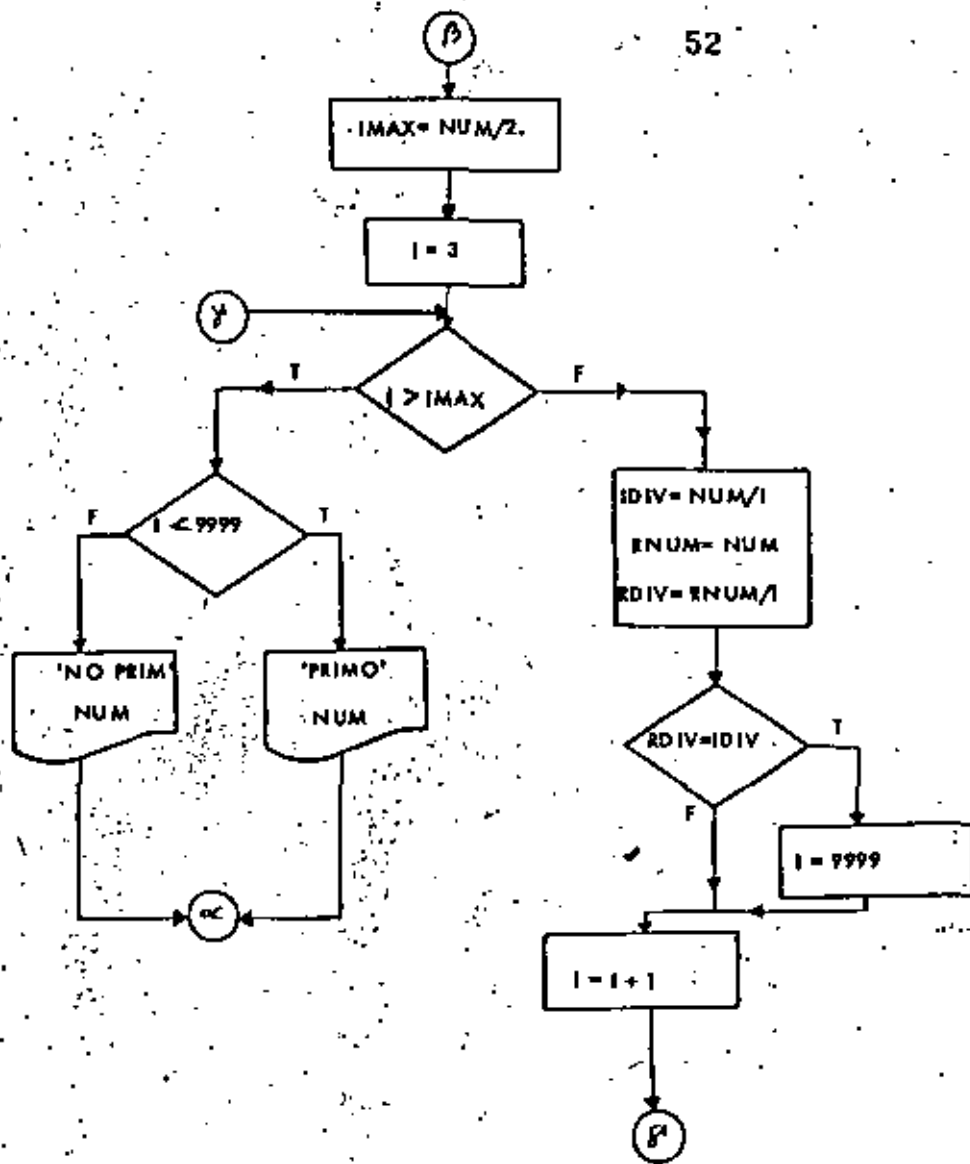


51



PROG - 7

52



PROG - 7

```

// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
C-----S I E I E-----
C   NUMEROS PRIMOS
100 FORMAT(13)
101 FORMAT(19H(1)PRIMOS  = NO 1)PRIMOS)
102 FORMAT(124.13)
103 FORMAT(1144.13)
   LEE=2
   IMP=3
   WRITE(IMP,101)
200 READ(LEE,100,END=290)NUM
   IF(NUM.GT.3)GO TO 210
C   NUM ES MENOR O IGUAL A 3(1)ODO NUMERO NATURAL MENOR O IGUAL A 3 ES
   WRITE(IMP,102)NUM
   GO TO 200
210 CONTINUE
C   NUM ES MAYOR QUE 3.
   IND=(-1)**NUM
   IF(IND.LT.0)GO TO 220
C   NUM ES PAR(1)ODO NUMERO PAR MAYOR QUE 3 NO ES PRIMO.
   WRITE(IMP,103)NUM
   GO TO 270
220 CONTINUE
C   NUM ES IMPAR(1)E INICIA PROCESO DE PRIMO O NO-PRIMO.
   IMAX=NUM/2
   I=3
230   IF(I.GT.IMAX)GO TO 240
C   SE OBTIENEN LAS DIVISIONES ENTERA Y REAL DE NUM/I
C   CON I DE 3 HASTA NUM/2.
   IDIV=NUM/I
   RNUM=NUM
   RDIV=RNUM/I
   IF(RDIV.EQ.IDIV)I=9999
C   I=9999 INDICA QUE NUM ES PRIMO.
   I=I+1
   GO TO 230
240 CONTINUE
   IF(I.LT.9999)GO TO 250
C   NUM NO ES PRIMO.
   WRITE(IMP,103)NUM
   GO TO 260
250 CONTINUE
C   NUM ES PRIMO.
   WRITE(IMP,102)NUM
260 CONTINUE
270 CONTINUE
280 CONTINUE
   GO TO 200
290 CALL EXIT
   END

```

```

// REG
1
2
3
4
5
6
7
8

```

53

PROG - 7

10  
11  
12  
13  
15  
17  
19  
21

54

RESULTADOS

PRIMOS	NO-PRIMOS
1	
2	4
3	6
5	10
7	12
11	15
13	21
17	
19	
21	

PROG - 7

P-7

```

// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*10CSICARD,1132 PRINTER)
-----D C H O-----
C SERIE DE FIBONACCI
100 FORMAT(13)
101 FORMAT(14,3X,15)
LEE=2
IMP=3
READ(LEE,100)NTERM
C NTERM REPRESENTA EL NUMERO DESEADO DE TERMINOS
NM1=1
NM2=0
WRITE(IMP,101);NM1
DO 200 J=2,NTERM
  NUEVO=NM1+NM2
  NM2=NM1
  NM1=NUEVO
C SE IMPRIME LA POSICION Y EL VALOR DEL TERMINO
WRITE(IMP,101);NUEVO
200 CONTINUE
CALL EXIT
END

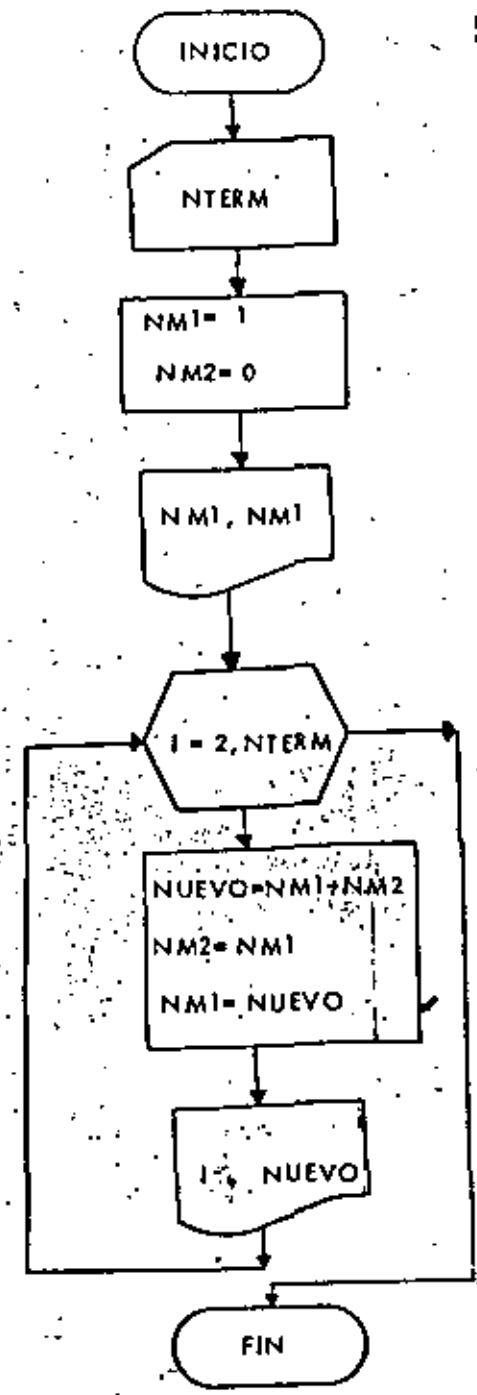
// XEQ
015
/*
    
```

55

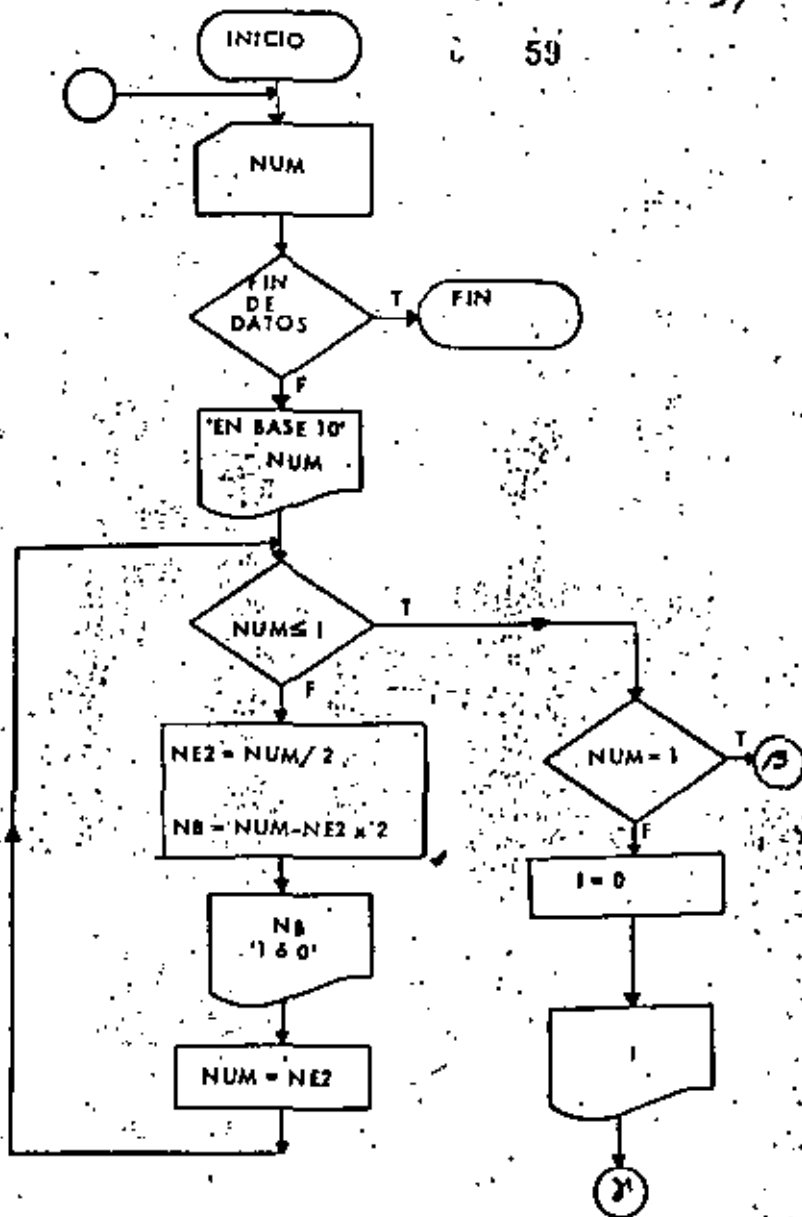
56

RESULTADOS

Posición	Valor
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21
9	34
10	55







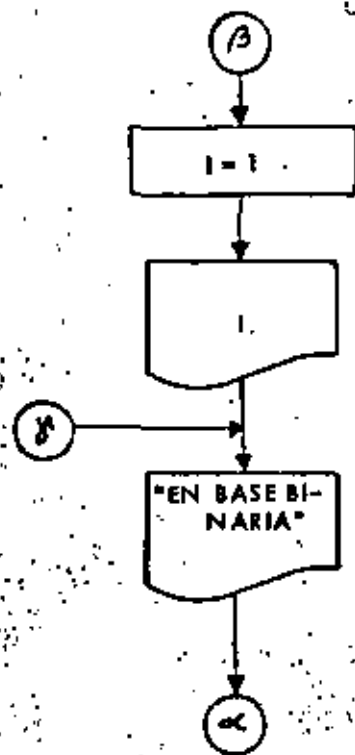
PROG - 10

```

// JOB 1
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IDCSICARD,1132 PRINTER1
-----D 1 E 2-----
C CAMBIO DE BASE 10 DECIMAL A BINARIA
100 FORMAT(15)
101 FORMAT(1X,15,10H EN BASE DECIMAL ES)
102 FORMAT(1A,11)
103 FORMAT(17H EN BASE BINARIA.)
LEE=2
IMP=3
200 READ(LEE,100,END=250)NUM
WRITE(IMP,101)NUM
210 IF(NUM,LE,1)GO TO 220
C NUM ES MAYOR QUE UNO, SE SIGUE DESCOMPONIENDO
NE2=NUM/2
NR=NUM-NE2*2
C NR ES UNO O CERO
WRITE(IMP,102)NR
NUM=NE2
GO TO 210
220 CONTINUE
IF(NUM,EQ,1)GO TO 230
C SE IMPRIME EL ULTIMO CERO EN LA REPRESENTACION BINARIA
I=0
WRITE(IMP,102)I
GO TO 240
230 CONTINUE
C SE IMPRIME EL ULTIMO 1 EN LA REPRESENTACION BINARIA
I=1
WRITE(IMP,102)I
240 CONTINUE
WRITE(IMP,103)
GO TO 200
250 CALL TRIT
END
// XEO
12
163
10
1-0001
011
-131
8
1
13
//
    
```

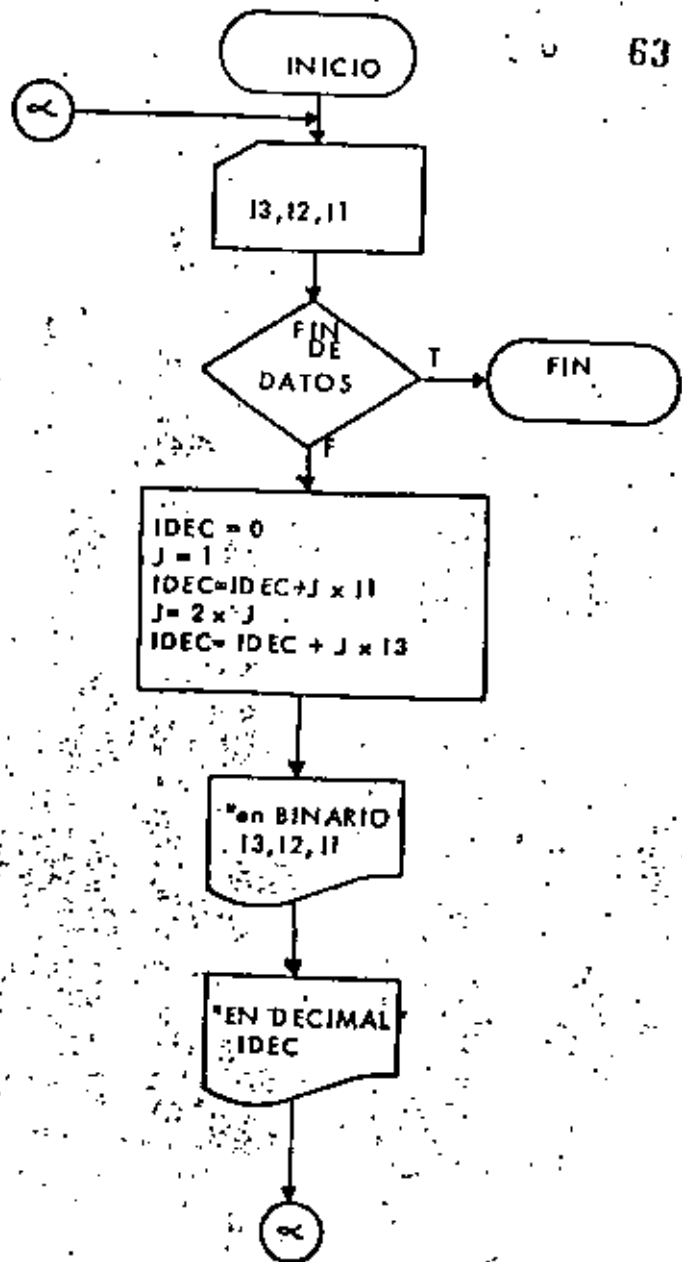
PROG - 10

61



RESULTADOS 62

12 EN BASE DECIMAL ES
0
EN BASE BINARIA,
163 EN BASE DECIMAL ES
1
0
0
EN BASE BINARIA,
10 EN BASE DECIMAL ES
0
0
EN BASE BINARIA,
1 EN BASE DECIMAL ES
1
EN BASE BINARIA,
13 EN BASE DECIMAL ES
1
0
EN BASE BINARIA,
131 EN BASE DECIMAL ES
0
EN BASE BINARIA,
0 EN BASE DECIMAL ES
0
EN BASE BINARIA,
1 EN BASE DECIMAL ES
1
EN BASE BINARIA,
13 EN BASE DECIMAL ES
1
EN BASE BINARIA,



```

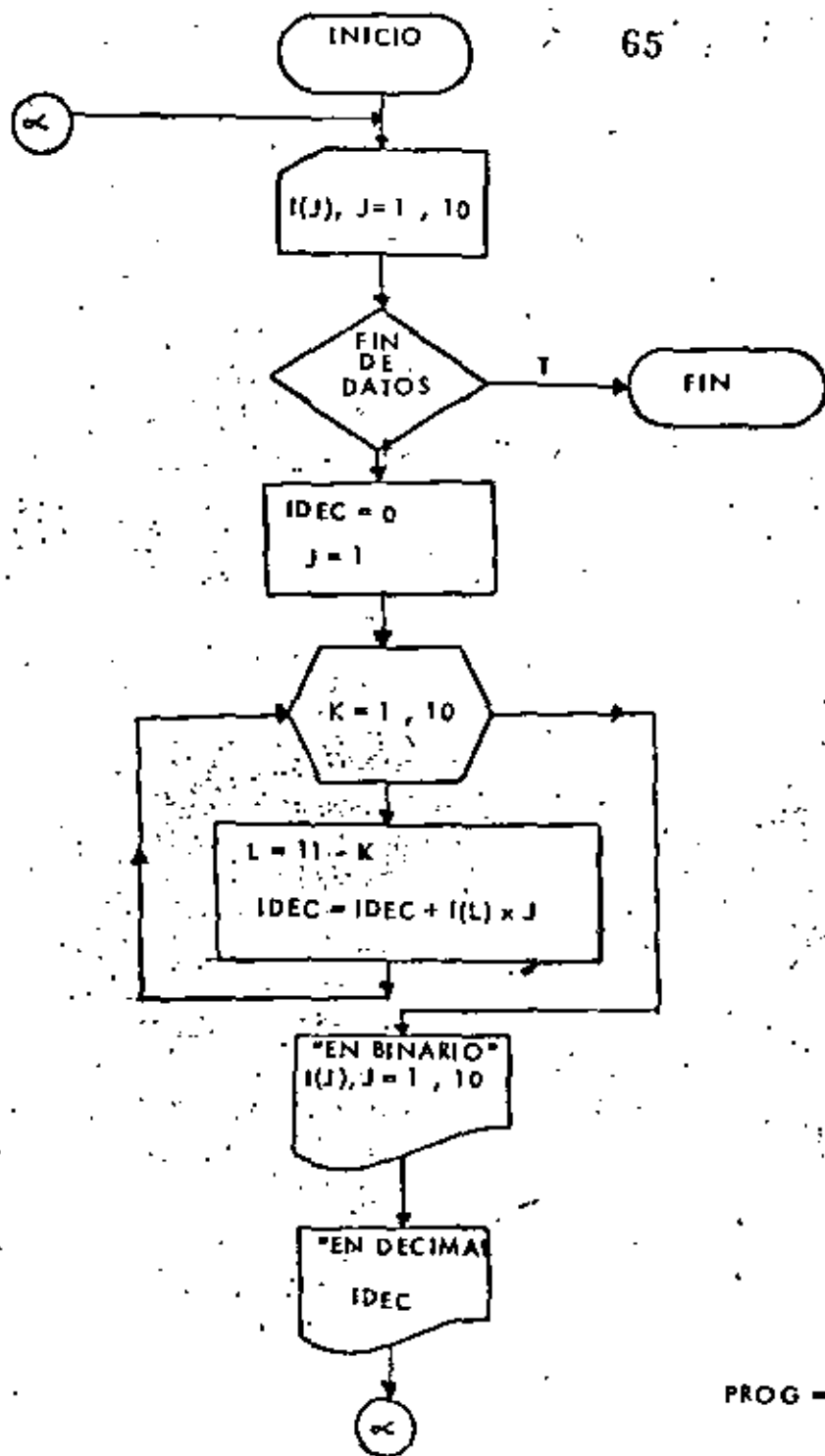
// SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(CARD,1132 PRINTER)
-----O N C E-----
C CAMBIO DE BASE I BINARIA A DECIMAL
100 FORMAT(3I1)
101 FORMAT(IX,3I1,19H EN BASE BINARIA ES)
102 FORMAT(IX,15,17H EN BASE DECIMAL.)
LEE=2
IMP=3
200 READ(2,100,END=210)13,12,11
IDEC=0
C J CONTIENE LAS POTENCIAS DE 2.
J=1
IDEC=IDEC+J*11
J=2*J
IDEC=IDEC+J*12
J=2*J
IDEC=IDEC+J*13
WRITE(IMP,101)13,12,11
C IDEC CONTIENE LA REPRESENTACION DECIMAL DEL NUMERO BINARIO.
WRITE(IMP,102)IDEC
GO TO 200
210 CALL EXIT
END
  
```

```

// XEQ
001
010
011
100
101
110
111
/*
  
```

RESULTADOS

001	EN BASE BINARIA ES
1	EN BASE DECIMAL ES
010	EN BASE BINARIA ES
2	EN BASE DECIMAL ES
011	EN BASE BINARIA ES
3	EN BASE DECIMAL ES
100	EN BASE BINARIA ES
4	EN BASE DECIMAL ES
101	EN BASE BINARIA ES
5	EN BASE DECIMAL ES
110	EN BASE BINARIA ES
6	EN BASE DECIMAL ES
111	EN BASE BINARIA ES
7	EN BASE DECIMAL ES



```

JOB T
FOR
- LIST SOURCE PROGRAM
- ONE WORD INTEGERS
- 10CS(CARD,1)32 PRINTER)
-----D O C E-----
C CAMBIO DE BASE 1 BINARIA A DECIMAL
C USANDO ARREGLOS
C DIMENSION I(10)
100 FORMAT(10I1)
101 FORMAT(1X,10I1,19H EN BASE BINARIA ES)
102 FORMAT(1X,15,17H EN BASE DECIMAL.)
LEE=2
IMP=3
200 READ(2,100+END+220)I
IDEC=0
J=1
DO 210 K=1,10
C SE ANALIZA EL VECTOR I DE DERECHA A IZQUIERDA
L=11-K
IDEC=IDEC+I(L)*J
C J CONTIENE LAS POTENCIAS DE 2
J=J*2
210 CONTINUE
WRITE(IMP,101)I(J),J,10)
WRITE(IMP,102)IDEC
GO TO 200
220 CALL EXIT
END
    
```

```

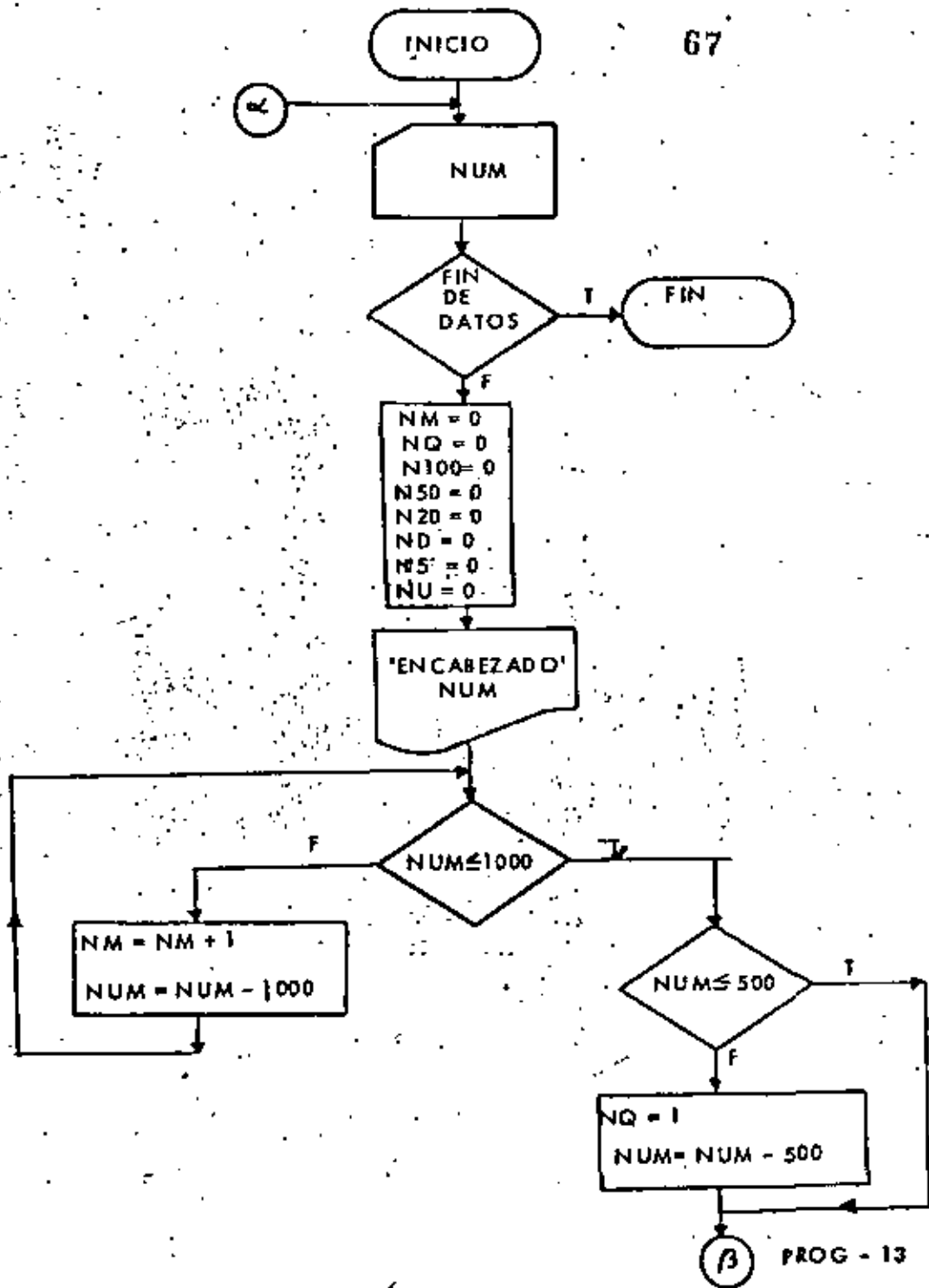
// XED
1
11
1000100010
1
1
1001001
00
    
```

RESULTADOS

0000000001 EN BASE BINARIA ES
1 EN BASE DECIMAL
0000000010 EN BASE BINARIA ES
2 EN BASE DECIMAL
1000100010 EN BASE BINARIA ES
546 EN BASE DECIMAL
1000000000 EN BASE BINARIA ES
512 EN BASE DECIMAL
1000000001 EN BASE BINARIA ES
513 EN BASE DECIMAL
0001001001 EN BASE BINARIA ES
73 EN BASE DECIMAL
0000000000 EN BASE BINARIA ES
0 EN BASE DECIMAL

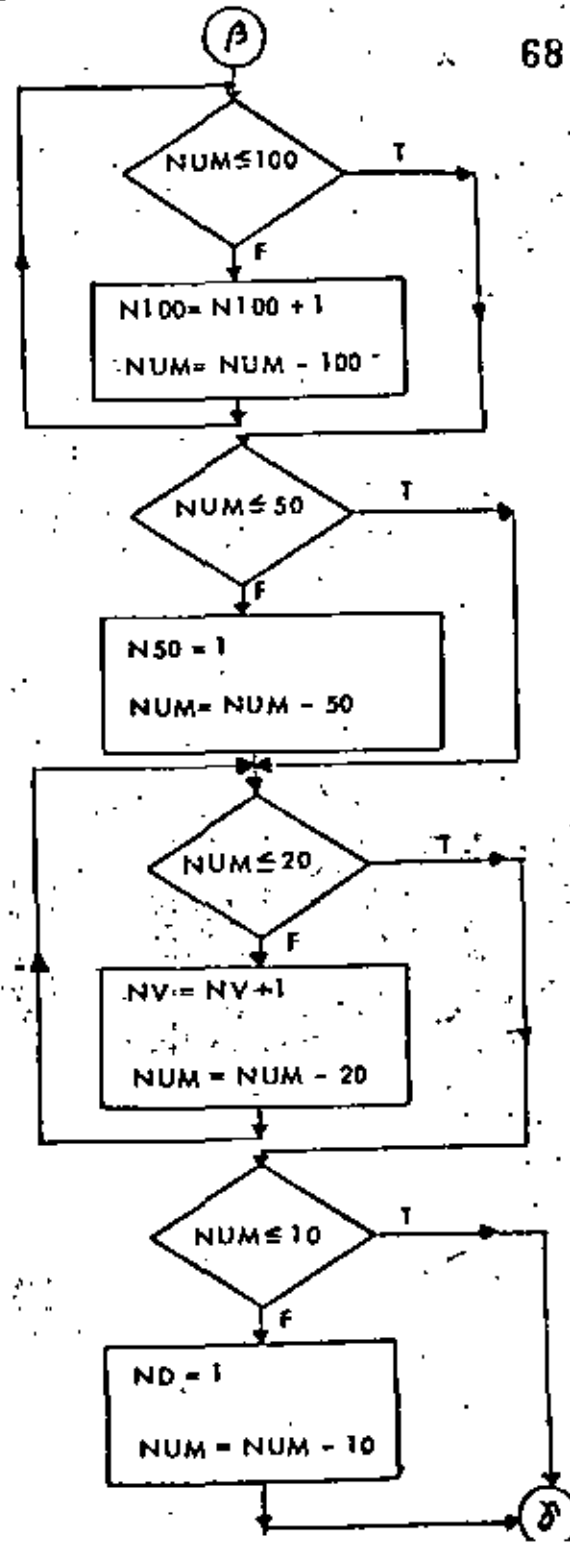


67



PROG - 13

68



PROG - 13

```
// JOB 1
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IDCS(CARD,1132 PRINTER)
```

71

```
C-----T R E C E-----
C   CALCULO DEL NUMERO DE BILLETES
100 FORMAT(14)
101 FORMAT(1,1)H EL NUMERO-,14,24H.00-PUEDA DESGLOSARSE EN)
102 FORMAT(15, 7H DE MIL)
103 FORMAT(15,14H DE QUINIENTOS)
104 FORMAT(15, 8H DE CIEN)
105 FORMAT(15,13H DE CINCUENTA)
106 FORMAT(15,10H DE VEINTE)
107 FDRMAT(15, 8H DE DIEZ)
108 FORMAT(15, 9H DE CINCO)
109 FORMAT(15, 7H DE UNO)
   LEE=2
   IMP=3
200 READ(LEE,100,END=330)NUM
   NM=0
   ND=0
   N100=0
   N50=0
   NV=0
   NU=0
   NS=0
   NU=0
   WRITE(IMP,101)NUM
210 IF(NUM.LE.1000)GO TO 220
C   NUM ES MAYOR QUE 1000
   NM=NM+1
   NUM=NUM-1000
   GO TO 210
220 CONTINUE
C   IF(NUM.LE.500)GO TO 230
   NUM ES MAYOR QUE 500 (MAXIMO UN BILLETE DE 500)
   NQ=1
   NUM=NUM-500
230 CONTINUE
C   IF(NUM.LE.100)GO TO 250
   NUM ES MAYOR QUE 100
   N100=N100+1
   NUM=NUM-100
   GO TO 240
250 CONTINUE
C   IF(NUM.LE.50)GO TO 260
   NUM ES MAYOR QUE 50 (MAXIMO UN BILLETE DE 50)
   NS=1
   NUM=NUM-50
260 CONTINUE
C   IF(NUM.LE.20)GO TO 280
   NUM ES MAYOR QUE 20
   NV=NV+1
   NUM=NUM-20
   GO TO 270
280 CONTINUE
C   IF(NUM.LE.10)GO TO 290
   NUM ES MAYOR QUE 10 (MAXIMO UN BILLETE DE 10)
   NN=1
   NUM=NUM-10
290 CONTINUE
   IF(NUM.LE.5)GO TO 300
```

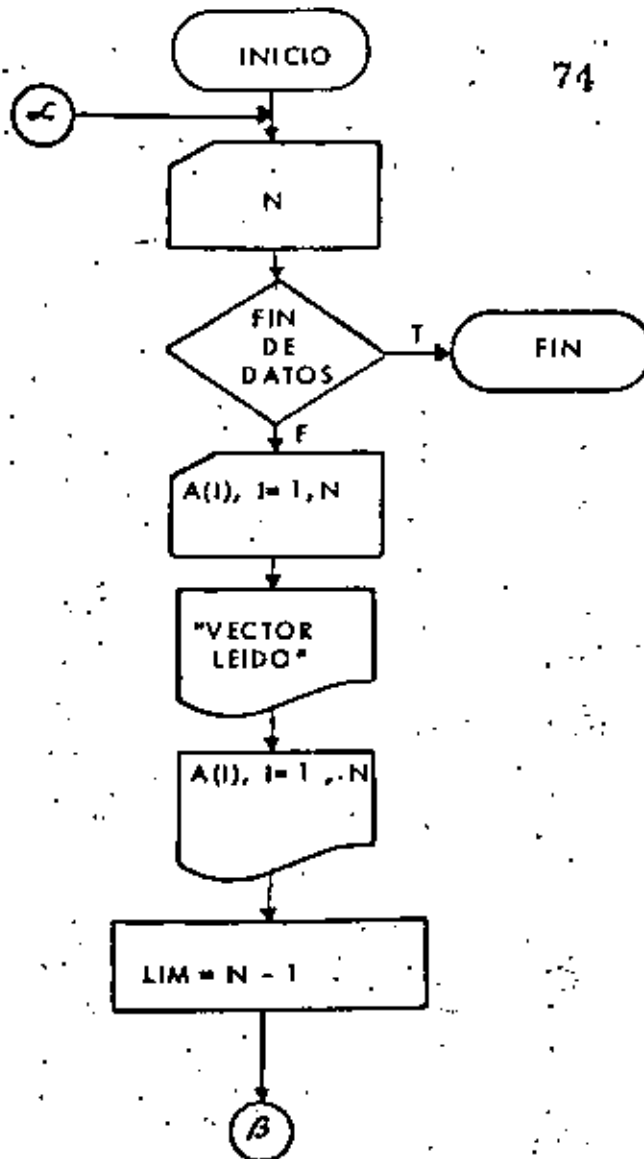
```
C   NUM ES MAYOR QUE 5 (MAXIMO UN BILLETE DE 5)
   NS=1
   NUM=NUM-5
300 CONTINUE
310 IF(NUM.LE.0)GO TO 320
C   NUM ES MAYOR QUE 1
   NU=NU+1
   NUM=NUM-1
   GO TO 310
320 CONTINUE
   WRITE(IMP,102)NM
   WRITE(IMP,103)NQ
   WRITE(IMP,104)N100
   WRITE(IMP,105)NS
   WRITE(IMP,106)NV
   WRITE(IMP,107)ND
   WRITE(IMP,108)NS
   WRITE(IMP,109)NU
   GO TO 200
330 CALL EXIT
   END

// XEQ
9000
1314
6893
1000
500
13
/*
```

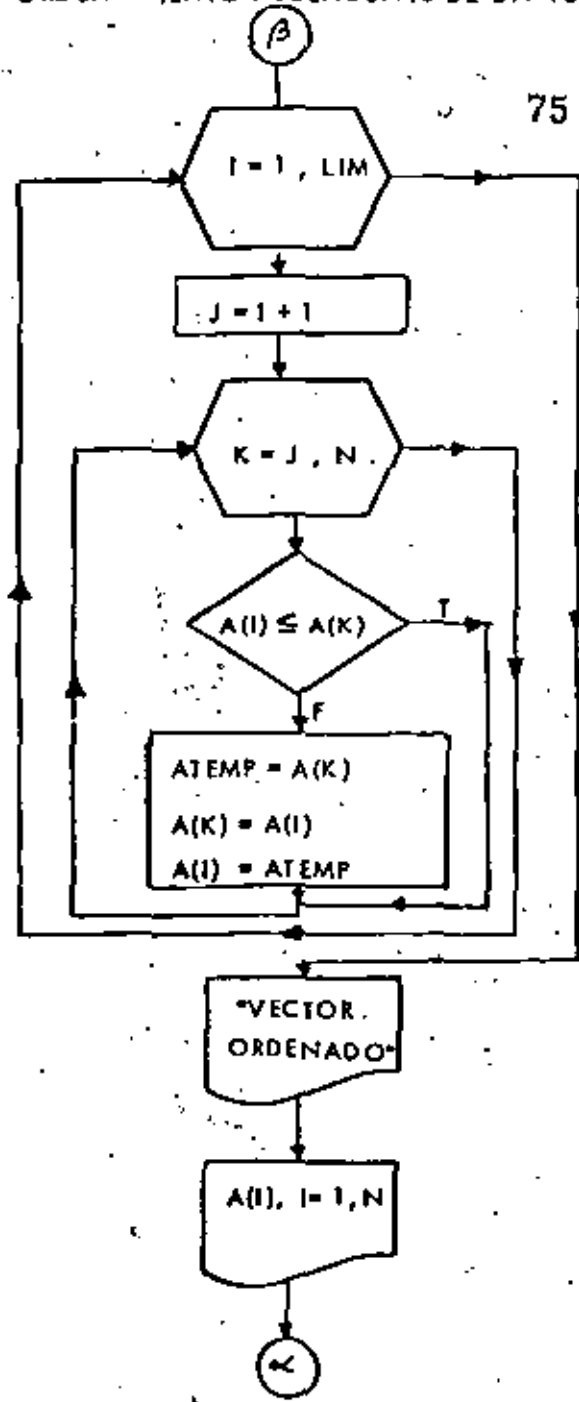
PROG.

PR

EL NÚMERO 9000.00 PUEDE DESGLOSARSE EN	
9	DE MIL
0	DE QUINIENTOS
0	DE CIENTO
0	DE CINCUENTA
0	DE VEINTE
0	DE DIEZ
0	DE CINCO
0	DE UNO
EL NÚMERO 1314.00 PUEDE DESGLOSARSE EN	
1	DE MIL
0	DE QUINIENTOS
3	DE CIENTO
0	DE CINCUENTA
0	DE VEINTE
0	DE DIEZ
4	DE CINCO
0	DE UNO
EL NÚMERO 8893.00 PUEDE DESGLOSARSE EN	
8	DE MIL
0	DE QUINIENTOS
8	DE CIENTO
0	DE CINCUENTA
0	DE VEINTE
0	DE DIEZ
3	DE CINCO
0	DE UNO
EL NÚMERO 1000.00 PUEDE DESGLOSARSE EN	
1	DE MIL
0	DE QUINIENTOS
0	DE CIENTO
0	DE CINCUENTA
0	DE VEINTE
0	DE DIEZ
0	DE CINCO
0	DE UNO
EL NÚMERO 500.00 PUEDE DESGLOSARSE EN	
0	DE MIL
5	DE QUINIENTOS
0	DE CIENTO
0	DE CINCUENTA
0	DE VEINTE
0	DE DIEZ
0	DE CINCO
0	DE UNO
EL NÚMERO 13.00 PUEDE DESGLOSARSE EN	
0	DE MIL
0	DE QUINIENTOS
0	DE CIENTO
0	DE CINCUENTA
0	DE VEINTE
0	DE DIEZ
0	DE CINCO
3	DE UNO



" ORDEN ENTO ASCENDENTE DE UN VECTOR "



76

```

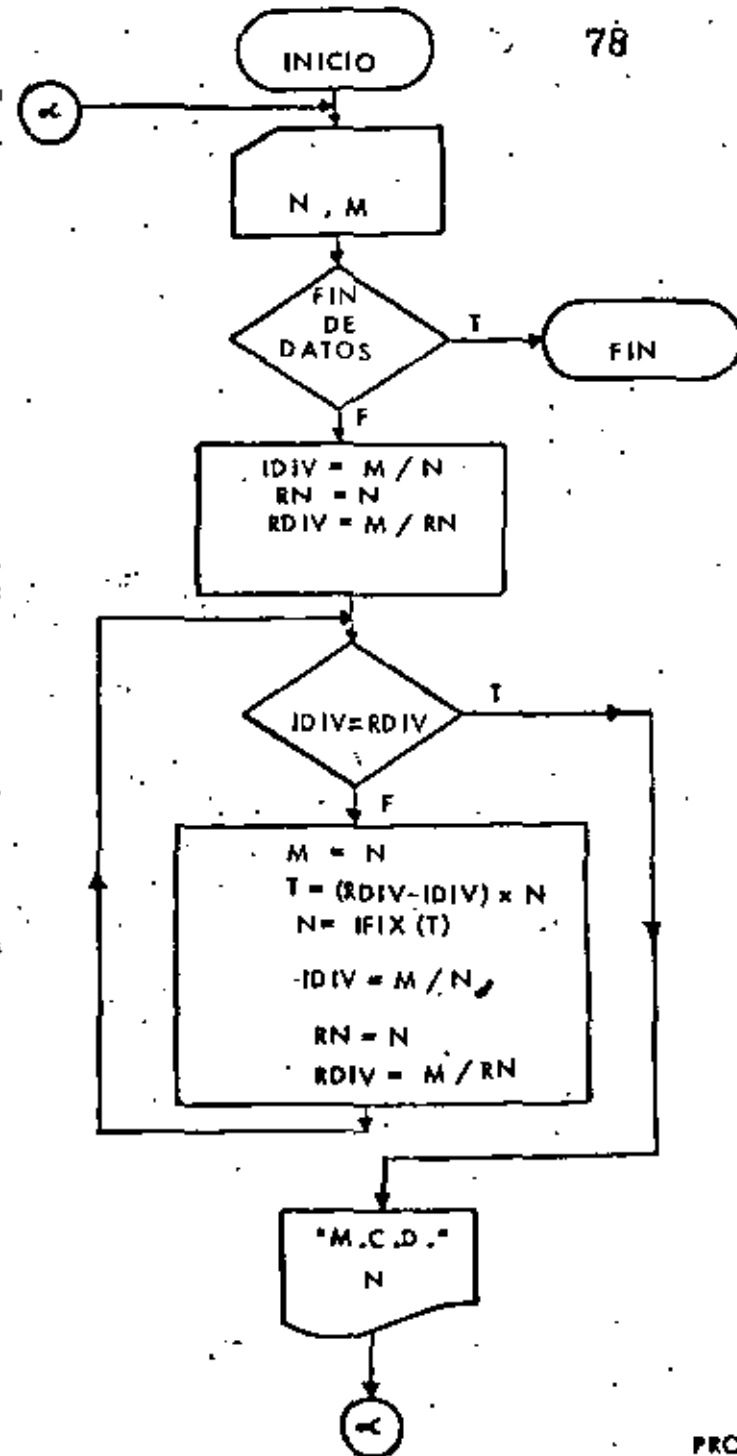
// JOB 1
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*10CS(CARD,11)2 PRINTER)
C-----C A T O R C E-----
C   URDENAMIENTO ASCENDENTE DE UN VECTOR
C   DIMENSION A(100)
100 FORMAT(I2)
101 FORMAT(BF10.0)
102 FORMAT(1)3H VECTOR LEIDO,/)
103 FORMAT(10(1X,F11.4))
104 FORMAT(16H VECTOR ORDENADO,/)
LEE=2
IMP=3
200 READ(LEE,100,END=240)N
C   N REPRESENTA EL NUMERO DE ELEMENTOS A ORDENAR
READILEE,101)A(I,1)=N)
WRITE(IMP,102)
WRITE(IMP,103)(A(I,1)=N)
LIM=N-1
DO 230 I=1,LIM
  J=I+1
C   SE ASUME QUE A(I) ES EL MENOR
DO 220 K=J,N
  IF(A(I).LE.A(K))GO TO 210
C   A(I) FUE MAYOR QUE A(K)
  ATEMP=A(K)
  A(K)=A(I)
  A(I)=ATEMP
210 CONTINUE
220 CONTINUE
C   AHORA SE TIENE EN A(I) EL MENOR
230 CONTINUE
WRITE(IMP,104)
WRITE(IMP,103)(A(I,1)=N)
GO TO 200
240 CALL EXIT
END

// REQ
04  -4.      1.      -3.      17.
03  0.      -267.   32.
04  -26.    -32.    11.     0.
/*
  
```

RESULTADOS

VECTOR LEIDO			
=4.0000	=1.0000	=3.0000	17.0000
VECTOR ORDENADO			
=4.0000	=3.0000	1.0000	17.0000
VECTOR ORDENADO			
0.0000	=287.0000	32.0000	
VECTOR LEIDO			
=287.0000	0.0000	32.0000	
VECTOR ORDENADO			
=28.0000	=32.0000	11.0000	0.0000
VECTOR LEIDO			
=32.0000	=28.0000	0.0000	11.0000

MAXIMO COMUN MULTIPLO ALGORITMO DE EUCLIDES \*



```

// JOB T
// FOR
%LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*1005ICARD,1132:PRINTER)
C-----B U I N C E-----
C   MAXIMO COMUN MULTIPLO
C   ALGORITMO DE EUCLIDES
100 FORMAT(2I3)
101 FORMAT(3M N=,13,3M M=,13)
102 FORMAT(1M N,C,D.=,13,/)
LEE=2
IMP=3
200 READ LEE,100,END=230 IN=N
C   SE CALCULA EL RESIDUO
IDIV=M/N
RN=M
RDIV=M/RN
210 IF (ID)V.EQ,RDIV)GO TO 220
M=N
T=(RDIV-IDIV)*M
N=FIX(T)
IDIV=M/N
RN=M
RDIV=M/RN
GO TO 210
220 CONTINUE
C   N REPRESENTA EL MAXIMO COMUN DIVISOR
WRITE(IMP,102)N
GO TO 200
230 CALL EXIT
END

```

```

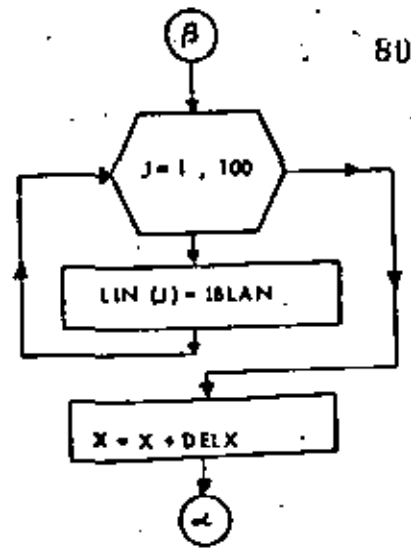
// IEO
5 7
3 6
8 16
11 98
16 24
E 12

```

RESULTADOS

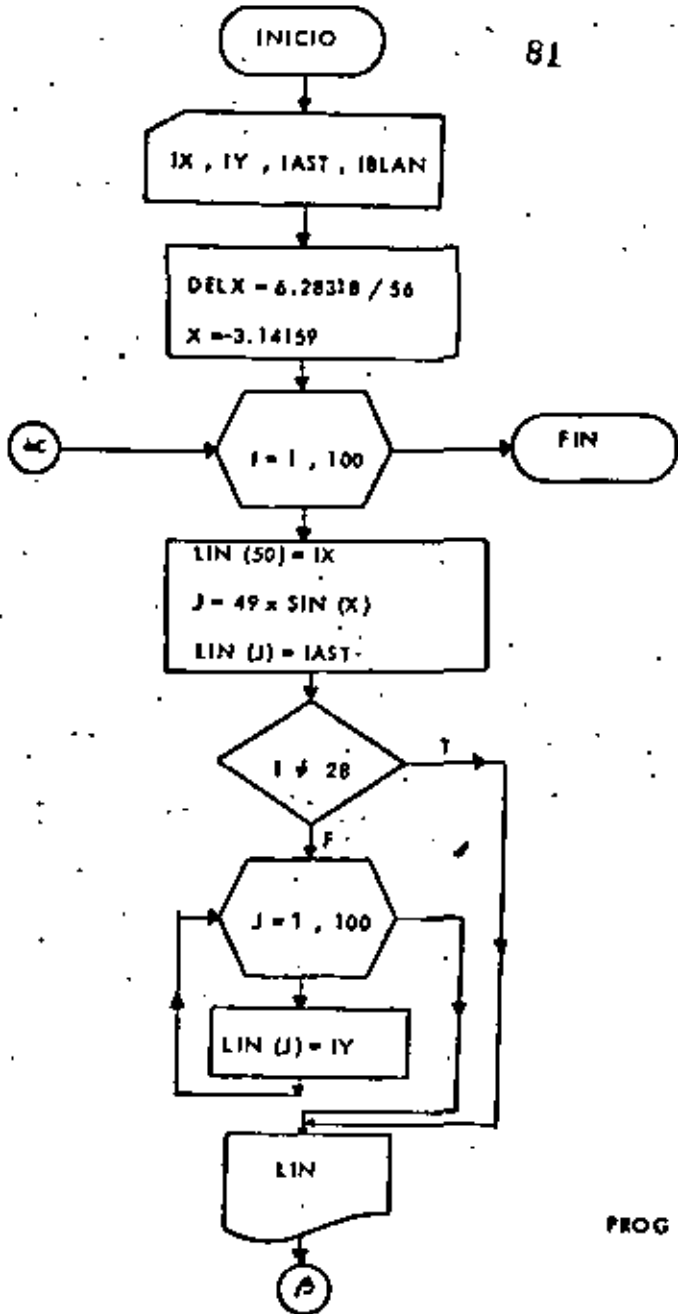
N.C.D.= 1
N.C.D.= 3
N.C.D.= 6
N.C.D.= 1
N.C.D.= 8
N.C.D.= 3

"GRAFICA DE SEN (X)"



\* GRAFICA DE SEN ( X ) \*

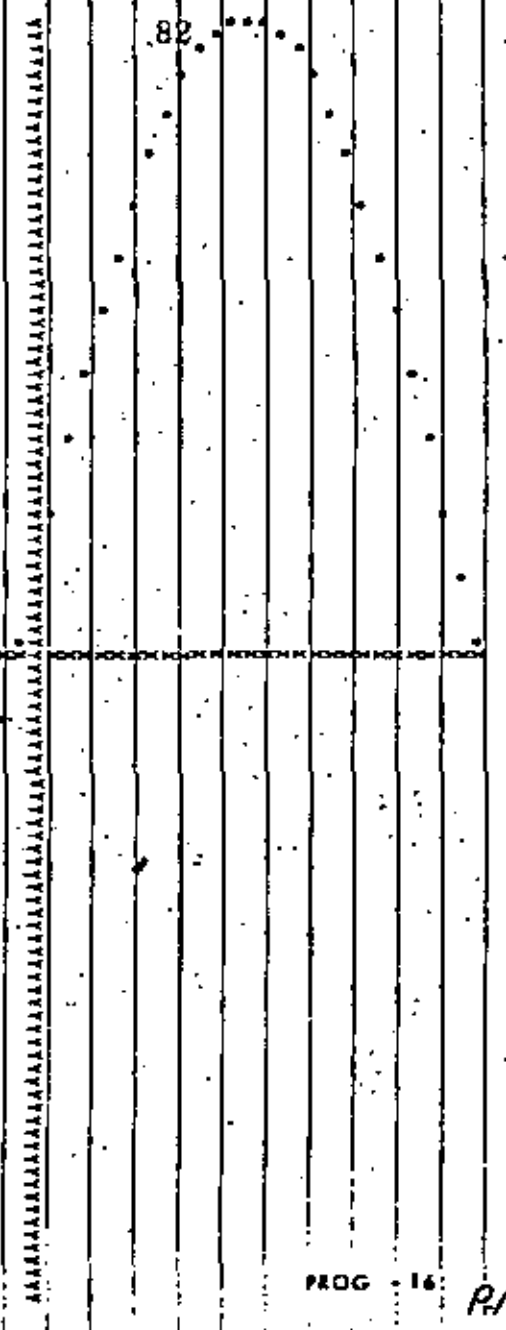
81



PROG -

RESULTADOS

82



```

// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCSICARD,1132 PRINTER)
C-----D I E C I S I E T E-----
C   GRAFICA DE SENIXI
   DIMENSION LIN(100)
100 FORMAT(6A1)
101 FORMAT(10X,100A1)
   LEE=2
   IMP=3
   READ(LEE,100)IX,IY,IJST,IOLAN
   DELX=6.28318/56
   X =3.14159
   DO 200 J=1,100
     LIN(J)=IOLAN
200 CONTINUE
   DO 200 I=1,56
     LIN(50)=IX
     J=0*SIN(IX)+50
     LIN(J)=IJST
     IF (J.NE.28)GO TO 220
     DO 210 J=1,100
       I FUE IGUAL A 20, SE IMPRIME EL EJE Y
       LIN(J)=IY
210 CONTINUE
220 CONTINUE
     WRITE(IMP,101)LIN
     DO 230 J=1,100
       LIN(J)=IOLAN
230 CONTINUE
     X=X-DELX
240 CONTINUE
     CALL EXIT
     END
// XEB
//*
/*

```

83

```

// JOB T
// FOR
*LIST SOURCE PROGRAM
*IOCSICARD,1132 PRINTER)
*ONE WORD INTEGERS
C-----D I E C I S I E T E-----
C   ESTE PROGRAMA CALCULA EL NUMERO DE GRANOS DE MAIZ QUE COBRÓ EL
C   INVENTOR DE AJEDRES
C   FILES
   LEE=2
   IMP=3
C   FORMATOS
100 FORMAT(10X,6HCUADRO,9X,4HSUMA,/)
101 FORMAT(1X,12,2E15,71)
102 FORMAT(//)
103 FORMAT(53X,11H*****
104 FORMAT(53X,11H* FIN *)
   WRITE(IMP,100)
   SUM=0.0
   DO 200 I=1,64
     CUA=2.0** (I-1)
     SUM=SUM+CUA
     WRITE(IMP,101)I,CUA,SUM
200 CONTINUE
   WRITE(IMP,102)
   WRITE(IMP,103)
   WRITE(IMP,104)
   CALL EXIT
   END

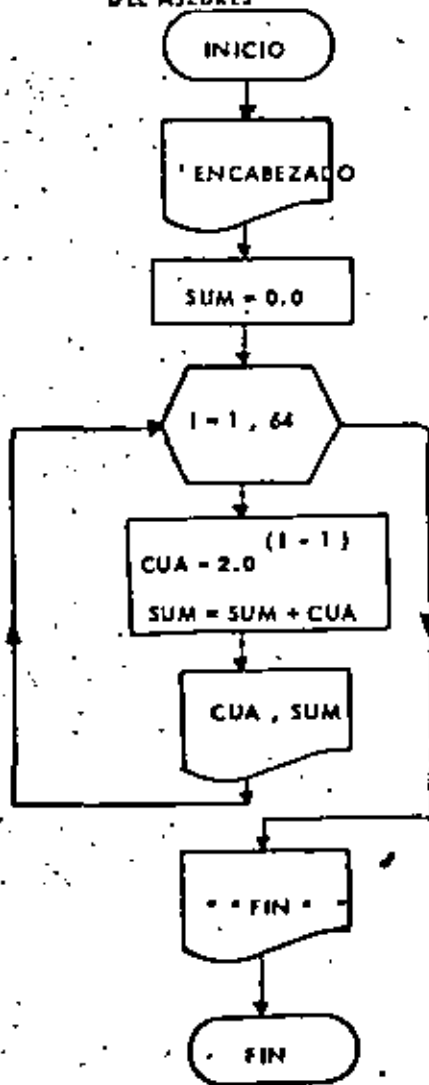
```

84

// XEB  
/\*



\* No. DE GRANOS DE MAIZ GANADOS POR EL INVENTOR DEL AJEDRES \*



85

CUADRO	SUMA
1	0.000000
2	2.000000
3	6.000000
4	14.000000
5	26.000000
6	42.000000
7	62.000000
8	86.000000
9	114.000000
10	146.000000
11	182.000000
12	222.000000
13	266.000000
14	314.000000
15	366.000000
16	422.000000
17	482.000000
18	546.000000
19	614.000000
20	686.000000
21	762.000000
22	842.000000
23	926.000000
24	1014.000000
25	1106.000000
26	1202.000000
27	1302.000000
28	1406.000000
29	1514.000000
30	1626.000000
31	1742.000000
32	1862.000000
33	1986.000000
34	2114.000000
35	2246.000000
36	2382.000000
37	2522.000000
38	2666.000000
39	2814.000000
40	2966.000000
41	3122.000000
42	3282.000000
43	3446.000000
44	3614.000000
45	3786.000000
46	3962.000000
47	4142.000000
48	4326.000000
49	4514.000000
50	4706.000000
51	4902.000000
52	5102.000000
53	5306.000000
54	5514.000000
55	5726.000000
56	5942.000000
57	6162.000000
58	6386.000000
59	6614.000000
60	6846.000000
61	7082.000000
62	7322.000000
63	7566.000000
64	7814.000000
65	8066.000000
66	8322.000000
67	8582.000000
68	8846.000000
69	9114.000000
70	9386.000000
71	9662.000000
72	9942.000000
73	10226.000000
74	10514.000000
75	10806.000000
76	11102.000000
77	11402.000000
78	11706.000000
79	12014.000000
80	12326.000000
81	12642.000000
82	12962.000000
83	13286.000000
84	13614.000000
85	13946.000000
86	14282.000000
87	14622.000000
88	14966.000000
89	15314.000000
90	15666.000000
91	16022.000000
92	16382.000000
93	16746.000000
94	17114.000000
95	17486.000000
96	17862.000000
97	18242.000000
98	18626.000000
99	19014.000000
100	19406.000000

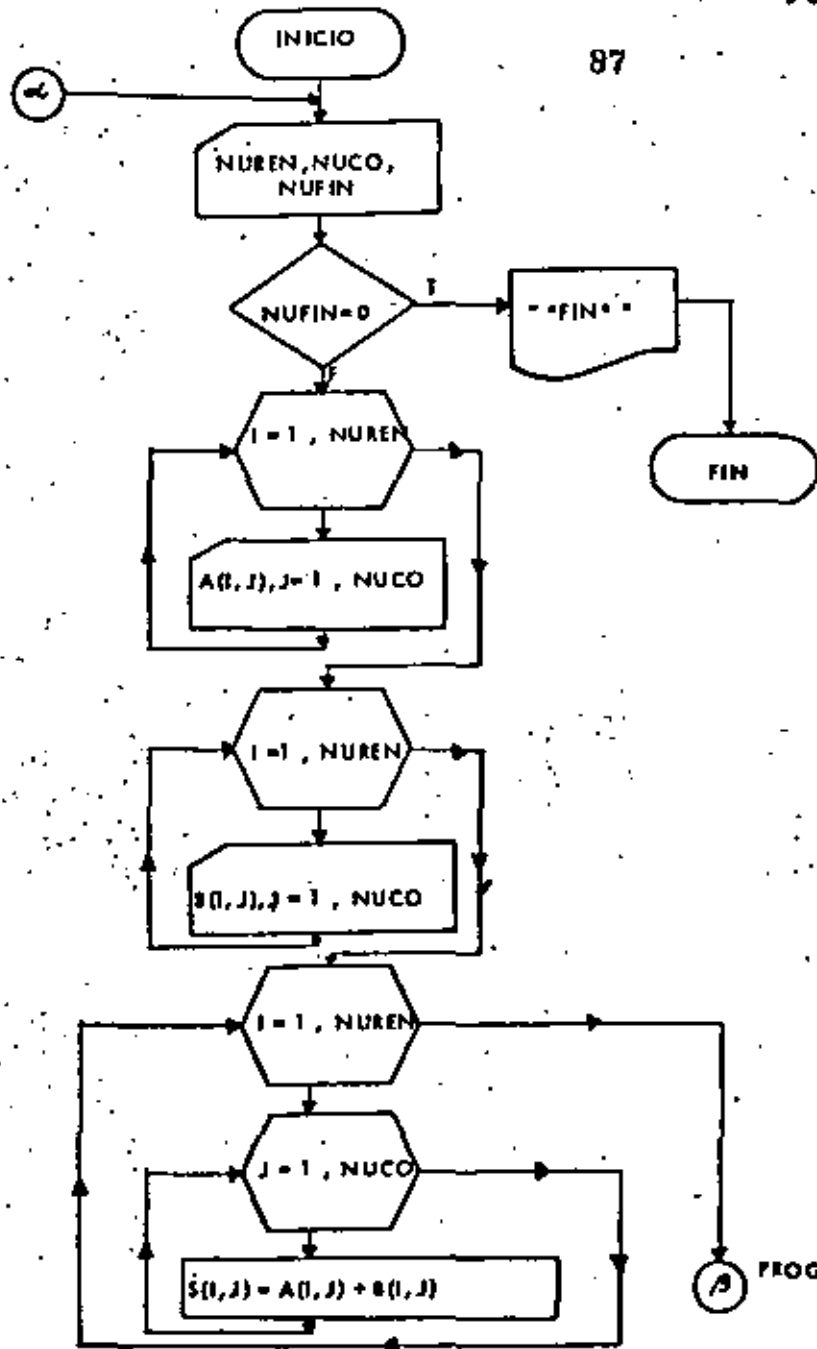
RESULTADOS

86

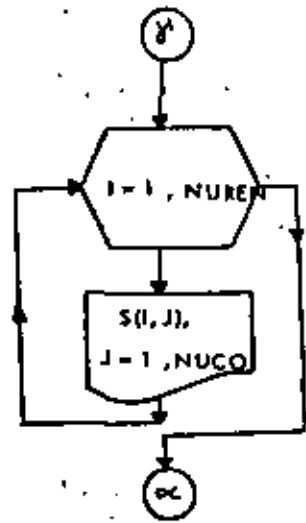
PROG - 17

\*\*\*\*\*  
FIN  
\*\*\*\*\*

P-H

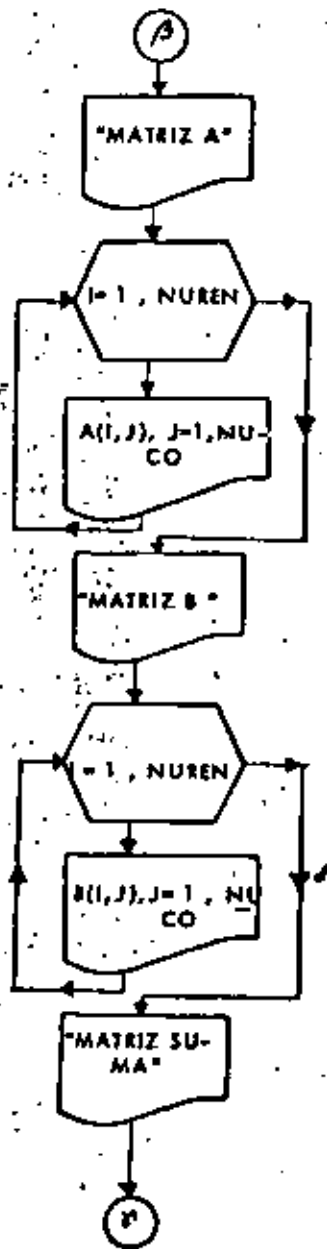


PROG - 18



PROG - 18

\* SUMA DE DOS MATRICES, A y B \*



89

CALL EAST  
END

90

10

```

// KEO
2 2 1
5.9      0.9
12.2     3.4
-4.7     2.1
0.9      -1.2
2 2 1
90.15   -87.82
5.478   12.22
-98.15  87.82
-5.478  -12.22
2 2 0
//
    
```

RESULTADOS

MATRIZ A	
13:288	9:888
MATRIZ B	
4:798	-7:188
0:000	
LA MATRIZ SUMA ES:	
18:288	2:188
MATRIZ A	
9:178	-15:278
MATRIZ B	
90:178	87:078
5:478	-12:278
LA MATRIZ SUMA ES:	
8:888	8:888
..... FIN .....	

PROG - 1B

PR

P-1B

```

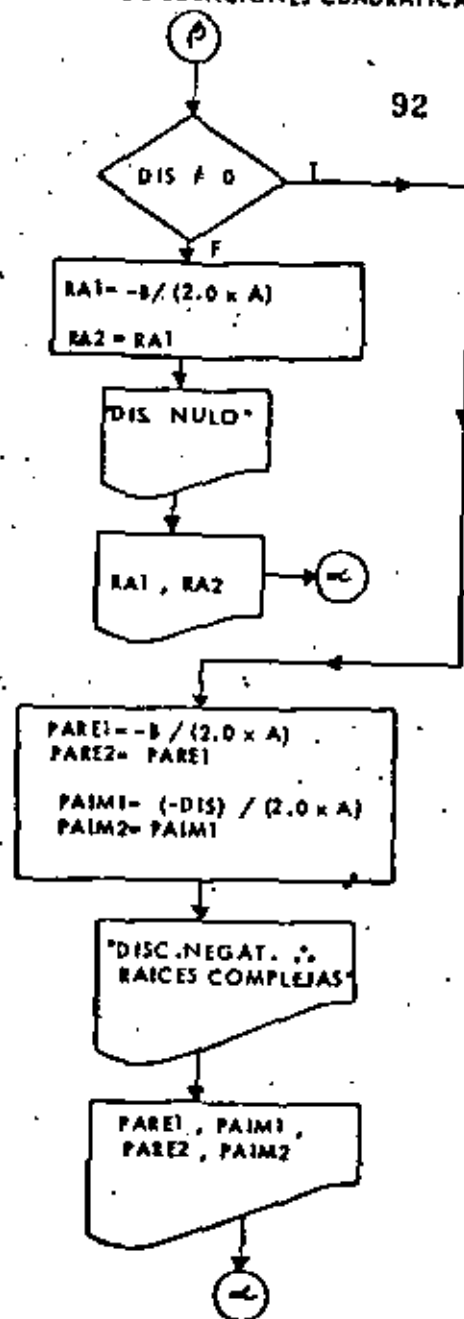
// JOB T
// FOR
*IDCSICARD,1132 PRINTER)
*ONE WORD INTEGERS
*LIST SOURCE PROGRAM
-----D I E C I O C H O-----
C   SUMA DE DOS MATRICES: A Y B
C   EL PROGRAMA ESTA HECHO PARA SUMAR DOS MATRICES DE 10x 10 MAXIMO,
C   SE RESERVAN LUGARES EN LA MEMORIA PARA LAS MATRICES A SUMAR Y PARA
C   LA MATRIZ SUMA.
C   DIMENSION A(10,10),B(10,10),S(10,10)
C   FILES
C       LEE=2
C       IMP=3
C   FORMATOS
100 FORMAT(3I2)
101 FORMAT(10F8,3)
102 FORMAT(///,5X,9HMATRIZ A,///)
103 FORMAT(5X,10(F8,3,2X),/)
104 FORMAT(///,5X,9HMATRIZ B,///)
105 FORMAT(///,5X,10MLA MATRIZ SUMA ES,///)
106 FORMAT(53X,11H*****
107 FORMAT(53X,11H* F I N *
C   LECTURA DEL NUMERO DE RENGLONES DE LAS MATRICES (NUREN) Y DEL NUME
C   RO DE COLUMNAS (NUCO), Y DE UN DETECTOR (NUFIN)
199 READ(LEE,10)NUREN,NUCO,NUFIN
C   ANALISIS DE NUFIN, SI VALE CERO YA NO SE EJECUTA EL PROGRAMA, DE LO
C   CONTRARIO SI.
C   IF(NUFIN.EQ.0)GO TO 1800
C   LECTURA POR RENGLONES DE LA MATRIZ A.
C   DO 200 I=1,NUREN
C   READ(LEE,10)A(I,J),J=1,NUCO)
200 CONTINUE
C   LECTURA POR RENGLONES DE LA MATRIZ B.
C   DO 201 I=1,NUREN
C   READ(LEE,10)B(I,J),J=1,NUCO)
201 CONTINUE
C   SE HARA LA SUMA ELEMENTO A ELEMENTO
C   DO 203 I=1,NUREN
C   DO 202 J=1,NUCO
C   S(I,J)=A(I,J)+B(I,J)
202 CONTINUE
203 CONTINUE
C   IMPRESION DE LA MATRIZ A POR RENGLONES
C   WRITE(IMP,102)
C   DO 204 I=1,NUREN
C   WRITE(IMP,103)A(I,J),J=1,NUCO)
204 CONTINUE
C   IMPRESION DE LA MATRIZ B POR RENGLONES
C   WRITE(IMP,104)
C   DO 205 I=1,NUREN
C   WRITE(IMP,103)B(I,J),J=1,NUCO)
205 CONTINUE
C   IMPRESION DE LA MATRIZ S POR RENGLONES
C   WRITE(IMP,105)
C   DO 206 I=1,NUREN
C   WRITE(IMP,103)S(I,J),J=1,NUCO)
206 CONTINUE
C   GO TO 199
1800 CONTINUE
C   WRITE(IMP,106)
C   WRITE(IMP,107)
C   WRITE(IMP,106)

```

PROC - 18

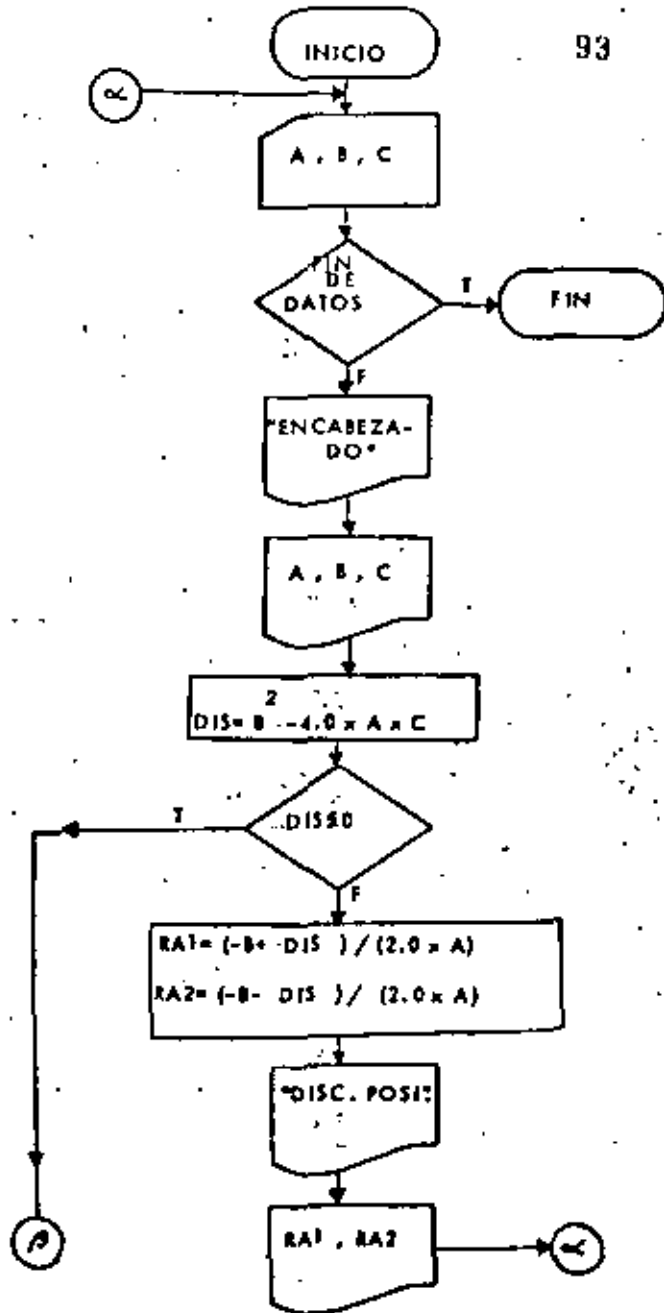
\* SOLUCION DE ECUACIONES CUADRATICAS \*

2a. 12



PROG - 19

SOLUCION DE ECUACIONES CUADRATICAS



94

RESULTADOS

LOS COEFICIENTES DE LA ECUACION SON		
A = 1.00000	B = 2.00000	C = 3.00000
EL DISCRIMINANTE ES NEGATIVO POR TANTO RAICES COMPLEJAS		
X1 = -1.00000 + 1.41421 IMA	X2 = -1.00000 - 1.41421 IMA	
LOS COEFICIENTES DE LA ECUACION SON		
A = 5.00000	B = 10.00000	C = 5.00000
EL DISCRIMINANTE ES NULO POR TANTO RAICES IGUALES		
X1 = -1.00000	X2 = -1.00000	
LOS COEFICIENTES DE LA ECUACION SON		
A = 1.00000	B = 10.00000	C = 27.00000
EL DISCRIMINANTE ES NEGATIVO POR TANTO RAICES COMPLEJAS		
X1 = -5.00000 + 1.41421 IMA	X2 = -5.00000 - 1.41421 IMA	
LOS COEFICIENTES DE LA ECUACION SON		
A = 1.00000	B = 20.00000	C = 1.00000
EL DISCRIMINANTE ES POSITIVO POR TANTO RAICES REALES		
X1 = -0.05038	X2 = -19.94962	

```

// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*DCS(ICARD,1132 PRINTER)
-----D I E C I N U E V E-----
C SOLUCION DE ECUACIONES CUADRATICAS.
200 FORMAT(3F11.5)
101 FORMAT(///,2X,36HLOS COEFICIENTES DE LA ECUACION SON,///)
102 FORMAT(2X,3HA= ,F11.5,2X,3HB= ,F11.5,2X,3HC= ,F11.5,///)
103 FORMAT(2X,52HEL DISCRIMINANTE ES POSITIVO,POR TANTO RAICES REALES,
///)
104 FORMAT(5X,3HX1=,F11.5,10X,3HX2=,F11.5,///)
105 FORMAT(2X,49HEL DISCRIMINANTE ES NULO,POR TANTO RAICES IGUALES,///)
106 FORMAT(5X,3HX1=,F11.5,10X,3HX2=,F11.5,///)
107 FORMAT(2X,55HEL DISCRIMINANTE ES NEGATIVO,POR TANTO RAICES COMPLEJ
IAS,///)
108 FORMAT(5X,3HX1=,F11.5,2H +,F11.5,4H IMA,10X,3HX2=,F11.5,2H -,F11.5
),4H IMA,///)
LEE=2
IMP=3
C LEE LOS COEFICIENTES
200 READ(LEE,100,END=240)A,B,C
C IMPRIME LA ECUACION
WRITE(IMP,101)
WRITE(IMP,102)A,B,C
C CALCULO DEL DISCRIMINANTE
DIS=B**2-4.0*A*C
IF(DIS.LE. 0.0)GO TO 210
C RAICES REALES DIFERENTES
RA1=(-B+SQRT(DIS))/2.0*A)
RA2=(-B-SQRT(DIS))/2.0*A)
WRITE(IMP,103)
WRITE(IMP,104)RA1,RA2
GO TO 230
210 CONTINUE
IF (DIS.EQ. 0.0) GO TO 220
C RAICES REALES IGUALES
RA1=-B/(2.0*A)
RA2=RA1
WRITE(IMP,105)
WRITE(IMP,106)RA1,RA2
GO TO 230
220 CONTINUE
C RAICES COMPLEJAS
PARE1=-B/(2.0*A)
PARE2=PARE1
PAIM1=SQRT(-DIS)/(2.0*A)
PAIM2=PAIM1
WRITE(IMP,107)
WRITE(IMP,108)PARE1,PAIM1,PARE2,PAIM2
GO TO 230
C ENDF
230 CONTINUE
GO TO 200
240 CONTINUE
CALL EXIT
END
// XEO
1.0 2.0 3.0 1.
5.0 10.0 5.0 1.
1.0 10.0 27.0 1.

```

95

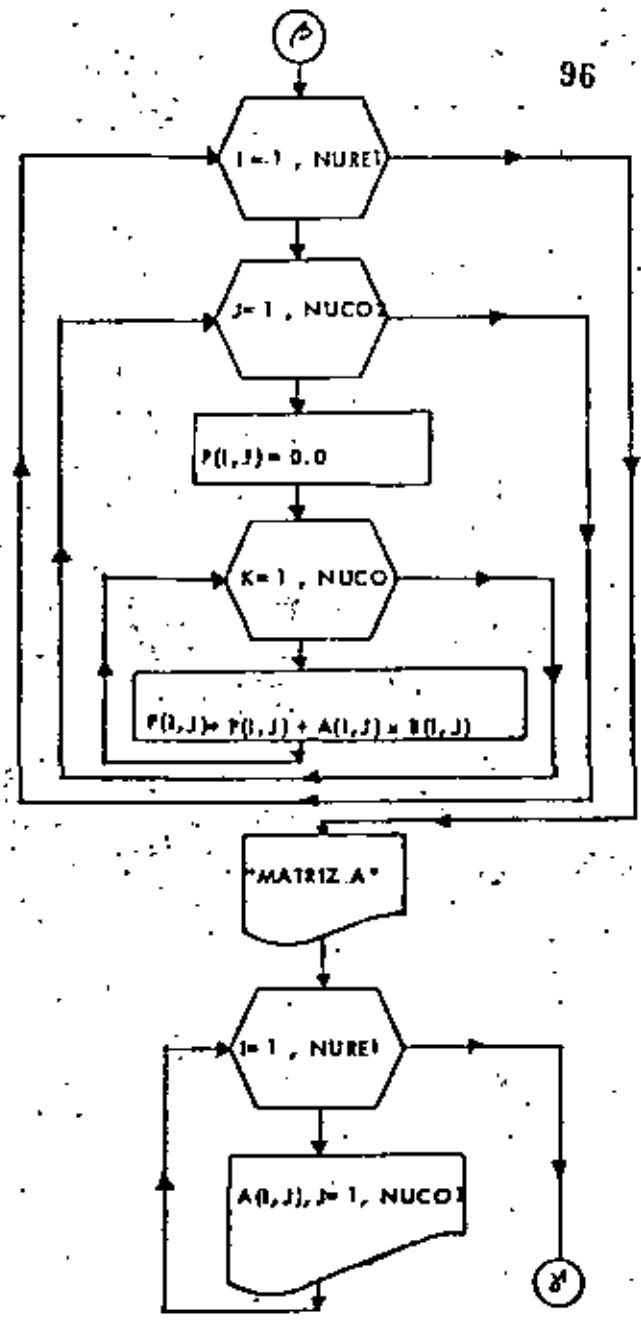
95

\* PRODUCTO DE DOS MATRICES \*

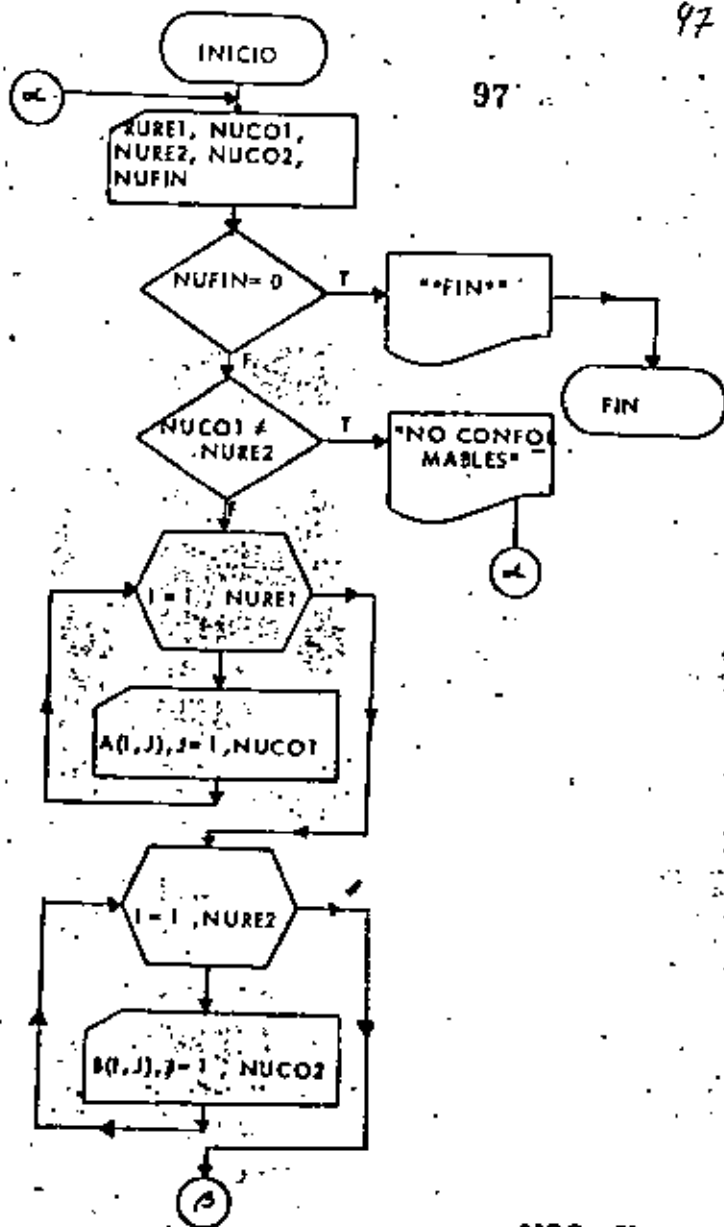
2a.

96

96



\* PRODUCTO DE DOS MATRICES \*



PROG - 20

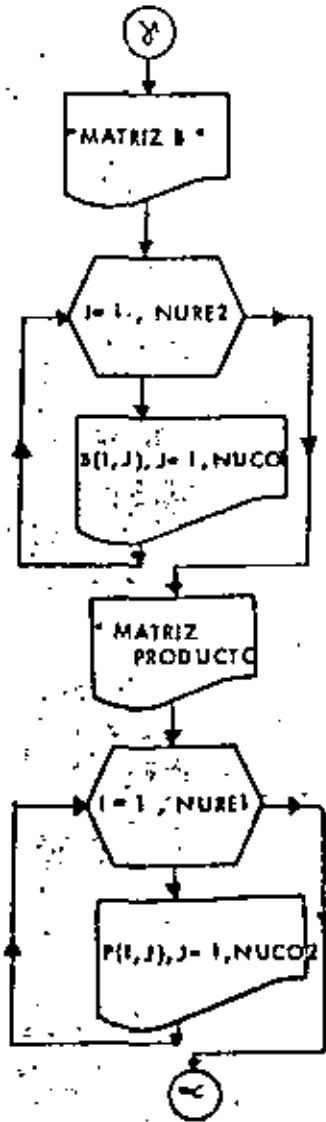
98

```

// JOB 1
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IOCS(=CARD,1332 PRINTER)
-----Y E. I. N. T. E-----
C EL PROGRAMA REALIZA EL PRODUCTO DE DOS MATRICES DE 10 X 10 MAXIMO.
C UNA ES LA MATRIZ A(NURE1,NUCO1).
C LA OTRA ES LA MATRIZ B(NUREZ,NUCO2).
C SE RESERVAN LUGARES EN LA MEMORIA PARA LAS MATRICES QUE SE VAN A
C MULTIPLICAR Y PARA LA MATRIZ PRODUCTO.
C DIMENSION A(10,10),B(10,10),P(10,10)
C FILES
    LEE=2
    IMP=3
C
C FORMATOS
100 FORMAT(5I2)
101 FORMAT(10F8,3)
102 FORMAT(///,5X,9#MATRIZ A:////)
103 FORMAT(5X,10I8,3,2X) //1
104 FORMAT(///,5X,9#MATRIZ B:////)
105 FDMAT(///,5X,22#LA MATRIZ PRODUCTO ES:////)
106 FORMAT(5X,10E15,7) //1
107 FORMAT(///,5X,76#EL PRODUCTO NO SE PUEDE LLEVAR A CABO YA QUE LAS
MATRICES NO SON CONFORMABLES:////)
108 FORMAT(////)
109 *FORMAT(53X,11#)*****
110 FDMAT(53X,11# FIN *)
C LECTURA DE LOS NUMEROS DE RENGLONES Y DE COLUMNAS DE CADA MATRIZ
C Y DEL DETECTOR NUFIN.
199 READ(LEE,100)NURE1,NUCO1,NUREZ,NUCO2,NUFIN
C ANALISIS DEL VALOR DE NUFIN. SI VALE CERO EL PROGRAMA NO SE LLEVA
C A CABO. DE LO CONTRARIO SI,
C IF(NUFIN.EQ.0)GO TO 1000
C SE VE SI LAS MATRICES SON CONFORMABLES.
C IF(NUCO1.NE.NUREZ)GO TO 999
C LECTURA POR RENGLONES DE LA MATRIZ A.
DO 200 I=1,NURE1
  READ(LEE,101) (A(I,J),J=1,NUCO1)
200 CONTINUE
C LECTURA POR RENGLONES DE LA MATRIZ B.
DO 201 I=1,NUREZ
  READ(LEE,101) (B(I,J),J=1,NUCO2)
201 CONTINUE
C SE REALIZA EL PRODUCTO
DO 204 I=1,NURE1
  DO 203 J=1,NUCO2
    P(I,J)=0.0
    DO 202 K=1,NUCO1
      P(I,J)=P(I,J)+A(I,K)*B(K,J)
202 CONTINUE
203 CONTINUE
204 CONTINUE
C IMPRESION DE LA MATRIZ A POR RENGLONES
WRITE(IMP,102)
DO 205 I=1,NURE1
  WRITE(IMP,103) (A(I,J),J=1,NUCO1)
205 CONTINUE
C IMPRESION DE LA MATRIZ B POR RENGLONES.
WRITE(IMP,104)
DO 206 I=1,NUREZ
  WRITE(IMP,103) (B(I,J),J=1,NUCO2)
206 CONTINUE
  
```

PROG - 20

• PRODUCTO DE DOS MATRICES •



99

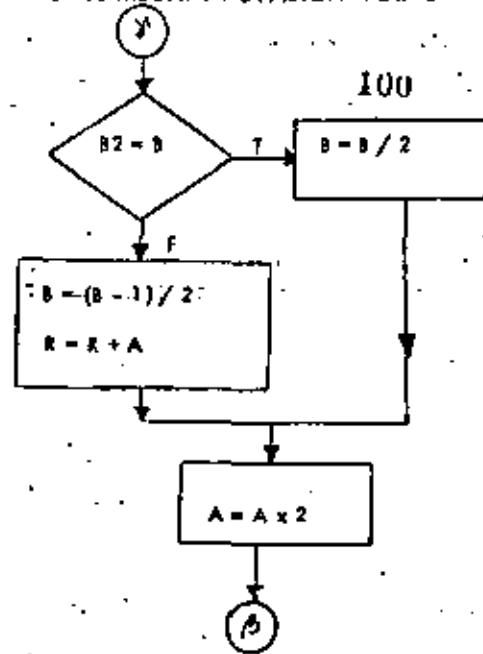
30.

99

• MULTIPLICACION DE DOS NUMEROS UTILIZANDO EXCLUSIVAMENTE MULTIP. Y DIVISION POR 2 •

20.

100



100



```

C IMPRESION DE LA MATRIZ P POR RENGLONES.
  WRITE(IMP,105)
  DD 207 I=I,NUREI
  WRITE(IMP,106)(P(I,J),J=1,NUC02)
207 CONTINUE
  GO TO 199
900 CONTINUE
  WRITE(IMP,107)
  GO TO 199
1000 CONTINUE
  WRITE(IMP,108)
  WRITE(IMP,109)
  WRITE(IMP,110)
  WRITE(IMP,109)
  CALL EXIT
  END
// XEQ
  2 2 2 210
  15.54 -42.07
  -1.22  0.0
  1950.75 -12.0
  0.001  5.0
  2 2 2 210
  0.2  90.75
  -12.5  32.52
  1000.01  0.0
  -1.52  15.51
  2 3 2 210
  2 2 2 200
  /

```

101

RESULTADOS

102

102

MATRIZ A:

15.540 -42.070  
0.001 5.000

MATRIZ B:

1950.750 -12.000  
0.001 5.000

LA MATRIZ PRODUCTO ES:

303161E+05 -3960300E+03  
-122000E+02 -1100000E+01

MATRIZ A:

0.200 90.750  
0.001 -12.500

MATRIZ B:

1000.010 0.000  
-1.520 15.510

LA MATRIZ PRODUCTO ES:

4990200E+02 -1531013E+04  
1000000E+02 -11910750E+03

EL PRODUCTO NO SE PUEDE LLEVAR A CABO YA QUE LAS MATRICES NO SON CONFORMABLES

\*\*\*\*\*  
\* FIN \*  
\*\*\*\*\*

```

// JOB 7
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*IDCSICARD,1132 PRINTER;
-----V C I N T I U N D-----
C   MULTIPLICACION DE DOS NUMEROS
C   UTILIZANDO SOLO MULTIPLICACIONES Y
C   DIVISIONES POR 2
C   INTEGER A,B,C,R,AA,HB
C   WRITE (3,101)
101 FORMAT (3I1)
C   WRITE (3,102)
102 FORMAT (19X,1MA,3X,1MX,4X,1MB,4X,1N,4X,1MC)
200 READ (2,100,END=260)A,B
100 FORMAT (2I3)
R=0
AA=A
BB=B
210 IF (B.EQ.1) GO TO 240
B1=B/2
B2=B1*2
IF (B2.EQ.B) GO TO 220
ES IMPAR
B=(B-1)/2
R=R+A
GO TO 230
220 CONTINUE
ES PAR
B=B/2
CONTINUE
AA*A*2
GO TO 210
240 CONTINUE
C=A+R
WRITE (3,103) AA,BB,C
103 FORMAT (3I10)
GO TO 200
260 CALL EXIT
END

```

103

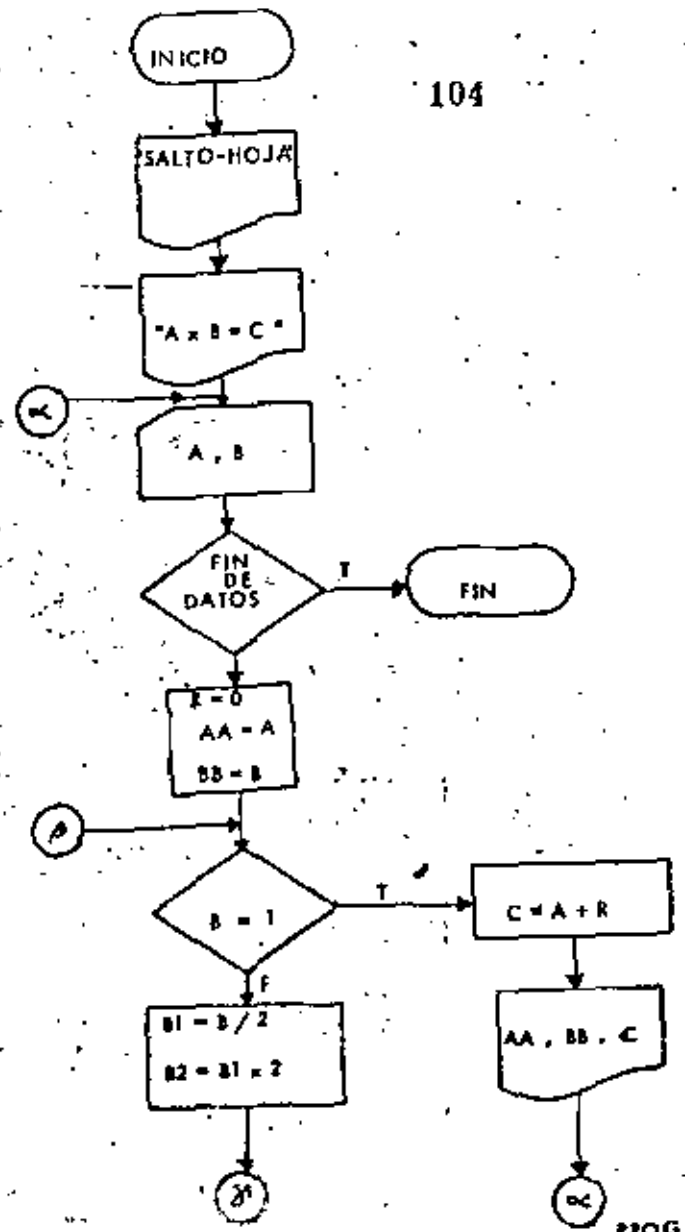
```

// XED
60 80
19 17
66 35
48 11
77 99
/M

```

RESULTADOS

101	3	1	1
102	19	1	1
103	3	1	1



104

```

// JOB T
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*JOCS: CARD.1132 PRINTER
----- V C I N T I U N O -----
C  MULTIPLICACION DE DOS NUMEROS
C  UTILIZANDO SOLO MULTIPLICACIONES Y
C  DIVISIONES POR 2
C  INTEGER A,B,C,R,AA,BB
C  WRITE (3,101)
101 FORMAT (11H1)
C  WRITE (3,102)
102 FORMAT (9X,1HA,3X,1HX,4X,1HB,4X,1H=,4X,1HC)
200 READ (2,100,END=260) A,B
100 FORMAT (2I3)
R=0
AA=A
BB=B
210 IF (B.EQ.1) GO TO 240
    B1=B/2
    B2=B1*2
    IF (B2.EQ.B) GO TO 220
    ES IMPAR
    B=(B-1)/2
    R=R+A
    GO TO 230
220 CONTINUE
    ES PAR
    B=B/2
230 CONTINUE
    A=A*2
    GO TO 210
240 CONTINUE
C=A*R
WRITE (3,103) AA,BB,C
103 FORMAT (3I10)
GO TO 200
260 CALL EXIT
END
// XEB
60 88
10 17
60 35
40 11
77 99
/*

```

105

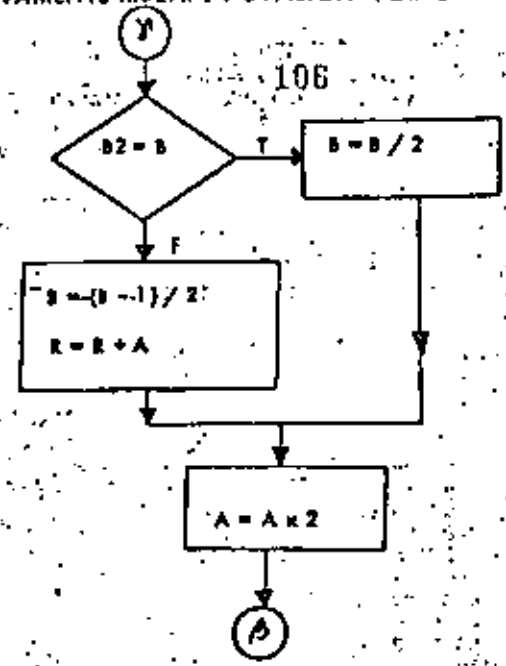
RESULTADOS

Variable	Valor
A	1000
B	1000
C	1000000
R	2000000
AA	1000
BB	1000

106

\* MULTIPLICACION DE DOS NUMEROS UTILIZANDO EXCLUSIVAMENTE MULT. Y DIVISION POR 2 \*

2a.



106



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

**INTRODUCCION A LA PROGRAMACION Y  
COMPUTACION ELECTRONICA**

**SISTEMAS DE ECUACIONES CON FORTRAN  
SOLUCION DE APLICACIONES MATEMATICAS**

**OCTUBRE, 1982**

## 1. ALGEBRA MATRICIAL

1

## 1.1 Introducción

Una matriz es un arreglo rectangular de elementos distribuidos en "m" renglones y "n" columnas, si a la matriz se le denota por la letra  $A$ , entonces al elemento del "i-ésimo" renglón y de la "j-ésima" columna se le representará por el símbolo  $a_{ij}$ . Generalmente una matriz se representa mediante paréntesis cuadrados como se muestra a continuación:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \quad (1.1)$$

Los elementos que componen una matriz pueden ser de diversos tipos: números reales, números complejos, funciones en el dominio del tiempo, etc.

Al ser una matriz un arreglo ordenado de elementos, permite que al aplicar cierta metodología a dicho arreglo se obtenga una serie de resultados que responden a las interrogantes por las que se originó el arreglo; entre algunos de los procesos en los que se utilizan arreglos matriciales se tienen: jerarquización de actividades, almacenamiento de datos, inventarios, representación de sistemas dinámicos, sistemas de ecuaciones, etc.

Existen ciertas distribuciones típicas de los elementos de una matriz y de acuerdo a ellas se clasifica a las matrices en diferentes tipos; entre los que se tienen:

## Matriz Cuadrada

Es una matriz en la que el número de renglones es igual al número de columnas, es decir,  $m=n$ . Por ejemplo:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad (1.2)$$

2

## Matriz Nula

Es una matriz de orden cualquiera, en la que todos los elementos son nulos; por ejemplo:

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (1.3)$$

Se acostumbra denotarla por el símbolo  $O$ .

## Matriz Identidad

Es una matriz cuadrada en la cual los elementos de la diagonal principal son unitarios y el resto son nulos, es decir:

$$\delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

Se suele denotarla como  $I_n$  donde "n" indica el orden de la matriz y al símbolo  $\delta_{ij}$  se le conoce como delta de Kronecker. Por ejemplo:

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.4)$$

## Matriz Diagonal

Es una matriz cuadrada en la que los elementos que no pertenecen a la diagonal principal son nulos, es decir:

$$a_{ij} = 0 \quad \text{si } i \neq j \quad (1.5)$$

Un ejemplo de este tipo de matriz sería:

$$A = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 20 \end{bmatrix} \quad (1.6)$$

## Matriz Transpuesta

Es una matriz cuadrada que se obtiene a partir de una matriz dada  $A$  intercambiando renglones con columnas. Se le denota con el símbolo  $A^T$  y se cumple que:

$$a_{ij}^T = a_{ji} \quad (1.7)$$

## Matriz Simétrica

Es una matriz cuadrada  $B$  para la que se cumple:

$$B = B^T \quad (1.8)$$

o equivalentemente:

$$b_{ij} = a_{ij} + b_{ij} \quad (2.9)$$

Entre las matrices se definen dos operaciones básicas:

- suma o resta de matrices,
- multiplicación matricial.

## 2.2 Suma Matricial

### 2.2.1 Objeto

Obtener la suma de dos matrices de igual orden, o sea:

$$C = A + B \quad (2.10)$$

### 2.2.2 Método

Para poder efectuar la suma de dos matrices  $[A+B]$  se requiere que sean conformables para la suma, lo cual implica que el orden de las dos matrices es igual. En otras palabras:

si  $A$  es de orden  $(m \times n)$

y  $B$  es de orden  $(n \times m)$

la suma  $C=A+B$  es posible solo si  $m=n$  y  $n=m$ .

Los elementos de la matriz suma están dados por la siguiente relación:

$$c_{ij} = a_{ij} + b_{ij} \quad (2.11)$$

El restar dos matrices equivale a cambiar el signo de todos los elementos de una de ellas y efectuar la suma, esto es:

$$M = X - Y = X + [-Y] \quad (2.12)$$

### 2.2.3 Descripción del Programa

#### a) Subrutinas requeridas:

SUBROUTINE SUMAT(A,B,C,M,N), esta subrutina efectúa la suma matricial, el programa principal lee e imprime resultados.

#### b) Descripción de las variables:

Para la subrutina SUMAT:

N cantidad de renglones de cada una de las matrices que se desea sumar.

M cantidad de columnas de cada una de las matrices que se desea sumar.

A(I,J) matriz sumando de orden  $N \times M$

B(I,J) matriz sumando de orden  $N \times M$

C(I,J) matriz suma

Para el programa principal:

N cantidad de renglones de las matrices

M cantidad de columnas de las matrices

A(I,J) matriz sumando de orden  $N \times M$

B(I,J) matriz sumando de orden  $N \times M$

C(I,J) matriz suma

#### c) Dimensiones:

la proposición DIMENSION, deberá ser modificada tanto en el programa principal como en la subrutina cuando:

$$N > 10 \quad \text{y/o} \quad M > 10$$

#### d) Formatos para los datos de entrada:

SEC. TARJETAS	FORMATO	INFORMACION
1	(2I5)	N, M
2	(8F10.0)	A(I,J), se dan los elementos de la matriz renglón por renglón. Emplear tantas tarjetas como se requiera.
3	(8F10.0)	B(I,J), igual que para la matriz A.

-----  
 otros paquetes de datos (opcional)  
 -----

N

TARJETA EN BLANCO, al finalizar toda la información.

#### e) Diagrama de bloques

Fig. 2.1 Diagrama de bloques para el programa principal

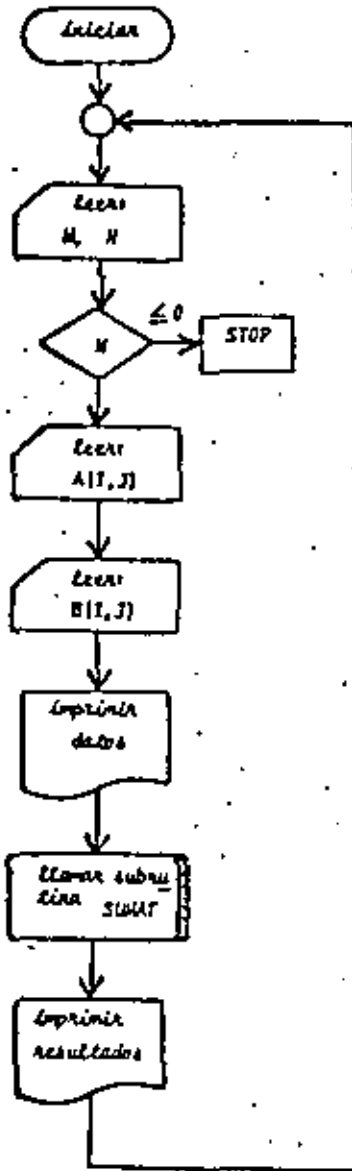
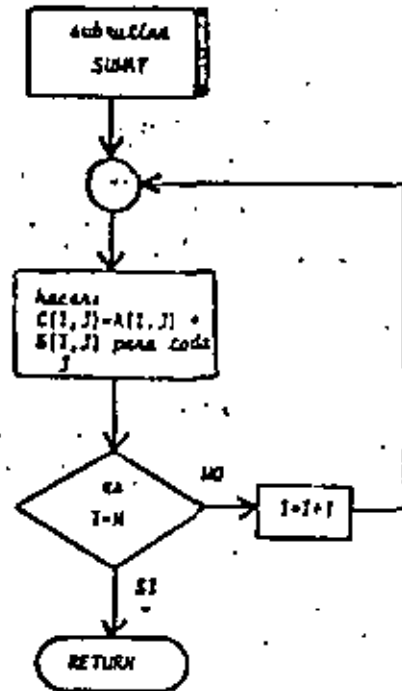


Fig. 2.2 Diagrama de bloques para la subrutina SUMAT



```

C PROGRAMAS PARA EFECTUAR LA SUMA DE DOS MATRICES
C EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C CANTIDAD DE renglones DE LAS MATRICES
C CANTIDAD DE COLUMNAS DE LAS MATRICES
C A Y MATRICES QUE SE VAN A SUMAR
C MATRIZ SUMA
-----
C DIMENSION A(10,10),B(10,10),C(10,10)
C LECTURA DE DATOS
C
1 READ(5,1001) N,M
  IF(N) 2,2,3
2 CALL LIT
3 DO 4 I=1,N
4 READ(5,150) (A(I,J),J=1,M)
  DO 5 J=1,M
5 READ(5,150) (B(I,J),J=1,M)
-----
C IMPRESION DE DATOS
C
WRITE(6,2001)
DO 6 I=1,N
6 WRITE(6,250) (A(I,J),J=1,M)
  WRITE(6,300)
  DO 7 J=1,M
7 WRITE(6,250) (B(I,J),J=1,M)
-----
C LLAMADO DE SUBROUTINA PARA EFECTUAR LA SUMA
C
CALL SUMAT(A,B,C,M,N)
-----
C IMPRESION DE RESULTADOS
C
WRITE(6,350)
DO 8 I=1,N
8 WRITE(6,250) (C(I,J),J=1,M)
  GO TO 1
-----
C FORMATOS DE LECTURA E IMPRESION
C
100 FORMAT(2I5)
150 FORMAT(7F10.0)
200 FORMAT(1H1,5(//),10X,'LAS MATRICES POR SUMAR SON:',3(//),10X,'MATRIZ',
  1A, //)
250 FORMAT(//,3X,10(E10,3,2X))
300 FORMAT(3(//),10X,'MATRIZ #1, //)
350 FORMAT(6(//),10X,'MATRIZ SUMA', //)
  END

```

Fig. 2.3 Listado del programa principal

```

C SUBROUTINE SUMAT(A,B,C,M,N)
C SUBROUTINA PARA SUMAR MATRICES
C SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C A Y MATRICES QUE SE VAN A SUMAR
C CANTIDAD DE renglones DE LAS MATRICES
C CANTIDAD DE COLUMNAS DE LAS MATRICES
C DIMENSION A(10,10),B(10,10),C(10,10)
DO 1 I=1,N
DO 1 J=1,M
1 C(I,J)=A(I,J)+B(I,J)
  RETURN
  END

```

Fig. 2.4 Listado de la subrutina SUMAT



## 2.2.4 Ejemplo

En una tienda de artículos eléctricos se venden resistencias eléctricas de 1/4, 1/2 y 1 Watt de potencia en seis diferentes valores resistivos.

Si las existencias un viernes por la tarde son:

	1/4	1/2	1
100 $\Omega$	200	380	275
150 $\Omega$	400	250	275
1.0 K	500	175	325
1.5 K	800	225	150
10.0 K	600	380	180
15.0 K	550	250	220

y el sábado se recibe una remesa con las siguientes características:

	1/4	1/2	1
100 $\Omega$	80	90	50
150 $\Omega$	90	100	55
1.0 K	75	90	60
1.5 K	65	95	55
10.0 K	80	100	60
15.0 K	75	110	60

Determine las resistencias que tendrá en inventario el establecimiento el lunes por la mañana dado que ni el sábado ni el domingo hubo ventas.

## • SOLUCION

TABLA 2.1 Datos para el problema del ejemplo 2.2.4

$N=6$

$M=3$

<u>A</u>	200	380	275
	400	250	275
	500	175	325
	800	225	150
	600	380	180
	550	250	220

<u>B</u>	80	90	50
	90	100	55
	75	90	60
	65	95	55
	80	100	60
	75	110	60

TABLA 2.2 Resultados del problema del ejemplo 2.2.4

LAS MATRICES POR SUMAR SON		
MATRIZ A		
.200E+03	.340E+03	.275E+03
.400E+03	.250E+03	.275E+03
.500E+03	.175E+03	.325E+03
.600E+03	.225E+03	.150E+03
.690E+03	.300E+03	.100E+03
.550E+03	.250E+03	.220E+03
MATRIZ B		
.300E+02	.900E+02	.500E+02
.900E+02	.100E+03	.550E+02
.750E+02	.900E+02	.600E+02
.450E+02	.950E+02	.550E+02
.800E+02	.100E+03	.600E+02
.750E+02	.110E+03	.600E+02
MATRIZ SUMA		
.200E+03	.470E+03	.325E+03
.490E+03	.350E+03	.330E+03
.575E+03	.265E+03	.385E+03
.845E+03	.320E+03	.250E+03
.600E+03	.400E+03	.240E+03
.625E+03	.360E+03	.280E+03

### 2.3 Multiplicación Matricial

#### 2.3.1 Objeto

Dadas dos matrices A y B, obtener el producto matricial C de la forma:

$$C = A \times B \quad (2.13)$$

#### 2.3.2 Método

Para efectuar el producto entre dos matrices [A × B] se requiere que las matrices sean conformables para la multiplicación, lo que equivale a que el número de columnas de la matriz premultiplicadora (A) sea igual al número de renglones de la postmultiplicadora (B), es decir:

si A es de orden (m × n)

y B es de orden (n × l)

el producto matricial AB será posible solo si n = n y el orden de la matriz producto será (m × l).

Si la matriz C representa la matriz resultante del producto matricial AB, entonces el elemento  $c_{ij}$  está dado por:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj} \quad i=1, \dots, m \quad j=1, \dots, l \quad (2.14)$$

Es importante hacer notar que el producto matricial no es conmutativo, esto es:

$$A \times B \neq B \times A$$

#### 2.3.3 Descripción del Programa

##### a) Subrutinas requeridas:

SUBROUTINE MULTMA(A,B,N,M,L,X), esta subrutina efectúa el producto matricial A × B. El programa principal se emplea para la lectura de datos e impresión de resultados.

##### b) Descripción de las variables:

Para la subrutina MULTMA:

A(I,J) matriz premultiplicadora de orden M × N

B(I,J) matriz postmultiplicadora de orden N × l

X(I,J) matriz producto de orden M × l

Para el programa principal:

A(I,J) matriz premultiplicadora de orden M × N

B(I,J) matriz postmultiplicadora de orden N × l

X(I,J) matriz producto de orden M × l

##### c) Dimensiones:

La proposición DIMENSION deberá ser modificada tanto en el programa principal como en la subrutina cuando:

$$M > 10 \text{ y/o } N > 10 \text{ y/o } l > 10$$

##### d) Formatos para los datos de entrada:

SEC. TARJETAS	FORMATO	INFORMACION
1	(3T5)	M, N, L
2	(8F10.0)	A(I,J), los elementos de la matriz se dan renglón por renglón. Emplear la cantidad de tarjetas que sea necesaria.
3	(8F10.0)	B(I,J), igual que en el caso anterior.

-----  
otros paquetes de datos [opcional]  
-----

x

TARJETA EN BLANCO, al finalizar toda la información

##### e) Diagrama de bloques:

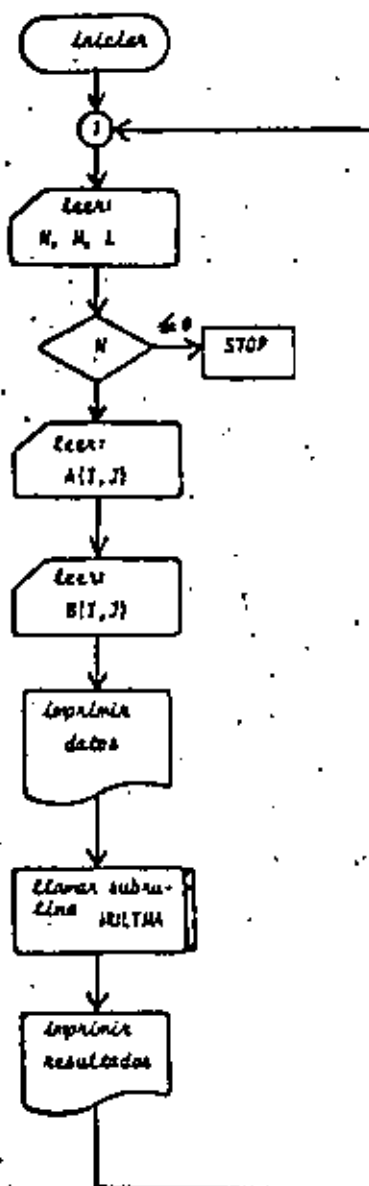


Fig. 2.5 Diagrama de bloques para el programa principal

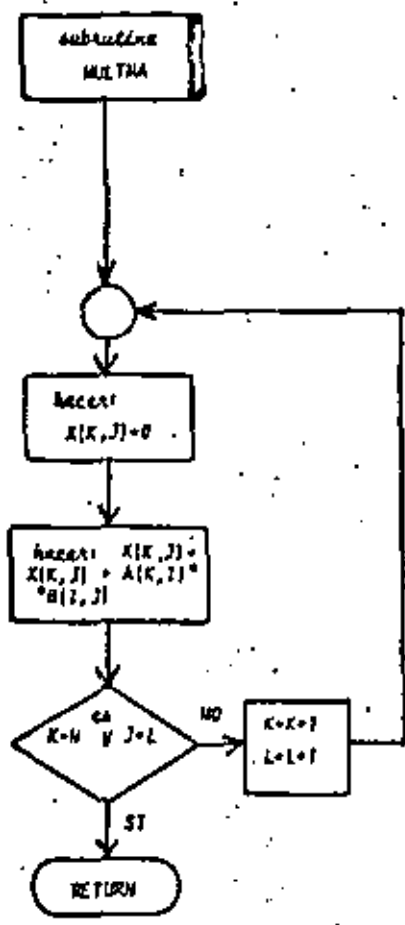


Fig. 2.6 Diagrama de bloques para la subrutina MULTMA

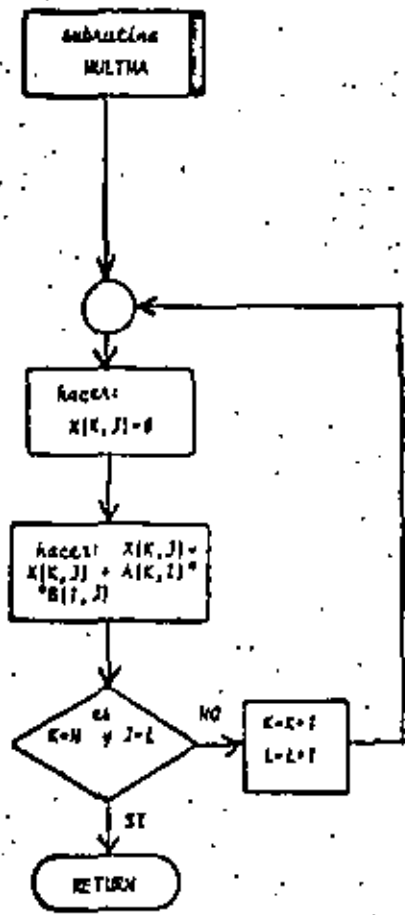


Fig. 2.6 Diagrama de bloques para la subrutina MULTMA

6) Listado:

```

C   PROGRAMA PARA EFECTUAR PRODUCTOS MATRICIALES
C   SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C   A=MATRIZ PREMULTIPLICADORA DE ORDEN (M*N)
C   B=MATRIZ POSTMULTIPLICADORA DE ORDEN (N*L)
C   X=MATRIZ PRODUCTO DE ORDEN (M*L)

DIMENSION A(10,10),B(10,10),X(10,10)
IM=5
IN=6

C   LECTURA DE DATOS
1  READ(1,10) M,N,L
   IF(N1=2,24)
2  CALL EXIT
3  GO TO 1010N
4  READ(1,11) ((A(I,J),J=1,N),I=1,M)
   GO TO 1010N
5  READ(1,12) ((B(I,J),J=1,L),I=1,N)
C   IMPRESION DE DATOS
WRITE(1,13)
DO 4 I=1,M
4  WRITE(1,14) ((A(I,J),J=1,N),I=1,N)
   WRITE(1,14)
GO TO 1010N
7  WRITE(1,15) ((B(I,J),J=1,L),I=1,N)
   WRITE(1,15)
C   LLAMADO DE SUBROUTINA PARA EFECTUAR PRODUCTO MATRICIAL
CALL MULTMA(A,B,M,N,L,X)
C   IMPRESION DE RESULTADOS
DO 2 I=1,M
2  WRITE(1,16) ((X(I,J),J=1,L),I=1,L)
   GO TO 1
C   FORMATO DE LECTURA E IMPRESION
10 FORMAT (3I5)
11 FORMAT(10,0)
12 FORMAT(40,3)
13 FORMAT(22,10,3,1)
14 FORMAT(4,3,5)
15 FORMAT(4,3,5)
END

```

Fig. 2.7 Listado del programa principal

```

C   SUBROUTINE MULTMA(A,B,M,N,L,X)
C   SUBROUTINA PARA MULTIPLICAR DOS MATRICES
C   EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C   A=MATRIZ PREMULTIPLICADORA DE ORDEN (M*N)
C   B=MATRIZ POSTMULTIPLICADORA DE ORDEN (N*L)
C   X=MATRIZ PRODUCTO
C
DIMENSION A(10,10),B(10,10),X(10,10)
DO 1 J=1,L
DO 1 I=1,M
X(I,J)=0.0
DO 1 K=1,N
X(I,J)=X(I,J)+A(I,K)*B(K,J)
RETURN
END

```

Fig. 2.8 Listado de la subrutina MULTMA

## 2.3.4 Ejemplo

Cuatro componentes de un automóvil requieren como materia prima de hule, aluminio y acero. Las unidades que se requieren de cada material para formar una unidad de cada componente del automóvil se proporcionan a continuación:

	hule	aluminio	acero
comp. 1	8	5	3
comp. 2	3	4	5
comp. 3	20	2	4
comp. 4	1	8	10

si los costos unitarios de cada material son:

	\$
hule	25.00
aluminio	30.00
acero	40.00

Determine el costo total de cada componente debido a la materia prima de que está compuesto.

## \*SOLUCION

TABLA 2.3 Datos para el problema del ejemplo 2.3.4

$N=4$

$M=3$

$L=1$

<u>A</u>	8	5	3
	3	4	5
	20	2	4
	1	8	10

<u>B</u>	25
	30
	40



TABLA 2.4 Resultados del problema del ejemplo 2.3.4

## MATRIZ A

.800E+01	.500E+01	.300E+01
.300E+01	.400E+01	.200E+01
.200E+01	.300E+01	.400E+01
.100E+01	.800E+01	.100E+02

## MATRIZ B

.250E+02
.300E+02
.400E+02

## MATRIZ PRODUCTO

.470E+03
.315E+03
.720E+03
.865E+03

### 2.4 Inversión de Matrices

#### 2.4.1 Objeto

Dada una matriz cuadrada  $A$  obtener su matriz inversa  $A^{-1}$ .

#### 2.4.2 Método

La matriz inversa de una matriz cuadrada  $A$  es otra matriz cuadrada que se representa por  $A^{-1}$  y que cumple la siguiente propiedad si la matriz  $A$  es de orden  $(n \times n)$ :

$$A A^{-1} = I_n = A^{-1} A \quad (2.15)$$

Se define a la matriz inversa como:

$$A^{-1} = \frac{A^*}{|A|} \quad (2.16)$$

donde  $A^*$  se conoce como la matriz adjunta de la matriz  $A$  y  $|A|$  representa el determinante de la matriz  $A$ .

De la ecuación (2.16) se infiere que para que exista la inversa de una matriz se requiere que  $|A| \neq 0$ , es decir, que la matriz sea no singular.

Para la obtención numérica de la matriz inversa es necesario acudir al método de Gauss-Jordan modificado. Esto se hace debido a que para obtener  $A^{-1}$  en una computadora digital mediante la ecuación (2.16) se requiere una gran cantidad de operaciones y consecuentemente de tiempo. Para obtener la inversa de una matriz  $(10 \times 10)$  se requieren más de 340 millones de operaciones con el método directo.

El método de Gauss-Jordan es un método de eliminación sistemática mediante el cual se transforma la matriz original  $A$  en una matriz identidad  $I_n$  y al mismo tiempo esta última se transforma en la matriz inversa  $A^{-1}$ , es decir, partiendo del arreglo:

$$\left[ \begin{array}{c|c} A_n & I_n \end{array} \right] \quad (2.17)$$

y aplicando algunas de las siguientes transformaciones al arreglo (2.17):

- intercambio de renglones,
  - multiplicación de un renglón por un escalar  $\lambda \neq 0$ ,
  - suma de equimúltiplos de un renglón a otro renglón.
- se llega al siguiente arreglo:

$$\left[ \begin{array}{c|c} I_n & A_n^{-1} \end{array} \right]$$

(2.18)

El método parte de la suposición de que  $A$  es una matriz no singular, lo cual implica que sus columnas son vectores linealmente independientes, en caso de no serlo el método lo puede detectar; en dicha situación se presenta que todos los elementos de un renglón de la matriz  $A$  o de sus matrices transformadas, son nulos.

A fin de minimizar los errores de redondeo, la eliminación de elementos se efectúa pivotando sobre los mayores elementos que quedan en la matriz  $A$  o en las matrices obtenidas a partir de esta última por transformaciones; debe tenerse cuidado de no emplear como pivotes elementos de renglones que ya hayan sido utilizados como pivotes.

#### 2.4.3 Descripción del Programa

##### a) Subrutinas requeridas:

SUBROUTINE MATINV(A,N,EPS,DET), obtiene la matriz inversa de la matriz  $A$ . El programa principal se emplea para la lectura de datos e impresión de resultados.

##### b) Descripción de las variables:

Para la subrutina MATINV:

- A(I,J) matriz de la que se buscará la inversa, durante el proceso se convierte en la matriz inversa.
- N orden de la matriz  $A$
- EPS criterio para determinar si el determinante de la matriz es nulo
- DET parámetro que indica si el determinante de la matriz es nulo
- C(I,J) matriz identidad que se emplea para obtener la matriz inversa
- HVR(I) ; HVC(I) contadores que indican cuáles renglones y cuáles columnas de la matriz  $A$  ya fueron empleadas como pivotes

RAMAX	mayor elemento de la matriz <u>A</u> o de sus transformaciones que se emplea como elemento pivote
TEMP	variable de localización temporal
Para el programa principal:	
A[I, J]	matriz de la que se busca la inversa, durante el proceso se convierte en la matriz inversa
N	orden de la matriz <u>A</u>
EPS	criterio para determinar si el determinante de la matriz <u>A</u> es nulo.
DET	variable que indica si el determinante de <u>A</u> es o no nulo

## c) Dimensiones:

La proposición DIMENSION del programa principal y de la subrutina deberá ser modificada cuando:

$$N > 10$$

## d) Formatos para los datos de entrada:

SEC.	TARJETAS	FORMATO	INFORMACION
1		{I5}	N
2		{8F10.0}	A[I, J], se proporcionan los elementos de la matriz renglón por renglón. Emplear tantas tarjetas como se requieran.

-----  
 otros paquetes de datos (opcional)  
 -----

TARJETA EN BLANCO, al finalizar toda la información.

## e) Diagrama de bloques:

Fig. 2.9 Diagrama de bloques del programa principal

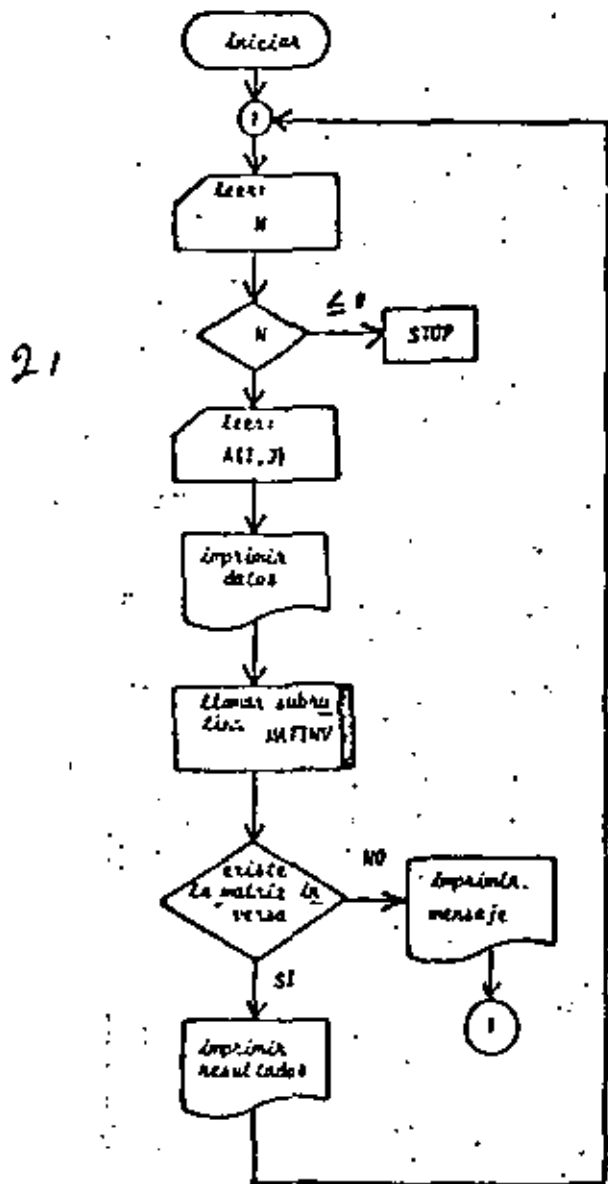
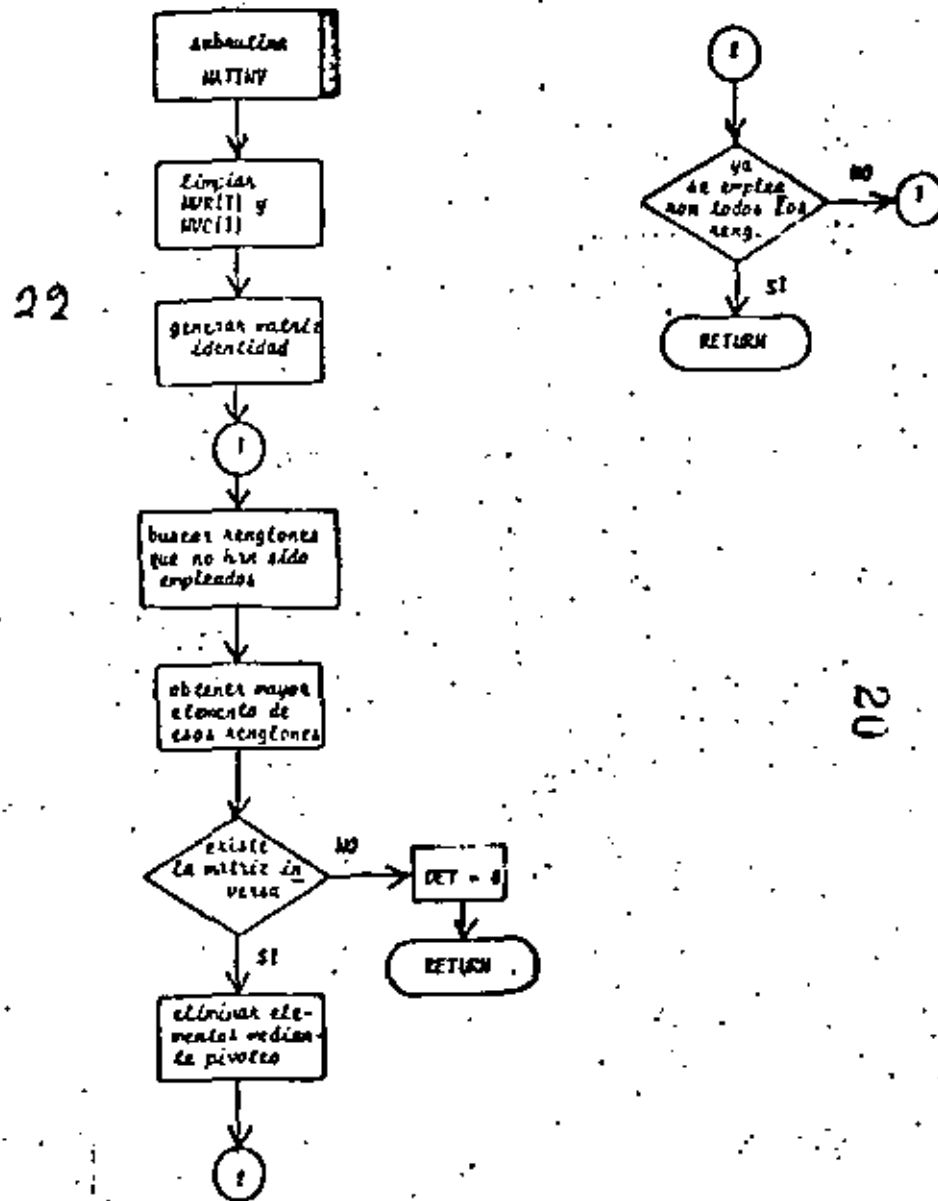


Fig. 2.10 Diagrama de bloques de la subrutina MATINV



## 6) Listado:

```

E   PROGRAMA PARA INVERTIR MATRICES POR EL METODO DE GAUSS-JORDAN
C   SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C   ORDEN DE LA MATRIZ A
C   MATRIZ DE LA QUE SE BUSCA SU INVERSA
C   EPS:CRITERIO PARA DETERMINAR SI EXISTE O NO LA INVERSA DE LA MATRIZ
C   DET:PARAMETRO QUE INDICA SI EXISTE O NO LA INVERSA DE LA MATRIZ

DIMENSION A(10,10),C(10,10)
INVS
I=0
EPS=0.00001
C   LECTURA DE DATOS
1 READ(19,19) M
IF (M) 2,3,3
2 CALL EXIT
3 DO 4 I=1,M
4 READ(19,20) ((A(I,J),J=1,M))
C   IMPRESION DE DATOS
WRITE(10,21)
DO 5 I=1,M
5 WRITE(10,22) ((A(I,J),J=1,M))
C   LLAMADO DE SUBROUTINA PARA OBTENER LA MATRIZ INVERSA
CALL MATINVER(A,M,EPS,DET)
IF (DET.CE.EPS) GO TO 7
WRITE(10,23)
GO TO 1
7 WRITE(10,24)
DO 8 I=1,M
8 WRITE(10,22) ((A(I,J),J=1,M))
GO TO 1
C   FORMATOS DE LECTURA E IMPRESION
19 FORMAT(15)
20 FORMAT(10.0)
21 FORMAT(//,5X,'MATRIZ A',//)
22 FORMAT(//,2X,10(E10.3,1X))
23 FORMAT(4X//,5X,'NO EXISTE LA MATRIZ INVERSA')
24 FORMAT(4X//,5X,'INVERSA DE LA MATRIZ A')
END

```

Fig. 2.11 Listado del programa principal

```

SUBROUTINE MATINVER(M,N,ACC,DETA,MYR,MYC)
C
C SOLUCION PARA OBTENER LA INVERSA DE UNA MATRIZ
C EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C MATRIZ A LA QUE SE BUSCA SU INVERSA Y QUE DURANTE EL PROCESO
C SE CONVIERTE EN LA MATRIZ INVERSA
C MORDEN DE LA MATRIZ
C EPS=CONSTANTE PARA DETERMINAR SI EL DETERMINANTE DE LA MATRIZ ES
C NULO
C LET=VALOR ABSOLUTO DEL DETERMINANTE DE LA MATRIZ
C CMATRIZ IDENTIDAD QUE SE UTILIZA PARA OBTENER LA MATRIZ INVERSA
C POR EL METODO DE GAUSS-JORDAN MODIFICADO
C MYR Y MYC=CONTADORES QUE INDICAN CULLES FILAS Y COLUMNAS YA
C FUERON UTILIZADAS COMO PIVOTES
C
C DIMENSION ACC(10),DETA(10),MYR(10),MYC(10)
C
C OBTENCION DE LA MATRIZ IDENTIDAD Y ACTUALIZACION DE VALORES PARA
C INICIAR EL PROCESO
C
DO 1 J=1,N
MYR(J)=0
MYC(J)=0
DO 4 I=1,N
DO 3 J=1,N
IF(I,CO,JI) GO TO 2
C(I,J)=0
GO TO 1
2 C(I,J)=1
3 CONTINUE
4 CONTINUE
C
C OBTENCION DE LA MATRIZ INVERSA
C
DO 12 M=1,N
RANX=0
LC=0
LR=0
DO 4 I=1,N
IF(MYR(I),EQ,1) GO TO 4
DO 3 J=1,N
IF(MYC(J),EQ,1) GO TO 3
IF(ABS(RANX)-ACC,AR5(A(I,J))) GO TO 3
RANX=A(I,J)
LR=I
LC=J
3 CONTINUE
4 DET=DETA*(RANX)
IF(DETA,LE,EPS) GO TO 14
IF(LR,EQ,LC) GO TO 5
DO 7 I=1,N
TEMP=A(LR,I)
A(LR,I)=A(LC,I)
A(LC,I)=TEMP
TEMP=A(LR,I)
A(LR,I)=A(LC,I)
A(LC,I)=TEMP
7 C(LC,3)=TEMP
8 DO 9 J=1,N
A(LC,3)=C(LC,3)+RANX*J
9 C(LC,3)=C(LC,3)+RANX*J
DO 11 I=1,N
IF(I,EQ,LC) GO TO 11
TEMP=A(LC,I)
DO 10 J=1,N
A(I,J)=A(I,J)-TEMP*A(LC,J)
10 C(I,J)=C(I,J)-TEMP*C(LC,J)
11 CONTINUE
MYR(LC)=1
MYC(LC)=1
12 CONTINUE
GO TO 13
DO 13 I=1,N
DO 11 J=1,N
13 ACC(J)=ACC(J)*DETA
14 RETURN
END

```

2.4.4 Ejemplo

Obtener la inversa de la matriz:

$$A = \begin{bmatrix} 10 & 2 & 3 & -1 \\ 1 & -20 & -1 & 3 \\ 1 & 1 & -10 & 2 \\ 2 & -1 & -1 & 30 \end{bmatrix}$$

\*SOLUCION

TABLA 2.5 Datos para el problema del ejemplo 2.4.4  
N=4

$$A = \begin{bmatrix} 10 & 2 & 3 & -1 \\ 1 & -20 & -1 & 3 \\ 1 & 1 & -10 & 2 \\ 2 & -1 & -1 & 30 \end{bmatrix}$$

TABLA 2.6 Resultados del problema del ejemplo 2.4.4

MATRIZ A

1.00E+02	2.00E+01	3.00E+01	-1.00E+01
1.00E+01	-2.00E+02	-1.00E+01	3.00E+01
1.00E+01	1.00E+01	-1.00E+02	2.00E+01
2.00E+01	-1.00E+01	-1.00E+01	3.00E+02

INVERSA DE LA MATRIZ A

9.61E-01	1.12E-01	2.77E-01	2.53E-03
3.42E-02	-4.76E-01	5.53E-02	4.71E-02
6.76E-02	-4.97E-02	-4.77E-01	7.24E-02
4.89E-02	-2.53E-02	-4.72E-02	3.37E-01

1. CARMANAN B., LUTHER W., WILKES J., "Applied Numerical Methods". New York: John Wiley and Sons Inc., 1969. pp. 210-218, 287-296.
2. HADLEY G., "Algebra Lineal". Bogotá: Fondo Educativo Interamericano, 1969. pp. 60-131.
3. HANWING Richard, "Numerical Methods for Scientists and Engineers". New York: Mc Graw Hill Book Co., 1968. pp. 366-367.
4. JOHNSTON J., BAILEY PRICE G., VAN VLECK F., "Linear Equations and Matrices". Reading Mass.: Addison-Wesley Co., 1966. pp. 95-157.
5. KAPLAN Lewis, "Calculus and Linear Algebra Vol. 2". New York: John Wiley and Sons Inc., 1971. pp. 718-803.
6. KUO S. SHAN, "Computer Applications of Numerical Methods". Reading Mass.: Addison-Wesley Co., 1971. pp. 178-179, 189-196.

## 3. SOLUCIÓN DE SISTEMAS DE ECUACIONES LINEALES

## 3.1 Introducción

Por sistemas de ecuaciones lineales se entiende un grupo de ecuaciones que presentan la siguiente estructura:

$$\left. \begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n &= b_m \end{aligned} \right\} \quad (3.1)$$

donde  $a_{ij}$  y  $b_i$  son constantes y las incógnitas del sistema son los valores  $x_i$ , donde  $1 \leq i \leq n$ .

Dichos sistemas se pueden representar en la forma:

$$\underline{A} \underline{x} = \underline{B} \quad (3.2)$$

donde  $\underline{A}$  se conoce como la matriz de coeficientes del sistema,  $\underline{B}$  como vector de términos independientes y  $\underline{x}$  como vector de incógnitas.

Si el vector de términos independientes es diferente de cero se habla de sistemas de ecuaciones no homogéneas y en caso contrario de sistemas homogéneos.

Antes de proceder a resolver un sistema de ecuaciones es necesario determinar si dicho sistema tiene solución y en caso de tenerla, cuántas posibles soluciones tiene. En base a lo anterior se tiene la siguiente clasificación:

$$\text{Sistema de ecuaciones lineales} \left\{ \begin{array}{l} \text{no homogéneo} \left\{ \begin{array}{l} \text{compatible} \left\{ \begin{array}{l} \text{determinado} \\ \text{indeterminado} \end{array} \right. \\ \text{incompatible} \end{array} \right. \\ \\ \text{homogéneo} \left\{ \begin{array}{l} \text{compatible} \left\{ \begin{array}{l} \text{determinado} \\ \text{(Sol. trivial)} \\ \text{indeterminado} \end{array} \right. \end{array} \right. \end{array} \right.$$



Sistema compatible es aquél que sí tiene solución y para que esto se cumpla se requiere:

$$\text{rango } [A] = \text{rango } [A|B] \quad (3.3)$$

donde a la matriz  $[A|B]$  se le conoce como la matriz ampliada del sistema.

Sistema incompatible es aquél que no tiene solución y se cumple que:

$$\text{rango } [A] < \text{rango } [A|B] \quad (3.4)$$

Sistema determinado es un sistema compatible que presenta solución única y se verifica que:

$$\text{rango } [A] = \text{número de incógnitas} \quad (3.5)$$

Cuando se presenta esta situación en sistemas homogéneos se habla de solución trivial, ya que  $\underline{x} = \underline{0}$ .

Un sistema compatible que presenta infinitud de soluciones se conoce como sistema indeterminado y se caracteriza por:

$$\text{rango } [A] < \text{número de incógnitas} \quad (3.6)$$

Para la solución de sistemas de ecuaciones lineales exist en diversos métodos de los cuales solo se tratarán: Método de Gauss-Jordan modificado y el Método de Gauss-Seidel.

### 3.2 Método de Gauss-Jordan modificado.

#### 3.2.1 Objeto

Obtener la solución de sistemas de ecuaciones lineales de la forma:

$$\underline{A} \underline{x} = \underline{B} \quad (3.7)$$

#### 3.2.1 Método

Dado el sistema de ecuaciones:

$$\underline{A} \underline{x} = \underline{B} \quad (3.8)$$

\*  $\text{rango } [A]$  es la cantidad de vectores linealmente independientes del conjunto de vectores columna que forman la matriz  $\underline{A}$ .

el método consiste en trabajar con la matriz de coeficientes y el vector de términos independientes, es decir, con la matriz ampliada del sistema:

$$\left[ \underline{A} | \underline{B} \right] \quad (3.9)$$

A dicha matriz se le aplican una serie de transformaciones que conducen a obtener otra matriz ampliada equivalente:

$$\left[ \underline{I}_n | \underline{C} \right] \quad (3.10)$$

donde  $\underline{C}$  representa la solución de cada una de las incógnitas del sistema.

El proceso equivale a premultiplicar la ecuación (3.9) - por  $\underline{A}^{-1}$ , es decir, el método de la matriz inversa, solo que este método consiste en una eliminación sistemática de valores.

La transformación de la matriz (3.9) en la matriz (3.10) se efectúa basándose en tres operaciones que no alteran el sistema de ecuaciones sino que proporcionan sistemas de ecuaciones equivalentes, ellas son:

- Intercambio de dos renglones, lo cual equivale a intercambiar dos ecuaciones.
- multiplicación de un renglón por un escalar diferente de cero, lo cual equivale a multiplicar ambos miembros de una ecuación por la misma constante.
- suma de equimúltiplos de un renglón a otro renglón, es decir, multiplicar una ecuación por una constante "k" y sumarla a otra ecuación.

Para aplicar las operaciones anteriores se procede en la siguiente forma:

- ① Seleccionar un renglón pivote y un elemento pivote dentro de dicho renglón.
- ② Normalizar el elemento pivote, es decir, convertirlo en unitario.
- ③ Cancelar elementos que se encuentren en la columna arriba y/o abajo del elemento pivote mediante la suma de equimúltiplos.
- ④ Regresar al paso ① y así sucesivamente hasta que se transforma la matriz de coeficientes  $\underline{A}$  en una matriz --

Identidad  $I_n$ .

Debido a que durante el proceso se presentan errores por redondeo, la forma óptima de escoger los elementos pivote es seleccionando el mayor elemento que quede en la matriz  $A$  o en sus transformaciones. Hay que tener presente que los elementos de un renglón que ya fue seleccionado como línea pivote no se pueden usar como pivotes, aún cuando el mayor elemento quede colocado en dicho renglón.

Al seleccionar los pivotes en la forma antes mencionada el error se reduce al mínimo y, debido a que puede quedar una matriz no identidad al término de las iteraciones, es necesario efectuar un intercambio de líneas hasta obtener  $I_n$ .

Cabe mencionar que el presente método es un método directo de solución que no requiere que se determine con anterioridad si el sistema es compatible y determinado, el método durante el proceso proporciona dicha información.

Si el sistema es compatible y determinado, el procedimiento descrito se puede llevar a cabo sin contratiempos hasta llegar a  $\begin{bmatrix} I_n & C \end{bmatrix}$ .

Si el sistema es compatible pero indeterminado, la matriz ampliada adquirirá la configuración:

$$\left[ \begin{array}{ccc|c} 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 \end{array} \right] \quad (3.11)$$

es decir, un renglón será nulo; en esta situación se obtienen las ecuaciones independientes que restan en el sistema y se aplica la metodología correspondiente a sistemas indeterminados.

Si el sistema es incompatible, se presentará lo siguiente:

$$\left[ \begin{array}{ccc|c} 1 & 1 & 2 & 1 \\ 0 & 2 & 3 & 2 \\ 0 & 0 & 0 & \lambda \neq 0 \end{array} \right] \quad (3.12)$$

o sea,  $0 = \lambda \neq 0$ , lo cual es una contradicción.

### 3.2.3 Descripción del Programa

a) Subrutinas requeridas:

SUBROUTINE GAUTOR (A, B, M, EPS, DET), esta subrutina obtiene la solución del sistema de ecuaciones por el método de Gauss-Jordan modificado, el programa principal solo sirve para entrada y salida de datos.

b) Descripción de las variables:

Para la subrutina GAUTOR:

A(I, J) matriz de coeficientes del sistema de ecuaciones.

B(I) vector de términos independientes del sistema de ecuaciones, durante el proceso se transforma en la solución.

N orden del sistema de ecuaciones.

RAMAX mayor elemento de la matriz A que se emplea como pivote.

MVR(I) y MVC(I) contadores que indican qué renglón y columna ya fueron empleados.

EPS criterio para determinar si el determinante de la matriz A es nulo.

DET parámetro que indica si el determinante de A es nulo.

IR y IC indicadores del renglón y columna que se utilizan.

TEMP variable de localización temporal.

Para el programa principal:

A(I, J) matriz de coeficientes del sistema de ecuaciones.

B(I) vector de términos independientes.

N orden del sistema de ecuaciones.

EPS criterio para determinar si el determinante de A es nulo.

DET parámetro que indica si el determinante de A es nulo.

c) Dimensiones:

La proposición DIMENSION del programa principal y de la subrutina se deberán modificar en el caso de que:

$$-N > 10$$

d) Formatos para los datos de entrada:

SEC. TARJETAS	FORMATO	INFORMACION
1	(I5)	N
2	(8F10.0)	A(I,J), se dan los elementos de <u>A</u> renglón por renglón, empleando tantas tarjetas como sean necesarias para cada renglón.
3	(8F10.0)	B(I), el vector de términos independientes se da en una tarjeta o más según la cantidad de elementos.

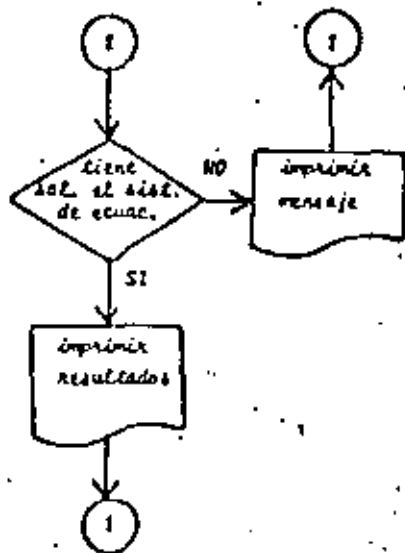
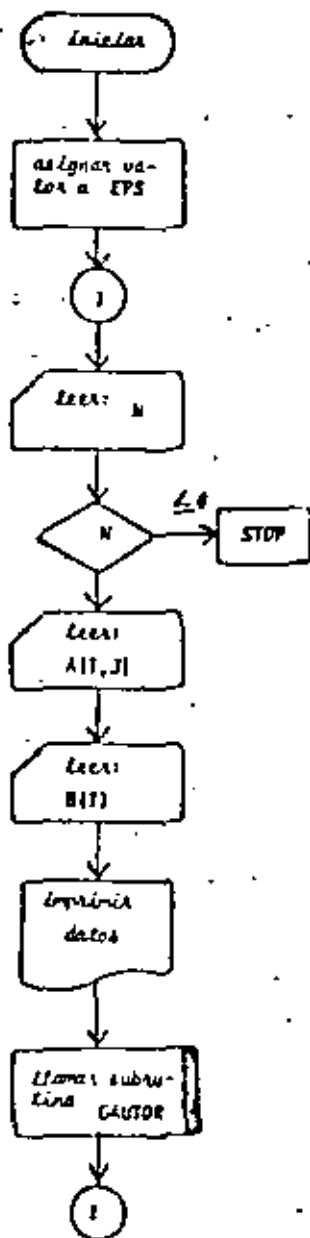
-----  
 otros paquetes de datos (opcional)  
 -----

n TARJETA EN BLANCO, al finalizar toda la información.

e) Diagrama de bloques:

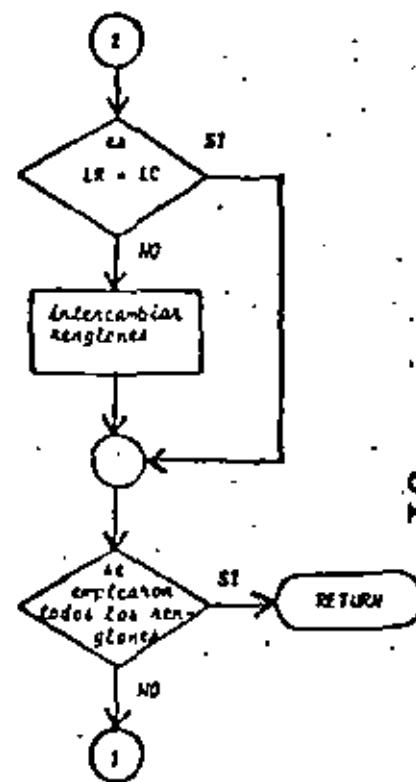
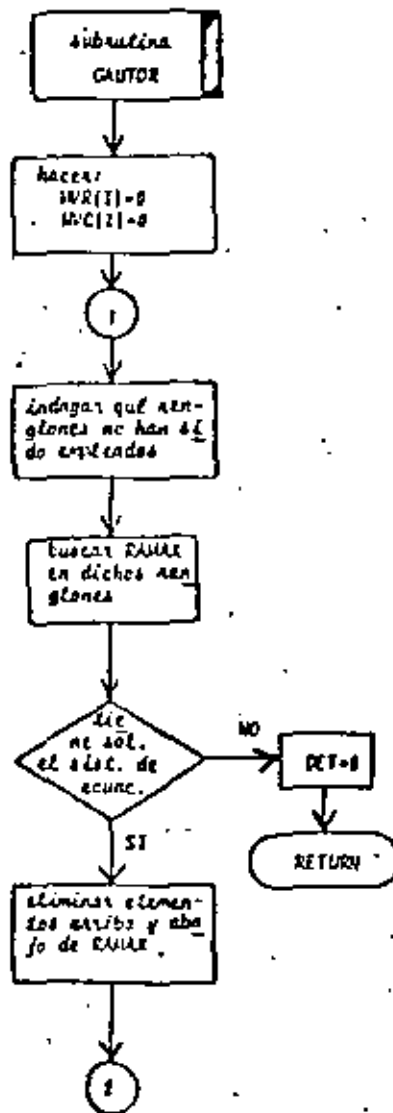
33

Fig. 3.1 Diagrama de bloques del programa principal.



34

Fig. 3.2 Diagrama de bloques de la subrutina GAUTOR.



31

6) Listado:

```

C   PROGRAMA PARA RESOLVER SISTEMAS DE ECUACIONES LINEALES POR EL METO
C   DO DE GAUSS-JORDAN
C   SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C   NRODEM DEL SISTEMA DE ECUACIONES
C   MATRIZ DE COEFICIENTES DEL SISTEMA DE ECUACIONES
C   VECTORES DE TERMINOS INDEPENDIENTES, SE CONVIERTE EN LA SOLUCION
C   CRITERIO PARA DETERMINAR SI EL DETERMINANTE DE A ES DIFERENTE DE 0
C   DET=VARIABLE QUE INDICA SI EL SISTEMA TIENE O NO SOLUCION

DIMENSION A(10,10),B(10)
IA=5
IN=6
EPS=0.00001
C   LECTURA DE DATOS
1  READ(1,20) N
   IF(N) 2,2,3
2  CALL EXIT
3  GO 4 1*100
4  READ(1,21) ((A(I,J)),J=1,N)
   READ(1,21) (B(I)),I=1,N
C   IMPRESION DE DATOS
5  WRITE(1,22)
   GO 3 1*100
6  WRITE(1,23) (A(I,J),J=1,N),B(I)
C   LLAMADO DE SUBROUTINA PARA RESOLVER EL SISTEMA DE ECUACIONES
   CALL GALTOR(A,B,N,EPS,CET)
C   IMPRESION DE RESULTADOS
7  WRITE(1,24)
   GO 4 1*100
8  WRITE(1,25) B(I)
   GO TO 1
9  WRITE(1,26)
   GO TO 1
C   FORMATS DE LECTURA E IMPRESION
20 FORMAT(I5)
21 FORMAT(2F10.0)
22 FORMAT(A(//),5X,'EL SISTEMA DE ECUACIONES ES',//)
23 FORMAT(//,2X,'(C(10,3,1))')
24 FORMAT(A(//),5X,'LA SOLUCION DEL SISTEMA DE ECUACIONES ES',//,5X,'(I
1',5X,'(E11)',//)
25 FORMAT(//,5X,'(2,4)E12.5)')
26 FORMAT(A(//),5X,'EL SISTEMA DE ECUACIONES NO TIENE SOLUCION')
END

```

Fig. 3.3 Listado del programa principal

```

SUBROUTINE GAUTOR(N,NB,CP,DET)
C
C SUBROUTINA PARA RESOLVER UN SISTEMA DE ECUACIONES POR EL METODO DE
C GAUSS-JORDAN MODIFICADO
C EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C MATRIZ DE COEFICIENTES DEL SISTEMA DE ECUACIONES
C VECTORES DE TERMINOS INDEPENDIENTES QUE DURANTE EL PROCESO SE
C TRANSFORMA EN LA SOLUCION DEL SISTEMA DE ECUACIONES
C ORDEN DEL SISTEMA DE ECUACIONES
C MAXIMUM ELEMENT OF LA MATRIZ A QUE SE USA COMO PIVOTE
C NVR Y NVC CONTACTORES QUE INDICAN QUE RENGLON Y QUE COLUMNA YA FUERON
C USADOS
C EPS CRITERIO PARA DETERMINAR SI EL DETERMINANTE DE LA MATRIZ A ES
C CERO
C DET VALOR ABSOLUTO DEL DETERMINANTE DE LA MATRIZ A
C
C DIMENSION A(10,10),B(10),NVR(10),NVC(10)
C
C ACTUALIZACION DE VALORES PARA INICIAR EL PROCESO
C
C DO 1 I=1,N
C NVR(I)=0
C NVC(I)=0
C
C SOLUCION DEL SISTEMA DE ECUACIONES
C
C DO 9 I=1,N
C RANX=0.0
C LC=0
C LR=0
C DO 3 J=1,N
C IF(NVR(I),EQ,1) GO TO 3
C DO 2 J=1,N
C IF(NVC(J),EQ,J) GO TO 3
C IF((NSCRANX).GE.ABS(A(I,J))) GO TO 2
C RANX=A(I,J)
C LR=I
C LC=J
C
C CONTINUE
C CONTINUE
C DET=ABS(RANX)
C IF(DET.LE.EPS) GO TO 10
C IF(LR.EQ.LC) GO TO 5
C DO 4 I=1,N
C TEMP=A(LR,I)
C A(LR,I)=A(LC,I)
C A(LC,I)=TEMP
C TEMP=A(LR)
C A(LR)=A(LC)
C A(LC)=TEMP
C
C DO 6 I=1,N
C A(LC,I)=A(LC,I)/RANX
C B(LC)=B(LC)/RANX
C DO 8 I=1,N
C IF(I.EQ.LC) GO TO 8
C TEMP=A(I,LC)
C B(I+B(I)) = TEMP*B(LC)
C
C DO 7 J=1,N
C A(I,J)=A(I,J) + TEMP*A(LC,J)
C
C CONTINUE
C NVR(LC)=LC
C NVC(LC)=LC
C CONTINUE
C RETURN
C END

```

Fig. 3.4 Listado de la subrutina GAUTOR

## 3.1.4 Ejemplo

Empleando las leyes de Kirchoff (ver referencia 2), se obtuvieron las siguientes ecuaciones lineales para el circuito mostrado en la figura 3.3:

$$\begin{aligned} i_1 - i_4 - i_A &= 0 \\ i_4 + i_5 + i_A - i_7 - i_3 &= 0 \\ i_1 - i_2 - i_B &= 0 \\ i_2 + i_B + i_3 + i_6 - i_7 &= 0 \\ i_C - i_8 - i_5 - i_4 - i_9 &= 0 \\ R_1 i_1 + R_2 i_2 - R_3 i_3 &= 0 \\ R_4 i_4 - R_5 i_5 + R_8 i_8 &= 0 \\ R_5 i_5 + R_6 i_3 - R_6 i_6 &= 0 \\ R_6 i_6 + R_7 i_7 - R_9 i_9 &= 0 \end{aligned}$$

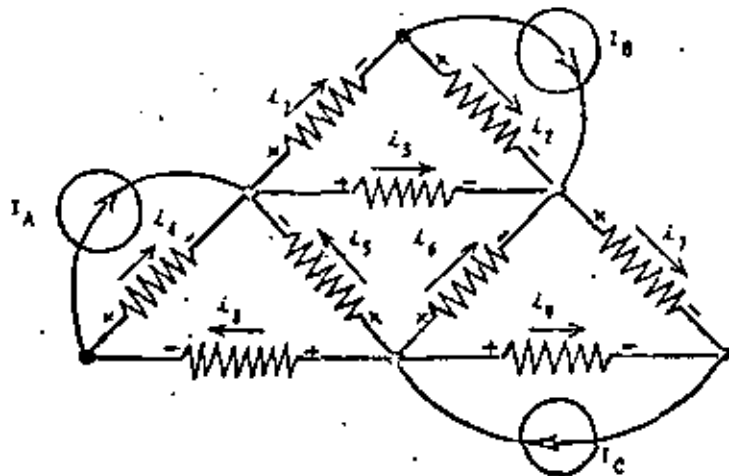


Fig. 3.3 Circuito del ejemplo 3.1.4

Si el valor de las fuentes es  $I_A = 2A$ ,  $I_B = 6A$ ,  $I_C = 4A$  y el de las resistencias:

$$R_1 = R_2 = 1 \Omega$$

$$R_4 = R_8 = 3 \Omega$$

$$R_5 = R_6 = 5 \Omega$$

$$R_7 = R_9 = 4 \Omega$$

$$R_3 = 6 \Omega$$

Obtenga las corrientes de rama  $i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9$ .

\* SOLUCION

TABLA 3.1 Datos para el problema del ejemplo 3.1.4  
M = 9

$$A = \begin{bmatrix} 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 2 & 2 & -6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & -5 & 0 & 0 & 3 & 0 \\ 0 & 0 & 6 & 0 & 5 & -5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 & 4 & 0 & -4 \end{bmatrix}$$

$$B = \begin{bmatrix} 2 \\ -2 \\ 6 \\ -6 \\ 4 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

TABLA 3.2 Resultados del problema del ejemplo 3.2.4

EL SISTEMA DE ECUACIONES ES

0.	C.	0.	-.100E+01	0.	0.	0.	-.100E+01	0.	-.200E+01
-.100E+01	C.	-.100E+01	.100E+01	.100E+01	0.	0.	0.	0.	-.200E+01
.100E+01	-.100E+01	0.	0.	0.	0.	0.	0.	0.	.100E+01
0.	.100E+01	.100E+01	0.	0.	.100E+01	-.100E+01	0.	0.	-.100E+01
0.	0.	0.	0.	.100E+01	.100E+01	0.	-.100E+01	.100E+01	.100E+01
-.200E+01	-.200E+01	-.400E+01	0.	0.	0.	0.	0.	0.	0.
0.	0.	0.	.100E+01	-.500E+01	0.	0.	-.200E+01	0.	0.
0.	0.	.400E+01	0.	.500E+01	-.500E+01	0.	0.	0.	0.
0.	0.	0.	0.	0.	.500E+01	.100E+01	0.	-.400E+01	0.

LA SOLUCION DEL SISTEMA DE ECUACIONES ES

- 1     .23741E+01
- 2     -.10239E+01
- 3     -.41594E+00
- 4     -.56159E+00
- 5     .92173E+00
- 6     .24940E+01
- 7     .19047E+01
- 8     .14364E+01
- 9     .20153E+01



5.5 Método de Gauss-Seidel

3.3.1 Objeto

Obtener la solución de sistemas de ecuaciones lineales --

con la configuración

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m$$

empleando el método de Gauss-Seidel.

3.3.1 Método

El método de Gauss-Seidel es un método de tipo iterativo

que sirve para la solución de sistemas de ecuaciones lineales

del tipo:

$$\bar{A} \bar{x} = \bar{b}$$

[3.14]

cuando los valores numéricos de los elementos de la diagonal

principal son mayores que los demás de su correspondiente sem-

blen.

Para asegurar la convergencia del método se requiere que:

a) los elementos no nulos de la matriz de coeficientes  $\{\bar{A}\}$

se acumulen en la diagonal principal.

b) los elementos de la diagonal principal de la matriz de

coeficientes  $\{\bar{A}\}$  sean mayores en valor absoluto que la

suma de los valores absolutos de los términos --

restantes del renglón correspondiente, es decir:

$$|a_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}| \quad i = 1, 2, \dots, n$$

[3.15]

Para aplicar el método se procede a despejar una incógnita

En de cada ecuación del arreglo (3.13), se despeja la

incógnita  $x_i$  de la "i-ésima" ecuación, o sea:

$$x_1 = \frac{a_{11}}{b_1} - a_{12}x_2 - a_{13}x_3 - \dots - a_{1n}x_n$$

$$x_2 = \frac{a_{22}}{b_2} - a_{21}x_1 - a_{23}x_3 - \dots - a_{2n}x_n$$

$$x_n = \frac{a_{nn}}{b_n} - a_{n1}x_1 - a_{n2}x_2 - \dots - a_{n,n-1}x_{n-1}$$

se establecen las siguientes ecuaciones iterativas:

$$x_1^{(k+1)} = \frac{a_{11}}{b_1} - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \dots - a_{1n}x_n^{(k)}$$

$$x_2^{(k+1)} = \frac{a_{22}}{b_2} - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \dots - a_{2n}x_n^{(k)}$$

[3.17]

$$x_n^{(k+1)} = \frac{a_{nn}}{b_n} - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - \dots - a_{n,n-1}x_{n-1}^{(k+1)}$$

donde  $x_i^{(k)}$  indica el valor de la "i-ésima" incógnita en la

iteración "k".

Para asegurar el método se establece una solución ini-

cial  $\bar{x}_0$ .

$$\begin{bmatrix} x_1^0 \\ x_2^0 \\ \vdots \\ x_n^0 \end{bmatrix}$$

[3.18]

dichos valores se sustituyen en el lado derecho de la ecuación (3.17) para obtener la siguiente solución aproximada:

$$\underline{x}_1 = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ \vdots \\ x_n^{(1)} \end{bmatrix} \quad (3.19)$$

y así sucesivamente hasta que

$$\left| x_{n+1} - x_n \right| < \underline{\epsilon} \quad (3.20)$$

Para poder emplear este método es necesario verificar con anterioridad que el sistema sea compatible y determinado; además de que cumpla con las condiciones de convergencia del método. Afortunadamente la mayoría de los problemas de tipo ingenieril cumplen los requisitos mencionados.

Ciertos sistemas que a primera vista no cumplen los requisitos del método pueden llenar los requisitos mediante un simple intercambio en la posición de las ecuaciones.

### 3.3.3 Descripción del programa

#### a) Subrutinas requeridas:

Ninguna.

#### b) Descripción de las variables.

A(I,J)	matriz de coeficientes del sistema
B(I)	vector de términos independientes
N	orden del sistema de ecuaciones
X(I)	valor inicial de las incógnitas del sistema y variable de localización temporal
V(I)	valor de las incógnitas en la iteración "n"
XN(I)	valor de las incógnitas en la iteración "n + 1"

N	máximo número de iteraciones a efectuar
E	criterio de convergencia
NCOM	contador de iteraciones efectuadas
SUM	sumador

#### c) Dimensiones:

La proposición DIMENSION deberá modificarse cuando se presente el caso de que  $N > 10$ .

#### d) Formatos para los datos de entrada:

SEC. TARJETAS	FORMATO	INFORMACION
1	(I15,F10.0)N, N, E	
2	(10F8.0)	A(I,J), los elementos de la matriz A se dan renglón por renglón empleando la cantidad de tarjetas necesaria para cada renglón.
3	(10F8.0)	B(I) el vector de términos independientes se da en una tarjeta o más según el orden del sistema.
4	(10F8.0)	X(I), la solución para arrancar el método se da en una tarjeta o más según sea el tamaño de N.

-----  
 otros paquetes de datos (opcional)  
 -----

n TARJETA EN BLANCO, al finalizar toda la información.

#### e) Diagrama de bloques:

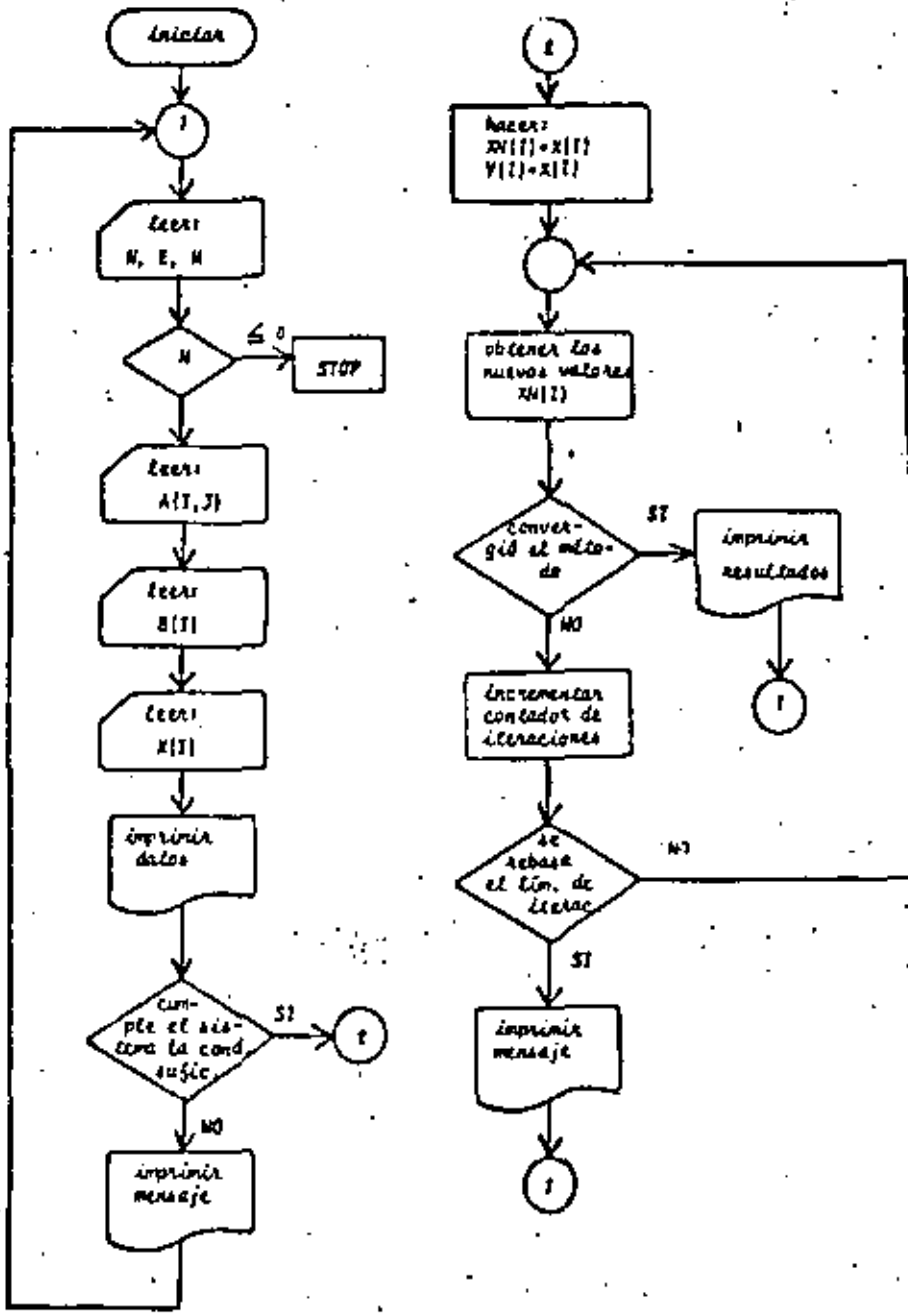


Fig. 3.6 Diagrama de bloques para el programa

## 6) Listado:

```

C   PROGRAMA PARA RESOLVER SISTEMAS DE ECUACIONES POR EL METODO DE
C   GAUSS-JORDAN
C   SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C   MATRIZ DE COEFICIENTES DEL SISTEMA DE ECUACIONES
C   DIFERENCIA DE TERMINOS INDEPENDIENTES
C   VALOR INICIAL DE LA SOLUCION DEL SISTEMA
C   TRANSFORMACION DEL SISTEMA DE ECUACIONES EN LA SIGUIENTE ITERACION
C   N° ORDEN DEL SISTEMA
C   VALOR DE LA SOLUCION DEL SISTEMA DE ECUACIONES EN LA ITERACION
C   ANTERIOR
C   MÁXIMO NÚMERO DE ITERACIONES
C   CRITERIO DE CONVERGENCIA

      DIMENSION A(20,20),B(20),X(20),Y(20),XN(20)
      LECTURA DE DATOS
      1 READ(5,200) N,M,N1
      IF(N) 2,2,3
      2 CALL EXIT
      3 GO 4 1,1,1,N
      4 READ(5,300) (A(I,J),J=1,M)
      READ(5,300) (B(I),I=1,M)
      READ(5,300) (X(I),I=1,M)
      5 IMPRESION DE DATOS
      WRITE(6,400)
      GO 5 1,1,1,N
      6 WRITE(6,500) (A(I,J),J=1,M),B(I)
      WRITE(6,600) (X(I),I=1,M)
      7 SE INDAGA SI EL SISTEMA CUMPLE LA CONDICION SUFICIENTE DE CONVER-
      GENCIA
      GO 7 1,1,1,N
      GO 8 1,1,1,N
      8 IF(ABS(A(1,1)) * ABS(A(1,2))) >= 0.0001
      9 CONTINUE
      7 CONTINUE
      GO TO 9
      8 WRITE(6,700) I,J,1,1
      GO TO 1
      9 OBTENCION DEL VALOR DE LAS INCOGNITAS
      NCON=1
      GO 10 1,1,1,N
      XN(I)=X(I)
      10 Y(I)=X(I)
      11 GO 14 1,1,1,N
      SUM=0.
      GO 13 1,1,1,N
      IF(N=1) 12,13,12
      12 SUM=SUM + A(N,1)*XN(I)
      13 CONTINUE
      XN(I)=(B(I)-SUM)/A(N,I)
      14 CONTINUE
      GO 15 1,1,1,N
      15 SE VERIFICA SI YA CONVERGIO EL METODO
      IF(ABS(XN(I)-Y(I))) <= 1E-16,16,16
      16 CONTINUE
      17 IMPRESION DE RESULTADOS
      WRITE(6,800) (XN(I),I=1,M)
      WRITE(6,900) NCON
      GO TO 1
      18 NCON=NCON + 1
      IF(NCON >= 10,17,17)
      17 WRITE(6,900) (XN(I),I=1,M)
      WRITE(6,950) NCON
      GO TO 1
      19 GO 14 1,1,1,N
      19 Y(I)=XN(I)
      GO TO 11
      20 FORMATOS DE LECTURA E IMPRESION
      200 FORMAT (2I5,F10.0)
      300 FORMAT (10I5,/)
      400 FORMAT(1M,5I5,15F10.5,'MATRIZ A=')
      500 FORMAT (1I5,10F10.5,/)
      600 FORMAT(4F10.5,12F10.5,'PRIMERA APROXIMACION DE LA SOLUCION',/,10X,10F10.5,/,2,2,1)
      700 FORMAT(4F10.5,15F10.5,'EL METODO PUEDE NO CONVERGER DADO QUE',/,15X,10F10.5,/,2,2,1,12,1) ES MAYOR DE 10,12,1,1,12,1)
      800 FORMAT(4F10.5,15F10.5,'LA SOLUCION DEL SISTEMA ES',/,5X,10F10.5,1,1)
      900 FORMAT(4F10.5,12F10.5,'NO SE LLEGA A LA SOLUCION',/,12X,10F10.5,1,1)
      950 FORMAT(4F10.5,12F10.5,'ITERACIONES REALIZADAS', 1,1)
      END

```

Fig. 3.7 Listado del programa

## 3.3.4 Ejemplo

Para el circuito eléctrico de la fig. 3.3 se sabe que ---  
 $I_1 = 1A$  e  $I_2 = 2A$ ,  $R_1 = R_2 = R_3 = R_4 = R_5 = R_6 = 1 \Omega$ .

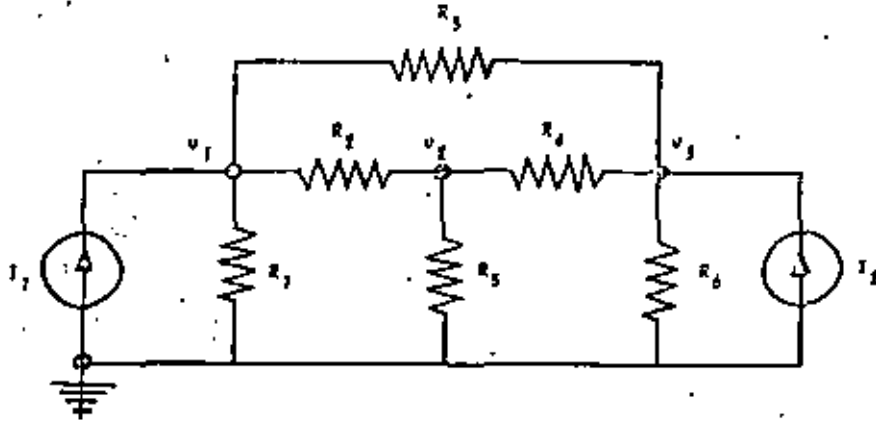


Fig. 3.3 Circuito eléctrico del problema del ejemplo 3.3.4

Se desea obtener el voltaje de los nodos  $v_1$ ,  $v_2$  y  $v_3$ .  
 Aplicando análisis nodal al circuito se obtiene:

$$3V_1 - V_2 - V_3 = 1$$

$$-V_1 + 3V_2 - V_3 = 0$$

$$-V_1 - V_2 + 3V_3 = 2$$

ejemplo que es un sistema de ecuaciones lineales con todas las características propias para aplicar el método de Gauss-Seidel.

Se seleccionará como solución inicial el siguiente vector:

$$\begin{bmatrix} V_1^* \\ V_2^* \\ V_3^* \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

\* SOLUCIÓN

TABLA 3.3 Datos del problema del ejemplo 3.3.4

$N = 3$   
 $M = 50$   
 $EPS = 0.0001$

$$A = \begin{bmatrix} 3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$$

$$X = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

TABLA 3.4 Resultados del problema del ejemplo 3.3.4

PASOS REALIZADOS			
1.000	-1.000	2.000	1.000
-1.000	3.000	-1.000	0.000
-1.000	-1.000	3.000	0.000

PRIMERA APROXIMACION DE LA SOLUCION

0.50 0.50 0.50

LA SOLUCION DEL SISTEMA ES

1.000000E+01 1.700000E+00 1.500000E+01

Ecuaciones Resueltas 40

## 3.4 Bibliografía

1. CARNAHAN B., LUTHER H., WILKES J., "Applied Numerical Methods". New York: John Wiley and Sons Inc., 1969. pp. 269-307.
2. GEREZ G. Vélaz, MURRAY LASSO M.A., "Teoría de Sistemas y Circuitos". México: Representaciones y Servicios de Ingeniería, S.A., 1972. pp. 99-123.
3. HADLEY G. "Algebra Lineal"; Bogalá: Fondo Educativo - Interamericano, 1969. pp. 162-187.
4. HAMMING Richard, "Numerical Methods for Scientists -- and Engineers". New York: Mc Graw Hill Book Co., 1962. pp. 360-365.
5. JAMES H., SMITH G., WOLFORD J., "Applied Numerical -- Methods for Digital Computation with FORTRAN". -- Scranton Penn: International Text Book Co., 1967. pp. 224-238.
6. JOHN STON J., BAILEY PRICE G., VAN VLECK F., "Linear - Equations and Matrices", Reading Mass.: Addison --- Wesley Co., 1966. pp. 1-94.
7. KUO S. Shan, "Computer Applications of Numerical --- Methods". Reading Mass.: Addison Wesley Co., 1972. pp. 178-212.
8. OLIVERA S. Antonio, "Apuntes de Métodos Numéricos". - México: Facultad de Ingeniería, UNAM, 1972. pp. 4.1-4.36.

## 7. APROXIMACION POLINOMIAL Y COEFICIENTE DE CORRELACION

## 7.1 Introducción

En muchas ocasiones a partir de una serie de valores muestrales, donde existe una variable dependiente y una o varias -- variables independientes, es necesario ajustar dichos puntos -- por una curva tal que permita determinar el valor de la variable dependiente para cualquier valor de las variables independientes. La curva de ajuste por el método de los mínimos cuadrados puede ser un polinomio de grado "n", una función de tipo logarítmico, etc.; dicha curva se escoge de acuerdo a la distribución de los puntos muestrales y en forma tal que se minimice la suma de los cuadrados de los errores. En procesos estadísticos a tal tipo de ajuste se le denomina regresión simple o múltiple de la variable dependiente sobre las variables independientes. El grado de relación existente entre la variable dependiente y la independiente se denomina correlación y a la medida de tal relación se le llama coeficiente de correlación, el cual se suele denotar con el símbolo  $\rho$  ó  $r$  (12.3...n). Donde:

$$\rho = \sqrt{\frac{\text{Variación explicada}}{\text{Variación total}}} \quad (7.1)$$

Si se considera a Y como la variable dependiente, las -- variaciones se definen en la siguiente forma:

$$\text{variación total} = \sum (Y - \bar{Y})^2 \quad (7.2)$$

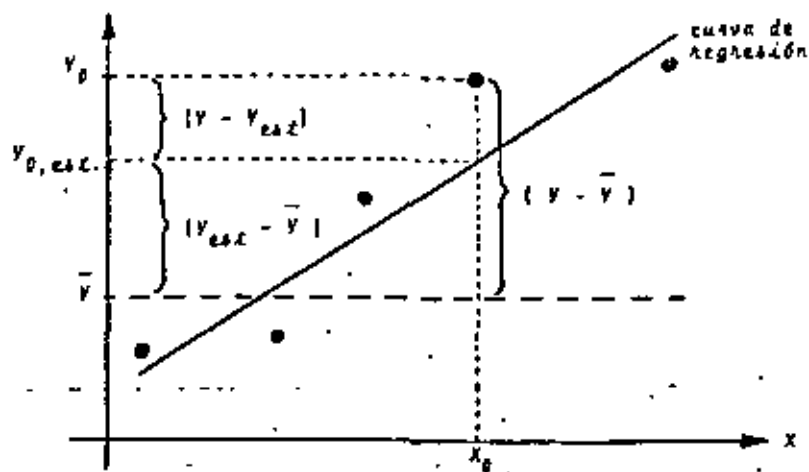
$$\text{variación explicada} = \sum (Y_{est} - \bar{Y})^2 \quad (7.3)$$

$$\text{variación no explicada} = \sum (Y - Y_{est})^2 \quad (7.4)$$

$$\sum (Y - \bar{Y})^2 = \sum (Y_{est} - \bar{Y})^2 + \sum (Y - Y_{est})^2 \quad (7.5)$$

$$\bar{Y} = \frac{\sum_{k=1}^N Y_k}{N}, \text{ para } N \text{ puntos muestrales}$$

Gráficamente, para el caso de regresión simple se tendrá:



## 7.2 Método de los Mínimos Cuadrados

### 7.2.1 Objeto

Efectuar la regresión lineal o exponencial, simple o múltiple, de la variable  $X_1$  sobre la(s) variable(s)  $X_2, \dots, X_n$  a partir de una tabla de "m" puntos muestrales con la siguiente configuración:

Punto	$X_1$	$X_2$	...	$X_n$
1				
...				
m				

Para efectuar el ajuste se emplea el método de los mínimos cuadrados. Además, se proporciona el coeficiente de correlación y las desviaciones estándar de los parámetros de la cur-

va.

Las curvas de ajuste serán del tipo:

$$\left. \begin{aligned} X_1 &= A_1 + A_2 X_2 + A_3 X_3 + \dots + A_n X_n \\ X_1 &= e^{A_1} e^{A_2 X_2} e^{A_3 X_3} \dots e^{A_n X_n} \end{aligned} \right\} \quad (7.6)$$

Si se desea como curva de ajuste un polinomio de grado " $(n-1)$ " o sea:

$$X_1 = A_1 + A_2 X_2 + A_3 X_2^2 + \dots + A_n X_2^{n-1} \quad (7.7)$$

solo se requerirá efectuar el siguiente cambio de variable:

$$\left. \begin{aligned} X_1 &= X_2 \\ X_3 &= X_2^2 \\ X_4 &= X_2^3 \\ &\vdots \\ X_n &= X_2^{n-1} \end{aligned} \right\} \quad (7.8)$$

en el momento de proporcionar los datos.

### 7.2.1 Método

Dadas un conjunto de "m" observaciones de la variable dependiente  $X_1$  sobre una o varias variables independientes  $X_2, \dots, X_n$  se busca ajustar los datos mediante la siguiente curva:

$$\hat{f}(X) = \hat{X}_1 = A_1 + A_2 X_2 + A_3 X_3 + \dots + A_n X_n \quad (7.9)$$

El valor de la variable dependiente correspondiente al valor de las variables independientes en un punto muestral  $X_{1,d}$  es  $X_{1,d}$  por lo que el error será:

$$e_d = f(X_d) - X_{1,d} \quad (7.10)$$

$$e_d = A_1 + A_2 X_{2,d} + A_3 X_{3,d} + \dots + A_n X_{n,d} - X_{1,d} \quad (7.11)$$

\*  $\hat{X}_1$  = valor estimado de  $X_1$

y la suma de los cuadrados de los errores considerando todos los puntos muestrales es:

$$\sum_{L=1}^m e_L^2 = \sum_{L=1}^m [A_1 + A_2 X_{2,L} + A_3 X_{3,L} + \dots + A_n X_{n,L} - X_{1,L}]^2 \quad (7.12)$$

para obtener el mínimo de la suma de los cuadrados de los errores se deriva la expresión (7.12) con respecto a los parámetros  $A_j$  y cada una de las derivadas se iguala a cero para toda  $j$ :

$$\frac{\partial}{\partial A_j} \sum_{L=1}^m e_L^2 = \frac{\partial}{\partial A_j} \sum_{L=1}^m [A_1 + A_2 X_{2,L} + \dots + A_n X_{n,L} - X_{1,L}]^2$$

$$= \sum_{L=1}^m 2 [A_1 + A_2 X_{2,L} + \dots + A_n X_{n,L} - X_{1,L}] X_{j,L} = 0 \quad (7.13)$$

Lo cual se cumple solo si:

$$A_1 \sum_{L=1}^m X_{1,L} + A_2 \sum_{L=1}^m X_{2,L} X_{1,L} + \dots + A_n \sum_{L=1}^m X_{n,L} X_{1,L} = \sum_{L=1}^m X_{1,L} X_{1,L} \quad (7.14)$$

al evaluar (7.14) para cada "j" se tiene:

$$\begin{aligned} m A_1 + A_2 \sum X_2 + A_3 \sum X_3 + \dots + A_n \sum X_n &= \sum X_1 \\ A_1 \sum X_2 + A_2 \sum X_2^2 + A_3 \sum X_3 X_2 + \dots + A_n \sum X_n X_2 &= \sum X_1 X_2 \\ \vdots & \vdots \\ A_1 \sum X_n + A_2 \sum X_2 X_n + A_3 \sum X_3 X_n + \dots + A_n \sum X_n^2 &= \sum X_1 X_n \end{aligned} \quad (7.15)$$

expresando en forma matricial:

$$\begin{bmatrix} 1 & & & & & \\ & \sum X_2 & & & & \\ & \sum X_2^2 & & & & \\ & & \sum X_3 X_2 & & & \\ & & & \ddots & & \\ & & & & \sum X_n X_n & \\ & \sum X_1 X_2 & & & & \\ & & \sum X_1 X_3 & & & \\ & & & \ddots & & \\ & & & & \sum X_1 X_n & \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} = \begin{bmatrix} \sum X_1 \\ \sum X_1 X_2 \\ \vdots \\ \sum X_1 X_n \end{bmatrix} \quad (7.16)$$

Al resolver el sistema de ecuaciones (7.16) se obtienen los parámetros  $A_j$  de la curva de regresión.

Dentro del programa se plantea el sistema de ecuaciones, obteniendo la sumatoria de los puntos muestrales para cada variable, para el cuadrado de la variable y para los productos cruzados. El sistema de ecuaciones se resuelve por el método de Gauss-Jordan modificado.

El coeficiente de correlación se obtiene de la siguiente forma:

$$r_{1(23\dots n)} = \sqrt{\frac{\text{variación explicada}}{\text{variación total}}} = \sqrt{\frac{\sum_{j=2,3,\dots,n} X_j^2}{\sum X_1^2}} \quad (7.17)$$

donde:

$$\sum X_1^2 = \sum_{j=2,3,\dots,n} X_j^2 + \sum_{j=2,3,\dots,n} X_j^2 \quad (7.18)$$

var. total    var. no explicada    var. explicada

Las componentes de la ecuación (7.16) están dadas por:

$$\sum_{L=1}^m X_{j,L}^2 = \sum_{L=1}^m X_{1,L}^2 - \frac{(\sum_{L=1}^m X_{1,L})^2}{m} \quad (7.19)$$



$$\sum_{j=1}^2 \dots \sum_{j=n} = A_1 \sum X_1 X_2 + A_2 \sum X_1 X_3 + \dots + A_n \sum X_1 X_n \quad (7.20)$$

$$\left. \begin{aligned} \sum X_1 X_2 &= \sum X_1 X_2 - \frac{\sum X_1 \sum X_2}{m} \\ \sum X_1 X_3 &= \sum X_1 X_3 - \frac{\sum X_1 \sum X_3}{m} \\ &\vdots \\ \sum X_1 X_n &= \sum X_1 X_n - \frac{\sum X_1 \sum X_n}{m} \end{aligned} \right\} \quad (7.21)$$

A los términos de las ecuaciones (7.19) y (7.21) se les denominan elementos de variación y covariación respectivamente, ya que  $x = X - \bar{X}$  y la variancia y covariancia se definen como:

$$\sigma_x^2 = E [X - \bar{X}]^2 \quad (7.22)$$

$$\text{cov}_{xy} = E [X - \bar{X}][Y - \bar{Y}] \quad (7.23)$$

A continuación se describe la obtención de la desviación estándar de los parámetros  $A_j$ .

Se define a los productos  $\sum X'X$  y  $\sum X'Y$  como:

$$\sum X'X = \begin{bmatrix} 1 & 1 & \dots & 1 \\ X_{1,1} & X_{1,2} & \dots & X_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n,1} & X_{n,2} & \dots & X_{n,m} \end{bmatrix} \begin{bmatrix} 1 & X_{2,1} & \dots & X_{n,1} \\ 1 & X_{2,2} & \dots & X_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_{2,m} & \dots & X_{n,m} \end{bmatrix} \quad (7.24)$$

$$\begin{bmatrix} n & \sum X_1 & \dots & \sum X_n \\ \sum X_1 & \sum X_1^2 & \dots & \sum X_1 X_n \\ \vdots & \vdots & \ddots & \vdots \\ \sum X_n & \sum X_1 X_n & \dots & \sum X_n^2 \end{bmatrix}$$

$$\sum X'Y = \begin{bmatrix} 1 & 1 & \dots & 1 \\ X_{2,1} & & & X_{2,m} \\ \vdots & & & \vdots \\ X_{n,1} & & & X_{n,m} \end{bmatrix} \begin{bmatrix} X_{1,1} \\ X_{1,2} \\ \vdots \\ X_{1,m} \end{bmatrix} = \begin{bmatrix} \sum X_1 \\ \sum X_1 X_2 \\ \vdots \\ \sum X_1 X_n \end{bmatrix} \quad (7.25)$$

Sea  $A$  el vector de los parámetros de la ecuación de regresión, el sistema de ecuaciones (7.16) se puede denotar en base a lo anterior como:

$$\sum X'X A = \sum X'Y \quad (7.26)$$

donde  $Y$  representa el vector de valores muestrales de la variable dependiente. Por lo tanto:

$$Y = X A + \epsilon \quad (7.27)$$

$$A = (\sum X'X)^{-1} \sum X'Y \quad (7.28)$$

Se asume que la ecuación de regresión es de la forma:

$$Y = X A + \epsilon \quad (7.29)$$

A partir de las expresiones anteriores se puede demostrar (ver ref. 1) que la variancia de los parámetros  $A_j$  está dada por:

$$E [A - \alpha] [A - \alpha]' = \sigma^2 (\sum X'X)^{-1} \quad (7.30)$$

El valor de  $\sigma^2$  se obtiene mediante la expresión:

$$\sigma^2 = \frac{\sum_{j=1}^2 \dots \sum_{j=n} X_{j,1}^2 \dots X_{j,m}}{m-n} \quad (7.31)$$

donde "m" representa la cantidad de puntos muestrales y "n" la cantidad de variables independientes.

Para efectos de regresión exponencial del tipo:

$$X_j = e^{A_1} e^{A_2 X_j} \dots e^{A_{n+1} X_j} \quad (7.32)$$

$\epsilon$  es el vector de diferencias entre los valores estimados y los valores reales.

el programa aplica el operador "Ln" en ambos terminos de la ecuación (1.32) con lo que dicha expresión se linealiza y esta expresión linealizada es la que se utiliza para la obtención de los parámetros  $A_j$ .

1.1.5 Descripción del Programa

a) Subrutinas requeridas:

- SUBROUTINE SISTOR(M,N,C,A,B), plantea el sistema de ecuaciones requerido para la obtención de los parámetros de la curva de regresión.
- SUBROUTINE GAUJORD(A,B,M,EPS,DET), obtiene la solución del sistema de ecuaciones mediante el método de Gauss-Jordan. Consultar el capítulo 3.
- SUBROUTINE MULTM(A,B,N,M,L,X), efectúa productos matriciales. Consultar el capítulo 1.
- SUBROUTINE MATINV(A,N,EPS,DET), obtiene la inversa de una matriz por el método de Gauss-Jordan. Consultar el capítulo 1.
- SUBROUTINE GRAF(A,M,N), obtiene la gráfica de los valores muestrales y de los valores estimados. Consultar el capítulo 1.

b) Descripción de las variables:

Para la subrutina SISTOR:

- M cantidad de puntos muestrales
- N cantidad de variables, incluyendo la de pendiente
- C(I,J) valor de la variable  $X_j$  para el punto muestral "i"
- A(I,J) matriz de coeficientes del sistema de ecuaciones
- B(I) vector de términos independientes del sistema de ecuaciones
- SUM variable que guarda la sumatoria de los puntos muestrales

Para el programa principal:

- M cantidad de puntos muestrales
- N cantidad de variables incluyendo la de

- pendiente
- NTIPO variable que indica el tipo de regresión a efectuar
- C(I,J) valor de la variable  $X_j$  para el punto muestral "i"
- A(I,J) matriz de coeficientes del sistema de ecuaciones
- B(I) vector de términos independientes del sistema de ecuaciones
- EPS criterio para determinar si el determinante de  $A$  es nulo
- DET variable que indica si el determinante de la matriz  $A$  es o no nulo
- PROD(I) sumatorias de los productos cruzados  $X_i X_j$
- CM(I,J) matriz modificada de la matriz  $C$  donde  $CM(I,1) = 1$
- XTR(I,J) matriz transpuesta de la matriz  $CM$
- XI(I,J) matriz inversa del producto matricial  $[XTR][CM]$
- AI coeficiente de correlación
- VAR variancia no explicada
- BVAR(I) desviación estándar de los parámetros de la curva de regresión
- ATEMP(I) variable de reemplazo

c) Dimensiones:

El programa está estructurado para trabajar como máximo con cinco variables independientes. La proposición DIMENSION deberá modificarse en el caso de que la cantidad de puntos muestrales sea mayor de 30.

d) Formatos para los datos de entrada:

SEC.	TARJETAS	FORMATO	INFORMACION
1		(I13,A4)	M, N, NTIPO, para la variable NTIPO se deberá poner LINC en el caso de regresión lineal y EXPO para regresión exponencial.

2

(8F10.0)

C(I,J), los elementos de la matriz se dan columna por columna. Emplear tantas tarjetas como sean necesarias.

otros paquetes de datos [opcional]

8

TARJETA EN BLANCO, al finalizar toda la información

e) Diagrama de bloques:

Fig. 7.1 Diagrama de bloques para el programa principal

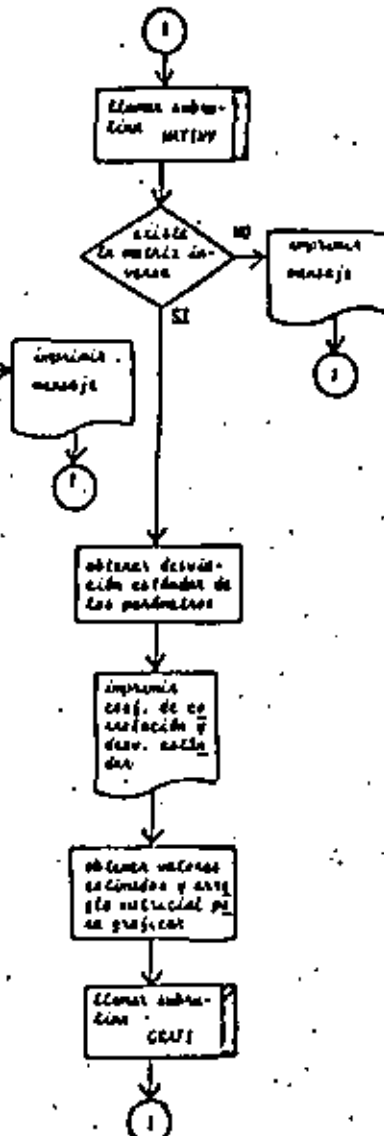
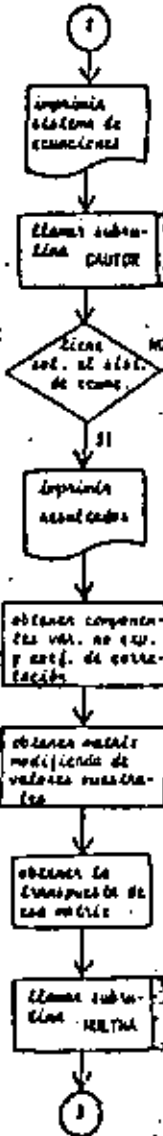
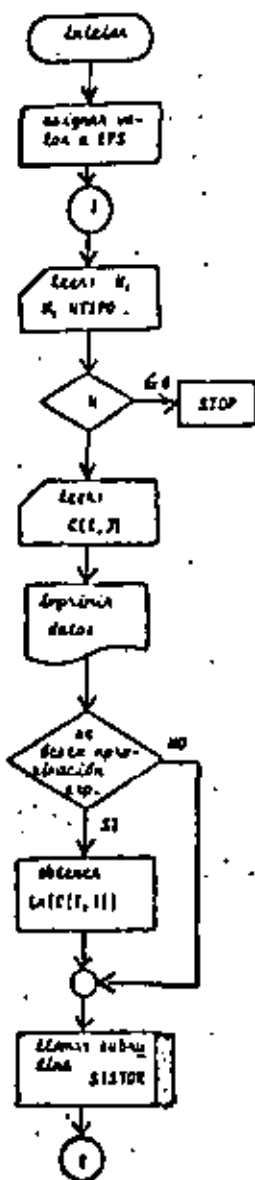
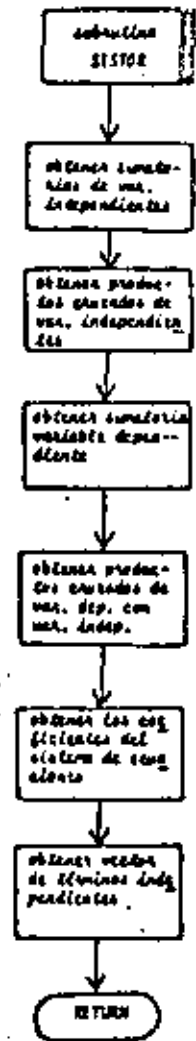


Fig. 7.2 Diagrama de bloques para la subrutina SITOR



56

```

C
C   PROGRAMA PARA EFECTUAR REGRESION LINEAL O EXPONENCIAL MULTIPLE
C   EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C   N=CANTIDAD DE PUNTOS MUESTRALES
C   M=CANTIDAD DE VARIABLES INCLUYENDO LA Y DEPENDIENTE
C   M TIPO=VARIABLE QUE INDICA EL TIPO DE REGRESION A EFECTUAR (ENPO O
C       LINE)
C   C(I)=VALOR DE LA VARIABLE X(I) PARA EL PUNTO MUESTRAL I
C   Y(I)=VARIABLE DEPENDIENTE
C   A=MATRIZ DE COEFICIENTES DEL SISTEMA DE ECUACIONES, QUE SIRVEN PA-
C   RA ENCONTRAR LOS PARAMETROS DE LA FUNCION DE REGRESION
C   B=PARAMETROS DE LA FUNCION DE REGRESION
C   EPS=VALOR CON EL QUE SE CONSIDERA NULO EL DETERMINANTE DE LA MA-
C   TRIZ A
C   DET=VALOR ABSOLUTO DEL DETERMINANTE DE LA MATRIZ A
C   PROD=SUMATORIAS DE LOS PRODUCTOS CRUZADOS CON LA Y DEPENDIENTE
C   CM=MATRIZ MODIFICADA DE LA MATRIZ C DONDE CM(I,J)=1
C   X=MATRIZ TRANSUESTA DE LA MATRIZ CM
C   XI=MATRIZ INVERSA DEL PRODUCTO MATRICIAL X*CM
C   BV=DESVIACION ESTANDAR DE LOS PARAMETROS B
C   VAR=VARIACION NO EXPLICADA
C   A1=COEFICIENTE DE CORRELACION
C
C   LA FUNCION DE REGRESION ES DEL SIGUIENTE TIPO
C   X(1)*B(1) + B(2)*X(2) + B(3)*X(3) + ... + B(M)*X(M)
C
C   DIMENSIONES PARA OBTENER LOS PARAMETROS DE LA FUNCION DE REGRESION
C
C   DIMENSION C(10,10),X(10),B(10),A(10,10),ATEMP(10)
C   DIMENSIONES PARA OBTENER LA DESVIACION ESTANDAR DE B(I)
C   DIMENSION PROD(10),CM(10,10),X(10,10),XI(10,10),BVVAR(10)
C
C   DATA X(1),X(2),X(3),X(4),X(5),X(6),X(7),X(8),X(9),X(10),
C   B(1),B(2),B(3),B(4),B(5),B(6),B(7),B(8),B(9),B(10)
C
C   LECTURA DE DATOS
C
C   EPS=0.010001
C   READ(5,50) M,M,TIPO
C   IF(M) 7,2,3
C   2 CALL EXIT
C   3 DO 4 J=1,M
C   4 READ(5,51) ((C(I,J),I=1,N)
C     LIN=4*LINE
C
C   IMPRESION DE
C
C   WRITE(6,52) M,M
C   WRITE(6,53) (X(I),I=1,M)
C   DO 5 I=1,M
C   5 WRITE(6,54) ((C(I,J),J=1,M)
C
C   LLAMADO DE SUBROUTINA PARA FORMAR EL SISTEMA DE ECUACIONES
C
C   IF(M TIPO.EQ.LIN) GO TO 26
C   DO 25 I=1,M
C   ATEMP(I)=C(I,I)
C   25 C(I,I)=ALOG(C(I,I))
C   26 CALL SIGTOR(M,C,A,M)
C
C   IMPRESION DEL SISTEMA DE ECUACIONES
C
C   WRITE(6,55)
C   DO 6 I=1,M
C   6 WRITE(6,56) ((C(I,J),J=1,M),B(I))
C
C   LLAMADO DE SUBROUTINA PARA RESOLVER EL SISTEMA DE ECUACIONES
C
C   CALL GAUTOREA(B,M,EPS,DET)
C   IF(DET.LE.EPS) GO TO 6
C
C   IMPRESION DE RESULTADOS
C
C   WRITE(6,57)
C   DO 7 I=1,M
C   7 WRITE(6,58) (B(I)
C   WRITE(6,59) M TIPO
C
C   OBTENCION DE SUM(XI)*A1
C
C   A1=0.0
C   DO 9 I=1,M
C   9 A1=A1 + C(I,I)

```

```

C
C
C      OBTENCION DE SUM(X(I)*X(I)*A2) SUM(X(I)*A3)
C
C      DO 11 I=1,N
C      A2=0.0
C      A3=0.0
C      DO 10 J=1,M
C      A2=A2 + C(J,I)*C(J,I)
10  A3=A3 + C(J,I)
C      PROD(I)=A2 - (A1+A3)/FLOAT(N)
11  CONTINUE
C      WRITE(6,54) (PROD(I),I=1,M)
C
C      OBTENCION DE VAR. NO EXPLICADA Y DEL COEF. DE CORRELACION
C
C      VAR=0.0
C      DO 12 I=2,M
12  VAR=VAR + B(I)*PROD(I)
C      A1=SQRT(VAR/PROD(1))
C      VAR=PROD(1) - VAR
C      VAR=VAR/FLOAT(N-M)
C      WRITE(6,54) VAR
C
C      OBTENCION DE LA MATRIZ MODIFICADA DE VALORES MUESTRALES
C
C      DO 13 I=1,M
C      DO 14 J=1,M
C      IF(J.EQ.I) GO TO 11
C      CK(I,J)=C(I,J)
C      GO TO 14
13  CK(I,J)=1.0
14  CONTINUE
15  CONTINUE
C
C      OBTENCION DE CH TRANSPUESTA
C
C      DO 16 I=1,M
C      DO 14 J=1,M
16  XTR(J,I)=CK(I,J)
C
C      LLAMADO DE SUBROUTINA PARA EFECTUAR PRODUCTO MATRICIAL
C
C      CALL MULTMATR(CH,M,M,M,XTR)
C
C      LLAMADO DE SUBROUTINA PARA OBTENER LA MATRIZ INVERSA
C
C      CALL MATINVER(M,XEPS,DET)
C      IF(DET.LE.XEPS) GO TO 19
C      DO 20 I=1,M
20  WRITE(6,50) CK(I,J),J=1,M)
C
C      OBTENCION DE LA DESVIACION ESTANDAR DE LOS PARAMETROS
C
C      DO 17 I=1,M
C      BVAR(I)=VAR*X(I),I)
17  BVAR(I)=SQRT(BVAR(I))
C
C      IMPRESION DE COEFICIENTES Y DESVIACIONES ESTANDAR
C
C      WRITE(6,50)
C      DO 18 I=1,M
18  WRITE(6,50) B(I),BVAR(I)
C      WRITE(6,60) 41
C
C      REACORDO DE DATOS PARA GRAFICAR VALORES REALES Y VALORES ESTIMADOS
C      CON LA FUNCION DE REGRESION
C
C      IF(MTIPO.NE.LIN) GO TO 27
C      DO 21 I=1,N
C      CK(I,1)=FLGAT(I)
C      CK(I,2)=C(I,1)
C      SUM=0.0
C      DO 20 J=2,M
20  SUM=SUM + C(I,J)*B(J)
C      CK(I,3)=C(I,1) + SUM
21  CONTINUE
C      GO TO 30
27  DO 28 I=1,N
C      CK(I,1)=FLGAT(I)
C      CK(I,2)=ATENP(I)
C      SUM=EXP(B(I))
C      DO 20 J=2,M
28  SUM=SUM*EXP(C(I,J)*B(J))
C      CK(I,3)=SUM
29  CONTINUE

```

## LLAMADO DE SUBROUTINA PARA GRAFICAR

```

30 CALL GRFT(CM,N,3)
   GO TO 1.
   8 WRITE(6,57)
   GO TO 1
  19 WRITE(4,61)
   GO TO 1

```

## FORMATOS DE LECTURA E IMPRESION

```

50 FORMAT(2I5,4F)
51 FORMAT(2F10.0)
52 FORMAT(1M1,5E/1,5X,35M1 LA CANTIDAD DE PUNTOS MUESTRALES ES +15,3E/
1,5X,20M1 LA CANTIDAD DE VARIABLES ES +15)
53 FORMAT(///5E/1,10X,20M1 LOS VALORES MUESTRALES SON,///,5X,4S,9E7X,4
15)///)
54 FORMAT(//2X,10(IPE11,4,1X))
55 FORMAT(///5E/1,10X,47M1 LOS COEFICIENTES DE LA FUNCION DE REGRESION
1 SCN,///,4X,1M1,37X,4M1C(1,1))
56 FORMAT(//45X,12,10X,1PE15.8)
57 FORMAT(5E/1,5X,27M1 EL SISTEMA DE ECUACIONES NO TIENE SOLUCION)
58 FORMAT(5M1,3E/1,10X,19M1 LOS DESVIACIONES ESTANDAR DE LOS COEFICIENT
1ES SCN,///,45X,4M1C(1,1))21X,4PD.5,///)
59 FORMAT(1E/1,4X,2(IPE15,9,10X))
60 FORMAT(1E/1,5X,34M1 EL COEFICIENTE DE CORRELACION ES +1PE15.8)
61 FORMAT(5E/1,5X,27M1 NO EXISTE LA INVERSA DE LA MATRIZ EMPLEADA PARA
1CALCULAR LAS DESVIACIONES ESTANDAR)
550 FORMAT(//2X,1PE10,3,10(1X,1PE10,3))
551 FORMAT(1M1,5E/1,5X,27M1 EL SISTEMA DE ECUACIONES (5,1)
552 FORMAT(//,5X,21M1 TIPO DE REGRESION " +A4)
END

```

Fig. 7.3 Listado del programa principal

## SUBROUTINE SISTOR(N=1,C,A,B)

```

C
C SUBROUTINA PARA PLANTEAR EL SISTEMA DE ECUACIONES QUE PERMITE OBTEN-
C NER LOS PARAMETROS DE UNA FUNCION DE REGRESION LINEAL MULTIPLE
C EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C N=CANTIDAD DE PUNTOS MUESTRALES
C M=CANTIDAD DE VARIABLES (INCLUYENDO LA DEPENDIENTE)
C C(I,J)=VALOR DE LA VARIABLE X(I) PARA EL PUNTO MUESTRAL I
C A=MATRIZ DE COEFICIENTES DEL SISTEMA DE ECUACIONES
C B=VECTOR DE TERMINOS INDEPENDIENTES DEL SISTEMA DE ECUACIONES
C
C EL PLANTAMIENTO APROVECHA LAS CARACTERISTICAS DE SIMETRIA DE LA
C MATRIZ A
C
C DIMENSION C(10,10),A(10,10),B(10)
C
C OBTENCION DEL SISTEMA DE ECUACIONES
C
   A(1,1)=A
   DO 1 I=1,M
   IF(I.NE.1) GO TO 4
   SUM=0.0
   DO 1 J=1,M
   1 SUM=SUM + C(I,J)
   B(I)=SUM
   IP=I+1
   DO 3 J=IP,M
   SUM=0.0
   DO 2 K=1,M
   2 SUM=SUM + C(K,I)*C(K,J)
   A(I,J)=SUM
   A(J,I)=SUM
   3 CONTINUE
   GO TO 4
   4 SUM=0.0
   DO 5 J=1,M
   5 SUM=SUM + C(J,1)*C(J,1)
   B(1)=SUM
   DO 7 J=1,M
   SUM=0.0
   DO 6 K=1,M
   6 SUM=SUM + C(K,1)*C(K,J)
   A(1,J)=SUM
   A(J,1)=SUM
   7 CONTINUE
   8 CONTINUE
   RETURN
   END

```

Fig. 7.4 Listado de la subrutina SISTOR

## 7.2.4 Ejemplo

Los valores observados para la demanda de energía eléctrica en el sector residencial desde 1962 hasta 1973 son:

ANO	DEMANDA RESIDENCIAL (MWH)
1962	1418.757
1963	1578.572
1964	1816.236
1965	1970.987
1966	2256.216
1967	2548.05
1968	2803.79
1969	3152.095
1970	3582.568
1971	3979.667
1972	4431.655
1973	4930.197

Si se sabe que la demanda de energía eléctrica se encuentra estrechamente relacionada con el tiempo, el PNB, la población y el producto bruto del sector eléctrico del año anterior. Obtenga una curva de tipo lineal y otra de tipo exponencial que se ajuste lo mejor posible a los valores muestrales de la demanda residencial de energía eléctrica.

## \* SOLUCION

TABLA 7.1 Datos del problema del ejemplo 7.2.4

$$N = 12$$

$$M = 4$$

$$NTIPO = LINE \text{ ó } EXPO$$

1418.757	62.	165310.	37439.	1609.
1578.572	63.	178516.	38727.	1753.
1816.236	64.	199390.	40059.	2170.
1970.987	65.	212320.	41437.	2529.
2256.216	66.	227037.	42863.	2769.
2548.05	67.	241272.	44338.	3157.
2803.79	68.	260901.	45863.	3533.
3152.095	69.	277400.	47441.	4228.
3582.568	70.	296600.	49031.	4812.
3979.667	71.	306800.	50778.	5357.
4431.655	72.	329100.	52539.	5784.
4930.197	73.	354100.	54560.	6297.

62

TABLA 7.2 Resultados del problema del ejemplo 7.2.4

68

EL SISTEMA DE ECUACIONES ES

1.270E+03	8.106E+02	3.889E+06	5.431E+05	4.400E+04	3.487E+04
8.170E+02	5.402E+04	2.082E+08	3.701E+07	3.013E+08	2.372E+06
3.889E+08	2.622E+08	8.188E+11	1.122E+13	1.224E+10	9.518E+09
5.431E+05	3.701E+07	1.422E+11	2.510E+10	2.087E+09	3.438E+09
4.400E+04	3.487E+04	1.224E+10	2.087E+09	1.897E+08	1.467E+08

LOS COEFICIENTES DE LA FUNCION DE REGRESION SON

i	B(i)
1	1.25536780E+01
2	-1.6028289E+02
3	-7.2692064E+04
4	4.61311087E+01
5	1.7602042E+01

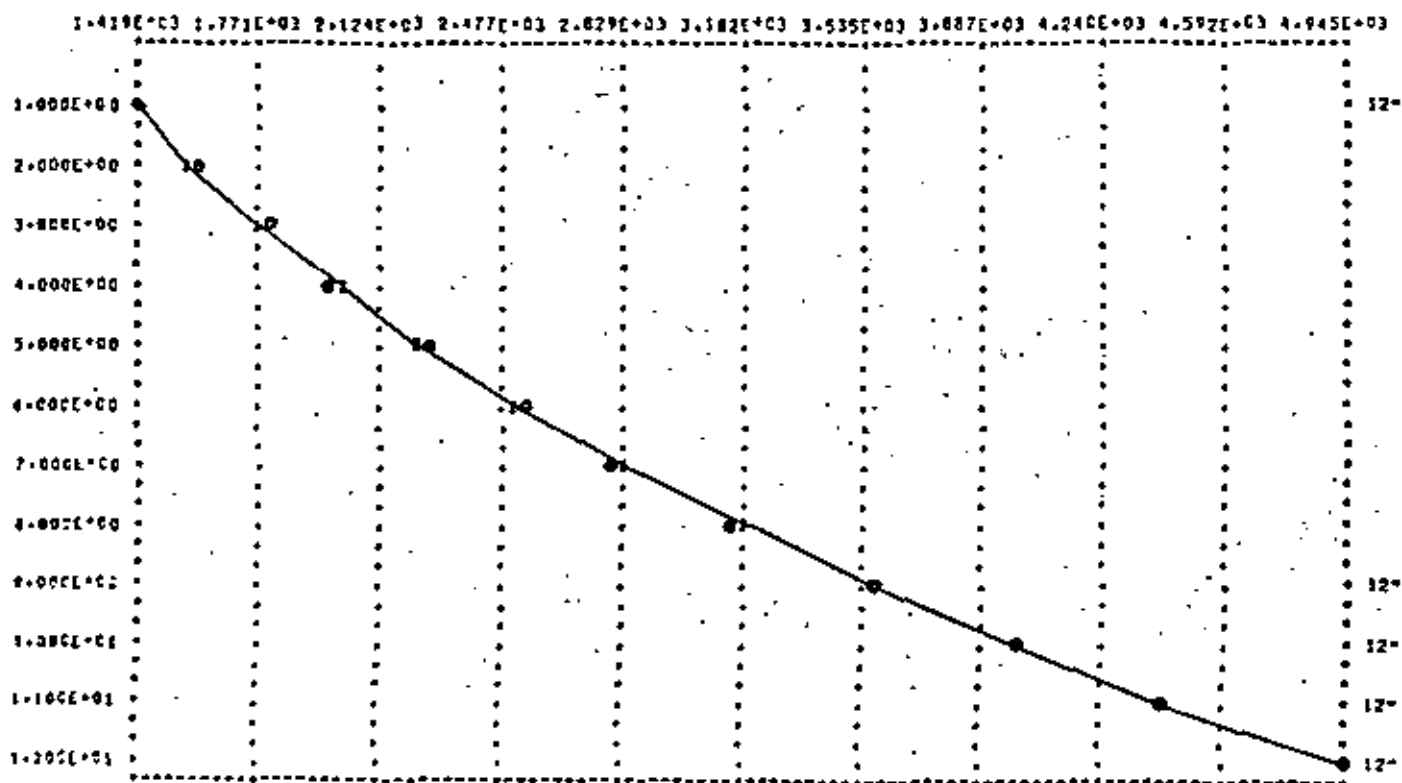
TIPO DE REGRESION LINE

LAS DESVIACIONES ESTANDAR DE LOS COEFICIENTES SON

i	S(i)	D.S.
1	1.25536780E+01	2.36876970E+03
2	-1.6028289E+02	5.72189127E+01
3	-7.2692064E+04	3.95854087E+01
4	4.61311087E+01	4.7619982E+02
5	1.7602042E+01	8.4634620E+02

EL COEFICIENTE DE CORRELACION ES 0.99730818E+01





62

133 69

EL SISTEMA DE ECUACIONES ES

1.200E+01	8.100E+02	1.000E+04	5.451E+05	6.400E+04	9.464E+01
8.100E+02	5.402E+04	2.052E+08	1.201E+07	1.033E+04	6.405E+03
1.047E+04	2.002E+08	8.144E+11	1.422E+11	1.224E+10	2.432E+07
5.451E+05	1.201E+07	1.422E+11	2.510E+10	2.097E+09	4.324E+06
6.400E+04	1.033E+04	1.224E+10	2.097E+09	1.897E+08	1.542E+05

LOS COEFICIENTES DE LA FUNCION DE REGRESION SON

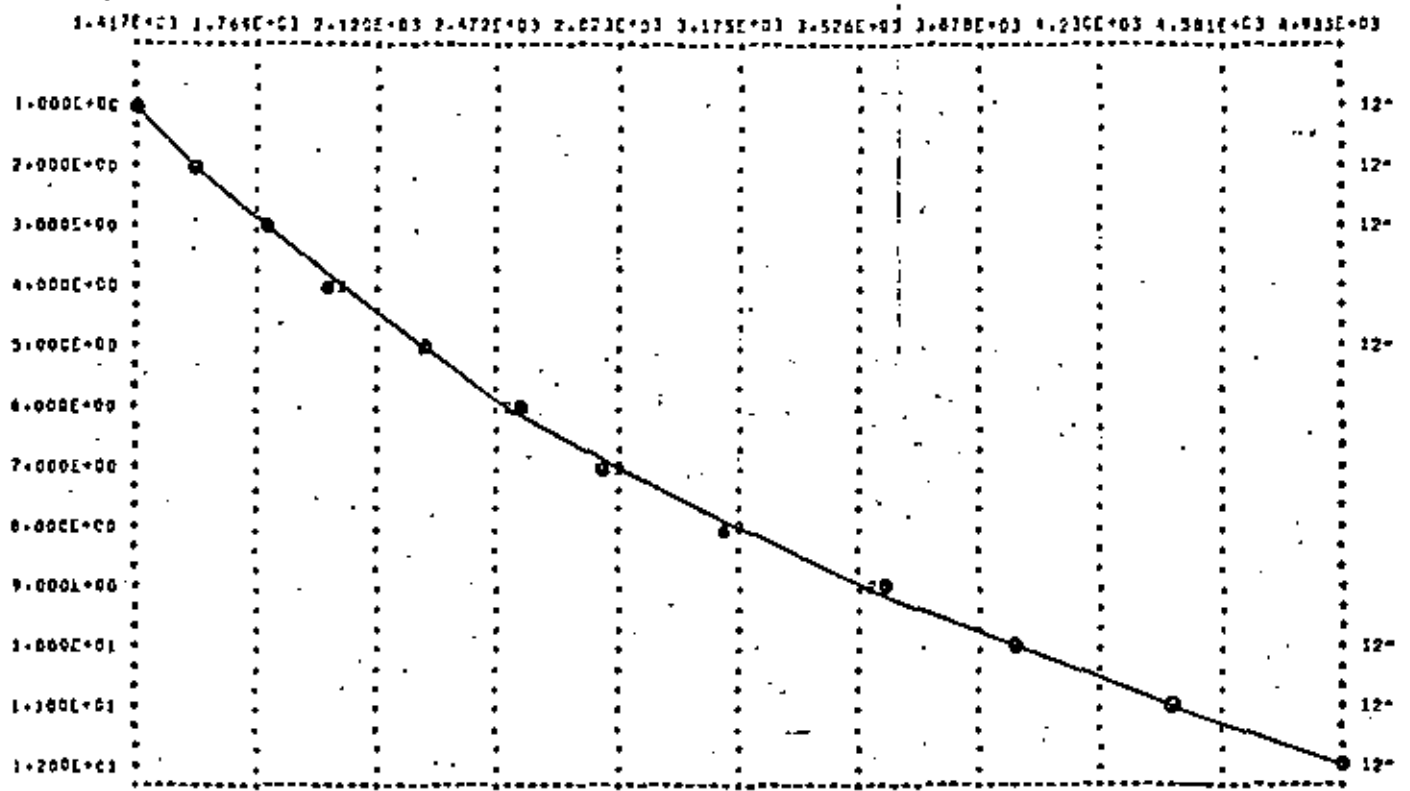
i	$b(i)$
1	-4.11430952E-01
2	1.38567095E-01
3	5.55134406E-07
4	-2.80140415E-05
5	2.00932652E-05

TIPO DE REGRESION EXPD

LAS DESVIACIONES ESTANDAR DE LOS COEFICIENTES SON

$b(i)$	$D_{b_i}$
-4.11430952E-01	7.91875704E-01
1.38567095E-01	1.92910641E-02
5.55134406E-07	1.33440103E-06
-2.80140415E-05	2.28054652E-05
2.00932652E-05	2.84014915E-05

EL COEFICIENTE DE CORRELACION ES 9.44769155E-02



1. CROXTON F., COMDEN D., BOLCH B., "Practical Business Statistics", Englewood Cliffs N.J., Prentice-Hall Inc., 1969.  
pp.192-263.
2. GEREZ G. Victor, GRIJALVA L. Luis, "El Enfoque de Sistemas". México: Editorial Limusa S.A., 1976.  
pp.150-164.
3. KUO S. Shan, "Computer Applications of Numerical Methods". Reading Mass.: Addison-Wesley Co., 1971.  
pp.252-261.
4. SPIEGEL Murray, "Estadística". México: Libros de Gran-Hill de México, S.A., 1961.  
pp.217-222.

## VI) SOLUCION ECUACIONES DIFERENCIALES ORDINARIAS.

Las ecuaciones diferenciales ordinarias son aquellas en las que la variable dependiente es función de una sola variable independiente :

$$y^{(n)} = f(x, y, y', \dots, y^{(n-1)})$$

### a) Método de Euler

Se tratará el caso de ecuaciones diferenciales ordinarias de primer orden :

$$dy = y' dx$$

Substituyendo por los incrementos en la expresión anterior se tiene :

$$\Delta y = y' \Delta x \quad (VI.0)$$

Tomando un punto inicial para arrancar y conservando un incremento constante  $\Delta x$  se obtiene la siguiente fórmula iterativa :

$$\begin{array}{l} Y_1 = Y_0 + Y' \left| \begin{array}{l} \Delta x \\ (X_0, Y_0) \end{array} \right. \\ Y_2 = Y_1 + Y' \left| \begin{array}{l} \Delta x \\ (X_1, Y_1) \end{array} \right. \\ \vdots \\ Y_{n+1} = Y_n + Y' \left| \begin{array}{l} \Delta x \\ (X_n, Y_n) \end{array} \right. \end{array} \quad (VI.1)$$

$$y^{(n)} \equiv \frac{d^n y}{dx^n}$$

La ecuación VI.1 nos da la fórmula recursiva de Euler. Para aplicar el método se requiere que  $\Delta x$  sea pequeño y además contar con un punto de inicio  $(X_0, Y_0)$ . El error producido es del orden de  $\Delta x^2$ .

b) Euler modificado

El procedimiento básico es el mismo solo que para cada  $Y_{i+1}$  se hace una serie de iteraciones con los valores obtenidos sucesivamente de  $Y_{i+1}$  a fin de obtener el valor más exacto de  $Y_{i+1}$ .

Al tener:

$$Y_{i+1} = Y_i + Y' \Big|_{(X_i, Y_i)} \Delta x \quad (VI.2)$$

se efectúan las siguientes iteraciones:

$$Y'_{i+1} = F(X_{i+1}, Y_{i+1})$$

$$\hat{Y}_{i+1} = Y_i + \frac{(Y' \Big|_{i+1} + Y' \Big|_{i+1})}{2} \Delta x$$

$$\hat{Y}'_{i+1} = F(X_{i+1}, \hat{Y}_{i+1})$$

$$\hat{\hat{Y}}_{i+1} = Y_i + \frac{(\hat{Y}' \Big|_{i+1} + \hat{Y}' \Big|_{i+1})}{2} \Delta x$$

y así sucesivamente hasta que:

$$|\hat{\hat{Y}}_{i+1} - \hat{Y}_{i+1}| < \epsilon \quad (VI.3)$$

al cumplirse, se procede a obtener  $Y_{i+2}$  y así sucesivamente.

Al igual que en el método anterior es necesario emplear incrementos ( $\Delta x$ ) pequeños. El error producido es del orden  $\Delta x^3$ .

c) Método de Runge - Kutta

Este método utiliza las fórmulas de integración antes vista para llegar a la obtención de su propia fórmula recursiva. Dicho proceso es bastante laborioso por lo que no se tratará.

La solución para una ecuación diferencial de primer orden  $Y' = f(x, y)$  está dado por:

$$Y_{n+1} = Y_n + \Delta Y_n$$

donde:

$$\Delta Y_n = \frac{\Delta x}{6} (K_0 + 2K_1 + 2K_2 + K_3)$$

$$K_0 = f(X_n, Y_n)$$

$$K_1 = f\left(X_n + \frac{\Delta x}{2}, Y_n + \frac{K_0 \Delta x}{2}\right)$$

$$K_2 = f\left(X_n + \frac{\Delta x}{2}, Y_n + \frac{K_1 \Delta x}{2}\right)$$

$$K_3 = f(X_n + \Delta x, Y_n + K_2)$$

La fórmula de Euler es la de Runge-Kutta de 4o. orden, hay otras fórmulas con mayor cantidad de términos que se obtienen empleando diferencias de mayor orden al deducir la fórmula.

Los parámetros  $K_j$  representan la pendiente de la función en los puntos en que se está evaluando. El método da un error del orden de  $\Delta x^5$  y es uno de los más precisos.

### Ejemplo

Obtener la solución de la ecuación diferencial  $Y' = 1 - X + 4Y$  y para 5 puntos consecutivos empleando los métodos de Euler, Euler mejorado y Runge-Kutta usando un incremento  $\Delta x = 0.1$ . Comparar dichos valores con la solución real si  $X_0 = 0$ ,  $Y_0 = 1$ .

Sol.

La solución exacta está dada por:

$$Y' - 4Y = 1 - X$$

$$Y_h = C e^{4X}$$

$$Y_p = A + BX$$

$$-4A - 4BX = 1 - X$$

$$A = -\frac{1}{4} \quad ; \quad B = \frac{1}{4}$$

$$Y(x) = C e^{4x} - \frac{1}{4} + \frac{1}{4} X$$

$$Y(0) = 1 = C - \frac{1}{4}$$

$$C = \frac{5}{4}$$

$$Y(x) = \frac{5}{4} e^{4x} - \frac{1}{4} + \frac{1}{4} X$$

Las fórmulas de solución para los métodos son:

$$Y_n = Y_{n-1} + Y'_{n-1} \Delta x \quad (\text{Euler})$$

$$\left. \begin{aligned} Y_n &= Y_{n-1} + Y'_{n-1} \Delta x \\ Y_n &= Y_{n-1} + \frac{(Y'_{n-1} + Y'_n) \Delta x}{2} \end{aligned} \right\} \text{Euler mejorado}$$

$$Y_n = Y_{n-1} + \frac{\Delta x}{6} [K_1 + 2K_2 + 2K_3 + K_4]$$

$$K_1 = f(X_{n-1}, Y_{n-1})$$

$$K_2 = f\left(X_{n-1} + \frac{\Delta x}{2}, Y_{n-1} + \frac{K_1 \Delta x}{2}\right) \quad \text{Runge Kutta}$$

$$K_3 = f\left(X_{n-1} + \frac{\Delta x}{2}, Y_{n-1} + \frac{K_3 \Delta x}{2}\right)$$

$$K_4 = f(X_{n-1} + \Delta x, Y_{n-1} + K_3 \Delta x)$$

Las soluciones se muestran en la siguiente tabla:

K	$X_k$	Euler	E. Mej.	R. Kutta	Real
0	0	1.	1.	1.	1.
1	0.1	1.5	1.595	1.608	1.609
2	0.2	2.19	2.463	2.505	2.505
3	0.3	3.146	3.737	3.829	3.830
4	0.4	4.774	5.609	5.792	5.794
5	0.5	6.324	8.369	8.709	8.717

d) Diferencias finitas

Este método se emplea cuando se tienen problemas con valores en la frontera.

El procedimiento consiste en lo siguiente: dividir el intervalo de integración en "n" espacios iguales, emplear las fórmulas de derivación de diferencias finitas en la ecuación diferencial (todas las diferencias deben ser del mismo orden), sustituir las condiciones de frontera y por último resolver el sistema de ecuaciones planteado. Se tiene que aplicar el operador diferencial a todos los pivotes del intervalo.

Ejemplo

Resolver la ecuación diferencial  $\frac{d^2 y}{dx^2} - y = 0$ , en el intervalo (0, 1)

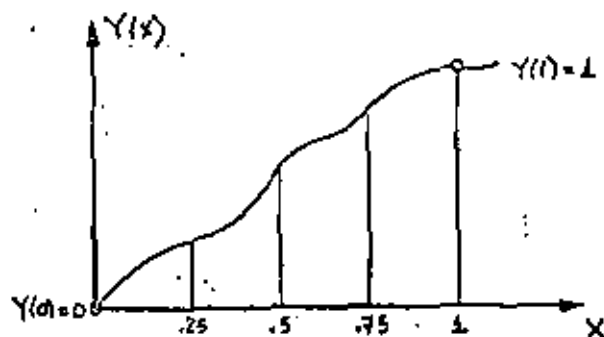
si  $y(0) = 0$ ,  $y(1) = 1$

Sol.

72

Se divide el intervalo en "n" partes iguales, sean 4 en este caso:

$$\Delta x = \frac{1-0}{4} = 0.25$$



empleando diferencias de 2o. orden:

$$y_i'' = \frac{1}{(\Delta x)^2} [y_{i-1} - 2y_i + y_{i+1}]$$

sustituyendo en la ecuación diferencial:

$$\frac{1}{\Delta x^2} [y_{i-1} - 2y_i + y_{i+1}] - y_i = 0$$

$$y_{i-1} - 2.0625 y_i + y_{i+1} = 0 \quad (VI.4)$$

las condiciones de frontera son:

$$y_0 = 0$$

$$y_4 = 1$$

aplicando VI.4 en los pivotes:

$$X_1 = 0.25$$

$$Y_0 - 2.0625 Y_1 + Y_2 = 0$$

$$-2.0625 Y_1 + Y_2 = 0$$

(VI.5)

$$X_2 = 0.5$$

$$Y_1 - 2.0625 Y_2 + Y_3 = 0$$

(VI.6)

$$X_3 = 0.75$$

$$Y_2 - 2.0625 Y_3 + Y_4 = 0$$

$$Y_2 - 2.0625 Y_3 = -1$$

(VI.7)

el sistema de ecuaciones es:

$$-2.062 Y_1 + Y_2 = 0$$

$$Y_1 - 2.062 Y_2 + Y_3 = 0$$

$$Y_2 - 2.062 Y_3 = -1$$

de donde:

$$Y_1 = 0.216$$

$$Y_2 = 0.445$$

$$Y_3 = 0.701$$

NOTA: Cuando se trata de ecuaciones diferenciales de mayor orden y se cuentan como condiciones  $y'' = 0$ , etc., hay que substituir en dichas ecuaciones las fórmulas de diferencias y despejar de ahí las condiciones de frontera desconocidas.

#### 10.4 Solución de Sistemas de Ecuaciones Diferenciales Lineales No Homogéneas de Primer Orden

##### 10.4.1 Objeto

Obtener la solución de sistemas de ecuaciones diferenciales no homogéneas, lineales, de primer orden mediante el método de Variación de Parámetros.

La representación en forma matricial para este tipo de sistemas de ecuaciones diferenciales es:

$$\underline{X}(t) = \underline{A} \underline{X}(t) + \underline{B} \underline{U}(t) \quad (10.16)$$

$$\underline{X}(t_0) = \underline{X}_0$$

donde  $\underline{U}(t)$  representa el vector de entradas externas al sistema de sistemas físicos.

Debido a que cuando se modelan sistemas dinámicos lineales las salidas no siempre corresponden a las variables empleadas en las ecuaciones diferenciales, en este programa se considera la representación completa mediante variables de estado de un sistema lineal, la cual es:

$$\left. \begin{aligned} \underline{X}(t) &= \underline{A} \underline{X}(t) + \underline{B} \underline{U}(t) \\ \underline{Y}(t) &= \underline{C} \underline{X}(t) + \underline{D} \underline{U}(t) \\ \underline{X}(t_0) &= \underline{X}_0 \end{aligned} \right\} \quad (10.17)$$

donde  $\underline{Y}(t)$  representa el vector de salidas del sistema.

##### 10.4.2 Método

El método de variación de parámetros establece que la solución del sistema de ecuaciones diferenciales lineales (10.17) tiene por solución:

$$\underline{X}(t) = e^{\underline{A}(t-t_0)} \underline{X}_0 + \int_{t_0}^t e^{\underline{A}(t-\tau)} \underline{B} \underline{U}(\tau) d\tau \quad (10.18)$$



$$X_1 = 0.25$$

$$Y_0 - 2.0625 Y_1 + Y_2 = 0$$

$$-2.0625 Y_1 + Y_2 = 0$$

(VI.5)

$$X_2 = 0.5$$

$$Y_1 - 2.0625 Y_2 + Y_3 = 0$$

(VI.6)

$$X_3 = 0.75$$

$$Y_2 - 2.0625 Y_3 + Y_4 = 0$$

$$Y_2 - 2.0625 Y_3 = -1$$

(VI.7)

el sistema de ecuaciones es:

$$-2.062 Y_1 + Y_2 = 0$$

$$Y_1 - 2.062 Y_2 + Y_3 = 0$$

$$Y_2 - 2.062 Y_3 = -1$$

de donde:

$$Y_1 = 0.216$$

$$Y_2 = 0.443$$

$$Y_3 = 0.701$$

NOTA: Cuando se trata de ecuaciones diferenciales de mayor orden y se cuenta como condiciones  $y'' = 0$ , etc., hay que substituir en dichas ecuaciones las fórmulas de diferencias y despejar de ahí las condiciones de frontera desconocidas.

#### 10.4 Solución de Sistemas de Ecuaciones Diferenciales Lineales No Homogéneas de Primer Orden

##### 10.4.1 Objeto

Obtener la solución de sistemas de ecuaciones diferenciales no homogéneas, lineales, de primer orden mediante el método de Variación de Parámetros.

La representación en forma matricial para este tipo de sistemas de ecuaciones diferenciales es:

$$\dot{X}(t) = A X(t) + B U(t) \quad (10.16)$$

$$X(t_0) = X_0$$

donde  $U(t)$  representa el vector de entradas externas si se habla de sistemas físicos.

Debido a que cuando se modelan sistemas dinámicos lineales las salidas no siempre corresponden a las variables empleadas en las ecuaciones diferenciales, en este programa se considera la representación completa mediante variables de estado de un sistema lineal, la cual es:

$$\left. \begin{aligned} \dot{X}(t) &= A X(t) + B U(t) \\ Y(t) &= C X(t) + D U(t) \end{aligned} \right\} \quad (10.17)$$

$$X(t_0) = X_0$$

donde  $Y(t)$  representa el vector de salidas del sistema.

##### 10.4.2 Método

El método de variación de parámetros establece que la solución del sistema de ecuaciones diferenciales lineales (10.17) tiene por solución:

$$X(t) = e^{A(t-t_0)} X_0 + \int_{t_0}^t e^{A(t-\sigma)} B U(\sigma) d\sigma \quad (10.18)$$

empleando la siguiente relación:

$$e^{-\Delta t} = I - \frac{A\Delta t}{1!} + \frac{A^2 \Delta t^2}{2!} - \frac{A^3 \Delta t^3}{3!} + \dots \quad (10.24)$$

en los términos entre corchetes se llega a:

$$e^{\frac{A}{\Delta t} \Delta t} \int_0^{\Delta t} e^{-\Delta t} \text{odo} = \frac{I[\Delta t]}{1!} + \dots + \frac{A^n [\Delta t]^{n+1}}{(n+1)!} + \dots \quad (10.25)$$

$$e^{\frac{A}{\Delta t} \Delta t} \int_0^{\Delta t} e^{-\Delta t} \text{do} = \frac{I[\Delta t]}{1!} + \dots + \frac{A^n [\Delta t]^{n+1}}{(n+1)!} + \dots \quad (10.26)$$

Para la evaluación de las series (10.25) y (10.26) la cantidad de términos a emplear dependerá de la exactitud deseada. Se fija un criterio de convergencia  $\epsilon$  tal que si  $Z$  representa a la matriz de la serie (10.25) y  $M$  a la matriz de la serie (10.26), se cumple que:

$$\begin{aligned} |z_{ij}^{(n+1)} - z_{ij}^{(n)}| &< \epsilon, \text{ para toda } i, j \\ |m_{ij}^{(n+1)} - m_{ij}^{(n)}| &< \epsilon, \text{ para toda } i, j \end{aligned} \quad (10.27)$$

Como las series (10.25) y (10.26) solo dependen del espaciamiento, se tendrán que evaluar una sola vez.

En el programa para obtener la relación (10.24) hay que evaluar el vector de entradas  $U(t)$  en cada uno de los puntos en que se subdivide el intervalo de integración.

En términos generales el proceso a seguir es:

- ① evaluar la matriz de transición  $e^{A(\Delta t)}$
- ② evaluar las series de las ecuaciones (10.25) y (10.26)
- ③ obtener la respuesta debida a las condiciones iniciales para  $t_i$

\*  $z_{ij}^{(n)}$  representa el elemento  $z_{ij}$  de la matriz  $Z$  compuesta por la sumatoria de "n" términos.

- ④ evaluar  $U(t)$  en  $t_i$  y  $t_{i+1}$
- ⑤ obtener la respuesta debida a las excitaciones externas mediante la relación (10.23)
- ⑥ hacer  $i=i+1$  y regresar al paso ③ hasta haber cubierto el intervalo de integración.

#### 10.4.3 Descripción del Programa

a) Subrutinas requeridas:

- SUBROUTINE EXPMA(DELTA, M, A, EXPO), obtiene la matriz de transición. Consultar sección 10.3.3.
- SUBROUTINE INTPE(DELTA, M, A, SUNA), obtiene la matriz de la serie (10.26); esta expresión se emplea para evaluar la respuesta debida a las excitaciones externas.
- SUBROUTINE INTRE(DELTA, M, A, RECTA), evalúa la expresión dada por la serie de la ecuación (10.25); esta expresión se utiliza para calcular la respuesta debida a las excitaciones externas.
- SUBROUTINE MULTMA(A, B, N, H, L, X), obtiene el producto matricial AB. Consultar el capítulo 1.
- SUBROUTINE GRAFI(A, N, M), grafica las soluciones de las variables dependientes y de las respuestas del sistema. Consultar el capítulo 1.
- SUBROUTINE EXCITA(T, F), evalúa el vector de entradas  $U(t)$  en el instante  $t_i$ .

b) Descripción de las variables:

Para la subrutina INTPE:

DELTA	espaciamiento entre los valores de la variable independiente
M	cantidad de ecuaciones diferenciales
A(I, J)	matriz de coeficientes constantes del sistema de ecuaciones diferenciales
EPS	criterio de convergencia
SUNA(I, J)	matriz resultante de evaluar la serie
CN	contador de iteraciones
DIV	factorial divisor
THEN	incremento de la variable independiente elevado a la potencia "n"
CHA(I, J)	matriz identidad

B(I, J) matriz A elevada a la potencia "n"  
 X(I, J) matriz resultante del producto AB

Para la subrutina INTRE:

DELT espaciamento entre los valores de la variable independiente  
 N cantidad de ecuaciones diferenciales  
 A(I, J) matriz de coeficientes del sistema de ecuaciones diferenciales

RECTA(I, J) matriz resultante de evaluar la serie  
 CMA(I, J) matriz identidad  
 B(I, J) matriz A elevada a la potencia "n"  
 X(I, J) matriz resultante del producto AB  
 EPS criterio de convergencia  
 TNEW incremento de la variable independiente elevado a la potencia "n"

CM contador  
 DIV factorial divisor

Para la subrutina EXCITA:

T valor del instante de tiempo en el cual se desea evaluar la expresión  $U(t)$   
 F(1, 1) valor del primer renglón de la expresión  $U(t)$  en el instante  $t_1$   
 F(2, 1) valor del segundo renglón de la expresión  $U(t)$  en el instante  $t_1$   
 F(3, 1) valor del tercer renglón de la expresión  $U(t)$  en el instante  $t_1$   
 F(4, 1) valor del cuarto renglón de la expresión  $U(t)$  en el instante  $t_1$   
 F(5, 1) valor del quinto renglón de la expresión  $U(t)$  en el instante  $t_1$

Para el programa principal:

N cantidad de ecuaciones diferenciales  
 N cantidad de subintervalos en que se divide el intervalo de integración  
 NS cantidad de salidas del sistema  
 NU cantidad de entradas del sistema  
 PERZO intervalo de integración

DELT magnitud de los subintervalos de integración

A(I, J) matriz A del sistema de ecuaciones  
 B(I, J) matriz B del sistema de ecuaciones  
 C(I, J) matriz C del sistema de ecuaciones  
 D(I, J) matriz D del sistema de ecuaciones  
 X(I, 1) condición inicial de la variable independiente  
 X(I, 2) condiciones iniciales para cada una de las variables dependientes,  $J > 1$   
 X(I, J) solución del sistema de ecuaciones  
 Y(I, J) valor de las salidas del sistema  
 SUMA(I, J) integral del término constante de la ecuación de la recta empleada para aproximar la entrada  
 RECTA(I, J) integral del término variable de la ecuación de la recta empleada para aproximar la entrada  
 SHOW(I, J) solución del sistema debida a las excitaciones externas  
 Y(I, J) valor de la entrada en el instante  $t_{j-1}$   
 PEND(I, J) pendiente de la recta empleada para aproximar la entrada  
 RE(I, J) variable de reemplazo  
 FI(I, J) variable de reemplazo

e) Dimensiones:

La proposición DIMENSION del programa principal y de las subrutinas deberá modificarse cuando:  
 $N > 100$  y/o  $M > 5$  y/o  $NS > 3$  y/o  $NU > 3$   
 Si se modifica la extensión de N, deberán modificarse los argumentos de la subrutina EXCITA.

d) Formatos para los datos de entrada:

SEC.	TARJETAS	FORMATO	INFORMACION
1		[415, F10.0]	N, M, NS, NU, PERZO
2		[6F10.0]	A(I, J), los elementos de la matriz se dan renglón por renglón. Emplear tan-

las tarjetas como sean ne-  
cesarias

(8F10.0)

B(I,J), igual que para la  
matriz A

(8F10.0)

C(I,J), igual que para la  
matriz A

(8F10.0)

D(I,J), igual que para la  
matriz A

(8F10.0)

X(I,J), el primer valor de  
bè corresponden a la condi-  
ción inicial de la varia-  
ble independiente.

-----  
otros paquetes de datos (opcional)  
-----

TARJETA EN BLANCO, al fina-  
lizar toda la información.

Diagrama de bloques:

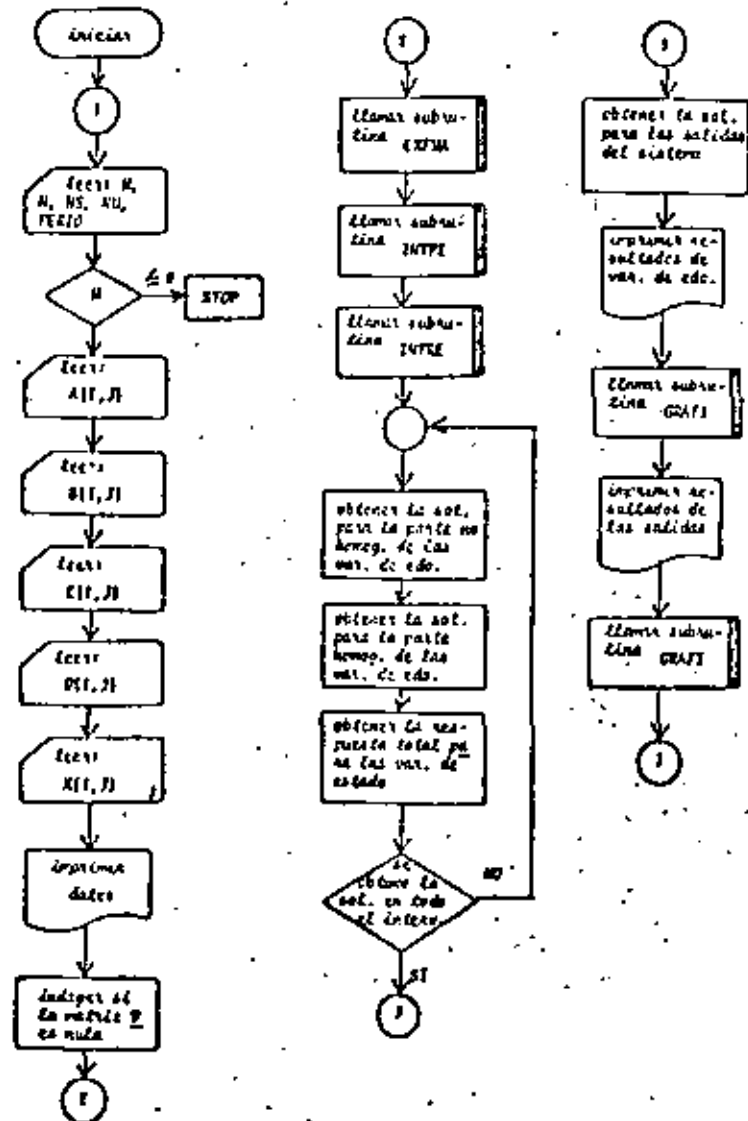


Fig. 10.12 Diagrama de bloques del programa principal

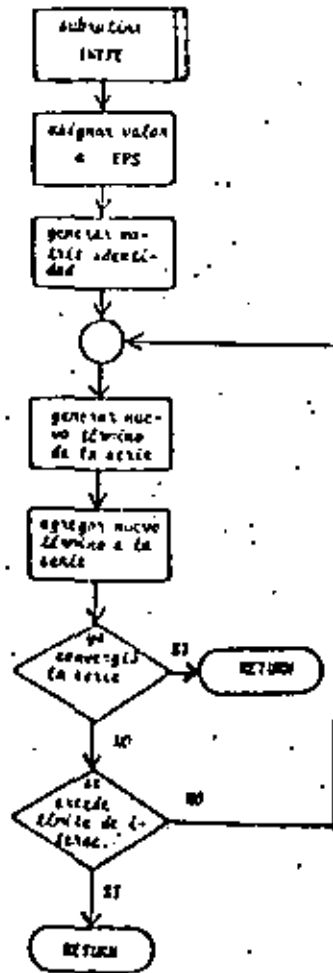


Fig. 10.13 Diagrama de bloques de la subrutina INTPE

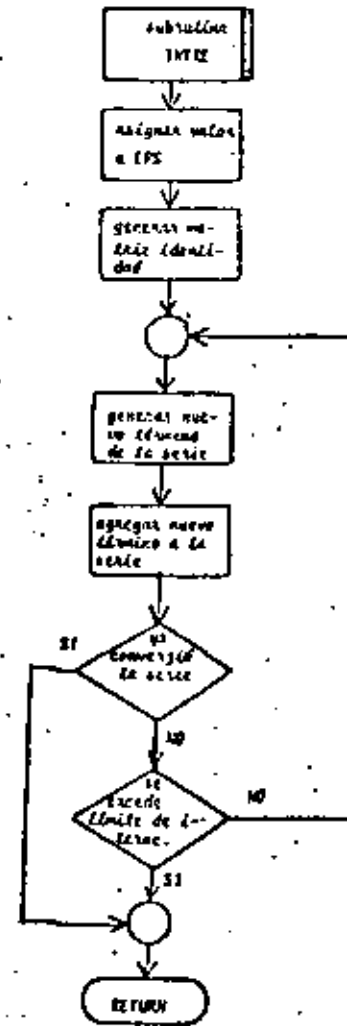


Fig. 10.14 Diagrama de bloques de la subrutina INTRE

6) Listado:

```

C PROGRAM PARA SIMULACION DE SISTEMAS DINAMICOS LINEALES EN UNA COMPUTA
C DORA DIGITAL MEDIANTE EL METODO DE VARIACION DE PARAMETROS
C SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C RESOLUCION DE ECUACIONES DIFERENCIALES
C RESOLUCION DE PARTES DE UNO DEL SUDDIVIC. EL INTERVALO DE INTEGRAC
C ION
C RESOLUCION DE SALIDAS
C FRECUENCIA DE ENTRADAS
C PERIODICIDAD DEL INTERVALO DE INTEGRACION
C DELIMITACION DE LOS SUBINTERVALOS
C MATRIZ A DEL SISTEMA DE ECUACIONES
C MATRIZ B DEL SISTEMA DE ECUACIONES
C MATRIZ C DEL SISTEMA DE ECUACIONES
C MATRIZ D DEL SISTEMA DE ECUACIONES
C X(1,1) CONDICION INICIAL DE LA VARIABLE INDEPENDIENTE
C X(1,2) CONDICIONES INICIALES
C RESOLUCION PARA LAS VARIABLES DE ESTADO
C VALORES DE LAS SALIDAS
C SIMULACION PARA LA RESPUESTA A LAS EXCITACIONES EXTERNAS
C MATRIZ DE TRANSICION
C SUMA INTEGRAL DE LA PARTE CONSTANTE DE LA ECUACION DE LA RECTA
C RECTA INTEGRAL DEL TERMINO VARIABLE DE LA ECUACION DE LA RECTA
C VALOR DE LA ENTRADA EN EL INSTANTE T(1)
C VALOR DE LA ENTRADA EN EL INSTANTE T(2)
C PENDIENTE EN LA RECTA
C RESOLUCION DE COLUMNAS DEL ARREGLO MATRICIAL X
C MATRIZ DE REEMPLAZO
C
C D(ENACION A(4,4),B(6,6),C(6,6),D(6,6),T(10),6),EXP(6,6),RECTA(6,6
C ),SUMA(6,6),T(10),6),T(6,6),RECTO(6,6),RE(6,6),F(6,6),T(10),6)
C
C
C LECTURA DE DATOS
C READ(10,100) (M,N),NU,PERIO
C IF(1) 2,2,3
C CALL T(1)
C MPI=+1
C DO 4 1,1,M
C READ(10,150) (A(1,1),J,1,NU)
C DO 5 1,1,N
C READ(10,150) (C(1,1),J,1,NU)
C IF(1) 6,6,1 GO TO 7
C DO 6 1,1,N
C READ(10,150) (D(1,1),J,1,NU)
C DO 7 1,1,N
C READ(10,150) (X(1,1),J,1,NU)
C 7) READ(10,150) (X(1,1),J,1,NU)
C IMPRESION DE DATOS
C WRITE(10,200)
C DO 8 1,1,M
C WRITE(10,250) (A(1,1),J,1,NU)
C WRITE(10,300)
C DO 9 1,1,N
C WRITE(10,250) (C(1,1),J,1,NU)
C IF(1) 7,7,1 GO TO 7
C WRITE(10,350)
C DO 10 1,1,N
C WRITE(10,250) (D(1,1),J,1,NU)
C WRITE(10,400)
C DO 11 1,1,N
C WRITE(10,250) (X(1,1),J,1,NU)
C 72) WRITE(10,450)
C WRITE(10,250) (X(1,1),J,1,NU)
C IF(1) 7,7,1 GO TO 7
C IMPRESION DE LA MATRIZ D EN FORMA
C DO 12 1,1,N
C WRITE(10,500) (D(1,1),J,1,NU)
C DO 12 1,1,N
C IF(1) 7,7,1 GO TO 12
C 10)
C GO TO 13
C 12) CONTINUE
C 73) IMPRO
C IMPRESION DE LA MATRIZ DE TRANSICION
C 13) DELIMITACION DE LOS SUBINTERVALOS
C CALL EXP(6,6),RECTA(6,6),SUMA(6,6)
C RECTO(6,6),RE(6,6),F(6,6)
C CALL IMPRES(6,6),RECTA(6,6)

```



```

SUBROUTINE INTPE(DELTA, A, N, EPS)
C
C SUMATORIA PARA EVALUAR LA INTEGRAL DEL TERMINO CONSTANTE DE LA
C RESTA
C SIGNIFICADO DE LAS VARIABLES EMPLEADAS
C DELTA=ESPACIAMIENTO ENTRE LOS VALORES DE LA VARIABLE INDEPENDIENTE
C MECANIDAD DE ECUACIONES DIFERENCIALES
C MATRIZ A DEL SISTEMA DE ECUACIONES
C SUMAMATRIZ RESULTANTE DE EVALUAR LA INTEGRAL
C ESPACIAMIENTO DE CONVERGENCIA
C MATRIZ IDENTIDAD
C CANTIDAD DE ITERACIONES
C DIFERENCIAL DIVISOR
C INICIAL DEL TIEMPO ELEVADO A LA POTENCIA M
C MATRIZ A ELEVADA A LA POTENCIA M
C MATRIZ RESULTANTE DEL PRODUCTO AB
C DIMENSION (6,6), C(6,6), B(6,6), X(6,6), SUMA(6,6)
C EPS=0.0001
C
C GENERAR MATRIZ IDENTIDAD
DO 3 I=1,6
DO 2 J=1,6
IF(I, EQ, J) GO TO 1
C(I, J)=0.0
GO TO 2
1 C(I, J)=1.0
2 CONTINUE
3 CONTINUE
C OBTENER LOS DOS PRIMEROS TERMINOS DE LA SERIE
DO 4 I=1,6
DO 4 J=1,6
SUMA(I, J)=C(I, J)*DELTA + (A(I, J)*DELTA+DELTA)/2.0
B(I, J)=A(I, J)
THC=DELTA*DELTA
C(I)=0
DIV=2.0
C OBTENER LOS TERMINOS RESTANTES DE LA SERIE
5 DIV=DIV*DELTA
CALL MULT(A, B, M, N, X)
DO 6 I=1,6
DO 6 J=1,6
B(I, J)=X(I, J)
6 C(I, J)=X(I, J)+THC/DIV
IF(C(I)=0.0) 9, 9, 13
13 AMAX=C(I, 1)
DO 7 I=1,6
DO 7 J=1,6
IF(ABS(C(I, J))-ABS(AMAX)) 7, 7, 14
14 AMAX=C(I, J)
7 CONTINUE
8 CONTINUE
C VERIFICAR LA CONVERGENCIA DE LA SERIE
IF(ABS(AMAX)-EPS) 11, 11, 9
9 DO 10 I=1,6
DO 10 J=1,6
SUMA(I, J)=SUMA(I, J) + C(I, J)
IF(C(I)=0.0) 15, 15, 11
15 C(I)=0.0
GO TO 5
11 RETURN
END

```

Fig. 10.16 Listado de la subrutina INTPE



```

SUBROUTINE INTRE(DELTA, A, RECTA)
C SUBROUTINA PARA EVALUAR LA INTEGRAL DE LA PARTE VARIABLE DE LA EN-
C SUSION DE LA SERIE
C SIGNIFICACION DE LAS VARIABLES EMPLEADAS
C DELIMITACION ENTRE LOS VALORES DE LA VARIABLE INDEPENDIENTE
C DELIMITACION DE ECUACIONES DIFERENCIALES
C MATRIZ A DEL SISTEMA DE ECUACIONES
C MATRIZ RESULTANTE DE EVALUAR LA SERIE
C MATRIZ IDENTIDAD
C MATRIZ A LLEVADA A LA POTENCIA N
C MATRIZ RESULTANTE DEL PRODUCTO AN
C ESPERAMIENTO DE CONVERGENCIA
C INCREMENTO DE LA VARIABLE INDEPENDIENTE ELEVADO A LA POTENCIA
C N
C CONTADOR DE ITERACIONES
C DIMENSIONAL DIVISOR
DIMENSION A(6,6), RECTA(6,6), CHA(6,6), D(6,6), X(6,6)
EPS=0.00001
C COMPARAR MATRIZ IDENTIDAD
DO 3 J=1,6
DO 2 J=1,6
IF(C(1,6), J) GO TO 1
C=C(1, J)+0.0
GO TO 2
1 CONTINUE
2 CONTINUE
3 CONTINUE
C OBTENER LOS DOS PRIMEROS TERMINOS DE LA SERIE
DO 4 J=1,6
DO 4 J=1,6
RECTA(1, J)=(CHA(1, J)+DELTA*DELTA)/2.0 + (A(1, J)+DELTA*A(1, J))/6.0
4 RECTA(2, J)=A(1, J)
Y=DELTA*DELTA*DELTA
C=0.0
DIV=0.0
C OBTENER EL RESTO DE LOS TERMINOS DE LA SERIE
5 DIV=DIV+CH
Y=DELTA*DELTA
CALL MULTPL(A, D, CH, M, X)
DO 6 J=1,6
DO 6 J=1,6
RECTA(2, J)=X(1, J)
6 CHA(1, J)=(RECTA(1, J)+Y*CH)/DIV
IF(ABS(LE, 0.0) GO TO 9
A=CHA(1, J)
DO 8 J=1,6
DO 7 J=1,6
IF(ABS(CHA(1, J)), LE, ABS(MAX)) GO TO 7
A=CHA(1, J)
7 CONTINUE
8 CONTINUE
C VERIFICAR CONVERGENCIA DE LA SERIE
IF(ABS(A*MAX), LE, EPS) GO TO 11
9 DO 10 J=1,6
DO 10 J=1,6
10 RECTA(1, J)=RECTA(1, J) + CHA(1, J)
IF(CM, GT, 20.1 GO TO 11
CM=CM + 1.0
GO TO 5
11 RETURN
END
    
```

Fig. 10.17 Listado de la subrutina INTRE

```

SUBROUTINE EXCITA(F)
DIMENSION F(6,6)
F(1,1)=5.4EXP(-0.1*PI)
F(2,1)=0.5*INTN(1,=PI)
F(3,1)=0.0
F(4,1)=0.0
F(5,1)=0.0
RETURN
END
    
```

Fig. 10.18 Listado de la subrutina EXCITA

Para el siguiente circuito eléctrico:

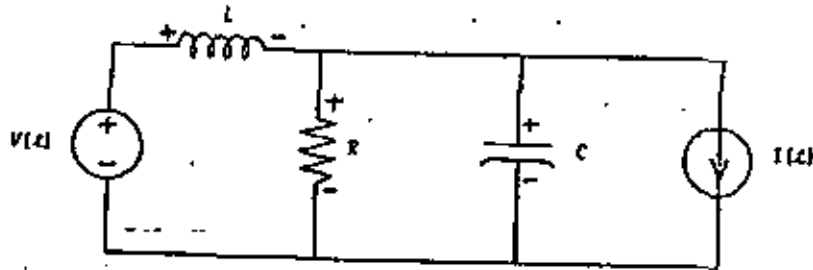


Fig. 10.19 Circuito eléctrico del problema del ejemplo 10.4.4

si se consideran como salidas  $I_c$  y  $V_R$ , su representación mediante variables de estado es:

$$\frac{d}{dt} \begin{bmatrix} I_L \\ V_C \end{bmatrix} = \begin{bmatrix} 0 & -1/L \\ 1/C & -1/RC \end{bmatrix} \begin{bmatrix} I_L \\ V_C \end{bmatrix} + \begin{bmatrix} 1/L & 0 \\ 0 & -1/C \end{bmatrix} \begin{bmatrix} V(t) \\ I(t) \end{bmatrix}$$

$$\begin{bmatrix} I_c \\ V_R \end{bmatrix} = \begin{bmatrix} 1 & -1/R \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I_L \\ V_C \end{bmatrix} + \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} V(t) \\ I(t) \end{bmatrix}$$

Determine los valores de  $I_L$ ,  $V_C$ ,  $I_c$  y  $V_R$  para  $t \geq 0$ , cuando

don

$$\begin{aligned} V(t) &= 5e^{-0.2t} u_{-1}(t) \text{ (V)} \\ I(t) &= 0.5e^{10t} u_{-1}(t) \text{ (A)} \\ R &= 100 \text{ ohms} \\ C &= 0.1 \text{ F} \\ L &= 1.0 \text{ H} \\ I_0 &= 0 \\ V_C(t_0) &= 2 \text{ V} \\ I_L(t_0) &= 0.3 \text{ A} \\ t_0 &= 10 \text{ s} \end{aligned}$$

## SOLUCION

TABLA 10.5 Datos para el problema del ejemplo 10.4.4

$N=2$   
 $M=100$   
 $NS=2$   
 $NU=2$   
 $PERTO=10$

$$A = \begin{bmatrix} 0 & -1 \\ 10 & -0.1 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 0 \\ 0 & -10 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & -0.01 \\ 0 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix}$$

$$X(1,1) = (0, 0.3, 2)$$

$$F(1,1) = 5 \cdot \text{EXP}[-0.2T]$$

$$F(2,1) = 0.5 * \text{SIN}(3.0 * T)$$

$$F(3,1) = 0.$$

$$F(4,1) = 0.$$

$$F(5,1) = 0.$$

TABLA 10.6 Resultados del problema del ejemplo 10.4.4

LA SOLUCION PARA LAS VARIABLES DE ESTADO EN  
TIEMPO x(1) x(2)

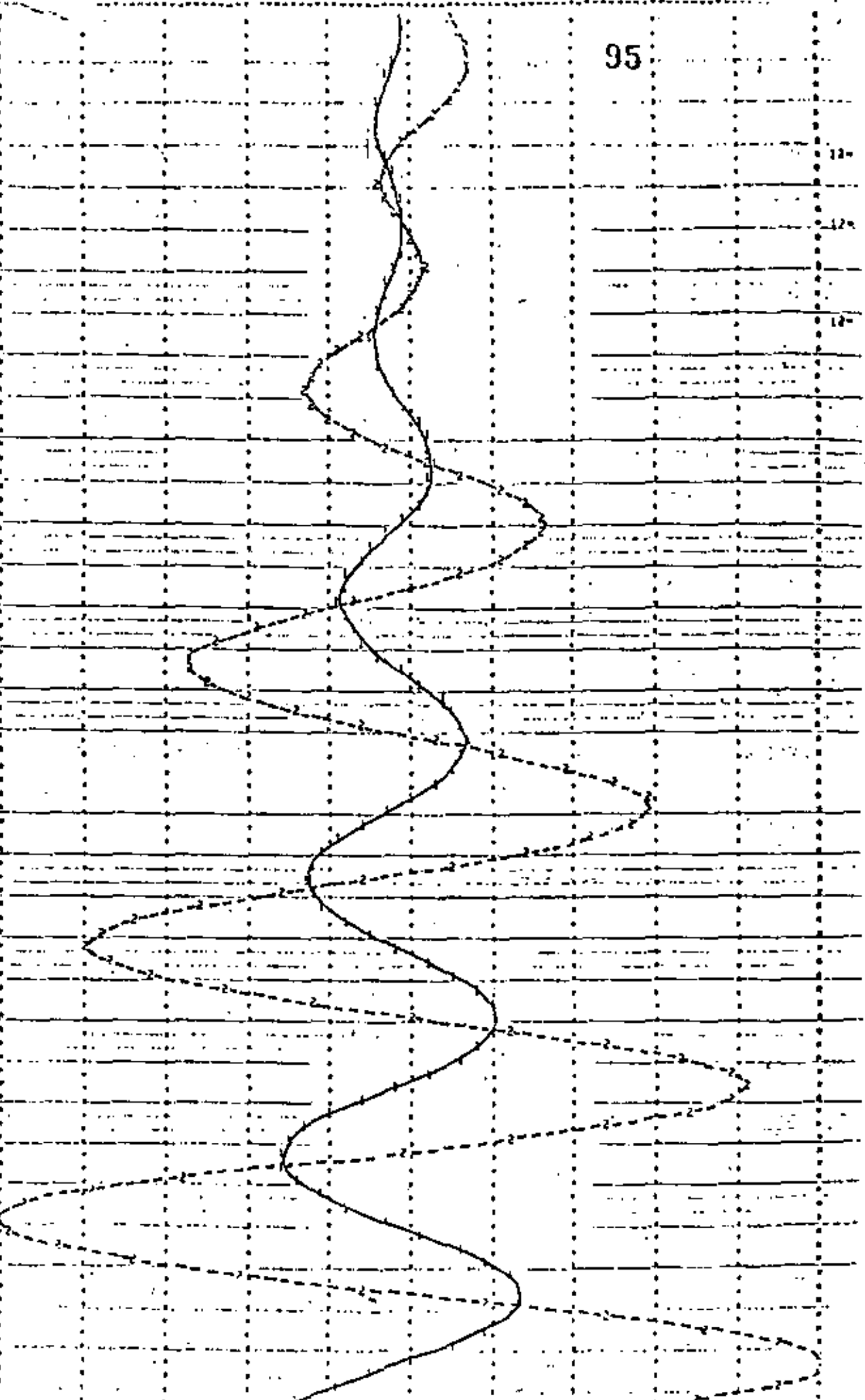
0.	,30000E+00	,20000E+01
,10000E+00	,50191E+00	,23220E+01
,20000E+00	,51425E+00	,25119E+01
,30000E+00	,45225E+00	,27405E+01
,40000E+00	,30903E+00	,20745E+01
,50000E+00	,13140E+00	,23099E+01
,60000E+00	-,25073E+01	,14208E+01
,70000E+00	-,15241E+00	,13500E+01
,80000E+00	-,21291E+00	,76953E+00
,90000E+00	-,2440E+00	,23115E+00
,10000E+01	-,25499E+00	-,17406E+00
,11000E+01	-,21483E+00	-,42943E+00
,12000E+01	-,14774E+00	-,44242E+00
,13000E+01	-,12577E+00	-,30147E+00
,14000E+01	-,10431E+00	-,23550E+01
,15000E+01	-,11985E+00	,32725E+00
,16000E+01	-,10355E+00	,07425E+00
,17000E+01	-,25022E+00	,93023E+00
,18000E+01	-,35072E+00	,10549E+01
,19000E+01	-,45171E+00	,07213E+00
,20000E+01	-,53738E+00	,07331E+00
,21000E+01	-,50102E+00	,17120E+00
,22000E+01	-,50012E+00	-,49049E+00
,23000E+01	-,48017E+00	-,12310E+01
,24000E+01	-,31999E+00	-,14051E+01
,25000E+01	-,01039E+01	-,25059E+01
,26000E+01	,17019E+00	-,20952E+01
,27000E+01	,49010E+00	-,51137E+01
⋮	⋮	⋮
,98000E+01	,13221E+01	-,13020E+02
,99000E+01	,20330E+01	-,12194E+02
,90000E+01	,37713E+01	-,93730E+01
,91000E+01	,47711E+01	-,54142E+01
,92000E+01	,43219E+01	-,12545E+01
,93000E+01	,47215E+01	,32940E+01
,94000E+01	,47212E+01	,70104E+01
,95000E+01	,32209E+01	,11205E+02
,96000E+01	,19080E+01	,13940E+02
,97000E+01	,47114E+01	,20733E+01
,98000E+01	,11497E+01	,15297E+02
,99000E+01	-,07450E+01	,13730E+02
,200000E+01	-,37110E+01	,10944E+02



EL VALOR DE LAS SALIDAS ES  
 JIC'PO ..... Y(1) ..... Y(2) .....

3	23020E+00	20020E+01
10330E+00	23020E+00	23270E+01
20330E+00	22520E+00	24110E+01
30330E+00	33145E+01	27455E+01
40330E+00	13773E+01	26745E+01
50330E+00	14002E+02	23909E+01
60330E+00	53147E+01	19200E+01
70000E+00	59757E+00	13500E+01
80030E+00	57828E+00	76451E+00
90030E+00	43080E+00	23115E+01
10000E+01	32361E+00	17400E+00
11000E+01	13392E+00	47443E+00
12000E+01	57922E+01	44242E+00
13000E+01	22113E+00	30147E+00
14000E+01	32072E+00	23550E+01
15000E+01	36559E+00	37723E+00
16000E+01	32220E+00	67420E+00
17000E+01	70350E+00	93971E+00
18000E+01	25329E+01	12594E+01
19000E+01	18739E+00	97213E+00
20000E+01	240041E+00	67331E+00
21000E+01	59110E+00	12120E+00
22000E+01	71760E+00	45840E+00
23000E+01	75727E+00	12110E+01
24000E+01	69718E+00	10051E+01
25000E+01	55419E+00	29359E+01
26000E+01	27413E+00	21572E+01
27000E+01	94527E+01	31137E+01
28000E+01	92323E+00	24477E+01
29000E+01	75550E+00	21033E+01
30000E+01	10500E+01	13102E+01
31000E+01	12729E+01	21039E+00
.	.	.
32000E+01	45247E+01	17545E+01
33000E+01	45051E+01	32904E+01
34000E+01	40902E+01	36167E+01
35000E+01	32103E+01	11205E+02
36000E+01	20022E+01	13700E+02
37000E+01	48707E+00	15330E+02
38000E+01	73110E+01	15287E+02
39000E+01	21700E+01	13708E+02
40000E+02	34007E+01	10944E+02

1.00E+01  
1.20E+01  
1.40E+01  
1.60E+01  
1.80E+01  
2.00E+01  
2.20E+01  
2.40E+01  
2.60E+01  
2.80E+01  
3.00E+01  
3.20E+01  
3.40E+01  
3.60E+01  
3.80E+01  
4.00E+01  
4.20E+01  
4.40E+01  
4.60E+01  
4.80E+01  
5.00E+01  
5.20E+01  
5.40E+01  
5.60E+01  
5.80E+01  
6.00E+01  
6.20E+01  
6.40E+01  
6.60E+01  
6.80E+01  
7.00E+01  
7.20E+01  
7.40E+01  
7.60E+01  
7.80E+01  
8.00E+01  
8.20E+01  
8.40E+01  
8.60E+01  
8.80E+01  
9.00E+01  
9.20E+01  
9.40E+01  
9.60E+01  
9.80E+01  
1.00E+02



100



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

**INTRODUCCION A LA PROGRAMACION  
Y COMPUTACION ELECTRONICA**

**CASOS DE APLICACIONES NO MATEMATICAS**

**OCTUBRE, 1982**



## A P L I C A C I O N E S .

## N O - M A T E M A T I C A S .

1.-	CONCEPTOS BASICOS	TECNICAS BASICAS
2.-	ALGUNOS TIPS	NUMEROS Y CARACTERES G R A F I C A C I O N .
3.-	BUSQUEDAS	BUSQUEDAS EN TABLAS CARACTERES Y PALABRAS
4.-	ACOMODOS.	STACKS Y COLAS.

## R E F E R E N C I A S :

GEREZ, GRIJALVA "EL ENFOQUE DE SISTEMAS" (CAP IV) LIMUSA, MEXICO  
 PFALTZ "COMPUTER DATA STRUCTURES" MC GRAW HILL  
 KNUTH "THE ART OF COMPUTER PROG. VOL I y III ADISION WESCEY  
 FORTS THE, ORGANIC... " TO COMPUTER SCIENCE" JOHN WILEY.

VAHOS A VER CASOS CONCRETOS DE APLICACIONES NO - MATEMATICAS,  
 "SUPONGAMOS" QUE LOS ASISTENTES YA PROGRAMAN EN FORTRAN MUY  
 BIEN

## 1.- USO DE BANDERAS ( FLAGS) Y SWITCHES.

SE UTILIZAN BANDERAS O SWITCHES, PARA DETECTAR UN ESTAD-  
 DO EN ALGUNA PARTE DEL PROGRAMA, LA BANDERA (FLG) PUEDE  
 TENER DOS ESTADOS FALSO (.FALSE.) Y VERDADERO (.TRUE.),  
 ESE ESTADO PUEDE SER SWICHEADO :

```
FLG = .NOT. FLAG
```

Y PODEMOS PROBAR EL ESTADO DE LA BANDERA

```
(TRUE)  
IF ( FLG ) GO TO 111
```

OTRA ALTERNATIVA ES UTILIZAR UNA VARIABLE ENTERA (IFLG).  
 LOS VALORES 1 y 2 SON USADOS Y EL SWITCH PUEDE TOMAR - -  
 CUALQUIERA DE LOS DOS VALORES.

```
I FLG = 3 - I IFLG CON UN GO TO
```

```
GO TO (10, 20) , IFLG.
```

SE PUEDE TAMBIEN PROBAR UN LOOP CON UN CONTADOR INICIA-  
 LIZANDO IFLAG, 1.e. SUPONGAMOS QUE IFLAG = 1,2,3,4,5,  
 1,2,.....

```
I FLG= 0
```

```
I FLAG = IFLG + 1
```

```
IF ( IFLAG.GT. 5) IFLG = 1
```

DO - LOOP'S:

UJ 3

LOS LOOP DEL DO, PUEDEN SER IMPLEMENTADOS DE MANERA MUY EFICIENTE EN FORTRAN, SIN EMBARGO LA VARIABLE DEL INDICE DEBE DE SER ENTERA Y EL INCREMENTO DEBE DE SER POSITIVO

EJEMPLOS: SE REQUIERE CONSTRUIR UN LOOP EN EL CUAL

J TOMA VALORES DE - 43 a + 43

DO 15 I = 1, 67

J = 1 - 44

SI EN UN LOOP J TOMA VALORES ENTRE 16 Y 2 (EL INCREMENTO ES - 2) ESTO SE PUEDE HACER.

DO 13 I = 2, 16, 2

J = 16 - I

UN LOOP QUE REQUIERE UN INDICE REAL A QUE TOMA VALORES

0.1, 0.3, 0.5, ..., 1.0 SE PUEDE ESCRIBIR ASI:

DO 14 I=1, 10

A = FLOAT (I)/ 10.0

EN ESE CASO A SE INICIALIZA CON 0.0 Y 0.1 ES SUMADO CADA VEZ QUE EL CICLO SE EFECTUE.

SI SE QUIERE DOBLAR EL VALOR DE UNA VARIABLE, POR EJEMPLO:

1, 2, 4, 8, 16, 32, 64, 128, 516, ETC EN CADA PASO DEL CICLO

N = 1

DO 33 I = 1, 10

33 N = N \* 2

DE OTRA FORMA:

4

DO 33 I = 1, 10

N = 2 \* N ( I - 1 )

¿ QUE FORMA ES MEJOR ?

EMPAcando NUMEROS.

PARA AHORRAR ESPACIO, ES MUY COMUN GUARDAR VARIOS NUMEROS ENTEROS EN UNA VARIABLE ENTERA, SIEMPRE Y CUANDO ESTOS NUMEROS NO SEA NEGATIVOS.

SI LOS ENTEROS TOMAN VALORES ENTRE 0 Y N EL METODO CONSISTE EN TRATARLOS COMO DIGITOS SUCCESIVOS EN LA BASE N + 1. ESTO SE HACE ASIGNANDO EL PRIMER ENTERO A LA VARIABLE, MULTIPLICANDOLA POR N + 1, SUMANDO EL SEGUNDO ENTERO.

POR EJ. SI LOS ELEMENTOS 1 a 5 DEL VECTOR I DIG VALEN 6, 8, 0, 3, 1, SE PUEDEN "ENPACAR" EN UNA VARIABLE NUM COMO DIGITOS DE UN NUMERO DECIMAL.

NUM = 0

DO 17 I = 2, 5

17 NUM = NUM \* 10 + 10DIG (I)

DESPUES DE EJECUTADO EL PROGRAMA NUM = 68031 (NOTA LO ANTERIOR ES VALIDO PARA NUMEROS BINARIOS EN DONDE NCM=2). HAY QUE TENER PRECAUCION DE NO EXCEDER EL POSITIVO MAXIMO DE LA COMPUTADORA.

PARA DESEMPACAR LOS NUMEROS SE UTILIZA LA FUNCION MOD. EL ULTIMO ENTERO EMPACADO ES EL RESIDUO, CUANDO LA VARIABLE SE DIVIDE ENTRE N + 1

SI NUM = 68031 Y QUEREMOS OBTENER LOS ORIGINALES DE I DIG  
 HACEMOS LO SIGUIENTE:

```
DO 19 I = 1, 5
J = 6 - I
I DIG (J) = MOD (NUM, 10)
19 NUM = NUM/10
```

NOTESE QUE SE UTILIZO UN INCREMENTO DE - 1, PARA DESEMPACAR  
 CORRECTAMENTE ( LIFO )

TRADUCCION DE TABLAS.

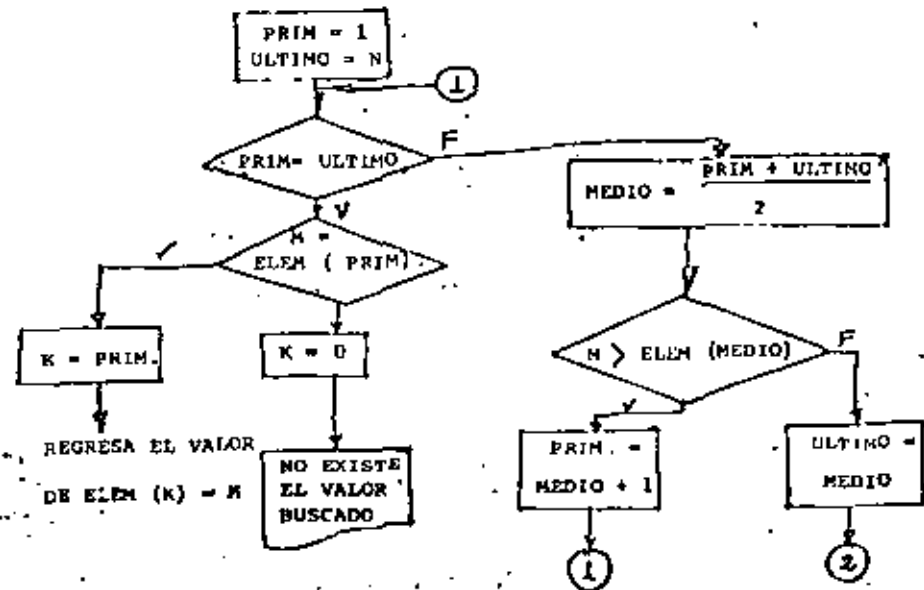
MUCHAS VECES SE NECESITA MAPEAR UN CONJUNTO DE NUMEROS A OTROS,  
 EN FORMA IRREGULAR, LA CUAL NO PUEDE SER COMPUTADA. SI EL PRI-  
 MER CONJUNTO DE NUMEROS CONTIENE ENTEROS, ESTOS SE PUEDEN TOMAR  
 COMO INDICES DE UN VECTOR QUE CONTIENE EL SEGUNDO CONJUNTO DE  
 NUMEROS.

SUPONGA QUE MES CONTIENE ENTEROS ENTRE 1 Y 12 REPRESENTANDO  
 CADA MES DEL AÑO. N DIAS CONTIENE EL NUMERO DE DIAS EN EL MES.  
 SI SE TIENE UN VECTOR LMES DE LONGITUD 12, SE PUEDE INICIALIZAR  
 CON EL VERBO

DATA LMES / 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /  
 Y EL NUMERO DE DIAS PUEDE SER COLOCADO EN N DIAS

N DIAS = LMES ( MES )

LO CUAL ES MUCHO MAS EFICIENTE QUE IR A TRAVES DE 12 "IFS", O  
 UN GO TO COMPUTADO CON 12 RAMAS.



BUSQUEDA BINARIA.

ELEM ( 1 ) = XXX  
 ELEM ( 2 ) = XXX

SI  
 ELEM ( K ) = N  
 N ESTA EN EL ARRÉGLO.

SUPONGAMOS QUE SE NECESITA UN LOOP EN EL CUAL EL INDICE TOMA VALORES DE 1.5, 2.0, 2.5 ..... 4.0 Y LUEGO <sup>5</sup> 4.25, 4.5, 4.75 (660) 11.5 Y LUEGO 13.0, 14.5, 16.0 Y POR ULTIMO 20. (9.5)

LO ANTERIOR REQUIERE 5 INCREMENTOS DE 0.5, SEIS DE 1.25 TRES DE 1.5 Y UN INCREMENTO DE 4.0. PARA HACER LO DESCRITO EN FORMA ELEMENTAL GUARDAMOS EL INCREMENTO EN UN VECTOR Y EN OTRO EL NUMERO DE VECES QUE HAY QUE HACER EL INCREMENTO.

```

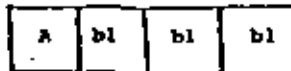
DIMENSION AINC (4), NINC (4)
DATA A INC / 0.5, 1.25, 1.5, 4.0/, NINC
C 4 / 5, 6, 3, 2/
A = 1.5
DO 1 I = 1, 4
J = NINC (I)
DO 2 K = 1, J
2 A = A + ZINC
1 CONTINUE

```

**CARACTERES**

EN FORTRAN LOS CARACTERES PUEDEN SER ASIGNADOS A VARIABLES DE CUALQUIER TIPO. (EL NUMERO DE CARACTERES Y EL # DE BITS EN UNA PALABRA VARIAN DE MAQUINA A MAQUINA ) IBM 1130 ES DE 4.

SI UNA PALABRA PUEDE TENER 4 CARACTERES Y SE GUARDA EL CARACTER A ESTO LUCE ASI:



EN FORTRAN SE PUEDE ASIGNAR CARACTERES A VARIABLES DE 3 FORMAS:

- 1.- LOS CARACTERES PUEDEN SER LEIDOS CON FORMATO A B
- 2.- UTILIZANDO EL VERBO DATA CON UNA CONSTANTE HOLLERITH.

```

DATA I / 2 H A B /
DATA I A ST / 1 H * /

```

- 3.- LOS CARACTERES PUEDEN SER COPIADOS HACIENDO UNA ASIGNACION.

```

DATA PUNTO / I H . / DATA (LINC) / 1 H *
L K = PUNTO

```

**BUSQUEDA LINEAL ( SE VERA MAS TARDE CON DETALLE )**

MUCHAS VECES SE DESEA IDENTIFICAR UN VALOR O ELEMENTO DE UNA LISTA. LA FORMA MAS FACIL ES UTILIZAR UNA BUSQUEDA LINEAL. SE GUARDAN LOS VALORES POSIBLES EN UN VECTOR. SE COMPARA CADA ELEMENTO DEL VECTOR CON EL "IDENTIFICADOR" HASTA QUE SE ENCUENTRE.

EN SUPONGA QUE EL VECTOR NUM ES DE LONGITUD 10 Y CONTIENE DIGITOS DECIMAL EN FORMA DE CARACTERES. LO ANTERIOR SE PUEDE HACER CON EL VERBO :

```

DATA NUM / 1 H 0, 1 H 1, 1 H 2, 1 H 3, 1 H 4, 1 H 5, 1 H 6,
1 H 7, 1 H 8, 1 H 9/

```

AHORA SUPONEMOS QUE UN CARACTER FUE LEIDO DE UNA TARJETA CON FORMATO A1 Y FUE COLOCADO EN

LA VARIABLE KAR. LAS SIGUIENTES INSTRUCCIONES DEBERAN DESLUBRIR QUE DIGITO NUMERICO ESTA EN KAR.

```

DO 15 I = 1, 10
  IF ( KAR. EQ. NUM (I) ) GO TO 24.
15 CONTINUE
18 .

```

SI EL CONTROL PASA A 24. I CONTIENE EL INDICE DEL DIGITO NUMERICO IGUAL A KAR. SI EL CONTROL PASA A 18 KAR NO CONTIENE UN DIGITO NUMERICO.

2.- NUMEROS Y CARACTERES.

VAHOS A VER COMO CAMBIAR UNA CADENA DE CARACTERES A UN NUMERO Y VICEVERSA. ESTAS TECNICAS SON NECESARIAS CUANDO SE LEEN NUMEROS EN FORMATO LIBRE. CUANDO SE MANIPULAN FORMATOS EN LA EJECUCION DE PROGRAMAS O CUANDO SE REQUIEREN IMPRIMIR DATOS EN FORMA ESPECIAL PARA LO CUAL FORTRAN NO FUE DISEÑADO.

2.1 CONVERSION DE CARACTERES A ENTEROS.

SI UNA CADENA DE CARACTERES REPRESENTA A UN NUMERO ENTERO. SE PUEDE CONVERTIR A UNA FORMA NUMERICA UTILIZANDO ALGUNA TECNICA YA VISTA. SI EL NUM. TIENE SIGNO, SE DEBE HACER UNA PRUEBA PARA EL SIGNO, CADA DIGITO ES IDENTIFICADO, UTILIZANDO UNA BUSQUEDA LINEAL ( LOS 10 DIGITOS) Y EL RESULTADO PUEDE SER EMPACADO EN UNA VARIABLE ENTERA.

UN NUMERO EN FORMATO LIBRE ES LEIDO. UN ESPACIO DEBE DARSE PARA LEER LOS CARACTERES Y LOS DIGITOS DECIMALES EN LA TARJETA. LOS 2 CARACTERES CON SIGNO

Y EL BLANCO DEBEN PROVERSE. LO ANTERIOR SE HACE CON LOS SIGUIENTES VERBOS:

```

DIMENSION NUM. (10), KAR ( 80 )
DATA NUM / 1 H0, 1 H1, 1 H2, 1 H3, 1 H4, 1 H5, 1 H6,
          1 H7, 1 H8, 1 H9 /
DATA 1 MAS, 1 MEN0, 1 BL / 1 H +, 1 H-, 1 H bl /
AHORA LOS CARACTERES SE LEEN EN KAR.
READ ( 5, 2 ) KAR
2 FORMAT ( 80 A1 )

```

TODOS LOS CARACTERES BLANCOS. DEL INICIO DE LA TARJETA SON IGNORADOS

```

DO 10 I = 1, 80
  IF ( KAR (I). NE. IBL ) GO TO 20
10 CONTINUE

```

SI TERMINA EL LOOP DEL DO QUIERE DECIR QUE LA TARJETA ESTA EN BLANCO. SI EL CONTROL SE VA A 20 KAR (I) CONTIENE EL PRIMER CARACTER ( NO BL ) UNA PRUEBA SE DEBE HACER PARA EL SIGNO, PARA LO CUAL SE GUARDA RECORD EN ISGN, SI HAY UN SIGNO, ENTONCES I SE INCREMENTA EN UNO, ANTES DE QUE EL DIGITO PUEDA SER IDENTIFICADO.

```

20 | ISGN = 1
   | IF ( KAR (I). EQ. 1 MAS ) GO TO 30
   | IF ( KAR (I). EQ. 1 MEN0 ) GO TO 40
   | I SGN = - 1
30 | I = I + 1
   | IF ( I. GT. 80 ) GO TO 999

```

EL CONTROL PASA A 999 SI LA TARJETA TENIA UN SIGNO EN LA COLUMNA 80 AHORA EL VALOR NUMERICO (APARTE DEL SIGNO) PUEDE CONSTRUIRSE EN INT

```

40  INT = 0
    DO 60 J = 1, 80
    KAR = KAR + ( J )
    DO 50 X = 1, 10
    IF ( KAR. NE. NUM ( K ) ) GO TO 50
EL CARACTER ES IGUAL AL DIGITO NUMERICO.
    INT = INT * 10 + X - 1
    GO TO 60
50  CONTINUE
55  GO TO 20
60  CONTINUE
C SI SE ENCONTRO UN MENOS CAMBIAR EL SIGNO
70  IF ( ISGN. LT. 0 ) INT = - INT

```

EL CONTROL PASA A LA ETIQUETA 70, SI EL NUMERO EN LA TARJETA ESTA EN LA COL 80, ( POR LA ETIQUETA 55 ) O SI UN CARACTER NO NUMERICO FUE DETECTADO.

## 2.2.- CONVERSION DE UN ENTERO A UN CARACTER

SE VIERON LAS TECNICAS PARA CONVERTIR UN NUMERO ASIGNADO A UNA VARIABLE ENTERA A UNA CADENA DE CARACTERES. LOS DIGITOS DECIMALES SON DESEMPACADOS DEL ENTERO Y LOS CARACTERES APROPIADOS SON SELECCIONADOS UTILIZANDO UNA TABLA DE TRADUCCION, DE UN VECTOR QUE CONTIENE LOS DIGITOS DECIMALES EN FORMA DE CARACTERES..

EL SIGNO ES VERIFICADO DE ANTEMANO. ( VALOR POSITIVO )

LOS NUMEROS MUY GRANDES SON DIFICILES DE LEER, PARA HACER ESTA LECTURA FACIL, UTILIZAMOS COMAS EJ 1, 3 14, 5 6 2, 1 3 4. LA SIGUIENTE SUBROUTINA CONVIERTE UN NUMERO ENTERO ASIGNADO A IDATUM A UN CONJUNTO DE CARACTERES INCLUYENDO COMAS, EL CUAL ES ASIGNADO A IARRAY, VECTOR DE LONGITUD N. ESTA CADENA DE CARACTERES DEBE DE EMPEZAR EN LA POSICION INIT DEL VECTOR Y SU LONGITUD

Y DEBE DE SER LARGO EL # DE CARACTERES.

```

SUBROUTINE COMAS ( IARRAY, N, INIT, LARGO, IDATUM )
DIMENSION IARRAY ( N ), NUM ( 10 )
DATA NUM / 1H0, 1H1, 1H2, ..... 1 H9/
DATA MENOS, KOMA, IBL, 1AST/ 1H-, 1H., / H, 1 H*/

```

LA POSICION DEL FIN DEL CARACTER DEL VECTOR ES AHORA CALCULADA, LA APROPIADA SECCION DEL VECTOR ES PUESTA CON BLANCO

```

    IFIN = INIT + LARGO - 1
    DO 1 I = INIT, IFIN
1    IARRAY ( I ) = IBL

```

SE GUARDA EL VALOR ABSOLUTO DE DATO CON GUARA.

```

    GUARA = IABS ( I DATO )

```

UN BLOQUE DE 3 DIGITOS ES DESEMPACADO, Y LOS CARACTERES SON GUARDADOS EN IARRAY. IFIN SE UTILIZA PARA GUARDAR UN RECORD DEL SIG. LUGAR DISPONIBLE EN IARRAY, (TRABAJANDO DE DER. A IZQ. SI YA NO HAY DIGITOS A DESEMPACAR IDATUM PASA EL CONTROL A 40, SI NO HAY ESPACIO SUFICIENTE EL CONTROL SE VA A 20.

```

5    NO 10 I = 1, 3
    J = MOD ( KEEP, 10 ) + 1
    IARRAY ( IFIN ) = NUM ( J )
    KEEP = KEEP / 10
    IF ( KEEP. EQ. 10 ) GO TO 40
    IFIN = IFIN - 1
    IF ( IFIN. LT. INIT ) GO TO 40
10   CONTINUE.

```

SI EL LOOP TERMINA NORMALMENTE, 3 DIGITOS SON GUARDADOS EN IARRAY Y YA NO HAY QUE DESEMPACAR. AHORA HAY QUE AGREGAR UNA COMA EL CONTROL REGRESA A 5 PARA COLOCAR MAS (3) DIGITOS

13

```

IARRAY ( IFIN ) = KOMA
IFIN = IFIN - 1
IF ( IFIN. GE. INIT ) GO TO 5

```

SI LA COMA SE COLOCO EN LA ULTIMA POSICION EL CONTROL PASA A LA SIGUIENTE PROPOSICION ( 20 ), SI HAY ERROR IARRAY DEBE CONTENER ASTERISCOS Y REGRESAR EL CONTROL.

```

20 | IFIN = INIT + LENGTH - 1
    | DO 30 I = INIT, IFIN
30 | IARRAY ( I ) = IAST
    | RETURN.

```

EL CONTROL PASA A 40 CUANDO TODOS LOS DIGITOS DECIMALES HAN SIDO DESEMPACADOS. SI EL SIGNO ES POSITIVO EL CONTROL SE REGRESA AL PROGRAMA QUE LLAMO. SI ES NEGATIVO, SE DEBE AGREGAR EL SIGNO, SI ES QUE HAY LOGAR.

```

40 | IF ( IDATUM. GE. 0 ) RETURN
    | IF ( IFIN. LE. INIT ) GO TO 20
    | IARRAY ( IFN - 1 ) = MENOS
    | END.

```

### 3.- GRAFICAS.

MUCHAS VECES SE REQUIERE IMPRIMIR GRAFICAS, Y NO SE TIENE UN CRT O UN PLOTTER. A CONTINUACION VAMOS A VER ALGUNAS TECNICAS PARA GRAFICAR, UTILIZANDO UNA IMPRESORA COMUN.

DE ANTEMANO HAY QUE SABER LAS CARACTERISTICAS DE LA IMPRESORA, NUMERO DE COLUMNAS ( 120 o 132 CARACTERES ) Y EL NUMERO DE LINEAS ( 66 USUALMENTE ). SIN EMBARGO EL TAMAÑO DE GRAFICA PUEDE SER DE CUALQUIER TAMAÑO.

LA TECNICA ESENCIAL A LA GRAFICACION EN UNA IMPRESORA ES LA DE TRANSFORMAR LOS VALORES ( USUALMENTE REALES ) A ENTEROS EN

14

- 1.- MULTIPLIQUE O DIVIDA, EL VALOR POR CUALQUIER CANTIDAD PARA TENER LA ESCALA ( DISPERSION ) DESEADA.
  - 2.- SUME ( O RESTE ) UNA CANTIDAD AL RESULTADO PARA TENER LOS VALORES EN EL RANGO DESEADO.
  - 3.- CONVERTIR LOS RESULTADOS ANTERIORES A UN ENTERO
- HISTOGRAMAS -

LOS HISTOGRAMAS SE PUEDEN CONSTRUIR CON LAS BARRAS HORIZONTALES O VERTICALES.

SUPONGAMOS QUE TENEMOS UN VECTOR MARCA ( 10 ) QUE CONTIENE ENTEROS POSITIVOS Y SE QUIEREN GRAFICAR COMO HISTOGRAMA.

```

DATA IAST / I H * /
DO 5 I = 1, 10
M = MARCA ( I )
5 WRITE ( 6, 4 ) ( IAST, J = 1, M )
4 FORMAT ( IX, 50 A 1 )

```

```

***
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

SE OBTENDRA UN RESULTADO ASI

NOTESE QUE  $1 \leq \text{MARCA} \leq 50$   
DE CLASE

SI SE TIENEN RESULTADOS DEMASIADO GRANDES, ESTOS SE PUEDEN ESCALAR DE LA SIGUIENTE FORMA:

- 1.- ESCALAR DEPENDIENDO DEL VALOR MAS GRANDE.
- 2.- SI LA LINEA A IMPRIMIR RESULTO DEMASIADO LARGA, ESCRIBIR UN CARACTER ESPECIAL QUE LO DENOTE.
- 3.- IMPRIMIR ASTERISCOS SI LA IMPRESION NO CABE EN LA HOJA, PARA IDENTIFICACION.





SE VAN A TRAZAR 4 GRAFICAS "1", "2", "3", Y "4"

DIMENSION LINE (100), KAR (4), VAL (4)

DATA IBL/ 1H/, KAR/ 1H1, 1H2, 1H3, 1H4/, LINE/ 100\* 1H/

C EMPEZAR EN HOJA NUEVA

```

11 WRITE (6, 11)
   FORMAT (1 H1)
   DO 10 I = 1, 120
   X = FLOAT (I-1) * 0.09
   VAL (1) = SIN (X)
   VAL (2) = COS (X)
   VAL (3) = VAL (1) + VAL (2)
   VAL (4) = VAL (1) - VAL (2)
   DO 5 J = 1, 4
   IND = VAL (J) * 25.0 + 50.
5   LINE (IND) = KAR (J)
   WRITE (6, 7) LINE
   FORMAT (5, X, 100 A1)
   DO 10 J = 1, 100
10  LINE (J) = IBL
C   DEJA UNA HOJA AL FIN
   WRITE (6, 11)
   STOP
   END

```

(SE HACEN BLANCOS)

NOTESE QUE LOS VALORES SE GUARDAN EN UN VECTOR VAL, CON EL FIN DE SER UTILIZADOS POR EL LOOP DEL DO

" BUSQUEDA BINARIA "

18

COMO EJEMPLO UTILIZAMOS UNA BUSQUEDA BINARIA PARA OBTENER EL NUMERO DE ELEMENTOS DE CADA INTERVALO DE CLASE PARA CONSTRUIR UN HISTOGRAMA.

LA LONGITUD DE TABLA DEBE DE SER  $2^n - 1$ , POR EJEMPLO SI TENEMOS 32 INTERVALOS, HABRA 31 PUNTOS DE SEPARACION.

```

DIMENSION INTER ( 31)
INDX = 0
INC = 16
1  IT = INDX + INC
   IF ( VAL, GT, INTER (IT) ) INDX = I *
   INC = INC / 2
   IF ( INC, GT, 0 ) GO TO 1

```

LOS VALORES DEBEN DE SER ASIGNADOS A INTER QUE MARCA LOS LIMITES DE LOS INTERVALOS, Y ESTOS VALORES DEBEN DE SER ASCENDENTES.

EL VALOR CENTRAL DEL INTERVALO ESTARA EN INTER (16) EL NUEVO VALOR SERA ASIGNADO A VAL. INC TOMA VALORES 16, 8, 4, 2, 1, Y 0. SI VAL ES MAYOR QUE TODOS LOS VALORES EN INTER, ENTONCES INDX TOMA SUCESSIVAMENTE LOS VALORES 0, 16, 24, 28, 30, Y 31, PERO SI VAL < INTER (1), INDX = 0.

EL VALOR FINAL DE INDX ESTARA ENTRE 0 Y 31, DADO -L LOGAR DEL INTERVALO EN EL CUAL CAE VAL

6-1  
Tree examples

The use of flowcharts to represent algorithms has helped us to recognize their underlying structure. Furthermore, attention paid to the structure of an algorithm usually results in a better understanding of the computational process, and often results in our recognizing alternatives and potential improvements to the original design. Similar rewards result from attention paid to the structural relationships among the components of a set of data.

There is, in fact, a close connection between the steps we need to express an algorithm and the way we choose to think about or represent the data that are to be transformed by that algorithm. Experience in constructing algorithms fosters an increased appreciation of this interdependence. You will gain some of this experience by studying the next several chapters. Your ability to analyze the structure of a set of data and how alternate representations of it can affect algorithms using such data, undoubtedly will improve. We have already considered two structural forms for data, lists and arrays. Another type of structure is called a *tree* (Figure 6-1). Tree structures are important in representing certain types of data and, oddly enough, the essential steps of a number of algorithms exhibit a tree-like structure.

First we will give two simple examples of a process whose strategy of execution (algorithm) can be pictured as a tree and two examples of data that can be pictured as a tree. Later, we will tackle three fascinating problems, the first one at the end of this chapter and the other two in Chapter 7. When we have finished this study, we may claim the title *tree expert*.

Let us agree now, before we get too far along, that trees in this chapter will be drawn upside down (Figure 6-2). We

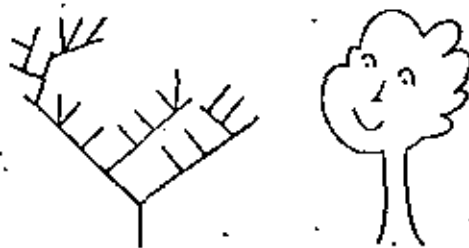


FIGURE 6-1

This is a tree

This is not a tree

do this only because it is convenient. You have to be willing to think of a tree growing toward the earth, its trunk "hanging" from the sky.

Example 1.  
Treelike AlgorithmsFIGURE 6-2  
This is an upside-down tree

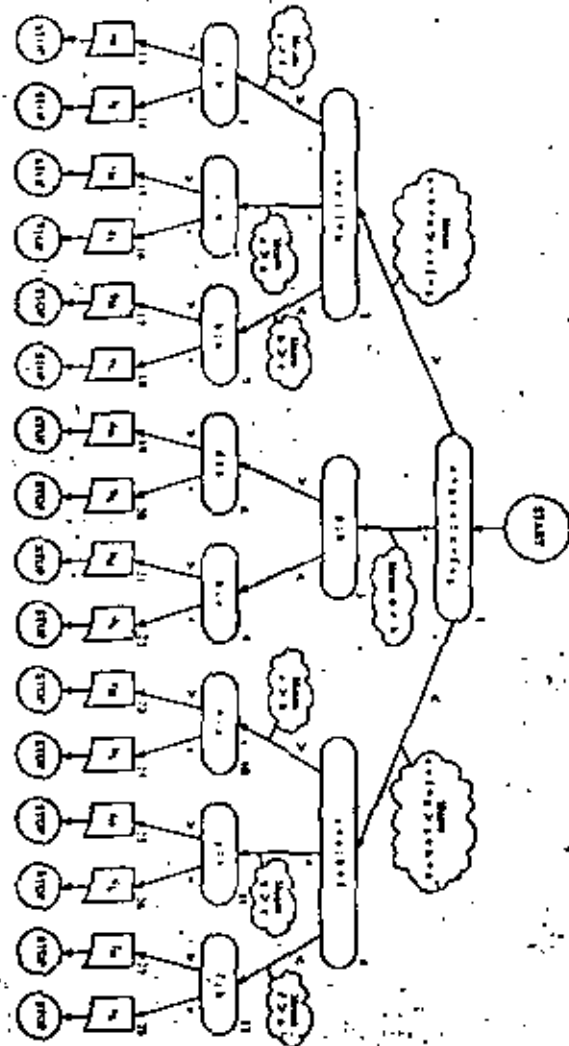
Our first example shows how we can represent the 16 conclusions to the well-known eight-coin problem as a "decision tree." The problem is this. You are given eight coins, *a*, *b*, *c*, *d*, *e*, *f*, *g*, and *h*, and are told that they are all of uniform weight except one, which is either heavier or lighter than the others. You are given an equal arm balance, but you may only use it three times for comparing coins or groups of coins. Your job is to determine the maverick coin and whether it is lighter or heavier than the rest.

Here is a strategy to use (see Figure 6-3) for all possible cases.

1. Compare the weights of two subsets of equal numbers of coins and consider the significance of the three possible outcomes. If the weights of the two subsets are equal, the coin in which we are interested cannot belong to either of the compared subsets.
2. Once we have isolated a pair containing the "odd" coin and we want to know whether one of them is heavy or light, we weigh one of the two candidates against any other that is known to be "standard."

There are 16 possible cases, each of which may occur, given the eight labeled coins. The algorithm shown in Figure 6-3 has a treelike structure. Conclusions are reached by follow-

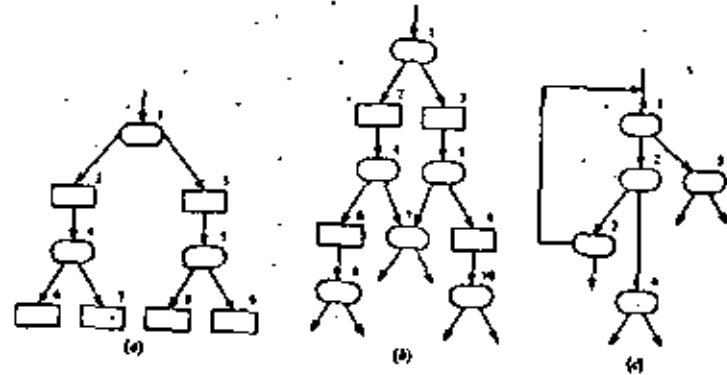
FIGURE 6-3  
 The diagram of a strategy to identify the odd coin.  
 Circles are subsequent weighings. Squares are subsequent decision boxes.



ing a unique path (a sequence of three weighings) from the top or root of the tree diagram to one of the terminal boxes or leaves at the bottom.

EXERCISES 6-1.  
 SET A

1. For decision box 2 in Figure 6-3, explain why:
  - (a) If the relation  $a + c > f + b$  is true, one may conclude that  $a > f$ .
  - (b) If the relation  $a + c = f + b$  is true, one may conclude that  $c > b$ .
  - (c) If the relation  $a + c < f + b$  is true, one may conclude that  $b > c$ .
2. Explain why  $b$  may be regarded as a "standard" coin at decision box 5 but not at decision box 7.
3. Suppose you are given 12 seemingly identical balls and are told that one ball is *heavier* than the others, which are of the same weight. Draw the tree diagram algorithm to identify the heavy ball in three weighings.
4. Suppose you are given 12 seemingly identical balls and are told that one ball is *different* in weight (either heavier or lighter). Draw a tree diagram algorithm to identify the odd ball and to determine whether it is heavier or lighter in three weighings.
5. Are all decision sequences tree structures? Consider the three flowcharts below.



- (a) Which of these are tree structures?
- (b) Consider the following attempts to define a tree structure.

- (1) A node having no segments extending from it is a *terminal node*.
- (2) A node having one or more segments extending from it is a *nonterminal node*.
- (3) A tree structure is a terminal node or a nonterminal node whose segments connect other of terminal nodes or tree structures.

What corrections or additions, if any, are needed in the above definition so that, when applied to flowcharts a, b and c, you will reach the same conclusions that you came to in the answer to part a of the problem?

### Example 2. Game Trees

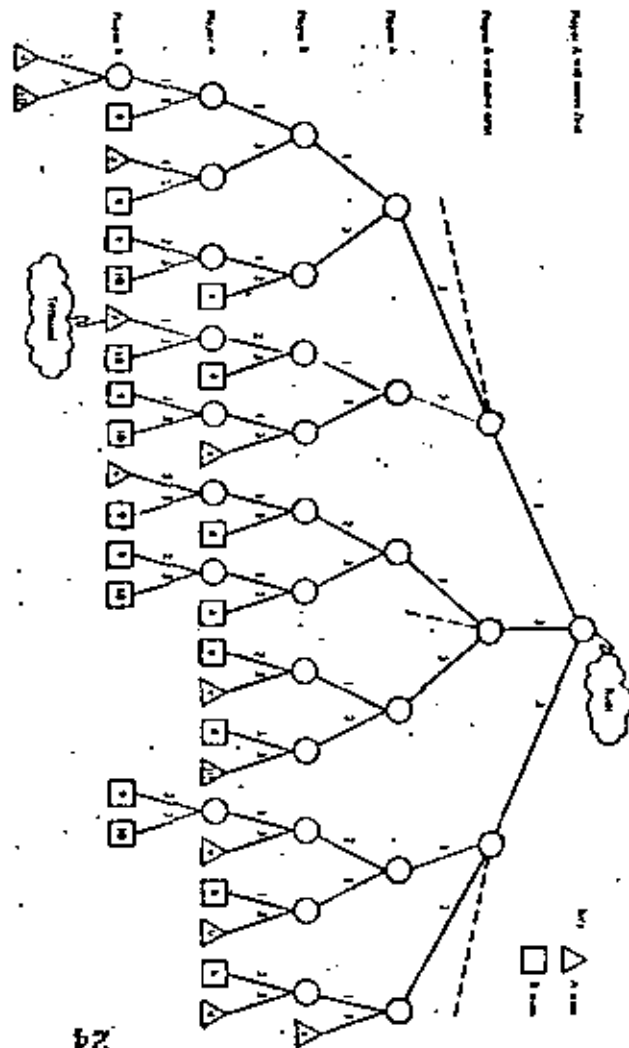
A more interesting type of decision tree, frequently referred to as a *game tree*, shows the moves made by the players. Each time a player makes a move, he selects among the available choices of "legal" moves. Each line segment of the tree represents one choice by one player during the playing of one game. Figure 6-4 is a tree for the game of "Eight." This two-player game is so trivial you may not enjoy playing it very long. Its tree is simple enough, however, that we can study it easily, and it serves as a good illustration of similar but far more complicated games.

### The rules of "Eight"

Each player takes a turn at picking a number from one to three, adding this number to a running sum that is initially set at zero. The first player has a free choice of numbers 1, 2, and 3. The choice in each play thereafter is restricted. A player may not choose the opponent's preceding selection. The player who brings the running sum to a total of exactly eight wins the game; a player exceeding eight loses. There is no draw possible.

When we study the game tree, we can observe that a complete game from start to finish is represented by one path (e.g., the colored line) from the beginning or root of the tree down to an end or terminal point. Player A always moves first. Thus, on the green line, A chooses 1 from among the three initial choices. Then B chooses 3, then A chooses 1, and so forth, until at the last move for A the running sum is 7, and his choices are 1 and 3. So he chooses 1 to make the sum 8 and wins. Triangular-shaped endpoints denote a win for A. Square-shaped endpoints denote a win for B.

FIGURE 6-4  
Tree for the game of Eight.



Each time a player makes a move, you can imagine he looks at three choices 1, 2, and 3. After the very first choice he rules out one of these, as the game rules demand. Inadmissible moves would not ordinarily appear in the tree because they tend to clutter the diagram. We have shown them as dashed lines for B's first move only as a reminder.

EXERCISES 6-1,  
SET B

- L (a) How many distinct games of "Eight" are there?

Imagine the game of Eight is played by children at the local kindergarten in the following way. Player A twirls the arrow of a



three-part spinner, to select the initial move. There-

after, each player flips a coin to decide among the two admissible choices. (heads the smaller, tails the larger.)

- (b) What percentage of games played will follow the color line path in Figure 6-4?
- (c) What are A's chances of winning? Express this result as the number of games won for every 100 games played.
- (d) If each player chooses each move as randomly as possible, what do you think are A's chances of winning if A plays first? The spinner is 6 times out of 100. See if you can develop a proof of this assertion. In Chapter 7 we will take another look at this problem.

In each of the next two exercises, you are given the rules of a simple, two-player game. Your job in each case is to show part or all of the game tree with at least four complete games displayed.

Column

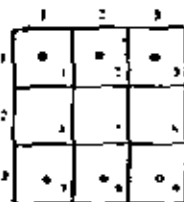


FIGURE 6-5  
Board position at the  
beginning of the game of tic-

2. The game of *tic* (or *Hexapawn*) uses a 3 by 3 checkerboard. Each player begins the game with three pawns on his base line, as shown in Figure 6-5. Play alternates between green and gray. The rules of the game are as follows:

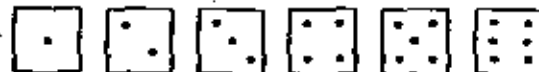
- (a) Either green or gray, in his turn, can move forward one space to an unoccupied position.
- (b) Or he can move diagonally one space to capture an opponent. A captured piece is removed from the board.
- (c) The game is won by reaching the opponent's baseline.
- (d) Or by leaving the opponent without a move.
- (e) Or by capturing all opponent pieces.

*Hint:* Each segment of the tree should be labeled to indicate the move it represents. One way would be to show the *before* and *after* row,

column values of the piece that is moved. For example, on green's first play he has three choices:  $(3,1) \rightarrow (2,1)$ , or  $(3,2) \rightarrow (2,2)$ , or  $(3,3) \rightarrow (2,3)$ . Each of these moves can be further abbreviated to four-digit numbers without loss of information, that is, 3121, 3222, and 3323, respectively.

Alternatively, if we give the squares of the board the explicit names shown as small digits in the lower right corner of each square in Figure 6-5, then we can use a somewhat more compact representation of a move. Instead of four-digit abbreviations (e.g., 3121, 3222, and 3323), we can use two-digit abbreviations (e.g., 71, 83, and 94, respectively) with no risk of confusion.

3. The game of "31." Take a die and roll it. The side that turns up gives the running sum's initial value. Thereafter, each player moves by tilting the die over on its side (one of four possible sides, of course, and remember that opposite faces



always add to seven). The side that turns up after the roll-over is then added to the running sum. The game proceeds tilt after tilt. A player whose tilt brings the total to exactly 31 wins the game; a player exceeding 31 loses. There are no draws.

### Data Trees

Flowcharts of algorithms often have the characteristic treelike structure, but it is also interesting that data can be arranged in a treelike structure. Here are two examples.

### Example 3. Monotone Subsequences

Suppose you are given a sequence (i.e., a list) of  $N$  numbers, all guaranteed to be different. What is the longest monotone subsequence in the given sequence?

By a subsequence we mean the list that remains after "crossing out" some numbers in the original list. If, for example, the given list is

5 0 9 6 1 12 3 7 2,

then one of the 511 ( $2^9 - 1$ ) possible subsequences of this sequence is:

5 0 9 6 1 12 3 7 2

that is, 9 6 12 3 2.

The reason for explaining this idea in terms of "crossing out" is to make it absolutely clear that the order of the remaining terms is not altered. By a *monotone* subsequence we mean one in which either the values are increasing from left to right or one in which they are decreasing. Thus the preceding subsequence,

9 6 12 3 2

is not monotone, but the following two are, the first being increasing and the second decreasing.

3 0 6 6 1 12 3 7 2  
5 0 9 6 1 12 3 7 2

You can check that the increasing subsequence is the longest possible; that is, there is no increasing subsequence with more than four elements. The decreasing one is not the longest possible, since the subsequence

9 6 3 2

is longer.

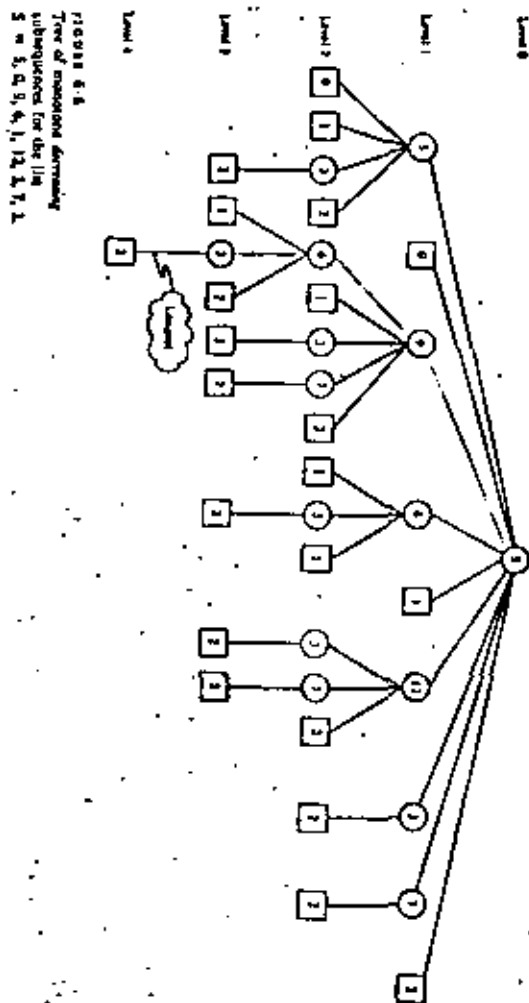
It is possible to develop an algorithm for determining longest monotone subsequences of a given sequence. Our interest here is a bit different. Suppose you are asked to picture all the possible monotone decreasing subsequences of our example sequence.

5 0 9 6 1 12 3 7 2

A hopeless task? Not if we think in terms of trees! See the answer in Figure 6-6.

A most revealing discovery! We have taken a string of numbers, posed a particular problem concerning that string, and discovered that the problem's answer could lie in inspecting a related tree. Notice that every monotone decreasing subsequence in  $S$  can be represented as a path running from the root  $S$  to one of the terminal squares. From now on, we'll call these circles and squares nodes.

The longest of such subsequences is easy for the eye to pick out once the tree is drawn. It is the one whose terminal node is found at a level of the tree farthest from the root node. In this example, only one path reaches to level 4, so there is only one longest monotone decreasing subsequence.



If a computer is to be used for this approach, we must have an algorithm that, in effect, systematically scans the course tree. This is the interesting part, which will be taken up starting with the next section.

EXERCISES 6-1,  
SET C

1. Construct a tree that displays all monotone increasing subsequences of the same sequence given in Figure 6-6.
2. Draw the tree that displays all monotone decreasing subsequences for the sequence  $S$  defined as

$$S = (3, 2, 1, 0)$$

3. Imagine you are a student registering at a university and you have decided to enroll in a particular group of five courses. The five courses, together with the available sections and the times each will be taught, are listed in Figure 6-7. We presume these data are extracted from the official class schedule. Notice that the time periods are letter coded for convenience.

Course	Open Sections	
ENG 132	D (9-10 MWF)	E (10-11 MWF)
	F (11-12 MWF)	
FRE 141	F (11-12 MWF)	H (1-2 MWF)
	Q (10-11:30 TTH)	
HIS 231	F (11-12 MWF)	H (1-2 MWF)
MTH 172	D (9-10 MWF)	F (11-12 MWF)
	Q (10-11:30 TTH)	
CSC 131	F (11-12 MWF)	H (1-2 MWF)

Timetable

Course	Letter Codes for Possible Sets				
	No. 1	No. 2	No. 3	No. 4	No. 5
ENG 132					
FRE 141					
HIS 231					
MTH 172					
CSC 131					

FIGURE 6-7  
Data taken from the printed  
class schedule.

Does a possible set of nonconflicting sections for the five courses exist? That is, is it possible to select a set with no time conflicts? If so, how many distinct feasible sets can be selected? To be distinct, a set need differ in no more than one section from other possible sets. Complete a column of the timetable shown in Figure 6-7 for each feasible set.

*Hint:* This problem and others like it can be solved systematically by constructing a tree of labeled nodes. The structure for the tree could be such that the set of nodes along any path emanating from the root represents a set of nonconflicting course sections. For example, labeled nodes at level 1 could represent the various available sections of ENG 132 (Figure 6-8). Nodes at level 2 could correspond to various sections of FRE 141, and so forth. Any path running from the root to level 5 such that every node has a different label would represent a feasible set of courses.

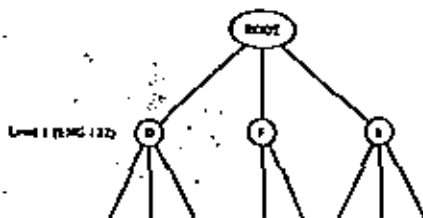


FIGURE 6-8

4. Imagine you are a student registering at a university and assume that you have decided to enroll in the following six courses:

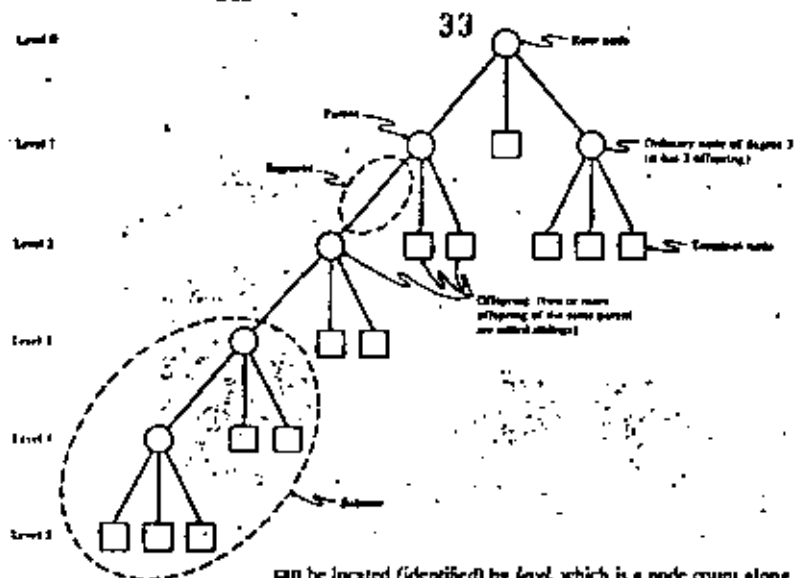
Communications 267	(COM 267)
English 337	(ENG 337)
French 231	(FRE 231)
Geography 233	(GGY 233)
Mathematics 272	(MTH 272)
Music 120A	(MUS 120A)

Below are data taken from the printed class schedule. Imagine that when you reach the registration desk, you find that certain sections of four of the desired courses are closed (as indicated). Prepare a tree diagram that shows whether there are one or more feasible programs open to you at this time, permitting you to enroll in all six of the desired courses with no time conflicts. If one (or more) program(s) is (are) available, prepare a filled-out timetable whose format is similar to that given in Figure 6-7.

5. A student who was planning to work every afternoon (1-5 p.m.) for the Athletics Department was also hoping to enroll in all of the following six courses: COM 267, MUS 120A, GGY 233, MTH 272,



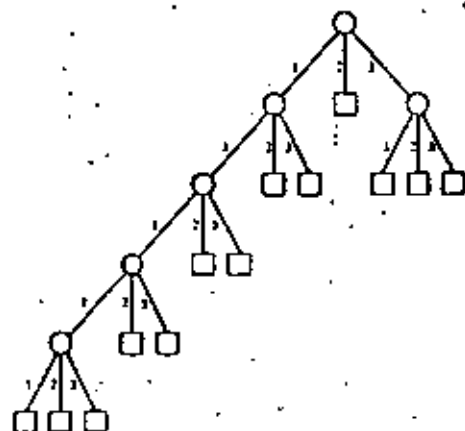


FIGURE 6-10a  
Tree terminology.

can be located (identified) by level, which is a node count along a path from the root node to a terminal. The level we associate with the root node is purely arbitrary, but we will usually take it to be zero. The nodes along a path are often thought of as having an ancestral relationship one to another. By analogy with family trees, moving from a root toward a terminal, each node is the parent of its immediate successor nodes (offspring). Two or more nodes having a common parent are sometimes called siblings. Finally, we can say that the degree of a node is the number of its immediate offspring.

A node may also be identified in terms of the path that leads from the root to that node. How can we represent that path? An answer comes to mind when we realize that representing a tree in two dimensions imposes an ordering on the segments that emanate from each node. And we might as well recognize this fact of life by numbering the segments in some way, say from left to right or right to left. For simplicity and consistency we will generally number segments from left to right, as suggested in Figure 6-10b. These ordinal numbers amount, in effect, to names for the segments.

FIGURE 6-10b



Now a node can be designated uniquely by listing the names of all the segments in the path leading to that node. For example, the light green node in Figure 6-10b may be designated by the list (1, 1, 1, 1, 2), the dark green node by the list (3, 1), the gray node by the one-element list (2), and the black node by (1, 1, 1). (How would the root node be designated in this scheme?) Distinct nodes have distinct paths and hence distinct lists.

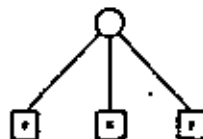
The expression tree, by its very structure, provides the key to evaluating the expression that is represented. For example, suppose values for the variables of our expression are:

a	w	b	d	y
3	2	2	-1	7

A subtree of the form shown in Figure 6-11, together with the above table, can be understood in this way: Lock up the values of  $a$  and  $w$ , compute  $a \times w = 3 \times 2 = 6$ , and replace the subtree by the terminal node 6.

Correspondingly, the subtree shown on the left can be interpreted as: Compute  $d \times y = -1 \times 7 = -7$  and replace the subtree by the terminal node  $-7$ .

Figure 6-12 represents a sequence of meaningful substi-



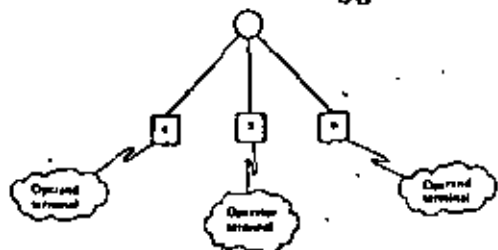


FIGURE 4-11

tations that, when carried out, will ultimately lead to the replacement of the whole tree, root node and all, by a single terminal, which represents the value of the expression.

Proper evaluation is guaranteed if we follow one simple replacement rule that says:

*Whenever you find a subtree consisting of a root node leading to three terminals (an operator and two operands), replace that subtree by a single terminal value.*

Thus the replacement sequence in Figure 6-13, although different from that of Figure 6-12, leads us irrevocably to the

same value,  $\frac{-256}{7}$ . A computer performing either sequence

would evaluate the same indicated quotient  $\frac{-256}{7}$ , notwithstanding

the fact that computer operations on floating-point numbers are nonassociative, a fact explained in Chapter 11. Another point to note from the figures is that the tree-like representation of a complicated arithmetic expression allows us to see all the meaningful subexpressions (all the subtrees) at a glance.

Once an expression is represented as a tree, evaluation depends only on repeatedly searching and finding subtrees that are subject to the replacement rule. At any given time, an expression tree, if not already fully evaluated, will exhibit at least one such subtree.

A question that has no doubt been uppermost in the minds of some readers is: How should tree structures be represented in storage? Linked lists, introduced in Chapter 4, suggest one

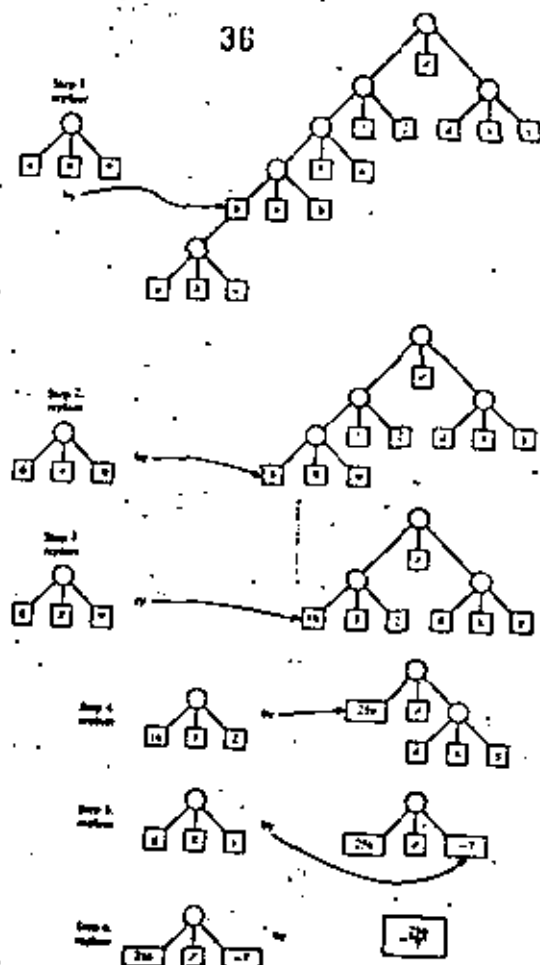


FIGURE 4-12  
A stepwise evaluation of a tree expression.

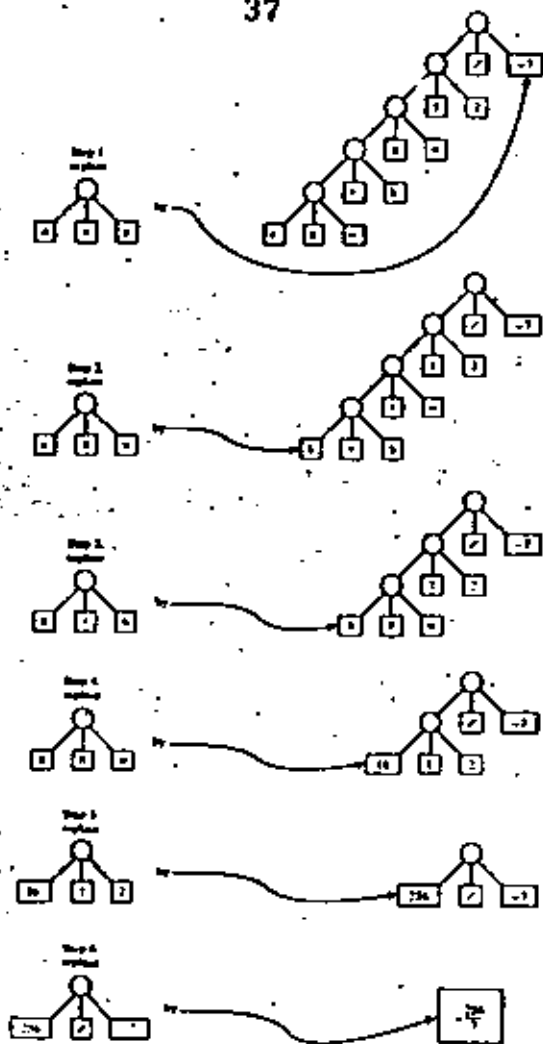


FIGURE 6-13  
Alternative replacement  
sequence for evaluating the  
tree expression.

way. With this approach, one storage representation for the expression

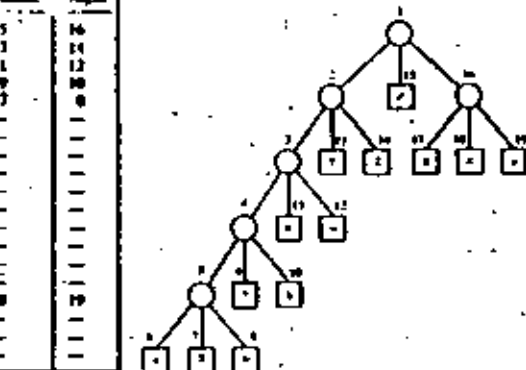
$$((a \times w + b) \times x) / (d \times y)$$

is shown in Figure 6-14. A four-column table is used. Each row represents a node with row 1 representing the root node.

The Table

Node Number	Value (String)	Links		
		Left	Middle	Right
1		2	15	16
2		3	13	14
3		4	11	12
4		5	9	10
5		6	7	8
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				

FIGURE 6-14  
Tabular representation of an  
expression tree.



The first column holds the value of the node if it is a terminal or some special mark, for example,  $\square$ , if the node is non-terminal. The remaining three columns hold node numbers that designate the left, middle, and right offspring. These positions may be left empty (undefined) for terminal nodes. Node numbers in the left, middle, and right columns serve as links to other nodes. Other tabular descriptions of tree structure using the linked list approach are discussed in Chapter 7.

#### A Peek at Some Future Models of SIMPLIS

We noticed in the expression tree of Figure 6-12 that two separate subtrees



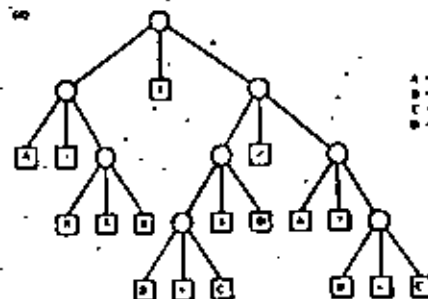
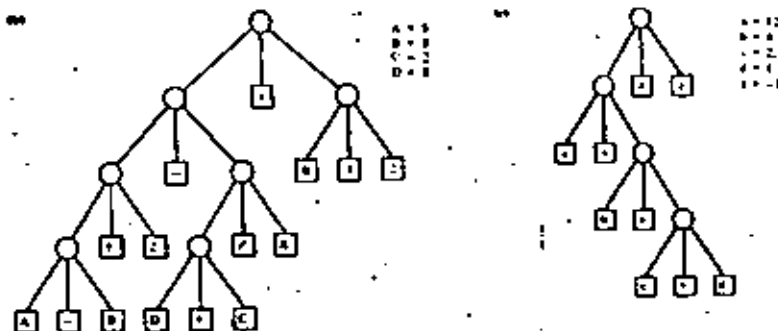
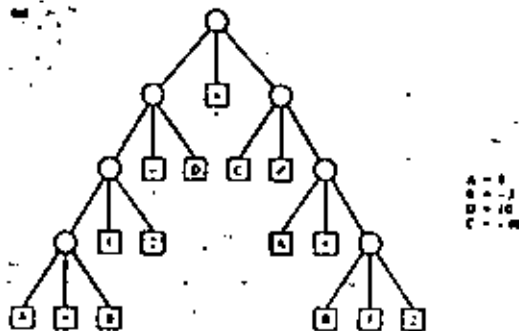
had only terminal nodes as offspring. At the start of evaluation we could invoke the replacement rule on either of these subtrees. It was immaterial which we picked first. This will always be the case for subtrees whose root nodes are siblings or the offspring of sibling nodes (i.e., cousins once or more removed). From the standpoint of expression evaluation, such subtrees are *mutually independent*.

Under what circumstances can a computer work on the evaluation of two or more independent subtrees at the same time? With our present SIMPLIOS model the answer is never, because at any one time there is only one team of personnel (Master Computer, Affixer, Reader, and Assigner) available to do work. On the other hand, advanced models of computer systems having several, perhaps many teams of personnel, are quite feasible.

Although it may boggle the mind to think about it, one may anticipate that future computers will evaluate mutually independent subtrees concurrently, that is, *in parallel*, whenever more than one team of personnel is available for the purpose. In cases where speed is essential the capability of concurrent computation offers the opportunity to solve problems that cannot be solved fast enough in any other way. Examples of such problems already abound in our technological society. More can be found on this topic in advanced texts.

EXERCISES 4-1,  
117 D

1. Evaluate the expression trees below, using the given values for the variables.



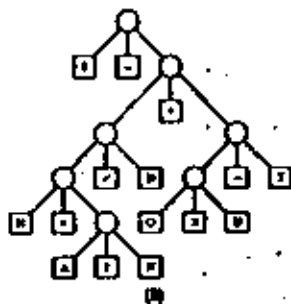
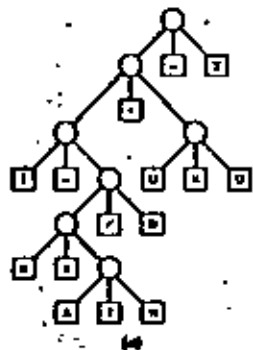
2. Draw a tree representation for the expression

$$B \div 4 \times A \times C$$

3. The following are two proposed tree representations for the expression

$$I - N \times A \div N \div D + O \times U - T$$

Which, if either, of these trees, evaluated by the replacement rule, yields a result computationally equivalent to the result we get by following the evaluation rules laid down in Tables 2-1 and 2-3? If the evaluation of either one of the trees is not compatible with these rules, describe the discrepancy.

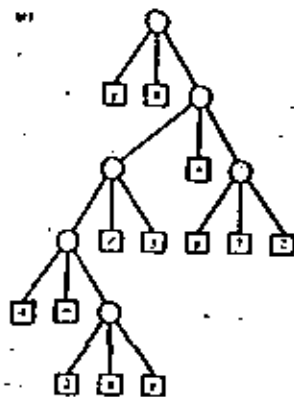
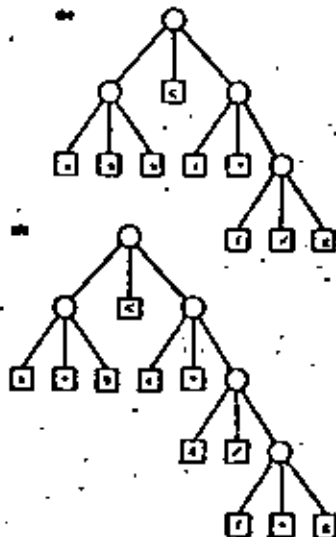
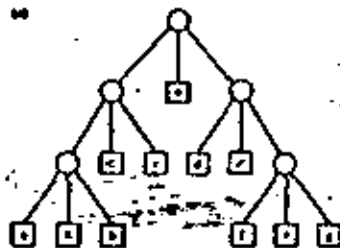
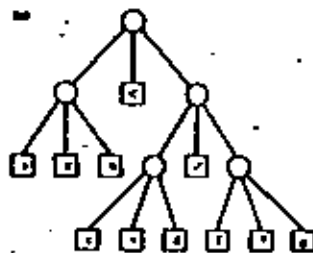
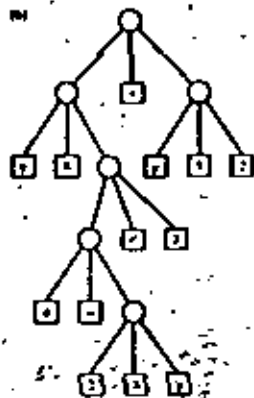
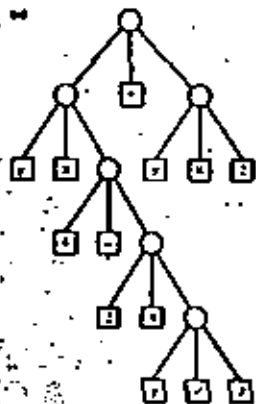


4. Draw a tree representation for the expression

$$(a - b) \times (c - d) / (e \times (f + g))$$

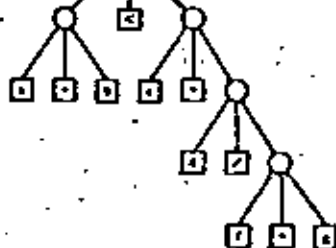
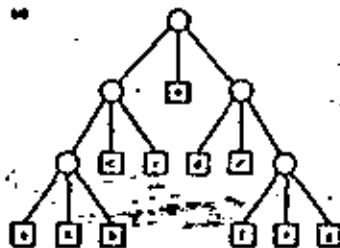
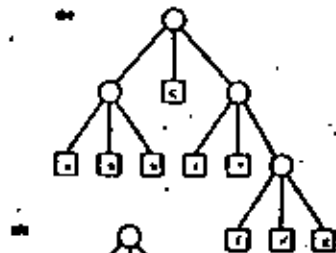
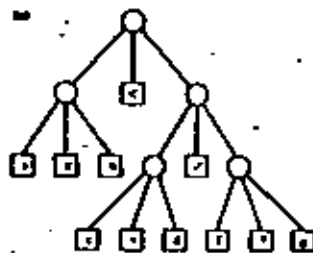
5. Find which of the three trees given below correctly represents the given expression and exhibit the expression represented by each of the other trees.

$$y \left( \frac{4-2y}{3} \right) + y^2$$

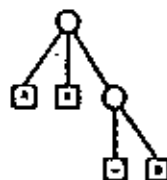
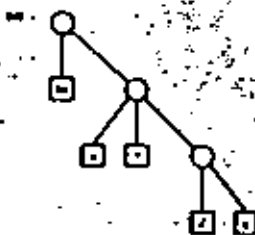
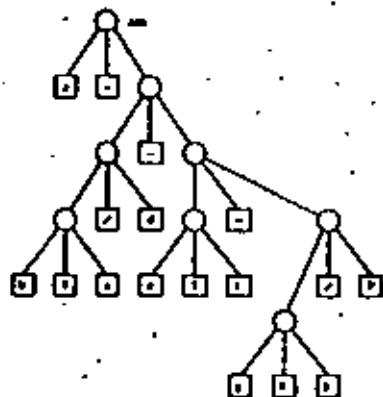
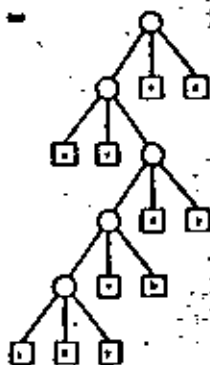
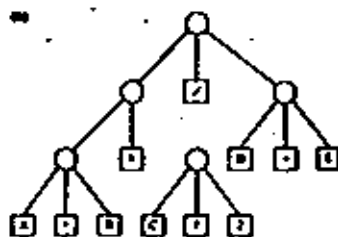
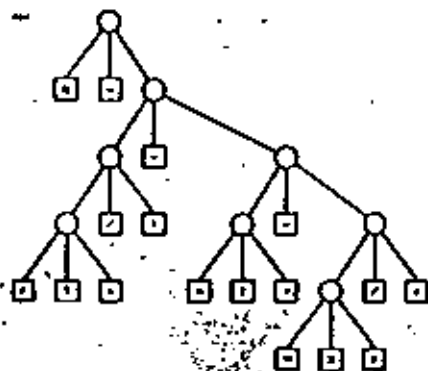


6. Find the tree among the four given below that represents the given expression correctly and exhibit the expression represented by each of the other trees.

$$a \times b < c + d / (f + g)$$



7. Convert the trees below into the corresponding arithmetic expression.



8. How would you draw the expression tree for the expression  $A \times (-B)$ ? The problem here is to decide how one should represent a unary operation. One possibility is shown in the tree on the left.

Here the subexpression  $-B$  is treated as a root node having a unary operator  $[-]$  and one argument  $[B]$  as offspring nodes. One can deduce that the operator is unary by the fact that the (left-handed) operand node is missing. Functions such as  $\sqrt{\quad}$ ,  $\cos$ , and so forth, can be thought of as unary operators. Using the above expression scheme, or another of your own choosing, develop expressions trees for:

- (a)  $A + \sqrt{B}$   
 (b)  $\cos(x^2 + y^2)$   
 (c)  $\sqrt{\frac{1}{2} \times (1 + \frac{q}{\sqrt{p^2 + q^2}})}$

9. Which of the following statements is false?

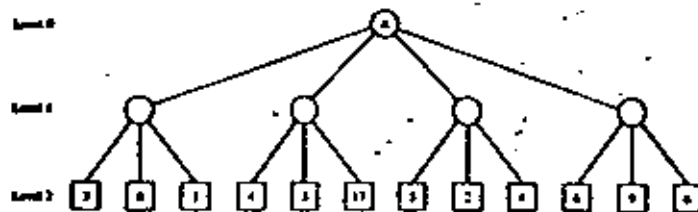
- (a) A terminal node has one ancestor node and no descendant nodes.  
 (b) A root node has no ancestor nodes and may have no descendant.  
 (c) A nonterminal node has no descendant nodes.  
 (d) A nonterminal node may have only one ancestor node.  
 (e) A terminal node can be connected to an ancestor.

*Hint:* If you have any question as to the meaning of "ancestor" and "descendant" just think of a family tree.

10. Any given two-dimensional matrix can be represented as a tree. For example, the matrix

$$A = \begin{bmatrix} 3 & 4 & 5 & 6 \\ 8 & 3 & 2 & 9 \\ 1 & 17 & 4 & 6 \end{bmatrix}$$

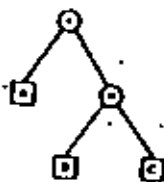
can be expressed as the tree:



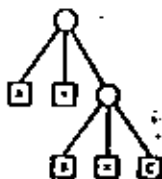
(a) Given the representation above, the four nodes at Level 1 correspond to (choose one):

- (1) The four elements of the main diagonal of the given matrix.  
 (2) The four elements of row 1 of the given matrix.

- (3) The four columns of the given matrix.  
 (4) The sum of the elements in each of the columns of the given matrix.  
 (5) None of the above.
- (6) Show at least one other way to represent the matrix  $A$  as a tree structure.
11. Using the notation, give the paths for the nodes labeled  $\square$ ,  $\square$ , and  $\square$ , in the second of the two trees referred to in Problem 3 of this exercise set.
12. Develop a scheme to denote the saving in time that, in principle, is possible in a computer having multiple processing units that can execute concurrently in the same expression. Show how your scheme would work on the following expressions.
- (a)  $a^2 + b^2 + c^2$   
 (b)  $(a - b) \times (c - d) / (e \times (f + g))$   
 (c)  $(\sqrt{x + \sin y}) / z$
13. In the text we have always shown the operator symbol of an expression tree as a terminal node, so each nonterminal node of the tree has three offspring if the operator has two operands, and two offspring if the operator has one operand. Another way to draw the tree is to place each operator symbol at its parent (nonterminal) node. For example, the tree for the expression
- $$A + B \times C$$
- may be drawn as



instead of as



The new form, which we shall call a binary expression tree, has fewer nodes but the same amount of information. Refer to Figure 6-14 and:

- (a) Produce the binary expression tree equivalent to the expression tree given in the figure.  
 (b) Show how the tree table in that figure can be changed in structure and in content to represent the binary tree you developed in part a.  
 (c) Which tree table would require less storage in a computer representation?

## 6-2 Tree searches

We have now seen enough of trees to have observed their main structural characteristics; segments of a subtree always connect to new nodes that form a continuation of the same subtree; there is no jumping back to nodes closer to the root; and there is no crossing or crisscrossing between subtrees.

There are many ways one can construct and store a tree structure. Depending on what use is to be made of the tree, some representations (we will call these storage structures) are better than others. Trees are searched for one reason or another, either to gain specific information, to reach a conclusion, or to modify the tree in a certain way. A tree search lies at the heart of a number of mathematical problems and a great number of games.

### Natural-order Tree Search

There is a systematic way to scan all the nodes of a tree that is used frequently in solving problems. We call it *natural-order searching*. Although a squirrel may have better ways of finding nuts in a tree, it will help us to understand natural-order search if we imagine a searching squirrel willing to follow these rules.

1. Start at the trunk (root) and don't stop trying segments until you reach a leaf (terminal node) unless you find a nut and choose to stop at that point.
2. Upon reaching a terminal without finding a nut, back up to the node you just passed, that is, to the *parent* node of the terminal.
3. Now, choose the next untried segment, if any, and move forward along it toward another leaf node.

4. If there are no untried segments, crawl backward to the predecessor (parent) node and repeat the process of trying to reach another leaf node.
5. If you ever find yourself back at the root having already tried all segments from the root without finding a nut, you have finished searching the entire tree in natural order and can report a failure to find a nut.

Figure 6-15 shows a natural-order search of a tree. The numbers beside the nodes indicate the sequence in which they

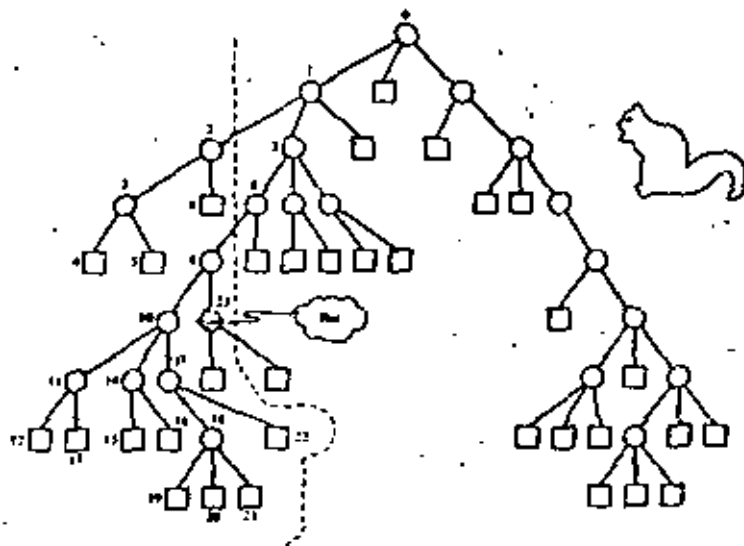


FIGURE 6-15  
Symmetric (natural order)  
search for a nut

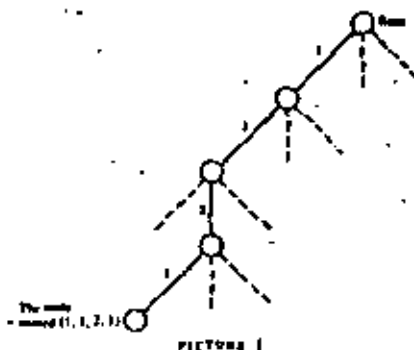
are first encountered (i.e., as the squirrel sees them in its forward progress). We picture one of these nodes as a nut. It is the 23rd node encountered. Notice the systematic, left-to-right selection of segments at each node.

#### Algorithm for Natural-Order Tree Search

Now suppose we wish to construct a tree search algorithm that generally follows the stated set of rules. One of our problems is how to interpret rule 3, that is, how to choose among the

remaining untried segments. If we recall, however, that the segments emanating from each node are, or always can be, ordered, then a simple interpretation comes quickly to mind: choose the segment, if any, whose ordinal number name is one higher than that of the segment previously tried.

To make this choice simpler, the algorithm can always identify the success or parent node from which the "previously tried" segment emanated. That is, the algorithm can only identify additional segments in terms of the common parent. This backup capability is assured if the algorithm at all times has an up-to-date record of where it is in the tree search and can represent this data in the form of a path list. For example, suppose it has been discovered that the segments from the node whose path designation is (1, 1, 2, 1) need be examined no further (Figure 1).

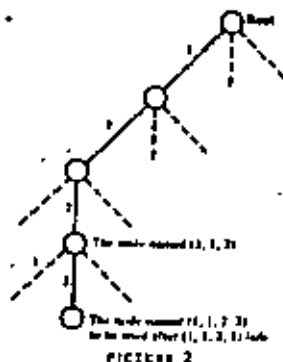


What is the path name for the parent of that node? The answer is (1, 1, 2).

How do we apply rule 3 in this parent (1, 1, 2)? (Rule 3: choose the next untried segment, if any, and move forward along it toward another leaf node.) The answer is, if there is a node whose path is (1, 1, 2, 1 + 1), try it (Figure 2).

In general, suppose we have tried the segment leading from node (1, 1, 2) to node (1, 1, 2, i) and the subtree whose root is (1, 1, 2, i) has failed to contain the nut we are looking for. To select the next untried segment, if any, of node (1, 1, 2), we have only to check whether there exists a valid node





whose path is  $(1, 1, 2, i + 1)$ . If no, select the segment leading to this node and if not, back up, and check whether there is a valid node  $(1, 1, 2 + 1)$ . If no, select the segment leading to this node, but if not, back up again and see whether  $(1, 1 + 1)$  is valid, and so on.

We now sense that by starting out with a path list that represents the root node (an empty list of segments), and by continuing to update that list as we move through the tree to reflect where we are in the search, then simple adjustments to the path allow us to determine each new direction of search.

Figure 6-16 shows a systematic procedure, that is, an algorithm for conducting natural-order tree search. The algorithm is represented in top-down style, with Figure 6-16a giving the topmost view. Any necessary data are input in box 1 and the tree search begins at box 2. In *Tree\_Search*, whose details are given in Figure 6-16b, there are two key variables: *level* and *path*. The value of *level* tells us the number of elements in *path*. Values of these two variables determine the current node of the search. In a sense the current node is the one we are standing on while we try to find the next node to move to. These variables are initialized in box 1 of Figure 6-16a. *Tree\_Search* sets some sort of switch to indicate success or failure. (Recall that a root node by itself does not constitute a tree. There must be at least one subtree. For this reason the first time box 2.2 is executed the Yes outlet will be taken.) Upon exit from *Tree\_Search*, the main program, in effect, tests

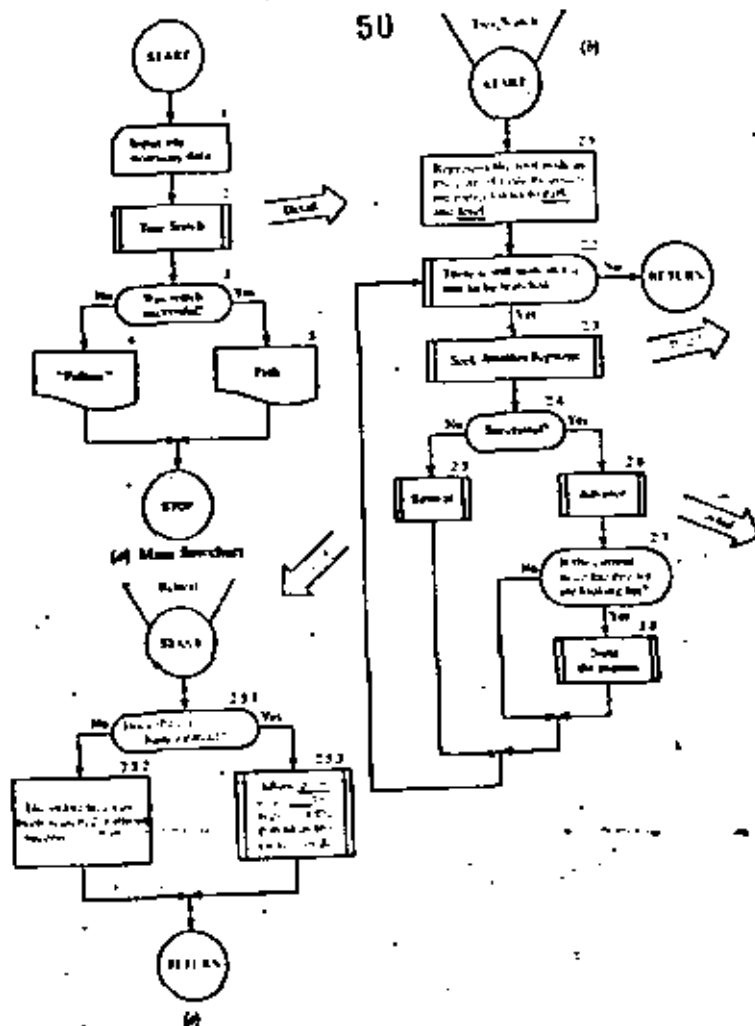
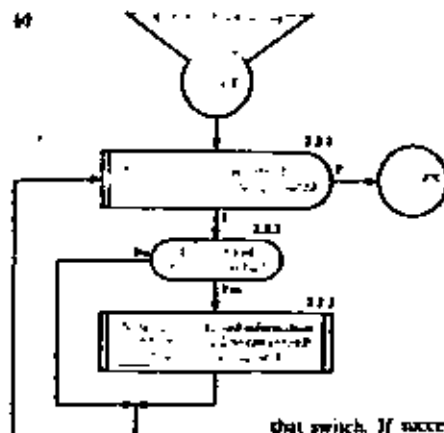


FIGURE 6-16  
Top-down development of an  
algorithm for natural-order  
tree search.

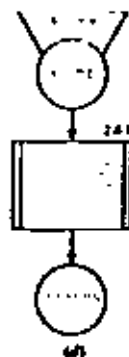


that switch. If success is indicated, the path list is displayed, identifying the location in the tree where the nut was found; otherwise a failure message is displayed.

To simplify the details of *Tree\_Search*, three of its boxes are given in more detail in Figure 6-16c, 6-16d, and 6-16e. Notice that rule 3 is implemented in box 2.3 as a call to a procedure, "Seek\_Another\_Segment," whose details (Figure 6-16c) include a test for admissibility of untried segments. Although not shown in the level of detail given in Figure 6-16c, we imagine that some sort of switch is set by *Seek\_Another\_Segment*, which can be tested upon return to *Tree\_Search* so that the former's success or failure can be determined at box 2.4. If successful, there is a new node to which the search may advance (details in Figure 6-16d). If unsuccessful, it is necessary to retreat to the parent node, if any (Figure 6-16e). (Remember that *Seek\_Another\_Segment* reports failure only after all segments have been tested.)

The bookkeeping of the retreat operation (box 2.5.3) is a two-step process.

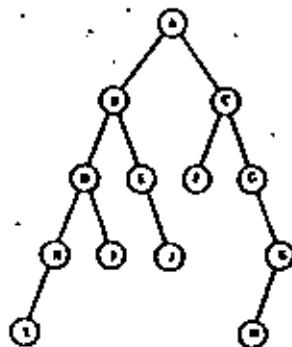
1. Detach the last element of *path*, which is a segment number, and save it to use the next time *Seek\_Another\_Segment* is called at box 2.3.
2. Decrement *level* by one to reflect the shortened length of *path*.



The segment number saved in step 1 is needed in boxes 2.3.1 and 2.3.3. When a new segment number is selected in box 2.3.3, it in turn is saved for use by *Advance*. The next time that procedure is called at box 2.6, *Advance* increments *level* by one and appends the new segment number on to the end of *path*. We leave to the reader the pleasure of reviewing these details and convincing himself of their correctness. As a parting remark, it is worth observing that the nature of the admissibility check hinted at in box 2.3.2 may be crucial to the success of the search. As many inadmissible structures as possible must be ruled out at each stage. For example, the squirrel should recognize each dead limb and not search it. Otherwise, the proportion of useless paths may grow rapidly, meaning that the efficiency of the search method can plummet toward zero. Next we examine several interesting problems that employ this type of search in their algorithmic solution.

## EXERCISE 6-2

1. List the nodes of the tree below in the order in which they would be encountered in a breadth-order search.

6-3  
The four-color  
problem

Maps are colored to make it easy to see at a glance the extent of each country. It is necessary that neighboring countries (i.e. countries with a common boundary line) be assigned different colors. Does the mapmaker then need more than four colors to do his job? He doesn't care, but we do.

This problem was one of the most celebrated challenges in mathematics. It is of great intellectual interest and has intrigued many people from all parts of life. Actually, its solution has little or nothing whatsoever to do with making maps. A mapmaker is and always will be able to print maps using as many different colors as he needs.

A checkerboard is an example of a map that can be colored with only two colors. The four-country map shown in Figure 6-17 requires four colors. Because each pair of countries is adjacent, no two can have the same color.

It didn't take us long to find a map requiring four colors. Yet, in over 100 years of searching, no one has succeeded in finding a map requiring five! It is natural to conjecture that every map can be colored with four colors, and many mathematicians have racked their brains trying to prove this conjecture. The best they have been able to do so far is to show that every map can be colored using no more than five colors.\*

We are about to see how computer methods can be applied to the four-color problem. We will not use the computer to show that the four-color conjecture is true. Indeed, it is entirely possible that no computer can ever prove this. However, true or false, we can use the computer to determine whether a particular map can be colored in only four colors. This is the task for which we want to construct an algorithm.

Before starting on this algorithm, a few remarks concerning the coloring of maps may be helpful.

A *minimal five-color map* is a map requiring five colors, so that every other map requiring five colors has at least as many countries. Of course, no minimal five-color map has ever been found. The mathematicians have shown that if such maps exist at all, then some of them satisfy these two conditions.

1. No point is a boundary point of more than three countries.
2. Each country is a neighbor of at least five others.

Moreover, it can be shown that every minimal map, if any exist, must satisfy the second condition.

It is therefore customary to consider as candidates for counterexamples to the four-color conjecture only maps satisfying these conditions.

\*A simple proof of the five-color theorem exists. It may be found, for example, in *How to Think About It*, Oxford University Press (1981), by Courant and Robbins.

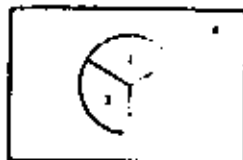


FIGURE 6-17

#### Four-Coloring as a Tree Search

We can model the problem of four-coloring a given map, say, the one pictured in Figure 6-18, as one of traveling along a path through a tree such as that shown in Figure 6-19. Each segment represents a decision to color a country, with colors 1, 2, 3, or 4. The  $i$ th segment in a path from the root corresponds to the coloring of the  $i$ th country of the map.



FIGURE 6-18



FIGURE 6-19



FIGURE 6-20

Observe that many paths through the tree turn out to represent identical colorings of the map except for renaming of the colors, and it is desirable to avoid searching through such duplicate patterns. (E.g., the two heavy-line paths in Figure 6-20 represent the same coloring patterns with different names used for the colors.) One way to avoid the unnecessary search is to fix at the outset in a quite arbitrary way the colors for neighboring countries 1, 2, and 3 and to begin the real search with the coloring of country 4.

In coloring all countries, from the fourth country on, as seen in Figure 6-21, we assume that all four choices are possible. Most of the time, however, as can be seen in Figure 6-22, only one, two, or three of these choices will be admissible. Sometimes even all four choices will be inadmissible, as ex-



FIGURE 6-21  
Showing coloring tree and one path representing the coloring of the first six countries (colored last).

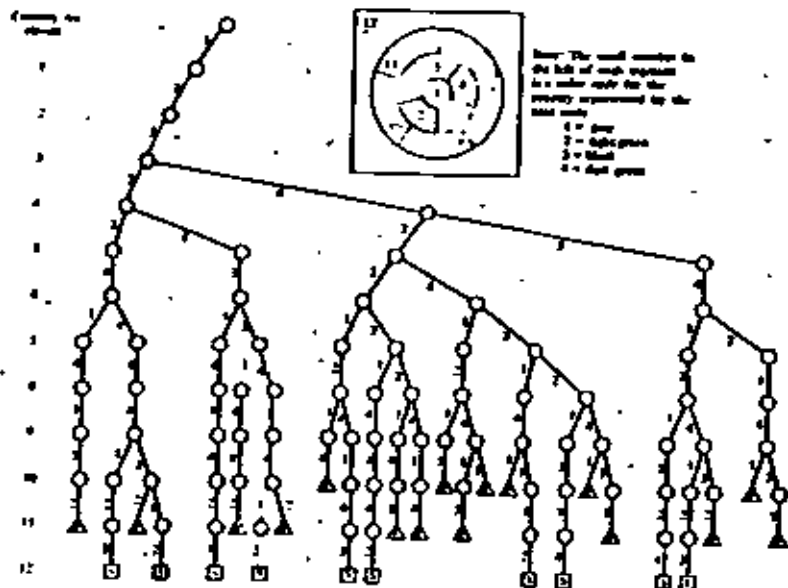


FIGURE 6-22  
An empty coloring tree showing how to traverse the map of Figure 6-21.

10 Successes, marked by terminal nodes of the form  $\square$

11 Failures, marked by terminal nodes of the form  $\triangle$

12 Successes, marked by terminal nodes marked F in Figure 6-22. Only 10 paths lead to S (success) terminals.

### EXERCISES 6-1

1. Compute the theoretical maximum number of possible terminal nodes for the coloring tree of the 13-country map in Figure 6-22. *Hint:* Use Figure 6-23 as a guide.

2. Assume that it takes only 1 micro-second to check another path to a terminal, and that the search of half of these paths is required before the desired terminal is reached. How long would the computer stay busy before it found what it was looking for on a 39-country map? Assume all operations to be admissible. Express your answer in units of years.
3. By renumbering the countries on the map of Figure 6-21, show that a coloring tree can have nodes with three and even four permissible requests emanating from them.
4. Using a form similar to that of Figure 6-22, draw a "coloring tree" for the map shown below.



### A Four-Coloring Algorithm

Let us see how to apply what we have just learned about tree search to an actual problem. It is one thing to discuss a tree in the abstract and another to start with a problem, define in some detail the tree search that is involved, and then develop a detailed flowchart algorithm. In this case, we will take as our problem statement: Develop a detailed flowchart algorithm for four-coloring any  $n$ -country map.

The first step toward this objective might be to devise a method to represent any  $n$ -country map. To do this we need a sample map for study as, for example, in Figure 6-23. The

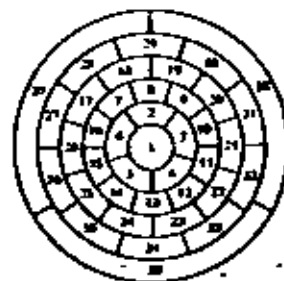


FIGURE 6-23  
Example of map to be four-colored by a computer algorithm.

map consists of 39 countries, and the countries have been numbered or indexed in the order that the algorithm will attempt to "color" them.

The efficiency of the algorithm will be greatly improved if each country borders on as many lower-numbered countries as possible. We do not absolutely insist on this but, if you have done Problem 3 of Exercise 6-3, you will appreciate why we recommend this approach. We do, however, require that the first three countries all be neighbors of each other.

How do we represent the map in computer storage? One way is to construct a "connection table," listing after each country all of its neighbors in increasing order. This is shown for our example in Table 6-1.

Our algorithm should consult this table when deciding how to color a particular country. For example, if we were coloring country 15, we could see in row 15 that countries 5, 6, and 14 are neighbors already colored. Our choice of color for 15, then, depends solely on the currently chosen colors for 5, 6, and 14.

Knowing that country 15 also has neighbors numbered 16, 25, and 26 appears to be superfluous. This leads us to the idea of a shaved-down table, which we will call the "reduced connec-

TABLE 6-1  
The Connection Table for the Map in Figure 6-21

Country	Neighbors	Country	Neighbors
1.	2 3 4 5 6	21.	10 11 20 22 31 32
2.	1 3 4 7 8 9	22.	11 12 21 25 32 33
3.	1 2 4 8 10 11	23.	12 13 22 24 33 34
4.	1 2 3 11 12 13	24.	13 14 23 29 34 35
5.	1 4 6 12 14 15	25.	14 15 24 26 35 36
6.	1 2 5 7 15 16	26.	15 16 17 25 27 36
7.	2 6 8 16 17 18	27.	17 26 28 36 37
8.	2 7 9 18 19	28.	17 18 27 29 37
9.	2 3 8 19 20	29.	18 19 28 30 37 38
10.	3 9 11 20 21	30.	19 28 29 31 38
11.	3 4 10 12 21 22	31.	20 21 30 32 38
12.	4 11 13 22 23	32.	21 22 31 33 38 39
13.	4 5 12 14 23 24	33.	22 23 32 34 39
14.	5 13 15 21 25	34.	23 24 33 35 39
15.	5 6 14 16 25 26	35.	24 25 34 36 39
16.	6 7 15 17 26	36.	25 26 27 35 37 39
17.	7 16 18 26 27 29	37.	27 28 29 36 38 39
18.	7 8 17 19 28 29	38.	29 30 31 32 37 39
19.	8 9 18 20 29 30	39.	32 33 34 35 36 37 39
20.	9 10 19 20 30 31		

TABLE 6-2  
Reduced Connection Table for the Map in Figure 6-21

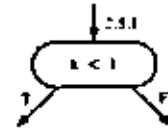
Country	Neighbors CONN <sub>i</sub>	Width w <sub>i</sub>	Country	Neighbors CONN <sub>i</sub>	Width w <sub>i</sub>
1		0	21	10 11 20	3
2	1	1	22	11 12 21	3
3	1 2	2	23	12 13 22	3
4	1 3	2	24	13 14 23	3
5	1 4	2	25	14 15 24	3
6	1 7 5	3	26	15 16 17 25	4
7	2 6	2	27	17 26	2
8	2 7	2	28	17 18 27	3
9	2 3 8	3	29	18 19 28	3
10	3 6	2	30	19 20 29	3
11	3 4 10	3	31	20 21 30	3
12	4 11	2	32	21 22 31	3
13	4 5 12	3	33	22 23 32	3
14	5 13	2	34	23 24 33	3
15	5 6 14 3	3	35	24 25 34	3
16	6 7 15 3	3	36	25 26 27 35	4
17	7 16 2	2	37	27 28 29 36	4
18	7 8 17 3	3	38	29 30 31 32 37	5
19	8 9 18 3	3	39	32 33 34 35 36 37 38	7
20	9 10 19 3	3			

tion table." It is constructed by striking out of each row in the table all  $n$ 's greater than the number of the row itself. The reduced connection table for our example is seen in Table 6-2 and can be thought of in this case as a 39-row by 7-column array called CONN. The number of nonnull elements in each row is given by elements of an associated list  $w$ . Thus the algorithm can search the first  $w_i$  elements in the  $i$ th row of CONN to determine which neighbors have already been colored.

If we are to apply our generalized tree search algorithm (Figure 6-16) to the map-coloring problem, we must also decide how to represent the *current node* (i.e., how to represent the variables *path* and *level*). The variable *path* is a list of elements, each of which designates a segment choice. Our decision to use color codes 1, 2, 3, and 4 for the four possible color choices leads us directly to the decision that a search from a node may be accomplished by selecting (trying) the segments in the same order, 1, 2, 3, and 4. The decision to make this correspondence between the color codes and the segment order imposes the required ordering on the segments from each node of our coloring tree. Moreover, the  $i$ th element



any new patterns. If we want to search the complete tree, however, including the four possible colors for the first country, we have only to change the test in box 2.5.1 to read

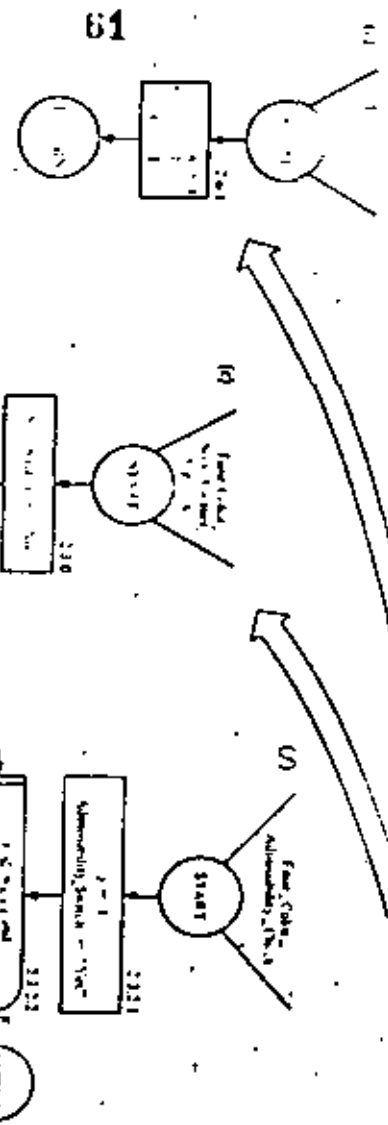


Whenever the level counter  $k$  is found to equal  $n$  (boxes 2.7 and 2.8 of *Tree\_Search*), *Success\_Switch* is set to "Yes." The variable *Seek\_Switch* tested by *Tree\_Search* in box 2.4 is set to either "Yes" or "No" by *Seek\_Another\_Segment* (Figure 6-24). This procedure in turn reports success if and only if the subprocedure *Admissibility\_Check* reports success. This latter procedure (Figure 6-21f) determines whether any of the previously colored neighbors (there are  $w_{k+1}$  of them) have the same color as the tentative color, i.e. that is being considered for the  $k + 1$ st country. If so, *Admissibility\_Switch* is set to "No" so that, after the RETURN to *Seek\_Another\_Segment*, another (the next) color may be tried. Notice that only in *Admissibility\_Check* is there any reference to the map's representation. This suggests that detailed flowcharts for different natural-order tree search problems will differ mainly in the details of this particular part of the search algorithm.

The bookkeeping of *Retreat* and *Advance* in Figure 6-24 uses auxiliary variable,  $tc$ , tentative color. This variable is also used in *Seek\_Another\_Segment* during the search for an admissible segment and, in box 2.3.5,  $tc$  is incremented whenever an inadmissible segment is found. In *Advance*,  $k$  is incremented to represent the longer successful coloring path. The successful tentative color is stowed away in  $Color_k$ , and the auxiliary variable  $tc$  is reset to 1. (See box 2.6.1.) During *Retreat* the current color choice for country  $k$  must be remembered so that the search for another segment of country  $k$ 's parent can resume at a value of  $tc$  that is one greater than the last one tried. The saving of this information is accomplished by the assignment step,

$$tc \leftarrow Color_k + 1$$

as seen in box 2.5.3. Then the path length  $k$  is shortened by 1.



Variable	Description
$n$	Number of countries
$Color_k$	A list of numbers, $1 \leq k \leq n$ , where the value in the $k$ th position is the color of the $k$ th country.
$tc$	Tentative color for the next country.
$w_{k+1}$	Number of neighbors of country $k+1$ .
$Success_Switch$	Boolean variable for success.
$Seek_Switch$	Boolean variable for success.

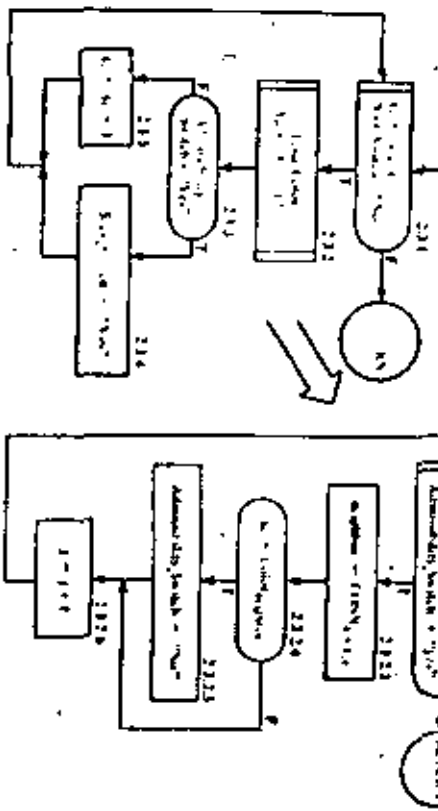


FIGURE 6-24  
Four-coloring algorithm.  
Top-down approach.



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

**INTRODUCCION A LA PROGRAMACION Y  
COMPUTACION ELECTRONICA**

**M A T R I C E S  
APLICACIONES NO MATEMATICAS**

**OCTUBRE, 1982**



## INVERSO DE UNA MATRIZ.

NUM. REALES:  $a \neq 0$   $a \cdot b = b \cdot a = 1$   $b = a^{-1} = \frac{1}{a}$   
 $B \cdot b = b \cdot B = 1 \Rightarrow b = \frac{1}{B} = .123$

MATRICES:  $A \cdot B = I$ ,  $B \cdot A = I$ .  
 $A \cdot B = B \cdot A = I \Rightarrow A$  cuadrada.

$B = A^{-1}$  INVERSA MULTIPLICATIVA

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$   $B = \begin{bmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{bmatrix}$

$A \cdot B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_2 = B \cdot A$

$A = B^{-1}$   $B = A^{-1}$

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$   $B = ? = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

$AB = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a+2c & b+2d \\ a+4c & b+4d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$\begin{cases} a+2c=1 \\ 3a+4c=0 \\ b+2d=0 \\ 3b+4d=1 \end{cases} \rightarrow \begin{cases} a=1-2c \\ 3(1-2c)+4c=0, -2c=-3 \\ c = -\frac{3}{2} \\ a = 1-2 \cdot \frac{3}{2} \end{cases}$

$A = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}; A^{-1} = ?$

$AA^{-1} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} c & d \\ e & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$c=1, c=0$   $d=0, d=1$

$A^{-1}$  no existe

MATRICES CUADRADAS SIN INVERSO: SINGULARES.  
 MATRICES CUADRADAS CON INVERSO: NO SINGULARES  
 O REGULARES

PROPIEDADES:

1) COMUTATIVIDAD:  $A \cdot A^{-1} = A^{-1} \cdot A = I$

2) UNICIDAD

$\Rightarrow (A^{-1})^{-1} = A$

3)  $(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$  (ANALOGA  $(AB)^T = B^T \cdot A^T$ )

DEM.  $(B^{-1} \cdot A^{-1})(AB) = B^{-1} \cdot \underbrace{A^{-1} \cdot A}_I \cdot \underbrace{B \cdot B^{-1}}_I = I$

$(AB)^{-1} = B^{-1} \cdot A^{-1}$

5)  $(A^T)^{-1} = (A^{-1})^T$

UTILIDAD DEL INVERSO:  $A \cdot x = b$

$x = A^{-1} \cdot b$   
 $x = A^{-1} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$   
 $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

$$\begin{cases} x_1 + 2x_2 = 4 \\ 3x_1 + 4x_2 = -2 \end{cases} \quad \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 4 \\ -2 \end{pmatrix}$$

$$\boxed{A \underline{x} = \underline{b}}$$

$$\underline{x} = A^{-1} \underline{b}$$

$$A^{-1} = \begin{pmatrix} -2 & 1 \\ 3/2 & -1/2 \end{pmatrix}, \quad \underline{x} = \begin{pmatrix} -2 & 1 \\ 3/2 & -1/2 \end{pmatrix} \begin{pmatrix} 4 \\ -2 \end{pmatrix} = \begin{pmatrix} -10 \\ 7 \end{pmatrix}$$

$$\underline{Ax} = \underline{b}$$

varias diferentes, invertir A

A VECES, LAS COMPONENTES DE  $A^{-1}$  TIENEN INTERPRETACION ESPECIAL.

AJUSTE POR MINIMOS CUADRADOS

$A^{-1}$ : COMPONENTES  $\rightarrow$  CLASE Y MAGNITUD DE ERRORES, EN LOS DATOS.

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = |A|$$

$$a(ei - fh) - (b(di - fg) - c(dh - eg))$$

## INVERSION DE MATRICES

METODOS:  $\left\{ \begin{array}{l} \text{ALGEBRAICOS (EXACTOS)} \\ \text{NUMERICOS (APROXIMACIONES SUCCESIVAS)} \end{array} \right.$

ALGEBRAICOS

ADJUNTA

$$A^{-1} = \frac{adj(A)}{|A|}$$

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

$$\begin{pmatrix} + \begin{vmatrix} e & f \\ h & i \end{vmatrix} & - \begin{vmatrix} d & f \\ g & i \end{vmatrix} & + \begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ - \begin{vmatrix} b & c \\ h & i \end{vmatrix} & + \begin{vmatrix} a & c \\ g & i \end{vmatrix} & - \begin{vmatrix} a & b \\ g & h \end{vmatrix} \\ + \begin{vmatrix} b & c \\ e & f \end{vmatrix} & - \begin{vmatrix} a & d \\ d & f \end{vmatrix} & + \begin{vmatrix} a & b \\ d & e \end{vmatrix} \end{pmatrix}^T$$

$$|A| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$= a(ei - fh) - b(di - fg) + c(dh - eg)$$

INVERSION POR PARTICIPACION

$$A = \left( \begin{array}{c|c} \Delta_1 & \Delta_2 \\ \hline \Delta_3 & \Delta_4 \end{array} \right) \quad A^{-1} = B = \left( \begin{array}{c|c} B_1 & B_2 \\ \hline B_3 & B_4 \end{array} \right)$$

$$\Delta A^{-1} - \Delta B = \left( \begin{array}{c|c} \Delta_1 & \Delta_2 \\ \hline \Delta_3 & \Delta_4 \end{array} \right) \left( \begin{array}{c|c} B_1 & B_2 \\ \hline B_3 & B_4 \end{array} \right) = I = \left( \begin{array}{c|c} I & O \\ \hline O & I \end{array} \right)$$

$$\left( \begin{array}{cc|cc} A_1 B_1 + A_2 B_3 & A_1 B_2 + A_2 B_4 & I & O \\ A_3 B_1 + A_4 B_3 & A_3 B_2 + A_4 B_4 & O & I \end{array} \right)$$

$$A^{-1} = B = \left( \begin{array}{c|c} A^{-1} (I - A_3 B_3) & -\Delta_1^{-1} A_2 B_4 \\ \hline -B_4 \Delta_3 A^{-1} & (A_4 - \Delta_3 A_1^{-1} A_2)^{-1} \end{array} \right)$$

INVERTIR,  $A^{-1}$  Y  $(A_4 - \Delta_3 A_1^{-1} A_2)^{-1}$

DE ORDENES MENORES QUE  $A$

5) SOL. NUMERICA SISTS. ECS

- SOL. NUM. ECS. DIFS. OR. DINARIAS Y PARCIALES. (5a)
- ANALISIS ESTRUCTURAL
- ANALISIS DE REDES
- OPTIMIZACION.
- ANALISIS DE DATOS.

$$A \bar{x} = \bar{b}$$

POE REGLA DE CRAMER:

NUMERO. OPERACIONES  $\sim N^4$   
IMPRACTICO

POE INVERSION DE MATRICES:

CUANDO HAY MUCHAS  $\bar{b}$  DIFERENTES

MATRIZ SIST. ECS.  $\left\{ \begin{array}{l} \text{LLENA} \\ \text{MUCHOS CEROS} \end{array} \right.$

METODOS DE SOL.  $\left\{ \begin{array}{l} \text{DIRECTOS} \\ \text{ITERATIVOS.} \end{array} \right.$

SOLUCION SISTEMA DE ECUACIONES (6)

$$A = \begin{pmatrix} 6 & 2 \\ 2 & 1 \end{pmatrix}$$

$$AB = I$$

$$\begin{pmatrix} 6 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{cases} 6a + 2c = 1 \\ 2a + c = 0 \\ 6b + 2d = 0 \\ 2b + d = 1 \end{cases} \quad \begin{matrix} a = 0.5 & c = -1 \\ b = -1 & d = 3 \end{matrix}$$

$$\begin{pmatrix} 6 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} a \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 6 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} b \\ d \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

A DE 3x3  $A^{-1} = B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$

HAY QUE RESOLVER

$$A \begin{pmatrix} b_{11} \\ b_{21} \\ b_{31} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad A \begin{pmatrix} b_{12} \\ b_{22} \\ b_{32} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad A \begin{pmatrix} b_{13} \\ b_{23} \\ b_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

DIAGONALIZACION (GAUSS-JORDAN) (70)

$$\begin{array}{c} [A \mid I] \\ \downarrow \\ [I \mid A^{-1}] \end{array}$$

$$\left[ \begin{array}{cccc|cccc} a_{11} & a_{12} & \dots & a_{1n} & 1 & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & a_{2n} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & 0 & 0 & \dots & 1 \end{array} \right]$$

$$\left[ \begin{array}{cccc|cccc} 1 & 0 & \dots & 0 & b_{11} & b_{12} & \dots & b_{1n} \\ 0 & 1 & \dots & 0 & b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & b_{n1} & b_{n2} & \dots & b_{nn} \end{array} \right]$$

POR OPERACIONES ELEMENTALES DE RENGLON:

- 1) INTERCAMBIO DE RENGLONES.
- 2) MULTIPLICAR UN RENGLON POR UN ESCALAR.
- 3) MULTIPLICAR UN RENGLON POR UN ESCALAR Y SUMARLO A OTRO RENGLON.

ESCOGER PIVOTE :



PIVOTE PARCIAL.



PIVOTE TOTAL.

# OPERACIONES GAUSS - JORDAN  $O(n^3)$   
 CRAMER  $O(n^4)$

MATRICES DE BANDA



$O(n)$

ALMACENAMIENTO MAT. LUEVA  $n^2 + 2n$   
 MAT. BANDA  $3n$

GAUSS - SEIDEL

$|x_i^{(k)} - x_i^{(k-1)}| \leq \epsilon$  CRITERIO ABSOLUTO CONVER.

$|\frac{x_i^{(k)} - x_i^{(k-1)}}{x_i^{(k)}}| \leq \epsilon$  CRITERIO RELATIVO CONVER.

CONV. GARANTIZADA SOLO SI:  $|c_{ii}| > \sum_{j \neq i} |c_{ij}|$   
 (DIAGONALMENTE DOMINANTE)

POTENCIAS DE UNA MATRIZ

$A^n = A \cdot A \dots A$

(n FACTORES)

$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, A^2 = A \cdot A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$

$A^0 = I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (a^0 = 1)$

$A^{-1} = \begin{pmatrix} -2 & 1 \\ 3/2 & -1/2 \end{pmatrix}$

$A^{-2} = (A^{-1})^2 = \begin{pmatrix} -2 & 1 \\ 3/2 & -1/2 \end{pmatrix} \begin{pmatrix} -2 & 1 \\ 3/2 & -1/2 \end{pmatrix}$

PROPIEDADES DE MATRICES

1) NUMEROS REALES:  $ab = 0 \Rightarrow \begin{cases} a = 0 \\ b = 0 \end{cases}$

MATRICES:  $AB = 0 \Rightarrow \begin{cases} A = 0 \\ B = 0 \end{cases}$

$\begin{pmatrix} 1 & -1 \\ 7 & -7 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$

$\neq 0$   $\neq 0$

$$2) \quad ab = ac \rightarrow b = c$$

$$AB = AC \not\rightarrow B = C$$

$$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 9 & 10 \\ 4 & 20 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} -2 & -2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 9 & 10 \\ 4 & 20 \end{pmatrix}$$

$$3) \quad ab = ba$$

$$AB \neq BA \quad \text{NO GENERAL}$$

$$4) \quad ab = I \rightarrow b = \frac{1}{a} = a^{-1}$$

$$AB = I \quad B = A^{-1} \quad \text{NO SIEMPRE EXISTE}$$

5) NO EXISTE TRANSPUESTO DE NUMERO REAL

$$A = A' \quad (A')' = A$$

8)

## APLICACIONES DE MATRICES

1) TEORIA DE REDES.

GRAFICAS QUIMICAS O ESTRUCTURALES.



RELACE : 1

NO RELACE : 0

MATRIZ DE INCIDENCIA

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

GRAFICA CERTICA

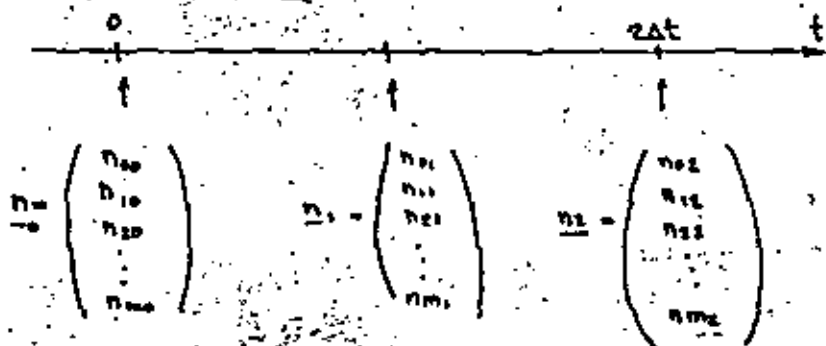


$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

## 2) DINAMICA DE POBLACION.

$\Delta t$  = INTERVALO DE TIEMPO

GRUPO DE EDADES	EDAD	NUMERO PROMEDIO DE HIJAS	PROBABILIDAD DE QUE UNA HEMBRA VIVA PARA ESTAR PRESENTE EN EL SIG. GRUPO DE EDADES $x+1$
$x=0$	$0-\Delta t$	$F_0$	$P_0$
$x=1$	$\Delta t-2\Delta t$	$F_1$	$P_1$
$x=2$	$2\Delta t-3\Delta t$	$F_2$	$P_2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$x=m$	$m\Delta t-(m+1)\Delta t$	$F_m$	$P_m = 0$



$n_{x,t}$  = NUMERO DE HEMBRAS EN EL GRUPO DE EDADES  $x$  AL TIEMPO  $t \Delta t$ .

NUM. HIJAS NACIDAS EN  $0 - \Delta t$

$$F_0 n_{0,0} + F_1 n_{1,0} + \dots + F_m n_{m,0} = n_{0,1}$$

$$P_x n_{x,t} = n_{x+1, t+\Delta t} \quad x = 0, 1, \dots, m-1$$

TRANSICION DE LA POBLACION DE  $t=0$  A  $t=\Delta t$ .

$$\begin{pmatrix} F_0 & F_1 & \dots & F_{m-1} & F_m \\ P_0 & 0 & \dots & 0 & 0 \\ 0 & P_0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & P_{m-1} & 0 \end{pmatrix} \begin{pmatrix} n_{0,0} \\ n_{1,0} \\ \vdots \\ n_{m,0} \end{pmatrix} = \begin{pmatrix} n_{0,1} \\ n_{1,1} \\ \vdots \\ n_{m,1} \end{pmatrix}$$

0 = 0

$$M n_0 = n_1$$

$$M n_1 = n_2 \quad n_2 = M n_1 = M(M n_0) = M^2 n_0$$

$$M n_2 = n_3 \quad n_3 = M n_2 = M(M n_1) = M^3 n_0$$

$$M n_{t-\Delta t} = n_t$$

$$n_t = M^t n_0$$

$M$  = MATRIZ DE PROYECCION.

*Faint handwritten notes at the bottom right of the page.*

(13)

$$\underline{n}_k = M \underline{n}_{k-1} \quad \underline{n}_k = M^k \underline{n}_0$$

1) DADA UNA POBLACION INICIAL Y SU DISTRIBUCION ( $\underline{n}_0$ ), Y DADA M:

SE PUEDE CALCULAR POBLACION FUTURA Y SU DISTRIBUCION.

2) DADA UNA POBLACION INICIAL Y SU DISTRIBUCION, ¿QUE PROPIEDAD DEBE TENER M PARA TENER POBLACION ESTABLE ( $\underline{n}_0 = \underline{n}_1 = \underline{n}_2 = \dots$ )?

$$\underline{n}_k = M \underline{n}_k$$

$$0 = (M - I) \underline{n}_k$$

$$\Rightarrow |M - I| = 0$$

$$F_0 + F_1 P_0 + F_2 P_0 P_1 + \dots + F_m P_0 P_1 \dots P_{m-1} = 1$$

3) DADA UNA POBLACION INICIAL Y SU DISTRIBUCION DE EDADES ESTABLE, ¿CUAL ES LA POBLACION?

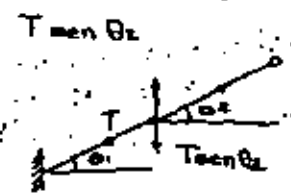
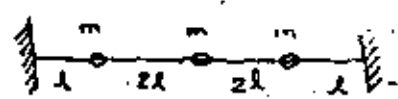
$$\underline{n}_{k+1} = \lambda \underline{n}_k$$

$$M \underline{n}_k = \lambda \underline{n}_k$$

$$\underline{T} = \dots$$

(14)

VALORES Y VECTORES CARACTERISTICOS.



$$T \cos \theta_1 \approx T \frac{x_1}{l}$$

$$T \sin \theta_2 \approx T \frac{x_2 - x_1}{2l}$$

$$F = m \omega^2 = m \frac{d^2 x}{dt^2}$$

$$m \frac{d^2 x_1}{dt^2} = T \frac{(x_2 - x_1)}{2l} - T \frac{x_1}{l}$$

$$m \frac{d^2 x_2}{dt^2} = -T \frac{(x_2 - x_1)}{2l} + T \frac{(x_3 - x_2)}{2l}$$

$$m \frac{d^2 x_3}{dt^2} = -T \frac{(x_3 - x_2)}{2l} - T \frac{x_3}{l}$$



$$X_i = X_i e^{i\omega t}$$

$$\lambda = \omega^2 ml/T$$

$$\begin{cases} (3-\lambda)x_1 - x_2 = 0 \\ -x_1 + (2-\lambda)x_2 - x_3 = 0 \\ -x_2 + (3-\lambda)x_3 = 0 \end{cases}$$

$$\begin{pmatrix} 3-\lambda & 0 & 0 \\ -1 & 2-\lambda & -1 \\ 0 & -1 & 3-\lambda \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \lambda \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$A\underline{x} = \lambda \underline{x}$$

$$(A - \lambda I) \underline{x} = 0$$

$$\exists \Rightarrow |A - \lambda I| = 0$$

$$\begin{vmatrix} 3-\lambda & -1 & 0 \\ -1 & 2-\lambda & -1 \\ 0 & -1 & 3-\lambda \end{vmatrix} = 0$$

$$(1-\lambda)(3-\lambda)(4-\lambda) = 0$$

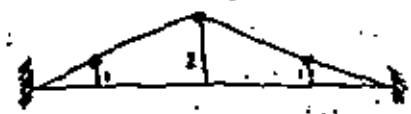
1)  $\lambda_1 = 1$

$$\begin{cases} 2x_1 - x_2 = 0 \\ -x_1 - x_2 - x_3 = 0 \\ -x_2 + 2x_3 = 0 \end{cases}$$

$$x_1 = \frac{x_2}{2} = x_3$$

$$\underline{x} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

MODO DE VIBRACION.

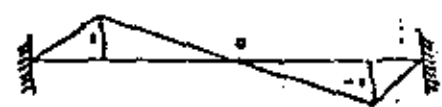


$$\omega^2 = \frac{\lambda T}{2ml} = \frac{T}{2ml}$$

FRECUENCIA ANGULAR DE VIBRACION.

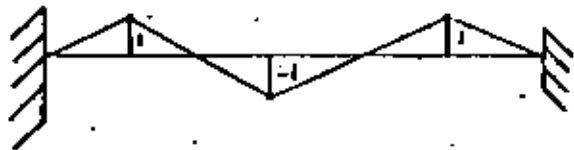
2)  $\lambda_2 = 3$

$$\underline{x} = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$$



$$\omega^2 = \frac{3T}{2ml}$$

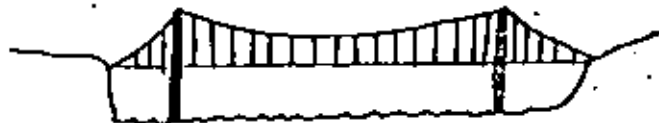
$\Rightarrow \lambda = 4, \quad \underline{x} = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad w = \frac{4T}{2m}$



RESONANCIA



FUENTE DE TACOMA



7 NOV 1940, 4 MESES DE INAUGURADO  
 VIENTO 67 km/h 12 m x 853 m.  
 OSCILACIONES TORSIONALES  
 IMPULSO INVESTIGACION AERODINAMICA.

(17)

FLUIDO CON OBSTACULO: VORTICES (18)  
DE VON KARMAN



TACOMA



VORTICES DE VON KARMAN:  
ALTERNADOS EN TIEMPO Y ESPACIO

- RUGOSIDAD MEDIC
- FORMA
- VELOCIDAD FLUIDO

PROBLEMA: SI LA FRECUENCIA DE LOS VORTICES DE VON KARMAN COINCIDE CON FREC. NATURAL DE OSCILACION TORSIONAL DEL PUENTE.

THEODORE VON KÁRMÁN (1881-1963) (19)

PIONERO, USO MATEMÁTICAS EN CIENCIAS BÁSICAS (AERONÁUTICA, ASTRONÁUTICA)

1911: ANALISIS DE LOS VORTICES.

$$A\underline{x} = \lambda\underline{x}$$

SOLUCIÓN  $\left\{ \begin{array}{l} \underline{x} = 0 \quad \text{TRIVIAL} \\ \underline{x} \neq 0 \iff |A - \lambda I| = 0 \end{array} \right.$

$$A\underline{x} = \lambda\underline{x}$$

$$A\underline{x} = \lambda I \underline{x}$$

$$(A - \lambda I)\underline{x} = \underline{0}$$

$$\Rightarrow |A - \lambda I| = 0 \quad \text{ECUACION CARACTERISTICA}$$

$$P(\lambda) = |A - \lambda I| = \text{POLINOMIO CARACTERISTICO}$$

POLINOMIO EN  $\lambda$  DE GRADO  $n$

RAICES:  $\lambda_1, \lambda_2, \dots, \lambda_n$  EIGEN VALORES

$$|A - \lambda I| = 0$$

VALORES PROPIOS

VALORES CARACTERISTICOS

$$\lambda_i \underline{x}_i = A \underline{x}_i$$

$\underline{x}_i$ : EIGENVECTOR

VECTOR CARACTERISTICO

VECTOR PROPIO

CORRESPONDIENTE A  $\lambda_i$

$\underline{x}_i$  SE OBTIENE SUSTITUYENDO EN

$$(A - \lambda I)\underline{x} = \underline{0}$$

CON  $\lambda = \lambda_i$

EJEMPLO

$$A = \begin{pmatrix} 3 & -1 \\ 4 & -2 \end{pmatrix}$$

$$|A - \lambda I| = 0$$

$$\begin{vmatrix} 3 - \lambda & -1 \\ 4 & -2 - \lambda \end{vmatrix} = 0$$

$$(3 - \lambda)(-2 - \lambda) + 4 = 0$$

$$\lambda^2 - \lambda - 2 = 0$$

$$\lambda = \frac{+1 \pm \sqrt{1 - 4(-2)}}{2} = \frac{+1 \pm 3}{2}$$

$$\lambda_1 = -1$$

$$(A - \lambda I) \underline{x} = 0 \quad \begin{cases} (3 - \lambda)x_1 - x_2 = 0 \\ 4x_1 - (2 + \lambda)x_2 = 0 \end{cases} \quad (21)$$

$$1) \lambda_1 = -1$$

$$\begin{cases} 4x_1 - x_2 = 0 \\ 4x_1 - x_2 = 0 \end{cases}$$

$$\therefore x_2 = 4x_1$$

$$\underline{x}_1 = \begin{pmatrix} x_1 \\ 4x_1 \end{pmatrix} = x_1 \begin{pmatrix} 1 \\ 4 \end{pmatrix} \quad \underline{x}_1 = \begin{pmatrix} 1 \\ 4 \end{pmatrix}$$

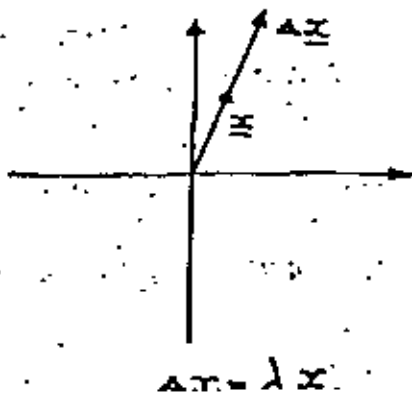
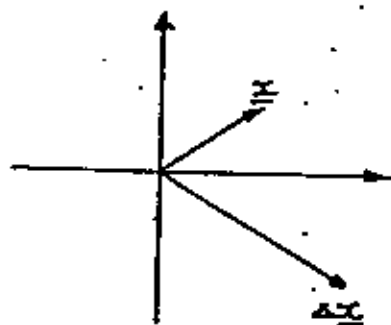
$$2) \lambda_2 = 2$$

$$\begin{cases} x_1 - x_2 = 0 \\ 4x_1 - 4x_2 = 0 \end{cases}$$

$$\therefore x_2 = x_1$$

$$\underline{x}_2 = \begin{pmatrix} x_1 \\ x_1 \end{pmatrix} = x_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \underline{x}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

### INTERPRETACION GEOMETRICA



### OBTENCION DE LOS VALORES CARACTERISTICOS

#### METODO DE KRYLOV

$$\left. \begin{aligned} |A - \lambda I| &= 0 \\ P(\lambda) &= 0 \end{aligned} \right\} \text{ECUACION CARACTERISTICA}$$

1º OBTENER NUMERICAMENTE LOS COEFICIENTES  $C_i$  DEL POLINOMIO CARACTERISTICO.

$$P(\lambda) = \lambda^n + C_{n-1}\lambda^{n-1} + \dots + C_1\lambda + C_0$$

2º OBTENER NUMERICAMENTE LOS VALORES CARACTERISTICOS  $\lambda_i$ , QUE SON LAS RAICES DE LA ECUACION CARACTERISTICA

$$P(\lambda) = 0$$

$$\lambda^n + C_{n-1}\lambda^{n-1} + \dots + C_1\lambda + C_0 = 0$$

3º OBTENER COEFICIENTES  $C_i$ .

#### TEOREMA DE CAYLEY-HAMILTON:

$$\text{SI } p(\lambda) = 0 \Rightarrow p(A) = 0$$

$$P(A) = 0$$

$$A^n + C_{n-1}A^{n-1} + \dots + C_1A + C_0 = 0$$

POSTMULTIPLICANDO POR UN VECTOR ARBITRARIO  $\underline{y}$  CONOCIDO,  $\underline{y} \neq 0$ :

$$A^n \underline{y} + C_{n-1}A^{n-1} \underline{y} + \dots + C_1A \underline{y} + C_0 \underline{y} = \underline{0}$$

REPRESENTA UN SISTEMA DE ECUACIONES CON INCOGNITAS  $C_{n-1}, \dots, C_1, C_0$

SE RESUELVE EL SISTEMA DE ECS. Y SE DETERMINAN LAS  $C_i$

EJEMPLO:

$$A = \begin{pmatrix} 3 & -1 \\ 4 & -2 \end{pmatrix}$$

$$\text{EC CARACT: } \lambda^2 + C_1 \lambda + C_0 = 0$$

$$\text{CAYLEY-HAMILTON: } A^2 + C_1 A + C_0 I = 0$$

$$\text{POR } \underline{y}: \quad A \underline{x} + C_1 \underline{y} + C_0 \underline{y} = \underline{0}$$

$$A^2 = \begin{pmatrix} 3 & -1 \\ 4 & -2 \end{pmatrix} \begin{pmatrix} 3 & -1 \\ 4 & -2 \end{pmatrix} = \begin{pmatrix} 5 & -1 \\ 4 & 0 \end{pmatrix}; \text{ SEA } \underline{y} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

(23)

SUST:

$$\begin{pmatrix} 5 & -1 \\ 4 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} + C_1 \begin{pmatrix} 3 & -1 \\ 4 & -2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} + C_0 \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 3 \\ 4 \end{pmatrix} + C_1 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + C_0 \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$C_1 + C_0 = -3$$

$$2C_0 = -4$$

$$C_0 = -2, \quad C_1 = -1$$

ECUACION CARACTERISTICA:

$$\lambda^2 + (-1)\lambda + (-2) = 0$$

SE DEBEN OBTENER NUMERICAMENTE LAS RAICES DE LA ECUACION CARACTERISTICA

METODO DE APROXIMACIONES SUCCESIVAS

MAYOR O MENOR EIGENVALOR Y EIGENVECTOR CORRESPONDIENTE

NOVA APROXIMACION A EIGENVECTOR.

$$A \underline{x}_1 = \lambda_1 \underline{x}_1 \quad \text{1ª APROXIMACION A } \lambda$$

2ª APROXIMACION

$$\underline{x}_0 \approx \underline{x}$$

(24)

$$A \underline{x}_0 = \lambda_1 \underline{x}_1$$

$$A \underline{x}_1 = \lambda_2 \underline{x}_2$$

$$A \underline{x}_n = \lambda_{n+1} \underline{x}_{n+1}$$

ALTO: SI  $|\lambda_{n+1} - \lambda_n| < \epsilon$

---

MENOR EIGENVALOR

$$A \underline{x} = \lambda \underline{x}$$

$$A^{-1} A \underline{x} = A^{-1} \lambda \underline{x}$$

$$\underline{x} = \lambda A^{-1} \underline{x}$$

$$A^{-1} \underline{x} = \frac{1}{\lambda} \underline{x}$$

$$A^{-1} \underline{x} = \lambda^* \underline{x}$$

MAYOR  $\lambda \longrightarrow$  MENOR  $\lambda$

$$\lambda^* = \frac{1}{\lambda}$$



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

**INTRODUCCION A LA PROGRAMACION Y  
COMPUTACION ELECTRONICA**

**PROGRAMACION LINEAL SIMPLEX, EL PROBLEMA  
DE TRANSPORTE UN CASO DE APLICACION**

**OCTUBRE, 1982**

## 14. PROGRAMACION LINEAL

1

## 14.1. Introducción

En diversas situaciones para la toma de decisiones relacionadas con el uso eficiente de una serie de recursos o asignación óptima de recursos limitados, es necesario construir un modelo matemático. Si dicho modelo matemático se puede expresar como un sistema de desigualdades lineales, dadas por las restricciones en la disponibilidad de los recursos, y una función que representa la meta u objetivo a alcanzar, también de tipo lineal, se dice que se trata de un problema de programación lineal. La función objetivo generalmente representa costos o utilidades.

En términos generales el sistema de desigualdades lineales tendrá más variables o incógnitas que desigualdades, para su solución se requerirá convertir el sistema de desigualdades en ecuaciones pero el sistema será de tipo indeterminado, sin embargo, la combinación de las restricciones con la función objetivo transforma el sistema determinado en un sistema determinado que tendrá solución única y además dicha solución optimizará la función objetivo.

El propósito de la programación lineal es encontrar dicha solución óptima. De los métodos existentes probablemente el más empleado sea el método Simplex de Dantzig.

La estructura general de un problema de programación lineal es:

$$\begin{aligned} \max \\ \text{ó} \\ \min \end{aligned} Z = \sum_{i=1}^n C_i X_i \quad (14.1)$$

sujeta a:

$$\left. \begin{aligned} X_i &\geq 0, \quad i=1, \dots, n \\ \sum_{i=1}^n A_{ki} X_i &\leq B_k, \quad k=1, \dots, f \end{aligned} \right\} \quad (14.2)$$

2

$$\left. \begin{aligned} \sum_{i=1}^n A_{ki} X_i &\geq B_k, \quad k=g+1, \dots, s \\ \sum_{i=1}^n A_{ki} X_i &= B_k, \quad k=g+1, \dots, n \end{aligned} \right\} \quad (14.2)$$

donde  $m < n$ ,  $Z$  es la función objetivo a optimizar, a los valores  $C_i$  se les denomina coeficientes de costos, las variables  $X_i$  son las incógnitas del problema, a los coeficientes  $A_{ki}$  se les denomina coeficientes estructurales y a las constantes  $B_k$  se les denomina estipulaciones. Las restricciones del problema están dadas por el sistema de desigualdades (14.2)

Las características que debe reunir un problema de programación lineal son:

- 1) que el modelo sea determinístico,
- 2) variables positivas,
- 3) que el único objetivo sea maximizar o minimizar
- 4) divisibilidad de las variables, es decir, que puedan adquirir valores fraccionarios.

Se denomina solución básica factible del sistema:

$$\sum_{i=1}^m A_{ki} X_i = B_k, \quad k=1, \dots, m \quad (14.3)$$

a un conjunto de "m" valores  $X_i$  tal que satisfaga el sistema de ecuaciones (14.3) y que no necesariamente optimiza la función objetivo. A cada una de las "m" variables  $X_i$  se les denomina variables básicas y deberán ser positivas para garantizar una posible solución y evitar que degenera el método a emplear.

Para convertir las desigualdades en igualdades es necesario agregar ciertas variables al sistema como se muestra a continuación.

Sea el sistema de desigualdades:



$$\begin{aligned} x_1 + x_2 + 2x_3 &\leq 10 \\ x_1 - 2x_2 + x_3 &= 5 \\ x_1 &- 3x_3 \geq 12 \end{aligned} \quad (14.4)$$

considerando que toda  $x_i$  es positiva, la conversión de las desigualdades en igualdades se logrará agregando o restando cantidades positivas como se muestra a continuación:

$$\begin{aligned} x_1 + x_2 + 2x_3 + x_4 &= 10 \\ x_1 - 2x_2 + x_3 + x_5 &= 5 \\ x_1 - 3x_3 - x_6 + x_7 &= 12 \end{aligned} \quad (14.5)$$

Las variables  $x_1, x_2, x_3$  se conocen como variables estructurales y tienen significado físico; las variables  $x_4$  y  $x_5$  se conocen como variables de holgura, tienen significado físico y sus coeficientes en la función objetivo son nulos; a las variables  $x_6$  y  $x_7$  se les denomina variables artificiales, no tienen significado físico alguno y se agregan para poder formar la primera solución básica del sistema, pero se debe garantizar su exclusión de la solución óptima del sistema. Esto último se logra asignándoles coeficientes de costos  $C_i$  muy negativos cuando se desea maximizar o mucho mayores que cero cuando se desea minimizar.

#### 14.1 Método Simplex

##### 14.1.1 Objeto

Dado el modelo de programación lineal para un sistema, en la forma:

$$\begin{aligned} \max & \quad z = \sum_{i=1}^n C_i x_i \\ \min & \\ \text{s/a} & \quad \sum_{i=1}^n A_{ki} x_i \geq b_k, \quad k=1, \dots, m \end{aligned} \quad (14.6)$$

obtener los valores  $x_i$  que optimizan la función  $z$  mediante el método Simplex en forma computacional.

##### 14.1.2 Método

Para obtener la solución del modelo (14.6) es necesario convertir las desigualdades en igualdades, o sea, llegar al siguiente sistema de ecuaciones:

$$\begin{aligned} \max & \quad z = \sum_{i=1}^l C_i x_i \\ \min & \\ \text{s/a} & \quad \sum_{i=1}^l A_{ki} x_i = b_k, \quad k=1, \dots, m \end{aligned} \quad (14.7)$$

donde las variables  $x_i$  para  $i=n+1, \dots, l$  son variables de holgura y/o artificiales y los coeficientes  $C_i$  desde  $i=n+1, \dots, l$  tienen el siguiente valor:

$$C_i = \begin{cases} 0 & \text{si corresponde a una variable de holgura} \\ \gg 0 & \text{si corresponde a una variable artificial y se está minimizando } z \\ \ll 0 & \text{si corresponde a una variable artificial y se está maximizando } z \end{cases}$$

A continuación se forma la siguiente tabla que suele denominarse como TABLEAU:

I		II	
Base	C de la base	Sol. básica	$C_1 \ C_2 \ \dots \ C_n \ \dots \ C_L$
$X_1^*$	$C_1^*$	$B_1^*$	$A_{11}^* \ A_{12}^* \ \dots \ A_{1n}^* \ \dots \ A_{1L}^*$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$X_k^*$	$C_k^*$	$B_k^*$	$A_{k1}^* \ A_{k2}^* \ \dots \ A_{kn}^* \ \dots \ A_{kL}^*$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$X_m^*$	$C_m^*$	$B_m^*$	$A_{m1}^* \ A_{m2}^* \ \dots \ A_{mn}^* \ \dots \ A_{mL}^*$
valor de Z	$Z_0$	$Z_1 - C_1 \ Z_2 - C_2 \ \dots \ Z_n - C_n \ \dots \ Z_L - C_L$	

Fig. 14.1 Tabla para aplicar el Método Simplex computacional

La primera solución básica estará formada por las variables de holgura y variables artificiales que tengan coeficiente unitario positivo, habrá "m" variables que cumplan la anterior; las variables que no forman parte de la base tienen valor nulo.

El proceso del método Simplex consiste en ir cambiando las variables que pertenecen a la base en forma sistemática hasta optimizar la función objetivo. Para describir el proceso se considerará que se desea  $\min Z$ , lo anterior no origina ninguna restricción puesto que  $\min Z$  equivale a  $\max | -Z |$ .

Al seleccionar la primera base o solución factible inicial del sistema, los elementos del Tableau de la fig. 14.1 tendrán el siguiente valor:

$$\begin{aligned} A_{k1}^* &= A_{k1} \\ B_k^* &= B_k \\ Z_1 - C_1 &= C_1 \end{aligned}$$

$$Z_0 = C_1^* B_1^* + \dots + C_m^* B_m^*$$

$X_1^*, \dots, X_m^*$  : variables que forman la primera solución básica

$C_1^*, \dots, C_m^*$  : coeficientes de costos correspondientes a las variables de la solución básica

Para resolver el sistema de ecuaciones representado por el Tableau se aplica el método de Gauss-Jordan a la sección II del Tableau, la selección del elemento pivote para efectos de minimización se describe a continuación:

- 1) Analizar los elementos  $Z_i - C_i$  para toda "i" a fin de determinar si ya se optimizó la función objetivo. El mínimo se obtiene cuando  $|Z_i - C_i| \leq 0$  para toda "i", si se cumple lo anterior se detiene el proceso y el valor de las incógnitas que forman la base es la solución óptima. En caso contrario continuar.
- 2) Buscar todos los  $|Z_i - C_i| > 0$  y seleccionar como columna del elemento pivote aquella columna "i" para la que se cumpla que  $|Z_i - C_i| > 0$  es el mínimo.
- 3) Para asegurar la factibilidad de la nueva solución, el renglón "h" del elemento pivote será aquel para el cual se tenga el  $\min_k \{ B_k^* / A_{ki}^* \}$  tal que  $A_{ki}^* > 0$ . "i" corresponde a la columna seleccionada en el paso 2). La solución será no acotada cuando  $A_{ki}^* \leq 0$  para toda "h".
- 4) Introducir en la base la nueva variable básica  $X_i$ , es decir, hacer  $X_k^* = X_i$  y  $C_k^* = C_i$ .
- 5) Aplicar el método de eliminación de Gauss-Jordan a la sección II del Tableau pivotando sobre el elemento  $A_{ki}^*$ . Al efectuar esta eliminación, los nuevos valores  $B^*$  corresponden a los valores de las variables que forman la base  $X^*$ .
- 6) Regresar al paso 1).

Se puede presentar el caso de que el problema sea cíclico o degenerativo, lo cual es una posibilidad muy remota en

problemas reales. En previsión a lo anterior, se debe establecer un máximo número de iteraciones al implementar el algoritmo en una computadora. Si se tiene un sistema con "m" restricciones se requerirán aproximadamente "2m" cambios de base para llegar a la solución óptima, en base a esto se puede establecer el máximo número de iteraciones a efectuar.

En el caso de que aparezcan variables artificiales, si es que se emplearon, en la solución óptima será necesario modificar sus coeficientes de costos y volver a aplicar el método Simplex al sistema de ecuaciones.

Las variables que no forman parte de la base correspondiente a la solución óptima tienen valor nulo.

#### 14.1.3 Descripción del Programa

##### a) Subrutinas requeridas:

Ninguna

##### b) Descripción de las variables:

M	cantidad de renglones de la matriz de coeficientes del sistema de ecuaciones
N	cantidad de columnas de la matriz de coeficientes = variables estructurales + variables de holgura + variables artificiales
B[I,J]	matriz de coeficientes que contiene los elementos $A_{ki}$
PO[I,1]	vector de términos independientes del sistema de ecuaciones o estipulaciones $B_k$
C[I,J]	coeficientes de costos de la función objetivo
ASK	variable que indica el tipo de optimización a efectuar, puede ser MAX o MIN
LA[I,1]	incógnitas que forman parte de la base
COT[I,1]	coeficientes de costos de las incógnitas que forman parte de la base
DI[I,J]	índices que indican cuando detener el proceso, son los valores $Z_k - C_k$

VALZ	valor de la función objetivo
X[I]	caracteres alfanuméricos que se utilizan en la impresión para representar las variables $X_i$
I	contador
K	identificador de las variables que formarán la primera base
ITERA	contador de iteraciones
LLL	variable de reemplazo
LTM	límite de iteraciones
AMAX	valor del máximo índice $Z_k - C_k > 0$
AMIN	valor del mínimo cociente $B_k/A_{kl}$
LR	identificador de la variable que saldrá de la base
LC	identificador de la variable que entrará a la base
ATEMP	variable de reemplazo

##### c) Dimensiones:

Las proposiciones DIMENSION y DATA deberán modificarse cuando:

$$M > 30 \quad \text{y/o} \quad N > 30$$

##### d) Formatos para los datos de entrada:

SEC.	TARJETAS	FORMATO	INFORMACION
1		(II)	M, N
2		(F10.0)	B[I,J], los elementos de la matriz se dan renglón por renglón. Emplear las tarjetas que sean necesarias.
3		(F10.0)	PO[I,1], los términos independientes se dan uno a continuación del otro en una o más tarjetas según se requiera.
4		(F10.0)	C[I,J], los coeficientes de costos se dan uno a conti-

nuación del otro. Emplear  
 las tarjetas que sean neces-  
 arias.  
 ASK puede adoptar dos va-  
 lores:  
 MIN cuando se desea minimi-  
 zar  
 MAX cuando se desea maxim-  
 zar

(43)

-----  
 otros paquetes de datos (opcional)  
 -----

TARJETA EN BLANCO, al fina-  
 lizar toda la información

e) Diagrama de bloques:

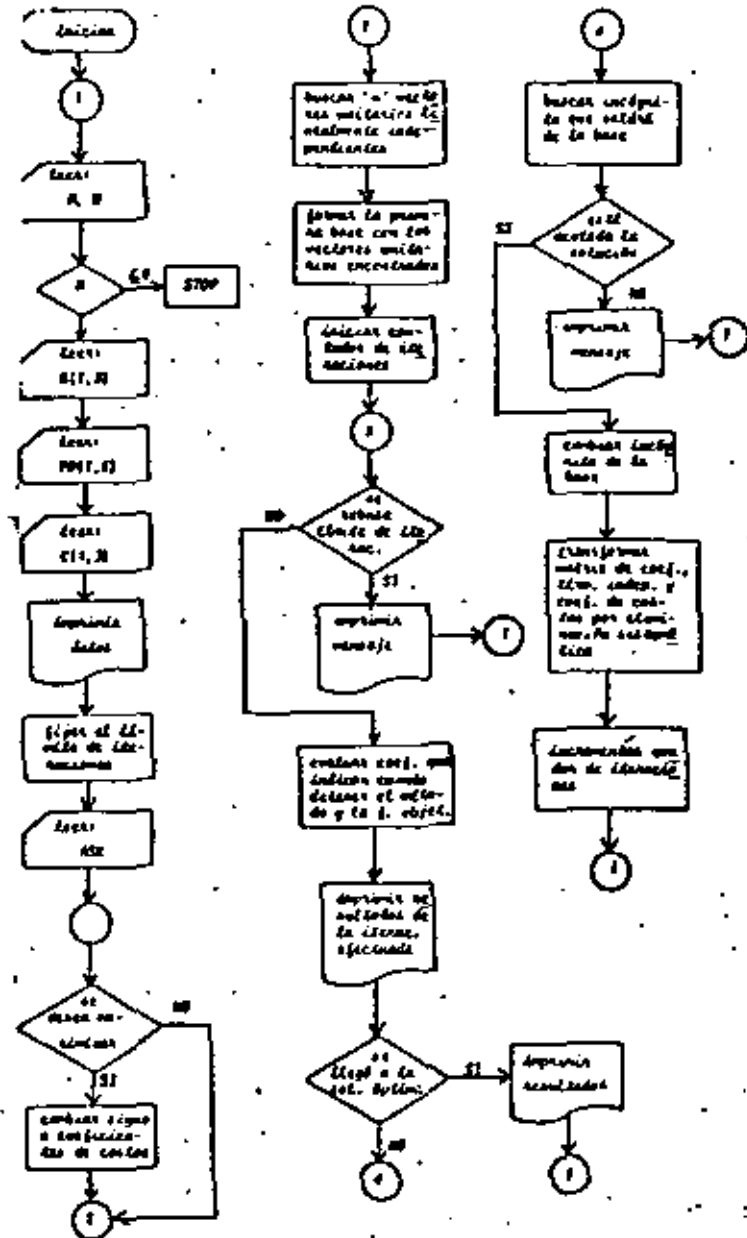


Fig. 14.1 Diagrama de bloques del programa principal

## 6) Listado:

```

C PROGRAMA PARA APLICAR EL METODO SIMPLEX EN UN SISTEMA DE ECUACIONES
C LINEALMENTE INDEPENDIENTES DE ORDEN MAX QUE CONTIENE M VECTORES
C UNITARIOS LINEALMENTE INDEPENDIENTES
C EL SIGNIFICADO DE LAS VARIABLES EMPLEADAS ES
C M NUMERO DE FILAS DE LA MATRIZ DE COEFICIENTES
C N NUMERO DE COLUMNAS DE LA MATRIZ DE COEFICIENTES
C M MATRIZ DE COEFICIENTES
C VECTOR DE CONSTANTES INDEPENDIENTES
C COEFICIENTES DE LA FUNCION A OPTIMIZAR
C ASIMETRIAS CON LA QUE SE INDICA EL TIPO DE OPTIMIZACION (MAX o MIN)
C LA Y COEFICIENTES Y SUS COEFICIENTES EN LA FUNCION A OPTIMIZAR
C QUE FORMAN LA BASE
C UNIDADES QUE INDICAN CUANDO DETENER EL METODO
C VALOR DE LA FUNCION A OPTIMIZAR
C LIMITE DE ITERACIONES

```

```

DIMENSION B(30,30),P(30,1),C(1,30),L(30,1),CO(30,1),X(30),OC(1,3)
10
DATA X(1),X(2),X(3),X(4),X(5),X(6),X(7),X(8),X(9),X(10),X(11),X(12)
1)X(13),X(14),X(15),X(16),X(17),X(18),X(19),X(20),X(21),X(22),X(23)
2)X(24),X(25),X(26),X(27),X(28),X(29),X(30),X(31),X(32),X(33),X(34),X(35)
3)X(36),X(37),X(38),X(39),X(40),X(41),X(42),X(43),X(44),X(45),X(46),X(47)
4)X(48),X(49),X(50),X(51),X(52),X(53),X(54),X(55),X(56),X(57),X(58),X(59)
5)X(60),X(61),X(62),X(63),X(64),X(65),X(66),X(67),X(68),X(69),X(70)
DATA ALF,MINI
C LECTURA DEL ORDEN DE LA MATRIZ DE COEFICIENTES
10
11
1 READ(10,100) M,N
IF(N) 3,3,3
2 CALL SRT
C LECTURA DE LA MATRIZ DE COEFICIENTES
3 DO 4 I=1,M
4 READ(10,150) (C(I,J),J=1,N)
C LECTURA DEL VECTOR DE CONSTANTES INDEPENDIENTES
READ(10,155) (P(I),I=1,M)
C LECTURA DE LOS COEFICIENTES DE LA FUNCION A OPTIMIZAR
READ(10,150) (C(I,J),J=1,N)
C IMPRESION DE DATOS
WRITE(10,100) M,N
WRITE(10,100)
DO 5 J=1,N
5 WRITE(10,150) (C(I,J),I=1,M)
WRITE(10,100)
DO 6 I=1,M
6 WRITE(10,155) P(I,1)
WRITE(10,100)
WRITE(10,250) (C(I,J),J=1,N)
C LECTURA DEL TIPO DE OPTIMIZACION DESEADO
LIM=0
READ(10,150) ASK
IF(ALF=0,ASK) GO TO 8
DO 7 J=1,N
7 C(I,J)=C(I,J)
C BUSQUEDA DE LOS M VECTORES UNITARIOS LINEALMENTE INDEPENDIENTES
8 DO 10 J=1,N
L=0
K=0
DO 9 I=1,M
IF(C(I,J).EQ.0.0) GO TO 9
L=L+1
9 CONTINUE
IF(L.NE.1) GO TO 10
IF(C(I,J).NE.1) GO TO 10
L=1
C=C(I,J)
C=C(I,J)+C(I,J)
10 CONTINUE
ITER=1
11 IF(ITER.EQ.LIM) GO TO 12
WRITE(10,300)
GO TO 1
12 DO 13 I=1,M
13 L(I)=0
VAL=0
C LECTURA DEL VALOR DE LA FUNCION QUE SE ESTA OPTIMIZANDO
DO 14 I=1,M
14 VAL=VAL+C(I,1)+C(I,2)

```

```

C  CALCULO DE LOS COEFICIENTES QUE INDICAN CUANDO OBTENER EL METODO
  DO 17 J=1,N
  DO 16 I=1,M
16  U(I,J)=C(I,J) + B(I,J)*COEF(I,J)
    U(I,J)*PC(I,J) = C(I,J)
17  CONTINUE
C  IMPRESION DE LOS VALORES OBTENIDOS EN LA ITERACION EFECTUADA
  WRITE(IW,25) ITERA
  WRITE(IW,26)
  LC=1
  DO 18 I=1,M
  LLL=L(I)+1
18  WRITE(IW,27) X(LLL),PC(I,J)
    WRITE(IW,28)
    WRITE(IW,29) (C(I,J),J=1,N)
    A(I)=B(I)+1
    LC=LC+1
C  SE INDICA SI YA TERMINO EL PROCESO EN CASO CONTRARIO SE BUSCA LA
C  INCOGNITA QUE ENTRA EN LA BASE
  DO 19 J=1,N
  IF(A(J)*SE,C(I,J)) GO TO 19
  A(I)=A(I)+J
  LC=J
19  CONTINUE
  IF(A(I)>.01) GO TO 21
C  IMPRESION DE LOS VALORES QUE OPTIMIZAN LA FUNCION
  WRITE(IW,40)
  DO 20 I=1,M
  LLL=L(I)+1
20  WRITE(IW,45) X(LLL),PC(I,J)
    WRITE(IW,45) VALZ
    GO TO 1
21  LOG
  NOB
C  BUSQUEDA DE LA INCOGNITA QUE SALDRA DE LA BASE
  DO 22 I=1,M
  IF(B(I)-LC).LE-.01) GO TO 22
  GO TO 22
22  LRL=I
  GO TO 25
23  K=K+1
  IF(K=NE+1) GO TO 24
  A(I)=PC(I)/PC(I,LC)
  LRV=I
  GO TO 25
24  A(I)=PC(I)/B(I)-LC
  IF(A(I).LE.ATEMP) GO TO 25
  A(I)=ATEMP
  LN=I
25  CONTINUE
C  INVESTIGANDO SI LA SOLUCION ESTA ACOTADA
  IF(LRV=0) GO TO 26
  WRITE(IW,46)
  GO TO 1
C  INSERCIÓN DE LA INCOGNITA EN LA BASE
26  LRL=LRL+LC
  C(I,LC)=C(I,LC)
C  TRANSFORMACION DE LA MATRIZ DE COEFICIENTES Y DEL VECTOR DE CON-
C  STANTES INDEPENDIENTES
  ATEMP=B(LR,LC)
  DO 27 J=1,N
  B(LR,J)=B(LR,J)/ATEMP
  PC(LR,J)=PC(LR,J)/ATEMP
  DO 28 I=1,M
  IF(C(I,LC).LR) GO TO 28
  ATEMP=C(I,LC)
  DO 28 J=1,N
  U(I,J)=C(I,J) - ATEMP*C(LR,J)
  PC(I,J)=PC(I,J) - ATEMP*PC(LR,J)
28  CONTINUE
  ITER=ITERA + 1
  GO TO 11

```

IF (ASK, EQ, ALF) WRITE (IW, 700) VALZ  
 IF (ASK, NE, ALF) WRITE (IW, 700) -VALZ

```

C   FORMATEO DE LECTURA E IMPRESION
100 FORMAT (2I3)
150 FORMAT (0F13.0)
200 FORMAT (3I7),45X, 'LA MATRIZ DE COEFICIENTES DEL SISTEMA ES'//)
250 FORMAT (7.5X,2E (13,5.2X))
300 FORMAT (3I7),45X, 'EL VECTOR DE CONSTANTES INDEPENDIENTES ES'//)
350 FORMAT (7.5X,2E (13,5))
400 FORMAT (3I7),42X, 'LOS COEFICIENTES DE LA FUNCION A OPTIMIZAR SON
1'//)
450 FORMAT(1)
500 FORMAT(3I7),54X, 'EL PROBLEMA ES CICLICO'
550 FORMAT(5I7),51X, '*** ITERACION NUMERO 'I2)
600 FORMAT(5I7),10X, 'INCÓGNITAS QUE FORMAN LA BASE'J2X,'VALOR DE LA
IS INCÓGNITAS'//)
650 FORMAT(7.1I7),43X,50X,E13.0)
700 FORMAT(3I7),35X, 'VALOR DE LA FUNCION QUE SE ESTA OPTIMIZANDO 'E
13.0)
750 FORMAT(3I7),35X, 'VALOR DE LOS COEFICIENTES QUE INDICAN CUANDO DE
TENGE EL PROCESO'//)
800 FORMAT(5I7),43X, '*** LOS VALORES QUE OPTIMIZAN LA FUNCION SON'//
17.70X, 'INCÓGNITA'J45X, 'VALOR'//)
850 FORMAT(3I7),41X, 'EL VALOR OPTIMO DE LA FUNCION ES 'E13.0)
900 FORMAT(5I7),51X, 'LA SOLUCION NO ESTA ACOTADA'
950 FORMAT(4I7),5E7),42X, 'EL ORDEN DE LA MATRIZ DE COEFICIENTES ES '
1I2, 'R'J12)
END

```

Fig. 14.3 Listado del programa principal

## 14.2.4 Ejemplo

En un taller se cuenta con tres tipos de máquinas: cortadora (A), torno (B) y taladro (C). En dicho taller se fabrican cuatro productos: 1, 2, 3, 4. Cada producto debe de pasar por cada máquina con la secuencia ABC. El tiempo (horas) empleado por cada máquina para el acabado del producto, la cantidad de horas efectivas de trabajo de cada máquina y la utilidad neta obtenida por la venta de cada producto terminado son:

Tipo de Mq.	Tiempo útil de trabajo	Productos			
		1	2	3	4
A	3,500	2	1.5	3.0	1.5
B	2,000	1.5	3.5	1.5	4.0
C	4,500	2	3.5	4.0	1.5
	utilidad unitaria (\$)	70	90	100	60

Para que la producción sea rentable se requiere que la cantidad de artículos elaborados sea mayor de 500.

Determine la cantidad de artículos de cada tipo que se deben producir en el año para que la ganancia obtenida sea máxima. Considere que todos los productos elaborados son vendidos.

## SOLUCION

El modelo matemático del problema planteado es:

$$\max Z = 70X_1 + 90X_2 + 100X_3 + 60X_4$$

$$s/a \quad 2X_1 + 1.5X_2 + 3.0X_3 + 1.5X_4 \leq 3500$$

$$1.5X_1 + 3.5X_2 + 1.5X_3 + 4.0X_4 \leq 2000$$

$$2X_1 + 3.5X_2 + 4.0X_3 + 1.5X_4 \leq 4500$$

$$X_1 + X_2 + X_3 + X_4 \geq 500$$

Eliminando desigualdades:

$$\max Z = 70X_1 + 90X_2 + 100X_3 + 60X_4 + 0X_5 + 0X_6 + 0X_7 + 0X_8 -$$

$$s/a \quad -10,000X_9$$

$$2X_1 + 1.5X_2 + 3X_3 + 1.5X_4 + X_5 = 3500$$

$$1.5X_1 + 3.5X_2 + 1.5X_3 + 4.0X_4 + X_6 = 2000$$

$$2.0X_1 + 3.5X_2 + 4.0X_3 + 1.5X_4 + X_7 = 4500$$

$$X_1 + X_2 + X_3 + X_4 - X_8 + X_9 = 500$$

TABLA 14.1 Datos para el problema del ejemplo 14.2.4

M = 4

M = 9

B =	2.0	1.5	3.0	1.5	1.0	0.0	0.0	0.0	0.0
	1.5	3.5	1.5	4.0	0.0	1.0	0.0	0.0	0.0
	2.0	3.5	4.0	1.5	0.0	0.0	1.0	0.0	0.0
	1.0	1.0	1.0	1.0	0.0	0.0	0.0	-1.0	1.0

PO =	3500
	2000
	4500
	500

$$C = (70, 90, 100, 60, 0, 0, 0, 0, -10000)$$

ASE = MAX

TABLA 14.2 Resultados del problema del ejemplo 14.2.4



EL ORDEN DE LA MATRIZ DE COEFICIENTES ES DE 4 X 4

LA MATRIZ DE COEFICIENTES DEL SISTEMA ES

0.	.20000E+01	.15000E+01	.30000E+01	.15000E+01	.10000E+01	0.	0.	0.
0.	.15000E+01	.30000E+01	.15000E+01	.40000E+01	0.	.10000E+01	0.	0.
0.	.30000E+01	.15000E+01	.40000E+01	.15000E+01	0.	0.	.10000E+01	0.
	.10000E+01	.10000E+01	.10000E+01	.10000E+01	0.	0.	0.	.10000E+01

EL VECTOR DE CONSTANTES INDEPENDIENTES ES

- .35000E+04
- .10000E+04
- .40000E+04
- .20000E+04

LOS COEFICIENTES DE LA FUNCION A OPTIMIZAR SON

- .70000E+03
- .40000E+02
- .10000E+01
- .10000E+02

\*\*\* ITERACION NUMERO 1 \*

INCÓGNITAS QUE FORMAN LA BASE

VALOR DE LAS INCÓGNITAS

X5	.15000000E+04
X6	.10000000E+04
X7	.45000000E+04
X8	.30000000E+04

VALOR DE LA FUNCION QUE SE ESTA OPTIMIZANDO +.50000000E+07

VALOR DE LOS COEFICIENTES QUE INDICAN CUANDO DETENER EL PROCESO

0.	.10000E+03	.10000E+03	.10000E+03	.10000E+03	0.	0.	0.	.10000E+03
----	------------	------------	------------	------------	----	----	----	------------

INCÓGNITAS QUE FORMAN LA BASE

17

VALOR DE LAS INCÓGNITAS

X5	.28000000E+04
X6	.72300000E+04
X7	.33000000E+04
X3	.90000000E+03

VALOR DE LA FUNCION QUE SE ESTA OPTIMIZANDO .58000000E+05

VALOR DE LOS COEFICIENTES QUE INDICAN CUANDO DETENER EL PROCESO

.100000E+02	.100000E+02	0.	.400000E+02	0.	0.	0.	.100000E+02
.101000E+03							

\*\*\* ITERACION NUMERO 3

INCÓGNITAS QUE FORMAN LA BASE

VALOR DE LAS INCÓGNITAS

X5	.12300000E+01
X6	.01175000E+01
X8	.42300000E+01
X3	.11250000E+04

VALOR DE LA FUNCION QUE SE ESTA OPTIMIZANDO .11250000E+04

VALOR DE LOS COEFICIENTES QUE INDICAN CUANDO DETENER EL PROCESO

.200000E+03	.250000E+01	0.	.225000E+02	0.	0.	.425000E+02	0.
.100000E+03							

\*\*\* ITERACION NUMERO 4

INCÓGNITAS QUE FORMAN LA BASE

VALOR DE LAS INCÓGNITAS

X4	.33333333E+03
X6	.91666667E+04
X8	.43333333E+03
X3	.10000000E+04

VALOR DE LA FUNCION QUE SE ESTA OPTIMIZANDO .12000000E+04

VALOR DE LOS COEFICIENTES QUE INDICAN CUANDO DETENER EL PROCESO

.100000E+03	.700000E+02	0.	0.	.400000E+02	0.	.200000E+02	0.
.100000E+03							

\*\*\* ITERACION NUMERO 3

INCÓGNITAS QUE FORMAN LA BASE

VALOR DE LAS INCÓGNITAS

X4	.14022609E+04
X3	.35832184E+03
X6	.12458770E+04
X2	.28735613E+03

VALOR DE LA FUNCION QUE SE ESTA OPTIMIZANDO .14694253E+04

VALOR DE LOS COEFICIENTES QUE INDICAN CUANDO DETENER EL PROCESO

.03057E+01	0.	0.	0.	.15747E+02	.44274E+01	-.18370E+02	0.
.10000E+03							

\*\*\* ITERACION NUMERO 4

INCÓGNITAS QUE FORMAN LA BASE

VALOR DE LAS INCÓGNITAS

X4	.11086957E+04
X3	.36000000E+03
X6	.14321729E+04
X1	.54347824E+03

VALOR DE LA FUNCION QUE SE ESTA OPTIMIZANDO .14994522E+04

VALOR DE LOS COEFICIENTES QUE INDICAN CUANDO DETENER EL PROCESO

0.	0.	.14607E+02	0.	.18000E+02	.24087E+01	-.13041E+02	0.
.10000E+03							

\*\*\* LOS VALORES QUE OPTIMIZAN LA FUNCION SON

INCÓGNITA

VALOR

X4	.11086957E+04
X3	.36000000E+03
X6	.14321729E+04
X1	.54347824E+03

EL VALOR OPTIMO DE LA FUNCION ES .14994522E+04

## 14.3 Bibliografía

1. AGUILAR J. Rodolfo, "Systems Analysis and Design in Engineering, Architecture, Construction and Planning". Englewood Cliffs N.J.: Prentice Hall Inc., 1973. pp.97-163.
2. GASS Saul, "Linear Programming". New York: Mc Graw-Hill Book Co., 1964. pp.1-31.
3. GEREZ G. Victor y GRIJALVA L. Luis, "El Enfoque de Sistemas". México: Editorial Limusa S.A., 1976. pp.301-345.
4. KUESTER James and NIZE Joe, "Optimization Techniques with FORTRAN". New York: Mc Graw-Hill Book Co., 1975. pp.9-63.
5. KUD S. Shan, "Computer Applications of Numerical Methods". Reading Mass.: Addison-Wesley Co., 1972. pp.346-379.
6. LOONBA N. Paul and TURBAN Efraim, "Applied Programming for Management". New York: Holt Rinehart & Winston Inc., 1974. pp.260-305.

## 16. RUTA CRITICA

## 16.1 Introducción

Todo sistema o proyecto complejo se encuentra formado por gran cantidad de actividades, las cuales pueden o no estar interrelacionadas. Mediante el proceso de jerarquización es posible determinar cuáles de esas actividades deben realizarse antes de poder efectuar algunas otras.

Cada actividad generalmente tendrá asociado un tiempo de duración y la subordinación entre actividades puede ser representada mediante una red gráfica de flujo.

Para poder lograr el correcto avance del proyecto o desarrollo del sistema así como para establecer el control y plan de avance es necesario contar con la información de los tiempos que emplea cada actividad. Por otro lado, el conocimiento acerca de las actividades a las que no se puede modificar su tiempo de inicio o de terminación, así como de las actividades que pueden ser modificadas en su tiempo de inicio o duración permite una asignación más eficiente de recursos, la reasignación de tiempos de duración y la redefinición de relaciones en forma tal que el sistema o proyecto cumpla su objetivo en el tiempo que se le había asignado y en la forma más económica posible.

Un método para efectuar el control antes mencionado es el método de Ruta Crítica (CPM), este método fue desarrollado en 1954 en U.S.A. y es una de las herramientas básicas en la planeación y dirección de proyectos.

## 16.2 Método de la Ruta Crítica

## 16.2.1 Objeto

Dada una serie de actividades que forman parte de un proyecto, las cuales pueden ser simultáneas o secuenciales así como sus tiempos de ejecución, determinar:

a) actividades que forman parte de la ruta crítica, son aquellas que no pueden adelantarse ni retrasarse y que por lo tanto con

- a) el tiempo total de duración del proyecto,  
 b) los tiempos de holgura existentes entre actividades secuenciales que no pertenecen a la ruta crítica y,  
 c) el tiempo total necesario para llevar a cabo el proyecto.

### 16.2.2 Método

Para la representación gráfica de la secuencia de actividades se utilizan nodos (círculos) y ramas (segmentos dirigidos). Los nodos representan eventos o actividades y los segmentos dirigidos la relación entre actividades. A toda red se le asigna dos nodos o actividades ficticias de duración nula: el nodo inicial y el nodo final.

Como ejemplo de una red de secuencia de actividades se tiene:

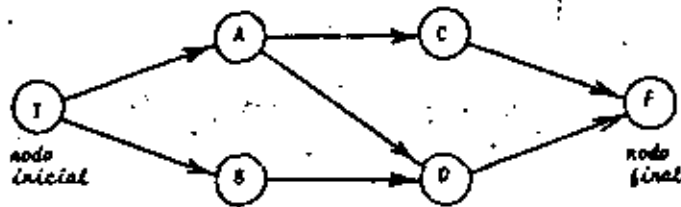


Fig. 16.1 Red gráfica de la secuencia de actividades

La duración de una actividad es el tiempo necesario para llevarla a cabo, dicho tiempo se determina en base al conocimiento que tenga del proyecto el encargado del mismo. Como se mencionó con anterioridad, la duración del evento inicial y del evento final es nula.

Para la descripción del método es necesario definir algunos conceptos:

- 1) Tiempo más próximo de inicio (EST) de una actividad, es lo más pronto que puede comenzarse su ejecución.
- 2) Tiempo más próximo de terminación (EFT) de una actividad, es igual al EST de la actividad más su duración.
- 3) Tiempo más lejano de terminación (LFT) de una actividad, es

el tiempo más remoto en que se puede terminar dicha actividad sin retrasar el inicio de cualquier actividad subordinada.

- 4) Tiempo más lejano de inicio (LST) de una actividad, es igual al LFT de la actividad menos su duración.
- 5) Tiempo de Holgura Total de una actividad, es la cantidad total de tiempo que se puede retrasar la actividad sin retrasar el tiempo de terminación del proyecto.
- 6) Tiempo de Holgura Libre de una actividad, es la cantidad de tiempo que se puede retrasar la actividad sin retrasar cualquier otra actividad.

Los primeros cuatro conceptos se representan en la gráfica de la red en la siguiente forma:

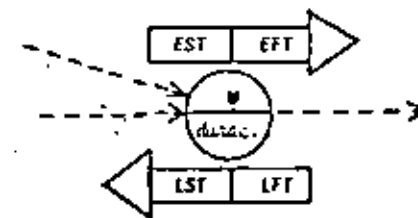


Fig. 16.2 Representación gráfica de una actividad con su duración, EST, EFT, LST, LFT.

Para obtener los valores EST y EFT se recorre la red en sentido directo desde el nodo inicial hasta el nodo final a través de todas sus ramas y para la "i-ésima" actividad de la red se tendrá:

$$EST_i = \max EFT \text{ de todas las actividades precedentes inmediatas de la actividad "i" } \quad (16.1)$$

$$EFT_i = EST_i + \text{Duración de la actividad} \quad (16.2)$$

Los valores LST y LFT se obtienen recorriendo la red en sentido inverso desde el nodo final hasta el nodo inicial a través de todas las ramas y para la "i-ésima" actividad se tendrá:

$LFT_i = \min$  LST de todas las actividades de las cuales la actividad "i" es precedente inmediato [16.3]

$LST_i = LFT_i - \text{Duración}$  [16.4]

Los tiempos de holgura para una actividad "i" se evalúan como a continuación se indica:

[HOLGURA TOTAL] $_i = LST_i - EST_i = LFT_i - EFT_i$  [16.5]

[HOLGURA LIBRE] $_i = (\min EST \text{ de todas las actividades de las cuales la actividad "i" es precedente inmediato}) - EFT_i$  [16.6]

Siempre se cumple que:

HOLGURA LIBRE  $\leq$  HOLGURA TOTAL [16.7]

La ruta crítica está formada por todas las actividades que tienen holgura total nula.

Para desarrollar el proceso en una computadora digital es necesario proporcionar las conexiones existentes entre las actividades mediante una matriz binaria  $G$ . Si existen "n" actividades, incluyendo las ficticias, la matriz será de orden (n,n) y sus elementos estarán dados en la siguiente forma:

$$G_{ij} = \begin{cases} 0 & \text{si } i = j \\ 1 & \text{si la actividad "j" es precedente inmediato de la actividad "i"} \\ 0 & \text{si la actividad "j" no es precedente inmediato de la actividad "i"} \end{cases}$$

### 16.2.3 Descripción del Programa

a) Subrutinas requeridas:  
Ninguna.

### b) Descripción de las variables:

N	cantidad de actividades incluyendo las dos actividades ficticias
G[I, J]	matriz binaria $G$ que indica cuáles actividades se encuentran directamente interconectadas
D[I]	vector con la duración de todas las actividades
GB[I, J]	matriz binaria que da las relaciones directas entre actividades al efectuar el recorrido de la red en sentido inverso contados
CS(I)	fecha más próxima de inicio
EST(I)	fecha más próxima de terminación
EFT(I)	fecha más lejana de inicio
LST(I)	fecha más lejana de terminación
MCON	contados
FF(I)	holgura libre
TF(I)	holgura total
RC(I)	variable en la que se archivan las actividades que forman parte de la ruta crítica
L(I)	Identificador de todas las actividades precedentes o consecuentes en forma inmediata a una actividad
IDASH(I)	guión (-) se emplea para impresión

### c) Dimensiones:

La proposición INTEGER deberá modificarse cuando:  
 $N > 100$

### d) Formatos para los datos de entrada:

SEC. TARJETAS	FORMATO	INFORMACION
1	(I3)	N
2	(40I2)	G(I, J) los elementos de la matriz binaria se dan renglón por renglón empleando la cantidad de -







16.2.4 Ejemplo

Obtener para la jerarquía no trivial del ejemplo 15.2.4 la ruta crítica de las actividades si la duración de las mismas es como se muestra a continuación:

Actividad	Duración (segundos)
A) Detener el coche	8
B) Poner freno de mano	2
C) Bajar del vehículo y chequear llanta mala	30
D) Abaja cajuela	30
E) Sacar llanta buena	30
F) Sacar gato y ponerlo	40
G) Sacar llave de cruz	15
H) Quitar tapón	20
I) Aflojar y quitar tuercas	60
J) Subir coche	35
K) Desmontar llanta mala	15
L) Hortal llanta buena	25
M) Apretar tuercas y poner tapón	70
N) Bajar el coche y guardar gato	20
O) Guardar llanta mala	20
P) Guardar llave de cruz	15
Q) Cerrar cajuela	5
R) Subirse al coche y quitar freno de mano	15
S) Arrancar	10

\* SOLUCION

La red de actividades aparece en la Fig. 16.5

Fig. 16.5 Red de la secuencia de actividades del problema del ejemplo 16.2.4

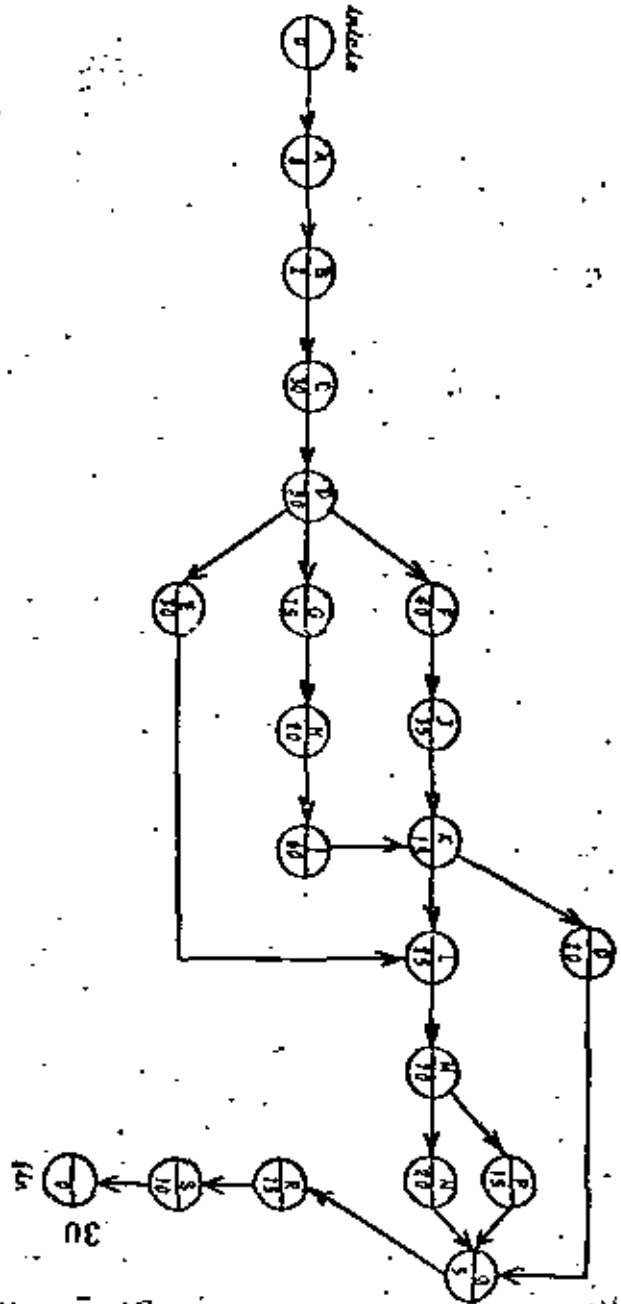


TABLA 16.1 Datos para el problema del ejemplo 16.1.4

<u>Actividad</u>	<u>Índice asociado</u>
Inicio	1
A	2
B	3
C	4
D	5
E	6
F	7
G	8
H	9
I	10
J	11
K	12
L	13
M	14
N	15
O	16
P	17
Q	18
R	19
S	20
FZn	21

$D = (0, 0, 2, 30, 30, 30, 40, 15, 20, 40, 55, 15, 25, 70, 20, 20, 15, 5, 15, 10, 0)$

$N = 21$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

TABLA 16.2 Resultados del ejemplo 16.1.4

MATRIZ DE SUBORDINACION DE ACTIVIDADES

ACTIVIDAD	DURACION	EST	EFT	EST	LFT	FF	TF
1	0:00	7:00	0:00	0:00	0:00	0:00	0:00
2	0:00	0:00	0:00	0:00	0:00	0:00	0:00
3	2:00	7:00	10:00	8:00	10:00	0:00	0:00
4	10:00	10:00	40:00	10:00	40:00	0:00	0:00
5	10:00	40:00	70:00	40:00	70:00	0:00	0:00
6	10:00	70:00	100:00	70:00	100:00	0:00	0:00
7	40:00	70:00	110:00	90:00	110:00	0:00	0:00
8	12:00	70:00	85:00	70:00	85:00	0:00	0:00
9	20:00	65:00	105:00	85:00	105:00	0:00	0:00
10	40:00	105:00	145:00	105:00	145:00	0:00	0:00
11	35:00	110:00	145:00	130:00	145:00	20:00	20:00
12	15:00	145:00	180:00	145:00	180:00	0:00	0:00
13	15:00	180:00	205:00	180:00	205:00	0:00	0:00
14	15:00	205:00	225:00	205:00	225:00	0:00	0:00
15	15:00	205:00	275:00	205:00	275:00	0:00	0:00
16	15:00	275:00	295:00	275:00	295:00	0:00	0:00
17	15:00	295:00	325:00	295:00	325:00	0:00	0:00
18	5:00	325:00	300:00	295:00	300:00	0:00	0:00
19	15:00	300:00	315:00	300:00	315:00	0:00	0:00
20	10:00	315:00	325:00	315:00	325:00	0:00	0:00
21	0:00	325:00	325:00	325:00	325:00	0:00	0:00

LOS RESULTADOS OBTENIDOS SON

ACTIVIDAD	DURACION	EST	EFT	EST	LFT	FF	TF
1	0:00	7:00	0:00	0:00	0:00	0:00	0:00
2	0:00	0:00	0:00	0:00	0:00	0:00	0:00
3	2:00	7:00	10:00	8:00	10:00	0:00	0:00
4	10:00	10:00	40:00	10:00	40:00	0:00	0:00
5	10:00	40:00	70:00	40:00	70:00	0:00	0:00
6	10:00	70:00	100:00	70:00	100:00	0:00	0:00
7	40:00	70:00	110:00	90:00	110:00	0:00	0:00
8	12:00	70:00	85:00	70:00	85:00	0:00	0:00
9	20:00	65:00	105:00	85:00	105:00	0:00	0:00
10	40:00	105:00	145:00	105:00	145:00	0:00	0:00
11	35:00	110:00	145:00	130:00	145:00	20:00	20:00
12	15:00	145:00	180:00	145:00	180:00	0:00	0:00
13	15:00	180:00	205:00	180:00	205:00	0:00	0:00
14	15:00	205:00	225:00	205:00	225:00	0:00	0:00
15	15:00	205:00	275:00	205:00	275:00	0:00	0:00
16	15:00	275:00	295:00	275:00	295:00	0:00	0:00
17	15:00	295:00	325:00	295:00	325:00	0:00	0:00
18	5:00	325:00	300:00	295:00	300:00	0:00	0:00
19	15:00	300:00	315:00	300:00	315:00	0:00	0:00
20	10:00	315:00	325:00	315:00	325:00	0:00	0:00
21	0:00	325:00	325:00	325:00	325:00	0:00	0:00

16.3 Bibliografía

1. AGUILAR J. Rodolfo, "Systems Analysis and Design in Engineering, Architecture, Construction and Planning". Englewood Cliffs N.J.: Prentice Hall Inc., 1973. pp. 126-163.
2. BARNETSON Paul, "Critical Path Planning". Great Britain: Newnes Books, 1961. pp. 1-43.
3. GEREZ G. Victor y GRIJALVA I. Luis, "El Enfoque de Soluciones". México: Editorial Limusa S.A., 1976. pp. 362-379.
4. GOMEZ C. Guillermo, "Planeación y Organización de Empresas". México: Editorial Diseño y Composición Litográfica S.A., 1974. pp. 66-91.

MODELO 1

EL PROBLEMA DE LA DIETA Y SU DUAL

DEFINICION DEL PROBLEMA DE LA DIETA.

Suponga que un dietista está tratando de seleccionar una combinación de cinco tipos de alimentos (naranja, manzana, lechuga, chicharo y zanahoria) de manera que el alimento resultante de esta combinación reúna ciertos requerimientos nutricionales y tenga un costo mínimo. Los requerimientos nutricionales que debe tener el alimento resultante es de al menos 21 unidades de vitamina A y al menos 12 unidades de vitamina B. Las propiedades de los cinco elementos disponibles son:

ALIMENTO	CONTENIDO DE VITAMINA A POR UNIDAD DE ALIMENTO.	CONTENIDO DE VITAMINA B POR UNIDAD DE ALIMENTO.	COSTO POR UNIDAD DE ALIMENTO
1 (Naranja)	1	0	20
2 (Manzana)	0	1	20
3 (Lechuga)	1	2	31
4 (Chichoro)	1	1	11
5 (Zanahoria)	2	1	12

El problema a que se enfrenta el dietista se puede modelar como un problema de programación lineal (o programa lineal), de la siguiente manera.

Sea  $x_i$  la cantidad de alimento  $i$  ( $i = 1, 2, \dots, 5$ ) que debe estar en el alimento resultante de la combinación de los cinco alimentos. Por lo tanto, el costo de introducir el alimento  $i$  en la mezcla será su costo unitario por la cantidad  $x_i$  que está presente en la mezcla. El costo total de la combinación de los cinco alimentos será la suma de los costos al combinar  $x_1, x_2, \dots, x_5$  unidades de cada alimento, i.e. si  $z$  es el costo total entonces

$$z = 20x_1 + 20x_2 + 31x_3 + 11x_4 + 12x_5$$

Ya que el objetivo del dietista es minimizar este costo total, entonces este objetivo se puede representar a través de la siguiente función objetivo

$$\min z = 20x_1 + 20x_2 + 31x_3 + 11x_4 + 12x_5 \tag{1}$$

Los requerimientos nutricionales de vitamina A se pueden representar en la siguiente forma: Si el alimento nutricional  $i$  está presente en una cantidad  $x_i$  entonces proporciona una cantidad de vitamina A igual al producto de vitamina A que contiene una unidad de alimento por la cantidad  $x_i$ . La cantidad total proporcionada por los cinco alimentos será la suma de vitamina A con que contribuye cada alimento y esta deberá ser mayor que el contenido mínimo requerido que es de 21 unidades, i.e.

$$x_1 + x_3 + x_4 + 2x_5 \geq 21 \tag{2}$$

Similarmente, los requerimientos de vitamina B se pueden representar por

$$x_2 + 2x_3 + x_4 + x_5 \geq 12 \quad (3)$$

37

Por último, otra restricción que debe estar presente en el problema del dietista es que la cantidad  $x_i$  que interviene en la mezcla debe ser mayor o igual a cero, i.e.

$$x_i \geq 0 \quad i = 1, 2, \dots, 5 \quad (4)$$

Esta restricción es impuesta ya que no tiene sentido hablar de que una cantidad negativa  $x_i$  está formando parte de la combinación de alimentos.

En resumen el problema del dietista es encontrar valores  $x_1, x_2, \dots, x_5$  para los cuales la función objetivo (1) alcance su mínimo y satisfagan las restricciones (2), (3) y (4). Reescribiendo las ecuaciones del (1) al (4), el problema del dietista está simbólicamente dado por

$$\min z = 20x_1 + 20x_2 + 31x_3 + 11x_4 + 12x_5 \quad (*)$$

$$x_1 + x_3 + x_4 + 2x_5 \geq 21 \quad (**)$$

$$x_2 + 2x_3 + x_4 + x_5 \geq 12 \quad (**)$$

$$x_i \geq 0$$

La formulación anterior, (\*) y (\*\*), se acostumbra representar en un tablero (llamado también tableau) que aparece abajo. Esta representación es solo una abreviación de escritura (a manera de una taquígrafía de programación lineal) que es útil en el algoritmo de solución, en el proceso convencional al procesar el problema por computador y por una gran claridad en la formulación del problema dual que se presentará después. La representación de un programa lineal en forma de tablero consiste representar cada ecuación o desigualdad únicamente por los coeficientes de las variables omitiendo la escritura de sus correspondientes variables. Para conocer a que variable pertenece un coeficiente que aparece en este esquema se da la posición del coeficiente, escribiéndolo en la columna encabezada por la variable que le corresponde.

Para nuestro problema (\*) y (\*\*), la representación a través de un tablero está dada por

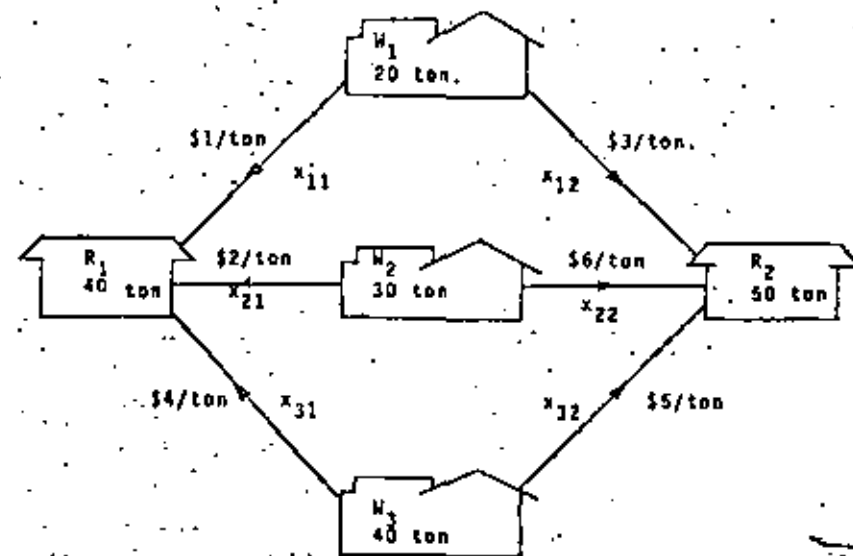
$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$z$ (min)
20	20	31	11	12	$\geq 21$
0	1	2	1	1	$\geq 12$

$$x_i \geq 0$$

MODELO 2

38

Una compañía tiene tres almacenes  $W_1, W_2,$  y  $W_3,$  y dos tiendas de ventas al por menor,  $R_1, R_2.$  Las demandas en las tiendas al por menor y el inventario en los almacenes, se muestra en las respectivas cajas de la siguiente figura. Los costos de envío por tonelada también se muestran en la figura. La compañía desea determinar la manera de realizar los envíos en forma tal que minimice los costos totales de envíos, satisfaga las demandas de las tiendas de menudeo, y no excedan los inventarios en los almacenes.



Sea  $x_{ij}$  las toneladas del almacén  $W_i$  a la tienda de menudeo  $R_j$ . Entonces  $x_{32}$  representa el tonelaje enviado del almacén  $W_3$  a la tienda de menudeo  $R_2$ .

Si  $z$  representa el costo total de envíos, entonces nuestro problema se puede formular por:

$$\min z = 1x_{11} + 3x_{12} + 2x_{21} + 6x_{22} + 4x_{31} + 5x_{32} \quad (*)$$

Sujeta a

$$\begin{aligned} \text{Restricciones sobre disponibilidad de almacenes} \\ x_{11} + x_{12} &\leq 20 \\ x_{21} + x_{22} &\leq 30 \\ x_{31} + x_{32} &\leq 40 \end{aligned}$$

$$\begin{aligned} \text{Restricciones sobre la demanda en tiendas de menudeo} \\ x_{11} + x_{21} + x_{31} &\geq 40 \\ x_{12} + x_{22} + x_{32} &\geq 50 \end{aligned} \quad (**)$$

La formulación anterior, (\*) y (\*\*), se acostumbra representar (por conveniencia del algoritmo de solución y del proceso convencional en el procesamiento en computadora) en la siguiente tabla:

	$x_{11}$	$x_{12}$	$x_{21}$	$x_{22}$	$x_{31}$	$x_{32}$	
	1	3	2	6	4	5	$= z$
	1	1	0	0	0	0	$\leq 20$
	0	0	1	1	0	0	$\leq 30$
	0	0	0	0	1	1	$\leq 40$
	1	0	1	0	1	0	$\geq 40$
	0	1	0	1	0	1	$\geq 50$

REPRESENTACION MATRICIAL. La formulación (\*) y (\*\*), se puede representar matricialmente como sigue:

$$\min z = \begin{bmatrix} 1 & 3 & 2 & 6 & 4 & 5 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \\ x_{31} \\ x_{32} \end{bmatrix}$$

sujeta a

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \\ x_{31} \\ x_{32} \end{bmatrix} \begin{matrix} \leq 20 \\ \leq 30 \\ \leq 40 \\ \geq 40 \\ \geq 50 \end{matrix}$$

COMENTARIOS. El problema de programación lineal anterior ocurre tan frecuentemente en la práctica, que se le ha dado un nombre especial: el problema de transporte. Los problemas de transporte en general, tienen tablas raras (o matrices raras), lo cual significa que la tabla tiene muchos ceros o sea pocos elementos distintos de cero. Dantzig y otros han desarrollado métodos especiales para la solución rápida de estos problemas...

Otro comentario importante, es la característica que presentan cada una de las columnas de la matriz de restricciones; observe que cada una tiene dos unos y los demás elementos son todos ceros.

Un inversionista tiene disponibles las actividades financieras A y B, al comienzo de cada uno de los siguientes cinco años. Cada peso invertido en A, al comienzo de un año, le regresa \$ 1.40 (una ganancia de \$ 0.40) dos años más tarde (en el momento preciso para una reinversión inmediata). Cada peso invertido en B al comienzo de un año, le regresa \$ 1.70 tres años después.

Además existen dos actividades financieras C y D que estarán disponibles solamente una vez en el futuro. Cada peso invertido en C en el comienzo del segundo año le regresa \$ 2.00 cuatro años más tarde. Cada peso invertido en D, en el comienzo del quinto año le regresa \$ 1.30 un año más tarde.

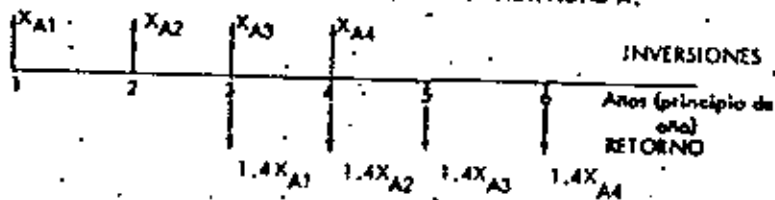
El inversionista comienza con \$ 10,000.00. El deseo conocer que plan de inversión maximiza la cantidad de dinero que el pueda acumular al comienzo del sexto año. Formule un modelo de programación lineal para este problema y también exprésalo en forma tabular.

**SOLUCION.**

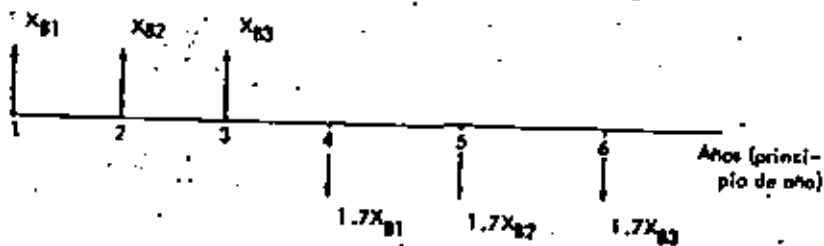
Sea  $X_i$  la cantidad de dinero invertida en la actividad  $i$  ( $i = A, B, C, D$ ) en el año  $i$  ( $i = 1, 2, 3, 4, 5$ ).

Las características dadas sobre las formas de inversión de cada una de las actividades A, B, C y D puedan mostrarse esquemáticamente como sigue.

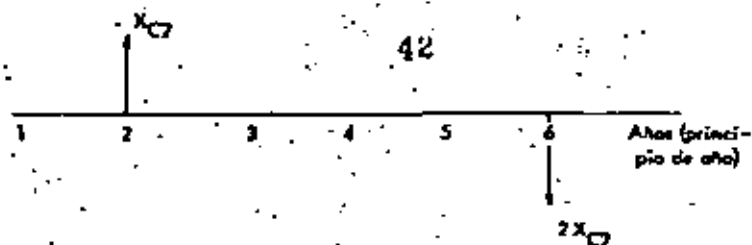
**CONDICIONES DE INVERSION EN LA ACTIVIDAD A.**



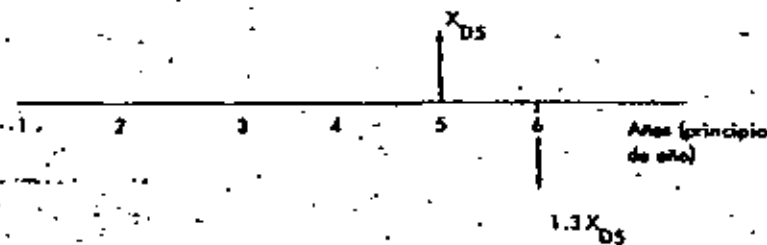
**CONDICIONES DE INVERSION EN LA ACTIVIDAD B**



**CONDICIONES DE INVERSION EN LA ACTIVIDAD C:**



**CONDICIONES DE INVERSION EN LA ACTIVIDAD D:**



La cantidad acumulada en el comienzo del sexto año es la cantidad original (10000) más la ganancia obtenida hasta esta fecha. Por lo tanto, el problema de maximizar la cantidad acumulada de dinero es equivalente a minimizar la ganancia, ya que la cantidad original disponible es una constante que no afecta el valor del dinero acumulado a través de cualquier plan de inversión que se siga.

Si  $Z$  es la ganancia total obtenida hasta el comienzo del sexto año, entonces la función objetivo será:

$$\max Z = 0.4 X_{A1} + 0.4 X_{A2} + 0.4 X_{A3} + 0.4 X_{A4} + 0.70 X_{B1} + 0.70 X_{B2} + X_{C2} + 0.3 X_{D5}$$

Del enunciado del problema, se observa que las restricciones al problema están dadas por la cantidad disponible para invertir en cada año, y por las características de las actividades A, B, C y D. Estas restricciones sobre las inversiones anuales se determinan como sigue:

**PRIMER AÑO:** La cantidad de dinero invertida en el primer año debe satisfacer:

$$X_{A1} + X_{B1} = 10000$$

Si  $U_j$  es una variable positivo o cero, que se adiciona a la desigualdad anterior, para que

esta desigualdad llegue a ser una igualdad, entonces

$$\begin{aligned} x_{A1} + x_{B1} + u_1 &= 10000 \\ u_1 &\geq 0 \end{aligned} \quad (1)$$

## NOTAS:

1. A la variable que se adiciona a una desigualdad para convertirla en igualdad se le llama una variable de holgura. Entonces  $u_1$  es una variable de holgura.
2. Observe que  $u_1$  representa la cantidad de dinero no invertido en el primer año, y por lo tanto también representa la cantidad disponible para invertir en el segundo año.

SEGUNDO AÑO: Las inversiones en este año deben satisfacer (observe en las figuras anteriores en que actividades financieras podemos invertir para el segundo año):

$$x_{A2} + x_{B2} + x_{C2} \leq u_1$$

Si introducimos una variable positiva  $u_2$  para pasar la desigualdad anterior a igualdad, entonces

$$\begin{aligned} x_{A2} + x_{B2} + x_{C2} + u_2 &= u_1 \\ u_2 &\geq 0 \end{aligned} \quad (2)$$

Observese que la variable  $u_2$  es una variable de holgura que representa la cantidad no invertida en el segundo año.

TERCER AÑO: En este año la cantidad de dinero disponible para inversiones proviene de tres fuentes:

- i) cantidad no invertida en el segundo año:  $u_2$
- ii) ganancia obtenida de inversiones anteriores:  $0.4x_{A1}$
- iii) cantidad recuperada de inversiones anteriores:  $\frac{x_{A1}}{u_2 + 1.4x_{A1}}$

Observando cada uno de los cuatro diagramas mostrados anteriormente, se tiene que para el tercer año las inversiones deben satisfacer

$$x_{A3} + x_{B3} \leq u_2 + 1.4x_{A1}$$

Introduciendo una variable de holgura  $u_3$  ( $u_3 \geq 0$ ), se tiene que

$$\begin{aligned} x_{A3} + x_{B3} + u_3 &= u_2 + 1.4x_{A1} \\ u_3 &\geq 0 \end{aligned} \quad (3)$$

Otra vez notese que  $u_3$  representa la cantidad no invertida en el tercer año.

CUARTO AÑO: En forma similar al análisis del tercer año, se tienen tres fuentes de dinero disponibles:

- i) cantidad no invertida en el tercer año:  $u_3$
- ii) ganancia obtenida entre el tercer y cuarto periodo:  $0.4x_{A2} + 0.7x_{B2}$
- iii) cantidad recuperada de inversiones anteriores:  $\frac{x_{A2} + x_{B1}}{u_3 + 1.4x_{A2} + 1.7x_{B1}}$

Por lo tanto, las inversiones en el cuarto periodo deben satisfacer

$$x_{A4} \leq u_3 + 1.4x_{A2} + 1.7x_{B1}$$

Introduciendo la variable de holgura positiva  $u_4$ , se tiene:

$$\begin{aligned} x_{A4} + u_4 &= u_3 + 1.4x_{A2} + 1.7x_{B1} \\ u_4 &\geq 0 \end{aligned} \quad (4)$$

QUINTO AÑO: La cantidad disponible en este periodo proviene de:

- i) cantidad no invertida en el cuarto año:  $u_4$
- ii) ganancia entre el tercero y el cuarto periodo:  $0.4x_{A3} + 0.7x_{B2}$
- iii) cantidad recuperada entre el periodo 3 y 4to.:  $\frac{x_{A3} + x_{B2}}{u_4 + 1.4x_{A3} + 1.7x_{B2}}$

Por lo tanto,

$$x_{D5} \leq u_4 + 1.4x_{A3} + 1.7x_{B2}$$

Si  $u_5$  es una variable de holgura, entonces

$$\begin{aligned} x_{D5} + u_5 &= u_4 + 1.4x_{A3} + 1.7x_{B2} \\ u_4, u_5 &\geq 0 \end{aligned} \quad (5)$$

Por lo tanto, nuestro modelo de programación lineal quedaría definido por la función objetivo, dada anteriormente y el conjunto de restricciones definidas por la ecuación del (1) a la (5).



Reescribiendo las ecuaciones anteriores, nuestro modelo de programación lineal queda expresado por:

$$\max z = 0.4x_{A1} + 0.4x_{A2} + 0.4x_{A3} + 0.4x_{A4} + 0.7x_{B1} + 0.7x_{B2} + 0.7x_{B3} + x_{C2} +$$

$$0.3x_{D5} \quad (0)$$

sujeto a (s.s.):

$$x_{A1} + x_{B1} + u_1 = 10\,000 \quad (1)$$

$$x_{A2} + x_{B2} + x_{C2} + u_2 = u_1 \quad (2)$$

$$x_{A3} + x_{B3} = u_2 + 1.4x_{A1} \quad (3)$$

$$x_{A4} + u_4 = u_3 + 1.4x_{A2} + 1.7x_{B1} \quad (4)$$

$$x_{D5} + u_5 = u_4 + 1.4x_{A3} + 1.7x_{B2} \quad (5)$$

$$x_{Aj} \geq 0 \quad (j = 1, 2, 3, 4)$$

$$x_{Bj} \geq 0 \quad (j = 1, 2, 3)$$

$$x_{C2} \geq 0$$

$$x_{D5} \geq 0$$

$$u_i \geq 0 \quad (i = 1, 2, \dots, 5)$$

Este problema expresado en la forma particionada

x	
c	r
A	b
x ≥ 0	

Se presenta a continuación:

La Compañía aérea Aeronaves del Pacífico, necesita decidir cuántas aeronaves contratar y adiestran en los próximos 6 meses. Los requerimientos expresados como horas-vuelo-aeronave son:

8000 en Enero; 9000 en Febrero; 7000 en Marzo; 10 000 en Abril; 9000 en Mayo; y 11 000 en Junio.

El entrenamiento para que una aeronave dé servicio en un vuelo dura un mes, por tanto cada muchacha debe contratarse por lo menos un mes antes de ser necesitada.

El entrenamiento requiere de 100 horas de supervisión de aeronaves ya entrenadas por lo tanto disponemos de 100 horas-vuelo-aeronave menos, durante un mes por cada aeronave en entrenamiento.

Cada aeronave entrenada puede trabajar hasta 150 horas en un mes y la compañía tiene 60 aeronaves entrenadas al principio de enero.

Si el máximo tiempo disponible de las aeronaves excede el requerido en el mes (horas vuelo + supervisión) trabajarán menos de 150 horas y no es despedido ninguna. Pero en cada mes, aproximadamente el 10% de las aeronaves con experiencia dejan el trabajo por matrimonio u otras razones.

Cada aeronave entrenada cuesta a la compañía \$ 8000.00 el mes y cada aeronave en entrenamiento \$ 4000.00, tomando en cuenta salarios y otros beneficios.

- Formule el problema de contratar y entrenar como un modelo de programación lineal haciendo  $x_i$  el número de aeronaves que principian su entrenamiento en el mes  $i$ , donde  $x_0 = 60$  representa las aeronaves disponibles al principio de enero. Defina cualquier símbolo adicional que necesite para expresar las variables de decisión.
- El inicio anterior supone un horizonte de 6 meses. Suponga que se agregan requerimientos de julio al modelo, por ejemplo 10 000 horas. ¿Cambiaría necesariamente la solución para los meses anteriores encontrado anteriormente? Explíquela.

Solución:

47

Sea  $x_t$  el número de personas contratadas que principian su entrenamiento al inicio del mes  $t$  ( $t = 1, 2, \dots, 6$ ).

Sea  $y_t$  el número de aeromozas experimentadas al final del mes  $t$  ( $t = 1, 2, \dots, 6$ ). Nótese que  $y_t$  también representa la cantidad de aeromozas experimentadas al inicio del mes  $t+1$ .

DISPONIBILIDAD DE AEROMOZAS EXPERIMENTADAS.

Observe que el número de aeromozas experimentadas  $y_t$  al final del mes  $t$ , está formado por las personas contratadas al inicio de este mes (las cuales fueron entrenadas en el transcurso del mes) más el 90% de las aeromozas experimentadas que había al final del mes anterior  $t-1$  (o sea al inicio del mes  $t$ ), lo:

$$y_t = x_t + .9 y_{t-1} \quad (t = 1, 2, \dots, 6)$$

con  $y_0 = x_0 = 60$  (\*)

ó sea  $y_1 = x_1 + .9 y_0 = x_1 + .9 x_0$  (1)

$y_2 = x_2 + .9 y_1$  (2)

$y_3 = x_3 + .9 y_2$  (3)

$y_4 = x_4 + .9 y_3$  (4)

$y_5 = x_5 + .9 y_4$  (5)

$y_6 = x_6 + .9 y_5$  (6)

DEMANDAS DE HORAS DE TRABAJO (VUELOS COMERCIALES Y ENTRENAMIENTO):

La demanda total de horas de vuelo por mes corresponde a la demanda de vuelos comerciales más la demanda de horas para entrenar a las nuevas personas contratadas en el inicio del mes. Para satisfacer esta demanda total en el mes  $t$  (inicio del mes  $t$ ); se dispone de  $y_{t-1}$  aeromozas con experiencia, las cuales pueden proporcionar 150 horas cada una de ellas. Por lo tanto, si  $D_t$  es la demanda de vuelos comerciales en el mes  $t$ , entonces:

Demanda en el mes  $t$ :  $150 y_{t-1} = D_t + 100 x_t \quad (t=1, 2, \dots, 6)$

con  $y_0 = x_0$

Introduciendo una variable de holgura a cada ecuación, entonces:

$150 y_{t-1} = D_t + 100 x_t + u_t \quad (t = 1, 2, \dots, 6)$  (\*\*)

$y_0 = x_0$   
 $u_t \geq 0$

Observe que  $u_t$  es una variable de holgura que representa el número de horas disponible no usadas al final del periodo  $t$ . Expresando esta restricción para cada  $t$  se tiene que:

Demanda en el mes 1:  $150 y_0 = 8000 + 100 x_1 + u_1$

Demanda en el mes 2:  $150 y_1 = 9000 + 100 x_2 + u_2$

Demanda en el mes 3:  $150 y_2 = 8000 + 100 x_3 + u_3$

Demanda en el mes 4:  $150 y_3 = 10000 + 100 x_4 + u_4$

Demanda en el mes 5:  $150 y_4 = 9000 + 100 x_5 + u_5$

Demanda en el mes 6:  $150 y_5 = 12000 + 100 x_6 + u_6$

48

FUNCION OBJETIVO:

Ya que el objetivo de la compañía es determinar cuantas aeromozas contratar en los próximos meses, entonces la función objetivo es minimizar los costos involucrados. Estos costos son los costos de las aeromozas experimentadas más los costos de las aeromozas que están siendo entrenadas. Por lo tanto, la función objetivo está dada por

$\min Z = 8000 [x_0 + y_1 + \dots + y_6] + 4000 [x_1 + x_2 + \dots + x_6]$

ya que  $x_0 = y_0$

$\min Z = 8000 [y_0 + y_1 + \dots + y_6] + 4000 [x_1 + x_2 + \dots + x_6]$  (\*\*\*)

Por lo tanto nuestro modelo de programación lineal para el problema dado, está definido por (\*), (\*\*), y (\*\*\*) . La representación de este problema de programación lineal en forma particionada (ó tableau ó tablero) es la siguiente.

$y_0$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$x_0$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$u_1$	$u_2$	$u_3$	$u_4$	$u_5$	$u_6$	$Z$ (min)
8000	8000	8000	8000	8000	8000	8000	4000	4000	4000	4000	4000	4000	4000							60000
-1																				0
	1																			0
		1																		0
			1																	0
				1																0
					1															0
						1														0
150																				8000
	150																			9000
		150																		8000
			150																	10000
				150																9000
					150															12000

$x = [y_0, y_1, y_2, y_3, y_4, y_5, y_6, x_0, x_1, x_2, x_3, x_4, x_5, x_6, u_1, u_2, u_3, u_4, u_5, u_6] \geq 0$

$x_{A1}$	$x_{B1}$	$u_1$	$x_{A2}$	$x_{B2}$	$x_{C2}$	$u_2$	$x_{A3}$	$x_{B3}$	$u_3$	$x_{A4}$	$u_4$	$x_{D5}$	$u_5$	
.4	.7		.4	.7	1		.4	.7		.4		.3		= z (max)
1	1	1												= 10 000
	-1		1	1	1	1								= 0
-1.4						-1	.1	1	1					= 0
	-1.7		-1.4					-1	1	1				= 0
				-1.7			-1.4			-1	1	1		= 0

$$x = [x_{A1} \ x_{B1} \ u_1 \ x_{A2} \ x_{B2} \ x_{C2} \ u_2 \ x_{A3} \ x_{B3} \ u_3 \ x_{A4} \ u_4 \ x_{D5} \ u_5] \geq 0$$

NOTA: Las restricciones del (1) al (5) pueden expresarse sin variables de holgura, con objeto de expresar estas restricciones como desigualdades en lugar de igualdades. El procedimiento para obtener estas igualdades es el siguiente:

Obviamente de la ecuación (1) se tiene

$$x_{A1} + x_{B1} \leq 10\ 000 \quad (1^1)$$

Sumando (1) y (2):

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} + u_2 = 10\ 000$$

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} \leq 10\ 000 \quad (2^1)$$

Sumando (1), (2) y (3):

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} + x_{A3} + x_{B3} + u_3 = 10\ 000 + 1.4x_{A1}$$

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} + x_{A3} + x_{B3} \leq 10\ 000 + 1.4x_{A1} \quad (3^1)$$

Sumando (1), (2), (3) y (4):

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} + x_{A3} + x_{B3} + x_{A4} + u_4 = 10\ 000 + 1.4x_{A1} + 1.4x_{A2} + 1.7x_{B1}$$

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} + x_{A3} + x_{B3} + x_{A4} \leq 10\ 000 + 1.4x_{A1} + 1.4x_{A2} + 1.7x_{B1} + 1.4x_{A3} + 1.7x_{B2} \quad (4^1)$$

Sumando (1), (2), (3), (4) y (5):

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} + x_{A3} + x_{B3} + x_{A4} + x_{D5} + u_5 = 10\ 000 + 1.4x_{A1} + 1.4x_{A2}$$

$$+ 1.7x_{B1} + 1.4x_{A3} + 1.7x_{B2}$$

$$x_{A1} + x_{B1} + x_{A2} + x_{B2} + x_{C2} + x_{A3} + x_{B3} + x_{A4} + x_{D5} \leq 10\ 000 + 1.4x_{A1} + 1.4x_{A2} + 1.7x_{B1} + 1.4x_{A3} + 1.7x_{B2} \quad (5^1)$$

Por lo tanto, las desigualdades del (1<sup>1</sup>) al (5<sup>1</sup>) son las restricciones a nuestro problema estas restricciones pueden obtenerse directamente del contexto del problema sin la introducción de variables de holgura, nuestro problema expresado a través de las restricciones de la (1<sup>1</sup>) a la (5<sup>1</sup>), queda representado en forma particionada como sigue:

$$x_{A1} \ x_{B1} \ x_{A2} \ x_{B2} \ x_{C2} \ x_{A3} \ x_{B3} \ x_{A4} \ x_{D5}$$

.4	.7	.4	.7	1	.4	.7	.4	.3	
1	1								≤ 10 000
1	1	1	1	1					≤ 10 000
-.4	1	1	1	1	1	1			≤ 10 000
-.4	-.7	-.4	1	1	1	1	1		≤ 10 000
-.4	-.7	-.4	-.7	1	-.4	1	1	1	≤ 10 000

$$x = [x_{A1} \ x_{B1} \ x_{A2} \ x_{B2} \ x_{C2} \ x_{A3} \ x_{B3} \ x_{A4} \ x_{D5}] \geq 0$$



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

**INTRODUCCION A LA PROGRAMACION Y  
COMPUTACION ELECTRONICA**

**PROBLEMA DE INVENTARIOS, APLICACIONES**

**OCTUBRE, 1982**

# INSYS

**an inventory system model**  
 computes inventory levels and factory output for a factory-wholesaler-retailer inventory system. Inventory replenishment and lead time policies may be changed in order to test the effect of these policies on the performance of the overall system.

The purpose of an inventory is to provide a separation in time or location between the production of goods and the consumption of goods. In our specialized economy a man is no longer his own butcher, baker, and candlestick-maker. Rather, we have production centers (factories) which are specialized, centrally located, and have high production rates. There is a great gain in production efficiency from this specialization, but it also requires a large increase in inventories to separate the centralized factory from the ultimate consumer. No longer do we follow the example of the Norse red hen who planted, trapped, milled, baked, and ate (without the help of the pig, cow, rabbit, or duck) her own loaf of bread.

The most common inventory system in our economy is the factory-wholesaler-retailer system. The wholesaler provides a time decoupling service between the factory and the retailer, in that he holds the factory output until ordered by the retailer. The wholesaler also provides a location decoupling, in that he generally ships goods over a wide geographic area. Similarly, the retailer provides a decoupling service between the wholesaler and the consumer, in that he maintains an inventory of goods on display for sale to customers.

The purpose of this exercise is to provide an illustration of the dynamic nature of the factory-wholesaler-retailer inventory system. A computer model is used to calculate week by week how the retail inventory, the wholesale inventory, and the factory output rate change in response to retail sales. The model user may make changes in retail and wholesale inventory policy in an attempt to control the overall inventory system.

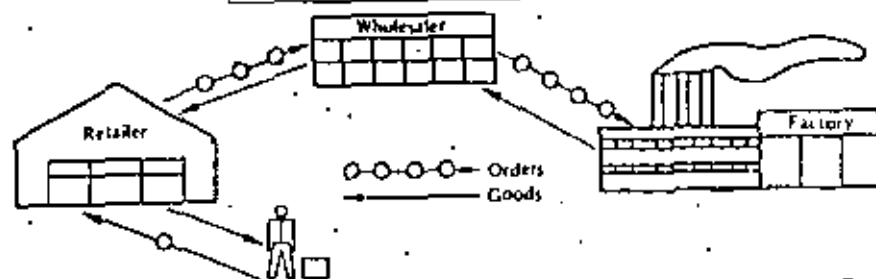


Figure 2-1 The factory-wholesaler-retailer system.

Section 2.1 explains the normal inventory systems and the rules for maintaining inventory levels. The following sections present three illustrated computer problems for maintaining and controlling the inventory system.

## 2.1 FACTORY-WHOLESALE-RETAILER MODEL

The normal system for the production and distribution of goods in our economy is through the factory-wholesaler-retailer system. A visual conceptualization of this system is shown in Figure 2-1. The function of the retailer in this system is to

- take orders from customers
- deliver goods to customers from on-the-shelf inventory
- reorder goods from the wholesaler
- receive shipments from the wholesaler

The function of the wholesaler is similar to the retailer except that the wholesaler's customer is the retailer and there is a time lag between the ordering and the delivery of goods. The wholesaler must

- receive orders from the retailer
- ship goods from the warehouse inventory
- reorder goods from the factory
- receive shipments from the factory.

Finally, the factory must produce the goods which are ultimately sold to the customers. The factory may or may not hold inventories. In the current model the factory does not maintain any inventory so that its only functions are to

- produce goods at some rate
- change production rate as requested by wholesaler

The model just described is a simple abstraction of that which is found in the industrial system. Durable goods, such as appliances, more or less follow the system described. There are variations so that some large retailers order directly from the factory, or the factory may maintain a

showroom and make direct retail sales. In other cases, the factory maintains large inventories and performs the function of the wholesaler. In all of these modifications, however, there is a dynamic interplay of sales with the inventories maintained and the factory rate as illustrated in this model.

#### Retailer Model Formulas

The parameters and formulas for the actual computer model of the retailer are presented in this section. These formulas are a mathematical statement of the verbal model description above. We also present some simple computations using the retail formulas.

*Retail sales* are controlled by the customer. They are part of the input to the program by the reader. Retail sales in the past have been about 100 units per week.

*Retail receipts* are the units received from the wholesaler each Monday morning that were ordered Friday one week (10 days) prior.

*Retail inventory level* is the number of units on hand Friday afternoon at the close of business. The inventory level varies through the week as shown in Figure 2-2. The formula for determining the inventory level is

$$\text{Inventory level} = \text{prior inventory level} + \text{retail receipts} - \text{retail sales}$$

*Retail orders* are placed with the wholesaler each Friday afternoon after determining the inventory level. The order policy is to order the retail sales for the week plus or minus enough units to return the base stock level to 100 units. Thus

$$\text{Retail order} = \text{retail sales} + (100 - \text{inventory level})$$

The effects of these formulas on inventory level and the retail order can be seen in the following sample computation.

In a normal week

$$\begin{aligned} \text{Retail sales} &= 100 \\ \text{Retail receipts} &= 100 \\ \text{Retail inventory level} &= 100 + 100 - 100 \\ &= 100 \\ \text{Retail order} &= 100 + (100 - 100) \\ &= 100 \end{aligned}$$

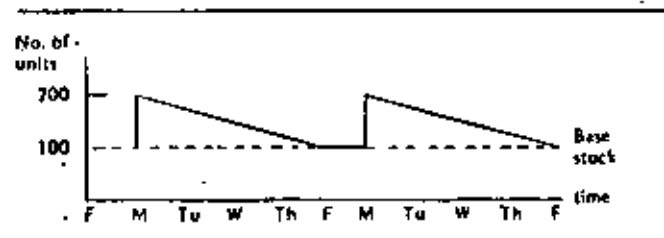


Figure 2-2 Retail inventory level.

In a week in which sales increase

$$\begin{aligned} \text{Retail sales} &= 110 \\ \text{Retail receipts} &= 100 \\ \text{Retail inventory level} &= 100 + 100 - 110 \\ &= 90 \\ \text{Retail order} &= 110 + (100 - 90) \\ &= 120 \end{aligned}$$

In a week in which sales decrease

$$\begin{aligned} \text{Retail sales} &= 90 \\ \text{Retail receipts} &= 100 \\ \text{Retail inventory level} &= 100 + 100 - 90 \\ &= 110 \\ \text{Retail order} &= 90 + (100 - 110) \\ &= 80 \end{aligned}$$

#### Wholesaler Model Formulas

The wholesaler's policies for maintaining inventory and reordering from the factory are similar to the retailer's policies. The formulas for the wholesaler and sample computations are now presented.

*Wholesale shipments* are dispatched each Wednesday from orders submitted by the retailer on the prior Friday. These orders arrive at the retailer's on the following Monday.

$$\text{Wholesale shipments} = \text{retail order (prior week)}$$

*Wholesale receipts* is the factory production of the previous week which is received each Monday morning.

$$\text{Wholesale receipts} = \text{factory production (prior week)}$$

*Wholesale inventory level* is the number of units on hand Friday afternoon at the close of business. The inventory level actually varies through the week as shown in Figure 2-3.

The formula for determining the Friday afternoon inventory level is as follows:

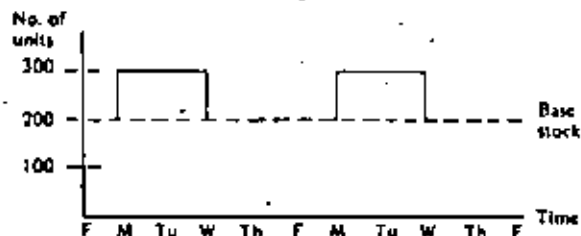


Figure 2-3 Wholesale inventory.

$$\text{Wholesale inventory level} = \text{prior inventory level} + \text{wholesale receipts} - \text{wholesale shipments}$$

Wholesale orders are placed with the factory each Friday afternoon after taking inventory. The factory, however, requires a week to change the production rate, so two weeks pass before the wholesaler receives the actual order. The policy is to order the current week's shipments plus enough units to return the base stock to a normal level of 200 units. The formula is

$$\text{Wholesale order} = \text{wholesale shipments} + (200 - \text{inventory level})$$

The effects of the wholesaler's policies can be seen in the following sample computation.

In a normal week:

$$\begin{aligned} \text{Wholesale shipments} &= 100 \\ \text{Wholesale receipts} &= 100 \\ \text{Wholesale inventory level} &= 200 + 100 - 100 \\ &= 200 \\ \text{Wholesale order} &= 100 + (200 - 200) \\ &= 100 \end{aligned}$$

In a week in which shipments increase

$$\begin{aligned} \text{Wholesale shipments} &= 110 \\ \text{Wholesale receipts} &= 100 \\ \text{Wholesale inventory level} &= 200 + 100 - 110 \\ &= 190 \\ \text{Wholesale order} &= 110 + (200 - 190) \\ &= 170 \end{aligned}$$

In a week in which shipments decrease

$$\begin{aligned} \text{Wholesale shipments} &= 90 \\ \text{Wholesale receipts} &= 100 \\ \text{Wholesale inventory level} &= 200 + 100 - 90 \\ &= 210 \\ \text{Wholesale order} &= 90 + (200 - 210) \\ &= 80 \end{aligned}$$

It should be noted that in the present simplified model, the wholesaler only services one retailer. This is an obvious oversimplification from the real world and allows the analysis to isolate the effect of a single retailer in the entire system.

Factory  
Production  
Rate

In this model, the factory maintains no inventory. The factory produces at the rate specified by the wholesale order. There is, however, a one-week delay when changing the production rate and a one-week delay for shipping. The net effect is that the wholesaler receives the actual order two weeks after it is placed with the factory. Thus, for example, one might have the situation shown in Figure 2-4.

Week number	Wholesale order	Factory rate	Wholesale receipts
1	100	100	100
2	120	100	100
3	80	100	100
4	100	120	100
5	100	80	120
6	100	100	80

Figure 2-4 Changing factory production rate.

## 2.1 NORMAL INVENTORY POLICY

This section presents the results of following a normal inventory policy. By "normal" we mean that the retailer and wholesaler follow the rules described in the preceding sections. The most significant rule, which will be analyzed in detail later, is the reorder rule. The normal reorder rule which is followed in the current problem is

*Order the current week's sales plus or minus enough to bring the base stock back to its normal level.*

Following this reorder rule and the other inventory policies outlined in Section 2.1, one can compute over a number of weeks the inventory level and orders in response to retail sales. For example, if retail sales are 100 in weeks 1 and 2, then increase to 110 in weeks 4, 5, 6, and 7, results will occur as shown in Figure 2-5.

These results are arrived at by following the computation formulas given in the preceding sections. For example, the formula for the retail order each week is as follows:

$$\text{Retail order} = \text{weekly sales} + (100 - \text{inventory level})$$

The retail order in week 5 is 120 units, derived from the above formula as follows:

$$\text{Retail order} = 110 + (100 - 90) = 120$$

Week No.	Retail				Wholesale			Factory Rate	
	Sales	Rec	Inv	Order	Ship	Rec	Inv		
1	100	100	100	100	100	100	200	100	100
2	100	100	100	100	100	100	200	100	100
3	110	100	90	120	100	100	200	100	100
4	110	100	80	130	120	100	180	140	100
5	110	120	90	120	130	100	150	180	100
6	110	130	110	100	120	100	130	190	140
7	110	120	120	90	100	140	170	130	180

Figure 2-5 "Normal" inventory policy.







EXERCISE	END BY	ROY HARRIS
25	50	90
01	100	
02	100	
03	110	
04	110	
05	110	
06	110	
07	110	
08	110	
09	110	
10	110	
11	110	
12	110	
13	110	
14	110	
15	110	
16	110	
17	110	
18	110	
19	110	
20	110	
21	110	
22	110	
23	110	
24	110	
25	110	

Figure 2-8 Computer input—replenishment rate.

PROGRAM INPUTS FOR EXERCISE TWO BY ROY HARRIS

WEEK	SALES	REC	INV	ORDER	SHIP	REC	INV	ORDER	RATE
1	100	100	100	100	100	100	200	100	100
2	100	100	100	100	100	100	200	100	100
3	110	100	90	110	100	100	200	100	100
4	110	100	80	120	110	100	185	120	100
5	110	110	75	110	120	100	165	130	100
6	110	120	70	110	110	100	145	140	100
7	110	110	60	100	110	120	150	130	100
8	110	110	50	100	100	110	160	110	100
9	110	100	40	100	100	110	170	90	100
10	110	100	30	100	100	110	180	80	100
11	110	100	20	100	100	110	190	70	100
12	110	100	10	110	110	90	200	60	100
13	110	110	0	110	110	80	210	50	100
14	110	110	0	110	110	80	210	40	100
15	110	110	0	110	110	90	165	20	100
16	110	110	100	110	110	100	150	10	100
17	110	110	100	110	110	110	160	10	100
18	110	110	100	110	110	110	170	10	100
19	110	110	100	110	110	110	180	10	100
20	110	110	100	110	110	110	190	10	100
21	110	110	110	110	110	110	200	10	100
22	110	110	110	110	110	110	210	10	100
23	110	110	100	110	110	90	210	10	100
24	110	110	100	110	110	90	200	10	100
25	110	110	100	110	110	90	190	10	100

25 WEEKS RUN 50 50 0 0

Figure 2-9 Computer output—replenishment rate.

However, all is still not perfect. There is still a long time lag before the factory "catches on" to the new rate. Moreover, a simple 10 percent increase in sales still causes a 70 percent change in the wholesale shipments and a 44 percent change in the factory rate. Section 2.4 considers additional control measures for bringing the inventory system under even tighter control.

## 2.4 CONTROL OF LEAD TIME

This section considers the problem of controlling fluctuations in the inventory system through a decrease in the lead time between the order and the receipt of goods. The basic concept is to change the lead time required to respond to changes in the system. This concept is implemented by testing the effect of faster delivery from the wholesaler and faster changeover to a new production rate by the factory.

**Lead Time Concept** Under the "normal" system setup the two basic lead times in the system were (1) between the order and receipt of goods by the retailer, and (2) between the order and receipt of goods from the factory. These were as follows:

*Retailer normal lead time*

Order on                      Delivered on  
Friday week 1                Monday week 2

*Wholesaler normal lead time*

Order on                      Change rate                Deliver goods  
Friday week 1                Week 3                      Monday week 4

The effects of these lead times are clearly seen in Section 2.2 when the retailer reorders every Friday to make up goods that have previously been ordered. In effect, he makes a double reorder for the same goods. In addition, the factory takes seven weeks to begin to respond to a change in retail sales.

In the current example we will consider the effects of decreasing this lead time. The new policy is to work the wholesaler on Saturday in order to deliver Friday-afternoon orders the very next Monday. Thus,

*Retailer decreased lead time*

Order on                      Delivered on  
Friday week 1                Monday week 2

Similarly the lead time for the wholesaler may be changed if the factory can shift to a new production rate without a week lag and if the factory ships over the weekend.

*Wholesaler decreased lead time*

Order on                      Change rate                Delivered in  
Friday week 1                week 2                      week 3

Computer  
Input  
-Lead  
Times

The *user name card* is the same as that in Section 2.2. The change in the lead times is implemented in the computer model by two fields in the *control card*. A speedup of one week in the wholesaler deliveries is accomplished by placing a 1 in column 31 of the control card. A speedup of one week in the changeover of the factory is accomplished by placing a 1 in column 41 of the control card.

The control card now reads

```

25      50      30      1      1
-----
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  
```

The weekly sales cards retain the same format.

The complete data deck setup to test the effects of the decreased lead time is shown in Figure 2-10.

Computer  
Output

The computer printout with lead time decreased is shown in Figure 2-11.

Analysis of  
Lead  
Time  
Control

The result of the new lead time policy is a further improvement in the overall performance of the inventory system.

Retail orders match the new sales rate within five weeks.

WEEK	SALES	WHOLESALE	FACTORY
25	90	90	1
26	100	100	1
27	100	100	1
28	110	110	1
29	110	110	1
30	110	110	1
31	110	110	1
32	110	110	1
33	110	110	1
34	110	110	1
35	110	110	1
36	110	110	1
37	110	110	1
38	110	110	1
39	110	110	1
40	110	110	1
41	110	110	1
42	110	110	1
43	110	110	1
44	110	110	1
45	110	110	1
46	110	110	1
47	110	110	1
48	110	110	1
49	110	110	1
50	110	110	1

Figure 2-10 Computer input - lead times.

## PROGRAM INVSYS FOR EXERCISE THREE BY RAY HARRIS

WEEK	DETAIL				WHOLESALE				FACTORY
NO.	SALES	WHOLE	WHOLE	ORDER	SHIP	WHOLE	WHOLE	ORDER	DATE
1	100	100	100	100	100	100	100	100	100
2	100	100	100	100	100	100	100	100	100
3	110	100	90	115	115	100	105	120	100
4	110	115	95	123	123	100	113	126	123
5	110	123	98	131	131	123	104	129	126
6	110	131	99	137	137	123	106	131	129
7	110	137	99	140	140	123	106	131	131
8	110	140	100	140	140	131	109	130	130
9	110	140	100	140	140	131	109	130	130
10	110	140	100	140	140	131	109	130	130
11	110	140	100	140	140	131	109	130	130
12	110	140	100	140	140	131	109	130	130
13	110	140	100	140	140	131	109	130	130
14	110	140	100	140	140	131	109	130	130
15	110	140	100	140	140	131	109	130	130
16	110	140	100	140	140	131	109	130	130
17	110	140	100	140	140	131	109	130	130
18	110	140	100	140	140	131	109	130	130
19	110	140	100	140	140	131	109	130	130
20	110	140	100	140	140	131	109	130	130
21	110	140	100	140	140	131	109	130	130
22	110	140	100	140	140	131	109	130	130
23	110	140	100	140	140	131	109	130	130
24	110	140	100	140	140	131	109	130	130
25	110	140	100	140	140	131	109	130	130

25 WEEKS RUN 50 50 1 1

Figure 2-11 Computer output - lead times.

Wholesaler orders match the new sales rate within eight weeks.  
Factory rate is set to the new sales level in nine weeks.

In addition to cutting response lags down, there is less fluctuation in the inventory levels.

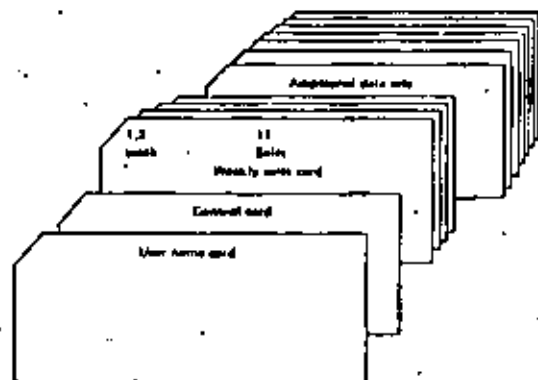
A simple increase in sales of 10 percent causes a 15 percent change in wholesale shipments, down from the prior 20 percent change. Also, the factory rate changes 20 percent, down from the prior 44 percent. Thus, in general, it may be said that the inventory system is now in better control.

There are, however, many complications one could add to the model before it would approximate the real world. For example, customers are never so kind as to provide such nice uniform retail sales. Hence, the user might want to try his hand at controlling the inventory system if retail sales were to randomly fluctuate between, say, 80 units and 120 units in any given week.

- REFERENCES
- Ansoff, H. I., and D. Stevin, "An Approximation of Industrial Dynamics," *Management Science*, vol. 14, no. 7 (March 1968), pp. 398-415.
- Bullis, E. S., *Modern Production Management*, New York: Wiley, 1969.
- , *Models for Production and Operations Management*, New York: Wiley, 1963.

- Foster, R. B., et al, *Decision Models for Inventory Management*, Homewood, Ill.; Irwin, 1961.
- Forrester, J. W., *Industrial Dynamics*, Cambridge, Mass.: M.I.T., 1961.
- , "Industrial Dynamics After the First Decade," *Management Science*, vol. 14, no. 2 (March 1968), pp. 398-415.
- Garett, J. W., *Production and Operations Management*, New York: Macaulay, Brace & World, 1968.
- Meier, R. C., W. T. Newell, and H. L. Pazer, *Simulation in Business and Economics*, Englewood Cliffs, N.J.: Prentice-Hall, 1969.
- Olson, R. A., *Manufacturing Management: A Quantitative Approach*, Swanton, Pa.: International Textbook Co., 1968.
- Plouff, G. W., et al, *Production and Inventory Control*, Englewood Cliffs, N.J.: Prentice-Hall, 1967.

## 2.5 INSYS DATA DECK STRUCTURE



Control Card	Card columns	Format	Item
	1, 2	I2	No. of weeks
	11, 12	F2	Percentage value for retailer
	21, 22	F2	Percentage value for wholesaler
	31	F1	Wholesaler lead time
	41	F1	Factory lead time

40  
COMPUTER ARCHETS

## 2.6 INSYS PROGRAM LISTING

```

C      PROGRAM INSYS                1000  1
C      INVENTORY SYSTEM CONTROL MODEL  1000  2
C      EQUIPMENT 1967 BY ROY D HARRIS  1000  3
C      THIS VERSION FOR IBM 360       1000  4
C      DIMENSION ALPHA(10)           1000  5
C                                     1000  6
C      READ AND PRINT STUDENTS NAME CARD  1000  7
C      M1 = 5                          1000  8
C      M2 = 6                          1000  9
C      READ (M1,20) ALPHA              1000 10
C      WHILE (M2,21) ALPHA             1000 11
C                                     1000 12
C      READ CONTROL CARD AND INITIALIZE  1000 13
C      READ (M1,20) M1, M2, M3, M4, M5  1000 14
C      IF (M1) 2,2,3                   1000 15
C      2  A = 1.0                        1000 16
C      GO TO 4                          1000 17
C      3  A = 10/100.                   1000 18
C      IF (M2) 5,5,6                   1000 19
C      4  W = 1.0                        1000 20
C      GO TO 7                          1000 21
C      5  W = 10/100.                   1000 22
C      6  M1 = 100.                     1000 23
C      7  M2 = 100.                     1000 24
C      M3 = 100.                       1000 25
C      M4 = 100.                       1000 26
C      M5 = 200.                       1000 27
C      M6 = 100.                       1000 28
C      M7 = 100.                       1000 29
C      M8 = 100.                       1000 30
C      PRINT HEADING FOR WEEKLY OUTPUT  1000 31
C      WHILE (M2,22)                   1000 32
C      WHILE (M3,23)                   1000 33
C      34
C      35
C      36
C      37
C      38
C      39
C      40
C      41
C      42
C      43
C      44
C      45
C      46
C      47
C      48
C      49
C      50
C      51
C      52
C      53
C      54
C      55
C      56
C      57
C      58
C      59
C      60
C      61
C      62
C      63
C      64
C      65
C      66
C      67
C      68
C      69
C      70
C      71
C      72
C      73
C      74
C      75
C      76
C      77
C      78
C      79
C      80
C      81
C      82
C      83
C      84
C      85
C      86
C      87
C      88
C      89
C      90
C      91
C      92
C      93
C      94
C      95
C      96
C      97
C      98
C      99
C      100

```



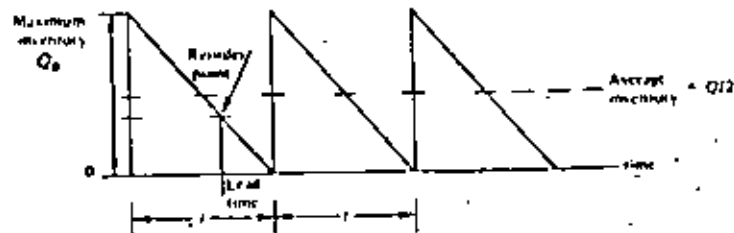


Figure 3-1 Inventory level and usage pattern for EOQ model.

The Economic Order Quantity Inventory Model

A basic assumption of the EOQ model is that the consumption of the inventory is constant over time and that it is possible to replenish the inventory on very short notice. The quantity in inventory at any point in time, for this circumstance, is shown in Figure 3-1. The basic EOQ model also assumes no quantity price discounts and no backorders.

In this "sawtooth" usage pattern the inventory is consumed over time until it is depleted. It is then instantly replenished (straight vertical line), and the usage continued.

The rational basis for determining inventory levels is to balance the cost of holding inventory against the cost of not holding inventory. In the consumption situation described above, the types of cost for holding the inventory are fairly obvious. Storage must be provided for the inventory and the inventory must be financed.

The cost of not holding inventory may not be so obvious. Since it was assumed that the inventory was easily and instantaneously obtainable, then there is no cost attached to being caught short. However, like a housewife who goes to the grocery store three times each day to purchase a single meal, there is a cost attached to procurement of the inventory. Not holding inventory may lead to very high procurement cost.

The issue in the economic order quantity model is to determine how much inventory to order each time. The cost of procurement per unit goes down if more is ordered each time, but the cost of holding inventory goes up. Figure 3-2 shows a graphic representation of these costs as they vary with the size of the order. The total incremental cost of inventory is shown as the top curve on the graph in the figure.

$$\text{Total incremental cost} = \text{holding cost} + \text{ordering cost}$$

The best inventory policy is to order the amount of inventory each time which yields the minimum total cost. This "correct quantity" to order is called the economic order quantity (EOQ).

The following definitions and variables will be used in deriving a mathematical expression for the EOQ.

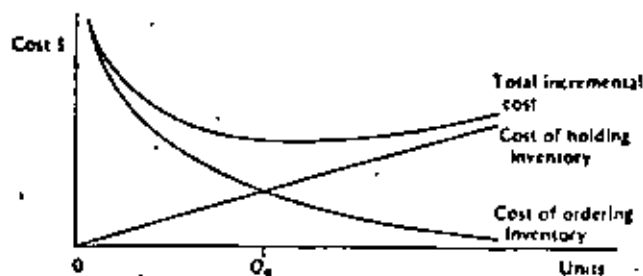


Figure 3-2 Costs of holding and ordering inventory.

$$\begin{aligned} \text{Holding cost} &= (\text{average inventory}) \times (\text{unit inventory holding cost per year}) \\ &= (Q/2) \times (P \times FH) \end{aligned}$$

where  $Q$  = quantity ordered

$P$  = price per unit

$FH$  = annual unit holding cost as percentage of the unit price

Order cost = (number of orders per year)  $\times$  (cost per order)

$$= \left(\frac{R}{Q}\right) \times (C_p)$$

where  $R$  = annual requirements in units, level demand

$C_p$  = procurement cost (includes costs of paperwork, handling, etc.)

$$\begin{aligned} \text{Cost of inventory} &= (\text{unit price}) \times (\text{annual requirements}) \\ &= (P) \times (R) \end{aligned}$$

Total cost = holding cost + order cost + cost of inventory

$$\text{Total cost} = \frac{Q \times P \times FH}{2} + \frac{R \times C_p}{Q} + P \times R$$

$$\text{Total incremental cost} = \frac{Q \times P \times FH}{2} + \frac{R \times C_p}{Q}$$

Solving for the economic order quantity  $Q_0$  by algebra: the minimum point on the total incremental cost curve is where the inventory holding cost and the procurement cost curves intersect. Where they intersect they must be equal. Therefore, at  $Q_0$ :

$$\frac{Q}{2} (P \times FH) = \left(\frac{R}{Q}\right) C_p$$

Clearing denominators:

$$Q(Q)(P \times FH) = 2(R)C_p$$

$$Q^2 = \frac{2RC_p}{P \times FH}$$

$$Q = \sqrt{\frac{2RC_p}{P \times FH}}$$







```

PROGRAM EOQ FDM HOGGARD (EOQ PROBLEM TWO: PRICE DISCOUNTS)
INPUT DATA IS *****
      B   LP   FM   P1   CS   B1   P2   H2   P3   V
      1000 5.00 .10 1.00   8   300   .90 2000   .00   8

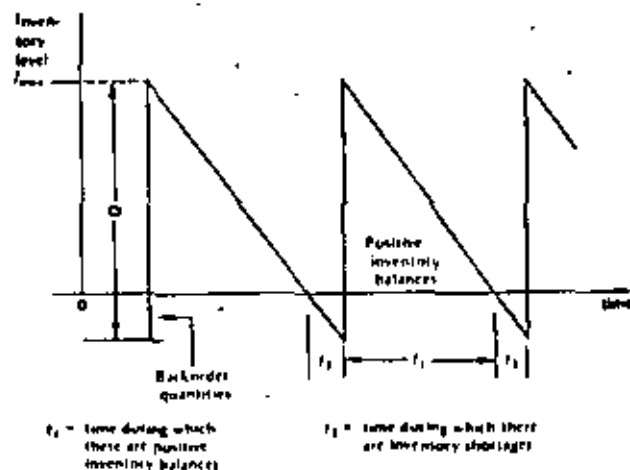
ANALYSIS RESULTS ARE ****
OPTIMUM ORDER QUANTITY IS                2400.00
AT A PRICE PER ITEM OF                    .80
YIELDING A TOTAL INVENTORY COST OF        1364.00
WHERE THE NUMBER OF ORDER CYCLES PER YEAR IS  .80
  
```

Figure 3-7. Computer output - Problem Two.

Computer Output - Problem Two. The computer printout resulting from the above data is shown in Figure 3-7.

### 3.3 SHORTAGE COSTS

Just as it is true that in the real world quantity price discounts exist, it is also true that backorders are a reality. By allowing backorders we are saying that if an order cannot be filled at this time due to stock shortages, then as soon as inventory is available previously unfilled orders, i.e., backorders, will be the first orders to be filled. However, in an inventory system allowing for backorders (see Figure 3-8) a shortage cost is usually input relating to the backorder quantities. Generally, this shortage cost consists of costs due to (1) possible lost sales due to stockouts, (2) decreased customer satisfaction, (3) additional costs associated with rush shipments, and so on.

Figure 3-8. An inventory system with  $(Q - I_{max})$  backorders allowed.

The basic EOQ formula may be modified to incorporate shortage costs as follows.

1.  $C_p$  = procurement cost per order (unchanged)
2.  $\frac{(P \times FH)(I_{max})}{2} t_2$  = the holding cost of the positive inventory balance during time  $t_2$  ( $t_2$  is on an annual basis, i.e.,  $t$  is a fraction of a year). Since  $t_2 = I_{max}/R$ , this becomes:  

$$\frac{(P \times FH)I_{max}^2}{2R}$$
3.  $C_s \frac{(Q - I_{max})^2}{2} t_2$  = the shortage cost of the backorders during time  $t_2$ . Since  $t_2 = \frac{Q - I_{max}}{R}$ , this becomes:  

$$C_s \frac{(Q - I_{max})^3}{2R}$$
 where  $I_{max}$  = maximum level of inventory and  
 $C_s$  = shortage cost.

Hence, the total incremental cost for one cycle,  $t_1 + t_2$ , of an inventory system which allows backorders is

$$C_p + (P \times FH) \frac{I_{max}^2}{2R} + C_s \frac{(Q - I_{max})^2}{2R}$$

The annual total incremental cost is now obtained by multiplying the above equation through by the number of orders placed per year,  $R/Q$ :

$$TIC = \frac{R \times C_p}{Q} + (P \times FH) \frac{I_{max}^2}{2Q} + C_s \frac{(Q - I_{max})^2}{2Q}$$

To determine optimal values for  $Q$  and  $I_{max}$ , take the partial derivatives of the above equations with respect to  $Q$  and  $I_{max}$ , equate to zero and obtain

$$Q = \sqrt{\frac{2RC_p}{P \times FH}} \times \sqrt{\frac{P \times FH + C_s}{C_s}}$$

$$I_{max} = \sqrt{\frac{2RC_p}{P \times FH}} \times \sqrt{\frac{C_s}{P \times FH + C_s}}$$

$$TIC = \sqrt{2(P \times FH)RC_p} \times \sqrt{\frac{C_s}{P \times FH + C_s}}$$

However, if either  $Q$  or  $I_{max}$  is constrained, their respective values are obtained as follows:

1. When  $Q$  is constrained, for example, fixed at price discount quantities, then  $I_{max}$  is calculated as

$$I_{max} = \frac{C_s Q}{C_H + C_s}$$

2. When  $f_{max}$  is constrained, for example, limited by storage constraints (see next section), then  $Q$  is calculated as

$$Q = \sqrt{\frac{2CPN + (P \times F)(f_{max} + C)f_{max}}{Cs}}$$

**Sample Problem Three** This problem is basically the same as Sample Problem One except that shortage costs ( $C_s$ ) are included. In this problem  $C_s = \$0.30$ .

$$Q = \sqrt{\frac{2(1600)(5.00)}{1.00(0.10)}} \times \sqrt{\frac{(100 \times 0.10) + 0.30}{0.30}} = 400(1.154) = 461.88 \text{ units}$$

$$TIC = \sqrt{2(5.00)(1.00 \times .10)1600} \times \sqrt{\frac{0.30}{1.00 \times 0.10 + 0.30}} = 40(8.66) = 34.64$$

$$TC = [1.00](1600) + 34.64 = \$1634.64$$

The reader should note that the effect of including shortage costs is to increase the size of  $Q$ . This is because the annual inventory holding costs are smaller due to the smaller average inventory. In addition, the total incremental costs ( $TIC$ ) are less than in the classical model because both holding costs and preparation costs are lower. (This may be verified by comparing the results of this analysis with those from Section 3.1.)

**Computer Input - Problem Three** The data card for this example problem follows the same general form outlined previously. One additional data input is necessary. The shortage cost of \$0.30 is punched in column 21-25.

```
1600 5.00 0.10 1.00 0.30
*****
*****
```

The complete computer input is shown in Figure 3-9.

```
PROGRAM EDD PROBLEM THREE, SHORTAGE COST
1600 5.00 0.10 1.00 0.30
```

Figure 3-9 Computer Input - Problem Three.

The computer printout, using this input, is shown in Figure 3-10.

```
PROGRAM EDD FOR HAGGARD EDD PROBLEM THREE, SHORTAGE COST
INPUT DATA IS *****
      R      CP      PM      PL      CS      H1      P2      W2      P3      W
      1600  5.00    .10    1.00    .30      0      0      0      0      0

ANALYSIS RESULTS ARE *****
OPTIMUM ORDER QUANTITY IS          461.88
WITH OPTIMUM INVENTORY OF          206.46
AT A PRICE PER ITEM OF              1.00
YIELDING A TOTAL INVENTORY COST OF  1634.64
WHERE THE NUMBER OF ORDER CYCLES PER YEAR IS  3.40
```

Figure 3-10 Computer Output - Problem Three.

3.4 STORAGE LIMITATIONS

In this section, an additional constraint of maximum storage limitations, either in terms of available warehouse space or available capital, will be placed upon the basic EOQ model. This is intended to be an illustrative example of an additional type of constraint which may be (and in the real world is) imposed upon the basic EOQ model. By incorporating this additional constraint the reader should get some additional insight into the problems that face management when determining what quantities to purchase from suppliers.

**Sample Problem Four** This problem is basically the same as Sample Problem One except that the additional constraint of maximum warehouse space available ( $W$ ) has been imposed.

In this problem,  $W = 100$  units. From Sample Problem One,  $Q = 400$  units, but since  $W < Q$ , the order quantity must be  $Q = W = 100$  units.

In this problem, the cost of the limited storage constraint is  $TC_{\text{short}} - TC_{\text{w}}$ .

$$TC_{\text{w}} = \frac{100(1.00)(0.10)}{2} + \frac{1600(5.00)}{100} + 1600(1.00)$$

$$= 5 + 80 + 1600 = \$1685.00$$

$$\text{Cost}_w = TC_0 - TC_{\text{w}} = \$1640.00 - \$1685.00 = \$45.00$$

**Computer Input - Problem Four** The data card of the above example problem follows the same form as outlined previously, with one addition. The maximum warehouse space available, expressed in units, is keypunched in columns 46-50. The data card looks like this:

```
1600 5.00 0.10 1.00 0. 0. 0. 0. 100.
*****
*****
```

The computer input for this example problem is shown in Figure 3-11.

```
PROGRAM EDD PROBLEM FOUR, STORAGE LIMITS
1600 5.00 0.10 1.00 0. 0. 0. 0. 100.
```

Figure 3-11 Computer Input - Problem Four

**Computer Output - Problem Four** The computer output will indicate whether or not the warehouse constraint has had an effect on the economic order quantity. If an economic order quantity has been determined which exceeds  $W$ , the output will indicate that this has happened.

Furthermore, on the output will be a statement to the effect that, if the warehouse restrictions are operative, the order quantity determined may not be optimal. Suggestions are made for a method to determine the optimal quantity if this restraint is present. The computer output is shown in Figure 3-12.

```

PROGRAM EDU FOR WAREHOUSE EDU PROBLEM FOUR, STORAGE LIMITS
INPUT DATA IS *****
R    CP    FH    P1    C1    B1    P2    B2    P3    W
1600 5.00  0.10 1.00  0.30 300  0.90 500  0.80 350

ANALYSIS RESULTS ARE ****
BEFORE THE WAREHOUSE STORAGE LIMITATION IS APPLIED
OPTIMUM ORDER QUANTITY IS          400.00
AT A PRICE PER ITEM OF              1.00
YIELDING A TOTAL INVENTORY COST OF  1640.00
WHERE THE NUMBER OF ORDER CYCLES PER YEAR IS    4.00

THE ORDER QUANTITY IS LIMITED BY THE WAREHOUSE SPACE
RESTRICTION AND IS NOT AT AN OPTIMUM. LOOSEN THE
RESTRICTION AND RUN AGAIN OBSERVING THE EFFECT.

ANALYSIS RESULTS ARE ****
AFTER THE WAREHOUSE STORAGE LIMITATION IS APPLIED
OPTIMUM ORDER QUANTITY IS          100.00
AT A PRICE PER ITEM OF              1.00
YIELDING A TOTAL INVENTORY COST OF  2045.00
WHERE THE NUMBER OF ORDER CYCLES PER YEAR IS    16.00
THIS ORDER QUANTITY IS AT THE MAXIMUM WAREHOUSE CAPACITY

```

Figure 3-12 Computer output—Problem Four.

To conclude our discussion of economic order quantity models and, in particular this computer model, we would point out:

1. that the model is, in its present form, limited to only two price breaks.
2. the inclusion of shortage costs certainly complicates the storage limitation problem. In this model, when backorders and storage limitations are included in the same problem, the assumption is made that the backorders are instantaneously filled and that the storage limitation  $W$  is a constraint upon  $f_{max}$  and not upon  $Q$ . The user must remember that if  $f_{max}$  is constrained by  $W$  then neither  $Q$  nor  $f_{max}$  will be optimal.
3. this model will solve problems including one or all of the constraints previously described in a single problem. To appreciate this fact the reader may wish to solve the following problem manually and then by the use of the herein described EOQ model.

```

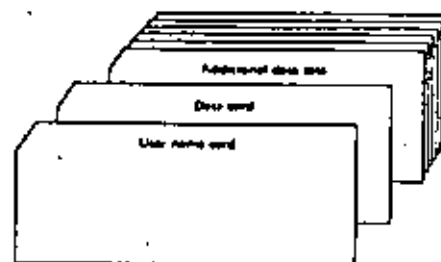
Data
R    Cp    FH    P1    C1    B1    P2    B2    P3    W
1600 5.00  0.10 1.00  0.30 300  0.90 500  0.80 350

```

- REFERENCES Bowman, E. H., and R. B. Fetter, *Analysis for Production and Operations Management*, 3d ed., Homewood, Ill.: Irwin, 1967.
- Bullis, E. S., *Modern Production Management*, 3d ed., New York: Wiley, 1969.
- , *Operations Management: Problems and Models*, 2d ed., New York: Wiley, 1968.
- , *Production-Inventory Systems: Planning and Control*, Homewood, Ill.: Irwin, 1968.

- Eilon, S., *Elements of Production Planning and Control*, New York: Macmillan, 1962.
- Fabrycky, W. J., and P. E. Torgersen, *Operations Economy: Industrial Applications of Operations Research*, Englewood Cliffs, N.J.: Prentice-Hall, 1966.
- Garrett, L. J., and M. Silver, *Production Management Analysis*, New York: Harcourt, Brace & World, 1966.
- Hadley, G., and T. M. Whitin, *Analysis of Inventory Systems*, Englewood Cliffs, N.J.: Prentice-Hall, 1963.
- Hopeman, R. J., *Systems Analysis and Operations Management*, Columbus, Ohio: Merrill, 1969.
- Magee, J. F., and D. M. Goodman, *Production Planning and Inventory Control*, 2d ed., New York: McGraw-Hill, 1967.
- Naddor, E., *Inventory Systems*, New York: Wiley, 1966.
- Olson, R. A., *Manufacturing Management: A Quantitative Approach*, Scranton, Pa.: International Textbook Company, 1968.
- Plouf, G. W., and O. W. Wright, *Production and Inventory Control*, Englewood Cliffs, N.J.: Prentice-Hall, 1967.
- Riggs, J. L., *Production Systems: Planning, Analysis and Control*, New York: Wiley, 1970.

## 3.5 EOQ DATA DECK STRUCTURE



Date Card	Card column	Format	Item
	1-3	F5.0	annual usage requirement
	6-10	F5.0	ordering cost
	11-13	F5.0	holding cost
	16-20	F5.0	unit price
	21-25	F5.0	shortage cost
	26-30	F5.0	minimum order quantity—first price discount
	31-35	F5.0	unit price—first price discount
	36-40	F5.0	minimum order quantity—second price discount
	41-45	F5.0	unit price—second price discount
	46-50	F5.0	maximum warehouse space available







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

**INTRODUCCION A LA PROGRAMACION Y  
COMPUTACION ELECTRONICA**

**PROBLEMA DE COLAS**

**OCTUBRE, 1982**

# QUESIM

a model for queueing systems

is a simulation model for single phase, multiple channel queueing systems. The performance of one to nine channels may be investigated under a variety of arrival and service parameters and for a variety of associated costs.

Queues (waiting lines) are common, everyday occurrences. Queues occur in grocery stores, in banks, in front of movie theaters, during university registration, and so on. Queues build up as a result of an interaction between customers arriving for service and a service facility. Almost everyone has experienced being frustrated by waiting in a line. The purpose of this computer exercise is to allow the user to experiment with situations in which queues occur. The model is concerned with more than waiting time in the queues, however. It allows one to look at the entire system from the operation manager's viewpoint, i.e., not only worrying about the customer but also being concerned about the utilization of facilities and the total cost of operation.

The first section provides background information on queueing concepts and how these concepts may be used by the operations manager as an aid for decision making.

The following sections present a situation in which the computer model may be used as a tool for analysis of some possible alternative management decisions. These illustrative problems include complete instructions on how to use the computer program QUESIM.

## 9.1 THE QUEUEING PROCESS

This section contains an introduction to some queueing concepts. No mathematical formulas are presented, although references are given for those readers interested in a detailed presentation of queueing theory.

### Definition of Queueing Terms

The queueing process is centered around a service system which has one or more service channels. Customers (arrivals) are drawn from an input source, or population. In queueing models, the customer arrivals from the input source are generally characterized by a probability distribution. The symbol commonly used to represent the mean arrival rate of customers is the Greek letter lambda ( $\lambda$ ). Any arrival entering the system joins a queue, or waiting line (a queue may be of zero length). The customer is selected from the queue for service according to a queue discipline or a priority rule. Usually, service times follow some probability distribution and the average service rate is commonly represented by the Greek letter mu ( $\mu$ ). In order to have a stable queueing process the average service rate ( $\mu$ ) must be greater than the mean arrival rate ( $\lambda$ ). After service is completed the customer exits the system. See Figure 9-1.

Generally, a queueing system is characterized by the following properties:

1. Its arrival pattern.
2. Its service time distribution.
3. Its queue discipline.
4. Its layout, or customer flow pattern.

### Arrival Patterns

The feature that makes some situations into queueing situations and other situations non-queueing is the nature of the arrivals to the system. Arrivals are the customers to the service facility, since they are the people or things that need to be processed.

In the situation where all arrivals are on hand, such as a large stock of raw materials, the service center may process arrivals at will, and there is no real queueing problem as such. In other situations where appointments are made ahead of time, there is also no real, or at least visible, queueing problem.

The really interesting problems, those worthy of being studied as queueing systems, exist when arrivals are not controlled or controllable by the service center. The most common assumption made in these cases is that the time intervals between consecutive arrivals are independent random variables. Each arrival is considered to be unaffected by the time at which

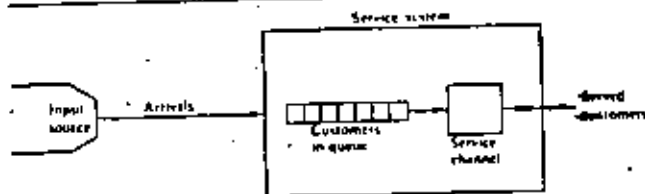


Figure 9-1 Single queue—single channel, single phase queueing process.

any other arrival occurs or by the number of arrivals which have already taken place. A good example is the placing of telephone calls when each customer does not really know, or care, who else is placing a call.

Studies of queuing systems have revealed that the time intervals between consecutive arrivals are often distributed according to the negative-exponential distribution. (Longer time intervals have a lower probability of occurrence.) In this case the number of arrivals expected forms a Poisson distribution. Other arrival distributions are possible, but the Poisson distribution is the most frequently occurring distribution.

**Service Time Distributions**

The simplest situation is that in which each arrival requires the same time for service as every other arrival. A vending machine, for example, is usually assumed to have a constant service time. The most commonly assumed service time density distribution, however, is the negative-exponential distribution. The service process may be further characterized as being single-phase (one operation) or multiple-phase (a series of operations).

**Queue Disciplines**

The queue discipline is the priority rule by which waiting jobs are selected from the queue for service. Because this represents a directly controllable decision variable, an extensive amount of research has been done in this area of queuing theory.

The most common priority rule is the first-come-first-served rule (FCFS). According to this rule, the first job to arrive in the queue will be the first job to be serviced. In addition, the following priority rules have received much attention: (1) the random rule selects the job which has the smallest value of a random priority assigned at the time of its arrival, and (2) the shortest operation time rule (SOT) selects from the queue the job which requires the least processing time at that service center.

**Layout or Customer-Flow Patterns**

The layout or flow pattern of a queuing system is largely determined by the specific servicing requirements of the arriving population, and by the physical limitations of the service facility.

This factor of specific servicing requirements is important when the job must be processed through a specific service channel or through a particular sequence of operations. If the job has no specific routing requirements, however, this factor becomes negligible. If the job requires several operations, for example, it is possible that the sequencing of these operations is of no consequence.

The physical limitations of the service facility are important, as they affect the facility layout. These physical limitations impose an additional constraint when they tend to limit waiting areas for jobs. As an example, one may prefer to have a single-queue, multiple-channel service facility, but adequate waiting space may not be available. The following arrangements are some examples of system geometry or job-flow patterns: single queue-single channel (Figure 9-1), single queue-multiple channels in parallel (Figure 9-2), multiple queue-multiple channels (Figure 9-3), service centers in tandem (Figure 9-4), and service centers in a network.

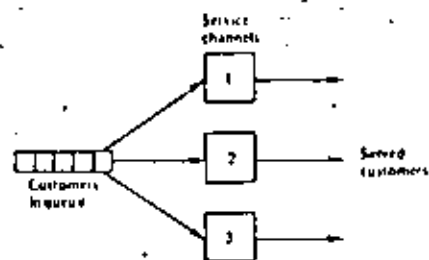


Figure 9-2 Single queue-multiple channel, single phase.

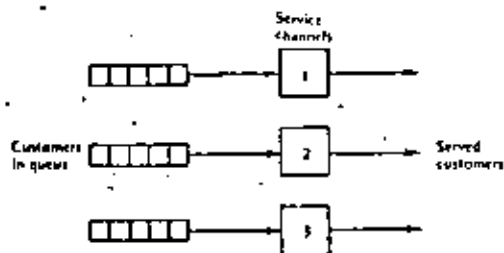


Figure 9-3 Multiple queue-multiple channel, single phase.

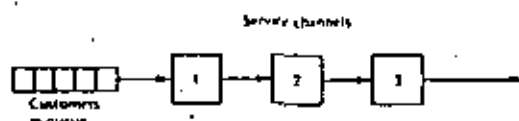


Figure 9-4 Single queue-single channel, multiple phase.

**Analysis for Management Decision**

One of the primary purposes for studying queuing theory is its predictive capabilities. In turn, this predictive capability is relevant to the design and control of operation systems. Some of the operational characteristics of a queuing system which may be of interest to a manager are the distributions of:

1. Queue length.
2. Customer waiting time (in queue and/or in the system).
3. Idle time of service facilities.
4. Number of customers in the service system.

These distributions may be described by their mean value, standard deviation, and the probability that the variable exceeds a specific value. With



information of this nature the queuing system could then be designed (or altered) so that

1. The resulting operational characteristics are within acceptable limits.
2. An economic criterion such as cost (or profit) may be minimized (or maximized).

An economic criterion may be established if one can associate dollar values with arrivals and service. For example, if one knew the revenues per arrival and the cost of service, it would be possible to set up a profit-maximization objective. Or, if the cost of an arrival waiting in line (or in the system) and the cost of service are known, the measure of effectiveness could be a cost-minimization function.

For solving operational problems which may be characterized as a queuing process there are two methods available. First, if the arrival and service time distributions are well-known mathematical distributions, it is possible to derive formulas for describing the operational characteristics. Secondly, even if the queuing process does not possess properties of well-known distributions, one can still attempt to solve the problem by means of Monte Carlo simulation. In this approach, empirical or assumed data for arrival and service time distributions are used as bases for generating a large number of arrivals and services, on paper. This may be done by hand, but for a large simulation it is most often done on a computer.

Since QUESIM is a computer simulation model for waiting lines, the development of the analytical formulas for solving these problems will not be presented here. For readers interested in the mathematical development of these formulas see Hillier and Lieberman, Morse, and Saaty, in the references for this exercise.

## 9.2 JOHN'S ICE CREAM SHOPPE

In this section, a sample problem suitable for the application of the computer model QUESIM will be presented. The QUESIM model itself will then be described. This description will point out to the user the types of systems for which QUESIM is applicable and the control options available to the user. Following the above, detailed descriptions of both the computer input and the computer output for the sample problem will be given.

### Problem Statement

John Entrepreneur, who is a senior at the local college of business administration, is going to open a campus ice cream shoppe. The store features seventy-eight varieties of ice cream (more than double the number of his competitor), and pricy cord's to serve them. Even though John's store will have more to offer than his competitor's store, called The Establishment, it is hypothesized that in the first few months local business will be roughly divided between both stores.

John also feels that the confection industry has a high index of substitutability and the improved availability of ice cream due to the opening of his shoppe in the near future will marginally increase the gross sales of both stores, rather than merely dividing the present sales market. John's marketing research efforts have turned up the following information:

the distribution of customer arrivals into his shoppe will probably follow a Poisson distribution, with a mean arrival rate of 60 people per hour. John has planned that upon arrival in his shoppe each customer will take a sequential number and await his turn for service. This will facilitate his servicing of customers on a first-come first-served basis.

John's preliminary analyses also show that one server (whose wages are \$1.80 per hour) can be expected to service about 30 customers per hour, following a negative-exponential distribution. Moreover, John's marketing research shows that when the service rate is slower than the arrival rate, customers will leave rather than wait in a long line. John estimates that his cost (possible opportunity cost) will be \$0.05 for every minute that a customer must wait.

Thus far, however, John has not been able to ascertain from his research data the optimum number of servers to have on duty. Logically, he knows that more than one server is required, because the mean arrival rate is twice the mean service rate. Furthermore, he realizes that these rates are mean rates of probability distributions, not constant rates.

John recognizes that his problem is one of design, i.e., how many service channels to provide in the above queuing system in order to minimize total costs. He could, in fact, actually operate his shoppe with one server, two servers, three servers, etc., each for a period of time, and calculate his total costs. However, John has an alternative, and that is to simulate his ice cream shoppe operations in a compressed time period with QUESIM. Using QUESIM, John could experiment with using one, two, three servers, and so on, until the model indicates to John the optimum number of service channels to have in his service facility. Moreover, the simulation can be done in a short period of time, as opposed to waiting for weeks of empirical data from actual operations.

### The Quesim Model

QUESIM is a computer model developed for simulating single-queue, single-phase, parallel queuing systems, that is, the type of system proposed by John Entrepreneur. The model determines, through an iterative process, the optimal number of service channels to allocate to a service facility. The configuration, or design, of the service facility is under the control of the user; hence, the user must specify the following characteristics of the queuing system to be studied:

1. The arrival distribution, the mean arrival rate, and the costs associated with arrivals waiting in line.
2. The service time distribution, the mean service time, and the costs of idle servers.
3. The simulation control limits, or the initial and maximum number of service channels (up to nine) which may be considered available to the service facility during a simulation run, and the maximum length of time for the simulation to run.

The above information on arrivals, services, and simulation control limits comprises the input for the computer program, QUESIM.

The computer output includes, at the top of the page, the user's



```

PROGRAM QUESIM FOR HADDARD QUESIM PROBLEM ONE
ARRIVAL TIME 1 RATE = 1.00 COST = .05
SERVICE TIME 2 TIME = 2.00 COST = .05
NO. CHANNELS START 1 MAX 4
MAX TIME 60
    
```

```

FIRST TWENTY OCCURRENCES FOR *****SERVICE CHANNELS
ARRIVAL -----DEPARTURE TIME AT CHANNEL NUMBER-----
TIME-- ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE
 0 2.4
 .8 3.3
 1.7 4.4
 3.1 5.1
 4.9 6.5
 6.8 8.4
 8.1 8.8
 9.4 9.4
11.2 12.7
11.4 12.8
14.6 15.2
15.1 17.4
17.8 18.2
17.4 21.8
19.3 24.4
19.5 26.0
20.1 26.8
20.2 27.3
20.8 28.3
21.4 28.4
    
```

```

AFTER 52 ARRIVED 36 SERVED 69 TIME UNITS
QUEUE-MAXIMUM LENGTH = 14
-MEAN LENGTH = 5.5
-MEAN WAIT TIME = 6.4
SERVICE UTILIZATION = 89.4 PERCENT
COSTS-WAIT IN QUEUE 331.1 UNITS AT $ .05 = $ 16.55
IDLE SERVICE 4.2 UNITS AT $ .01 = $ .42
TOTAL COST OF OPERATIONS $ 16.97
    
```

Figure 9-6 Computer output—John's Ice Cream Shoppe.

length of the queue, the mean number of customers in queue, and the mean waiting time in the queue are printed out. Then, the percent utilization of the service facilities is printed out. This is followed by the cost calculations. First, the total waiting time cost, which is the total waiting time of all customers in the system multiplied by the cost per unit waiting time, is given. The next line gives idle time cost, calculated as units of idle time (in the service facility) multiplied by the cost per unit of idle time.

The last line printed out is the total cost of operations, which is the idle time cost and the waiting time cost added together. It is this value that the simulator uses as a basis for comparing each iteration to the previous iteration for the purpose of determining when to terminate the simulation.

In the sample problem, the basic issue facing John Entrepreneur is the number of service channels (servers) he should have in his ice cream shoppe. The summary statistics for one channel (Figure 9-6) indicate that he

should at least have more than one server. The average length of the queue line is greater than two customers, and as fourteen at one time, so customers must wait on the average for about 6 time units (minutes). In real life, one would not expect the ice cream shoppe customers to be so patient! Moreover, as one would expect with so many customers waiting, the server is busy almost 90 percent of the time.

For the cost structure given in the problem, the waiting time cost for the customers far exceeds the idle time cost of the single server. The total cost of \$16.74 consists almost entirely of waiting time costs.

The simulation control card for this problem requested that program QUESIM initially simulate John Entrepreneur's system with one service channel and continue to simulate his system, adding one additional service channel each run, until the simulation terminated. The simulation will terminate either when the total cost of the system with  $M$  servers exceeds the total cost with  $M - 1$  servers or when the maximum number of allowable service channels has been reached.

For John's problem, the effect of adding additional servers in his shoppe is shown below. The cost figures in Figure 9-7 come directly from the computer printouts for the solution to John's problem. These figures like all simulation results, are a function of a random number generator which may be different for each computer. Hence, these figures may be slightly different on different computers.

Number of channels	Waiting cost (\$)	Idle time cost (\$)	Total cost of operations (\$)
1	16.55	0.19	16.74
2	7.14	0.49	7.63
3	0.26	2.61	2.87
4	0.96	3.71	4.67

Figure 9-7 The total cost of operations for one to four service channels.

Figure 9-7 shows that adding a second service channel dramatically reduces the cost of customer waiting from \$16.55 to \$7.14. Adding a third service channel (server) continues to decrease waiting time cost and is the number of servers which minimizes the total cost of operations. Thus the solution to John Entrepreneur's problem, as given by QUESIM, is for John to provide three servers in his shoppe operating under his expected steady state conditions. Keep in mind, however, that if we treat each simulation time unit as one minute, we have simulated only one hour of operation. Thus, John may want better data and/or a longer simulation run before coming to a definite conclusion about his problem.

### 9.3 JOHN'S ICE CREAM SHOPPE REVISITED

**John's New Data** Grand Opening!! John Entrepreneur's ice cream shoppe is now open. He is so excited about testing out his previous conclusions on the number of servers to have in his shoppe that after observing only the first fifteen arrivals



The complete input data deck, one card per typewritten line, now appears as shown in Figure 9 B.

Computer Output  
—John's Ice Cream  
Shoppes  
Revisited

Shown in Figure 9.9 is the computer output for the results of this problem for one channel. On the computer output, the identification information at the top of the page is changed to reflect the changes in the input data. The arrival type, service type, and simulation-run time are different from the previous situation. Furthermore, the simulation table is for only fifteen arrivals and services, as that is the total number of arrivals read in. Notice that on this output a warning message (\*\*WARNING\*\*OUT OF DATA BEFORE TIME 1.000\*\*) has been printed out. This message is printed out by QUESIM because the last arrival occurred at time 69, which was before the simulation time limit of 100 units. Moreover, the service facility was idle from time 77 to time 100; thus, the actual idle time in the service facility is overstated in the summary statistics. When this message appears, the user of QUESIM must exercise some care in the interpretation of the results or go back and rerun with a lower, more realistic, time limit. The summary statistics and cost information printed out below the simulation table is the same for all computer printouts.

#### REFERENCES

- Bhaskar, A., and A. Garg, "Basic Structure of Queuing Problems," *The Journal of Industrial Engineering*, vol. 14, no. 1 (January-February 1963), pp. 13-17.
- Billio, E. S., *Modern Production Management*, 3d ed., New York: Wiley, 1969.
- Operations Management: Problems and Models, 2d ed., New York: Wiley, 1969.
- Conrad, J. W., *Production and Operations Management*, New York: Macraut, Bruce & World, 1964.
- Hillier, F. S., and G. J. Lichtenman, *Introduction to Operations Research*, San Francisco, Calif.: Holden-Day, 1963.
- Lee, A. M., *Applied Queuing Theory*, London: Macmillan, 1966.
- McMillan, C., and R. T. Gonzalez, *Systems Analysis*, rev. ed., Homewood, Ill.: Irwin, 1965.
- Shaw, P. M., *Queues, Inventories and Maintenance*, New York: Wiley, 1958.
- Parsons, J. A., *Queueing Theory*, Englewood Cliffs, N.J.: Prentice-Hall, 1969.
- Richardson, S. B., *Operations Research for Management Decisions*, New York: Ronald, 1964.
- Sachs, I. L., *Elements of Queuing Theory with Applications*, New York: McGraw-Hill, 1963.

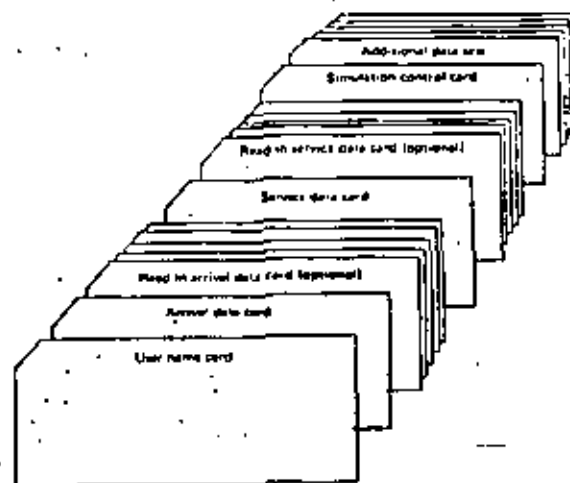
#### 9.4 QUESIM DATA DECK STRUCTURE

Arrival  
Card

1. Arrival data card must always be included.

Card columns	Format	Item
1	I1	code* for distribution
11-15	F5.0	mean arrival rate per unit time
21-25	F5.0	cost per unit of waiting time
31-33	F3.0	no. of arrivals to be read in

2. Read-in arrival data card is optional, depending upon the data. Its format is 12F5.0, so that arrival times are punched, 12 to a card, in columns 1-3, 6-10, 11-15, 16-20, etc.



3. Service data card must always be included.

Card columns	Format	Item
1	I1	code* for distribution
11-15	F5.0	mean service time
21-25	F5.0	cost per unit of idle time
31-33	F3.0	no. of service times to be read in

4. Read-in service data card is optional, depending upon the data. Its format is 12F5.0, so that service times are punched, 12 to a card, in columns 1-3, 6-10, 11-15, 16-20, etc.

5. Control card must always be included.

Card columns	Format	Item
1	I1	beginning no. of service channels
11	I1	maximum no. of service channels
21-25	F5.0	simulation run time

\*Code for distribution

- 1 the Poisson distribution
- 2 the negative exponential distribution
- 3 a constant rate
- 4 historical input data, read-in





```

M = 0
UPDATE HOURS SPENT IN QUEUE
CUMQUEIN(I) = CUMQUEIN(I) + INHARRV * TIME
UPDATE CLOCK TIME TO NEXT ARRIVAL
TIME = INHARRV
CHECK EACH CHANNEL: IF STATUS = 0 IT IS AVAILABLE
DO I = 1, N
IF ISTATUS(I) .GT. 0, GO TO 11
DO FIRST PART OF DEPART PROCESSING
STATUS(I) = 1
GO TO 12, I, N
POISSON SERVICE RATE
M = RAND(0)
I = INT(DEPT * ALG(0, I))
GO TO 7
NEGATIVE EXPONENTIAL SERVICE TIME
M = RAND(0)
I = ABS(LN(1 - RAND(0)))
GO TO 7
CONSTANT SERVICE TIME
I = DEPT
GO TO 7
HEAD-IN SERVICE TIME
IUSERV = RCUS * I
NUM = SNUM
IF IUSERV .LE. NUM GO TO 6
MFLAW = 0
I = 000000
GO TO 7
I = SMIUSERV
SET TIME OF NEXT DEPARTURE
INHARRV(I) = TIME + I
STORE FIRST TENTH DEPARTURE TIMES IN CN
ACUS = RCUS * I
IF IACUS .GT. 20 GO TO 8
II = 1
CNHARRV(I) = INHARRV(I)
CONTINUE
CHECK IF OUT OF SIMULATION TIME
IF STIME .LT. INHARRV(I) GO TO 9
ACCUMULATE PROCESSING TIME
CUMULI = CUMULI + I
CUSERV = CUSERV + I
GO TO 10
END OF SIMULATION UPDATING
LAST DEPARTURE FORCED OUT AT TIME
I = 1 - (CNHARRV(I) - TIME)
IF I .GT. 0 GO TO 11
TA = I
INHARRV(I) = TIME
CUMULI = CUMULI + TA
GO TO 10
IF I .GT. N GO TO 13
CONTINUE
ALL CHANNELS ARE BUSY, ADD ONE TO THE QUEUE
QUEUE = QUEUE + 1
SELECT ARRIVAL TIME (STORE FIRST TENTH IN CN)
GO TO (15, 10, 17, 18) * KA
POISSON ARRIVAL TIME DISTRIBUTION
M = RAND(0)
INHARRV = ABS(LN(1 - RAND(0)))
INHARRV = INHARRV * TIME

```

```

M 17
R 18
F 19
B 20
H 21
H 22
B 23
B 24
B 25
B 26
B 27
B 28
H 29
H 30
B 31
H 32
B 33
B 34
H 35
H 36
H 37
H 38
H 39
H 40
R 41
H 42
B 43
H 44
H 45
H 46
H 47
H 48
H 49
H 50
B 51
H 52
H 53
H 54
B 55
B 56
H 57
H 58
H 59
B 60
B 61
H 62
B 63
B 64
B 65
H 66
H 67
H 68
H 69
H 70
H 71
B 72
H 73
H 74
H 75
H 76
H 77
H 78

```

```

I 12 = 12 + 1
IF I .GT. 20 GO TO 28
CH(I) = I * INHARRV
GO TO 28
NEGATIVE EXPD ARRIVAL TIME DISTRIBUTION
R = RAND(0)
INHARRV = ABS(LN(1 - RAND(0)))
INHARRV = INHARRV * TIME
I 12 = 12 + 1
IF I .GT. 20 GO TO 28
CH(I) = I * INHARRV
GO TO 28
CONSTANT ARRIVAL TIMES
INHARRV = TIME + ARRTH
I 12 = 12 + 1
IF I .GT. 20 GO TO 28
CH(I) = I * INHARRV
GO TO 28
HEAD IN ARRIVAL TIMES
I 12 = 12 + 1
NUM = ANUM
IF I .LE. NUM GO TO 14
ARR(I) = 777777
MFLAW = 0
INHARRV = ARR(I)
IF I .GT. 20 GO TO 28
CH(I) = I * INHARRV
RETURN
END
SUBROUTINE DEPART
COMMON ALPHA(10), ANUM, AM(500), ARRTH, ARRTH, CH(10), I, CUMQUEI,
IUSERV, IZ, KA, RCUS, K5, N
Z = MLAG(QUEUE + SNUM, S(500), STATUS(I), I, TIME, TIME, INHARRV, INHARRV)
THIS SUBROUTINE PROCESSES THE DEPARTURE OF EVERY CUSTOMER
IT UPDATES THE HOURS SPENT IN THE QUEUE
IT UPDATES THE CLOCK TO THE NEXT DEPARTURE TIME (PREVIOUSLY
SELECTED)
IT CHECKS THE LENGTH OF THE QUEUE
IF NO ONE IN QUEUE IT SETS THE CHANNEL AT AN IDLE STATUS (THIS
DEPARTURE WAS PREVIOUSLY PARTIALLY PROCESSED EITHER BY
ARRIVE OR BY A PRIOR PASS THROUGH DEPART)
IF A QUEUE EXISTS THEN TAKE ONE FROM THE QUEUE, SET ITS DEPARTURE
TIME, SET THE CHANNEL AT A BUSY STATUS AND RETURN
M = QUEUE
CHECK LENGTH OF QUEUE; IF OVER 99 HOLD AT 99
IF I .LE. 99 GO TO 1
M = 0
UPDATE THE HOURS SPENT IN QUEUE
CUMQUEIN(I) = CUMQUEIN(I) - INHARRV(I) - TIME
UPDATE THE CLOCK TO NEXT DEPARTURE TIME
TIME = INHARRV(I)
IF I .GT. 0 GO TO 2
THIS SECTION COMPLETES THE PROCESSING OF A CUSTOMER
WHEN NO ONE IS WAITING IN THE QUEUE
STATUS(I) = 0
INHARRV(I) = 000000.0
RETURN
THIS SECTION DOES THE DEPART PROCESSING
WHEN THE CHANNEL HAS BEEN BUSY
QUEUE = QUEUE - 1

```

```

R 79
R 80
B 81
B 82
B 83
H 84
B 85
B 86
H 87
H 88
B 89
H 90
H 91
H 92
H 93
H 94
B 95
B 96
H 97
B 98
B 99
H 100
H 101
H 102
H 103
B 104
B 105
B 106
H 107
C 1
C 2
C 3
C 4
C 5
C 6
C 7
C 8
C 9
C 10
C 11
C 12
C 13
C 14
C 15
C 16
C 17
C 18
C 19
C 20
C 21
C 22
C 23
C 24
C 25
C 26
C 27
C 28
C 29
C 30
C 31
C 32

```



```

GO TO (3,4,5,6): NS
SELECT NEXT DEPARTURE TIME
POISSON SERVICE RATE
3 M = RAND(0)
T = ABS(DEPT*ALOG(R))
GO TO 4
4 NEGATIVE EXPONENTIAL SERVICE TIME
R = RAND(0)
T = ABS(DEPT*ALOG(R))
GO TO 5
5 CONSTANT SERVICE TIME
S = ULPTM
GO TO 6
6 MEAN-IN SERVICE TIME
IUSERV = ACUS+1
NUM = SMUK
IF (IUSERV .LE. NUM) GO TO 7
WFLAG = 70
T = 0.00000
GO TO 8
7 T = SM(IUSERV)
INPR(1) = TIME-T
STORE FIRST TWENTY DEPARTURE TIMES IN CN
ACUS = ACUS+1
IF (ACUS .GT. 20) GO TO 9
J3 = 3+1
CHICUS( J3 ) = INPR(1)
9 CONTINUE
CHECK IF OUT OF SIMULATION TIME
IF (ETIME .LT. INPR(1)) GO TO 14
ELSE SET STATUS BACK TO DUST AND RETURN
CUMUL = CUMUL+T
CUSERV = CUSERV+1.0
STATUS(1) = 1.0
RETURN
10 ADJUST T AND CUMUL AT TERMINATION OF SIMULATION
LAST CUSTOMER FORCED TO DEPART AT TIME
TK = 1-(INPR(1)-ETIME)
IF (TK .GT. 0.0) GO TO 11
TK = 0
11 INPR(1) = TIME
CUMUL = CUMUL+TK
STATUS(1) = 1.0
RETURN
END

FUNCTION RAND (N)
NACHAN( ULPONENT RANDOM NUMBER GENERATOR 10 TO 31)
THIS VERSION IS FOR 32 BIT WORD IBM 3601
N SET AT POSITIVE ODD INTEGER TO INITIALIZE
N SET AT ZERO TO CONTINUE STRING OF RANDOM NUMBERS
SEE NATIONAL COMPUTER SIMULATION TECHNIQUES FILE *SUMS-1360
IF (N) 2+1)
1 N = N
2 N = N*16807
IF (N) 3+1)
3 N = N*2347483647)
4 N = N
RAND = AN/2147483647.
RETURN
END
    
```

193  
QUESTIM

C	31
C	32
C	33
C	34
C	35
C	36
C	37
C	38
C	39
C	40
C	41
C	42
C	43
C	44
C	45
C	46
C	47
C	48
C	49
C	50
C	51
C	52
C	53
C	54
C	55
C	56
C	57
C	58
C	59
C	60
C	61
C	62
C	63
C	64
C	65
C	66
C	67
C	68
C	69
C	70
C	71
C	72
C	73
C	74
C	75
C	76
C	77
D	1
D	2
10M	3
D	4
D	5
D	6
D	7
D	8
10M	9
D	10
10M	11
D	12
10M	13
D	14
D	15
D	16

PROGRAMA QUEUES (COIAS)

```

YOUR NAME = 01 PREMIUM TRUCK
1 .00 .1667 2
2 15. 1000. .133
1 .00 .1667
YOUR NAME = 02 DELUX TRUCK
1 .00 .1667 2
2 12. 1400. .167
1 .00 .1667
YOUR NAME = 03 TWO PREMIUM TRUCKS
1 .00 .1667 2
2 15. 1000. .133
1 .00 .1667
YOUR NAME = 04 NEW PREMIUM TRUCK
1 .00 .1667 2
2 15. 800. .16
1 .00 .1667
YOUR NAME = 05 ONE DELUX TRUCK
1 .00 .1667 2
2 12. 1400. .167
1 .00 .1667
YOUR NAME = 06 TWO NEW PREMIUM TRUCKS
1 .00 .1667 2
2 15. 800. .13
1 .00 .1667
YOUR NAME = 07 PREMIUM / 4 PER HOUR
1 .00 .1667 2
2 15. 1000. .133
1 .00 .1667
YOUR NAME = 08 DELUX / 4 PER HOUR
1 .00 .1667 2
2 12. 1400. .167
1 .00 .1667
YOUR NAME = 09 TWO PREMIUM / 4 PER HOUR
1 .00 .1667 2
2 15. 1000. .133
1 .00 .1667
YOUR NAME = 10 NOT
1 .00 .1667 3
2 12. 1400. .167
1 .00 .1667
YOUR NAME = 11 NOT
1 .00 .1667 3
2 15. 1000. .133
1 .00 .1667
YOUR NAME = 12 NOT
1 .00 .1667 3
2 15. 1000. .133
1 .00 .1667
YOUR NAME = 13 RANDOM
1 .00 .1667 3
2 15. 1000. .133
1 .00 .1667
YOUR NAME = 14 RANDOM
1 .00 .1667 3
2 15. 1000. .133
1 .00 .1667
YOUR NAME = 15 RANDOM
1 .00 .1667 3
2 15. 1000. .133
1 .00 .1667
STOP
    
```

PROGRAM QUEUES FOR YOUR NAME & RI AEN MFFIUM TRACR  
 ARRIVAL TIME 1 UNIT & COST & SCHEDULE RULE 2  
 (PCFS)

24 25

SERVICE TYPE / TIME & VARIABLE COST \$ 10  
 FILED COST \$ 1000.00  
 NO. CHANNELS START / MAX 1 MAX TIME 4000

FIRST THIRTY BEGINNING FOR 1 SERVICE CHANNELS

ARRIVAL TIME	DEPARTURE TIME	TIME AT CHANNEL NUMBER
20.0	20.0	ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE
20.7	25.4	
20.7	25.4	
20.6	26.0	
170.2	173.0	
175.6	179.4	
140.0	270.7	
140.0	260.7	
140.0	250.0	
140.0	230.0	
200.1	247.0	
201.3	203.0	
203.0	205.0	
243.3	240.0	
244.4	241.1	
244.7	242.0	
244.7	245.0	
242.2	240.0	
310.0	330.0	
310.0	330.0	
370.1	421.0	

AFTER 270 ARRIVED 270 SERVED 6000 TIME UNITS  
 QUEUE-MAXIMUM LENGTH 7  
 -MEAN LENGTH 1.0  
 -MEAN WAIT TIME 21.0  
 SERVICE UTILIZATION 70.0 PERCENT  
 AVERAGE ARRIVAL PER TIME UNIT 1000  
 AVERAGE SERVICE TIME 10.0000  
 MEAN NO. IN THE SYSTEM 1.7300  
 MEAN TIME IN THE SYSTEM 17.0000

COST INFORMATION OF OPERATION  
 COSTS-ALL IN UNITS 4000.0 UNITS AT \$ 10 = \$ 40000.00  
 SERVICE COST VARIABLE 270.0 UNITS AT \$ 10 = \$ 2700.00  
 SERVICE COST FIXED 1000.00 WITH 1 CHANNELS = \$ 1000.00  
 TOTAL COST OF OPERATION \$ 42700.00

PROGRAM QUEUES FOR YOUR NAME & RI AEN MFFIUM TRACR  
 ARRIVAL TIME 1 UNIT & COST & SCHEDULE RULE 2  
 (PCFS)

26

SERVICE TYPE / TIME & VARIABLE COST \$ 10  
 FILED COST \$ 1000.00  
 NO. CHANNELS START / MAX 1 MAX TIME 4000

FIRST THIRTY BEGINNING FOR 1 SERVICE CHANNELS

ARRIVAL TIME	DEPARTURE TIME	TIME AT CHANNEL NUMBER
20.0	20.0	ONE TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE
20.8	25.4	
20.2	25.4	
20.6	26.0	
120.2	123.0	
125.6	129.4	
140.0	200.7	
140.0	200.7	
140.0	200.7	
140.0	200.7	
200.1	247.0	
201.3	203.0	
203.0	205.0	
243.3	240.0	
244.4	241.1	
244.7	242.0	
244.7	245.0	
242.2	240.0	
310.0	330.0	
310.0	330.0	
370.1	421.0	

25

AFTER 270 ARRIVED 270 SERVED 6000 TIME UNITS  
 QUEUE-MAXIMUM LENGTH 7  
 -MEAN LENGTH 1.0  
 -MEAN WAIT TIME 21.0  
 SERVICE UTILIZATION 70.0 PERCENT  
 AVERAGE ARRIVAL PER TIME UNIT 1000  
 AVERAGE SERVICE TIME 10.0000  
 MEAN NO. IN THE SYSTEM 1.7300  
 MEAN TIME IN THE SYSTEM 17.0000

COST INFORMATION OF OPERATION  
 COSTS-ALL IN UNITS 4000.0 UNITS AT \$ 10 = \$ 40000.00  
 SERVICE COST VARIABLE 270.0 UNITS AT \$ 10 = \$ 2700.00  
 SERVICE COST FIXED 1000.00 WITH 1 CHANNELS = \$ 1000.00  
 TOTAL COST OF OPERATION \$ 42700.00

PROBLEN QUEUES FOR YOUR NAME - 01 PREFIX 2 A NEW MOIM  
ARRIVAL TIME 1 RATE 0 FOR COST 0 17 SCHEDULE RULE 3  
(SOT)

SERVICE TYPE 2 RATE 0 17.0  
FIXED COST \$ 1000.00 VARIABLE COST \$ 0.13  
NO. CHANNELS NAMED 1 MAX 1 MAX TIME 6000

26

FIRST TWENTY OCCURRENCES FOR 1 SERVICE CHANNELS  
ARRIVAL DEPARTING TIME AT CHANNEL NUMBER  
TIME-- ON TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE

TIME	ON	TWO	THREE	FOUR	FIVE	SIX	SEVEN	EIGHT	NINE
0	2.0								
18.5	39.0								
19.6	34.1								
20.4	49.0								
48.1	101.0								
1-1.0	107.0								
117.9	176.0								
115.9	170.0								
130.5	200.0								
140.0	215.0								
145.9	231.0								
147.2	232.0								
142.4	240.0								
143.2	249.0								
144.9	250.0								
147.2	252.0								
219.8	290.0								
230.0	295.0								
233.1	290.0								
240.5	340.0								

AFTER 270 ARRIVED 370 SERVED 6000 TIME UNITS  
 QUEUE-MAXIMUM LENGTH 57  
 -MEAN LENGTH 5.5  
 -MEAN WAIT TIME 11.0  
 SERVICE UTILIZATION 79.0 PERCENT  
 AVERAGE ARRIVALS PER TIME UNIT 1.0667  
 AVERAGE SERVICE TIME 15.7217  
 MEAN NO. IN THE SYSTEM 8.342  
 MEAN TIME IN THE SYSTEM 97.4071

COST INFORMATION IN OPERATION  
 COSTS-WAIT IN QUEUE 3179.0 UNITS AT \$ .17 = \$ 520.83  
 SERVICE COST VARIABLE 370.0 UNITS AT \$ .13 = \$ 48.10  
 SERVICE COST FIXED 1000.00 WITH 1 CHANNELS = \$ 1000.00  
 TOTAL COST OF OPERATION \$ 2068.93

PROBLEN QUEUES FOR YOUR NAME - 01 SOT  
ARRIVAL TIME 1 RATE 0 FOR COST 0 17 SCHEDULE RULE 3  
(SOT)

SERVICE TYPE 2 RATE 0 17.0  
FIXED COST \$ 1000.00 VARIABLE COST \$ 0.17  
NO. CHANNELS NAMED 1 MAX 1 MAX TIME 6000

FIRST TWENTY OCCURRENCES FOR 1 SERVICE CHANNELS  
ARRIVAL DEPARTING TIME AT CHANNEL NUMBER  
TIME-- ON TWO THREE FOUR FIVE SIX SEVEN EIGHT NINE

TIME	ON	TWO	THREE	FOUR	FIVE	SIX	SEVEN	EIGHT	NINE
0	2.0								
26.0	43.0								
26.2	41.0								
48.0	50.0								
128.2	100.0								
135.6	105.0								
148.0	105.0								
148.0	192.0								
184.0	201.0								
240.1	203.0								
271.3	230.0								
273.0	237.0								
243.3	255.0								
240.4	250.0								
248.7	257.0								
240.7	259.0								
242.2	317.0								
310.0	300.0								
310.9	320.0								
370.1	412.0								

AFTER 270 ARRIVED 270 SERVED 6000 TIME UNITS  
 QUEUE-MAXIMUM LENGTH 0  
 -MEAN LENGTH 0  
 -MEAN WAIT TIME 0  
 SERVICE UTILIZATION 59.0 PERCENT  
 AVERAGE ARRIVALS PER TIME UNIT 1.0667  
 AVERAGE SERVICE TIME 15.7217  
 MEAN NO. IN THE SYSTEM 1.0667  
 MEAN TIME IN THE SYSTEM 23.0667

COST INFORMATION IN OPERATION  
 COSTS-WAIT IN QUEUE 2000.0 UNITS AT \$ .17 = \$ 340.00  
 SERVICE COST VARIABLE 270.0 UNITS AT \$ .17 = \$ 45.90  
 SERVICE COST FIXED 1000.00 WITH 1 CHANNELS = \$ 1000.00  
 TOTAL COST OF OPERATION \$ 1485.90

29

PROGRAM QUEUES FOR YOUR NAME & ID NUMBER  
 ARRIVAL TIME 1 UNIT = 15.00 COST = .17 SCHEDULE RULE 1  
 (RANDOM)  
 SERVICE TYPE 1 RATE = 15.00 28  
 FIXED COST \$ 1000.00 VARIABLE COST \$ .17  
 NO. CHANNELS 3 MEAN TIME 6000

FIRST TWENTY OCCURRENCES FOR 1 SERVICE CHANNELS  
 ARRIVAL TIME --- DEPARTURE TIME AT COUNTER NUMBER ---  
 TIME --- 1 2 3 4 5 6 7 8 9

0	2.0
26.0	45.0
26.2	45.0
40.0	56.0
128.2	153.0
135.0	199.0
148.0	200.0
148.0	220.0
184.0	247.0
200.0	247.0
221.0	263.0
223.0	270.0
243.0	281.0
244.0	313.0
246.0	314.0
249.0	317.0
292.0	318.0
306.0	329.0
310.0	329.0
374.0	421.0

AFTER 270 ARRIVAL 270 SERVED 6000 TIME UNITS  
 QUEUE-MAXIMUM LENGTH 7  
 MEAN LENGTH 1.7  
 MEAN WAIT TIME 22.3  
 SERVICE UTILIZATION 74.2 PERCENT  
 AVERAGE ARRIVAL PER TIME UNIT 1.400  
 AVERAGE SERVICE TIME 16.3209  
 MEAN NO. IN THE SYSTEM 1.7779  
 MEAN TIME IN THE SYSTEM 78.0376

COST INFORMATION OF OPERATION  
 COSTS-WAIT IN QUEUE 6150.0 UNITS AT \$ .17 = \$ 1026.00  
 SERVICE COST VARIABLE 270.0 UNITS AT \$ .17 = \$ 36.57  
 SERVICE COST FIXED 1000.00 WITH 3 CHANNELS = \$ 1000.00  
 TOTAL COST OF OPERATIONS \$ 2063.37

30

23

ANALYSIS OF RESULTS

1. The three alternatives available to Rio Valley Electric Co-op (herein referred to as Rio) are:

Alt.	Mean service rate	Variable cost/unit	Fixed cost/simulation run	Description
A-1	15	0.17	\$1000	one premium truck
A-2	12	.17	1400	one deluxe truck
A-3	15	.17	2000	two premium trucks

For all the alternatives the FCFS priority rule was used and the simulated runs were for 6000 minutes. The results for the runs are:

Alt.	Mean length Queue	Mean wait time	Service Util.	Cost wait	Var. cost	Fix cost	Total cost
A-1	1.0	21.5	74.2%	\$987.12	\$36.18	\$1000	\$2023.29
A-2	.5	11.5	59.4	\$29.42	45.42	1400	1975.22
A-3	.1	1.1	37.2	\$1.25	36.44	2000	2087.69

From the runs Rio should choose alternative A-2 as it has the lowest total cost. However, if the Rio Company is planning to grow it might prefer to choose alternative A-3 since this alternative has the lowest service utilization and thus would allow for the most growth (note the very low waiting cost as compared to alternatives A-1 and A-2).

2. The lower cost of the premium truck definitely changes the optimal cost alternative of question 1. The alternatives now available are:

Alt.	Mean service rate	Variable cost/unit	Fixed cost/simulation run	Description
A-1	15	0.10	\$ 800	new premium truck
B-2	same as alternative A-3			
B-3	15	.10	1600	two new premium trucks

The results from the computer runs are:

Alt.	Mean length Queue	Mean wait time	Service Util.	Cost wait	Var. cost	Fix cost	Total cost
A-1	1.0	21.5	74.2%	\$987.12	\$37.12	\$ 800	\$1814.22
B-2	.5	11.5	59.4	\$29.42	45.42	1600	1975.22
B-3	.1	1.1	37.2	\$1.25	27.40	1600	1678.65

Clearly Rio should select B-3 as it has by far the lowest cost and because of the low service utilization will allow for the greatest expansion.

3. The alternatives C-1, C-2, and C-3 are the same as A-1, A-2, and A-3. In these runs the arrival rate was increased from 3 to 4 per hour (Poisson). The results of the runs are as follows:

Alt.	Mean length Queue	Mean wait time	Service util.	Cost wait	Var. cost	Fix cost	Total cost
C-1	5.3	81.8	98.04	\$5288.55	24874	21000	28838.29
C-2	1.3	19.8	79.8	1285.82	64.42	1400	1750.22
C-3	.3	7.3	49.9	137.50	51.60	2000	2169.10

Alternative C-3 is the best selection. This is due, as was mentioned in part 1, to the fact that the alternative C-3 had room for expansion while the others were already being highly utilized. There is still quite a difference between the waiting time cost.

4. In this part we are asked to compare the result of part 1 using the other two priority rules, SOT (D-18, D-25, and D-31) and SAMCON (D-18, D-28, and D-31). The results of the computer runs are shown below:

Alt.	Mean length Queue	Mean wait time	Service util.	Cost wait	Var. cost	Fix cost	Total cost
A-1	1.0	21.5	74.34	1887.12	634.18	11000	12921.29
D-18	.8	19.0	76.2	1711.53	36.18	1000	1949.70
D-28	1.0	22.3	74.7	1926.80	36.57	1000	2063.37
A-2	.5	11.5	59.4	\$529.42	145.42	14000	14875.22
D-25	.5	10.8	59.4	494.89	45.41	1400	1440.32
D-28	.5	10.7	59.4	493.39	45.42	1400	1538.81
A-3	.3	1.1	37.2	\$ 51.25	138.44	12000	12687.69
D-31	.3	1.1	37.2	51.01	36.44	2000	2087.45
D-31	.3	1.1	37.2	52.84	36.44	2000	2089.29

In the first set of alternatives, D-25 is the best selection. This is due primarily to the fact that the mean waiting time is less for the SOT rule since the shorter jobs are worked on first and gotten out of the system quickly.

The runs using the deluxe truck, D-28 gave the best result. Here the range was not nearly as wide as for the single premium truck.

In the case of the two premium trucks (A-1, D-25, and D-31) the run using the SOT rule gave the best results. The best explanation for this is that because of the large amount of slack in the system (35% utilization) it really doesn't make too much difference which rule is used.

D-28 has the lowest cost of all nine runs. This indicates that under present conditions the deluxe truck would be the best selection and that the SAMCON priority rule should be used. In actually the company should probably select the deluxe truck and use a modified SOT rule (such SOT between 8am and 4pm, all jobs in QUEUE at 4pm on FCFS until 8am following morning).

#### CONCLUDING REMARKS

From the results of the run, the following would be recommended to Rio:

1. If Rio expects the arrival rate of calls to remain about 3/hr, they should buy the deluxe truck, provided the premium is not available at the reduced rate.
2. If arrival rate is expected to increase, buy the two premium trucks.

As is suggested, the two premium trucks are probably the best selection all around. One major reason for this would be that if one of the trucks broke down, the other would still be able to make service calls.

PROGRAM	VALUES		
MARCH 1972	M J HAWARD		1
THIS VERSION FOR CUC 3100			2
DICTIONARY OF VARIABLES			24
ATTIME	THE NUMBER OF SYSTEM IDLE TIME - TOTAL		3
ANUM(SUM)	NUMBER OF READS IN ARRIVALS AND SERVICE		4
ARR(SUM)	AN ARRAY OF READ IN ARRIVAL TIMES		5
ARR RATE	THE ARRIVAL RATE AND TIME		6
CM	AN ARRAY OF ARRIVAL AND SERVICE ON FIRST 20 CUSTOMERS		7
CIDLE	THE COST OF SYSTEM IDLE TIME - TOTAL		8
CUMQUE(LIST)	AN ARRAY WHICH STORES IDLE CUSTOMER HOURS		9
COST1	THE COST PER TIME UNIT OF IDLE SERVICE		10
COST2	THE COST PER TIME UNIT OF IDLE CUSTOMERS		11
CUSENVALS	THE NUMBER OF CUSTOMERS BEING SERVED		12
CWAIT	THE COST OF CUSTOMERS HOURS IN QUEUE - TOTAL		13
DEP RATE	THE SERVICE RATE AND MEAN DURATION		14
HRSND	THE HOURS OF CUSTOMER TIME IN QUEUE - TOTAL		15
ISJ	THE CHANNEL NUMBER BEING PROCESSED		16
IZ	NUMBER OF ARRIVALS WHICH HAVE OCCURED		17
MARKS	OPTION CODES FOR ARRIVALS AND SERVICE		18
NUMARR	BEGINNING-MAXIMUM NUMBER CHANNELS		19
PCUTIL	THE PERCENT UTILIZATION OF THE SERVICE FACILITY		20
QUE	THE NUMBER OF CUSTOMERS IN QUEUE AT ANY POINT		21
SRISND	AN ARRAY OF READ IN SERVICE TIMES		22
TCCOST	THE TOTAL COST OF THE SYSTEM		23
TIME,TIME	CLOCK TIME, MAX SIMULATION TIME		24
INARY	THE LATEST ARRIVAL TIME		25
INOPM	THE DEPARTURE TIME OF THE LATEST DEPARTURE		26
LNIN,ZNIN	MAY BE SET SPECIFICALLY FOR PROGRAM EFFICIENCY		27
AVARR	MEAN WAIT TIME, MEAN NUMBER IN QUEUE		28
AVSERV	AVERAGE ARRIVALS PER TIME UNIT		29
AMHS	AVERAGE SERVICE TIME		30
AMHS	MEAN NO. IN THE SYSTEM		31
AMTS	MEAN TIME IN THE SYSTEM		32
VCOST	TOTAL VARIABLE COST OF OPERATIONS		33
VCOST	VARIABLE COST PER UNIT		34
FCOST	TOTAL FIXED COST OF OPERATIONS		35
FCOST	FIXED COST PER UNIT		36
USVC	AN ARRAY OF QUEUED SERVICE TIMES		37
RULE	SCHEDULE RULE CODE (1=HARDON, 2=FCFS, 3=SOT)		38
COMMON	1=HARDON, 2=FCFS, 3=SOT, 4=ARRIVAL, 5=ARRIVAL, 6=ARRIVAL, 7=ARRIVAL, 8=ARRIVAL, 9=ARRIVAL, 10=ARRIVAL, 11=ARRIVAL, 12=ARRIVAL, 13=ARRIVAL, 14=ARRIVAL, 15=ARRIVAL, 16=ARRIVAL, 17=ARRIVAL, 18=ARRIVAL, 19=ARRIVAL, 20=ARRIVAL, 21=ARRIVAL, 22=ARRIVAL, 23=ARRIVAL, 24=ARRIVAL, 25=ARRIVAL, 26=ARRIVAL, 27=ARRIVAL, 28=ARRIVAL, 29=ARRIVAL, 30=ARRIVAL, 31=ARRIVAL, 32=ARRIVAL, 33=ARRIVAL, 34=ARRIVAL, 35=ARRIVAL, 36=ARRIVAL, 37=ARRIVAL, 38=ARRIVAL, 39=ARRIVAL, 40=ARRIVAL, 41=ARRIVAL, 42=ARRIVAL, 43=ARRIVAL, 44=ARRIVAL, 45=ARRIVAL, 46=ARRIVAL, 47=ARRIVAL, 48=ARRIVAL, 49=ARRIVAL, 50=ARRIVAL, 51=ARRIVAL, 52=ARRIVAL, 53=ARRIVAL, 54=ARRIVAL, 55=ARRIVAL, 56=ARRIVAL, 57=ARRIVAL, 58=ARRIVAL, 59=ARRIVAL, 60=ARRIVAL, 61=ARRIVAL, 62=ARRIVAL, 63=ARRIVAL, 64=ARRIVAL, 65=ARRIVAL, 66=ARRIVAL, 67=ARRIVAL, 68=ARRIVAL, 69=ARRIVAL, 70=ARRIVAL, 71=ARRIVAL, 72=ARRIVAL, 73=ARRIVAL, 74=ARRIVAL, 75=ARRIVAL, 76=ARRIVAL, 77=ARRIVAL, 78=ARRIVAL, 79=ARRIVAL, 80=ARRIVAL, 81=ARRIVAL, 82=ARRIVAL, 83=ARRIVAL, 84=ARRIVAL, 85=ARRIVAL, 86=ARRIVAL, 87=ARRIVAL, 88=ARRIVAL, 89=ARRIVAL, 90=ARRIVAL, 91=ARRIVAL, 92=ARRIVAL, 93=ARRIVAL, 94=ARRIVAL, 95=ARRIVAL, 96=ARRIVAL, 97=ARRIVAL, 98=ARRIVAL, 99=ARRIVAL, 100=ARRIVAL		39
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		40
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		41
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		42
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		43
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		44
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		45
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		46
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		47
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		48
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		49
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		50
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		51
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		52
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		53
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		54
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		55
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		56
	1 DEPT, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100		57





```

COMMON LPHNAT(10),AMUM,ARRPT,ARPTM,LMIGD,LOI,CUMUL,CUSERV,DEPART,
1 DEPTH=1+IMSEV+IZ+I+VPCUS+K+M+INFLAG+SNUM+STATUS(1)+T
2 TIME+TIME,INARV,INDEP(1),AYAHHE,YSERV,APRIS,APLIS,
3 MVLCT,VCOST,FCOST,KNOLE,LOUE,CUMQUE(100)
4 CUMQ(100)+MISW(1)+SRIS(1)
C THIS SUBROUTINE CALLED WHEN AN ARRIVAL IS THE NEXT OCCURANCE
C IT UPDATES THE TIME SPENT IN QUE
C IT UPDATES THE CLCKX TO THE TIME OF THE NEW ARRIVAL (PREVIOUSLY
C SELECTED)
C IF CHANNEL EACH CHANNEL TO SEE IF THE NEW ARRIVAL CAN BEGIN SERVICE
C IF A CHANNEL IS AVAILABLE IT DOES THE FIRST PART OF THE
C DEPART PROCESSING OTHERWISE IT ADDS ONE TO THE QUE
C LASTLY IT SELECTS THE TIME FOR THE NEXT ARRIVAL TO OCCUR
C IF TIME.EQ.TJUV) GO TO 11
C CHECK LENGTH OF QUE, IF OVER 99 MULD AT 99
C QUE=100
C UPDATE MINS SPENT IN QUEUE
11 CONTINUE
CUMQUE(IJ)=CUMQUE(IJ)+INARV+TIME
C UPDATE LINCA TIME TO NEXT ARRIVAL
TIME=INARV
C CHECK EACH CHANNEL, IF STATUS = 0 IT IS AVAILABLE
GO TO 21
C IF (TIME.EQ.TJUV) GO TO 21
C DO FIRST PART OF THE DEPART PROCESSING
STATUS(IJ)=1
C SET TIME OF NEXT DEPARTURE
INDEP(IJ)=TIME+OSVCT(IJ)
C STORE FIRST TWENTY DEPARTURE TIMES IN CM
NCUS*RCUS+1
C IF (ALUS.EQ.2) GO TO 21
11J=1
CMINCUS+1+INDEP(IJ)
21 CONTINUE
C CHECK IF OUT OF SIMULATION TIME
C IF (TIME.EQ.TJUV) GO TO 31
C ACCUMULATE PROCESSING TIME
CUMUL=CUMUL+OSVCT(IJ)
CUSERV=CUSERV+1
GO TO 41
C END OF SIMULATION UPDATING
LAST DEPARTURE FORCED OUT AT TIME
31 CONTINUE
1K=OSVCT(IJ)-INDEP(IJ)+TIME
C IF (1K.EQ.0) GO TO 41
1AP=
41 CONTINUE
INDEP(IJ)=TIME
CUMUL=CUMUL+1K
GO TO 41
51 CONTINUE
C IF (1K.EQ.0) GO TO 71
61 CONTINUE
ALL CHANNELS ARE MUST, ADD ONE TO THE QUE
71 CONTINUE
LOUE=LOUE+1
C SELECT ARRIVAL TIME (STORE FIRST TWENTY IN CM)
81 CONTINUE
GO TO (1)+101+11+12)+ K4
C POISSON ARRIVAL TIME DISTRIBUTION
91 CONTINUE
R=NUM(1)+1
TNARV=ABS(ARRPT*ALUB(1))
TNARV=TNARV+TIME
GO TO 101
C NEGATIVE EXP ARRIVAL TIME DISTRIBUTION

```

38  
 39  
 40  
 41  
 42  
 43  
 44  
 45  
 46  
 47  
 48  
 49  
 50  
 51  
 52  
 53  
 54  
 55  
 56  
 57  
 58  
 59  
 60  
 61  
 62  
 63  
 64  
 65  
 66  
 67  
 68  
 69  
 70  
 71  
 72  
 73  
 74  
 75  
 76  
 77  
 78  
 79  
 80  
 81  
 82  
 83  
 84  
 85  
 86  
 87  
 88  
 89  
 90  
 91  
 92  
 93  
 94  
 95  
 96  
 97  
 98  
 99  
 100  
 101  
 102  
 103  
 104  
 105  
 106  
 107  
 108  
 109  
 110  
 111  
 112  
 113  
 114  
 115  
 116  
 117  
 118  
 119  
 120  
 121  
 122  
 123  
 124  
 125  
 126  
 127  
 128  
 129  
 130  
 131  
 132  
 133

```

101 CONTINUE
R=NUM(1)+1
TNARV=ABS(ARRPT*ALUB(1))
TNARV=TNARV+TIME
GO TO 101
C CONSTANT ARRIVAL TIME
111 CONTINUE
TNARV=TIME+ARPTM
GO TO 101
C READ IN ARRIVAL TIMES
121 CONTINUE
NUM=NUM
IZ=IZ+1
C IF (IZ.EQ.LE.NUM) GO TO 131
ARIZ(IZ)=77777777
NFLAG=76
131 CONTINUE
INARV=ARIZ(IZ)
CONTINUE
141 CONTINUE
IZ=IZ+1
C IF (IZ.EQ.20) GO TO 151
CMIZ(IZ)=INARV
C BEGIN LOGIC TO STORE ARRIVAL AND SERVICE TIMES IN QUEUE ARRAYS
C FOR EACH WAITING CUSTOMER/PRODUCT
C DETERMINE SERVICE TIME FOR THE NEXT ARRIVAL
151 CONTINUE
GO TO (101)+11+101)+ K3
C POISSON SERVICE RATE
161 CONTINUE
R=NUM(1)+1
TNARV=DEPRT+ALUB(1)
GO TO 211
C NEGATIVE EXPONENTIAL SERVICE TIME
171 CONTINUE
R=NUM(1)+1
TNARV=DEPRT*ALUB(1)
GO TO 211
C CONSTANT SERVICE TIME
181 CONTINUE
TNARV=
GO TO 211
C READ IN SERVICE TIME
191 CONTINUE
NUM=NUM
C IF (IZ.EQ.LE.NUM) GO TO 201
NFLAG=76
TNARV=000000
GO TO 211
201 CONTINUE
TNARV=
211 CONTINUE
OSVCT(IJ)=T
C IF ONLY ONE IN QUE + NO SCHEDULING NECESSARY
C IF (LOUE.EQ.2) GO TO 241
IDUE=LOUE
C USE SCHEDULE RULE TO REOPEN THE QUEUE FOR PROCESSING
C RULE = 1 FOR RANDOM
C RULE = 2 FOR FLYS
C RULE = 3 FOR SUT
GO TO (23)+24+22)+ KRULE
C NO SCHEDULE RULE
221 CONTINUE
C IF (NUM.EQ.2) GO TO 241
C IF (OSVCT(IJ).GE.OSVCT(IJ-1)) GO TO 241
OSVCT(IJ)=1+OSVCT(IJ)

```







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

**INTRODUCCION A LA PROGRAMACION Y  
COMPUTACION ELECTRONICA**

**A N E X O S**

**OCTUBRE, 1982**

## TARJETAS PARA USAR EL PROGRAMA GRANM

	Columna 1
Tarjeta 1	// JOB 1
Tarjeta 2	// XEQ GRANM 1
Tarjeta 3	*LOCALINIT, PIVOT, TABNU, RMØVE, CLEAN, TABPR
	-
	- Tarjetas de datos (Ver página 3)
	-
	-
Tarjeta final	^

### NOTAS:

- Este programa está listo para usarse en la computadora IBM 1130 de CECAFI.
- La tarjeta 1 es la tarjeta anaranjada obtenida del CECAFI.
- El número 1 que aparece en la 2ª tarjeta se perfora en la columna 17.
- El programa en la IBM, tiene una capacidad de 10 restricciones y 15 variables incluyendo de holgura y artificiales.
- Este programa también se encuentra disponible en la Burroughs del CIMASS, bajo el nombre de I/SIMPLEX. Las instrucciones para convertirlo en el CIMASS aparecen en la siguiente hoja. Este admite una capacidad mayor sobre el número de restricciones y variables como se indica en la segunda hoja.
- Este programa utiliza el método de la gran M.

## TARJETAS PARA USAR EL PROGRAMA I/SIMPLEX

	Columna 1
Tarjeta 1	# USER <u>clave</u> / _____
Tarjeta 2	# RUN (JOB) I/SIMPLEX
Tarjeta 3	# DATA FILE 5
	-
	- Tarjetas de datos (Ver página 3)
	-
	-
Tarjeta final	# END

### NOTAS:

- Este programa está listo para usarse en la computadora B 6700 de CIMAS/CSC.
- La tarjeta 1 es la tarjeta roja obtenida del CIMASS.
- El símbolo "#" significa un carácter inválido. Este se obtiene presionando los teclas MULTIPUNCH Y NUMERIC simultáneamente y perforando los números 1, 2, 3, 4.
- Este programa tiene una capacidad de 30 restricciones y 40 variables incluyendo de holgura y artificiales.

## TARJETAS DE DATOS PARA EL PROGRAMA GRANM O I/SIMPLEX

La siguiente información deberá proporcionarse en la que se indica como tarjetas de datos en las hojas anteriores.

### TARJETA DE IDENTIFICACION DEL PROBLEMA.

En esta tarjeta puede usar desde la columna 1 a la 70 para poder dar cualquier identificación que desea dar a su problema.

### TARJETA DE DIMENSION Y ETIQUETACION DEL PROBLEMA Y CONTROL PARA CORRER MAS DE UN PROBLEMA.

El usuario debe dar cuatro números enteros con formato (4110) en la siguiente forma:

Columnas 1-10: Número de renglones del problema.

Columnas 11-20: Número de columnas del problema.

Columna 30: Escriba el número 1 si desea poner etiquetas a los renglones y a las columnas. Escriba el número 0 en caso contrario.

Columna 40: Escriba un 1 si desea correr un problema adicional. Escriba un 0 en caso contrario.

### NOTAS:

El número de renglones no incluye la función objetivo.

Si escribe un 1 en la columna 30, el usuario, después de la tarjeta deberá dar el grupo de tarjetas para etiquetar de renglones y el grupo de tarjetas para etiquetar de columnas. Si en lugar de un 1 escribe cero deberá omitir este grupo de tarjetas y pasar a las tarjetas de coeficientes de las variables artificiales en la función objetivo.

Si escribe un 1 en la tarjeta 40 vea las notas generales.

### TARJETAS PARA ETIQUETAS DE RENGLONES.

Las etiquetas para identificar a los renglones de las restricciones, pueden tener como máximo 6 caracteres de cualquier tipo.

En una tarjeta puede escribir hasta 7 etiquetas. Estas etiquetas deben ir en las columnas 1-6, 11-16, 21-26, 31-36, 41-46, 51-56, 61-66.

### TARJETAS PARA ETIQUETAS DE COLUMNAS (VARIABLES)

Las tarjetas para identificar a las columnas o sea a las variables involucradas en el problema (incluyendo de holgura y artificiales) deberán escribirse de acuerdo a las reglas anteriores para etiquetar renglones.

### TARJETAS DE COEFICIENTES DE LAS VARIABLES ARTIFICIALES EN LA FUNCION OBJETIVO.

A cada variable artificial asignela un 1 y a las variables no artificiales asignela un 0. Estos números escríbalos en las columnas 10, 20, 30, 40, 50, 60, 70, de acuerdo al orden en que etiquetó a sus variables (columnas). **IMPORTANTE.** Esta tarjeta es requerida aún si el problema no tiene variables artificiales.

### TARJETAS DE COEFICIENTES DE LAS VARIABLES NO ARTIFICIALES EN LA FUNCION OBJETIVO.

Escriba los coeficientes de la función objetivo con el formato (7 F 10.0). Estos coeficientes debe escribirlos de acuerdo al orden en que etiquetó sus variables (columnas). Los coeficientes de las variables de holgura y artificiales deberá ser cero.

**IMPORTANTE:** Los coeficientes de la función objetivo deben corresponder al problema de minimizar. Por lo tanto, si su problema es de maximizar multiplique por -1 y considere los coeficientes que resulten como los datos de entrada en este programa.

### TARJETAS DE LOS COEFICIENTES DE LA MATRIZ DE RESTRICCIONES.

Cada renglón de restricciones va en una o varias tarjetas, escribiendo los elementos sucesivamente en una tarjeta con un formato (7 F 10.0). Cada vez que proporcione un nuevo renglón debe empezarlo en otra tarjeta.

### TARJETAS DE LOS LADOS DERECHOS DE LAS RESTRICCIONES.

Los coeficientes del lado derecho de restricciones se proporcionan sucesivamente en una tarjeta o en caso de ser insuficiente use otra tarjeta. El formato es (7 F 10.0).

### TARJETAS PARA INDICAR EL CONJUNTO INICIAL DE VARIABLES BASICAS.

En una tarjeta programe sucesivamente los números de las columnas que van a ser usadas como columnas (variables) básicas iniciales. Use formato (7 I 10).

### NOTAS GENERALES:

1. El orden de las tarjetas debe ser como el indicado.
2. Si en la TARJETA DE DIMENSION Y ETIQUETACION escribió un 1 en la columna 40 entonces su nuevo problema debe ir después de la TARJETA PARA INDICAR EL CONJUNTO INICIAL DE VARIABLES ARTIFICIALES. Es importante que en el nuevo problema emplee con la TARJETA DE IDENTIFICACION DEL PROBLEMA,

EJEMPLO 1. Considere el problema lineal

$$\max z = x_4 + x_5$$

$$\begin{aligned} \text{s.t.} \\ 2x_2 - x_3 - x_4 + x_5 &\geq 0 \\ -2x_1 + 2x_3 - x_4 + x_5 &\geq 0 \\ x_1 - 2x_2 - x_4 + x_5 &\geq 0 \\ x_1 + x_2 + x_3 &= 1 \\ x_1 &\geq 0 \end{aligned}$$

Deberemos multiplicar la función objetivo por  $-1$  para que el problema sea de minimización y también agregar variables de holgura a las primeras tres restricciones para que lleguen a ser igualdades. Con estas observaciones el problema lineal estará en forma estándar, lo cual es una condición para aplicar el programa GRAN M. Si definimos  $x^* = z$ , nuestro problema en forma estándar es

$$\begin{aligned} \min z^* &= -x_4 + x_5 \\ -2x_2 + x_3 + x_4 - x_5 + x_1 &= 0 \\ 2x_1 - 2x_3 + x_4 - x_5 + x_2 &= 0 \\ -x_1 + 2x_2 - x_4 + x_5 + x_3 &= 0 \\ x_1 + x_2 + x_3 &= 1 \\ x_1 &\geq 0, \quad i=1, 2, \dots, 5 \\ x_2 &\geq 0, \quad i=1, 2, 3 \end{aligned}$$

Obsérvese que aunque el problema lineal ya está en forma estándar, todavía no está listo para empezar el algoritmo de la Gran M porque en la última restricción no existe una variable que aparezca en esta restricción pero no se encuentre en las otras restricciones. (i.e., no se tiene una solución básica factible inmediata). Por lo tanto, deberemos agregar una variable artificial que llamaremos  $t_1$ , a la cuarta restricción para así completar nuestra solución básica factible en la cual se inició el algoritmo. Sin embargo, al introducir esta variable artificial en la restricción deberemos agregarla en la función objetivo multiplicada por una cantidad positiva  $M$  muy grande. Así nuestro problema resulta ser

$$\begin{aligned} \min z^* &= -x_4 + x_5 + Mt_1 \\ -2x_2 + x_3 + x_4 - x_5 + t_1 &= 0 \\ 2x_1 - 2x_3 + x_4 - x_5 + x_2 &= 0 \\ -x_1 + 2x_2 - x_4 + x_5 + x_3 &= 0 \\ x_1 + x_2 + x_3 + t_1 &= 1 \\ x_1 &\geq 0, \quad i=1, 2, \dots, 5 \\ t_1 &\geq 0, \quad i=1, 2, 3 \end{aligned}$$

Es conveniente representar el programa lineal en un tablero (o tableau), para poder entender más fácilmente la información que deberemos proporcionar al programa de computadora GRAN M o I/SIMPLEX. Esta representación aparece abajo

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_1$	$x_2$	$x_3$	$t_1$	$z^*$
Función Obj. (F.O.)	0	0	0	-1	1	0	0	0	M	0
Restrición 1 (R. 1)	0	-2	1	1	-1	1	0	0	0	0
Restrición 2 (R. 2)	2	0	-2	1	-1	0	1	0	0	0
Restrición 3 (R. 3)	-1	2	0	1	-1	0	0	1	0	0
Restrición 4 (R. 4)	1	1	1	0	0	0	0	0	1	1

Var. Holgura  
ro  
Var.  
Art.  
Solución Inicial  
básica factible

Este tablero contiene toda la información necesaria y la notación apropiada para correr el programa GRAN M o el I/SIMPLEX. A continuación se presenta su codificación para el GRAN M. Para correr el I/SIMPLEX la codificación es idéntica excepto por las tarjetas de control como se mencionó en la explicación de estos programas.

UNAM		HOJA DE CODIFICACION Y/O DATOS FORTRAN		PROGRAMA DE INVESTIGACION	
PROGRAMA	PROBLEMA	HOJA	IN	SEC	CONTINUA
11	11	11	11	11	11
12	12	12	12	12	12
13	13	13	13	13	13
14	14	14	14	14	14
15	15	15	15	15	15
16	16	16	16	16	16
17	17	17	17	17	17
18	18	18	18	18	18
19	19	19	19	19	19
20	20	20	20	20	20
21	21	21	21	21	21
22	22	22	22	22	22
23	23	23	23	23	23
24	24	24	24	24	24
25	25	25	25	25	25
26	26	26	26	26	26
27	27	27	27	27	27
28	28	28	28	28	28
29	29	29	29	29	29
30	30	30	30	30	30
31	31	31	31	31	31
32	32	32	32	32	32
33	33	33	33	33	33
34	34	34	34	34	34
35	35	35	35	35	35
36	36	36	36	36	36
37	37	37	37	37	37
38	38	38	38	38	38
39	39	39	39	39	39
40	40	40	40	40	40
41	41	41	41	41	41
42	42	42	42	42	42
43	43	43	43	43	43
44	44	44	44	44	44
45	45	45	45	45	45
46	46	46	46	46	46
47	47	47	47	47	47
48	48	48	48	48	48
49	49	49	49	49	49
50	50	50	50	50	50
51	51	51	51	51	51
52	52	52	52	52	52
53	53	53	53	53	53
54	54	54	54	54	54
55	55	55	55	55	55
56	56	56	56	56	56
57	57	57	57	57	57
58	58	58	58	58	58
59	59	59	59	59	59
60	60	60	60	60	60
61	61	61	61	61	61
62	62	62	62	62	62
63	63	63	63	63	63
64	64	64	64	64	64
65	65	65	65	65	65
66	66	66	66	66	66
67	67	67	67	67	67
68	68	68	68	68	68
69	69	69	69	69	69
70	70	70	70	70	70
71	71	71	71	71	71
72	72	72	72	72	72
73	73	73	73	73	73
74	74	74	74	74	74
75	75	75	75	75	75
76	76	76	76	76	76
77	77	77	77	77	77
78	78	78	78	78	78
79	79	79	79	79	79
80	80	80	80	80	80
81	81	81	81	81	81
82	82	82	82	82	82
83	83	83	83	83	83
84	84	84	84	84	84
85	85	85	85	85	85
86	86	86	86	86	86
87	87	87	87	87	87
88	88	88	88	88	88
89	89	89	89	89	89
90	90	90	90	90	90
91	91	91	91	91	91
92	92	92	92	92	92
93	93	93	93	93	93
94	94	94	94	94	94
95	95	95	95	95	95
96	96	96	96	96	96
97	97	97	97	97	97
98	98	98	98	98	98
99	99	99	99	99	99
00	00	00	00	00	00

EJEMPLO 2

$$\begin{aligned} \max z &= x_1 + x_2 \\ \text{s.o.} \\ x_1 + x_2 &\leq 1 \\ x_1 + x_2 &\leq 1 \\ -x_1 + x_2 &\leq 1 \\ x_1 &\geq 0 \end{aligned}$$

Expresando la función objetivo en términos de minimización e introduciendo variables de holgura, artificiales, el problema es equivalente a:

$$\begin{aligned} \min (-z) &= -x_1 - x_2 + Mx_1 \\ x_1 + x_2 - s_1 &= 1 \\ x_1 - x_2 + s_2 &= 1 \\ -x_1 + x_2 + s_3 &= 1 \end{aligned}$$

En forma de tableau:

Func. Obj. (F.O.)	$x_1$	$x_2$	$s_1$	$s_2$	$s_3$	$-z$	
Renglón 1 (R.1)	1	1	-1	1	0	0	1
Renglón 2 (R.2)	1	-1	0	0	1	0	1
Renglón 3 (R.3)	-1	+1	0	0	0	1	1

Solución básica factible inicial.

UNAM		HOJA DE CODIFICACION Y/O DATOS FORTRAN		CARRERA DE INGENIERIA EN SISTEMAS DE COMPUTACION	
GRUPO	SECCION	FECHA	OR	SECUNDA	TERCERA
1	1				
2	2				
3	3				
4	4				
5	5				
6	6				
7	7				
8	8				
9	9				
10	10				
11	11				
12	12				
13	13				
14	14				
15	15				
16	16				
17	17				
18	18				
19	19				
20	20				
21	21				
22	22				
23	23				
24	24				
25	25				
26	26				
27	27				
28	28				
29	29				
30	30				
31	31				
32	32				
33	33				
34	34				
35	35				
36	36				
37	37				
38	38				
39	39				
40	40				
41	41				
42	42				
43	43				
44	44				
45	45				
46	46				
47	47				
48	48				
49	49				
50	50				
51	51				
52	52				
53	53				
54	54				
55	55				
56	56				
57	57				
58	58				
59	59				
60	60				
61	61				
62	62				
63	63				
64	64				
65	65				
66	66				
67	67				
68	68				
69	69				
70	70				
71	71				
72	72				
73	73				
74	74				
75	75				
76	76				
77	77				
78	78				
79	79				
80	80				
81	81				
82	82				
83	83				
84	84				
85	85				
86	86				
87	87				
88	88				
89	89				
90	90				
91	91				
92	92				
93	93				
94	94				
95	95				
96	96				
97	97				
98	98				
99	99				
100	100				

EJEMPLO 3 Resolver el dual del siguiente par de problemas primal - dual.

Primal

$$\begin{aligned} \min z &= 2x_1 - 3x_2 \\ 2x_1 - x_2 - x_3 &\geq 3 \\ x_1 - x_2 + x_3 &\geq 2 \\ x_1 &\geq 0 \end{aligned}$$

Dual

$$\begin{aligned} \max w &= 3\lambda_1 + 2\lambda_2 \\ 2\lambda_1 + \lambda_2 &\leq 2 \\ -\lambda_1 - \lambda_2 &\leq -3 \\ -\lambda_1 - \lambda_2 &\leq 0 \\ \lambda_1 &\geq 0 \end{aligned}$$

Este dual es equivalente a

$$\begin{aligned} \min (-w) &= -3\lambda_1 - 2\lambda_2 + M\lambda_3 \\ 2\lambda_1 + \lambda_2 + \lambda_3 &= 2 \\ \lambda_1 + \lambda_2 - \lambda_3 &= 3 \\ -\lambda_1 + \lambda_2 + \lambda_3 &= 0 \end{aligned}$$

En forma de tableau, el dual está dado por

	$\lambda_1$	$\lambda_2$	$\lambda_3$	$\lambda_4$	$\lambda_5$	
F.O.	-3	-2	0	0	M	0
R1	2	1	0	1	0	2
R2	1	1	-1	0	1	3
R3	-1	1	0	0	0	1





UNAM

HOJA DE CODIFICACION Y/O DATOS FORTRAN

FACULTAD DE  
INGENIERIA

PROGRAMA	PROGRAMACION				FECHA	IDENTIFICACION Y FRECUENCIA	
// XEGRANM							
LOCALINIT PIVOT, IASNU, RMOVE, CLEAN, IASPR							
EJEMPLO 4.							
	4	7	1	0			
	1.2	1.3	1.4				
X1	X2	X3	X4	X5	X6	X7	
0	0	0	0	0	0	0	
-1.0	1.0	-1.0	0.0	-1.0	1.0	2.0	
0.0	0.0	0.0	0.0	1.0	1.0	0.0	
0.0	1.0	2.0	-1.0	0.0	0.0	0.0	
1.0	0.0	0.0	0.0	0.0	-1.0	0.0	
0.0	0.0	1.0	0.0	0.0	1.0	1.0	
0.0	10.0	0.0	-4.0				
	5	2	1	7			

Directorio de Alumnos del curso: INTRODUCCION A LA PROGRAMACION Y

Computación Electrónica Del 8 de octubre al 4 de noviembre de

1982

- |  |  |  |  |
|--|--|--|--|
| 1. Laura Castellana Martínez<br>Instituto Mexicano del Petróleo<br>Eje Central Lázaro Cárdenas 156<br>México, D.F.                               | Centrall No. 06<br>Centro<br>Cuernavaca<br>México, D.F.<br>521 63 63                         | 9. Víctor Chávez Castro<br>6007 Xanayuan<br>México, D.F.<br>792 32 88 Ext. 218   | Sur 77 6 4234<br>y. Piedad<br>México, D.F.<br>530 25 69                                    |
| 2. Maritza Mirto Viquez<br>Universidad Autónoma de Tlaxcala<br>Carr. a Arizauque s/n<br>Apizaco, Tlaxcala<br>77546                               | Manuel Saldaña 14<br>Sta. Ana Chantapan, Tlaxcala<br>2 13 67                                 | 10. Alberto Espinosa Bustiata<br>Cecilia Fornet y Asociados Arquitectos, S.C.<br>Bartola Novedades 113<br>Del Valle<br>B. Juárez<br>México 18, D.F.<br>575 53 73 | Calle Vienna 219<br>Coyacahuacan<br>México 21, D.F.<br>534 78 43                           |
| 3. Rafael de la Cruz Guadalupe<br>Facultad de Ingeniería<br>U N A M<br>México, D.F.<br>550 52 15 Ext. 3639                                       | Av. 16 de Sept. No. 45<br>Barrío Baltazar<br>Zochimilco, D.F.<br>16000 México, D.F.          | 11. José A. Espinosa Pérez<br>Universidad Autónoma de Chapingo<br>Chapingo, Edo. de Méx.<br>505 45 55 Ext. 5506  | Salazar 120<br>Tehuacan, Tex.<br>4 00 64   |
| 4. Néstor Aguilar Gutiérrez<br>AMOP<br>Plaza de la Constitución No. 7<br>Centro<br>Cuernavaca<br>06000 México, D.F.<br>510 31 04                 | Av. 16 de Sept. No. 45<br>Barrío Baltazar<br>Zochimilco, D.F.<br>16000 México, D.F.          | 12. Javier Fuentes Rojas<br>PPAC<br>Facultad de Ciencias, UNAM, de Méx.  | Arturo 410 H. 25 X 40  |
| 5. Elites Luis Antonia Rodríguez<br>Teleconstrutora<br>Río Pánuco 30<br>México, D.F.<br>564 97 08  | Ralph Roeder No. 2802<br>Col. Istacriltepetl<br>B. Juárez<br>07530 México, D.F.<br>579 12 27 | 13. Guadalupe García Guerrero<br>S A R B<br>Reforma 35 10° Piso<br>Centro<br>Cuernavaca<br>06030 México, D.F.<br>582 50 22 Ext. 42                               |  |
| 6. Gabriel Arullano Martínez<br>S A R O P<br>Palomas 622<br>Narvarte<br>México, D.F.   | Ayuntamiento 113-12<br>Centro<br>Cuernavaca<br>06040 México, D.F.<br>512 53 79               | 14. Octavio González Paul<br>S A R B<br>Reforma 35 10° Piso<br>Centro<br>Cuernavaca<br>06030 México, D.F.<br>564 11 23   | Piedad 121 - Capta. 4<br>Sta. Sta. María<br>Acapulcalco<br>02000 México, D.F.<br>150 25 92 |
| 7. José Luis Castellana Eraso<br>I. la Católica No. 639-4<br>Alamos<br>B. Juárez<br>México, D.F.<br>530 73 01                                    | León 70-804<br>Nativitas<br>B. Juárez<br>03300 México, D.F.                                  | 15. Federico Guerra Córdova<br>Secretaría del Dr. Flores<br>Castro, Pab. 59<br>Estado de México<br>672 14 21   | Playa Revolucionaria 570<br>Bd. Istacriltepetl<br>B. Juárez<br>México, D.F.<br>590 03 72   |
| 8. Hospital Colecto Jiménez<br>Dirección General de Comunicaciones<br>del Estado de Veracruz<br>Cuernavaca 11<br>Salina<br>Xalapa, Ver.<br>76210 | Manuel No. Contreras 5 A<br>Dr. Rafael Lucio<br>Xalapa, Ver.<br>545 27                       | 16. Gilberto Gutiérrez García<br>SAMP<br>Dirección General de Ciencias Sociales  |  |
|  |  | 17. Antonio Bernaldo Villanueva<br>Instituto Mexicano del Petróleo<br>Eje Central Lázaro Cárdenas<br>C.A. Madara<br>México, D.F.<br>3681722 - Ext. 24            | Chilpancingo 77-15<br>Valle Coyula<br>Tlalampantla<br>Edo. de México<br>390 22 54          |
|  |  | 18. Filigonia Lara Jaime<br>Dirección General de Comunicaciones del Edo. de Ver.<br>Cuernavaca 11<br>Xalapa, Ver.<br>7 42 10                                     | León 71<br>Cuernavaca, Ver.<br>4 00 63   |

19. Antonio López López  
I N P  
Eje Central Lázaro Cárdenas 152  
México, D.F.
20. Alejandro López Villagómez  
Plásticos Alpa S.A.  
P. Carrillo Pto. 623  
Paseo  
Toluca  
México, D.F.  
527 28 66
21. Carlos Montiel Martínez  
Constructora y Urbanizadora Car. S.A.  
Av. Eze. León 144- Entradas  
Hipódromo Condesa  
Cuauhtémoc  
06170 México, D.F.  
574 90 15
22. Roberto Estens  
SAMP  
Vértiz 1243  
Vértiz Harvarte  
B. Juárez  
México, D.F.  
674 30 91 Ext. 211
23. Genaro G. Rodríguez Bravo  
SAMP  
Kola y Av. Universidad  
Harvarte  
B. Juárez  
03079 México, D.F.  
530 30 00 Ext. 369
24. Enrique Rodríguez Delgado  
IMM  
Coyocacán  
México, D.F.  
548 96 63
25. María de L. Sánchez Ríos  
S A N O P  
Kola y Av. Universidad  
Harvarte  
B. Juárez  
03079 México, D.F.  
530 30 00
26. Jorge Téllez Serrano  
Facultad de Estudios Superiores Cuautitlán  
Estado de México
27. Waldo Terry Carrillo  
PROCEL  
Luzburgo 11  
Juárez  
06600 México, D.F.  
566 57 97
- Cransón 116  
Ex-Hipódromo de Paralvillo  
Cuauhtémoc  
583 26 58
- Libertad 18  
Arboledas  
Edo. de México  
527 28 66
- Rosé Rangel Vergara  
Constructora Rarver, S.A.  
Av. Revolución 1387-2  
Los Alpes  
A. Obregón  
01010 México, D.F.  
651 46 53
- Rubén 31  
México.  
B. Juárez  
México, D.F.
- Sur 121 # 2288-3  
Juvencino Rosas  
Ixtacalco  
09700 México, D.F.
- Calzada Orul. Anaya 328  
Coyocacán  
04100 México, D.F.  
549 83 76
- Privada "A" Manz 1 L. 26  
Pantitlán  
Ixtacalco  
México, D.F.  
763 41 83
- Vía de Tierra 28  
Fracc. Arco del Alba  
Cuautitlán Izcalli, Edo. de México  
30554
- Retorno 103 # 18 Q  
Lomas de Sotelo  
M. Hidalgo  
México, D.F.  
537 44 20
28. César R. Ulloa de León  
Dirección General de Grandes Irrigación  
Reforma 15 Piso 10°  
México 1, D.F.  
Centro  
Cuauhtémoc  
06030 México, D.F.  
582 50 22 Ext. 60
29. Rosa M. Vargas López  
SAMP  
Petón 552  
México, D.F.  
674 11 04 Ext. 245
30. Gabriel Vázquez Guerrero  
I N P  
Av. Lázaro Cárdenas 152  
México, D.F.
- Carriola 15 Casa 37  
Fracc. la Sabaña  
Conjunto Villa Sol  
Atizapán Edo. de México  
566 11 63
- Av. 525 No. 146  
Unidad Sn. J. de Aragón  
C.A. Madero  
07920 México, D.F.  
551 83 40
- Regina 26 # 6  
Cuauhtémoc  
06000 México, D.F.  
512 27 51