CNEA-AC21

COMISION NACIONAL DE ENERGIA ATOMICA
DEPENDIENTE DE LA PRESIDENCIA DE LA NACION

AREA INVESTIGACION, DESARROLLO Y SERVICIOS

TEMAS DE COMPUTACION
APUNTES DE LAS CLASES DICTADAS EN JUNIO-SETIEMBRE 1974

M.G. DEL FRANCO Y E.C. MARENGO

Departamento de Metalurgia
Buenos Aires-Argentina
1975

I N D I C E

INTRODUCCION

Estos apuntes corresponden a las clases que se dictaron en el Dto. de Metalurgia de la Comisión Nacional de Energía Atómica durante los meses de junio a setiembre de 1974.

En general, los apuntes no son originales; se pretende con ellos recopilar información que está dispersa en muchas publicaciones diferentes y definir ciertos conceptos fundamentales (Caps. II, III y IV) sin los cuales es inútil consultar la bibliografía especializada o los manuales de uso de las diversas computadoras, crípticos y farragosos.

Durante todo el curso se hizo referencia a dos sistemas: el Bull G.E. 635 implementado en YPF y -primordialmente- el IBM/360-65 del Ministerio de Bienestar Social. Esto se debió al uso preferente que hace Metalurgia de esos dos equipos. Sólo los capítulos II, III, IV y VI son independientes de los sistemas que el Departamento emplea.

El capítulo V es un extracto del Manual Fortran Programmer's Guide de IBM. El capítulo VI es un resumen de la publicación Fortran Optimization Techniques, producida por el Grupo de Desarrollo de Computación del Centro de Harwell de la Autoridad de Energía Atómica Británica.

CAPITULO I

REPASO DE INSTRUCCIONES POCO HABITUALES EN FORTRAN IV BASICO: Common, Data, Equivalence, Entry, External, Block Data.

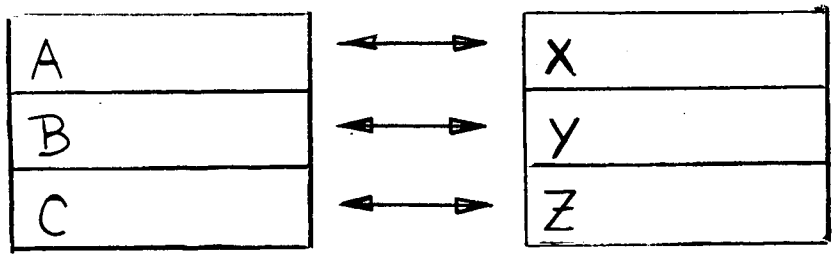
1. Common

1.1 Comunicación entre programa principal y subrutinas en el caso de parámetros explícitos.

Ejemplo:

```
DIMENSION  B (20)                SUB1  (X,Y, Z)
     --                          DIMENSION  Y (20)
     --                               --
     --                               --
CALL  SUB1 (A,B,C)                    --
```

Main Sub1



i) se multiplica memoria tantas veces como subrutinas haya.

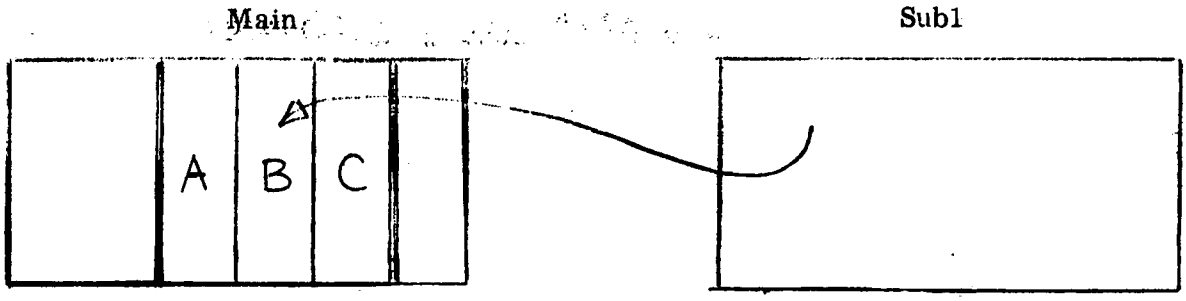
ii) mecanismo: cada vez que se llama a Sub1 se transfiere el contenido en ese momento de cada variable.

1.2 Comunicación por Common

Programa Principal

```
COMMON   A,B(20),C               SUB1
     --                          COMMON  X,Y(20), Z
     --                               --
     --                               --
CALL  SUB1                            --
     --                               --
     --                               --
```

Diagrama de memoria



Usa la misma zona y no transfiere cada vez.

i) hay una misma zona de memoria para cada variable compartida que se haya incluído en Common.

ii) no hay transferencia de valores con cada llamada.

1.3 Common rotulado

Se usa por comodidad
a) para no repetir Common muy largo: hay variables mudas.
b) Seguridad: olvidos en Common largos.
c) claridad

2. Data

2.1 Sirve para asignar valores numéricos o alfanuméricos a variables o arreglos tanto en programas como en subprogramas. La asignación se efectúa durante la compilación: al iniciarse la ejecución del programa esas variables ya tienen sus valores asignados.

2.2 Puede haber varios data.

2.3 Los valores que se asignan deben corresponder en tipo y en número con la definición de variables que los contendrán.

2.4 Una variable cuyo valor es asignado por Data no puede ser pasada en Common.

2.5 La asignación en los arreglos bidimensionales se hace por columna.

Ejemplo 1:

```
DATA  VAR/3.2/ , NEG/-4/
```

Ejemplo 2:

```
REAL * 8   VEC (2)              alfanuméricos
DATA  VEC/'NEGATIVO' , 'POSITIVO'/
o bien
DATA  VEC/8HNEGATIVO , 8HPOSITIVO/
```

Ejemplo 3:

```
DIMENSION AMAT1 (2,2),  AMAT2 (2,2)
DATA AMAT1/3.1,02,5.8,2.4/
DATA AMAT2/4*0.0/
          3.1    5.8                    0    0
AMAT1                          AMAT2
          0.2    2.4                    0    0
```

3. Equivalence

3.1 Definición: dos o más variables declaradas equivalentes comparten la misma ubicación de memoria dentro de un mismo programa .

3.1.1 Forma

EQUIVALENCE $(a_1, a_2, a_3 \ldots a_n)$, $(b_1, b_2, b_3 \ldots b_p)$

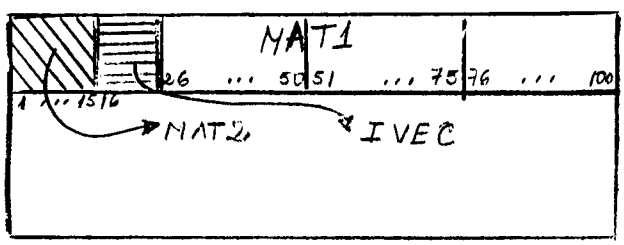
Todas las variables dentro de un paréntesis comparten la misma ubicación de memoria.

3.1.2 Las variables pueden ser del mismo tipo (en cuyo caso la equivalencia es matemática) o de distinto tipo, en cuyo caso la equivalencia se reduce a compartir memoria.

3.1.3 Los a_i pueden ser variables escalares o elementos de un arreglo.

Ejemplo 1:

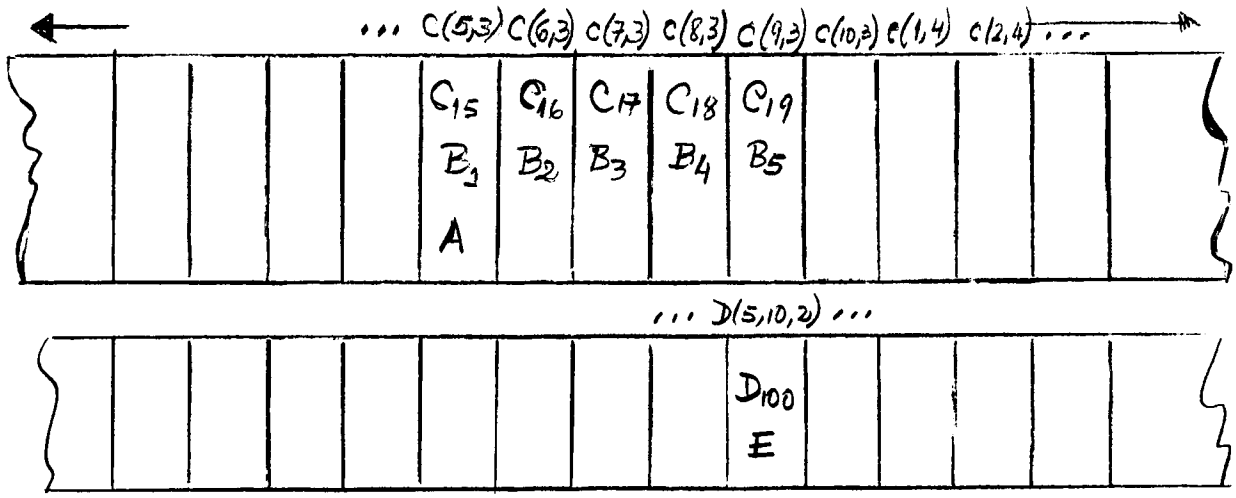
```
DIMENSION    MAT1(10,10) , MAT2 (5,3) , IVEC (10)
EQUIVALENCE  (MAT1 (1) , MAT2 (1) ) , (MAT1 (16) , IVEC (1)
```



En algún momento del programa se deja de usar MAT1.

Ejemplo 2:

```
DIMENSION  B(5) , C(10,10) , D(5,10,15)
EQUIVALENCE (A,B(1), C(5,3) ) , (D (5,10,2) , E).
```



Nota: escribir

EQUIVALENCE (B (2) , C (8,3)) es contradictorio ya que B (2) C(6, 3)

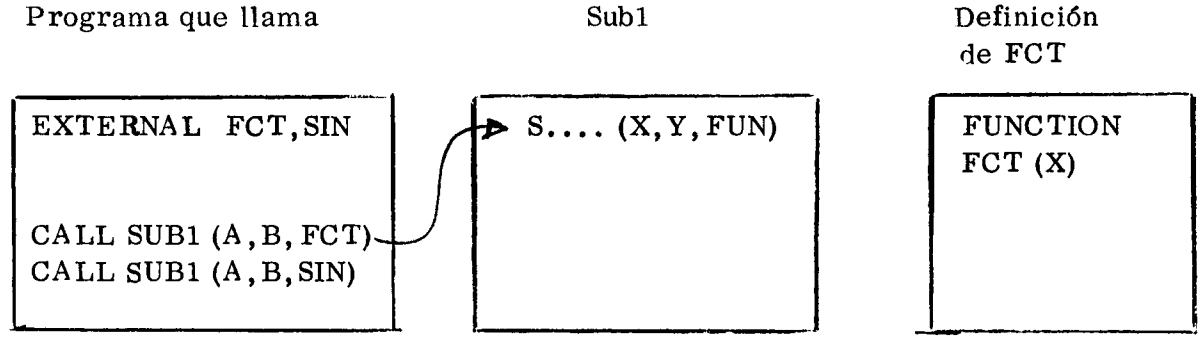
Ejemplo 3:

```
COMMON    A,B,C
DIMENSION X (5)
EQUIVALENCE (B,  X(1) )          EQUIVALENCE  (B, X(3) )
A                                             X1
B              X1                A            X2
C              X2                B            X3
                 X3                           X4
extensión        X4                           X5
                 X5
```

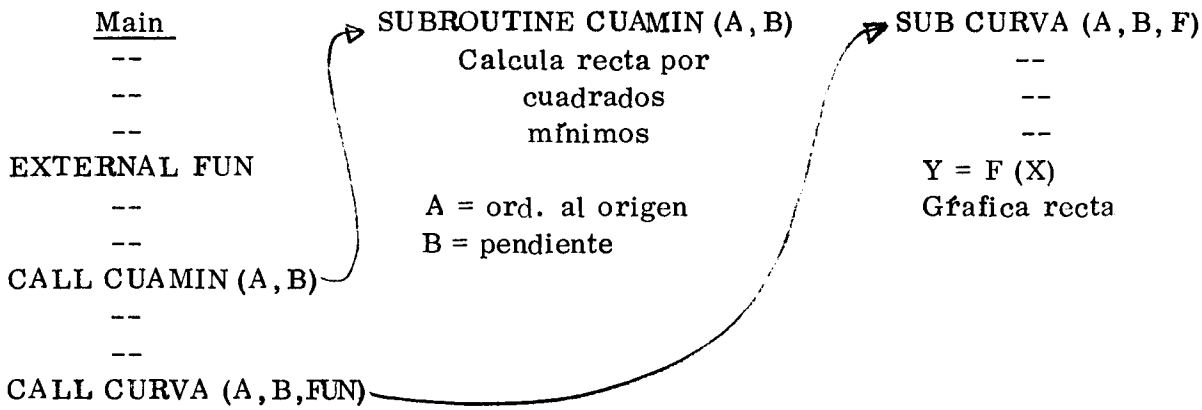
NO

4. External

Cuando una llamada a un subprograma utiliza dentro de la lista de argumentos el nombre de otro subprograma (ya sea subrutina o función) este nombre debe ser definido dentro del programa donde está el CALL como de tipo external.



Ejemplo:




```
FUNCTION FUN (A ,B,X)
FUN = A+B*X
RETURN
END
```

Es la última de las no ejecutables (incluso va después de la DATA).

5. Entry

La sentencia Entry permite entrar a un subprograma en puntos distintos del punto inicial de entrada. Es propia del FORTRAN IV (IBM).

5.1 Forma: ENTRY nombre $(a_1 \ldots . a_n)$

a_i : argumentos (pueden no coincidir en número con los declarados en la sentencia Subroutine pero deben coincidir en nombre con uno al menos de ellos.

1. No puede aparecer dentro del rango de un DO.

2. Cuando la sentencia está en un subprograma de función, o bien el nombre de la función o el nombre de uno de sus puntos de entrada debe recibir un valor.

3. Los argumentos sí deben coincidir en número, tipo y orden con los de su respectiva llamada.

Ejemplo 1:

```
      --                          SUBROUTINE  SUB1 (B,C,D)
      --                                      --
      --                                      --
CALL  SUB1 (X,Y,Z)                            --
      --                          ENTRY   ENT1 (E)
      --                                      --
      --                                      --
CALL  ENT1 (A)                                --
      --                          RETURN
      --                          END
```

Ejemplo 2:

```
      --                          FUNCTION  FUNC1 (A,B,C,D)
      --                                      --
      --                                      --
  A = FUNC1 (X,Y,Z,T)                         --
1 C = FUNC2 (R)                   ENTRY FUNC2 (A)
      --                                      --
      --                                      --
      --                                      --
  AN= EXP (-C)                    FUNC2 = (SIN (A)*B) **C
  GO TO 1                         RETURN
                                  END
```

6. Block Data (subprograma)

Cumple las funciones de la Data pero para inicializar variables que están en Common etiquetado. Es la única manera de inicializar variables en Common etiquetado. No se puede inicializar en Common no rotulados.

6.1 Contiene la sentencia BLOCK DATA, definiciones de tipo, Dimensión, Data, Common y Equivalence.

6.2 No puede tener sentencias ejecutables ni Fonmat ni Subroutines, ni Function, ni Entry.

6.3 Orden de las sentencias:

1) Block Data
2) Implicit (si existe)
3) los Common
4) sentencias que adjudican los valores

6.4 Se pueden definir varios bloques con un BLOCK DATA pero no usar varios BLOCK DATA para un solo bloque.

CAPITULO II

ACCESO A PERIFERICOS

1. Funciones de la computadora

Las funciones que cumple la computadora se pueden dividir en 5 categorías:

1) almacenamiento de información
2) control de información
3) transformación
4) flujo de información (circulación de la información entre las distintas partes del sistema)
5) entrada/salida (relación del sistema con el exterior)

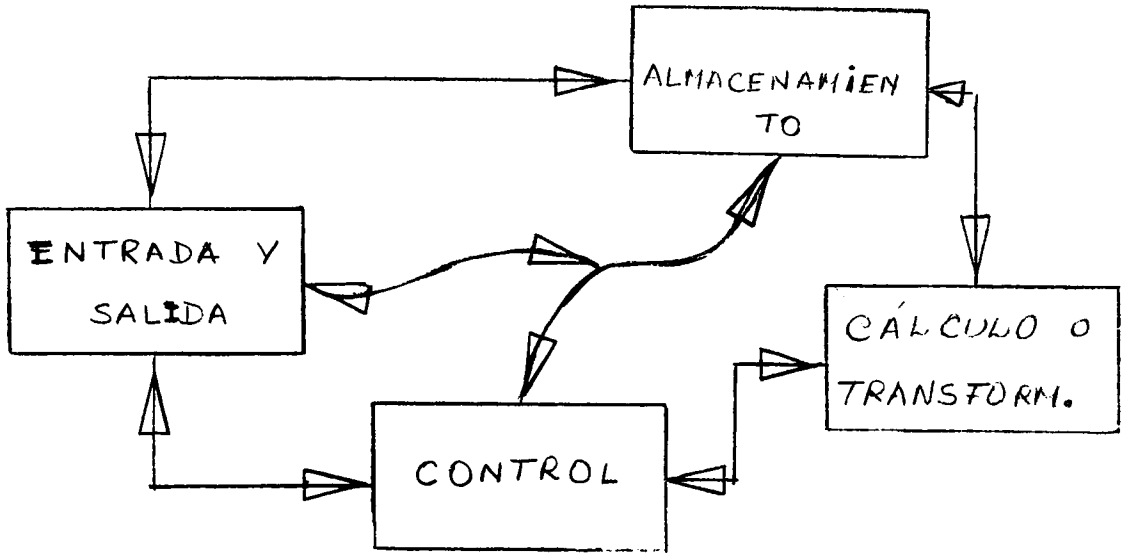
NOTA: es la función de flujo lo que permite que el sistema funcione como tal. Veamos qué elementos cumplen cada una de estas funciones.

1) Función de almacenamiento:
   Se trata de conservar lo que ha llegado al sistema y tenerlo a disposición de los distintos órganos del mismo.
   Memoria central, memorias auxiliares (discos, cintas tambores, etc.)

2) Control:
   a) es responsable de que los algoritmos se ejecuten tal como el usuario los piensa
   b) coordinan los recursos del sistema (internamente)
   La ejerce la UCP(Unidad Central de Procesamiento)

3) Transformación:
   Transforma la información que ya está en el sistema. La ejerce la unidad aritmética-lógica (hace operaciones sobre información almacenada).

4) Flujo de información:
   Determina cómo, cuando y qué información circula dentro del sistema. En general responde a un esquema de este tipo.



5) Entrada-salida: comunicación con el exterior
La ejercen en general los periféricos (lectoras, impresoras, cintas magnéticas, televisores, etc.)

2. Memorias : definición y clasificación

Definición de Von Neuman

Una memoria debe poder cumplir las siguientes condiciones:

i) acceso repetido sin destrucción de la información que contiene
ii) grabación de un dato desplazando lo anterior

Clasificación según su función

i) Memoria Central: componente interno de la computadora
ii) Memorias Auxiliares: son periféricos a los que se accede a través de la computadora. Más lentas. Mayor capacidad.

Clasificación según su acceso

i) Acceso directo: cuando se necesita un dato que está en cualquier lugar de la memoria, el instrumento de lectura se dirige a esa dirección y lo toma (tambor, disco, memoria central).

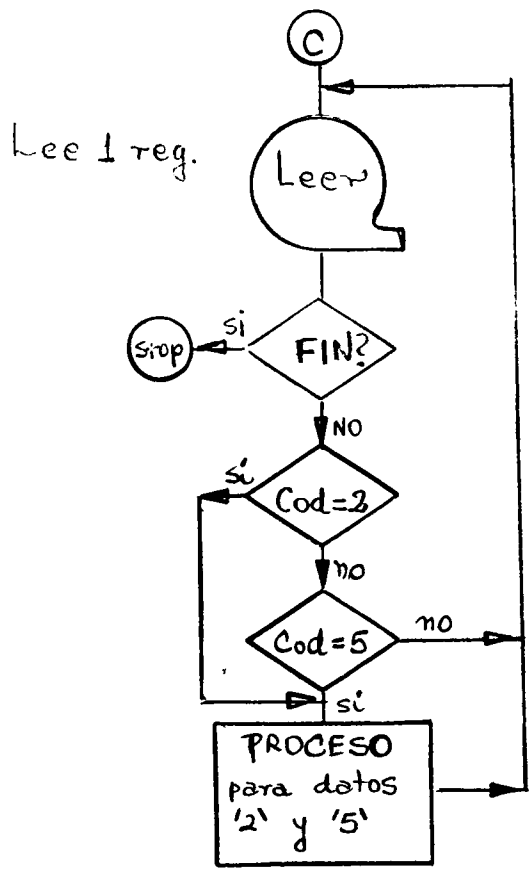
ii) Secuencial: cada vez que se pide un dato aparece el próximo componente de una cierta secuencia. O dicho de otro modo, no se puede tomar un dato sin haber leído todos los anteriores (tarjetas, cinta magnética, cinta de papel)

iii) Asociativa:

Se busca un dato cuya porción inicial o identificador se conoce , y se necesita el resto (discos magnéticos con búsqueda por identificadores, cinta magnética con subrutinas de lectura).

Ejemplo de cinta magnética con subrutina de lectura (o sea memoria asociativa):

Se tienen grabados en cinta, registros cuyo primer carácter es un código que puede valer 1, 2, 3, 4 ó 5, según sea el tipo de información que contiene el registro. Para un cierto proceso se necesitan los datos del tipo '2' y '5'. Entonces la cinta, como archivo, se convierte en memoria asociativa, si se la considera en conjunto con la siguiente subrutina:

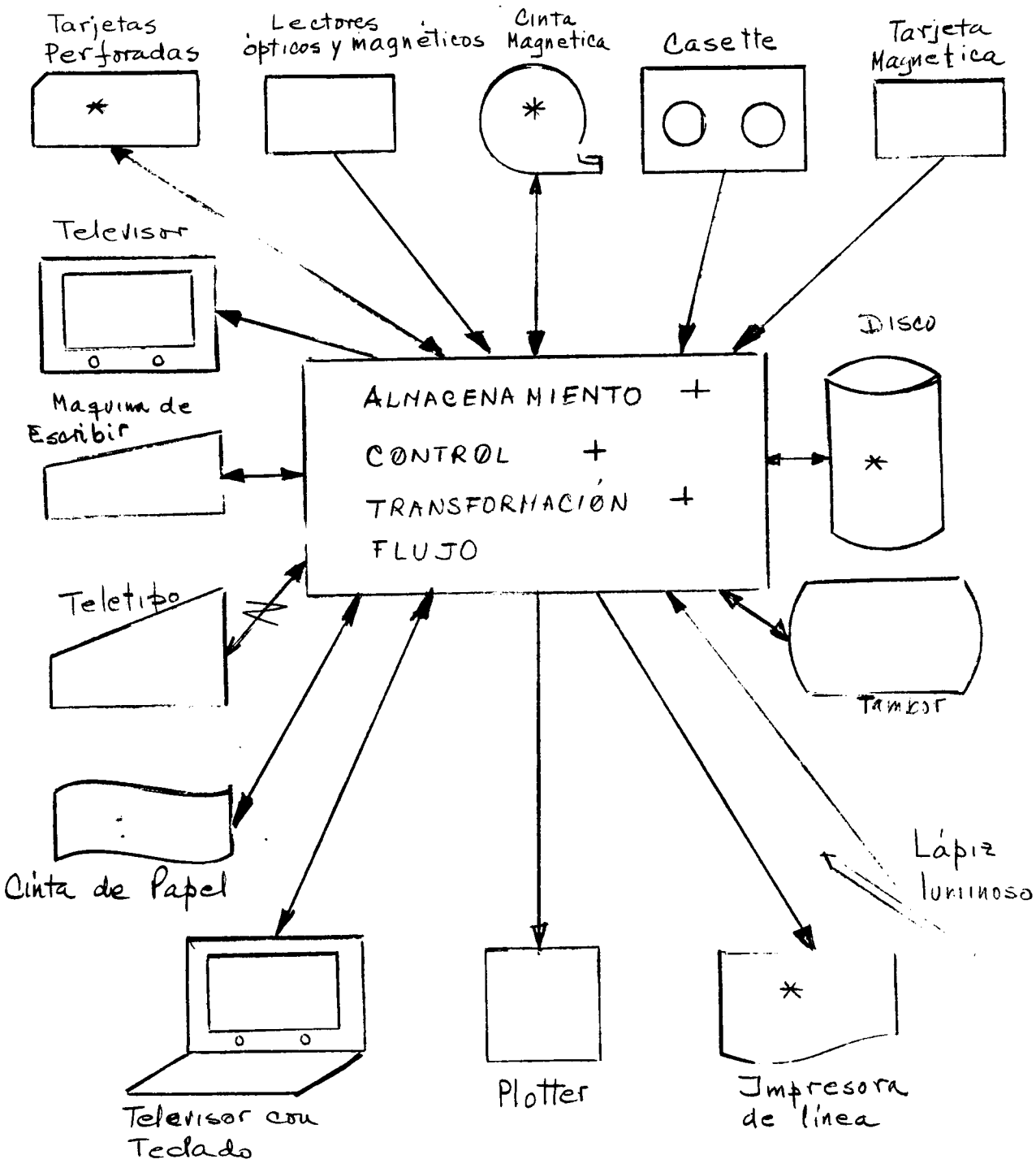


3. Periféricos

a) Definición y generalidades

Es un elemento de la computadora que cumple las siguientes condiciones:

i) Introduce información
ii) Almacena información temporariamente para que luego sea emitida o usada.



Esquema general de la computadora y periféricos

i) Los periféricos que aparecen con * serán explicados en detalle.

ii) Observar el (o los) sentido(s) de las flechas:

- cuando apuntan sólo al periférico — que es sólo de salida, no se pueden ingresar datos
- cuando apuntan sólo al computador — que éste no puede crear un data set para ese periférico

- Los lectores ópticos y magnéticos
- El lápiz luminoso
- El teléfono
- La máquina de escribir
- La teletipo
- El televisor con teclado

} Sirven para entrar datos u órdenes en forma inmediata

- El teléfono
- El televisor

} Reciben información inmediata y legible sin necesidad de dejar documentos escritos

- Máquina de escribir
- Teletipo
- Plotter
- Impresora

} Dejan un documento escrito

- Cinta de papel: para registrar información y mandarla a distancia (más barata que cinta magnética, mejor que el telégrafo).

- Cintas magnéticas
- Tarjetas
- Cassettes

} En entrada: se preparan de antemano y se procesa en lote

- Tambor: es muy caro y muy rápido } se usa sólo para programas y no para datos

- Tarjetas magnéticas: muy lentas, se utilizan para contabilidad del equipo.

b) Descripción detallada

1. Lectora de tarjetas

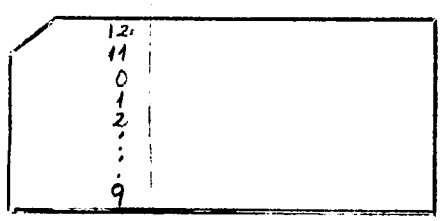


Fig. 1

La tarjeta está compuesta por 80 columnas y 12 filas. Estas últimas numeradas del modo indicado por la figura 1.

El dispositivo de lectura está formado por 2 filas de 80 células fotoeléctricas cada una. Una fila realiza la lectura fila a fila en el siguiente orden: 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 11, 12., y la otra fila de células realiza la verificación. Entre el bolsillo previo a la lectura y el bolsillo de caída hay rodillos que hacen avanzar la tarjeta.

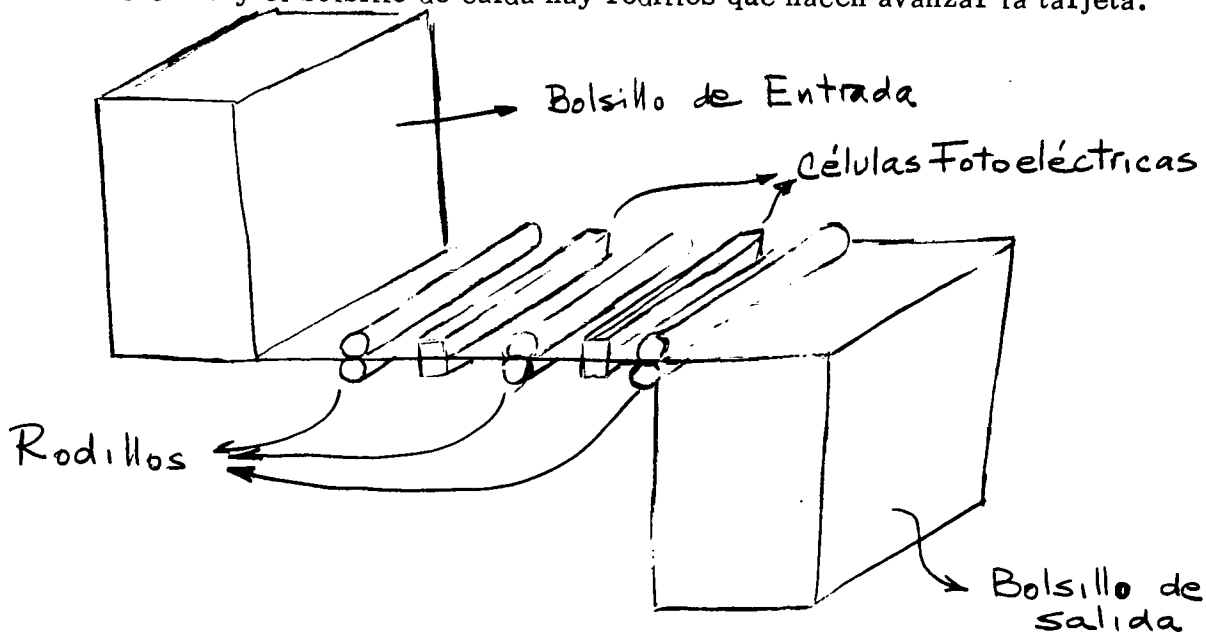


Diagrama de la lectora de tarjetas

Fig. 2

La tarjeta comienza su movimiento desde el bolsillo de entrada, por medio de la puesta en marcha de otro rodillo con una uña que se pone en movimiento cuando se ejecuta la orden de lectura.

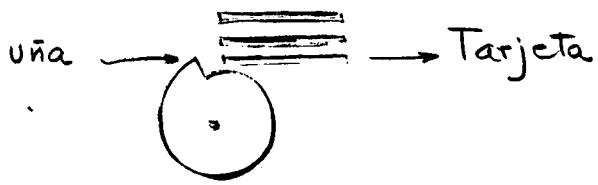


Fig. 3

La posición de la uña en el momento de emitirse la orden de lectura puede ser cualquiera alrededor del eje, de modo que, entre la orden de lectura y el momento de salida de la tarjeta del bolsillo (momento de arranque) transcurre una cantidad de tiempo que no es fija, pero tiene límites entre 0 (si la uña está junto a la tarjeta) y 75 milisegundos (si la uña tiene que dar toda la vuelta al eje)

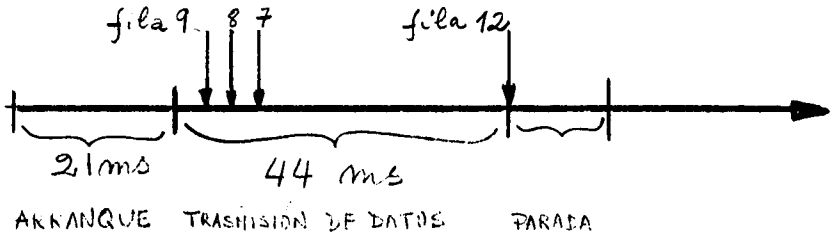


Fig. 4

Los archivos de tarjetas se utilizan para:

* Programas
* Lotes de datos de prueba

2. Impresora de línea

La impresora de línea puede dar una longitud de 120 ó 132 caracteres en formularios standard, o la que fije el programa para formularios especiales. Puede funcionar a tambor o a cadena. Cada posición tiene un martillo. Se arma la línea y se larga con la información de salida.

TIEMPOS INVOLUCRADOS
- Salto de papel
- Transmisión de datos
- Impresión propiamente dicha

Son todos tiempos mecánicos.

En adelante conviene pensar los tiempos en los siguientes términos para poder comparar:

Mecánicos _____ del orden del milisegundo
Electromecánicos _____ microsegundos
Electromagnéticos _____ nanosegundos

3. Cinta magnética

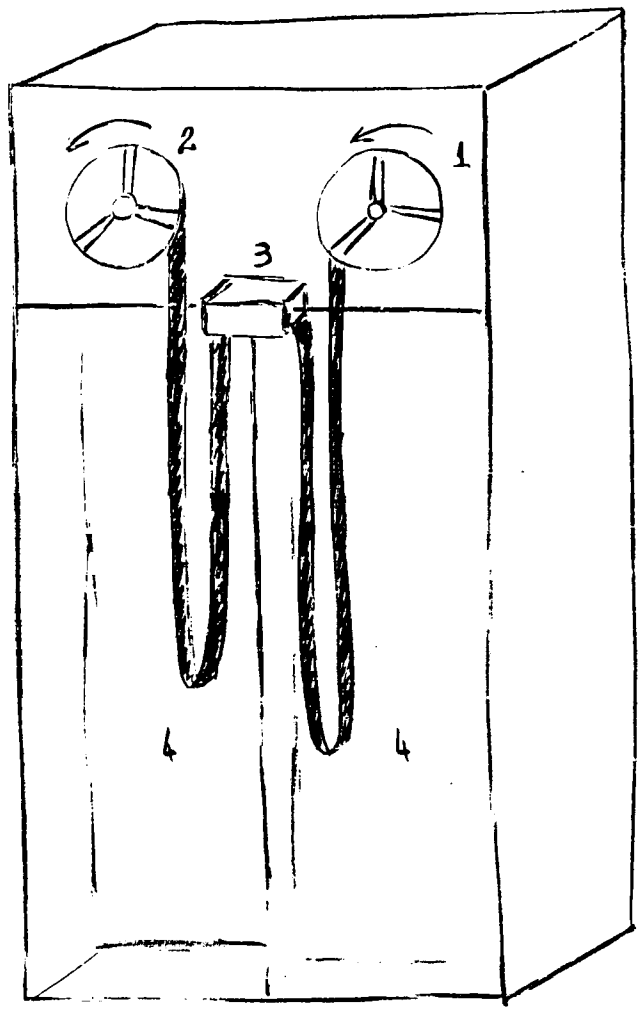
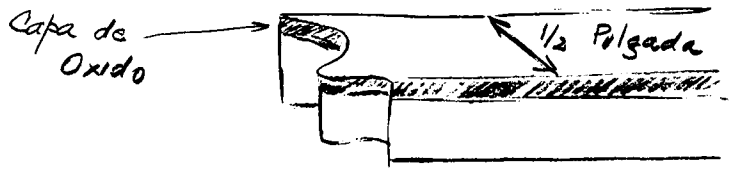


Fig. 5

1: Carrete de entrada (para grabar o leer)
2: Carrete de salida (cinta grabada o leída)
3: Cabeza lectora/grabadora
4: Dispositivos de vacío para no estirar la cinta



La información está sobre una cinta de plástico con una capa de óxido magnetizable sobre la cual se graban puntos magnetizados que tienen el mismo valor que las perforaciones en una tarjeta, pero se pueden borrar.

La cabeza lectora-grabadora (y verificadora) es un electroimán con polos vecinos, debajo del cual se mueve la cinta mientras se graba o lee

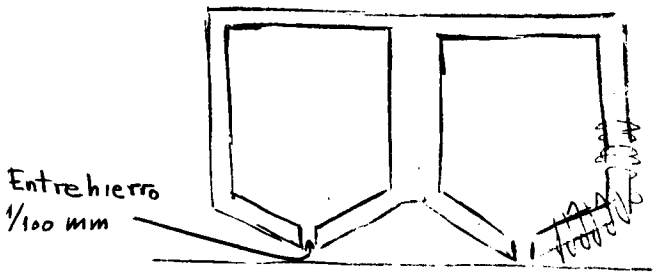


Fig. 6

Cuando cambia la polaridad (se invierte el imán) se produce una discontinuidad que se puede detectar y éstos son los bits '1'. De modo que, al grabar, cada '1' se traducirá en un cambio de polaridad, mientras que los ceros serán mantenerla como está.

Al leer, la cabeza recibe esos cambios de polaridad como '1'.

Ejemplo:

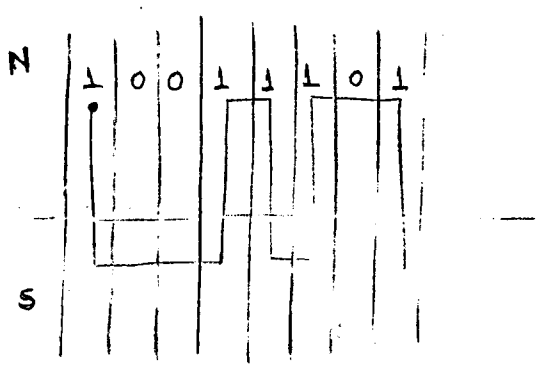


Fig. 7

Sobre el ancho de la cinta, hay 7 ó 9 cabezas que trabajan simultáneamente. Cuando el sistema es 8 bits = 1 byte se utilizan 9 cabezas, si el sistema es 6 bits = 1 byte se utilizan 7 cabezas.

El recorrido de 1 cabeza se llama canal y es así que se habla de 7 6 9 canales (o pistas).

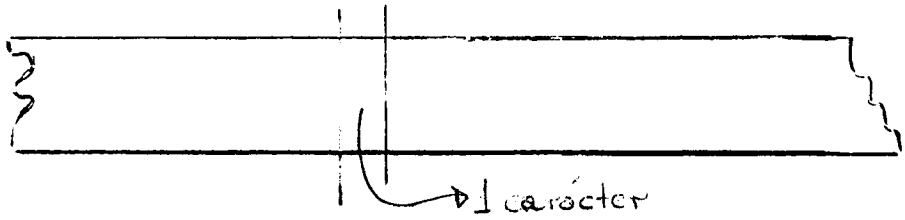


Fig. 8

El bit de "exceso" es el llamado bit de paridad,que se graba con cada carácter, y luego sirve de verificación. Se utiliza debido al manipuleo al que es sometido este tipo de archivos. Al grabar,se calcula la paridad y se graba; al leer, se calcula la paridad y se compara con la grabada,que debe ser igual. Cuando falla, se da un tipo de error de lectura.

Cuando se va a leer, la cinta se pone en movimiento y la información está disponible en el momento en que se adquiere la "velocidad de régimen". De igual modo, ese es también el momento en que se puede comenzar a grabar.

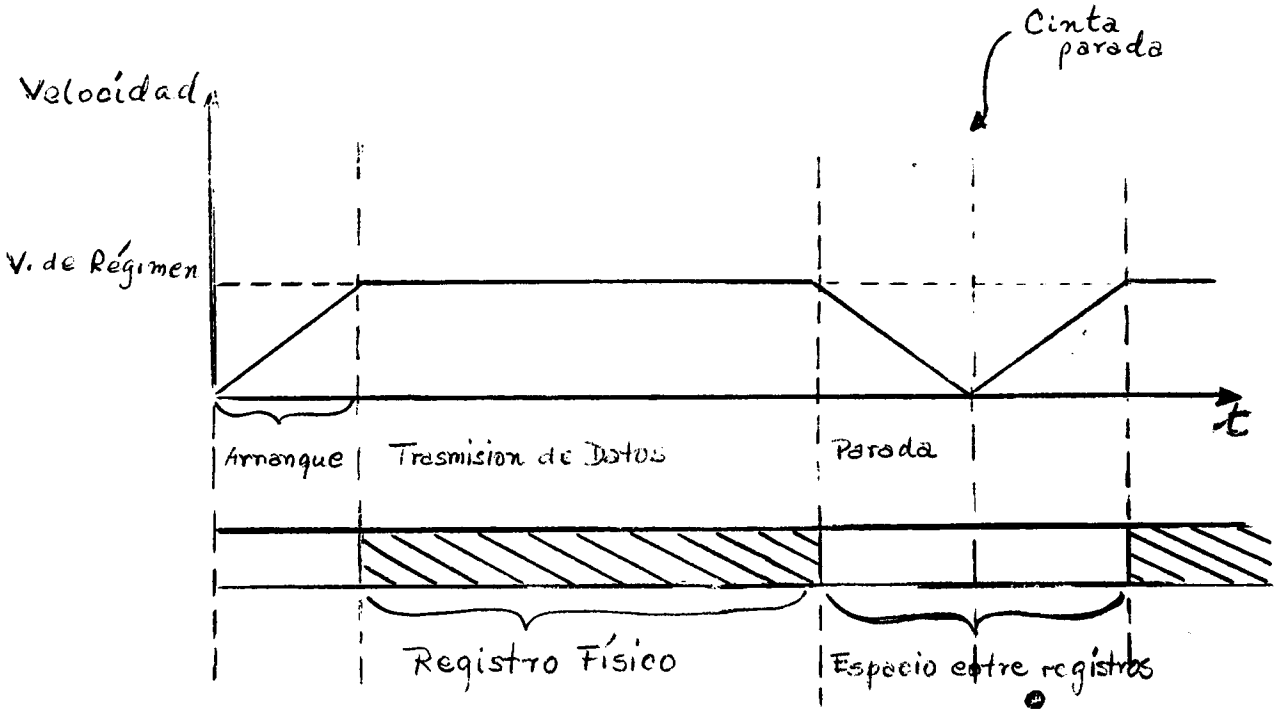


Fig. 9

Al determinar el tamaño de los registros físicos se tendrá en cuenta este gráfico para evitar que la cinta esté "prácticamente vacía". No se puede leer y grabar en una misma cinta cuando hay que modificar registros. En general, funciona del siguiente modo

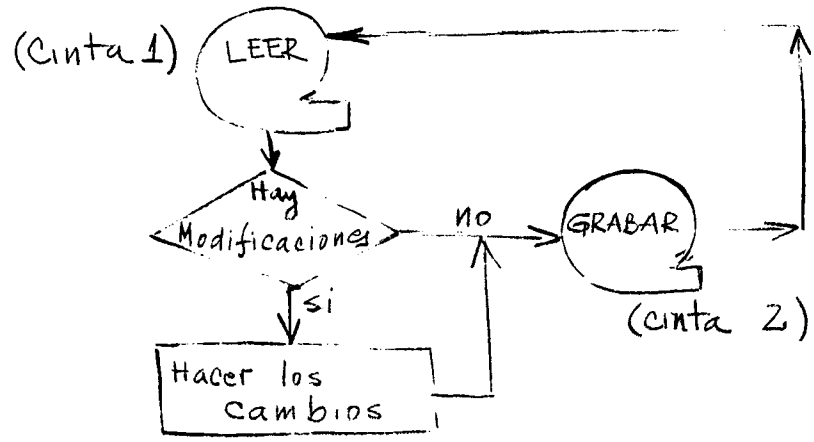


Fig. 10

Nomenclatura:

Volumen: es el identificador del carrete, por ejemplo TA0588
(VOL=SER=) VOL = SER = TA0588

Data Set Name (DSN): PINVIE, es el nombre del archivo que interesa. En un mismo volumen puede haber varios DS. Un mismo DS puede ocupar varios volúmenes. El usuario debe tener control de este tipo de detalles.

Registros físicos y registros lógicos:

Se llama registro lógico a la unidad de trabajo del programa.

Ejemplo:	código de obs.	1 car	El conjunto es el registro de trabajo, sobre el cual se opera
	año de la obs.	2 car	
	mes	2 car	
	día	2 car	
	hora	2 car	
	resultado	7 car	
		16 car	

Se llama registro físico a la unidad de trabajo de la computadora. Está compuesto por un número de registros lógicos a determinar por el programador. Cuando se da la primera orden de lectura, se lee todo un registro físico o bloque y se traslada el primer registro lógico al área de lectura del programa.

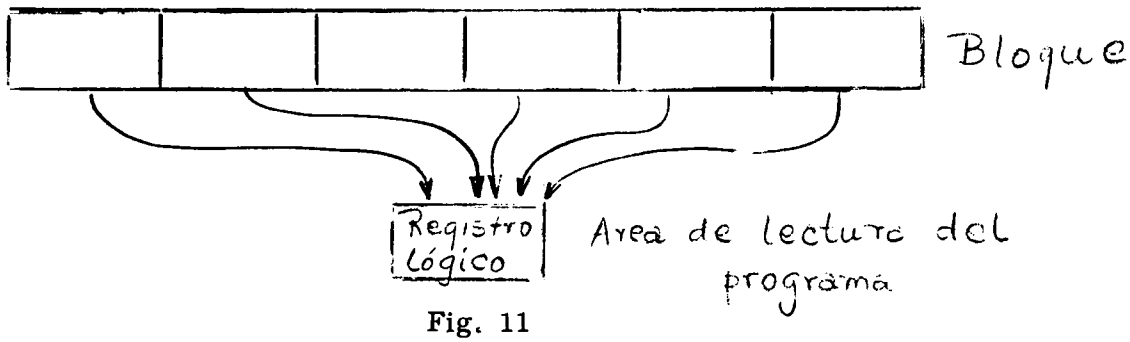


Fig. 11

Se sigue "leyendo" del área de trabajo y cuando se acaban los registros del área de trabajo, con la próxima lectura se repite el procedimiento de la 1a. lectura y así hasta que se llega al fin del archivo.

Hay que considerar que el área de trabajo está en memoria y cuantos más registros lógicos tenga, mayor será la memoria que ocupe el programa.

Capacidad de información de la cinta:

La cantidad de bytes que se graban en un carrete de cinta depende de:

- longitud de cinta: por lo general de 720 metros = 2400 pies
- densidad de grabación: bytes/seg
  200 b/seg, 556 B/seg, 800 b/seg; 1600 b/seg
  las densidades mayores dan mayor velocidad y, a mayor velocidad, mayor detectabilidad de los puntos magnetizados.
- Tamaño del registro físico (bloque)
  el EER es en general de 15 mm. Cuanto mayor sea el tamaño del registro físico menor será el número de EER que se crearán y así se puede aumentar la capacidad de la cinta.

Ejemplo: Se quiere grabar con una densidad de 800 c/seg un archivo de tarjetas
1 registro lógico = 1 tarjeta = 80 caract. (LRECL = 80)
Esta es la unidad lógica de trabajo del programa

$$\frac{80 \text{ car}}{800 \text{ car/seg}} = 0{,}1 \text{ seg} = 2.5 \text{ mm}$$

quiero grabar 50 tarjetas / registro físico

1 registro físico = 50 tarjetas x 80 car/tarj. = 4000 car
(BLKSIZE = 4000)
Esta es la unidad física de trabajo de la computadora
50 x 2.5 mm = 125 mm registro físico

$$+ \frac{15 \text{ mm}}{140 \text{ mm}} \text{ EER}$$

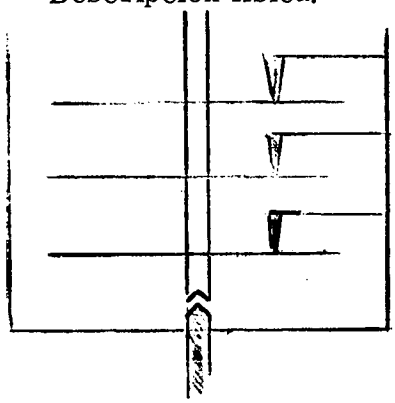
en 720 m da aprox, 5000 registros = 250.000 tarj. =
= 20.000.000 caracteres

Se toma como capacidad media de 15 a 20 millones de caracteres.

Si bien se aumenta la capacidad de cinta con más bloqueo, también se aumenta el área de memoria necesaria para la lectura. Hay que usar un óptimo que es siempre un compromiso entre ambas exigencias.

4. Disco magnético

Descripción física:



Esquema del paquete de discos y peine de cabezas grabadoras (en corte)

Fig. 12

Los mal llamados "discos" son, en realidad, paquetes de discos intercambiables.

Está girando constantemente y la cabeza en cierta posición genera una pista circular.
Las distintas posiciones generan pistas concéntricas, todas de la misma capacidad, con distinta densidad de grabación.

Cada pista está dividida en sectores y cada sector contiene un cierto número de bytes. Esto sirve para indicar con mayor rapidez el lugar de un registro : (No. de cara, No. de pista, No. de sector) .

La información se graba en serie bit a bit . Hay tres etapas de selección en la búsqueda (o ubicación de un registro).

- Selección estática (cuál es la cara que se va a leer) pone en funcionamiento una cabeza por medio de selección de cables (del orden del nanoseg).
- Selección mecánica (cuál es la pista que se va a leer) hace correr el brazo hasta la pista que interesa por medio de un movimiento mecánico (del orden del miliseg).
- Selección dinámica (cuál es el registro que interesa) abre las compuertas para dejar pasar la información cuando está disponible, o sea, pasando bajo la cabeza, el registro buscado (del orden del nanoseg).

Capacidad de disco aproximada

200 pistas/cara
20 caras/paquete
10 sectores/pista
700 bytes/sector

} 7 a 30 millones de caracteres por unidad

Recordar que se graba bit a bit (en serie) mientras que en cinta se graba en paralelo byte a byte.

2314 vs 3330 (modelos de IBM)

	2314	3330
No. bytes/pista	7294	13030
No. pistas/cilindro	20	19
No. cilindros/pack	200	404
No. bytes/pack	29 Mega	100 Mega
vel. transf. (KB/seg)	312	806
rotación (mseg)	25	16.7

c) Ventajas y desventajas de cada periférico.

Atributo en estudio	Tarjeta	Cinta	Disco
Organización	Secuencial	Secuencial Asociativa	Secuencial Acceso libre Asociativa
Costo	Es el más barato	Costo intermedio, relativo al uso	El más caro en costo absoluto, pero también es relativo al uso
Velocidad	Totalmente mecánico, es el más lento de los tres	Velocidad intermedia; Transferencia: 30, 60, 120 kb/seg. Velocidad de cinta: 18, 37½, 75, 112 "/seg	El más rápido de los tres Transferencia: 300-800 kb/seg Rotación: 27-17 miliseg
Capacidad	Ilimitadamente grande pero proporcionalmente más lenta	Mucha capacidad de acuerdo a la densidad y al bloqueo con el límite de memoria que se vió	Gran capacidad hasta 100 Megab/pack Standard 30 Megabytes
Uso en general	Lotes de prueba. Datos reales hasta poner operativo el sistema. Siempre que haya datos que cambien, Programas.	Archivo de programas Archivo de Datos (Mestros) Imágenes de disco. Soporte intermedio de información. etc.	Todo tipo de archivo Bibliotecas de programas de uso frecuente Imagen de impresión etc.
Seguridad del soporte	Muy poca debido a: -gran manipuleo -destrucción o desgaste -caída o pérdida del lote	Se puede cortar, ensuciar, etc. Muy segura en cuanto a destrucción por operador ya que se protege con aro	Muy seguro en cuanto a manipuleo, pero se puede destruir con órdenes de operador. Se protege haciendo imagen en cinta y con algunos parámetros

d) Uso de cinta o disco

Para emplear cinta o disco magnético es necesario definir ciertos elementos de manera precisa.

Se debe describir el archivo que se usará definiendo parámetros que:

i) lo identifiquen unívocamente (describiendo sus rótulos, p.ej.)
ii) le asignen un soporte físico
iii) le asignen un volumen determinado
iv) describan sus atributos físicos
v) indiquen que uso se dará al archivo
vi) indiquen la forma de lectura o grabación

Se debe ubicar el archivo en relación con un dispositivo de E/S.

Detallando:

. Descripción del ARCHIVO

i) Identificación:(Parámetro:DSN)
Se debe decir si el archivo tiene un nombre particular o no. Si lo tiene se pone DSN = nombre
Ejemplo: archivo que se llama ARCH1
// DSN = ARCH1...
Este nombre es una marca grabada magnéticamente al comienzo del archivo.

ii) Asignación de soporte físico (Parámetro UNIT)
Se debe indicar que tipo de soporte se usará (cinta o disco)
Ejemplo:
// UNIT = TAPE... (cinta de 1600 bpi)
// UNIT = TAPE 800... (cinta de 800 bpi)
// UNIT = 2314...(discos 2314)
// UNIT = 3330... (discos 3330)
// UNIT = SYSDA... (sistema de acceso directo)

iii) Asignación de volumen (Parámetro: VOL)
Se debe indicar que número de volumen se usará
Ejemplo:
// VOL = SER = T0021...

iv) Descripción de atributos físicos
En el caso de cintas se deberá indicar si admite 800 bpi o 1600 bpi (subparámetro DEN dentro del parámetro DCB)

v) Utilización del archivo (Parámetro : DISP)
- Se debe decir si se lee o se graba el archivo, o sea: si se usa información vieja ya grabada o se incorpora información nueva

DISP = OLD,...
NEW,...

- Se debe decir si el archivo será usado en otro paso del trabajo en curso o no; si debe ser borrado al terminar el trabajo o no; si debe ser conservado al terminar el trabajo.

DISP = (OLD, PASS) ——→ se lo usa en otro paso
(, DELETE) ——→ se lo borra
(NEW, KEEP) ——→ se lo conserva

vi) Forma de lectura o grabación (Parámetro: DCB)
Se debe precisar siempre cuál es la forma de lectura o escritura de los datos del archivo, indicando:

- longitud del bloque físico (subparámetro BLKSIZE)
- longitud del registro lógico (subparámetro LRECL)
- densidad de grabación SOLO PARA CINTA (subparámetro DEN)

DEN = 3 1600 bpi
2 800 bpi

- modo de grabación dentro del bloque (subparámetro RECFM)

Ejemplo:

```
//        DCB = (RECFM = FB, BLKSIZE = 7200, LRECL = 80,
                 DEN = 3 ).
```

. Ubicación del archivo en relación con dispositivos de E/S

i) En el nivel lógico o simbólico: se debe fijar un No. lógico para designar el archivo que se leerá o escribirá

Ejemplo:

```
          WRITE (11, 25) A

     25   FORMAT (....
```

11 es el nombre lógico de la unidad (de salida en este caso)

ii) En el nivel de ejecución: se debe indicar con precisión la unidad física a la cual se hace referencia con el número lógico. Esto se logra definiendo para la etapa de ejecución que el número simbólico dado significa tal unidad física

Ejemplo:

```
// GO.FT11F001   DSN = ARCH1, DISP = (NEW, KEEP),
//               UNIT = TAPE, VOL = SER = T00066,
//               DCB=(RECFM=FB,BLKSIZE=3200,LREC=80,DEN=3)
```

CAPITULO III

SISTEMA OPERATIVO

1. Introducción a la noción de Sistema Operativo

El trayecto que cumple dentro de la máquina el "trabajo usuario" es supervisado por un conjunto de programas que provee el fabricante (en general). Este conjunto de programas "supervisores" se llama Sistema Operativo. Podemos decir que el S.O. proporciona la interfase entre el usuario y la computadora.

Para que en lo sucesivo no haya confusiones en la terminología daremos algunas definiciones.

Definición 1: TRABAJO

Llamaremos TRABAJO a la unidad básica independiente de trabajo que se presenta a la máquina (o sea, la unidad que no está conectada con otras ni depende de otras). En la jerga de computación : "job".

Definición 2: SISTEMA OPERATIVO

Es un conjunto de datos y procedimientos * necesarios para:

1) administrar el sistema, o sea:
   - saber quién es quién dentro del sistema (para que ubique sus diferentes partes constitutivas: periféricos, programas, etc.)
   - cuánto usa cada actividad de cada recurso.

2) asignar los recursos de la computadora:
   - memoria central
   - archivos
   - tiempo del procesador central
   - tiempo de la UAYL, etc.

3) mantener el buen funcionamiento del sistema y no perder información:
   - ubicar errores
   - garantizar conservación de cierta información cuando se corta el proceso, etc.

---

* Datos y procedimientos: entiéndese por tales a: rutinas, métodos de uso, programas, manuales de operación, etc.

2. Papel del Sistema Operativo dentro del sistema

Los Sistemas Operativos fueron desarrollados como respuesta a dos necesidades :

a) aumentar el rendimiento de la UCP
b) aumentar el rendimiento de periféricos

3. Idea somera sobre el funcionamiento del SO

3.1 Idea esencial

Todo programa está "supervisado" por el Sistema Operativo pues todo programa delega al sistema operaciones cuya ejecución por él mismo sería riesgosa (acarrearía frecuentemente errores fatales). Esto exige dos niveles de operación:

Nivel 1: normal = algunas órdenes están prohibidas (programas usuarios)
Nivel 2: privilegiado = todas las órdenes están permitidas (programas supervisores)

La exigencia de que la computadora no se detenga nunca implica que cualquier instrucción normal de parada o error que provocaría detenciones del procesador en sistemas simples, provocan (cuando hay S.O.) una transferencia del control a un programa del nivel superior.

3.2 Funciones primordiales del Sistema Operativo

1) automatizar la corriente de trabajos (antes se hacía a mano, con llaves y palancas).

2) decidir en cada momento cómo manejar los recursos disponibles en la escala de tiempos de la máquina.

3) organizar parte del almacenamiento de discos.

4) encargarse de algunas funciones de E/S.

Definición 3: PASOS (de un trabajo)

Un trabajo puede estar formado por distintos pasos o etapas (steps), tales como: compilar, vincular, ejecutar, copiar archivo, imprimir archivo, etc.

4. Qué se debe pedir de un Sistema Operativo

A. Respecto del manejo de trabajos (job management)

i) permitir al usuario cualquier número de operaciones como "pasos".
ii) organizar el pasaje de información de un paso a otro.
iii) hacer que la ejecución de ciertos pasos o etapas dependa del fin correcto de otro (Ejemplo: ejecución y compilación).

B. Respecto del manejo de datos (data management)

iv) controlar periféricos
v) proveer medios para almacenar y recuperar información sin que sea necesario especificar el soporte (Ejemplo: salida del compilador).
vi) proteger información archivada.
vii) secuenciar el trabajo interno (en este momento el procesador se dedicará a tal cosa, etc.)

5. Constitución del Sistema Operativo

Tres tipos de programas coexisten en una máquina:

A progrmas de control
B programas de servicio
C programas usuarios

A. Programas de control

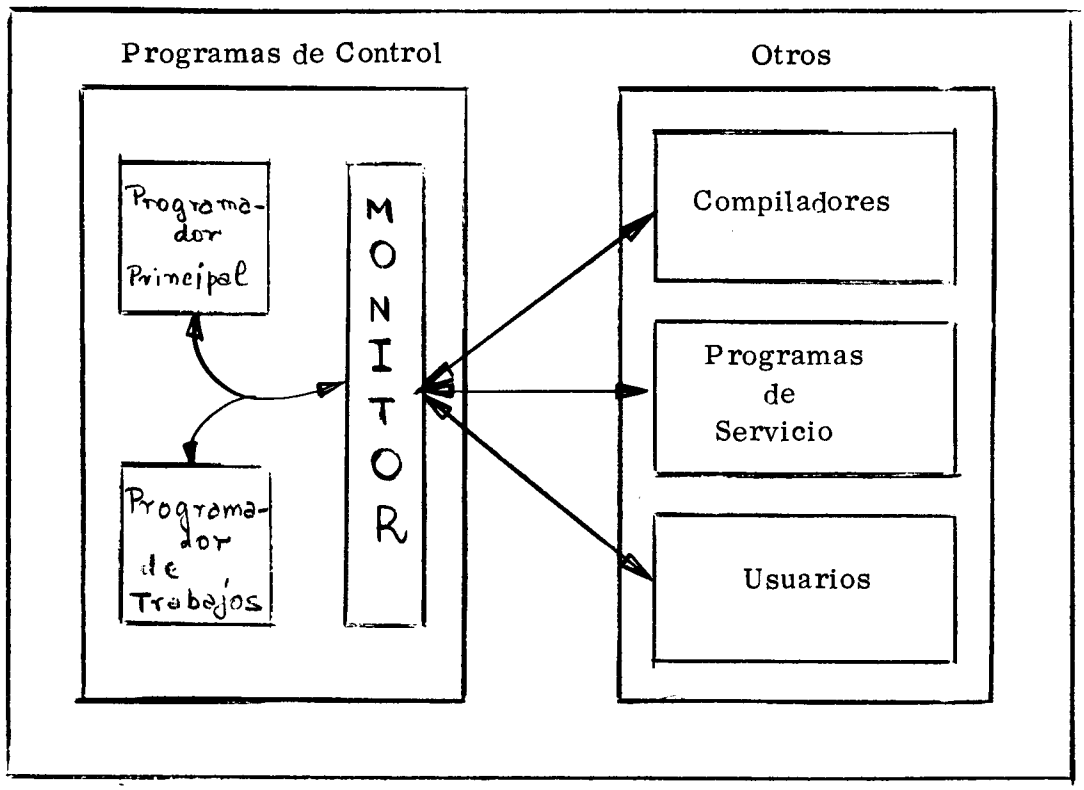
Sirven para:

a) determinar el orden en que serán ejecutados todos los otros programas (de servicio y usuarios)
b) administrar los recursos que son pedidos en cada instante.

B. Programas de Servicio

Compaginadores
Editor de vínculos
Utilitarios

Esquema general de funcionamiento



6. Trayectoria de un trabajo dentro del sistema

6.1 Comunicación con el Sistema Operativo

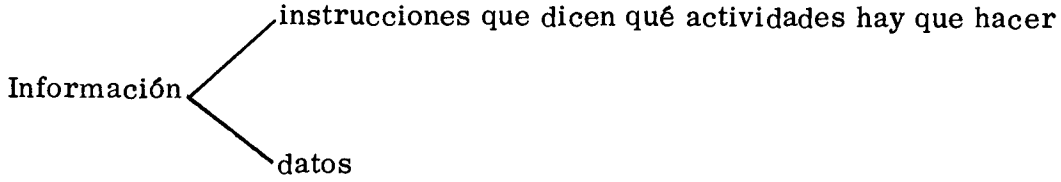
El usuario y el operador se comunican con el Servicio Operativo mediante instrucciones de control. El conjunto de estas posibles instrucciones constituye el J.C.L. (job control language) o Lenguaje de Control de Trabajos.

6.2 Etapas en el tratamiento de un trabajo

Un trabajo comprende, en general:

- uno o varios programas (con o sin subprogramas)
- datos

Etapa 1: ¿Cuál es la información que se brinda al equipo ? Es información de dos tipos diferentes:



Ahora bien, el equipo no sabe interpretar esta información, resulta así necesario traducirla a un código de ceros y unos que sí pueda entender. Esta "traducción" la hace la propia máquina por medio de programas Compiladores y Compaginadores que suministran, luego de actuar, una versión de las instrucciones fuente en lenguaje de máquina, directamente interpretable por la computadora.

Hay muchos compiladores: FORTRAN, COBOL, PL1, ALGOL, etc. Nos ocuparemos de aquí en más de trabajos en FORTRAN o Assembler.

Resumiendo: Un programa escrito en FORTRAN o Assembler es "traducido" a lenguaje de máquina por el Compilador FORTRAN o por el Compaginador. Y hay que destacar esta noción: las tarjetas fuente escritas en FORTRAN (Assembler) son tarjetas-dato (cumplen función de datos) para el programa Compilador FORTRAN (Compaginador).

Definición 4: MODULO FUENTE

Un conjunto de instrucciones escritas en FORTRAN (Assembler) se denomina MODULO FUENTE.

Definición 5: MODULO OBJETO

El conjunto de instrucciones de máquina resultante de una traducción (compilación o compaginación) se llama MODULO OBJETO. Puede ser colocado en cinta, disco, tambores o en tarjetas (= DECK o "binario").

Etapa 2: Como cada programa es traducido por separado (son considerados cada uno como una unidad). Se hace necesario entonces establecer los vínculos entre ese programa recién traducido que sale de un compilador o compaginador y otros programas, que pueden ser subprogramas del mismo, subrutinas que ya están residiendo en memoria, etc. Es necesario "vincular" los diferentes módulos objeto que deben funcionar conjuntamente en cada caso. De esta tarea se encarga un programa de servicio: el Editor de Vínculos (Linkage Editor). Después de haberse resuelto las conexiones se tiene un módulo de carga.

Definición 6: MODULO DE CARGA

La unidad que puede ser cargada directamente en memoria central para ser ejecutada se denomina MODULO DE CARGA.

BIBLIOGRAFIA

1. Apuntes de clase del Ing. Gustavo POLLITZER.
Facultad de Ciencias Exactas y Naturales (UBA), 1972.

2. Apuntes de clase del Lic. Mauricio MILLSCHBERG.
Facultad de Ciencias Exactas y Naturales (UBA), 1973.

3. IBM/360, Principios de Operación.

4. IBM Systems Reference Library.
System/360: Concepts and Facilities.

5. IBM Systems Reference Library.
System 360/370: Fortran Language.

6. IBM Systems Reference Library.
Fortran Programmer's Guide.

7. IBM Systems Reference Library.
OS/VS: Linkage Editor and Loader.

8. IBM Systems Reference Library.
OS/VS: Utilities

9. IBM Systems Reference Library.
OS/360: Job control Language Reference.

10. IBM Systems Reference Library.
OS/360: Supervisor and Data Management Services.

11. BULL G.E.
Fortran IV

12. Grupo de Desarrollo en Técnicas de Computación, AERE, Harwell, UKAEA
Fortran Optimization Techniques.

Soportes de aplicación:
Acceso directo
Cinta

IEHLIST

Tipo: del sistema

Acciones: . Lista catálogos o el directorio de uno o más archivos particionados, dando la cantidad de pistas y/o cilindros vacíos de cada archivo y del volumen entero

Soportes de aplicación: Acceso directo.

AGRADECIMIENTOS

Queremos expresar nuestro reconocimiento a la Dra. Fanny Dyment por el aliento inicial dado para la realización del curso y de la presente publicación. También deseamos agradecer la eficiente colaboración de la Sra. Dolores Zembel de Maluvini en la pesada tarea de descifrar y mecanografiar los originales.

Uno de los autores desea agradecer la inapreciable generosidad de Teresita y Héctor Schilman, del Dto. de Reactores (CNEA) a quienes debe, en gran medida, sus conocimientos sobre muchos de los temas expuestos.

IEHMOVE

Tipo: de sistema

Acciones:
- Mueve o copia
  - archivo que reside en hasta 5 volúmenes
  - un grupo de archivos catalogados
  - un catálogo o partes de un catálogo
  - un volumen de archivos
- Intercala miembros de dos o más particionados
- Incluye o excluye miembros elegidos
- Renombra miembros copiados o movidos
- Reemplaza miembros
- Incluye o excluye archivos en una operación de mover o copiar

Soportes de aplicación:
- Disco a Disco
- en casos particulares: cinta a disco
  disco a cinta

IEHPROGM

Tipo: de sistema.

Acciones:
- Mantiene archivos y modifica sus secciones de control
- Borra archivo o miembro
- Renombra archivo o miembro
- Cataloga o saca del catálogo un archivo
- Conecta o libera un archivo que está en dos volúmenes
- Construye, mantiene o borra el índice del volumen

Soportes de aplicación: Acceso directo - Disco

IEHDASDR

Tipo: de sistema

Acciones:
- Crea backup o copia transportable de un volumen de acceso directo
- Lista el contenido en un sistema de salida
- Copia (dump) de una cinta a un volumen de acceso directo

IEBISAM

Definición 1: Organización secuencial indexada ; organización lógica secuencial y física al azar. Cada registro apunta al siguiente y a veces al anterior.

Tipo: de archivos

Acciones:
- Copia archivo secuencial indexado de un acceso directo a otro
- Reorganiza el secuencial indexado pasándolo a secuencial, para llevarlo a una cinta de un modo no usable directamente, pero que permite recrearlo en disco. Este tipo de archivo no usable directamente se llama unloaded
- Imprime un archivo secuencial indexado

Soportes de aplicación: para un archivo de acceso directo.

IEBPTPCH (OSLISTA)

Tipo: de archivos

Acciones:
- Imprimir o perforar un archivo secuencial, un particionado o una sección elegida del mismo. Cada registro empieza en una nueva tarjeta o línea de impresión
- Imprime o perfora el directorio de un particionado

El OSLISTA, aparte de hacer lo que el IEBPTPCH, elige los registros con intervalos determinados, registro inicial y registro final, pone títulos, etc.

Soportes de aplicación: cualquiera.

IEBUPDTE

Tipo: de archivos.

Acciones:
- Incorpora modificaciones en lenguaje fuente a archivos secuenciales o particionados
- Crea y actualiza bibliotecas en simbólico
- Cambia organización: de secuencial a particionado y viceversa.

Soportes de Aplicación:

Desde	Tarjeta Cinta Acceso directo	a	Tarjeta Impresora Cinta Acceso directo

- Intercala archivos
- Recrea un archivo que agotó el lugar que tenía
- Da lista de bloques de directorio sin usar y el número de pistas sin usar a disposición de nuevos miembros

Soporte de aplicación: Acceso directo.

IEBEDIT

Tipo: de archivos

Acciones:
- Crea archivo de salida que contenga un conjunto elegido de trabajos o pasos de trabajo, a partir de otro archivo secuencial
- Copia un job entero o pasos de trabajo seleccionados

Soportes de aplicación:

	tarjeta	}		{ disco
de	cinta	}	a	{
	disco	}		{ cinta

IEBGENER (OSGENER)

Tipo: de archivos

Acciones:
- Copia un archivo de un soporte a otro
- Genera un archivo a partir de otro
- Copia miembros de particionados
- Crea particionados a partir de secuenciales
- Agrega miembros a un particionado, intercalando en el directorio
- Rebloquea o cambia la longitud del registro lógico

Soportes de aplicación: Todas las combinaciones.

Diferencias entre OSGENER (es un programa escrito especialmente por un analista del CUPED) e IEBGENER (provisto por IBM): Aparte de ser un programa utilitario, el OSGENER permite la realización de tareas tales como pesquisas, acumulaciones, selecciones y varias otras de uso general, con la codificación de unas pocas tarjetas de control.

IBCRCVRP

Tipo: independiente

Acciones: . Recuperación de datos útiles
. Reemplazo de datos malos

Son operaciones sobre pistas defectuosas. Cuando se reemplazan datos malos se hace una intercalación para reconstruir la secuencia correcta.

Cuando se recuperan, se sacan los datos, se los vuelca en cinta y se hace una lista de los registros malos tal como se pudieron leer.

Soportes de aplicación: Acceso directo a cinta.

IEBCOMPR

Tipo: para archivo

Acciones: . Compara dos archivos secuenciales o particionados a nivel de registro lógico para verificar una copia de reserva
- Verifica si el número de registros es igual
- Si los registros son iguales
. Si es particionado compara:
- Directorio igual
- Nombre de particiones igual

Soportes de aplicación: todas las combinaciones posibles

IEBCOPY

Tipo: de archivos

Acciones: . Copia uno o más particionados
. Intercala particionados
. Selecciona o excluye miembros en el proceso de copia
. Crea copias de reserva
. Copia archivos
. Elige archivos para copiar
. Reemplaza miembros con el mismo nombre
. Cambia el nombre de los miembros
. Excluye miembros
. Comprime en el lugar

Los primeros se aplican a volúmenes enteros y realizan acciones y controles generales a nivel del sistema. Los segundos se aplican a los archivos, son los que -en general- usamos nosotros y realizan acciones y controles sobre un archivo que se da como entrada del programa. Los independientes son los que trabajan sin el sistema operativo.

Descripción de los usos de los programas utilitarios

IBCDASDI

Tipo: es un utilitario independiente.

Acciones:
- Inicializa un volumen de acceso directo con control de sus pistas defectuosas:
  - las marca para que no se graben en ellas
  - si se dió la opción de pistas alternativas, se utilizan para reemplazar a las defectuosas cuando se requiera grabar en ellas.
- Escribe rótulo físico del volumen en la pista cero.
- Construye y escribe el índice (o directorio) del volumen (VTOC).

Soportes de aplicación: Acceso directo

IBCDMPRS

Tipo: independiente

Acciones:
- Vuelca el contenido de un volumen de acceso directo
- Restaura ese contenido

Soportes de aplicación:

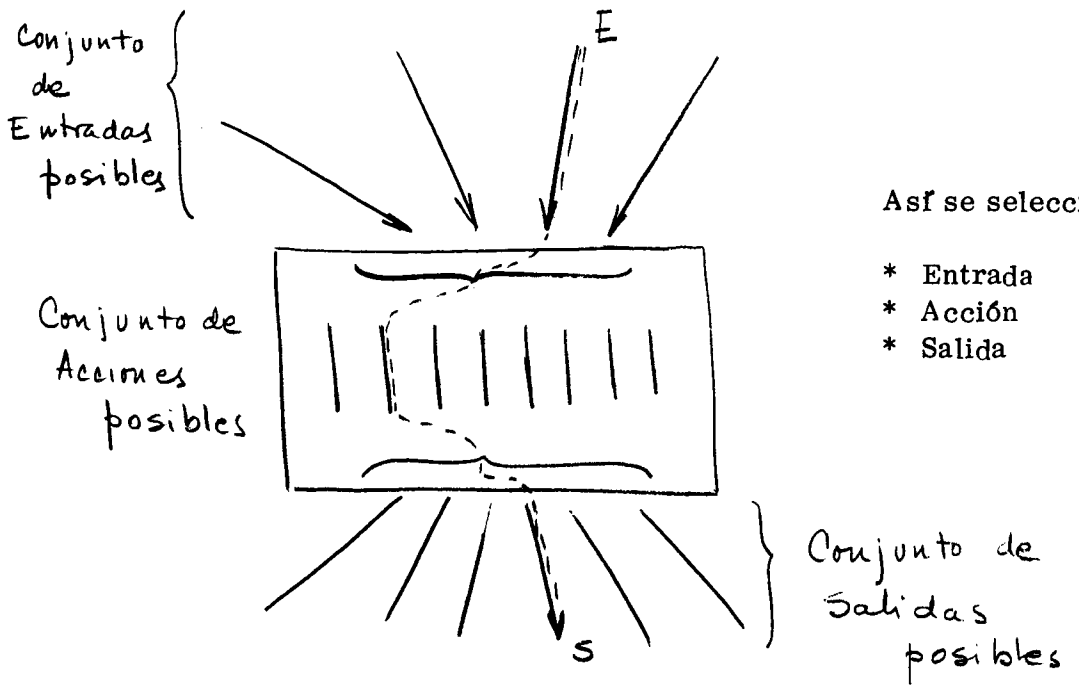
Vuelco desde disco a { cinta / disco }

Restauración de { cinta / disco } a disco

CAPITULO VIII

UTILITARIOS

Se llaman utilitarios, a aquéllos programas que están incorporados al sistema y pueden ser llamados por el usuario. Cada uno de estos programas puede realizar distintas tareas y la elección de las opciones se realiza por medio de tarjetas de control y uso de parámetros indicadores.



Así se selecciona:

* Entrada
* Acción
* Salida

Hay tres tipos de programas utilitarios:

- Utilitarios para el sistema
- Utilitarios para archivos
- Utilitarios independientes

Descripción:

```
OVERLAY         UNO
    INSERT      AJUS
OVERLAY         UNO
    INSERT      AMI,XAP
OVERLAY         REGION 2 (REGION)
    INSERT      BAS
OVERLAY         REGION 2
    INSERT      TIDE
OVERLAY         DOS
    INSERT      CRIO
OVERLAY         DOS
    INSERT      ADDIC
OVERLAY         REGION 2
    INSERT      VORT
OVERLAY         REGION 3 (REGION)
    INSERT      AMIN,DASH
OVERLAY         REGION 3
    INSERT      SUM
```

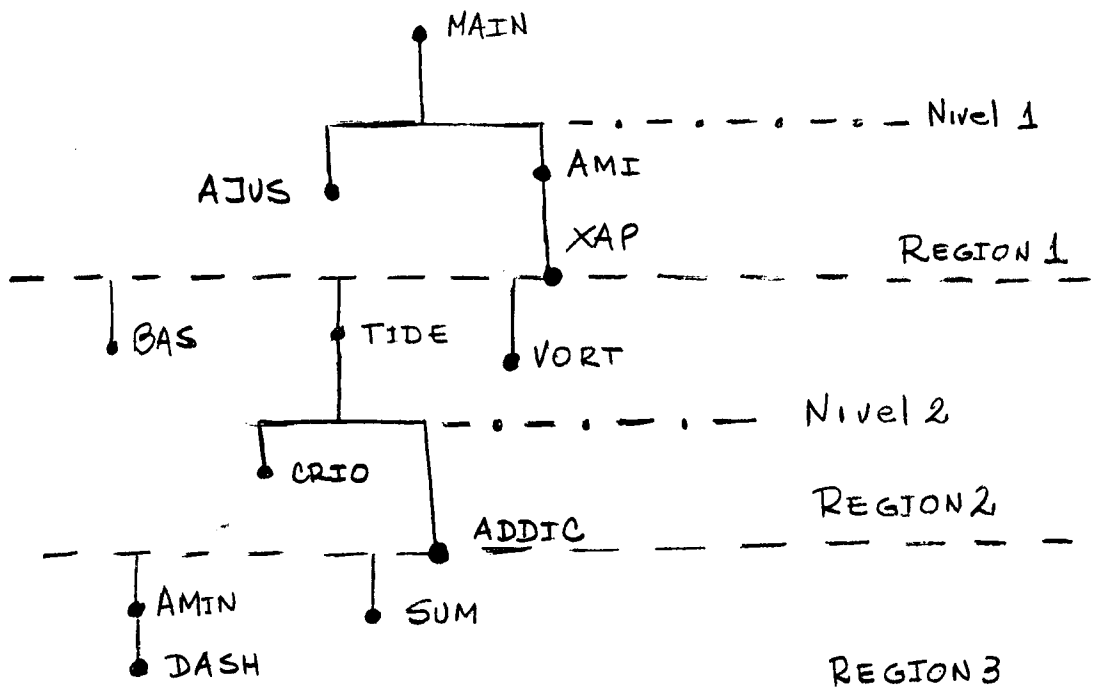
Representación con tarjetas

- Se transmite región por región
- La región uno (1) se transmite por un lote de tarjetas OVERLAY e INSERT como si fuese única
- Para las otras regiones no existe más la noción de segmento raíz
- El comienzo de la región n se considera punto de partida de la parte de árbol que contiene ⟶ debe tener un nombre simbólico
- Los lotes de tarjetas de diferentes regiones se colocan sucesivamente por orden de aparición de las regiones

Ejemplo 3: el árbol anterior se transmitiría así:

```
ENTRY      A
OVERLAY    ALFA
    INSERT B
OVERLAY    ALFA
    INSERT C
OVERLAY    BETA (Region)
    INSERT D
OVERLAY    BETA
    INSERT E
```

Ejemplo 4: árbol con 3 regiones y 2 niveles



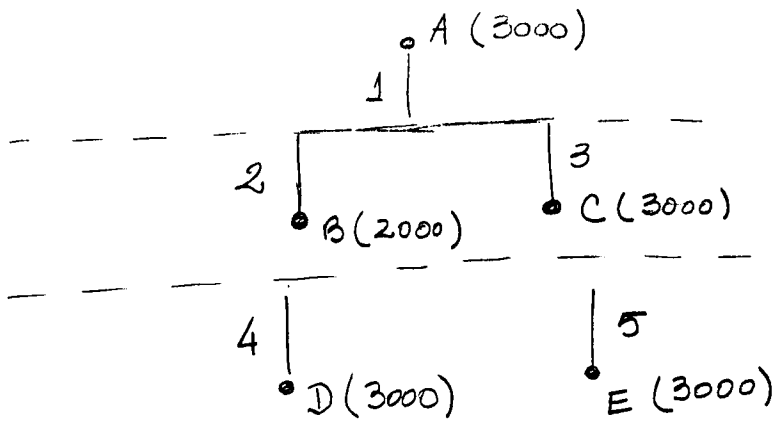
Principio del overlay en varias regiones

. El árbol se diseñará en partes disjuntas
. A cada parte del árbol se le asigna 1 región de memoria
. El conjunto de las regiones = partición

a) Reglas

1. En el interior de una región todo ocurre como si fuese la única
2. El concepto de camino y segmentos excluyentes no existen entre 2 regiones diferentes.

Ejemplo 2: El problema anterior se resolvería:



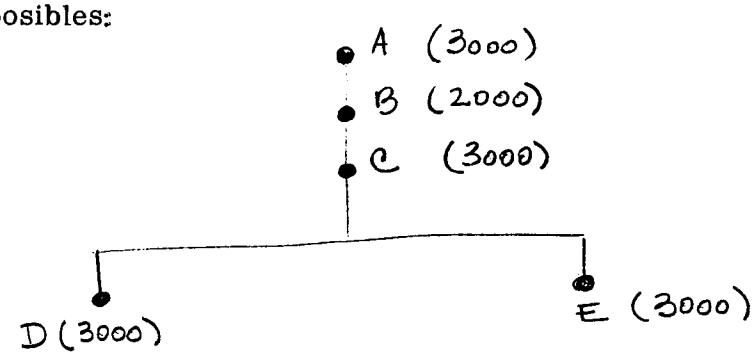
Cómo opera el sistema en este caso

1)	A llama a B	se carga B en región 1
2)	B llama a D	se carga D en región 2
3)	se vuelve a B	
4)	B llama a E	se carga E en región 2 borrando D
5)	retorno a B, retorno a A	
6)	A llama a C	se carga C en región 1 borrando B
7)	C llama a D	se carga D en región 2 borrando E
8)	retorno a C	
9)	C llama a E	se carga E en región 2 borrando D
10)	retornos	

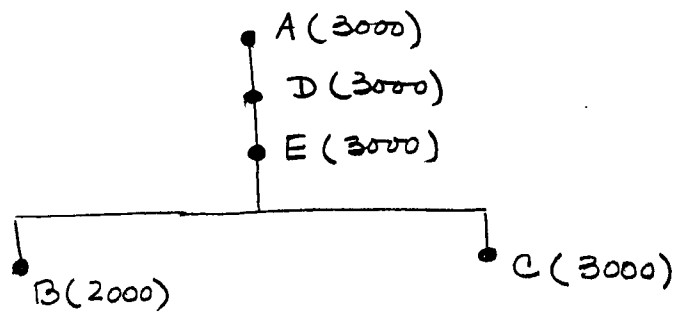
Llamadas y retornos:

1 A llama a B
2 B llama a D
3 D retorna a B
4 B llama a E
5 E retorna a B
6 B retorna a A
7 A llama a C
8 C llama a D
9 D retorna a C
10 C llama a E
11 E retorna a C
12 C retorna a A

Memoria disponible: 10000
Arboles posibles:



longitud: 11000

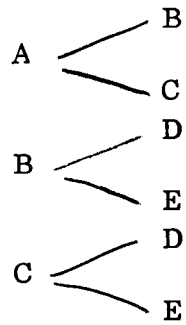


longitud: 12000

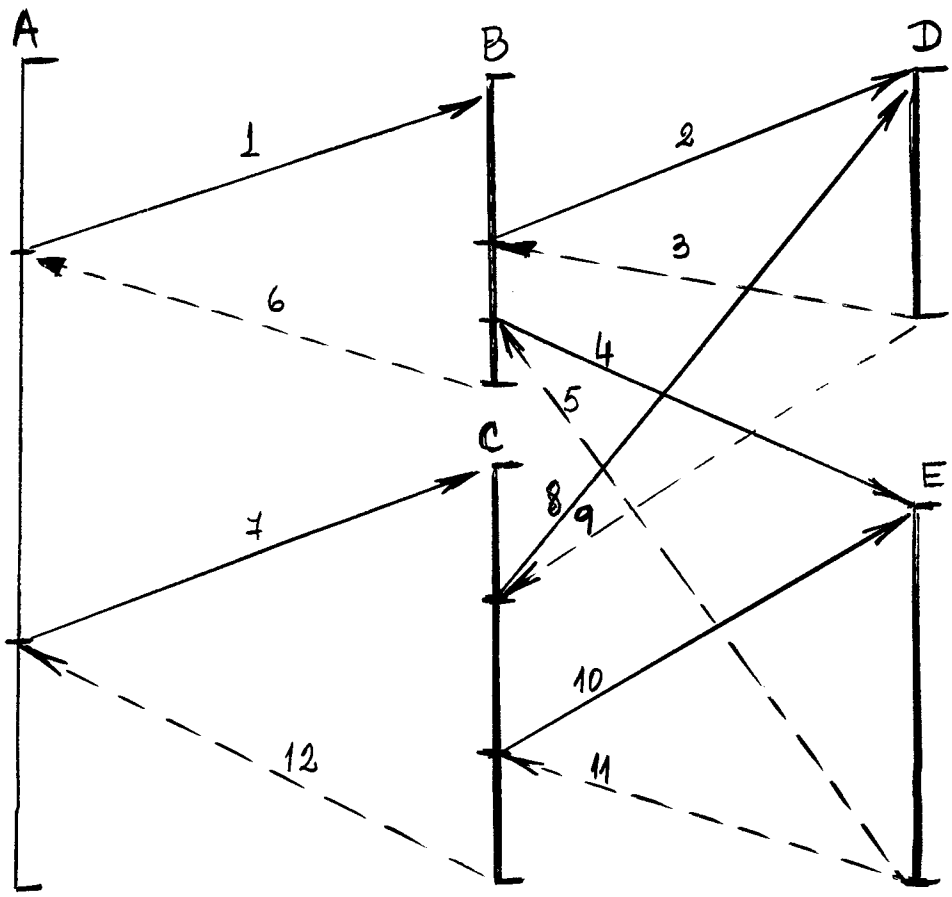
Ejemplo 1

		longitud
Main	A	3000
	B	2000
	C	3000
	D	3000
	E	3000

llama a



Esquemáticamente:



3) Presentación de un trabajo con segmentación

Las tarjetas que definen el Overlay deben incorporarse en la etapa de resolución de vínculos.

Ejemplo 8: no hay lotes objeto

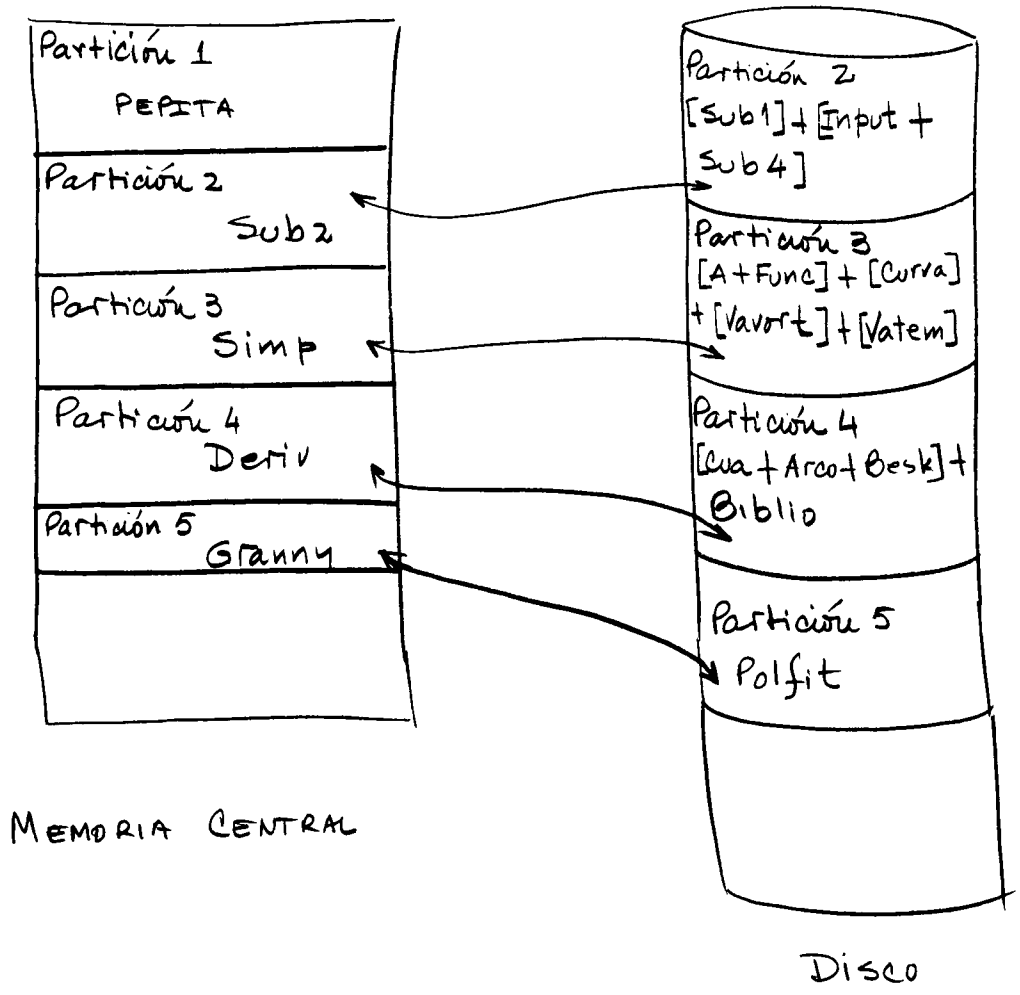
```
//EXEC   FORTGCLG,PARM.LKED = 'OVLY'
//FORT.SYSIN   DD *
        { lotes Fortran
/*
//LKED.SYSIN   DD *
        { tarjetas de control de la segmentación
/*
//GO   , ... etc.
```

Ejemplo 9: hay lotes objeto

```
//   EXEC   FORTGCLG,PARM.LKED = 'OVLY'
//FORT.SYSIN   DD *
        { lotes Fortran
/*
//LKED.SYSIN   DD *
        {lotes objeto
        {tarjetas de control del Overlay
/*
//GO, . . . .  etc.
```

4. Segmentación con regiones múltiples

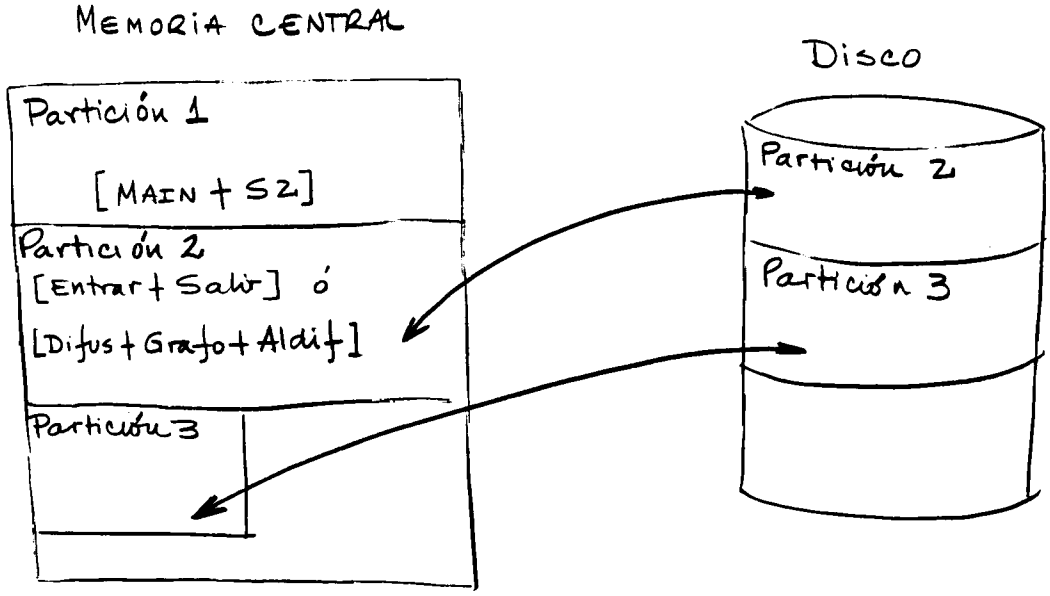
Las reglas anteriores son a veces insuficientes.



La figura indica qué hay en memoria central y disco cuando está en memoria el camino de origen 9:

1: PEPITA
2: [SUB1] o [SUB2] o [INPUT + SUB4]
3: [A + FUNC] o [CURVA] o [SIMP] o [VAVORT] o [VATEM]
4: [DERIV] o [CUA + ARCO + BESK]o [BIBLIO]
5: [POLFIT] o [GRANNY]

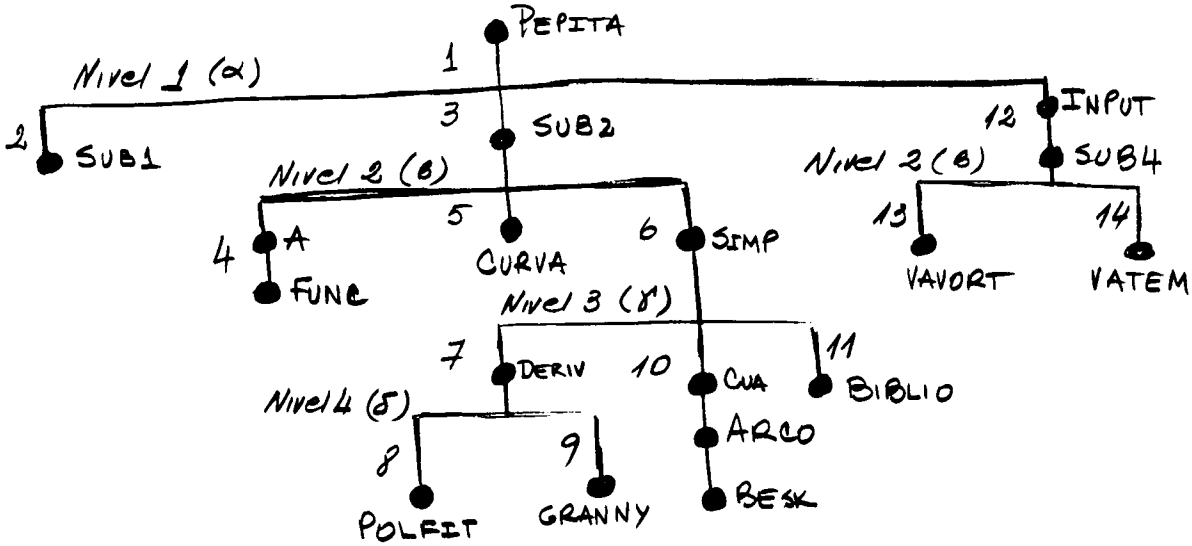
```
ENTRY          PEPITA
OVERLAY        ALFA
  INSERT       SUB1
OVERLAY        ALFA
  INSERT       SUB2
OVERLAY        BETA
  INSERT       A, FUNC
OVERLAY        BETA
  INSERT       CURVA
OVERLAY        BETA
  INSERT       SIMP
OVERLAY        GAMA
  INSERT       DERIV
OVERLAY        DELTA
  INSERT       POLFIT
OVERLAY        DELTA
  INSERT       GRANNY
OVERLAY        GAMA
  INSERT       CUA, ARCO, BESK
OVERLAY        GAMA
  INSERT       BIBLIO
OVERLAY        ALFA
  INSERT       INPUT,SUB4
OVERLAY        BETA
  INSERT       VAVORT
OVERLAY        BETA
  INSERT       VATEM
```

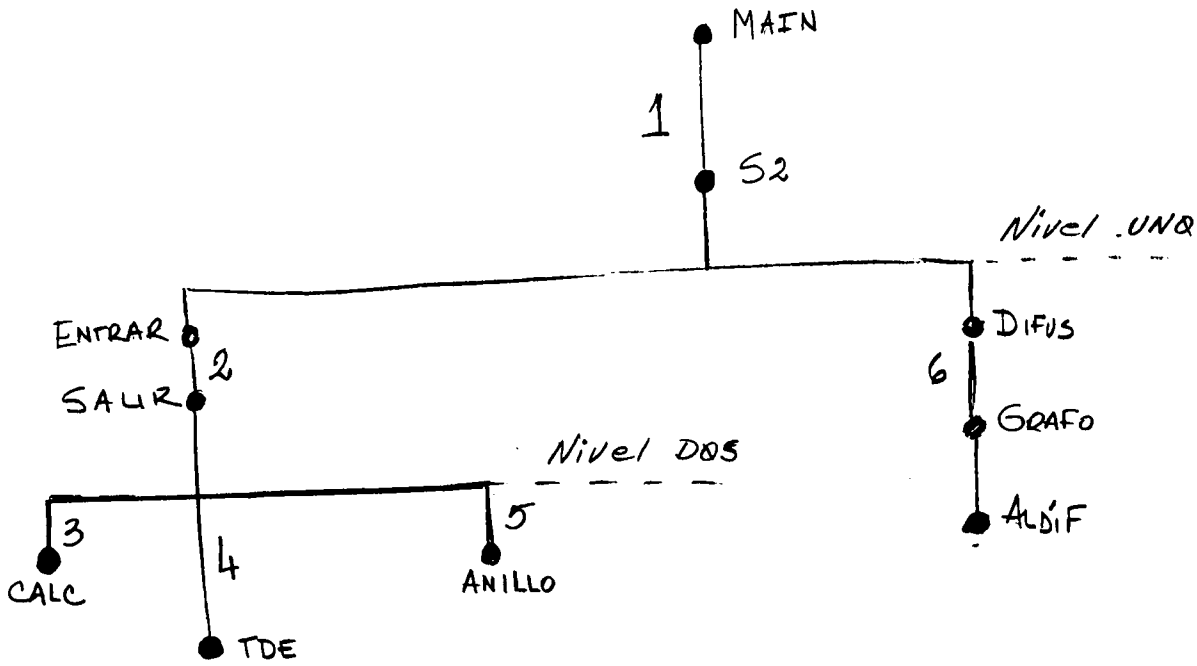


contenido de cada partición:

1: Main + S2 (siempre)
2: [Entrar + Salir] o [Difus + Grafo + Aldif]
3: [Calc] o [Tde] o [Anillo]

Ejemplo 7: Sea el siguiente árbol





OVERLAY	UNO
INSERT	ENTRAR, SALIR
OVERLAY	DOS
INSERT	CALC
OVERLAY	DOS
INSERT	TDE
OVERLAY	DOS
INSERT	ANILLO
OVERLAY	UNO
INSERT	DIFUS, GRAFO, ALDIF
ENTRY	MAIN
INSERT	MAIN, S2 (opcional)

hay tantas sentencias Overlay como (segmentos -1)

Representación de lo que ocurre durante el proceso:

- Se usa una sentencia OVERLAY para definir el segmento y su nivel.
- Se usan una o varias sentencias INSERT para definir los programas que componen el segmento.

Reglas de codificación

i) un espacio en blanco en la columna 1
ii) la palabra OVERLAY o INSERT
iii) un blanco después
iv) nivel del segmento (para la sentencia OVERLAY)
v) nombres de los programas (para la sentencia INSERT) separados por comas
vi) no pasarse de la columna 71 (si no alcanza, agregar otra INSERT)

Para el ejemplo 5, haríamos:

```
␣ OVERLAY        NIVEL 1
␣ INSERT         DIFUS, GRAFO, ALDIF
```

o bien:

```
␣ OVERLAY        NIVEL 1
␣ INSERT         DIFUS
␣ INSERT         GRAFO
␣ INSERT         ALDIF
```

y hay otras variantes equivalentes.

ATENCION: 1) el orden de los programas dentro del segmento debe ser respetado
2) los nombres de programas pueden ser de rutinas en FORTRAN o Assembler.

2) Representación de un árbol por instrucciones

Un árbol completo se transmite al Editor de Vínculos segmento por segmento, de arriba hacia abajo y de izquiera a derecha.
La raíz no se transmite como el resto. Simplemento se dice por qué programa se inicia el árbol con la sentencia ENTRY.

Ejemplo 6: sea el árbol siguiente

iii) un segmento n se refiere a otro segmento que es excluyente
Esto es un ERROR (por ahora).

2. Tarjetas de control de la segmentación

Es necesario indicar al sistema y en particular al Linkage Editor si un programa tiene (o no) estructura de overlay. Para esto se usa el parámetro OVLY de la sentencia EXEC del lenguaje de control de trabajos.

```
//... EXEC  FORTGCLG, PARM.LKED = 'OVLY'
```

y en el CUPED

```
//...EXEC    FORTGCLG, PARM.LKED=(IEWL,'DD,=SYSLMOD',
    'PARM=XREF, LIST, LET, OVLY').
```

La estructura de la segmentación se da mediante dos sentencias:

```
OVERLAY
INSERT
```

1) Representación de un segmento por instrucciones

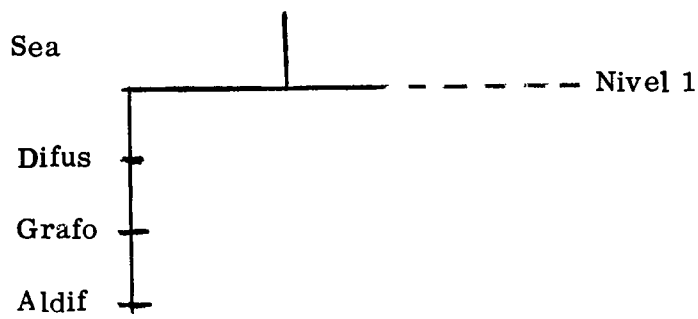
La estructura del árbol tiene varios niveles, para describirlos se da un nombre simbólico a cada nivel

Segmentos 2 y 6: pertenecen al nivel 1
Segmentos 3,4 y 5: nivel 2

Regla 1: A los segmentos que tienen el mismo nivel les corresponde un mismo símbolo de nivel, sea UNO para 2 y 6
DOS para 3,4 y5
(se los puede denominar MARIA Y PEPE)

Regla 2: Si dos segmentos no pertenecen al mismo nivel no pueden tener el mismo símbolo.

Ejemplo 5: de codificación de un segmento mediante instrucciones.



b) Construcción de un árbol

a) reglas sobre la carga de segmentos
b) reglas sobre la comunicación entre segmentos

a) si el segmento n está en memoria central implica;
   i) el camino de origen n está todo en memoria central
   ii) ningún segmento excluyente está en memoria central

   Consecuencias:
   - el segmento 1 siempre está en memoria y por ende contiene el programa principal. No siempre el principal pero sí la(s) rutina(s) que deben permanecer en memoria durante todo el proceso. Se lo llama RAIZ.
   - el camino más largo debe caber en la memoria.

b) hay cuatro tipos de comunicación entre segmentos:
   i) un segmento n se refiere a un segmento m que forma parte del camino de origen n (P.ej. el RETURN de ANILLO se refiere a SALIR). Por la regla (i) el camino de origen n está todo en memoria central y no hay problema.

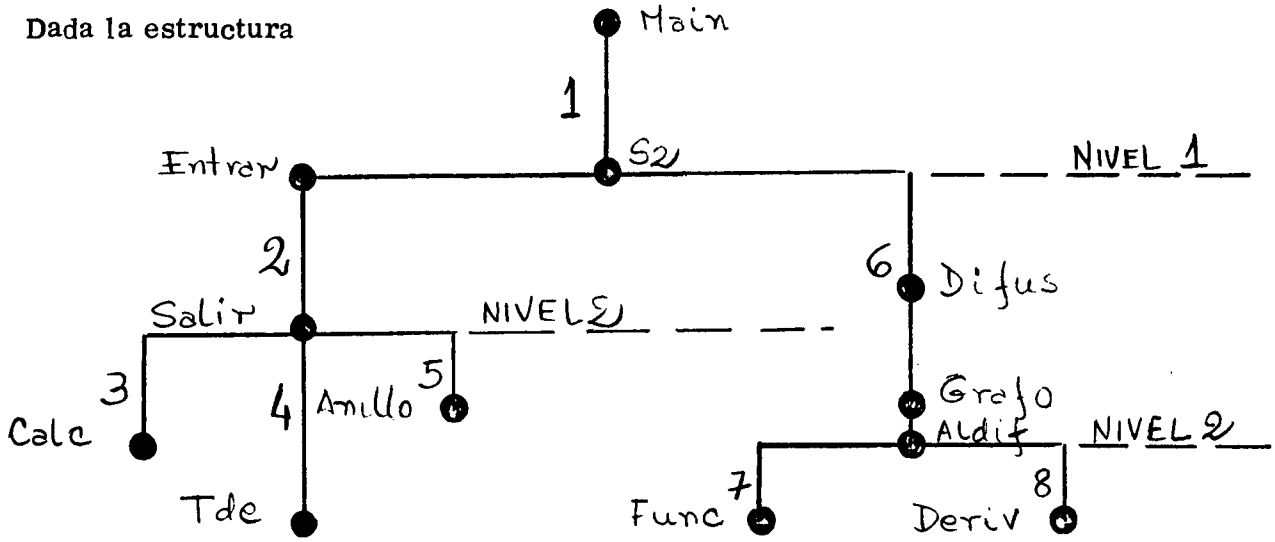
   ii) un segmento n que forma parte del camino de origen m se refiere al segmento m
   Puede ocurrir:
   - m está en memoria
     Ejemplo 3: S2 llama a SALIR y en el momento de la llamada la memoria contiene:

     MAIN
     S2
     ENTRAR
     SALIR
     ANILLO

   - m no está en memoria: la implementación de la segmentación hace que los segmentos faltantes sean traídos a memoria y estamos en el caso anterior
     Ejemplo 4: S2 llama a SALIR y en el momento de llamada la memoria contiene

MAIN	Después	MAIN
S2	del	S2
DIFUS	reemplazo	ENTRAR
GRAFO	dinámico	SALIR
ALDIF		

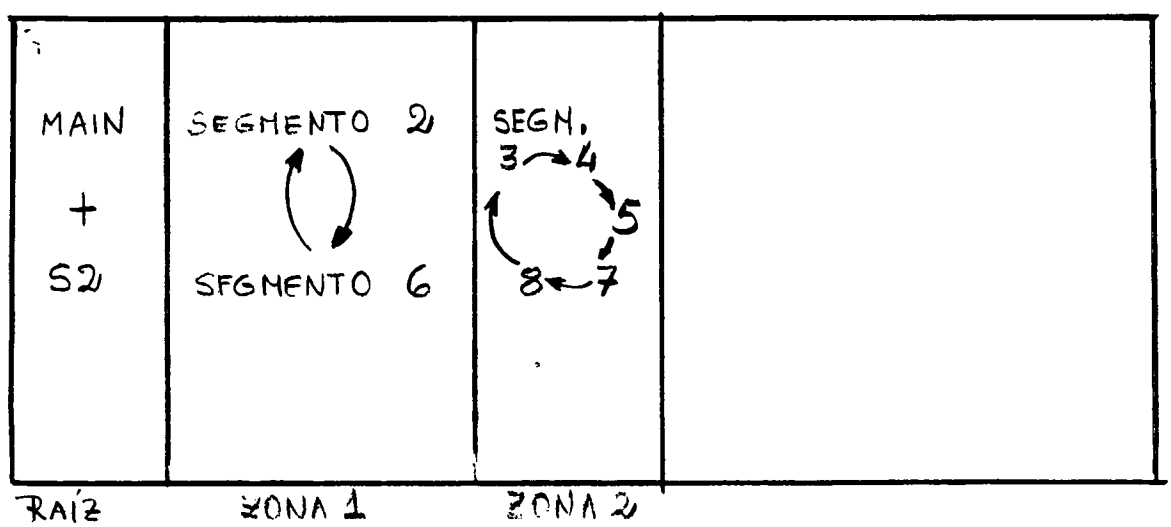
Dada la estructura



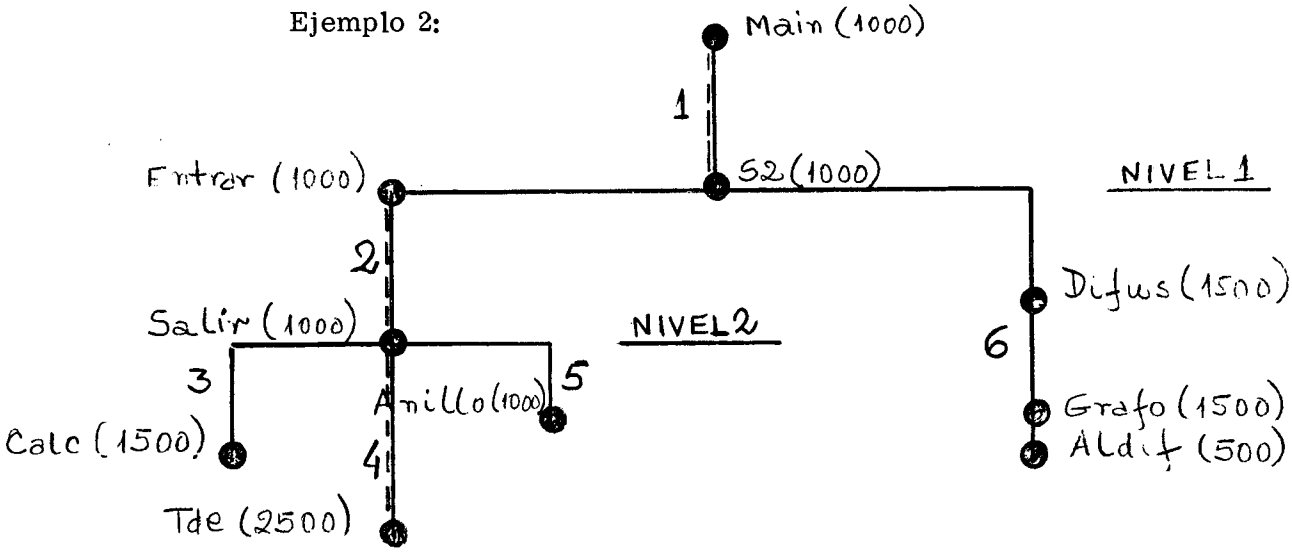
Main y S2: están en memoria central todo el tiempo.

El segmento 2 que contiene ENTRAR y SALIR ⟷ Usa la misma zona de memoria que ⟷ Segmento 6, Difus, Grafo, Aldif

Los segmentos 3, 4,5 usan otra zona ⟷ que intercambian con ⟷ Segmentos 7 y 8



Ejemplo 2:



Segmento unidad funcional más pequeña (comprende una o varias secciones de control) que puede ser cargada como una única entidad lógica durante la ejecución.

1, ..., 6: Segmentos, se numeran de arriba hacia abajo y de izquierda a derecha.
Cada segmento simboliza un conjunto de subprogramas.
Cada rama del árbol está integrada por una unidad del programa cargable separadamente y a cada uno el LE asigna un número.
Camino de origen n (donde n es un número de segmento) es el conjunto de segmentos ubicados entre el segmento 1 y n, ambos incluídos. P.ej. el camino de origen 4 es el marcado: está compuesto por los segmentos 4, 2 y 1.

Longitud de un segmento: es la suma de las longitudes de los subprogramas que lo constituyen. (No puede exceder de 512 K.)

Segmentos excluyentes: se dicen excluyentes dos segmentos que no pueden pertenecer a un mismo camino, P.ej.: 5 y 6.

CAPITULO VII

OPTIMIZACION II : MEMORIA

1. Segmentación: definiciones y motivación de la técnica

Para ser ejecutado un módulo de carga debe ser cargado en memoria. Si la memoria es muy chica puede ocurrir que no quepa o que se requiera una proporción muy grande de ella.

En estos casos se recurre a la segmentación del trabajo: una parte del módulo de carga, llamada raíz, permanece en memoria; el resto es transferido a memoria dinámicamente cuando la lógica del trabajo lo requiera. Los diferentes segmentos se almacenan en soportes externos. El proceso de transferencia de memoria masiva a central se inicia durante la ejecución sólo si una sección de control que está en memoria hace referencia a otra que no lo está. El programa de control del proceso ubica el segmento donde está la sección de control aludida y, si es necesario, lo carga.

El esquema lógico de segmentación debe hacerlo el autor del programa o un usuario que lo haya estudiado con minucia. Mediante instrucciones de control dirigidas al Editor de Vínculos para que resuelva los enganches con ciertas reglas especiales, se realiza el traslado dinámico de programas a memoria central.

a) Arboles, segmentos, caminos

Ejemplo 1: trabajo con los programas Main, S2, Entrar, Salir, Calc, Tde, Anillo, Difus, Grafo, Aldif.

	Long.		Long.
Main	1000	Anillo	1000
S2	1000	Difus	1500
Entrar	1000	Grafo	1500
Salir	1000	Aldif	500
Calc	1500		
Tde	2500	Total	12500 palabras

Supongamos una memoria de 10000 palabras = 40 K. Es imposible ejecutar el trabajo según la secuencia normal. Pero supongamos además que las llamadas a los subprogramas se hacen según el siguiente esquema lógico.

4. Variables auxiliares

Se usan en general para clarificar el aspecto del programa cuando hay que hacer un cálculo muy extenso.

```
T1 = A**2+B**2+C**2
T2 = W+X+Y-W*X*Y
T3 = (D+E-F) **3* (D-E)
VARY = T1+T2+T3
```

Esta serie de sentencias debe ser combinada en una sola, en función de la claridad puede escribirse:

```
     VARY = A**2+B**2+C**2
1          + W+X+Y-W*X*Y
2          + (D+E-F)**3*(D-E)
```

La sentencia IF será compilada exactamente como si hubiera sido escrita:

```
IF (I.LT.9)  GO TO  10          100 veces
IF (I.GT.50)  GO TO 10           92 veces
                                192 veces
```

si en cambio se escribe:

```
IF (I.GT.50.OR.I.LT.9)  GO TO  10
```

será:

```
IF (I.GT.50)  GO TO  10         100 veces
IF (I.LT.9)  GO TO 10            50 veces
                                150 veces
```

Lo mismo cuando se utiliza AND.

9. Varios

1. Sentencia Data

Es conveniente usarla toda vez que haya que inicializar variables que no serán cambiadas durante el programa.

2. Factoreo

Tratar de realizar las sentencias de cálculo utilizando factor común para evitar operaciones de más.

3. Cálculos redundantes

```
A = B+C+D*E+X**2-F/G
-                          B-F/G y D*E aparecen en todas
-
S = S+D*E-Y+B-F/G+D*E
-
-
T = T-F/G+B+2+D*E
```

```
AUXI = D*E+B-F/G                          conviene hacer
A = C+X**2+AUXI
-
-
S = S+AUXI-Y+D*E
-
T = T+AUXI+2
```

7. Sentencia GO TO

La forma de optimizar la aparición de varias sentencias go to dirigidas por sentencias if, es usar go to computado o asignado.

Ejemplo

```
1   ASSIGN 10 TO I
      -
      -
      -
    GO TO I, (10,20,30,40,50)
```

```
2   I = 1
      -
      -
      -
    IF (I.EQ.1) GO TO 10
```

```
3   I = 1
      -
      -
      -
    GO TO (10,20,30,40,50), I
```

De las tres formas de bifurcación a la marca 10, la primera es la más rápida.

8. Sentencia IF

Un IF lógico se efectúa en igual o menor tiempo que el If aritmético equivalente.

Ejemplo 1

```
IF (A.GT.B) GO TO 20
```

```
    IF (A-B) 10,10,20
10  CONTINUE
```

más rápido y más conveniente
cuando se analiza el programa

Ejemplo 2

```
    DO 10  I = 1,100
    IF (I.LT.9.OR.I.GT.50) GO TO 10
          -
          -
10  CONTINUE
```

Ejemplo 4

```
      X = 0.
      DO  10   I = 1,N
10    X = X + A(I) * 10. **I
```

Cuando hay que elevar una constante a potencias dentro de un ciclo, es preferible tener una tabla con las potencias resueltas y multiplicar por sus elementos.

```
      X = 0.
      DO  10   I = 1,N
10    X = X + A(I) * TABLE (I)
```

5. Expresiones Mixtas (reales y enteros)

Es necesario evitar las expresiones mixtas dentro de una asignación, puesto que es muy costoso el proceso de conversión de una forma de representación a otra.

Ejemplo 1

A = J+AJ+K+AK+L+AL+J*K+AJ*AK+J*L+AJ*AL+J*AL*K

Si se agrupan por tipo en grupos de la máxima extensión posible:

A = (J+K+L+J*K+J*L) + (AJ+AK+AL+AJ*AK+AJ*AL+J*K*AL)

se efectúan en la mitad de tiempo.

Ejemplo 2

```
      DO  10   I = 1,N
      A(I) = C*FLOAT (I)
10    B(I) = D+FLOAT (2*I)
```

```
      XI = 0.
      DO  10   I = 1,N
      XI = XI+1.0
      A(I) = C*XI
10    B(I) = D+XI+XI
```

Ahorra dos conversiones y un producto

6. Area de Común

Ya se han visto los beneficios que reporta el uso de áreas de común (tanto el simple como el rotulado). Unicamente hay que tener cuidado con la alineación de variables enteras y reales dentro de un mismo common, para evitar errores de alineación.

```
A = SIN (FI) **2          A = SIN (FI) **2
B = COS (FI) **2          B = 1. - A
```

4. Exponenciación

Las potencias producen un gran desarrollo de instrucciones de máquina, por lo que deben ser evitadas en lo posible.

Hay algunas formas más "expansivas" que otras:

```
a)  A = B**4
b)  A = B**4
c)  I = 4
    A = B**I
d)  A = B*B*B*B
```

El primer método es el más económico ya que evita el llamado a rutinas Fortran, como hacen el b) y el c). El método a) calcula A por medio de dos multiplicaciones registro a registro, mientras que el d) necesita cuatro multiplicaciones de registro a memoria.

Ejemplo 1

```
T1 = A**3 + B**3          ACUB = A**3
T2 = C*A**3 + 2*B**3      BCUB = B**3
T3 = D*A**3*B**3          T1 = ACUB + BCUB
                          T2 = C*ACUB + 2*BCUB
A3 y B3 se calculan tres  T3 = D*ACUB*BCUB
veces cada uno            Se calculan una sola vez
```

Ejemplo 2

```
X(I) = C*Y**I + D*Y** (I + 1)      X(I) = (C + D*Y) * Y**I
```

Ejemplo 3

```
      DO  10  I = 1,K                TEMP = 1.0
10    X(I) = C*Y**I + D*Y** (I + 1)  DO  10  I = 1,K
                                     TEMP = TEMP * Y
                                10   X(I) = (C+D*Y) * TEMP
```

Cuando el cálculo de potencias aparece dentro de un DO, hay que tratar de eliminarlo.

```
EQUIVALENCE (W(1),A),(W(2),B),.....,(W(10), Z)
      -
      -
      -
WRITE (4) W
```

Ejemplo d)

```
      WRITE (6,111) (A(I), I = J,K)
  111 FORMAT (10 (1X,F8.3) )
```

Hay que dar salida sólo a 1 segmento de tamaño desconocido. Para esto se utiliza una subrutina

```
      NUMB = K - J + 1
      CALL OUTPUT (A(J),NUMB)
            -
            -
            -
      SUBROUTINE OUTPUT (ARRAY,N)
      DIMENSION ARRAY (N)
      WRITE (6,111) ARRAY
  111 FORMAT (10 (1X,F8.3) )
```

3. Biblioteca de Subprogramas Fortran (Matem.)

Ejemplo 1

```
A = SIN (FI) + 1
B = COS (FI) + D
C = 2. * SIN (FI) + 3. * COS (FI)
```

```
FISIN = SIN (FI)
FICOS = COS (FI
A = FISIN + 1
B = FICOS + D
C = 2. * FISIN + 3. * FICOS
```

Reduce a la mitad el uso de las funciones

Ejemplo 2

```
      DO  20  I = 1,10
      DO   20  J = 1,10
      A (I,J) = SIN (C(I))
 20   B (I,J) = COS (C(I))
```

Se usa 100 veces cada una de las dos funciones

```
       DO  20  I = 1,10
      SINC = SIN (C(I))
      COSC = COS (C(I))
      DO  20  J = 1,10
      A (I,J) = SINC
 20   B (I,J) = COSC
```

Se usa 10 veces cada una

Recordar sobre todo las identidades trigonométricas que reducen el uso de las funciones

NOTAR.

En los dispositivos de acceso directo el bloqueo debe ser inferior a 1 pista llena

BLKSIZE $\leq$ 7294 = 1 pista del 2314
13030 en el 3330

2 Arreglos para Entrada/Salida

Ejemplo a)

```
      DIMENSION A (100)
      WRITE (6, 111) (A (I), I = 1,100)          1er. método
  111 FORMAT (10 (1X, F10 . 2) )

      WRITE (6, 111) A                           2do. método
```

Ambos métodos obtienen el mismo resultado. El método 2 es, sin embargo, más rápido que el 1. La diferencia aparece en la forma en la cual la lista de ítems es manejada. Por cada item de la lista se llama desde el programa a la rutina de entrada/salida Fortran. En el método 1 hay 100 llamadas (con un overhead asociado). Además hay un DO implícito. En el método 2 el arreglo es tratado como un solo ítem de lista, independientemente del hecho de que tenga 100 elementos, por lo tanto, hay una sola llamada. La salida es controlada desde la rutina del sistema a diferencia del 1 donde es controlada por el programa.

Ejemplo b)

```
DIMENSION A (1000)                    DIMENSION A (1000), B (100)
      -                               EQUIVALENCE (A(701), B(1) )
      -                                       -
      -                                       -
WRITE (8) (A(I), I=701 800)           WRITE (8) B
```

Aquí, el uso de la sentencia Equivalence, permite reducir el caso anterior, aunque se deba dar salida sólo a un segmento de A.

Ejemplo c)

```
WRITE (4) A,B,C,D,E,F,G,X,Y,Z      reemplazado por

DIMENSION  W (10)
```

Conclusión

Si se conocen los topes del DO es preferible no realizarlo (cuando el rango es chico).

Funciones y Subrutinas

Las funciones y subrutinas se utilizan, en general, para reducir memoria y/o por conveniencia del programa. Sin embargo, hay que notar que si los subprogramas son chicos, el tiempo empleado en el pasaje de argumentos, reserva de registros, etc. puede ser mayor que el que requieren las operaciones del subprograma en sí. En estos casos debe tomarse como regla la codificación de las sentencias del subprograma dentro del programa principal.

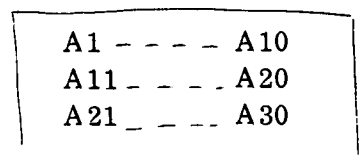
Entrada / Salida

Hay algunas formas de reducir el tiempo de carga en sentencias de entrada/salida.

1. Bloqueo

El bloqueo reduce el número de operaciones de entrada/salida reales. El ahorro de tiempo de CPU es muy considerable. Se ve limitado por la memoria.

```
      REAL * 8 A (1000)
             -
             -
             -
      K = 1
      L = 10
      DO 10 J = 1,100
      WRITE (8) (A(I), I = K,L)
      K = K + 10
10    L = L + 10
```




```
// FT08F001    DD - - - -  DCB = (-- LRECL = 84, BLKSIZE = 84)
```

En este ejemplo hay que llenar 100 veces el área de salida, uno por cada registro. Si se hace:

```
// FT08F001    DD -  - -  DCB = (    LRECL = 84, BLKSIZE = 4200)
```

hay que llenar dos veces el área de salida, una vez cada 50 registros.

Esta es la forma más común de realizar el producto entre dos vectores. Sin embargo, la sentencia:

```
SUM = D(1)* E(1) + D(2) *E(2) + ----------+ D(6) *E(6)
```

toma 1/3 del tiempo del ciclo anterior.

Ejemplo c))

```
      DO   10   I = 1,3                    DO   10   J = 1, N
      DO   10   J = 1,N                    X (1,J)  =  Y (1,J) + Z (J,1)
10    X (I,J) = Y (I,J) +Z (J,I)     10    X (2,J)  =  Y (2,J) + Z (J,2)
                                           X (3,J)  =  Y (3,J) + Z (J,3)
```

Ejemplo d))

Si se supiese que la mayoría de las veces N = 6 convendría elegir la versión 2.

Versión 1

```
      DO   10   I = 1,N
10    A(I) = B(I)
```

Versión 2

```
      IF (N.EQ.6)  GO TO  11
      DO   10   I = 1,N
10    A(I) = B (I)
      GO  TO   12
11    A (1) = B (1)
      A (2) = B (2)
      A (3) = B (3)
      A (4) = B (4)
      A (5) = B (5)
      A (6) = B (6)
12    CONTINUE
```

Ejemplo d')

```
      A (1) = B (1)
      A (2) = B (2)
        ¦                 si se supiese que  N  >  5
      A (5) = B (5)
      IF (N.EQ.5)  GO  TO  11
      DO   10   I = 6,N
10    A (I) = B (I)
11    CONTINUE
```

Ejemplo e)

```
      DIMENSION   X (10,10)
                  -
                  -
                  -
      DO   20  I = 1,10                    DO   21  I = 1,100
      DO   20  J= 1,10                21   X(I,1) = O
20    X(I,J) = O
```

2/3 de tiempo del anterior

Al ser X un arreglo bidimensional, puede ser tratado como unidimensional, con lo cual se ahorra el cálculo de índices dentro del ciclo, utilizando en el 2o. caso 2/3 de tiempo del 1er. caso.

Conclusiones

i) Considerando que las asignaciones de índices llevan más tiempo, hacer el cálculo dentro de los índices.

ii) Para hacer el mismo tratamiento a todos los elementos de un arreglo de más de una dimensión, intentar tratarlo como de una sola dimensión.

3. Eliminación de ciclos

Los DO con rango chico deben ser evitados por el costo intrínseco del DO: { control del ciclo / cálculo de índices

Ejemplo a)

```
      DO    10  J = 1,6                K1(3) = 4
10    K1 (3*J) = J + 3                 K1(6) = 5
                                       K1(9) = 6     13 instrucciones en
                                       K1(12) = 7    lenguaje de máquina
92 instrucciones en lenguaje de        K1(15) = 8
máquina                                K1(18) = 9
```

Ahorro de 85% aproximadamente.

Ejemplo b)

```
      SUM  = 0.
      DO   10  I = 1,6
10    SUM = SUM + D(I) * E(I)
```

2. Uso de subíndices

Ejemplo a)

```
    DO 10  J = 1,6
10  K(3*J) = J + 3
```

10% más rápido

```
    K = O
    DO 11  J = 1,6
    K = K + 3
11  K 1 (K) = J + 3
```

porque es mayor el tiempo empleado en asignar y mantener el valor de K que el de la multiplicación del subíndice.

Ejemplo b)

```
    DO 10  I = 1,6
    J = I + 1
    K = I + 2
    L = I + 3
    M = I +4
10  A(J,K) = B(L,M)
```

```
    DO 1  I = 1,6
11  A(I+1,I+2) = B(I+3,I+4)
```

A pesar de simplificarse los índices en el 1o., el 2o. método se hace en 2/3 de tiempo que el 1o.

Ejemplo c)

```
    DO  10  I = 2,80,2
10  A (I) = B(I+2, I+2)
```

```
    DO 11  J = 2,80,2
    J = I + 2
11  A(I) = B(J,J)
```

Ejemplo d)

```
    DO  10  I = 1,6
    LL(K) = LL(K) + 1
10  C(LL(K)) = D(LL(K))*E(LL(K))
```

seis productos desde LL(K) + 1
hasta LL(K) + 5

```
    LLA = LL(K) + 1
    LLB = LLA + 5
    DO 11 I = LLA, LLB
11  C(I) = D(I)*E(I)
    LL(K) = LL(K) + 6 tiene que tener
                      este valor
```

40% más rápido
LL(K) se incrementa sólo una vez

CAPITULO VI

OPTIMIZACION I

TECNICAS DE OPTIMIZACION PARA FORTRAN IV

OPTIMIZACION DE PROGRAMACION (Tiempo y Memoria)

Sentencia DO

1. Evitar cálculos redundantes

Ejemplo a)

Versión inicial

```
      DO 10 I = 1,30
      D(I) = C(I)*A*B
10    E(I) = E(I) + (A*B)**2
```

A*B	60	productos
$(A*B)^2$	30	productos
C(I)*(A*B)	30	productos
	120	productos
	30	sumas

Versión mejorada

```
      AB = A*B
      ABSQ = AB**2
      DO 10 I = 1,30
      D(I) = C(I)*AB
10    E(I) = E(I) + ABSQ
```

32 productos
30 sumas

Ejemplo b)

```
      SUM = 0.1
      DO 10 I = 1,30
10    SUM = SUM + C(I) * B
```

30 sumas
30 productos

```
      SUM = 0.1
      DO 10 I = 1,30
10    SUM = SUM + C(I)
      SUM = SUM * B
```

30 sumas
1 producto

Conclusiones

En lo posible, factorear al máximo las expresiones que entran en un ciclo de modo que se eliminen las operaciones de exceso (sin subíndice).

7) Ubicación exacta del error:
con la dirección relativa 6E2 y sabiendo que el error es en DQG16 se consulta el listado de instrucciones de máquina de DQG16
Calculo una nueva dirección relativa (esta vez referida al origen de DQG16; según el Editor de Vínculos origen = 3E0)
(6E2 - 3E0) = 302
Se busca en la columna 1 (LOCATION) de la lista de instrucciones de máquina el número 302. La sentencia de máquina que figura en ese renglón (o la inmediatamente anterior si ese número no aparece en la columna) es la causante de la interrupción. Buscando luego en la columna 2 (STA NUM) se obtiene el número de instrucción FORTRAN (en este caso número 4) que provocó el error.

LOCATION	STA NUM	LABEL	OP	OPERAND
000000				
000004				
000008				
00000C				
000010				
-				
-				
-				
000302	4		ST	0,160 (0,13)

* protección
* direccionamiento (se pretende acceder a datos fuera del área del programa
* especificación (una instrucción de máquina no reúne los requisitos que se exigen)
* datos (hay errores en la representación interna de datos)
* división por cero (en punto fijo)
* overflow de exponente
* underflow de exponente
* división por cero (en punto flotante)

Ejemplo 2:

```
IHC207I    IBCOM     PROGRAM  INTERRUPT  OVERFLOW   OLD PSW IS.....9C8DA
TRACEBACK ROUTINE CMLED FROM   ISN      REG.14      REG.15   REG.0  REG.1
            DQG16              0011    6209C50A    0009C5D8   ---    ---
            MAIN                       0000651C    0109C1F8   ---    ---
ENTRY POINT = 0109C1F8
```

Interpretación:

1) El programa principal (MAIN) fue cargado en la dirección absoluta de memoria 0109C1F8 (Entry Point)
2) La interrupción se produjo en DQG16
3) DQG16 fue llamada en la sentencia 11 (FORTRAN) del MAIN
   o en la dirección absoluta anterior a 9C50A (reg.14),
   o en la dirección relativa anterior a (9C50A-9C1F8) = 312
   (En el listado de instrucciones de máquina 312 corresponde a la instrucción CALL DQG16)
4) La PSW anterior (OLD) indica la dirección absoluta de la instrucción siguiente a aquélla en que se produjo la interrupción: 9C8DA
   Cálculo de la dirección relativa: (9C8DA-9C1F8) = 6E2 (dirección relativa del punto adonde volvería el control)
5) Con la dirección relativa puedo ir al mapa del E. de V., donde dice:

CONTROL SECTION	ORIGIN	LENGTH	etc.
MAIN	0000	3DE	
DQG16	3E0	7A4	

   Como 6E2 > 3E0 la interrupción fue en DQG16 (ya lo sabíamos)
6) También puedo verificar lo anterior en la tabla cruzada

---

* NOTA: sería largo explicar con precisión cómo funciona la PSW. Nos basta saber para nuestros fines que hay dos PSW (anterior y actual) en cada momento del proceso. La PSW anterior conserva datos referidos a lo que ocurría en el sistema antes de la interrupción que se está tratando. Esta información se vuelca en parte en los mensajes de interrupción, y es esta PSW, la anterior a la interrupción, la que se imprime en el texto.

Un código de error indica un error de E/S o un error en el uso de bibliotecas del sistema.
Un mensaje de interrupción se refiere a una condición interna de ejecución que el sistema no es capaz de resolver.

1. Diagnósticos con código de error

Veremos el significado de cada parte del diagnóstico en un ejemplo.

Ejemplo 1:

```
IHC215I     CONVERT-ILLEGAL DECIMAL     CHARACTER
TRACEBACK  ROUTINE CALLED FROM  ISN  REG. 14   REG. 15   REG. 0    REG. 1
           IBCOM                 35  0004D254 0004D200 000000GF 0004D210
           MAIN                      00004004 0104C2D0 FFFFFF38 000647F8
ENTRY POINT = 0104C2D0
```

Interpretación:

1) MAIN fue cargada en la dirección absoluta 0104C2D0
2) MAIN llamó a IBCOM
3) IBCOM habría retornado el control a la dirección 4D254 en MAIN
4) MAIN habría retornado el control a la dirección 4004 en el supervisor
5) La primer rutina de la lista es la que llamó al subprograma de biblioteca donde ocurrió el error (si el primer nombre que aparece es IBCOM puede haberse producido en varias rutinas distintas del sistema).
6) REG.14 da la ubicación (absoluta) adonde se volvería desde la rutina que figura en el mismo renglón. Usando el ENTRY POINT se puede calcular una relativa. En este caso: 4D254 - 4C2D0 = F84
7) REG.15 da la dirección del punto de entrada de cada rutina
8) REG.0 resultados del subprograma
9) REG.1 dirección de la lista de argumentos que se pasó a ROUTINE
10) ISN indica el número de sentencia (en lenguaje fuente) desde donde fue llamada ROUTINE (en el ejemplo, IBCOM fue llamada en la sentencia 35 del Main).

2. Mensajes de interrupción

Los mensajes de interrupción del programa contienen siempre la Palabra Indicadora del Estado del Programa (Program Status Word o PSW) anterior a la interrupción (*). Se producen cuando ocurre alguno de estos problemas.

(*) NOTA en página siguiente

* col.1: NAME
  nombre de la sección de control (las áreas de común no rotuladas se indican: $ BLANKCOM
* col.2: ORIGIN
  origen de la sección de control (referido al cero) después de la vinculación
* col.3: LENGTH
  longitud (en bytes) de cada sección en notación hexadecimal

Para cada sección de control el mapa indica, a la derecha los nombres de entrada si los hay y la dirección donde se define ese nombre.
Al final del mapa figura el punto de entrada: dirección relativa del punto de entrada principal. Inmediatamente aparece la longitud total en bytes del módulo (o del camino más largo si se trata de un programa segmentado, como ya veremos en el capítulo VII).
Ver Fig.4.

2. Tabla cruzada

Está compuesta por un mapa del módulo de carga y una lista de referencias cruzadas para cada sección de control. En la lista figuran, ordenadamente, todas las direcciones desde las cuales se hacen referencia a símbolos definidos en otra sección de control, según el siguiente esquema:

* col.1: LOCATION
  dirección relativa (al punto de entrada) desde la cual se hace una referencia al símbolo que figura en la columna siguiente.
* col.2: SYMBOL
  símbolo al que se hace referencia externa.
* col.3: IN CONTROL SECTION
  nombre de la sección de control donde se define el símbolo anterior
* col.4: número de segmento en que aparece esa sección de control (si hay segmentación)

Ver Fig.5.

c) Salida del módulo de carga

En el momento de la ejecución, se generan tres tipos de mensajes:

* diagnósticos con código de error
* mensajes de interrupción
* mensajes del operador

de máquina generadas por el compilador se imprime en el archivo definido con la sentencia SYSPRINT DD (en general, la impresora). El listado que aparece tiene un formato parecido al del Assembler de la máquina ifiere según el compilador y su nivel para una misma implementación de un sistema.
El listado está dispuesto en columnas, con los siguientes encabezamientos:

* col.1: LOCATION
  dirección (en hexadecimal en 360 y octal en Bull GE) de la instrucción.
* col.2: STA NUM
  numeración de la sentencia correspondiente del lenguaje fuente (si la sentencia es ejecutable).
* col.3: LABEL
  rótulos provenientes del módulo fuente o generados por el compilador (estos últimos llevan letras delante).
* col.4: OP
  código de operación de la instrucción de máquina.
* col.5: OPERAND
  operandos de la instrucción de máquina
* col.6: BCD OPERAND
  elementos significativos a los cuales se refiere esa instrucción de máquina en particular.
  P.ej.: puntos de entrada, etiquetas, nombres de variables y constantes.
  Ver Fig. 3

b) Salida del Editor de Vínculos (Linkage Editor)

Según se haya elegido la opción MAP o la XREF entre los parámetros del Editor de Vínculos en la sentencia EXEC, aparece un mapa después de la vinculación o un mapa más una tabla cruzada (cross-reference list, en la jerga).

1. Mapa del Editor de Vínculos

El mapa del módulo de carga es escrito en el archivo definido por la sentencia SYSPRINT DD (para el Editor de Vínculos cada programa -principal o subordinado- y cada bloque de común, rotulado o no, se denomina Sección de Control). El mapa contiene la siguiente información: todas las secciones de control del módulo objeto y todos los nombres de entrada en cada sección de control.

CAPITULO V

EMPLEO DE MAPAS PARA BUSQUEDA DE ERRORES

SALIDA DEL SISTEMA (nos referimos a máquinas IBM/360 ó 370 en los ejemplos

El compilador, el editor de vínculos y otros programas del sistema suministran información adicional útil como documentación y/o como ayuda para ubicar problemas.

a) Salida del compilador

1. Listado del programa fuente

Cuando se elige la opción SOURCE de los parámetros de la sentencia EXEC, un listado de las instrucciones fuente se escribe en el archivo que se haya definido con la sentencia SYSPRINT DD (en general, la impresora). Ver Fig.1.

2. Mapa de compilación

Cuando se elige la opción MAP en la sentencia EXEC, aparece una tabla de nombres en el archivo definido con la SYSPRINT DD (en general la impresora). La tabla está dividida en secciones, correspondientes a 7 clases posibles de variables usadas en el módulo fuente.

* variables que están en bloques de común
* variables que figuran en una sentencia EQUIVALENCE
* variables escalares
* arreglos(multidimensionales)
* subprogramas llamados
* variables que figuran en un NAMELIST
* sentencias FORMAT

El mapa desglosa los 7 tipos y lista las variables, cada una con su ubicación en memoria (al compilar) [en hexadecimal en 360]
Ver Fig.2.

3. Listado del módulo objeto

Cuando se elige la opción LIST (o similares) entre los parámetros del compilador en la sentencia EXEC, el listado de las instrucciones

Paso CREAR: genera la cinta ejecutando el programa utilitario IEBGENER
Se define:

a) Archivo de salida de resultados: SYSPRINT (oblig.). Se dice que este archivo tiene salida (SYSOUT) por impresora (=A)

b) Archivo de trabajo: ARCH2 (el nombre lo da el programador). Aquí cinta magnética.
DSNAME: No existe, no hay rótulo
UNIT: tipo de unidad, cinta 1600 bpi
DISP: New = archivo nuevo
Pass = se usa en otro paso posterior
DCB: bloque de control de los datos
DEN: densidad { 2 = 800 bpi / 3 = 1600 "
RECFM: tipo de registro físico
FB: fijo bloqueado
BLKSIZE: long. del bloque físico = 3200 bytes
LRECL = long. del registro lógico (80 bytes = 1 tarjeta)
LABEL: (NL), no hay rótulos para los diferentes archivos.
El archivo nuestro es el 5o.
VOL=SER= Número de serie del volumen en biblioteca.

c) Archivo de entrada: ARCH1
se trata de un archivo que viene en tarjetas (sigue inmediatamente a la DD). Es el programa que se quiere grabar ubicado en (1).

d) Archivo de control: SYSIN no existe (en el programa)

e) En el paso de ejecución //GO se usa la cinta grabada antes y el programa FORTRAN que la lee hace READ (8, No. de formato)
↑ ← FT08F001

DCB = *.CREAR.ARCH2
nombre del paso — nombre del archivo

Se dice que el DCB es el mismo que figura para el archivo ARCH2 del paso CREAR, según se definió en tarjetas (*).
DISP = OLD ya estaba creado
KEEP hay que guardarlo

f) Ejemplo de un trabajo que usa cinta magnética

Se quiere (en un solo trabajo)

a) crear una cinta (donde se guardará un programa Fortran) y hacerlo sólo con instrucciones de control.

b) ejecutar un programa FORTRAN cuya función es leer esa cinta

Son dos pasos { CREAR / LEER

```
//     JOB
// CREAR   EXEC   PGM = IEBGENER
// SYSPRINT  DD  SYSOUT = A
// SYSIN  DD  DUMMY
// ARCH 2   DD    UNIT=TAPE, DISP = (NEW, PASS)

//         DCB = (DEN=3, RECFM=FB, BLKSIZE=3200, LRECL=80),
//           LABEL = (5,{SL / NL}),

//           VOL = SER = T1231
//ARCH  1    DD   *
  (1)

 / *
// LEER   EXEC   FORTGCLG
// FORT.SYSIN    DD    *
   (2)

// GO.FT08F001   DD  UNIT = TAPE,
//           LABEL = (5,{SL / NL}),
//           DCB = * . CREAR . ARCH2,
//           DISP = (OLD, KEEP)
/ *
```

Paso LEER: composición standard, se ejercita el procedimiento FORTGCLG. El lote FORTRAN que es nuestro programa de lectura se coloca en (2)
Lee la cinta (los datos del programita Fortran están en cinta y de ahí que //GO. defina una cinta).

d) Compaginación más ejecución. Hay lotes y datos binarios

```
// ... JOB ...
//JUAN    EXEC  ASMFCLG
//ASM . SYSIN   DD   *
         lote Assembler
/ *
//LKED .SYSIN   DD   *
         binarios
/ *
//GO .SYSIN   DD   *
          datos
/ *
```

e) Ejecución de un trabajo que comprenda:
assembler, Fortran, lotes binarios y datos

i) si hay uno solo lote Assembler ⟶ dos pasos
Paso 1: llama al procedimiento ASMFC
Paso 2: llama al procedimiento FORTGCLG

ii) si hay n lotes Assembler ⟶ (n + 1) pasos
Paso 1: ⎫
  - ⎪
  - ⎬ llaman al ASMFC
  - ⎪
Paso n ⎭
Paso (n + 1) llama al FORTGCLG

La estructura sería, tanto para el caso i) como para el ii):

```
// ... JOB ...
// PASOA  EXEC  ASMFC,PARM . ASM = LOAD
//ASM . SYSIN   DD   *
          lote Assembler
/ *
//PASOB    EXEC   FORTGCLG
//FORT . SYSIN   DD  *
         lotes Fortran
/ *
// GO . SYSIN   DD  *
         lotes binarios
/ *
// GO. SYSIN   DD  *
           datos
/ *
```

4. Ejemplos de trabajos comunes

a) Compilación simple. Fortran (se pide binario)

```
// ...   JOB  ...
     opt
//COMPIL   EXEC  {FORTGC} ,PARM.FORT = DECK ....
                 {FORTHC}
//OBJECT  = 'SYSOUT = B'
// FORT.SYSIN      DD     *
          }
          } lotes Fortran
          }
/ *
```

b) Compaginación simple

Se trata de un solo lote en Assembler con salida perforada del módulo objeto.

```
// ...  JOB  ...
     opcional
//ASSEM   EXEC   ASMFC,PARM.ASM = DECK
//     OBJECT  =  'SYSOUT = B'
// ASM.SYSIN     DD    *
              lote Assembler
/  *
```

c) Compilación más ejecución

Contiene: uno o varios programas F, lotes objeto y datos. Se pide listado de instrucciones de máquina.

```
// ...  JOB  ...
// PEPE   EXEC  {FORTGCLG} , PARM.FORT = (LIST,ID)
                {FORTHCLG}
// FORT.SYSIN     DD    *
          }
          } lotes Fortran
          }
/  *
// LKED.SYSIN   DD    *
            binario
/ *
// GO.SYSIN   DD  *
           datos
/ *
```

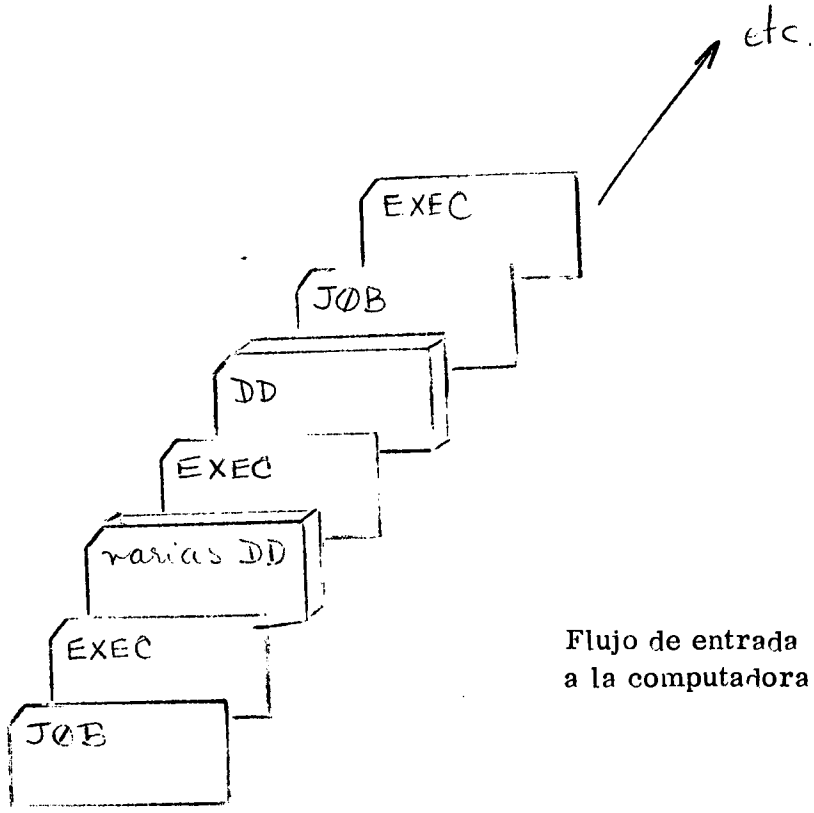
a) El * que sigue a la palabra DD (data definition) indica que un archivo (data set) sigue inmediatamente a la tarjeta DD.

b) FORT.SYSIN significa que las tarjetas dato que siguen (son datos para programas de nivel superior) están destinadas al compilador Fortran (FORTran SYStem INput).

Un lote Assembler va precedido por:
// ASM . SYSIN DD *
Un lote binario por:
// LKED . SYSIN DD *
Un lote de datos por:
//GO . SYSIN DD *

3. Presentación de un trabajo

Un trabajo comprende varias etapas y cada una se compone de una tarjeta EXEC y una o varios DD.



Flujo de entrada
a la computadora

iii) Parámetros del Editor de Vínculos

Recordemos que el Editor de Vínculos trabaja sobre módulos objeto y que produce módulos directamente cargables.

Acción que se desea	Parámetro	Standard
1) mapa con la longitud de cada programa y direcciones relativas de carga	sí: MAP NOMAP	MAP
2) listado de todas las instrucciones de control para el Editor de Vínculos	LIST NOLIST	LIST
3) estructura del trabajo respecto de la segmentación de memoria (overlay)	OVLY	OVLY
4) anular referencias a bibliotecas externas (se puede ejecutar un módulo de carga aunque no estén resueltas todas)	NCAL	
5) ejecución de módulo de carga aún con errores de severidad 2	LET	LET implica XCAL
6) llamadas entre segmentos excluyentes en una estructura de overlay	XCAL	

Aparecen como:
PARM.LKED = (L_1, L_2, ... , L_m) L_i = parám.

NOTAS:
1) No debe haber blancos entre el primer carácter del nombre del procedimiento y el último carácter de los parámetros especificados
2) No se puede perforar más allá de la 71.
3) No se puede interrumpir entre zonas
4) Tarjetas de continuación: // y el resto

Tarjeta DD

Un lote en Fortran debe estar precedido por la tarjeta:

```
// FORT . SYSIN      DD    *
```

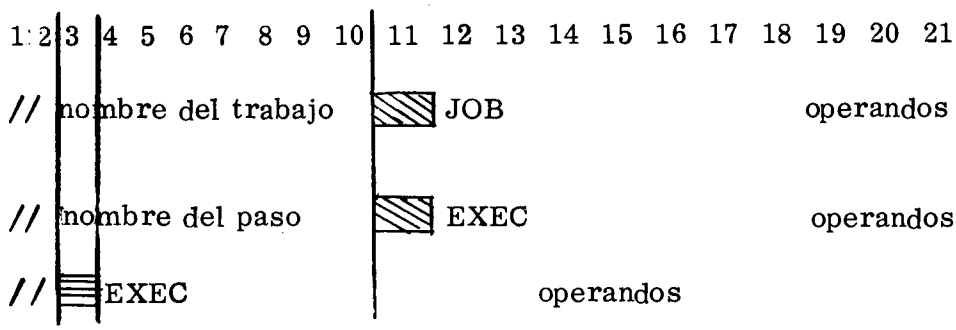
Acción que se desea	Parámetro	Standard
3)lote perforado del módulo objeto (binario, deck)	sí: DECK no: NODECK	NODECK
4)mapa con el nombre de las variables y formatos	sí: MAP no: NOMAP	MAP
5)código por el cual cada instrumento, Call o cada referencia a función es asociada a un identificador numérico (usado por el traceback) Sequence number	si: ID no: NOID Aparece	NOID ISN=Internal
6)Código de perforación: BCD o EBCDIC	BCD EBCDIC	EBCDIC
7)número de líneas por página de listado	LINECNT=YY	YY=50 ó 60
8)nivel de optimización		

Todo esto aparece como variables dentro de PARM.FORT = (ID, MAP, LIST, etc.)

ii) Parametros del compaginador Assembler

Acción que se desea	Parámetro	Standard
1)listado de las instrucciones originales	LIST NOLIST	LIST
2)lote binario	DECK NODECK	NODECK
3)líneas de impresión	LINECNT=YY	
4)carga del módulo objeto	LOAD NOLOAD	LOAD
PARM.ASM = (A_1, A_2, ..., A_k)		

Resumen:



Detalle de la tarjeta EXEC

Función principal: permite llamar a un programa o procedimiento catalogado.

Uso de la información codificada aquí: transmitir, a un programa de control o a uno de servicio, órdenes precisas sobre la ejecución.

a) Contenido de la tarjeta

1. nombre de la etapa (step) dentro del trabajo o dentro del procedimiento
2. nombre del procedimiento o programa a ejecutar
3. opciones para: compilador
   compaginador
   L.E.
4. indicaciones sobre: condiciones de ejecución del paso (relativas a etapas precedentes); sobre tiempo adjudicado al paso; memoria

Veremos ahora los parámetros que controlan cada paso de un job o de un procedimiento.

b) i) Parámetros para el compilador FORTRAN

Se puede pedir que, al terminar la compilación, la máquina brinde:

Acción que se desea	Parámetro	Standard
1) listado de las instrucciones FORTRAN originales	sí: SOURCE no: NOSOURCE	SOURCE
2) listado de instrucciones de máquinas generadas	sí: LIST no. NOLIST	NOLIST

A. Descripción

Col 1 y 2: //
Col 3 a 10 (opcional): nombre del paso
Blanco (al menos uno)
palabra EXEC
Blanco
nombre del procedimiento a ejecutar (FORTGC p. ej.) o nombre del programa a ejecutar (debe ser un miembro de biblioteca)

parámetros para cada etapa del procedimiento (los veremos después)

B. Procedimientos catalogados más usuales en nuestro tipo de trabajo

a) compilar Fortran (nivel G) FORTGC
b) compilar, vincular y ejecutar FORTGCLG
c) vinculación y ejecución LKEDGFTR
d) compaginación Assembler ASMFC
e) compaginación, vinculación, ejecución ASMFCLG

Los diferentes pasos de un trabajo se separan por medio de tarjetas EXEC. Dentro de un mismo paso puede haber lotes de origen distinto (Fortran, binarios, assembler, datos). Es necesario separar los diferentes lotes. Este es el objeto de las tarjetas separadoras, de las cuales hay dos tipos:

a) tarjeta DD: se coloca delante del lote al cual se refiere (define el archivo o data set).
b) /* detrás del lote que termina (indica fin del archivo)

NOTAR:
- un programa FORTRAN es un archivo (data set) para el compilador
- un conjunto de lotes binarios es un archivo (data set) para el Linkage Editor
- un conjunto de datos es un archivo (data set) para el Cargador (Loader)

Col 1 y 2:	//
Col 3 a 10:	nombre del trabajo
Blanco:	(al menos uno)
Col 12 a 15:	JOB
Blanco:	(al menos uno)
Col 16 a 20:	nombre de la cuenta
Col 21:	,
Col 22 a 33:	nombre del responsable
Col 34:	,

Después se codifica:

MSGLEVEL = (x, y)

x=
- 0 Se imprime sentencia JOB; errores en sentencias de control; diagnósticos
- 1 Sentencias de entrada; sentencias de proc. catalogados y sustituciones simbólicas
- 2 Sólo aparecen sentencias que entran

y=
- 0 No aparecen mensajes con errores de alocación; o errores por fin anormal sin lograr terminar
- 1 Aparecen todos los mensajes aunque logre terminar

CLASS =	B	no usa cintas
	C	usa una cinta
	D	usa dos cintas o bien disco
	E	usa más de dos cintas

PRTY Señala prioridades

Llamado a un procedimiento catalogado:

Tarjeta EXEC

Sirve para llamar a procedimientos catalogados.

3) Sentencia DD

Describe archivos (data sets). Puede contener:

i) nombre del archivo para procesar
ii) tipo y número de periféricos de E/S para ese archivo
iii) volúmen(es) en donde reside el archivo
iv) cantidad y tipo de espacio reservado en un volumen de acceso directo
v) información sobre los rótulos del archivo
vi) disposición del archivo después de la ejecución del paso
vii) ubicación de los archivos respecto de la optimización de canales
viii) si un archivo particular se puede usar sólo para entrada o sólo para salida.

4) Procedimientos catalogados

Con programas escritos en FORTRAN o Assembler los trabajos más sencillos que se pueden pensar (y también los más comunes) son:

a) compilación FORTRAN < para detectar errores sintaxis / para obtener módulo objeto
b) compilación más vinculación más ejecución
c) compaginación (como en a)
d) compaginación más vinculación más ejecución
e) compilación más compaginación más vinculación más ejecución
f) ejecución de un programa que entra en forma de módulo objeto

Aún para trabajos tan simples las instrucciones de control son muchas y difíciles entonces, para evitar repeticiones, las compañías suministran procedimientos de empleo para cada una de las 6 tareas enumeradas. Estos procedimientos tienen un nombre y se almacenan juntos constituyendo un catálogo.

Un procedimiento catalogado puede comprender un número cualquiera de tarjetas EXEC seguidas de tarjetas DD.

5) Descripción detallada de las tarjetas de control

2) Sentencia EXEC

Indica el comienzo de un paso y lo describe. Contiene:

i) nombre del paso de trabajo o del paso de procedimiento
ii) nombre del procedimiento o módulo de carga que se debe ejecutar
iii) opciones del compilador y/o editor de vínculos que se desean
iv) información para contabilidad
v) condiciones para saltear la ejecución del paso
vi) límite de tiempo para el paso
vii) límite de memoria requerida para un paso o para todo un procedimiento.

d) Funciones esenciales de los lenguajes de control de trabajos

1) Mantener la correspondencia permanente entre los nombres internos y los nombres simbólicos externos
FORTG ——————— Compilador Fortran

2) proveen las órdenes que permiten pasar resultados de un paso (step) de un trabajo a otro por medio de archivos con nombre
(salida del comp. Fortran: SYSLIN
entrada al comp. Fortran: FORT.SYSIN
salida del Editor de Vínculos: SYSLMOD, etc.)

3) suministran la manera de ejecutar condicionalmente ciertos pasos de un trabajo.

2. Descripción de un Lenguaje de Control de Trabajos (JCL-OS/360)

Sentencias de control de trabajos

// JOB: indica el comienzo de un nuevo trabajo y lo describe (a fines de la contabilidad)

// EXEC: indica un paso de trabajo y lo describe; indica un procedimiento catalogado o un módulo de carga.

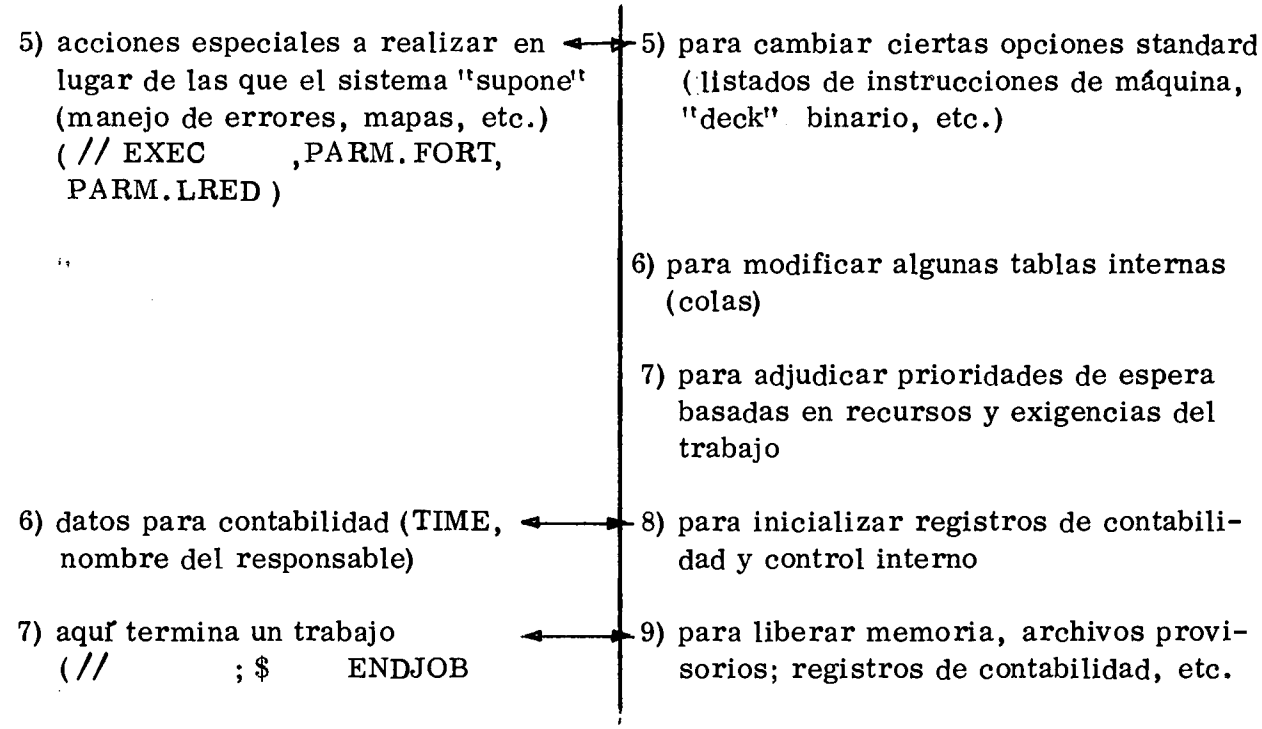
// DD : describe archivos (data sets) y controla la asignación del periférico y del volumen.

* : separa data sets en el flujo de entrada de sentencias de control. Aparece después de cada data set.

1) Sentencia JOB

Es la primera de la sucesión de sentencias que describe un trabajo. Contiene la siguiente información:

i) nombre del trabajo
ii) información para contabilidad
iii) nombre del programador
iv) si las sentencias de control se imprimen o no
v) condiciones para terminar la ejecución de un trabajo
vi) asignación de prioridad
vii) clase de salida para los mensajes del asignador de prioridad
viii) especificaciones de memoria
ix) especificaciones de tiempo máximo



En sistemas de multiprogramación es necesario que la información de control esté íntimamente asociada al trabajo (que venga en el flujo de tarjetas de entrada)

Se hacen necesarios los lenguajes de control de trabajos (Job Control Language ó JCL).

c) Definición elemental del lenguaje de Control de Trabajos

El lenguaje de control de trabajos es un conjunto de instrucciones destinadas a decidir el manejo de los trabajos que entran y -según sus exigencias- las actividades internas que deben realizarse.

Ejemplo:

Enumeramos algunas instrucciones básicas de un JCL imaginario para visualizar esto.

Instrucción	Operando
1) COMENZAR	nombre del trabajo
2) COMPILAR	nombre del módulo fuente
3) CARGAR	nombre del módulo de carga
4) ELEGIR	nombre del periférico
5) TERMINAR	(evita que un trabajo con errores lea más allá de sus datos)

CAPITULO IV

EL SISTEMA OPERATIVO DESDE EL PUNTO DE VISTA DEL USUARIO

1. Papel del lenguaje de control de trabajos

La interacción entre "el sistema" y el usuario si establece mediante los siguientes elementos:

a) opciones sobre ejecución de trabajos
b) lenguajes de control de trabajos
c) lenguajes de tiempo real
d) comunicación con el operador

Control de la ejecución de un trabajo

Se trata de saber qué "le digo" al Sistema Operativo para que haga todo lo que yo quiero y tal cual lo quiero. Es imprescindible para esto, indicar a un Sistema Operativo particular los requerimientos y opciones para cada trabajo

Veremos en primer lugar:

a) la información básica necesaria
b) para qué usa el S.O. esa información

a) Información básica necesaria	b) Para qué usa esta información el S.O.
1) aquí empieza un nuevo trabajo (//JOB; $ SNUMB)	1) para identificar el comienzo de una nueva unidad de trabajo
2) ejecute la siguiente tarea (//EXEC ; $ OPTION)	2) para identificar una actividad particular que debe ejecutar
3) se necesitarán los siguientes recursos (memoria; E/S; etc.) (En // EXEC; $ LIMITS)	3) para asegurar que todos los recursos necesarios estén disponibles y ubicarlos (dentro del sistema). Se pretende que a partir de este momento no puedan disponer de ellos otras actividades(*)
4) se usarán y/o crearán los siguientes archivos (// FORT. SYSIN DD, etc.)	4) para preparar el acceso a archivos existentes y reservar lugar para archivos nuevos

(*) O los compartan eficientemente.

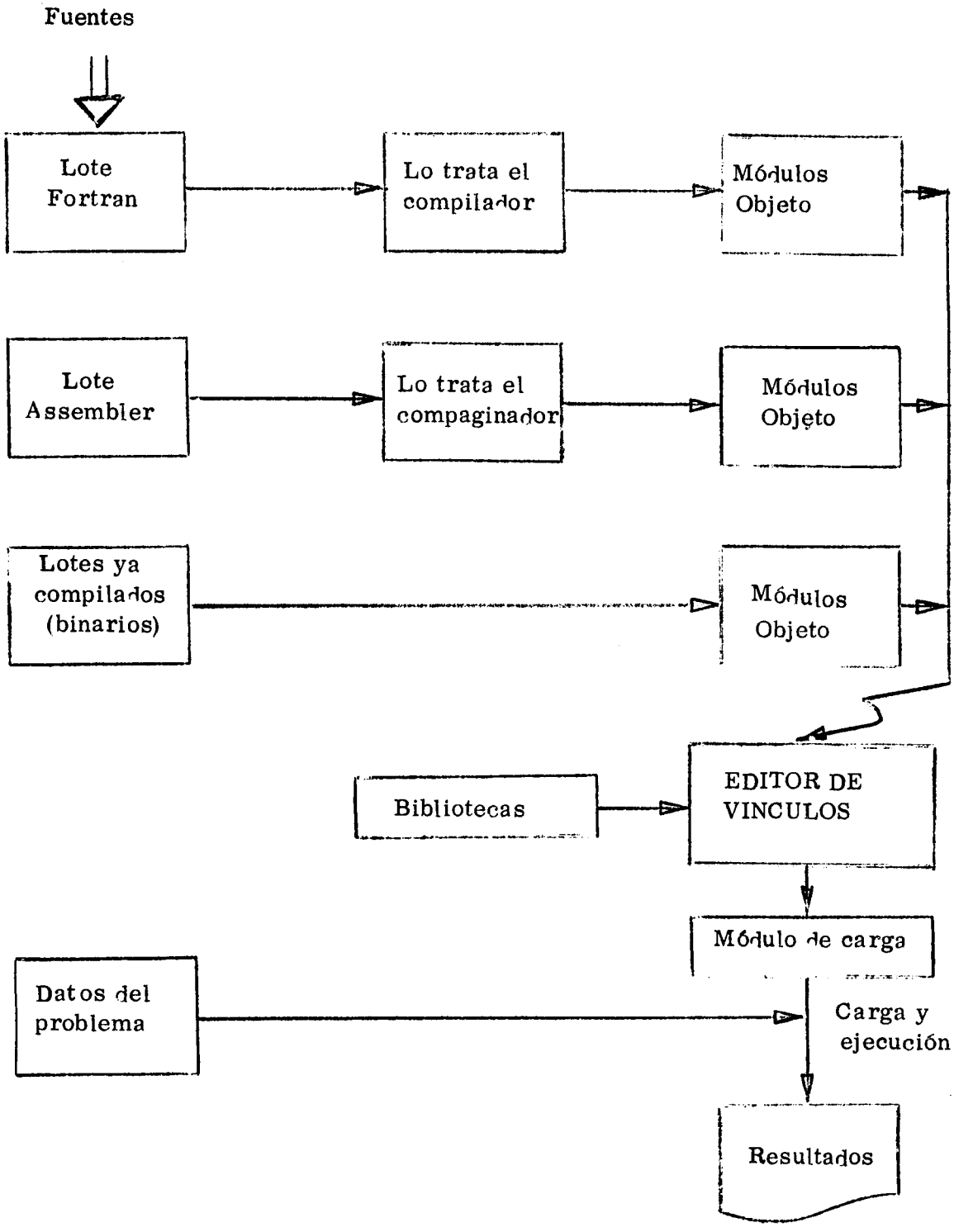
6.3 Parámetros que controlan cada etapa

En cada etapa, cada programa necesita cierto número de informaciones precisas.

1) al compilar
   - nombre del compilador y nivel
   - código de perforación
   - si se quiere o no deck binario
   - si se quiere o no ciertas tablas que permiten ubicar errores
   - lista de instrucciones de máquina, etc.

2) análogamente para la etapa de vinculación.

Esquema general de las etapas principales



Ejemplo 1:

Sea un trabajo de Fortran con una subrutina

Para Para conseguir la generación interna de un código "entendible y ejecutable" por la máquina, se debe:

Se obtiene

a) compilar el programa principal → 1) módulo objeto del programa principal

b) compilar el subprograma → 2) idem para el subprograma

c) cada módulo objeto comprende:

i) la traducción de las instrucciones fuente en instrucciones de máquina
ii) cada instrucción de máquina tiene una "dirección" en memoria
iii) la asignación de zonas de memoria para las variables, vectores, tablas matrices, etc.
iv) diccionarios de símbolos a los que el programa fuente hace referencia y que forman parte de otros módulos objeto

(Para hacer efectivos estos diccionarios es que trabaja el Editor de Vínculos.)

Etapa 3: Carga: el trabajo es cargado -no interesa cómo- en memoria central, de modo que es directamente ejecutable.