

PRODIN
003.3
G11

E



INFORME FINAL AVANCE DE PROYECTO

TRABAJO DE INVESTIGACION Y DESARROLLO :

"SIMULACION DE SISTEMAS DE DESARROLLO BASADOS EN MICROPROCESADORES"

DIRECTOR : ING. JOSE I. GALLARDO
AYUDANTE : DANIEL W. BERNIS

DEPENDENCIA : CIUNPAT-FAC.de INGENIERIA-DPTO. SISTEMAS-
UNIVERSIDAD NACIONAL DE LA PATAGONIA "S.J.BOSCO"

LUGAR DE TRABAJO : DPTO. SISTEMAS Y CENTRO DE COMPUTOS-
CIUDAD UNIVERSIATRIA- KM 4 -CRO. RIVADAVIA.

FECHA : 30 de DICIEMBRE DE 1989.



UNIVERSIDAD NACIONAL DE LA PATAGONIA SAN JUAN BOSCO BIBLIOTECA CENTRAL	
REGISTRO	34759
CLASIFICACION	PRODIN 003.3

G11

INFORME FINAL DE TRABAJO DE INVESTIGACION Y DESARROLLO

NOMBRE DEL PROYECTO : "SIMULACION DE SISTEMAS DE DESARROLLO
BASADOS EN MICROPROCESADORES"

DIRECTOR : Ing. Jose I. GALLARDO
AYUDANTE : Daniel W. BERNIS

UNIDAD EJECUTORA : DPTO. SISTEMAS-FAC. INGENIERIA-UNIV. NAC. PATAGONIA "S. J. BOSCO"

PERIODO INFORMADO : JULIO-OCTUBRE/89

FECHA APROBACION HCS: 19.10.87 RES. Nro. : 146/87

DURACION : OCT/87 a OCT/89 - 24 meses

FECHA FINALIZACION : 30.10.89

PRORROGAS OTORGADAS : Del 30.03.89 al 30.07.89 y del 30.07.89 al 30.10.89

LUGAR DE TRABAJO : Dpto. Sistemas y Centro de Computos-Ciudad Universitaria-
KM. 4-COMODORO RIVADAVIA-CHUBUT

- Comprende :
- Descripción (fs. 1 a 6)
 - Apéndice I de CONCLUSIONES (fs. 7 a 14)
 - Apéndice II "USO DE BIBLIOTECAS DE CODIGO"
(Apunte - 17 fs.)
 - Apéndice III Impresión listado de programa
"SIMULADOR de 8080 y 8085"
 - Apéndice IV Impresión listado de programa
"ENSAMBLADOR"

DESCRIPCION

=====

1) OBJETIVOS GENERALES

a) FINALIDAD:

La carencia de elementos didacticos para la ense#anza de la programacion assembler de microprocesadores de 8 y 16 bits, conocidos como sistemas de desarrollo necesarios para la ejecucion de trabajos practicos en las catedras de "Arquitectura de los Sistemas de Computo I" y "Microprocesadores y Microcomputadores", hace necesaria la implementacion en el sistema IBM/370, disponible en el Centro de Computos, de software de simulacion.

b) OBJETIVOS:

Tras el analisis de los microprocesadores a ser simulados, se pasara al estudio del lenguaje de programacion disponible, que se haya utilizado en proyectos similares por sus cualidades de portabilidad, modularidad y facil comprension por terceros, ademas de su adecuacion a los requerimientos especificos del presente proyecto.

Se encarara entonces, el desarrollo de un sistema de simulacion versatil en cuanto a sus prestaciones y posible de ser transportado a otras maquinas mas populares, pero respetando su fin cientifico. Se intentara lograr asi una simulacion de funcionamiento y arquitectura interna de los distintos procesadores, que refleje lo mas fielmente posible todas las transferencias de informacion que desencadena la ejecucion de una instruccion en el interior de una CPU.

Por ultimo, en la documentacion del sistema se recogeran las experiencias obtenidas en su desarrollo de modo tal que sirva como referencia para el estudio de metodologias de programacion para un tamafio inusual de sistema, para los alumnos de la Carrera de Licenciatura en Ciencias de la Computacion.

Asimismo consideramos que el trabajo a desarrollar sera una pieza de intercambio valida con otras universidades y centros tecnologicos que posean similares Centros de Computos.-

2) METAS PROPUESTAS

ETAPAS A CUMPLIR:

1. Analisis de microprocesadores (uP) a ser simulados
2. Estudio de lenguaje Pascal
3. Desarrollo de un simulador de uP 8080 y 8085 de Intel con entrada en codigo hexadecimal
4. Desarrollo de un programa ensamblador para los uP 8080 y 8085
5. Desarrollo de un simulador de uP Z80 de Zilog con entrada en codigo hexadecimal
6. Desarrollo de un programa ensamblador para el uP Z80
7. Desarrollo de un simulador de uP 8086 de Intel con entrada en codigo hexadecimal
8. Desarrollo de un programa ensamblador para 8086.

3) METODOLOGIA UTILIZADA

Se estudiaron primeramente pequeñas partes del problema central de cada etapa, para proponer luego las diversas soluciones que surgieron del análisis y tras un periodo de prueba, se paso a implementar la mas adecuada.

Para ello se previeron reuniones semanales de estimacion de propuestas donde se examinaron las opciones tanto desde el punto de vista del soft como del hardware. Implementado un modulo de programa, se procedio a probar lo dentro del contexto general con que se contaba hasta ese momento.

Se utilizo una metodologia de programacion bottom-up (de abajo hacia arriba), implementando pequeños procedimientos y funciones que se unieron para formar modulos cada vez mas complejos, facilitando asi la expansion del sistema a las necesidades y posibilidades.

Una vez completa la codificacion de todo el sistema, se hizo un repaso general para asegurar la integridad conceptual del sistema.

4) RESULTADOS

a) ACTIVIDADES DESARROLLADAS:

Tomando como referencia lo detallado en el Cuarto Informe de Avance de Proyecto de fecha 28.08.89, los avances producidos han sido los siguientes en cuanto al plan de Actividades presentado:

.ACTIVIDAD A : "Análisis de microprocesadores a ser simulados"

.Objetivo: Lograr un conocimiento mas cabado de los microprocesadores en estudio.

.Meta 1 : Estado 100% --> %Ejecucion Parcial= 11%

.ACTIVIDAD B : "Estudio de Lenguaje Pascal"

.Objetivo: Familiarizacion con la herramienta de software en el Centro de Computos.

.Meta 2 : Estado 100% --> %Ejecucion Parcial= 11%

.ACTIVIDAD C : "Desarrollo de Simulador de 8080 y 8085 de 8 bits con entrada hexadecimal"

.Objetivo: Desarrollar primer simulador .Análisis de los microprocesadores en cuestion. Formalizacion de la experiencia adquirida.

.Meta 3 : Estado 100% --> %Ejecucion Parcial= 22%

ACTIVIDAD D+F+H : "Desarrollo de Ensamblador general para 8080 y 8085, 280 y 8086"

.Objetivo: Desarrollar una interfase mas familiar al usuario que el codigo hexadecimal.

.Metas 4+6+8 : Estado 100% --> %Ejecucion Parcial= 28%

b) RESUMEN DE AVANCE DE PROYECTO: De lo expuesto precedentemente, se logra el siguiente porcentaje de ejecucion:

11 + 11+ 22+ 28= 72% Segun Cronograma Original

15.28 +15.28 +30.55 +38.89=100% S/Reformulacion del 4to. Informe,

====>concluyendo con las metas fijada

s en nuestro Cuarto Informe de fecha 28.08.89 , donde preveiamos este punto como final del proyecto.-

5) DISCUSION Y CONCLUSIONES

5.1-Como expusieramos en nuestro Tercer y Cuarto Informe de Avance de Proyecto, entendemos que respecto al cronograma original se han producido retrasos en el estado de avance. Ello se debe a :

- i) Inicio de las actividades con seis meses de atraso respecto del cronograma del proyecto (Mayo/87),
- ii) No disponibilidad operativa del Centro de Computos en un 100% debido a cortes por fallas del sistema en reiteradas oportunidades.
- iii) Un calculo erroneo en la estimacion del tiempo necesario para la realizacion del proyecto, previsto originalmente en 18 meses.

5.2-Al final del proyecto se cuenta con un sistema de software de unas 10000 lineas de codigo, lo que constituye un logro muy importante en cuanto al desarrollo de soft de base en nuestra universidad y aun en nuestro pais.

5.3-Entendemos que se deben priorizar y fomentar este tipo de trabajos de investigacion ,considerando que nuestro despegue tecnologico como pais en vias de desarrollo tendra uno de sus "nichos" en el area informatica como productor de valor agregado de software y no de hardware ,donde los paises desarrollados nos llevan a#os de ventaja.

5.4-Durante el desarrollo de la programacion del sistema se probaron distintas tecnicas y estilos de programacion y codificacion , lo que incremento sustancialmente nuestra experiencia de programadores. Esto, sumado a lo extenso del proyecto, dio pie a que distintas secciones del codigo generado difieran en tecnica, estilo y calidad.

Consideramos provechosa la divulgacion, luego de su correspondiente evaluacion, de las conclusiones extraidas sobre los siguientes temas, cuya ampliacion se anexa como APENDICE I de CONCLUSIONES, a saber:

-
- . Reglas de Codificacion de Programas
 - . Metodo de Dise#o de Sistemas de Programas
 - . Uso de Bibliotecas de Codigo
 - . Generacion y uso de Diccionario de datos
 - . Programacion de Automatas
 - . Programacion de Ensambladores.-

6) BIBLIOGRAFIA

- a. Fundamentos de Computadoras Digitales- Thomas C. Bartee-Edit. Mc Graw Hill.
- b. Fundamentos de los microprocesadores - Roger L. Tokheim-Edit. Mc Graw Hill.
- c. Microprocesadores: Arquitectura, Programacion y Desarrollo de Sistemas -Jose M. Angulo Usategui-Edit. Paraninfo.
- d. Organizacion de Computadoras. (2da. Edicion)-Hamacher, Vranesic y Zaky-Edit. Mc Graw Hill.
- e. Pascal Introduccion -J. Welsh y J. Elder-Edit. Prentice Hall.A
- f. Manual del Usuario Pascal/VS IBM Sistema 370.
- g. Complejidad- Gabriel Baum- Edit. Kapeluz.
- h. Manual del Usuario Sistema de desarrollo CODAS 09.
- i. Un emulador circuital del 8031-George Dinwiddie-Revista BYTE-Julio/86.
- j. Emulador ZMU-80- Robert Baumann- Revista BYTE- Octubre/86.
- k. Comunicacion entre computadoras en un ambiente de desarrollo con microprocesadores-E. Martinez, M. Romeo y O. Rosso-Revista Telefonica Electronica-Octubre/85.

7) PUBLICACIONES EMERGENTES DEL PROYECTO

Al presente se cuenta con la siguiente documentacion:

- a) Informes de Avance de Proyecto.
- b) Apunte en uso en la Catedra "Metodologia de programacion y Estructuras de datos"

Titulo: "Uso de bibliotecas de codigo" **APENDICE II. (17 fs.)**

8) TRANSFERENCIA

Ninguna al presente. Una vez elaborada la documentacion respectiva se estima poder transferirla a otras Universidades que cuentan con equipo IBM/370 similar al que cuenta la nuestra.

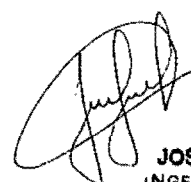
9) ACTIVIDADES FORMATIVAS DEL PERSONAL

La formacion del personal, en base a la experiencia recogida, se limito a:

- . analisis, dise#o y programacion de sistemas de desarrollo de microprocesadores,
- . adquisicion de conocimientos varios sobre microprocesadores,
- . lenguajes de programacion,
- . tecnicas y metodos de programacion y,
- . detalles operativos del sistema IBM/370.

10) OTROS ELEMENTOS DE JUICIO

Segun el libro "El arte de probar el software" de Myers la prueba de un programa solo puede detectar la presencia de errores y jamas su ausencia. Por lo tanto es imposible garantizar la correccion completa de un sistema de software como el presente, a pesar de haber sido testeado intensivamente en sus etapas de desarrollo. Por este motivo proseguiremos su verificacion despues de la entrega del presente informe.-



JOSE I. GALLARDO
INGENIERO ELECTRONICO

APENDICE I DE CONCLUSIONES

1) Reglas de codificación de programas .

La lectura de programas implica un esfuerzo intelectual considerable , debido a la longitud del código y a las multiples relaciones e interacciones que distintos módulos presentan entre si . Es posible reducir tal esfuerzo si se adoptan reglas para la codificación de programas . No importa tanto que reglas son como que existan reglas . Las reglas elaboradas tras la lectura del código generado por el proyecto son las siguientes :

1.1) Reglas de extensión :

- a) La longitud máxima óptima de declaraciones de tipos y variables es de aproximadamente 6 líneas (el código de declaración de un tipo o una variable debe caber en aproximadamente 6 líneas) ;
- b) La longitud máxima óptima de declaraciones de funciones , procedimientos y bloques de código es de una pantalla (el código debe ser completamente visible en una pantalla) ;
- c) La longitud máxima de declaraciones de tipos y variables es de una pantalla (el código de declaración de un tipo o variable no debe exceder una pantalla) ;
- d) La longitud máxima de declaraciones de funciones , procedimientos y bloques de código no debe exceder una hoja de impresión ;
- e) El ancho máximo del código no debe exceder el ancho de la pantalla ;

1.2) Reglas de indentado :

- a) El código se indenta para indicar el alcance de una constante , tipo , variable , función , procedimiento o bloque de código ;
- b) El encabezado de programa debe comenzar en la primer columna de la pantalla ;
- c) Las palabras claves CONST , TYPE y VAR deben comenzar en la misma columna donde comienza el encabezado del programa , procedimiento y función al que pertenezcan ;
- d) Las declaraciones de constantes , tipos y variables se deben desplazar dos columnas hacia la derecha respecto de las palabras claves correspondientes .
- e) Los símbolos = en las declaraciones de constantes y tipos y los símbolos : en las declaraciones de variables se deben alinear en una misma columna .
- f) los símbolos ; en las declaraciones de constantes , tipos y variables se deben alinear en una única columna ;
- g) Las palabras claves PROCEDURE y FUNCTION deben comenzar dos columnas hacia la derecha respecto del programa , procedimiento y función que las contenga ;

APENDICE I DE CONCLUSIONES

h) las sentencias iterativas se deben indentar asi

```
REPEAT
    sentencia1 ;
    sentencia2 ;
    ...
    sentenciaM ;
UNTIL ( condicion1 ) op
      ( condicion2 ) op
      ...
      ( condicionN ) ;

WHILE ( condicion1 ) op
      ( condicion2 ) op
      ...
      ( condicionN ) DO

BEGIN
    sentencia1 ;
    sentencia2 ;
    ...
    sentenciak ;
END ;

FOR i := V1 TO V2 DO
BEGIN
    sentencia1 ;
    ...
    sentencia2 ;
END ;
```

j) las sentencias selectivas se deben indentar asi

```
IF ( condicion1 ) op
    ...
    ( condicionM )
    THEN
    BEGIN
        sentencia1 ;
        ...
        sentenciak ;
    END
ELSE
BEGIN
    sentencia1 ;
    ...
    sentenciaJ ;
END ;
```

APENDICE I DE CONCLUSIONES

```
CASE variable OF
  V1 : BEGIN
    sentencial ;
    ...
    sentencial ;
  END ;
  ...
  VN : BEGIN
    sentencial ;
    ...
    sentencial ;
  END ;
  ELSE BEGIN
    sentencial ;
    ...
    sentencial ;
  END ;
END ;
```

- k) Las sentencias secuenciales se deben indentar asi

```
BEGIN
  sentencial ;
  ...
  sentencialK ;
END ;
```

1.3 Reglas de anidamientos :

- a) Cuando existen anidamientos , sus longitudes jamás deben superar la longitud de la pantalla .
- b) Cuando se aniden procedimientos y funciones , se deben separar con comentarios que contengan una línea del dígito correspondiente al nivel de anidamiento .

1.4 Reglas de comentarios :

- a) Los programas , segmentos y unidades (módulos) deben tener inmediatamente luego de su encabezado un comentario sobre versión , fecha de codificación , modificaciones , entradas , salidas , acciones y convenciones .
- b) Los procedimientos y funciones deben tener inmediatamente luego de su encabezado un comentario sobre sus entradas , salidas , acciones y convenciones .
- c) Cuando se use una variable global en un procedimiento o función , debe haber un comentario que repita si es posible la declaración , y que diga donde encontrarla . También en el lugar donde este declarada , debe haber un comentario sobre las convenciones de uso de dicha variable , y los nombres de los módulos que están autorizados a acceder a dicha variable (si es posible)

2) Método de diseño de sistemas de programas .

Durante un proyecto de software es imprescindible contar con un método de diseño , para poder generar un sistema que responda a sus especificaciones razonablemente libre de inconsistencias y errores . (Notese que es imposible asegurar que un programa sea libre de inconsistencias y errores) . Una característica del diseño de programas es la necesidad de revisar el trabajo ya realizado y retroceder y reelaborarlo si es necesario . Aquí se esboza uno de los posibles métodos de diseño .

La gente usa objetos en un ambiente determinado . Necesita conocer distintos datos sobre dichos objetos y sobre el ambiente que los rodea , y disponer dichos datos en forma determinada . Estas necesidades son las que determinan las especificaciones de un sistema de manejo de información . Este manejo de información puede hacerse manualmente , en papeles ordenados en carpetas , o automáticamente , en computadoras y sus sistemas de soporte de información .

El sistema debe contar con subsistemas de entrada , almacenamiento , recuperación , proceso y salida de información . Para poder diseñar estos subsistemas e integrarlos entre si es necesario conocer la información a ser manejada , pues esta debe estar "impresa" en el código de los programas (en lenguajes procedurales) , en forma de pantallas de captura y visualización de información , archivos de almacenamiento , formatos de impresiones , interfases , etc . Así , los requisitos previos del diseño son :

- a) un análisis detallado y formal de la gente , los objetos y el ambiente cuya información se maneja ;
- b) un análisis detallado y formal de los inconvenientes en el manejo actual de la información (si posible) ;
- c) un análisis detallado y formal del ambiente del programa , sus usuarios y sus necesidades ;

Con estos análisis disponibles , se definen las salidas deseadas del sistema (tipo y formato de la información , tiempo disponible para su procesamiento , forma de visualización de datos , etc) . En esta etapa debe ponerse el acento en definir las salidas de tal forma que satisfagan totalmente las necesidades de los usuarios del sistema . En este punto es aconsejable una revisión y reelaboración de lo hecho , con el objetivo de unificar criterios , encontrar posibles errores de concepto , y lograr un correcto entendimiento entre programador y usuario .

La siguiente etapa es definir que información es necesaria para poder obtener las salidas definidas anteriormente , a través de búsquedas en archivos , como por medio de cálculos . Para una mejor visualización de los resultados de este paso es conveniente disponer en dos columnas rotuladas entradas y salidas las entradas y salidas del sistema , y luego unirlos por flechas . Aquí se puede detectar la falta de datos de entrada y la existencia de cálculos previos comunes a dos o mas salidas . En este

APENDICE I DE CONCLUSIONES

punto se debe poder expresar cada una de las salidas como función de una o mas entradas . Si existen entradas sin salidas , o salidas sin entradas se debe revisar todo el trabajo efectuado hasta el momento .

Para modularizar el sistema , se agrupan las salidas con entradas comunes . Si se obtiene un número grande de tales grupos , es un indicio de que el sistema en diseño esta intentando abarcar una "visión del mundo" demasiado grande , siendo posible y aconsejable su subdivisión .

Si se intenta la subdivisión del sistema , un método de documentación como el diccionario de datos (v.a.c) es esencial para mantener la compatibilidad de convenciones y técnicas de programación .

Si todo parece bien , se generan los casos de prueba con sus correspondientes resultados .

En este punto se debe haber generado la siguiente documentación :

- a) definición de las salidas del sistema ;
- b) definición de las entradas del sistema ;
- c) definición de relaciones entre entradas y salidas del sistema ;
- d) casos de prueba y resultados .

Aquí se debe analizar si las entradas y salidas definidas no generan problemas sustitutos de los problemas originales . A continuación se definen las formas de almacenar y recuperar la información , teniendo en cuenta los requerimientos de los usuarios en tiempo , los procesos y calculos a efectuar y las prestaciones del equipo disponible .

En este punto es aconsejable presentar un informe de avance a los usuarios para que den su opinión sobre el trabajo .

Si estos dan su visto bueno , se continua definiendo la interfase de entrada y salida de datos . Esta parte se debe hacer con la estrecha colaboración del usuario final . Esta parte del sistema y la rapidez y exactitud con que responda serán los principales motivos de aceptación por parte del usuario . Aquí se debe escribir una versión del manual del usuario para uso como referencia en la codificación de entradas , salidas , y procesos y para que el usuario pueda tener una visión del sistema cercana a la definitiva .

En cuanto al diseño del código es aconsejable emplear la técnica de pseudocódigo con refinamiento progresivo ampliamente explicada en la referencia [1] , con una gráfica del arbol de invocaciones de procedimientos y funciones . Es aconsejable vigilar las alturas de las distintas ramas del arbol , que deben ser similares entre si . De lo contrario , pueden existir módulos monolíticos difíciles de entender por su longitud (ramas cortas) , o módulos con excesivas invocaciones (ramas largas) . Cada módulo , procedimiento y función debe documentarse prolijamente (que entradas y salidas tiene , que transformación realiza , como lo hace , que convenciones usa , modificaciones , etc) . Cada módulo debe compilarse por separado , y si es posible tambien probarse por

APENDICE I DE CONCLUSIONES

separado , antes de integrarse con otros módulos .
Es necesario también generar y usar un diccionario de datos o diccionario del sistema donde se puedan encontrar por diversos ordenes distintos detalles de diseño y uso .
Una vez integrados los distintos módulos , se debe probar el conjunto con los diversos casos de prueba generados oportunamente . Es importante recalcar que periódicamente se debe revisar la labor efectuada hasta el momento y retroceder para una reformulación si es necesario , pues la única forma de obtener un producto confiable y de calidad es supervisando todas las etapas de su diseño y producción.

3) Uso de bibliotecas de código .

En un sistema de software es importante mantener un ordenamiento del código para disminuir su complejidad de lectura e incrementar su facilidad de uso . Esto se puede lograr con la utilización de bibliotecas de código , tema sobre el cual se ha redactado un apunte , en uso en la cátedra METODOLOGIA DE PROGRAMACION Y ESTRUCTURAS DE DATOS, que se anexa como APENDICE II (solo el original) .

4) Generación y uso de diccionario de datos .

Un diccionario de datos es el documento de un sistema donde se pueden encontrar definiciones y comentarios de las distintas partes , y acceder a tal información de distintas formas . Su uso es absolutamente necesario , aun con sistemas pequeños , para evitar inconsistencias , duplicación de funciones , código y nombres , mantener la compatibilidad entre distintos módulos y mantener la coordinación de un proyecto ejecutado por un grupo de personas .

La estructura mas conveniente para el diccionario depende del grado de avance del proyecto : en las primeras etapas del proyecto , es suficiente los distintos datos e informaciones listados por orden alfabético de sus identificadores , nombres o títulos . En etapas mas avanzadas , cuando se va delineando la estructura del sistema , es aconsejable un reordenamiento por módulo o subsistema , manteniendo un índice alfabético para todo el sistema .

5) Programación de autómatas .

Cuando se desea verificar la entrada y salida de datos de un programa , ya sea para mantener la consistencia de información como para evitar la aparición de errores que aborten la ejecución del programa , se recurre al uso de autómatas o máquinas analizadoras de cadenas de información que deciden sobre la pertenencia o no de la cadena a un conjunto determinado . Por ejemplo , en un programa que deba leer fechas , se debe decidir si la fecha ingresada es o no correcta , y si no es correcta y el usuario lo desea volver a leerla . En el caso de verificación de la salida del programa , el autómata analiza la información sobre el estado de los dispositivos de salida , y decide si están en el o los estados correctos para una salida sin errores . La teoría de autómatas puede ser vista en detalle en el libro de Gabriel Baum "Complejidad" de editorial Kapeluz , usado en la II EBAI , y en general en cualquier libro sobre complejidad .

En el proyecto realizado , se usaron varios autómatas . En este escrito se discuten sus implementaciones . Básicamente un autómata queda definido por un conjunto de cadenas A de entrada , un conjunto de cadenas B de salida , un conjunto de estados C , un estado inicial c_0 un subconjunto de estados finales C' y una tabla de transiciones de estado o por un conjunto de reglas que definan cuales son las cadenas de entradas válidas . El autómata decide (no en todos los casos) si la cadena de entrada pertenece o no a un subconjunto A' previamente determinado . El autómata consigue esto tomando partes de la cadena de entrada , cambiando de estado según la tabla de transiciones en función de su estado y entrada actual , hasta llegar a uno de los estados finales y detenerse (El resultado esta dado por el estado final y/o las cadenas de salida) o por aplicación de las reglas mencionadas .

La implementación de un autómata puede hacerse "impresa" en el código , con sentencias CASE para la tabla de transiciones (ej : el procedimiento Execute del simulador) sobre una matriz (ej : la función OP del ensamblador) o con funciones de tipo lógico (ej : el ensamblador) . Cada una de las implementaciones tiene ventajas y desventajas . La implementación "impresa" es la mas veloz en ejecución , pues la tabla de transiciones queda codificada en el programa , pero es difícil de modificar , y si la tabla es grande , el procedimiento o la función que contiene al autómata excede largamente una hoja impresa de longitud . El autómata implementado sobre matriz es muchísimo mas sencillo de modificar , y puede ser modificado sin recompilar el programa , pero es mas difícil de codificar . Implementado con funciones es mucho mas sencillo de codificar e inteligilible , y es la forma mas cómoda de programarlo : sin embargo , se corre el riesgo de usar excesivamente variables globales (por comodidad) y asi oscurecer el código .

APENDICE I DE CONCLUSIONES

6) Programación de ensambladores .

Una conclusión del proyecto , es que difícilmente el módulo del ensamblador pueda ser parametrizado para procesar un programa para un microprocesador con una estructura diferente a los mp. del trabajo , sin cambios en su codificación . Esto se debe particularmente a los modos de direccionamiento . Aun así los cambios solo se ven limitados a la Function OP .

APENDICE II DE CONCLUSIONES: "USO DE BIBLIOTECAS DE CODIGO" (Apunte)

PROGRAMACION MODULAR

Un problema

En programación, es de importancia fundamental limitar lo que llamaremos la complejidad del texto, o simplemente complejidad, de los programas. Definimos complejidad como la dificultad de lectura, entendimiento y modificación del texto de un programa. En programas pequeños, una adecuada división del código fuente en procedimientos y funciones, y un diseño elemental de las estructuras de datos es suficiente para mantener la complejidad en límites manejables. Sin embargo en programas de cierta extensión, y en sistemas de programas la complejidad se incrementa muy rápidamente con la cantidad de líneas de código. La simple cantidad de estructuras de datos, funciones y procedimientos complica la lectura de un programa.

Ejemplos del problema

Supongamos que cada definición de una estructura de dato, un procedimiento o función ocupa una página impresa (55 líneas; lo ideal sería 20 ~ 25 líneas para que se vea en una pantalla común). Un abm tiene la definición de la estructura de datos, un procedimiento de lectura de datos, funciones de validación de datos, funciones de validación de registros, funciones de búsqueda, procedimientos de modificación y baja, y eventualmente procedimientos de acceso a archivos y aparcado de archivos (~ 10 hojas impresas). Un sistema puede tener 3 abm, los correspondientes procedimientos de listados, backup, y menú de acceso (~ 40 hojas impresas). Aun en un sistema mínimo como este, se deben revisar 40 hojas para modificar algo; aparte de la correspondiente documentación. Una modificación de una estructura de datos implicaría modificar por lo menos 2 programas distintos. Aquí vemos que el problema es el código compartido entre programas: si fuera posible mantener el código compartido en un solo lugar, las modificaciones insumirían menos tiempo y habría menor probabilidad de error. Supongamos que desarrollamos otro sistema, que también tiene menús. Nos gustaría poder usar el código de menú que ya escribimos. También nos gustaría mantener la apariencia de todos nuestros sistemas, así como las funciones de interface asignadas a cada tecla. Aquí también tenemos el problema de compartir código, bajo una forma distinta: nos interesa la función que dicho código implementa.

La solución: programación modular.

Se propone como solución un desarrollo estructurado de programas, basado en la descomposición de programas en módulos de programa. Cada módulo es un conjunto de declaraciones de estructuras de datos y procedimientos y funciones relacionadas. Los procedimientos y funciones de cada módulo cooperan entre sí para asegurar la gestión de un mismo conjunto de informaciones o datos.

Los datos de dicho conjunto son accedidos por otros módulos o partes del programa, utilizando los procedimientos y funciones que integran el módulo correspondiente. La solución al problema planteado con anterioridad queda definida así:

nombre del módulo	comentario
utilidades	procedimientos, funciones y variables globales
menu	definición de menús y manejador de menús
basedato	definición base de datos del programa
archivos	definición de archivos (accesos, índices, etc)
entrada	procedimientos de entrada de datos
validacion	funciones de validación de registros
abm	procedimientos de alta, baja y modificación
consultas	procedimientos de respuesta a consultas

impresión. procedimientos de impresión de registros
Con todos estos módulos es muy fácil armar un sistema, e incluso hay algunos que pueden ser comunes a muchos sistemas, como utilidades, menú e incluso abm.

Se hace notorio un inconveniente de esta metodología: se debe documentar cuidadosamente la interface (los parámetros de funciones y procedimientos de cada módulo, o en otras palabras, la manera de intercambiar datos) para evitar retardos en la codificación por dudas o correcciones de errores. Es evidente que una modificación en un módulo repercute en el resto del código del sistema pero ahora es obvio, por ejemplo, que un cambio en el módulo de impresión no tiene porque afectar al código de entrada o al de menú, lo que circunscribe el problema de revisión y compatibilización de código a un área mucho mas pequeña. A continuación se desarrollarán los fundamentos de la programación modular, y se expondrán dos implementaciones de PASCAL que soportan herramientas para un enfoque modular de programación. Identicas herramientas se encuentran en el standard ANSI del lenguaje C, que es respetado por casi todas las versiones.

Definiciones y reglas de la programación modular

Se define un módulo como un conjunto de procedimientos y funciones de acceso y de gestión de ciertas estructuras de datos: el acceso a tales estructuras de datos y la modificación de sus contenidos solo será posible a través de los procedimientos y funciones del módulo, estando vedado a procedimientos y funciones de otros módulos.

Un punto para aclarar es que preferentemente en los módulos solo se declaran estructuras (los tipos de datos del usuario en PASCAL y C) pero no variables globales. Pueden declararse variables locales a procedimientos y funciones usando una estructura de datos (un tipo de variable en PASCAL o C) de otros módulos la exigencia de la programación modular es que en tal caso, se usen los procedimientos y funciones del módulo correspondiente para el acceso a los valores de tales variables.

La interface de un módulo como el modo de intercambiar información con el. En el caso de trabajar en PASCAL o C, la interface de un módulo queda formada por:

- las declaraciones de los procedimientos y funciones (solamente es necesario tener en cuenta los encabezados de procedimientos y funciones, sin prestar atención al cuerpo de procedimientos y funciones, implementación de algoritmos, variables locales),
- los tipos de variables definidos, y
- las variables globales y estáticas.

La descomposición de un programa en módulos debe lograrse en base a los datos a ser manejados, haciendo un inventario de todos los datos y agrupandolos con criterio lógico. Para que un módulo sea fácil de entender, es necesario que sus estructuras de datos no sean muy diversas ni complejas, y por consiguiente solo deben ser una parte reducida del conjunto total de las estructuras utilizadas en el programa. Además, en el análisis, diseño y codificación de un módulo que use procedimientos y funciones de otros módulos, solamente se debe conocer la interfase de dichos módulos, sin necesidad de preocuparse por su implementación.

Un punto para aclarar es que preferentemente en los módulos solo se declaran estructuras (los tipos de datos del usuario en PASCAL y C) pero no variables globales, porque en tal caso la acción de procedimientos y funciones puede depender de la historia de invocaciones (que alteran el contenido de las variables globales).

Un ejemplo de programación modular:

PROBLEMA

Se desea escribir un programa que lea e imprima una lista de datos sobre libros como se detalla a continuación:

PARTE DE LA SOLUCION

En letra **BOLD** las interfaces de los módulos.

(VERSION TURBO PASCAL 4.0 y 5.0)

const

type

libro = record

```
editor : tema : string [ 80 ] ;
```

codigo	0	1.000
--------	---	-------

cedure imprimir_mascara libro :

got

```
gotoxy ( 5 , 3 ) ; write ( ' titulo' , ' ' ) ;
```

```
gotoxy ( 5 , 5 ) ; write ( 'tema' , ' ' ) ;
```

```
gotoxy ( 5 , 7 ) ; write ( 'codigo' , 1 ) ;
```

```
procedure leer_campos_libro (var un libro : libro;
                             var un campo : campo);
```

begin

begin

```
gotoxy ( 14 , 3 ) ; read ( titulo )
```

```
gotoxy ( 14 , 5 ) ; read ( tema ) ;
```

gotoxy (14 , 7) ; read (código) ;

... ..

begin

begin

```
gotoxy ( 14 , 3 ) ; write ( titulo )
```

```
gotoxy( 14 , 5 ) ; write ( tema ) ;
```

```
gotoxy ( 14 , 7 ) ; write ( código )
```

[illegible]

```

procedure leer_libro ( var un_libro : libro ) ;
begin

```

```

    clrscr ;
    gotoxy ( 25 , 1 ) ;
    write ( 'lectura de datos de libro' ) ;
    imprimir_mascara_libro ;
    leer_campos_libro ( un_libro ) ;
end ;

```

```

procedure mostrar_libro ( var un_libro : libro ) ;
begin

```

```

    clrscr ;
    gotoxy ( 25 , 1 ) ;
    write ( 'datos de libro' ) ;
    imprimir_mascara_libro ;
    mostrar_campos_libro ( un_libro ) ;
end ;

```

```

FIN módulo LIBRO )

```

```

((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((
A continuación escribimos el módulo lista :

```

```

( módulo LISTA )

```

```

type

```

```

    enlace = ^nodo ;
    nodo = record
        un_libro : libro ;
        siguiente : enlace ;
    end ;

```

```

procedure agregar_nodo ( una_lista , un_enlace : enlace ) ;
begin

```

```

    if ( una_lista <> nil )
    then
        un_enlace^.siguiente := una_lista ;
        una_lista := un_enlace ;
    end ;

```

```

procedure crear_nodo
( var un_enlace : enlace ; var el_libro : libro ) ;

```

```

begin
    if ( freemem > sizeof ( nodo ) )
    then
        begin
            new ( un_enlace ) ;
            un_enlace^.un_libro := el_libro ;
        end
    else

```

```

        un_enlace := nil ;
    end ;

```

```

procedure mostrar_lista ( una_lista : enlace ) ;
begin

```

```

    while una_lista <> nil do
        begin
            mostrar_libro ( una_lista^.un_libro ) ;
            repeat until keypressed ;
            una_lista := una_lista^.siguiente ;
        end ;
    end ;

```

```

function nodo_valido ( un_enlace : enlace ) : boolean ;
begin

```

```

    nodo_valido := un_enlace <> nil ;
end ;

```

```

((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((
Ahora solo falta el módulo principal :

```



```

procedure leer_libro ( var un_libro : libro ) ;
begin
  cms ( 'clrscr' , rc ) ;
  write ( 'lectura de datos de libro' ) ;
  leer_campos_libro ( un_libro ) ;
end ;
procedure mostrar_libro ( var un_libro : libro ) ;
begin
  cms ( 'clrscr' , rc ) ;
  write ( 'datos de libro' ) ;
  mostrar_campos_libro ( un_libro ) ;
end ;
( FIN módulo LIBRO )
((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((
A continuación escribimos el módulo lista :
( módulo LISTA )
type
  enlace = @nodo ;
  nodo   = record
    un_libro : libro ;
    siguiente : enlace ;
  end ;
procedure agregar_nodo ( una_lista , un_enlace : enlace ) ;
begin
  if ( una_lista <> nil )
  then
    un_enlace.siguiente := una_lista ;
    una_lista := un_enlace ;
  end ;
procedure crear_nodo
  ( var un_enlace : enlace ; var el_libro : libro ) ;
begin
  new ( un_enlace ) ;
  un_enlace.un_libro := el_libro ;
end
end ;
procedure mostrar_lista ( una_lista : enlace ) ;
begin
  while una_lista <> nil do
  begin
    mostrar_libro ( una_lista.un_libro ) ;
    readln ;
    una_lista := una_lista.siguiente ;
  end ;
end ;
function nodo_valido ( un_enlace : enlace ) : boolean ;
begin
  nodo_valido := un_enlace <> nil ;
end ;
((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((
Ahora solo falta el módulo principal :

```

```

{ módulo PRINCIPAL }
var
  auxiliar ,
  la_lista : enlace ;
  el_libro : libro ;
  respuesta : char ;
  fin      : boolean ;
begin { programa }
  repeat
    leer_libro ( un_libro ) ;
    crear_nodo ( auxiliar , un_libro ) ;
    if nodo_valido ( auxiliar )
      then fin := true
    else
      begin
        agregar_nodo ( una_lista , auxiliar ) ;
        write ( 'desea terminar ?' ) ;
        read ( respuesta ) ;
        fin := respuesta in ( 's' , 'S' ) ;
      end ;
    until fin ;
  mostrar_lista ( una_lista ) ;
end . { programa }

```

Ahora el interrogante es como unir estos módulos en un programa que funcione efectivamente . Una posibilidad es concatenar el texto de los tres módulos en un solo archivo pero entonces las ventajas del diseño modular se pierden , pues para revisar o modificar un módulo primero se lo debe ubicar dentro del conjunto de módulos cayendo en la necesidad de leer otros módulos , lo que se desea evitar . Para conseguir resultados óptimos de la programación modular se deben usar las técnicas de inclusión de código fuente y la compilación independiente de módulos , que se describen a continuación . Los mejores compiladores soportan estas técnicas , con distintas implementaciones .

La inclusión de código fuente y las bibliotecas o librerías de código fuente

Debe crearse un archivo X con el código de un módulo , y después , dependiendo de la implementación , agregarlo a una librería de código fuente . Las bibliotecas o librerías de código fuente , son conjuntos de archivos conteniendo código de módulos a los que se aplica la técnica de inclusión . Cada librería tiene un nombre propio y permite agrupar código por función u origen , estructurando y organizando su uso .

Cuando se desea tener acceso al código de un módulo , se da un comando de inclusión de código junto con el nombre del archivo X a incluir , y el nombre de la librería en la que se encuentra , si corresponde . Cuando el compilador encuentra este comando , continúa la compilación con el archivo X como fuente . Al llegar al final del archivo X , retoma la compilación del archivo donde encontro el comando de inclusión . Los comandos de inclusión pueden o no anidarse . Cuando se pueden anidar , existe un límite en la profundidad de anidación pues en estos casos el compilador guarda una pila con los nombres de los archivos incluidos , para saber donde retomar la compilación cuando encuentra un fin de archivo .

Los segmentos o unidades de programas y las bibliotecas o librerías de código objeto :

Cuando se desea que dos o mas programas compartan funciones y procedimientos comunes , se agrupan estas funciones y procedimientos

en módulos de programas , compilables por separado , que se pueden usar como bloques para la construcción de programas . Una vez compilados , estos bloques pueden guardarse en bibliotecas o librerías de código objeto , similares en uso e intención a las librerías de código fuente . Para unir distintos módulos a un programa , se usa un programa llamado linker , que según las implementaciones puede correr automáticamente después del compilador , o requerir una invocación explícita . El acceso a las bibliotecas de código objeto requiere el uso de una opción del linker .

Las implementaciones y las metodologías de trabajo correspondientes IBM 370

El sistema operativo VM/CMS de la IBM 370 , soporta el uso de librerías de código fuente , llamadas librerías de macros . Cada compilador que corra en este sistema operativo tiene su forma de acceso a la librería . Nos concentraremos en el compilador PASCAL . Las librerías son conjuntos particionados de datos (Partitioned Data Set) . Cada partición o miembro tiene un nombre mediante el cual accedemos a él .

creación de librerías de macro en IBM 370

Para poder crear una librería de macros debemos seguir los siguientes pasos :

a) Crear con el editor un archivo con el código fuente , de tipo COPY .

ejemplo :

x ejemplo copy al

Un detalle importante es la cantidad de columnas de este archivo : debe ser 71 columnas . Sino el sistema operativo no lo reconocerá como archivo válido . No debe escribirse más allá de la columna 71 , pues se perderá código .

b) Una vez salvado el archivo , podemos generar una librería nueva cuyo nombre sea primera .

Demos el siguiente comando :

maclib gen primera ejemplo

El resultado de este comando debe ser un archivo en el disco a primera maclib al

c) Para poder acceder a la librería debemos hacer varias modificaciones al archivo profile exec a0 y al archivo nn exec que usamos para compilar los programas pascal . (Los listados de los archivos modificados están en el apéndice A) .

Debemos agregar al archivo profile exec a0 la siguiente línea global maclib primera

con el cual declaramos nuestra librería , y la ponemos dentro del alcance del compilador PASCAL .

Si se desea agregar más librerías se deben escribir sus nombres a continuación de primera . Solamente se pueden tener disponibles 8 librerías , pero este no es un límite demasiado restrictivo .

Después de salvar la modificación del archivo profile exec a0 , pasamos a CP READ , con <alt><ap1> , damos el siguiente comando

I CMS

y presionamos dos veces <enter> .

Debemos también agregar algunas adaptaciones al archivo nn exec con el que compilamos . Aquí es preferible adoptar el listado sugerido en el apéndice A .

Así podremos usar y compilar código fuente contenido en librerías .

Agregado de nuevas macros

Supongamos que hemos creado los siguientes archivos ,

```
uno    copy al
dos    copy al
tres   copy al
cuatro copy al
```

y queremos convertirlos en miembros de la librería de macros

```
primera maclib al
```

que ya tiene el miembro

```
ejemplo
```

Usemos entonces el siguiente comando

```
maclib add primera uno dos tres cuatro
```

Directorio e impresión de miembros de una librería

El comando

```
maclist
```

nos dara una el directorio de miembros de una librería .

Si tipeamos

```
maclist primera
```

obtendremos el listado de los miembros pertenecientes a primera

```
maclib al
```

```
?
```

En la columna de comandos podemos dar el comando

```
print
```

para imprimir miembros de la librería .

Modificación de miembros contenidos en librerías

También en dicha columna podemos invocar al editor colocando una * al lado del miembro . Podremos modificar así el miembro , quitando o agregando líneas . Una vez terminada la modificación , debemos dar el comando

```
save
```

Debemos notar que el miembro modificado se guarda al final de la librería alterandose el directorio de la librería para recuperar la última versión del miembro . El miembro original no se borra y continúa en la librería hasta que esta se compacta con el comando

```
maclib comp primera
```

Si no compactamos regularmente una librería sujeta a frecuentes modificaciones , rápidamente ocupara el espacio disponible en disco .

Borrado de los archivos nn copy

Una vez agregados los archivos nn copy a la librería como miembros , se deben borrar , para no tener copias innecesarias de código , que nos pueden confundir .

Demos el siguiente comando

```
erase * copy al
```

Metodología de trabajo aconsejada

El límite de 8 librerías de macros usadas al mismo tiempo no es restrictivo en absoluto ; pues con 4 alcanza para todas las necesidades .

Se recomienda usar las siguientes librerías

- a) **structur maclib al**
en esta librería se guardan las estructuras de datos usadas ;
- b) **procfunc maclib al**
en esta , los procedimientos y funciones que operan sobre las estructuras de datos ;
- c) **progseg maclib al**
en esta , se guardan programas y segmentos de programa
(ver Segmentos y unidades de programa) ;
- d) **external maclib al**
en esta , declaraciones de procedimientos y funciones external ,
necesarias para el uso de segmentos de programa .

Inclusión de código fuente

Suponiendo la existencia de un miembro `strlibro` en una librería `structur maclib a1`, existen varias posibilidades de incluirlo en la compilación del programa `biblos`, usando un metacomando propio de esta versión de PASCAL.

1) se especifica explícitamente la librería

```
PROGRAM BIBLOS ( INPUT , OUTPUT , BIBLIOTECA ) ;  
  ZINCLUDE STRUCTUR (strlibro);  
  ---  
  ---  
BEGIN  
  ---  
  ---  
END .
```

2) se asume implícitamente una librería por defecto.

Esta opción es bastante incomoda pues hay que modificar el `exec` usado para invocar al compilador si se desea cambiar la librería usada.

```
PROGRAM BIBLOS ( INPUT , OUTPUT , BIBLIOTECA ) ;  
  ZINCLUDE strlibro ;  
  ---  
  ---  
BEGIN  
  ---  
  ---  
END .
```

Para poder compilar este programa se debe modificar el archivo `copascal exec` (ver apéndice a).

TURBO PASCAL

Trabajaremos en este apunte con la versión 5 de TURBO PASCAL, indicando la diferencia con la versión 4 en apartados cuando sea necesario.

TURBO PASCAL implementa las librerías de código fuente usando subdirectorios. La recompilación después de una corrección es muy ágil, debido a la opción `MAKE` de TURBO PASCAL 4 y 5, o el manejo de archivo principal y archivo de trabajo en TURBO PASCAL 3. En el menú principal de TP 4/5 al que se accede con `F10`, en `Options`, adonde se accede con `'O'`, se dispone de la posibilidad de especificar el subdirectorio donde se guarda el código fuente para inclusiones. Luego de dar el nombre del subdirectorio de inclusión, se puede guardar la nueva configuración con la opción `Save options`, adonde se accede con `'S'`. Así, durante el desarrollo de un proyecto, no existirá la necesidad de especificar continuamente el subdirectorio de inclusiones.

Inclusión de código fuente .

Para la inclusión de código fuente en TP en todas sus versiones se usa un metacomando del compilador, propio de esta versión de PASCAL.

Suponiendo la estructura de datos libro en un

archivo `c:\pascal\biblos\strlibro.inc` tenemos dos opciones:

a)

```
PROGRAM BIBLOS ( INPUT , OUTPUT , BIBLIOTECA ) ;  
  uses crt ;  
  ({$I c:\pascal\biblos\strlibro.inc })  
  ---  
  ---  
BEGIN  
  ---  
  ---  
END .
```

b) definir c:\pascal\biblos como subdirectorio de archivos para inclusión

```
PROGRAM BIBLOS ( INPUT , OUTPUT , BIBLIOTECA ) ;  
uses crt ;  
  (##I strlibro.inc #)  
--  
--  
BEGIN  
--  
--  
END .
```

Segmentos o unidades de programas : compilación independiente .

Un módulo (conjunto de declaraciones de constantes , tipos , variables , procedimientos y funciones) puede compilarse independientemente para formar un segmento o unidad (según la implementación) y formar así un bloque constructor de programas . Para producir un programa ejecutable se unen los segmentos o unidades necesarios previamente compilados . Un segmento o unidad puede unirse a distintos programas ejecutables , compartiéndose no solo el código fuente , sino también el código objeto , lograndose la optimización total de espacio .

● Aparecen algunos conceptos nuevos con el uso de segmento o unidades . El primero es el de código público y privado . El código público es aquel que podrá ser alcanzado por el programa principal y otros segmentos o unidades . El código privado es aquel que solamente es alcanzable dentro del segmento o unidad .

Otro es el de encapsulamiento de datos y sus estructuras : en un segmento o unidad pueden definirse estructuras de datos y los procedimientos y funciones que manejan tales estructuras como código privado , quedando como código público procedimientos y funciones que usan los procedimientos y funciones privados para insertar datos en las estructuras de datos definidas en aquel módulo .

Las implementaciones y las metodologías de trabajo correspondientes .

IBM 370 :

En esta implementación los módulos son llamados segmentos .

La estructura de un segmento es parecida a de un programa :

El tipo del archivo debe ser PASCAL ;

el encabezado comienza con la palabra 'Segment' seguida por el nombre del segmento que debe coincidir con el nombre del archivo ;

● Las declaraciones de constantes , tipos y variables globales son privadas ; para poder acceder a ellas en otro segmento o programa debe repetirse la declaración en el segmento o programa invocante .

Las declaraciones de procedimientos y funciones cuyo encabezado sea seguido de la palabra 'Entry' serán públicas y las demás serán privadas por defecto ;

Cuando se desee acceder a procedimientos y funciones públicos de otros segmentos , se escribira solamente el encabezado seguido de la palabra 'External' ;

Luego de la ultima declaración , debe ir la palabra 'end' .

Ejemplos

```
SEGMENT <Nombre> ;
CONST
  ZINCLUDE STRUCTUR (<cntNombre>); (* constantes del segmento *)
TYPE
  ZINCLUDE STRUCTUR (<strNombre>); (* estructuras del segmento *)
  ZINCLUDE EXTERNA (<pfXXXXXX>); (* proc. y func. de otros seg. *)
  ZINCLUDE PROCFUNC (<pfNombre>); (* proc. y func. del segmento *)
END .
SEGMENT <Nombre> ;
CONST
  ZINCLUDE STRUCTUR (<cntNombre>); (* constantes del segmento *)
TYPE
  ZINCLUDE STRUCTUR (<strNombre>); (* estructuras del segmento *)
  procedure externo ( var un_objeto : objeto ) ; external ;
  procedure publico ( var una_cosa : cosa ) ; entry ;
  procedure privado ( var una_cosa , otra_cosa : cosa ) ;
END .
```

Metodología de trabajo :

Se debe tener la estructura de datos en un miembro de la librería de macros structur maclib , e incluirla tanto en el segmento como en el programa principal , así cualquier modificación que se haga , será en un solo lugar ;
También los procedimientos y funciones , tanto públicos como privados, se deben tener en un miembro de procfunc maclib .
En la librería external maclib al se deberán colocar las declaraciones de procedimientos y funciones external ;
El código del segmento se deberá guardar como miembro en progsegm maclib .

TURBO PASCAL :

En TURBO PASCAL 5 los módulos se llaman units (unidades) de programa .

La estructura de las unidades es la siguiente .

```
UNIT <Nombre de la unidad>;
INTERFACE
  { declaración del código publico }
  USES < lista de nombres de unidades > ;
  { declaración de uso del código publico de las unidades
    nombradas }
  { se declaran los encabezados de los procedimientos y funciones
    publicos }
IMPLEMENTATION
  { declaración del código privado en su totalidad , y del
    encabezado y el cuerpo de los procedimientos y funciones
    publicos }
  USES < lista de nombres de unidades > ;
  { declaración de uso del código publico de las unidades
    nombradas }
  { se declaran los encabezados de los procedimientos y funciones
    publicos }
BEGIN
  { código de inicialización }
END .
```

Gracias a la opción MAKE de TP5 , se puede recompilar un programa completo con una sola tecla pulsada , si se modifica cualquiera de sus unidades .

El código de inicialización sirve para dar valores iniciales a constantes cuyos valores son desconocidos en tiempo de compilación:

este código es ejecutado antes de la primer sentencia de programa principal . Una aplicación perfecta del código de inicialización es la lectura de archivos de configuración del programa , detección de plaquetas gráficas , impresoras , etc .

Para mayor información , consultar apunte de TURBO PASCAL 5 .

Librería de código objeto

Una vez depurado y compilado un módulo puede incorporarse como miembro a una librería de código objeto . Esto simplifica el uso del compilador cuando se produce el código ejecutable .

Implementación y metodología de trabajo

IBM 370 :

El sistema operativo VM/CMS soporta el uso de librerías de código objeto que los compiladores pueden acceder para unir miembros a los módulos y programas que compilan . Estas librerías son conjuntos particionados de datos (partitioned data set) , en las que cada miembro tiene un nombre por el cual puede ser accedido .

Creación de un archivo de código objeto

Compilando un segmento x pascal , se genera un archivo x txt , que puede ser convertido en miembro de una librería de tipo txtlib .

Para eso , demos el siguiente comando

txtlib gen segunda x

que dara como resultado la creación de un archivo segunda txtlib al

El agregado de miembros a esta librería tipo txtlib se hace en forma similar a la de la librería tipo maclib

txtlib add segunda y z u v ...

Modificación de un miembro de librería tipo txtlib

Los miembros de una librería tipo txtlib son unidades de código objeto , por lo tanto , debe modificarse el código fuente , recompilar , y reemplazar el miembro correspondiente .

El siguiente comando

txtlib del segunda z

borra el miembro z de la librería segunda .

Con el comando:

txtlib add segunda z

agregamos la nueva versión de z .

Metodología de trabajo

A medida que se depuran segmentos o partes de programas , pueden incluirse como miembros de una librería tipo txtlib los archivos tipo txtlib correspondientes .

Al formarse los miembros de las librerías , pueden borrarse los archivos que les dieron origen , y así nos aseguramos que usamos siempre la última versión compilada . Los archivos \$ exec usados para compilar y ejecutar programas Pascal utilizan la librería tipo txtlib para generar los archivos tipo module ejecutables . Como este proceso es totalmente transparente solo es necesario copiar los listados del apéndice A .

TURBO PASCAL

El programa TPUMOVER.EXE se usa para agregar y quitar unidades a la librería de código objeto de TPS , TURBO.TPL . Una característica notable de todos los programas del sistema TURBO que requieren al menos un argumento en su invocación (en el caso de TPUMOVER.EXE el nombre de la unidad a ser agregada o quitada) es que si no se les da el argumento , su respuesta es una pantalla de ayuda , que especifica claramente su modo de uso .

Para mayor información consultar el apunte de TURBO PASCAL 5 .

Ejemplos :

- a) Con las herramientas presentadas, se resolvera el problema de unir los distintos modulos de BIELOS, en un programa ejecutable.

IBM 370

Una posible solución es :

- ```

1) crear los miembros correspondientes de MACLIBs con los
 fragmentos de código ya generados , con las modificaciones
 del caso .

2) crear las MACLIBs :
 maclib gen progseg deflibro deflista defprincipal programa
 maclib gen structur entlibro strlibro strlista
 maclib gen phocfunc pflibro pflista pfprincipal
 maclib gen externa exlibro exlista exprincipal

3) crear los archivos

```

archivo deflibro copy al

((((((

C VERSION VMPASCAL 3

{ módulo LIEBO }

```
segment deflibro :
```

const

```
%include structur (entlibros)
```

Type

```
%include structur (strlibros)
```

(\* procedimientos y funciones \*)

```
%include procfunc (pflibros)
```

and

C FIN módulo LIBRO 3

[illegible]

C. VERSION VMPASCAL 1

3. módulo LISTA 3

segment deflista a

type

```
%include structur (strlista)
```

(\* procedimientos y funciones \*)

```
%include procfunc (pflista)
```

end .

C. FIN módulo LISTA 3

[illegible]

C VERSION VMPASCAL 2

[ módulo principal ]

segment principa ;

työ

```
%include structur (strlibro)
```

(\* procedimientos y funciones \*)

```
%include externa (exlibro)
```

```
%include externa (exlista)
```

```
%include procfunc (pfprincipal)
```

end .

{ FIN módulo LIBRO }

=====

archivo entlibro copy al

```
largo_cadena = 80
```

)))))))))









## APENDICE III.

### IMPRESION PROGRAMA

"SIMULADOR" DE 8080

Nota: Se aclara que la impresora disponible en el Centro de Comput  
no imprime ciertos símbolos como: ; (punto y coma)  
7 y < (mayor y menor)  
: (dos puntos)  
etc.

PROGRAM SIMULA ( INPUT , OUTPUT , CPU BACKUP , MEM BACKUP )

CONST

LSHEXA = 2  
MSHEXA = 1  
LSBIT = 0  
MSBIT = 7  
BYTE NULO = 0  
PAGINA TOPE = 255  
MSB = 128  
(\* FLAGS \*)  
SIGNE = 128  
ZERO = 64  
AUX CARRY = 16  
PARITY = 4  
CARRY = 1

TYPE

BYTE SIMPLE = 0 .. 255  
TIPO INDICE HEXA = MSHEXA .. LSHEXA  
TIPO HEXA = PACKED ARRAY ( . TIPO INDICE HEXA . )  
OF CHAR  
TIPO INDICE BIT = LSBIT .. MSBIT

TIPO REGISTROS = ( ACC , FLAGS , TEMP , IR ,  
B , C , D , E , H , L ,  
PC HI , PC LO ,  
SP HI , SP LO ,  
MAR HI , MAR LO , MDR ,  
W , Z , REG ERROR )

TIPO CPU 8085A = ARRAY ( . TIPO REGISTROS . ) OF  
BYTE SIMPLE

TIPO MEMORIA = ARRAY ( . BYTE SIMPLE , BYTE SIMPLE . )  
OF BYTE SIMPLE

TIPO INCREMENTO = 0 .. 7  
TIPO BACKUP = FILE OF TIPO HEXA

TIPO MENSAJE = STRING ( 80 )

VAR

CPU TIPO CPU 8085A  
MEMORIA TIPO MEMORIA  
COM CMS INTEGER  
PASO A PASO ,  
ACARREO ,  
MEDIO ACARREO BOOLEAN

```
PROCEDURE CMS (CONST S STRING VAR RC INTEGER) EXTERNAL
VAR RC INTEGER
```

```
FUNCTION RESPUESTA (MENSAJE TIPO MENSAJE) BOOLEAN
CONST
```

```
SI = 'SI'
NO = 'NO'
```

```
VAR
 RESP STRING (2)
```

```
BEGIN
WRITELN (MENSAJE)
READ (RESP)
WHILE (RESP SI) AND (RESP NO) AND NOT EOF DO
BEGIN
 WRITELN (' CONTESTE SI O NO ')
 WRITELN (MENSAJE)
 READ (RESP)
END
RESPUESTA = (RESP = SI)
END
```

```
FUNCTION ANDEADO (BYTE1 , BYTE2 BYTE SIMPLE) BYTE SIMPLE
CONST
```

```
UNC = 1
BASE BIN = 2
```

```
VAR
 BYTE3 BYTE SIMPLE
 POTENCIA INTEGER
 INDICE BIT TIPO INDICE BIT
```

```
BEGIN
BYTE3 = 0
POTENCIA = 1
FOR INDICE BIT = LSBIT TO MSBIT DO
BEGIN
 IF ODD (BYTE1) AND ODD (BYTE2)
 THEN BYTE3 = BYTE3 + POTENCIA
 BYTE1 = BYTE1 DIV 2
 BYTE2 = BYTE2 DIV 2
 POTENCIA = POTENCIA * 2
END
ANDEADO = BYTE3
END
```

```
FUNCTION OREADO (BYTE1 , BYTE2 BYTE SIMPLE) BYTE SIMPLE
CONST
```

```
CERC = 0
BASE BIN = 2
```

```
VAR
 BYTE3 BYTE SIMPLE
 POTENCIA INTEGER
 INDICE BIT TIPO INDICE BIT
```

```
BEGIN
BYTE3 = 0
POTENCIA = 1
FOR INDICE BIT = LSBIT TO MSBIT DO
```

```

BEGIN
 IF ODD (BYTE1) OR ODD (BYTE2)
 THEN BYTE3 = BYTE3 + POTENCIA
 BYTE1 = BYTE1 DIV 2
 BYTE2 = BYTE2 DIV 2
 POTENCIA = POTENCIA * 2
END
CREADO = BYTE3
END

```

```

FUNCTION XDREADO (BYTE1 , BYTE2 BYTE SIMPLE) BYTE SIMPLE
CONST
 CERO = 0
 BASE BIN = 2
VAR
 BYTE3 BYTE SIMPLE
 POTENCIA INTEGER
 INDICE BIT TIPO INDICE BIT

```

```

BEGIN
 BYTE3 = 0
 POTENCIA = 1
 FOR INDICE BIT = LSBIT TO MSBIT DO
 BEGIN
 IF ODD (BYTE1) ODD (BYTE2)
 THEN BYTE3 = BYTE3 + POTENCIA
 BYTE1 = BYTE1 DIV 2
 BYTE2 = BYTE2 DIV 2
 POTENCIA = POTENCIA * 2
 END
 END
 XDREADO = BYTE3
END

```

```

FUNCTION COMPLEMENTO UNO (BYTE1 BYTE SIMPLE) BYTE SIMPLE
VAR
 INDICE BIT LSBIT .. MSBIT
 BYTE BYTE SIMPLE
 POTENCIA INTEGER
BEGIN
 POTENCIA = 1
 BYTE = 0
 FOR INDICE BIT = LSBIT TO MSBIT DO
 BEGIN
 IF NOT ODD (BYTE1)
 THEN BYTE = BYTE + POTENCIA
 BYTE1 = BYTE1 DIV 2
 POTENCIA = POTENCIA * 2
 END
 END
 COMPLEMENTO UNO = BYTE
END

```

```

FUNCTION SUMA (. BYTE1 , BYTE2 BYTE SIMPLE) BYTE SIMPLE
BEGIN
 SUMA = (BYTE1 + BYTE2) MOD 256
 MEDIO ACARREO =

```

```

 ((ANDEADO (BYTE1 , 15) + ANDEADO (BYTE2 , 15)) 9)
 ACARREO = (BYTE1 + BYTE2) 255

```

END

```

FUNCTION COMPLEMENTO DOS (BYTE1 BYTE SIMPLE) BYTE SIMPLE
BEGIN

```

```

 COMPLEMENTO DOS = SUMA (COMPLEMENTO UNO (BYTE1) , 1)
END

```

```

FUNCTION TEST PARIDAD PAR (UN BYTE BYTE SIMPLE) BOOLEAN
VAR

```

```

 A BOOLEAN
 I INTEGER

```

BEGIN

```

 A = FALSE

```

```

 FOR I = 0 TO 7 DO

```

```

 BEGIN

```

```

 A = A XOR ODD (UN BYTE)

```

```

 UN BYTE = UN BYTE DIV 2

```

```

 END

```

```

 TEST PARIDAD PAR = NOT A

```

END

```

PROCEDURE SET FLAG (FLAG BYTE SIMPLE)

```

BEGIN

```

 CPU (. FLAGS .) = OREADO (CPU (. FLAGS .) , FLAG)

```

END

```

PROCEDURE RESET FLAG (FLAG BYTE SIMPLE)

```

BEGIN

```

 CPU (. FLAGS .) = ANDEADO (CPU (. FLAGS .) ,
 COMPLEMENTO UNO (FLAG))

```

END

```

FUNCTION TEST FLAG (FLAG BYTE SIMPLE) BOOLEAN

```

BEGIN

```

 TEST FLAG = (FLAG = ANDEADO (CPU (. FLAGS .) , FLAG))

```

END

```

PROCEDURE S Z P ACTUALIZAR

```

BEGIN

```

 IF ANDEADO (CPU (. ACC .) , MSB) = MSB

```

```

 THEN SET FLAG (SIGNE)

```

```

 ELSE RESET FLAG (SIGNE)

```

```

 IF CPU (. ACC .) = 0

```

```

 THEN SET FLAG (ZERO)

```

```

 ELSE RESET FLAG (ZERO)

```

```

 IF TEST PARIDAD PAR (CPU (. ACC .))

```

```

 THEN SET FLAG (PARITY)

```

```

 ELSE RESET FLAG (PARITY)

```

END

```

PROCEDURE FLAGS ACTUALIZAR

```

BEGIN

```

 S Z P ACTUALIZAR

```

```

 IF ACARREO

```

```

 THEN SET FLAG (CARRY)
 ELSE RESET FLAG (CARRY)
 IF MEDIO ACARREC
 THEN SET FLAG (AUX CARRY)
 ELSE RESET FLAG (AUX CARRY)
END

```

```

PROCEDURE HEXAENTERO (HEXA TIPO HEXA
 VAR BYTE BYTE SIMPLE
 VAR BIEN BOOLEAN)

```

CONST

BASE HEX = 16

VAR

AUX      BYTE SIMPLE  
I      TIPO INDICE HEXA

BEGIN

BIEN = TRUE

AUX = 0

FOR I = MSHEXA TO LSHEXA DO

BEGIN

AUX = AUX \* BASE HEX

CASE HEXA ( . I . ) OF

'0'

'1'      AUX = AUX + 1

'2'      AUX = AUX + 2

'3'      AUX = AUX + 3

'4'      AUX = AUX + 4

'5'      AUX = AUX + 5

'6'      AUX = AUX + 6

'7'      AUX = AUX + 7

'8'      AUX = AUX + 8

'9'      AUX = AUX + 9

'A','a'      AUX = AUX + 10

'B','b'      AUX = AUX + 11

'C','c'      AUX = AUX + 12

'D','d'      AUX = AUX + 13

'E','e'      AUX = AUX + 14

'F','f'      AUX = AUX + 15

OTHERWISE BIEN = FALSE

END

END

IF BIEN

THEN BYTE = AUX

ELSE BEGIN

  WRITELN ( 'CONVERSION HEXA ENTERO FALLIDA' )

  BYTE = 0

  END

END

```

PROCEDURE ENTEROHEXA (BYTE BYTE SIMPLE
 VAR HEXA TIPO HEXA
 VAR BIEN BOOLEAN)

```

CONST

BASE HEX = 16

```

VAR
 I TIPO INDICE HEXA
BEGIN
 BIEN = TRUE
 FOR I = LSHEXA DOWNT0 MSHEXA DO
 BEGIN
 CASE (BYTE MOD BASE HEX) OF
 0 HEXA (. I .) = '0'
 1 HEXA (. I .) = '1'
 2 HEXA (. I .) = '2'
 3 HEXA (. I .) = '3'
 4 HEXA (. I .) = '4'
 5 HEXA (. I .) = '5'
 6 HEXA (. I .) = '6'
 7 HEXA (. I .) = '7'
 8 HEXA (. I .) = '8'
 9 HEXA (. I .) = '9'
 10 HEXA (. I .) = 'A'
 11 HEXA (. I .) = 'B'
 12 HEXA (. I .) = 'C'
 13 HEXA (. I .) = 'D'
 14 HEXA (. I .) = 'E'
 15 HEXA (. I .) = 'F'
 OTHERWISE BEGIN
 HEXA (. I .) = ' '
 BIEN = FALSE
 WRITELN ('CONVERSION ENTERO HEXA FALLIDA')
 END
 END
 END
 BYTE = BYTE DIV BASE HEX
END
END

```

PROCEDURE MODIFICAR

```

(
 PAGINA
 DESPLAZAMIENTO ,
 CONTENIDO BYTE SIMPLE
 VAR BIEN BOOLEAN)

```

```

BEGIN
 BIEN = TRUE
 IF PAGINA < PAGINA TOPE
 THEN BEGIN
 BIEN = FALSE
 WRITELN ('ACCESO ILEGAL A MEMORIA ')
 END
 ELSE
 MEMORIA (. PAGINA , DESPLAZAMIENTO .) = CONTENIDO
 END

```

PROCEDURE RECUPERAR

```

(
 PAGINA
 DESPLAZAMIENTO ,
 VAR CONTENIDO BYTE SIMPLE
 VAR BIEN BOOLEAN)

```

```

BEGIN
BIEN = TRUE
IF PAGINA > PAGINA TOPE
THEN BEGIN
 BIEN = FALSE
 WRITELN ('ACCESO ILEGAL A MEMORIA ')
 END
ELSE CONTENIDO = MEMORIA (. PAGINA , DESPLAZAMIENTO .)
END

```

```

PROCEDURE DISK DIRECCION (VAR ENTRADA TEXT
 VAR HEXHI , HEXLO TIPO HEXA
 VAR HI , LO BYTE SIMPLE
 VAR BIEN BOOLEAN)

```

```

BEGIN
BIEN = TRUE
READ (ENTRADA , HEXHI)
HEXAENTERO (HEXHI , HI , BIEN)
IF BIEN
THEN IF (HI = PAGINA TOPE)
 THEN BEGIN
 READ (ENTRADA , HEXLO)
 HEXAENTERO (HEXLO , LO , BIEN)
 END
 ELSE BEGIN
 BIEN = FALSE
 WRITELN ('ERROR LECTURA DE DIRECCION')
 END
END

```

```

PROCEDURE TECL PAGINA (VAR HEXPAGINA TIPO HEXA
 VAR PAGINA BYTE SIMPLE)

```

```

VAR
 OK BOOLEAN
BEGIN
WRITELN ('PAGINA ')
READ (HEXPAGINA)
HEXAENTERO (HEXPAGINA , PAGINA , OK)
WHILE NOT OK AND NOT EOF (INPUT) DO
 BEGIN
 WRITELN ('ERROR DISCULPABLE')
 OK = TRUE
 WRITELN
 WRITELN ('PAGINA ')
 READ (HEXPAGINA)
 HEXAENTERO (HEXPAGINA , PAGINA , OK)
 END
END

```

```

PROCEDURE TECL DESPLAZAMIENTO (VAR HEXDESPLAZAMIENTO TIPO HEXA
 VAR DESPLAZAMIENTO BYTE SIMPLE)

```

```

VAR
 OK BOOLEAN
BEGIN
WRITELN ('DESPLAZAMIENTO')

```

```

READ (HEXDESPLAZAMIENTO)
HEXAENTERO (HEXDESPLAZAMIENTO , DESPLAZAMIENTO , OK)
WHILE NOT OK AND NOT EOF (INPUT) DO
 BEGIN
 WRITELN ('ERROR DISCULPABLE')
 OK = TRUE
 WRITELN
 WRITELN ('DESPLAZAMIENTO')
 READ (HEXDESPLAZAMIENTO)
 HEXAENTERO (HEXDESPLAZAMIENTO , DESPLAZAMIENTO , OK)
 END

```

END

```

PROCEDURE LEE PAGINA (VAR HEXPAGINA TIPO HEXA
 VAR PAGINA BYTE SIMPLE)

```

```

BEGIN
 TECL PAGINA (HEXPAGINA , PAGINA)
 WHILE (PAGINA PAGINA TOPE) AND NOT EOF (INPUT) DO
 BEGIN
 WRITELN ('ERROR , PAGINA NO EXISTENTE . TIENE OTRA OPORTUNIDAD')
 TECL PAGINA (HEXPAGINA , PAGINA)
 END

```

END

```

PROCEDURE TECL DIRECCION (VAR HEXPAGINA
 , TIPO HEXA
 VAR PAGINA
 DESPLAZAMIENTO BYTE SIMPLE
 VAR BIEN BOOLEAN)

```

```

BEGIN
 LEE PAGINA (HEXPAGINA , PAGINA)
 IF NOT EOF (INPUT)
 THEN TECL DESPLAZAMIENTO (HEXDESPLAZAMIENTO , DESPLAZAMIENTO)
 BIEN = NOT EOF (INPUT)
 IF NOT BIEN THEN WRITELN ('ERROR FATAL ')
END

```

```

PROCEDURE ACTUALIZAR PC (VAR BIEN BOOLEAN)

```

```

BEGIN
 BIEN = TRUE
 IF CPU (. PC LO .) = 255
 THEN IF CPU (. PC HI .) = PAGINA TOPE
 THEN BEGIN
 BIEN = FALSE
 WRITELN ('CONTADOR PROGRAMA MAL')
 END
 ELSE BEGIN
 CPU (. PC HI .) = SUMA (CPU (. PC HI .) , 1)
 CPU (. PC LO .) = SUMA (CPU (. PC LO .) , 1)
 END
 ELSE CPU (. PC LO .) = SUMA (CPU (. PC LO .) , 1)
 END

```

```

PROCEDURE CPU INICIALIZACION
 VAR

```

# REGISTRO TIPO REGISTROS

```
BEGIN
FOR REGISTRO = ACC TO Z DO CPU (. REGISTRO .) = BYTE NULO
END
```

PROCEDURE CPU BACKUP RECUPERACION ( VAR BIEN BOOLEAN )

```
VAR
CPU BACKUP TEXT
REGISTRO TIPO REGISTROS
HEXBYTE TIPO HEXA
BYTE BYTE SIMPLE
```

```
BEGIN
RESET (CPU BACKUP)
FOR REGISTRO = ACC TO SP LO DO
BEGIN
READLN (CPU BACKUP , HEXBYTE)
HEXAENTERO (HEXBYTE , BYTE , BIEN)
CPU (. REGISTRO .) = BYTE
END
CLOSE (CPU BACKUP)
END
```

PROCEDURE CPU BACKUP ACTUALIZACION ( VAR BIEN BOOLEAN )

```
VAR
CPU BACKUP TEXT
REGISTRO TIPO REGISTROS
HEXBYTE TIPO HEXA
BYTE BYTE SIMPLE
```

```
BEGIN
REWRITE (CPU BACKUP)
FOR REGISTRO = ACC TO Z DO
BEGIN
BYTE = CPU (. REGISTRO .)
ENTEROHEXA (BYTE , HEXBYTE , BIEN)
WRITELN (CPU BACKUP , HEXBYTE)
END
CLOSE (CPU BACKUP)
END
```

PROCEDURE MEM INICIALIZACION

```
VAR
PAGINA , DESPLAZAMIENTO BYTE SIMPLE
```

```
BEGIN
FOR PAGINA = 0 TO PAGINA TOPE DO
FOR DESPLAZAMIENTO = 0 TO 255 DO
MEMORIA (. PAGINA , DESPLAZAMIENTO .) = BYTE NULO
END
```

PROCEDURE DEMORA

```
VAR
AUXI CHAR
BEGIN
```

```

WRITELN
WRITELN (' APRETE ENTER PARA SEGUIR ')
READLN (AUXI)
END

```

```

PROCEDURE MEM BACKUP RECUPERACION (VAR BIEN BOOLEAN)
VAR

```

```

 MEM BACKUP TEXT
 HEXPAGINA ,
 HEXDESPLAZAMIENTO ,
 HEXCONTENIDO TIPO HEXA
 PAGINA ,
 DESPLAZAMIENTO ,
 CONTENIDO BYTE SIMPLE
 LINEA STRING (80)
BEGIN
 RESET (MEM BACKUP) BIEN = TRUE
 WHILE NOT EOF (MEM BACKUP) AND BIEN DO
 BEGIN
 READLN (MEM BACKUP , LINEA)
 READSTR (LINEA , HEXPAGINA 2 , HEXDESPLAZAMIENTO 2 ,
 HEXCONTENIDO 2)
 HEXAENTERO (HEXPAGINA , PAGINA , BIEN)
 HEXAENTERO (HEXDESPLAZAMIENTO , DESPLAZAMIENTO , BIEN)
 HEXAENTERO (HEXCONTENIDO , CONTENIDO , BIEN)
 MODIFICAR (PAGINA , DESPLAZAMIENTO , CONTENIDO , BIEN)
 END
END

```

```

END
PROCEDURE MEM BACKUP ACTUALIZACION (VAR BIEN BOOLEAN)
VAR

```

```

 MEM BACKUP TEXT
 HEXPAGINA ,
 HEXDESPLAZAMIENTO ,
 HEXCONTENIDO TIPO HEXA
 PAGINA ,
 DESPLAZAMIENTO ,
 CONTENIDO BYTE SIMPLE
BEGIN
 REWRITE (MEM BACKUP)
 FOR PAGINA = 0 TO PAGINA TOPE DO
 FOR DESPLAZAMIENTO = 0 TO 255 DO
 IF MEMORIA (. PAGINA , DESPLAZAMIENTO .) BYTE NULO
 THEN BEGIN
 CONTENIDO = MEMORIA (. PAGINA , DESPLAZAMIENTO .)
 ENTEROHEXA (PAGINA , HEXPAGINA , BIEN)
 ENTEROHEXA (DESPLAZAMIENTO , HEXDESPLAZAMIENTO , BIEN)
 ENTEROHEXA (CONTENIDO , HEXCONTENIDO , BIEN)
 WRITELN (MEM BACKUP , HEXPAGINA , HEXDESPLAZAMIENTO ,
 HEXCONTENIDO)
 END
 END
 CLOSE (MEM BACKUP)
END

```

```

PROCEDURE VER FLAGS (VAR SALIDA TEXT)
CONST

```

```

C1 = - 6 (* FORMATEO DE SALIDA *)
BEGIN
WRITE (SALIDA , 'SIGNE = ' , TEST FLAG (SIGNE) C1)
WRITE (SALIDA , ' ZERO = ' , TEST FLAG (ZERO) C1)
WRITE (SALIDA , ' AUX CARRY = ' ,
 TEST FLAG (AUX CARRY) C1)
WRITE (SALIDA , ' PARITY = ' , TEST FLAG (PARITY) C1)
WRITE (SALIDA , ' CARRY = ' , TEST FLAG (CARRY) C1)
END

```

```

PROCEDURE VER CPU (VAR SALIDA TEXT VAR BIEN BOOLEAN)
CONST

```

```

C2 = -4 (* PARA CENTRADO DE IMPRESION *)
VAR

```

```

HEXCONTENIDO TIPO HEXA
BEGIN
CMS ('CLRSCRN' , COM CMS)
ENTEROHEXA (CPU (. ACC .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'ACC =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. B .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'B =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. C .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'C =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. D .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'D =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. E .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'E =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. H .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'H =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. L .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'L =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. FLAGS .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'FLAGS =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. PC HI .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'PC =' , HEXCONTENIDO)
ENTEROHEXA (CPU (. PC LO .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. IR .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'IR =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. W .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'W =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. Z .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'Z =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. TEMP .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'TEMP =' , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. SP HI .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'SP =' , HEXCONTENIDO)
ENTEROHEXA (CPU (. SP LO .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , HEXCONTENIDO)
ENTEROHEXA (CPU (. SP LO .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , 'MAR =' , HEXCONTENIDO)
ENTEROHEXA (CPU (. MAR LO .) , HEXCONTENIDO , BIEN)
WRITE (SALIDA , HEXCONTENIDO C2)
ENTEROHEXA (CPU (. MDR .) , HEXCONTENIDO , BIEN)

```

```

WRITELN (SALIDA , 'MDR =' , HEXCONTENIDO)
VER FLAGS (SALIDA)
END
PROCEDURE VER PAGINA (HEXPAGINA TIPO HEXA PAGINA BYTE SIMPLE)
VAR
 HEXDESPLAZAMIENTO ,
 HEXCONTENIDO TIPO HEXA
 NIBLE HI 0 .. 15
 DESPLAZAMIENTO ,
 CONTENIDO BYTE SIMPLE
 BIEN BOOLEAN
 PROCEDURE UNA FILA (NIBLE HI BYTE SIMPLE)
 VAR NIBLE LO 0 .. 15
 BEGIN
 ENTEROHEXA (NIBLE HI * 16 , HEXDESPLAZAMIENTO , BIEN)
 WRITE (HEXPAGINA , HEXDESPLAZAMIENTO - 6)
 FOR NIBLE LO = 0 TO 15 DO
 BEGIN
 DESPLAZAMIENTO = NIBLE HI * 16 + NIBLE LO
 CONTENIDO = MEMORIA (. PAGINA , DESPLAZAMIENTO .)
 ENTEROHEXA (CONTENIDO , HEXCONTENIDO , BIEN)
 WRITE (HEXCONTENIDO 3)
 END
 END
 BEGIN
 HEXAENTERO (HEXPAGINA , PAGINA , BIEN)
 FOR NIBLE HI = 0 TO 15 DO
 BEGIN
 UNA FILA (NIBLE HI)
 WRITELN
 END
 END
 END
PROCEDURE VER MEMORIA (VAR BIEN BOOLEAN)
CONST
 FIN VER = '.'
VAR
 HEXPAGINA TIPO HEXA
 PAGINA BYTE SIMPLE
 CAR CHAR
 BEGIN
 REPEAT
 CAR = ' '
 CMS ('CLRSCRN' , COM CMS)
 LEE PAGINA (HEXPAGINA , PAGINA)
 WHILE (PAGINA PAGINA TOPE) AND
 RESPUESTA ('PAGINA DE MEMORIA NO VALIDA . TERMINA ') DO
 LEE PAGINA (HEXPAGINA , PAGINA)
 CMS ('CLRSCRN' , COM CMS)
 VER PAGINA (HEXPAGINA , PAGINA)
 WRITELN ('ENTER PARA CONTINUAR , PUNTO PARA PARAR ')
 READLN
 READLN (CAR)
 UNTIL (CAR = FIN VER)
 END
END

```

```

PROCEDURE CPU EDICION (VAR BIEN BOOLEAN)
 FUNCTION ESTE REGISTRO (CONST REG STRING) TIPO REGISTROS
 BEGIN
 IF (REG = 'ACC') OR (REG = 'ACC')
 THEN ESTE REGISTRO = ACC
 ELSE IF (REG = 'TEMP') OR (REG = 'TEMP')
 THEN ESTE REGISTRO = TEMP
 ELSE IF (REG = 'IR') OR (REG = 'IR')
 THEN ESTE REGISTRO = IR
 ELSE IF (REG = 'FLAGS') OR (REG = 'FLAGS')
 THEN ESTE REGISTRO = FLAGS
 ELSE IF (REG = 'B') OR (REG = 'B')
 THEN ESTE REGISTRO = B
 ELSE IF (REG = 'C') OR (REG = 'C')
 THEN ESTE REGISTRO = C
 ELSE IF (REG = 'D') OR (REG = 'D')
 THEN ESTE REGISTRO = D
 ELSE IF (REG = 'E') OR (REG = 'E')
 THEN ESTE REGISTRO = E
 ELSE IF (REG = 'H') OR (REG = 'H')
 THEN ESTE REGISTRO = H
 ELSE IF (REG = 'L') OR (REG = 'L')
 THEN ESTE REGISTRO = L
 ELSE IF (REG = 'PC HI') OR (REG = 'PC HI')
 THEN ESTE REGISTRO = PC HI
 ELSE IF (REG = 'PC LO') OR (REG = 'PC LO')
 THEN ESTE REGISTRO = PC LO
 ELSE IF (REG = 'MAR HI') OR (REG = 'MAR HI')
 THEN ESTE REGISTRO = MAR HI
 ELSE IF (REG = 'MAR LO') OR (REG = 'MAR LO')
 THEN ESTE REGISTRO = MAR LO
 ELSE IF (REG = 'MDR') OR (REG = 'MDR')
 THEN ESTE REGISTRO = MDR
 ELSE IF (REG = 'Z') OR (REG = 'Z')
 THEN ESTE REGISTRO = Z
 ELSE IF (REG = 'W') OR (REG = 'W')
 THEN ESTE REGISTRO = W
 ELSE ESTE REGISTRO = REG ERROR
 END
 FUNCTION NUEVO CONTENIDO BYTE SIMPLE
 VAR
 HEXACONTENIDO TIPO HEXA
 CONTENIDO BYTE SIMPLE
 OK BOOLEAN
 BEGIN
 READ (HEXACONTENIDO)
 HEXAENTERO (HEXACONTENIDO , CONTENIDO , OK)
 IF OK
 THEN NUEVO CONTENIDO = CONTENIDO
 ELSE NUEVO CONTENIDO = 0
 END
 VAR
 REG STRING (6)
 CONST
 FIN EDICION = '.'

```

```

BEGIN
 VER CPU (OUTPUT , BIEN)
 WRITELN
 READ (REG)
 WHILE (REG FIN EDICION) DO
 BEGIN
 CPU (. ESTE REGISTRO (REG) .) = NUEVO CONTENIDO
 CMS ('CLRSCRN' , COM CMS)
 VER CPU (OUTPUT , BIEN)
 WRITELN
 READ (REG)
 END
END

PROCEDURE MEM EDITAR (VAR BIEN BOOLEAN)
CONST
 FIN EDICION = '.'
VAR
 HEXPAGINA ,
 HEXDESPLAZAMIENTO ,
 HEXCONTENIDO TIPO HEXA
 PAGINA ,
 DESPLAZAMIENTO ,
 CONTENIDO BYTE SIMPLE
 STATUS (LEO DIRE ,
 LEO CONT ,
 CAMBIA CONT ,
 CONTINUACION ,
 MAL , FIN)

PROCEDURE INCREMENTO
 (VAR PAGINA , DESPLAZAMIENTO BYTE SIMPLE)
BEGIN
 DESPLAZAMIENTO = SUMA (DESPLAZAMIENTO , 1)
 IF DESPLAZAMIENTO = 0 THEN PAGINA = SUMA (PAGINA , 1)
END
BEGIN
 STATUS = LEO DIRE
 BIEN = TRUE
 REPEAT
 CASE STATUS OF
 LEO DIRE BEGIN
 CMS ('CLRSCRN' , COM CMS)
 WRITELN ('PAGINA , DESPLAZAMIENTO ')
 TECL DIRECCION (HEXPAGINA , HEXDESPLAZAMIENTO ,
 PAGINA , DESPLAZAMIENTO ,
 BIEN
)
 IF BIEN
 THEN STATUS = LEO CONT
 ELSE BEGIN
 WRITELN ('ERROR FATAL')
 STATUS = MAL
 END
 END
 LEO CONT BEGIN

```

```

CMS ('CLRSRN' , COM CMS)
VER PAGINA (HEXPAGINA , PAGINA)
READ (HEXCONTENIDO)
IF (HEXCONTENIDO FIN EDICION) THEN BEGIN
 HEXAENTERO (HEXCONTENIDO , CONTENIDO , BIEN)
 IF BIEN
 THEN STATUS = CAMBIA CONT
 ELSE STATUS = LEO CONT
END
ELSE STATUS = FIN
END
CAMBIA CONT BEGIN
 MODIFICAR (PAGINA , DESPLAZAMIENTO , CONTENIDO , BIEN)
 IF BIEN
 THEN STATUS = CONTINUACION
 ELSE STATUS = MAL
END
CONTINUACION BEGIN
 INCREMENTO (PAGINA , DESPLAZAMIENTO)
 ENTEROHEXA (PAGINA , HEXPAGINA , BIEN)
 ENTEROHEXA (DESPLAZAMIENTO , HEXDESPLAZAMIENTO , BIEN)
 STATUS = LEO CONT
END
MAL BEGIN
 WRITE ('ERROR EN EDICION DE MEMORIA')
 STATUS = LEO CONT
END
FIN WRITE ('FIN EDICION DE MEMORIA')
END
UNTIL STATUS = FIN
END

```

```

PROCEDURE BUSCAR ARGUMENTO
(UN REGISTRO TIPO REGISTROS VAR BIEN BOOLEAN)
BEGIN
 RECUPERAR (CPU (. PC HI .) , CPU (. PC LO .) , CPU (. MDR .) ,
 BIEN)
 IF BIEN
 THEN
 BEGIN
 CPU (. UN REGISTRO .) = CPU (. MDR .)
 ACTUALIZAR PC (BIEN)
 END
END

```

```

END
PROCEDURE ADC (UN REGISTRO TIPO REGISTROS)
VAR
 AUX1 , AUX2 BOOLEAN
BEGIN
 CPU (. TEMP .) = CPU (. UN REGISTRO .)
 CPU (. ACC .) = SUMA (CPU (. ACC .) , CPU (. TEMP .))
 AUX1 = ACARREO AUX2 = MEDIO ACARREO
 CPU (. TEMP .) = ANDEADO (CPU (. FLAGS .) , CARRY)
 CPU (. ACC .) = SUMA (CPU (. ACC .) , CPU (. TEMP .))
 ACARREO = ACARREO OR AUX1
 MEDIO ACARREO = MEDIO ACARREO OR AUX2

```

FLAGS ACTUALIZAR

END

PROCEDURE ADC M ( VAR BIEN BOOLEAN )

BEGIN

CPU ( . MAR LO . ) = CPU ( . L . )

CPU ( . MAR HI . ) = CPU ( . H . )

RECUPERAR ( CPU ( . MAR HI . ), CPU ( . MAR LO . ), CPU ( . MDR . ), BIEN )

IF BIEN

THEN ADD ( MDR )

END

PROCEDURE ADD ( UN REGISTRO TIPO REGISTROS )

BEGIN

CPU ( . TEMP . ) = CPU ( . UN REGISTRO . )

CPU ( . ACC . ) = SUMA ( CPU ( . ACC . ), CPU ( . TEMP . ) )

FLAGS ACTUALIZAR

END

PROCEDURE ADD M ( VAR BIEN BOOLEAN )

BEGIN

CPU ( . MAR LO . ) = CPU ( . L . )

CPU ( . MAR HI . ) = CPU ( . H . )

RECUPERAR ( CPU ( . MAR HI . ), CPU ( . MAR LO . ), CPU ( . MDR . ), BIEN )

IF BIEN

THEN ADD ( MDR )

END

PROCEDURE SUB ( UN REGISTRO TIPO REGISTROS )

VAR

BORROW BOOLEAN

ACUM , REGI BYTE SIMPLE

BEGIN

ACUM = CPU ( . ACC . )

REGI = CPU ( . UN REGISTRO . )

CPU ( . ACC . ) = SUMA ( CPU ( . ACC . ) ,

COMPLEMENTO DOS ( CPU ( . UN REGISTRO . ) ) )

BORROW = ( ( REGI = 127 ) AND ( ACUM 127 ) ) OR  
( ACUM REGI )

ACARREO = BORROW

FLAGS ACTUALIZAR

END

PROCEDURE SUB M ( VAR BIEN BOOLEAN )

BEGIN

CPU ( . MAR LO . ) = CPU ( . L . )

CPU ( . MAR HI . ) = CPU ( . H . )

RECUPERAR ( CPU ( . MAR HI . ), CPU ( . MAR LO . ), CPU ( . MDR . ), BIEN )

IF BIEN

THEN SUB ( MDR )

END

PROCEDURE SBB ( UN REGISTRO TIPO REGISTROS )

BEGIN

CPU ( . TEMP . ) = SUMA ( CPU ( . UN REGISTRO . ) ,  
ORD ( TEST FLAG ( CARRY ) ) )

SUB ( TEMP )

END

PROCEDURE SBB M ( VAR BIEN BOOLEAN )

BEGIN

CPU ( . MAR LO . ) = CPU ( . L . )

```

CPU (. MAR HI .) = CPU (. H .)
RECUPERAR (CPU (. MAR HI .),CPU (. MAR LO .),CPU (. MDR .),BIEN)
IF BIEN
 THEN SBB (MDR)

```

```

END
PROCEDURE ANA (UN REGISTRO TIPO REGISTROS)

```

```

BEGIN
 CPU (.TEMP.) = CPU (. UN REGISTRO .)
 CPU (. ACC.) = ANDEADO (CPU (. ACC .) , CPU (. TEMP .))
 S Z P ACTUALIZAR
 RESET FLAG (CARRY)

```

```

END
PROCEDURE ANA M (VAR BIEN BOOLEAN)

```

```

BEGIN
 CPU (. MAR LO .) = CPU (. L .)
 CPU (. MAR HI .) = CPU (. H .)
 RECUPERAR (CPU (. MAR HI .),CPU (. MAR LO .),CPU (. MDR .),BIEN)
 IF BIEN
 THEN ANA (MDR)

```

```

END
PROCEDURE XRA (UN REGISTRO TIPO REGISTROS)

```

```

BEGIN
 CPU (.TEMP.) = CPU (. UN REGISTRO .)
 CPU (. ACC.) = XOREADO (CPU (. ACC .) , CPU (. TEMP .))
 S Z P ACTUALIZAR
 RESET FLAG (CARRY)
 RESET FLAG (AUX CARRY)

```

```

END
PROCEDURE XRA M (VAR BIEN BOOLEAN)

```

```

BEGIN
 CPU (. MAR LO .) = CPU (. L .)
 CPU (. MAR HI .) = CPU (. H .)
 RECUPERAR (CPU (. MAR HI .),CPU (. MAR LO .),CPU (. MDR .),BIEN)
 IF BIEN
 THEN XRA (MDR)

```

```

END
PROCEDURE ORA (UN REGISTRO TIPO REGISTROS)

```

```

BEGIN
 CPU (.TEMP.) = CPU (. UN REGISTRO .)
 CPU (. ACC.) = OREADO (CPU (. ACC .) , CPU (. TEMP .))
 S Z P ACTUALIZAR
 RESET FLAG (CARRY)
 RESET FLAG (AUX CARRY)

```

```

END
PROCEDURE ORA M (VAR BIEN BOOLEAN)

```

```

BEGIN
 CPU (. MAR LO .) = CPU (. L .)
 CPU (. MAR HI .) = CPU (. H .)
 RECUPERAR (CPU (. MAR HI .),CPU (. MAR LO .),CPU (. MDR .),BIEN)
 IF BIEN
 THEN ORA (MDR)

```

```

END
PROCEDURE CMP (UN REGISTRO TIPO REGISTROS)

```

```

BEGIN
 CPU (. W .) = CPU (. ACC .)

```

```

CPU (. TEMP .) = CPU (. UN REGISTRO .)
SUB (TEMP)
CPU (. ACC .) = CPU (. W .)
END
PROCEDURE CMP M (VAR BIEN BOOLEAN)
BEGIN
 CPU (. MAR LO .) = CPU (. L .)
 CPU (. MAR HI .) = CPU (. H .)
 RECUPERAR (CPU (. MAR HI .),CPU (. MAR LO .),CPU (. MDR .),BIEN)
 IF BIEN THEN CMP (MDR)
END
PROCEDURE INX (UN REGISTRO TIPO REGISTROS)
BEGIN
 CPU (. SUCC (UN REGISTRO) .) =
 SUMA (CPU (. SUCC (UN REGISTRO) .) , 1)
 IF CPU (. SUCC (UN REGISTRO) .) = 0 THEN
 CPU (. UN REGISTRO .) = SUMA (CPU (. UN REGISTRO .) , 1)
END
PROCEDURE DCX (UN REGISTRO TIPO REGISTROS)
BEGIN
 CPU (. SUCC (UN REGISTRO) .) =
 SUMA (CPU (. SUCC (UN REGISTRO) .) , 255)
 IF CPU (. SUCC (UN REGISTRO) .) = 255 THEN
 CPU (. UN REGISTRO .) = SUMA (CPU (. UN REGISTRO .) , 255)
END
PROCEDURE POP
 (UN REGISTRO TIPO REGISTROS VAR BIEN BOOLEAN)
BEGIN
 CPU (. MAR LO .) = CPU (. SP LO .)
 CPU (. MAR HI .) = CPU (. SP HI .)
 RECUPERAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN BEGIN
 CPU (. SUCC (UN REGISTRO) .) = CPU (. MDR .)
 INX (SP HI)
 CPU (. MAR LO .) = CPU (. SP LO .)
 CPU (. MAR HI .) = CPU (. SP HI .)
 RECUPERAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN BEGIN
 CPU (. UN REGISTRO .) = CPU (. MDR .)
 INX (SP HI)
 END
 END
END
PROCEDURE RET (VAR BIEN BOOLEAN)
BEGIN
 POP (PC HI , BIEN)
END
PROCEDURE RN FLAG (UN FLAG BYTE SIMPLE VAR BIEN BOOLEAN)
BEGIN
 CPU (. TEMP .) = ANDEADO (CPU (. FLAGS .) , UN FLAG)
 IF CPU (. TEMP .) = 0 THEN RET (BIEN)
END
PROCEDURE R FLAG (UN FLAG BYTE SIMPLE VAR BIEN BOOLEAN)

```

```

BEGIN
 CPU (. TEMP .) = ANDEADO (CPU (. FLAGS .) , UN FLAG)
 IF CPU (. TEMP .) = UN FLAG THEN RET (BIEN)
END
PROCEDURE INR (UN REGISTRO TIPO REGISTROS)
BEGIN
 IF (UN REGISTRO = ACC)
 THEN
 BEGIN
 CPU (. ACC .) = SUMA (CPU (. UN REGISTRO .) , 1)
 FLAGS ACTUALIZAR
 END
 ELSE
 BEGIN
 CPU (. W .) = CPU (. ACC .)
 CPU (. ACC .) = SUMA (CPU (. UN REGISTRO .) , 1)
 FLAGS ACTUALIZAR
 CPU (. UN REGISTRO .) = CPU (. ACC .)
 CPU (. ACC .) = CPU (. W .)
 END
 END
END
PROCEDURE INR M (VAR BIEN BOOLEAN)
BEGIN
 CPU (. MAR LO .) = CPU (. L .)
 CPU (. MAR HI .) = CPU (. H .)
 RECUPERAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN BEGIN
 INR (MDR)
 MODIFICAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 END
END
PROCEDURE LXI
(UN REGISTRO TIPO REGISTROS VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 BUSCAR ARGUMENTO (W , BIEN)
 IF BIEN THEN BEGIN
 CPU (. UN REGISTRO .) = CPU (. W .)
 CPU (. SUCC (UN REGISTRO) .) = CPU (. Z .)
 END
 END
END
PROCEDURE MVI (UN REGISTRO TIPO REGISTROS VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN CPU (. UN REGISTRO .) = CPU (. Z .)
END
PROCEDURE MVI M (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. MAR LC .) = CPU (. L .)

```

```

CPU (. MAR HI .) = CPU (. H .)
CPU (. MDR .) = CPU (. Z .)
MODIFICAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
END
END
PROCEDURE DCR
(UN REGISTRO TIPO REGISTROS)
BEGIN
 IF (UN REGISTRO = ACC)
 THEN
 BEGIN
 CPU (. ACC .) = SUMA (CPU (. UN REGISTRO .) , 255)
 FLAGS ACTUALIZAR
 END
 ELSE
 BEGIN
 CPU (. W .) = CPU (. ACC .)
 CPU (. ACC .) = SUMA (CPU (. UN REGISTRO .) , 255)
 FLAGS ACTUALIZAR
 CPU (. UN REGISTRO .) = CPU (. ACC .)
 CPU (. ACC .) = CPU (. W .)
 END
 END
END
PROCEDURE DCR M
(VAR BIEN BOOLEAN)
BEGIN
 CPU (. MAR LO .) = CPU (. L .)
 CPU (. MAR HI .) = CPU (. H .)
 RECUPERAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN BEGIN
 DCR (MDR)
 MODIFICAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 END
END
END
PROCEDURE STAX (UN REGISTRO TIPO REGISTROS VAR BIEN BOOLEAN)
BEGIN
 CPU (. MAR LO .) = CPU (. SUCC (UN REGISTRO) .)
 CPU (. MAR HI .) = CPU (. UN REGISTRO .)
 CPU (. MDR .) = CPU (. ACC .)
 MODIFICAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
END
END
PROCEDURE LDAX (UN REGISTRO TIPO REGISTROS VAR BIEN BOOLEAN)
BEGIN
 CPU (. MAR LO .) = CPU (. SUCC (UN REGISTRO) .)
 CPU (. MAR HI .) = CPU (. UN REGISTRO .)
 RECUPERAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN CPU (. ACC .) = CPU (. MDR .)
END
END
PROCEDURE DAD (UN REGISTRO TIPO REGISTROS)
VAR UN REGI BYTE SIMPLE

```

```

BEGIN
 CPU (. L .) = SUMA (CPU (. L .) , CPU (. SUCC (UN REGISTRO) .))
 UN REGI = CPU (. UN REGISTRO .)
 IF ACARREO THEN BEGIN
 SET FLAG (CARRY)
 CPU (. H .) = SUMA (CPU (. H .) , 1)
 END
 CPU (. H .) = SUMA (CPU (. H .) , UN REGI)
 IF ACARREO THEN SET FLAG (CARRY)
END
PROCEDURE PUSH
 (UN REGISTRO TIPO REGISTROS VAR BIEN BOOLEAN)
BEGIN
 DCX (SP HI)
 CPU (. MAR LO .) = CPU (. SP LO .)
 CPU (. MAR HI .) = CPU (. SP HI .)
 CPU (. MDR .) = CPU (. UN REGISTRO .)
 MODIFICAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN BEGIN
 DCX (SP HI)
 CPU (. MAR LO .) = CPU (. SP LO .)
 CPU (. MAR HI .) = CPU (. SP HI .)
 CPU (. MDR .) = CPU (. SUCC (UN REGISTRO) .)
 MODIFICAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 END
END
PROCEDURE CALL (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 BUSCAR ARGUMENTO (W , BIEN)
 IF BIEN THEN BEGIN
 PUSH (PC HI , BIEN)
 IF BIEN THEN BEGIN
 CPU (. PC LO .) = CPU (. Z .)
 CPU (. PC HI .) = CPU (. W .)
 END
 END
 END
END
END
PROCEDURE CN FLAG (UN FLAG BYTE SIMPLE VAR BIEN BOOLEAN)
BEGIN
 CPU (. TEMP .) = ANDEADO (CPU (. FLAGS .) , UN FLAG)
 IF CPU (. TEMP .) = 0 THEN CALL (BIEN)
 ELSE BEGIN
 INX (PC HI)
 INX (PC HI)
 END
END
PROCEDURE C FLAG (UN FLAG BYTE SIMPLE VAR BIEN BOOLEAN)
BEGIN
 CPU (. TEMP .) = ANDEADO (CPU (. FLAGS .) , UN FLAG)
 IF CPU (. TEMP .) = UN FLAG THEN CALL (BIEN)

```

```

ELSE BEGIN
 INX (PC HI)
 INX (PC HI)
END
END
PROCEDURE JUMP (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 BUSCAR ARGUMENTO (W , BIEN)
 IF BIEN THEN BEGIN
 CPU (. PC LO .) = CPU (. Z .)
 CPU (. PC HI .) = CPU (. W .)
 END
 END
END
END
PROCEDURE JN FLAG (UN FLAG BYTE SIMPLE VAR BIEN BOOLEAN)
BEGIN
 CPU (. TEMP .) = ANDEADO (CPU (. FLAGS .) , UN FLAG)
 IF CPU (. TEMP .) = 0
 THEN JUMP (BIEN)
 ELSE BEGIN
 INX (PC HI)
 INX (PC HI)
 END
END
END
PROCEDURE J FLAG (UN FLAG BYTE SIMPLE VAR BIEN BOOLEAN)
BEGIN
 CPU (. TEMP .) = ANDEADO (CPU (. FLAGS .) , UN FLAG)
 IF CPU (. TEMP .) = UN FLAG
 THEN JUMP (BIEN)
 ELSE BEGIN
 INX (PC HI)
 INX (PC HI)
 END
END
END
PROCEDURE PCHL
BEGIN
 CPU (. PC LO .) = CPU (. L .)
 CPU (. PC HI .) = CPU (. H .)
END
PROCEDURE SHLD(VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO(MAR LO , BIEN)
 IF BIEN THEN BEGIN
 BUSCAR ARGUMENTO(MAR HI , BIEN)
 IF BIEN THEN BEGIN
 CPU(. MDR .) = CPU(. L .)
 MODIFICAR (CPU(. MAR HI .) , CPU(. MAR LO .) , CPU(. MDR .) , BIEN)
 IF BIEN THEN BEGIN
 INX (MAR HI)
 IF BIEN THEN BEGIN
 CPU(. MDR .) = CPU(. H .)
 MODIFICAR (CPU(. MAR HI .) , CPU(. MAR LO .) , CPU(. MDR .) , BIEN)
 END
 END
 END
 END
END

```

```

 END
 END
END
PROCEDURE LHLD(VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO(MAR LO , BIEN)
 IF BIEN THEN BEGIN
 BUSCAR ARGUMENTO(MAR HI , BIEN)
 IF BIEN THEN BEGIN
 CPU(.MDR.) = CPU(.L.)
 RECUPERAR (CPU(.MAR HI.),CPU(.MAR LO.),CPU(.MDR.),BIEN)
 IF BIEN THEN BEGIN
 CPU(.L.) = CPU(.MDR.)
 INX (MAR HI)
 IF BIEN THEN BEGIN
 CPU(.MDR.) = CPU(.H.)
 RECUPERAR (CPU(.MAR HI.),CPU(.MAR LO.),CPU(.MDR.),BIEN)
 IF BIEN THEN CPU(.H.) =CPU(.MDR.)
 END
 END
 END
 END
END
END
PROCEDURE CMA
BEGIN
 CPU(.ACC.) = COMPLEMENTO UNC(CPU(.ACC.))
END
PROCEDURE LDA (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (MAR LO , BIEN)
 IF BIEN THEN BEGIN
 BUSCAR ARGUMENTO (MAR HI,BIEN)
 RECUPERAR(CPU(.MAR HI.),CPU(.MAR LO.),CPU(.MDR.),BIEN)
 IF BIEN THEN CPU(.ACC.) = CPU(.MDR.)
 END
END
PROCEDURE STA(VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO(MAR LO,BIEN)
 IF BIEN THEN BEGIN
 BUSCAR ARGUMENTO(MAR HI,BIEN)
 IF BIEN THEN BEGIN
 CPU (. MDR .) = CPU (. ACC .)
 MODIFICAR(CPU(.MAR HI.),CPU(.MAR LO.),CPU(.MDR.),BIEN)
 END
 END
END
END
PROCEDURE STC
BEGIN
 SET FLAG (CARRY)
END
PROCEDURE CMC
BEGIN
 CPU(.FLAGS.) = XCREADO(CPU(.FLAGS.),CARRY)

```

```

END
PROCEDURE ADI (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. TEMP .) = CPU (. Z .)
 CPU (. ACC .) = SUMA (CPU (. ACC .) , CPU (. TEMP .))
 FLAGS ACTUALIZAR
 END
END
PROCEDURE ACI (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. TEMP .) = ANDEADO (CPU (. FLAGS .) , CARRY)
 CPU (. ACC .) = SUMA (CPU (. ACC .) , CPU (. TEMP .))
 CPU (. TEMP .) = CPU (. Z .)
 CPU (. ACC .) = SUMA (CPU (. ACC .) , CPU (. TEMP .))
 FLAGS ACTUALIZAR
 END
END
PROCEDURE SUI (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. TEMP .) = CPU (. Z .)
 SUB (TEMP)
 END
END
PROCEDURE SBI (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. TEMP .) = ANDEADO (CPU (. FLAGS .) , CARRY)
 CPU (. TEMP .) = SUMA (CPU (. TEMP .) , CPU (. Z .))
 SUB (TEMP)
 END
END
PROCEDURE ANI (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. TEMP .) = CPU (. Z .)
 CPU (. ACC .) = ANDEADO (CPU (. ACC .) , CPU (. TEMP .))
 S Z P ACTUALIZAR
 RESET FLAG (CARRY)
 END
END
PROCEDURE XRI (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. TEMP .) = CPU (. Z .)
 CPU (. ACC .) = XOREADO (CPU (. ACC .) , CPU (. TEMP .))
 S Z P ACTUALIZAR
 END
END

```

```

 RESET FLAG (CARRY)
 RESET FLAG (AUX CARRY)
END
END
PROCEDURE ORI (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. TEMP .) = CPU (. Z .)
 CPU (. ACC .) = OREADO (CPU (. ACC .) , CPU (. TEMP .))
 S Z P ACTUALIZAR
 RESET FLAG (CARRY)
 RESET FLAG (AUX CARRY)
 END
END
PROCEDURE CPI (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. W .) = CPU (. ACC .)
 CPU (. TEMP .) = CPU (. Z .)
 SUB (TEMP)
 CPU (. ACC .) = CPU (. W .)
 END
END
PROCEDURE XCHG
BEGIN
 CPU (. W .) = CPU (. D .)
 CPU (. Z .) = CPU (. E .)
 CPU (. D .) = CPU (. H .)
 CPU (. E .) = CPU (. L .)
 CPU (. H .) = CPU (. W .)
 CPU (. L .) = CPU (. Z .)
END
PROCEDURE RST (VALOR BYTE SIMPLE VAR BIEN BOOLEAN)
BEGIN
 PUSH (PC HI , BIEN)
 IF BIEN THEN BEGIN
 CPU (. PC HI .) = 0
 CPU (. PC LO .) = 8 * VALOR
 END
END
PROCEDURE SPHL
BEGIN
 CPU (. SP LO .) = CPU (. L .)
 CPU (. SP HI .) = CPU (. H .)
END
PROCEDURE XTHL (VAR BIEN BOOLEAN)
BEGIN
 CPU (. MAR LO .) = CPU (. SP LO .)
 CPU (. MAR HI .) = CPU (. SP HI .)
 RECUPERAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN BEGIN
 CPU (. Z .) = CPU (. MDR .)
 END
END

```

```

CPU (. MDR .) = CPU (. L .)
MODIFICAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
INX (SP HI)
CPU (. MAR LO .) = CPU (. SP LO .)
CPU (. MAR HI .) = CPU (. SP HI .)
RECUPERAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
IF BIEN THEN BEGIN
 CPU (. W .) = CPU (. MDR .)
 CPU (. MDR .) = CPU (. L .)
 MODIFICAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN BEGIN
 CPU (. L .) = CPU (. Z .)
 CPU (. H .) = CPU (. W .)
 END
END

```

END

END

END

PROCEDURE RLC

VAR MSB BYTE SIMPLE

BEGIN

MSB = CPU (. ACC .) DIV 128

CPU (. ACC .) = ( ( CPU (. ACC .) \* 2 ) MOD 256 ) + MSB

IF ( MSB = 1 ) THEN

SET FLAG ( CARRY )

ELSE RESET FLAG ( CARRY )

END

PROCEDURE RRC

VAR LSB BYTE SIMPLE

BEGIN

LSB = CPU (. ACC .) MOD 2

CPU (. ACC .) = ( CPU (. ACC .) DIV 2 ) + LSB \* 128

IF ( LSB = 1 ) THEN

SET FLAG ( CARRY )

ELSE RESET FLAG ( CARRY )

END

PROCEDURE RAL

VAR MSB BYTE SIMPLE

BEGIN

MSB = CPU (. ACC .) DIV 128

CPU (. TEMP .) = ANDEADO ( CPU (. FLAGS .) , CARRY )

CPU (. ACC .) = ( ( CPU (. ACC .) \* 2 ) MOD 256 )

CPU (. ACC .) = SUMA ( CPU (. ACC .) , CPU (. TEMP .) )

IF ( MSB = 1 ) THEN

SET FLAG ( CARRY )

ELSE RESET FLAG ( CARRY )

END

PROCEDURE RAR

VAR LSB BYTE SIMPLE

BEGIN

LSB = CPU (. ACC .) MOD 2

CPU (. TEMP .) = ANDEADO ( CPU (. FLAGS .) , CARRY )

CPU (. TEMP .) = ( CPU (. TEMP .) \* 128 ) MOD 256

```

CPU (. ACC .) = (CPU (. ACC .) DIV 2)
CPU (. ACC .) = SUMA (CPU (. ACC .) , CPU (. TEMP .))
IF (LSB = 1) THEN
 SET FLAG (CARRY)
ELSE RESET FLAG (CARRY)
END
PROCEDURE OUT (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. MAR LO .) = CPU (. Z .)
 CPU (. MAR HI .) = PAGINA TOPE
 CPU (. MDR .) = CPU (. ACC .)
 MODIFICAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 END
END
PROCEDURE MOV M R
(UN REGISTRO TIPO REGISTROS VAR BIEN BOOLEAN)
BEGIN
 CPU (. MAR LO .) = CPU (. L .)
 CPU (. MAR HI .) = CPU (. H .)
 CPU (. MDR .) = CPU (. UN REGISTRO .)
 MODIFICAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
END
PROCEDURE MOV R M
(UN REGISTRO TIPO REGISTROS VAR BIEN BOOLEAN)
BEGIN
 CPU (. MAR LO .) = CPU (. L .)
 CPU (. MAR HI .) = CPU (. H .)
 RECUPERAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN CPU (. UN REGISTRO .) = CPU (. MDR .)
END
PROCEDURE INP (VAR BIEN BOOLEAN)
BEGIN
 BUSCAR ARGUMENTO (Z , BIEN)
 IF BIEN THEN BEGIN
 CPU (. MAR LO .) = CPU (. Z .)
 CPU (. MAR HI .) = PAGINA TOPE
 RECUPERAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN CPU (. ACC .) = CPU (. MDR .)
 END
END
PROCEDURE DAA
PROCEDURE AJUSTE
(VAR BYTE BYTE SIMPLE
 VAR AUX FLAG BOOLEAN
 UN FLAG INTEGER)
BEGIN
 IF (BYTE 9) OR TEST FLAG (UN FLAG)
 THEN
 BEGIN

```

```

 BYTE = BYTE + 6
 AUX FLAG = (BYTE 15)
 BYTE = BYTE MOD 16
END

```

```

END
VAR BYTE HI , BYTE LO BYTE SIMPLE
BEGIN
 BYTE HI = CPU (. ACC .) DIV 16
 BYTE LO = ANDEADO (CPU (. ACC .) , 15)
 AJUSTE (BYTE LO , MEDIO ACARREO , AUX CARRY)
 IF MEDIO ACARREO THEN BYTE HI = BYTE HI + 1
 AJUSTE (BYTE HI , ACARREO , CARRY)
 CPU (. ACC .) = BYTE HI * 16 + BYTE LO
 FLAGS ACTUALIZAR

```

```

END
PROCEDURE EXECUTE (VAR BIEN BOOLEAN)
VAR
 STATUS (BUSQUEDA , EJECUTAR , FIN)
 HEXDIRHI , HEXDIRLO ,
 HEXHI , HEXLO , HEXCONT TIPO HEXA
 DIRHI , DIRLO ,
 HI , LO , CONT BYTE SIMPLE
 CONTADOR INTEGER
 FINAL BOOLEAN
 PROCEDURE FASE BUSQUEDA (VAR BIEN BOOLEAN)
 BEGIN
 CPU (. MAR LO .) = CPU (. PC LO .)
 CPU (. MAR HI .) = CPU (. PC HI .)
 RECUPERAR (CPU (. MAR HI .) , CPU (. MAR LO .) ,
 CPU (. MDR .) , BIEN)
 IF BIEN THEN BEGIN
 CPU (. IR .) = CPU (. MDR .)
 INX (PC HI)
 END
 END
END

```

```

PROCEDURE FASE EJECUCION (VAR BIEN , FINAL BOOLEAN)
BEGIN
 CASE CPU (. IR .) OF
 0 (* 00 NOP *)
 1 (* 01 LXI B *) LXI (B , BIEN)
 2 (* 02 STAX B *) STAX (B , BIEN)
 3 (* 03 INX B *) INX (B)
 4 (* 04 INR B *) INR (B)
 5 (* 05 DCR B *) DCR (B)
 6 (* 06 MVI B *) MVI (B , BIEN)
 7 (* 07 RLC *) RLC
 8 (* 08 *) BIEN = FALSE
 9 (* 09 DAD B *) DAD (B)
 10 (* 0A LDAX B *) LDAX (B , BIEN)
 11 (* 0B DCX B *) DCX (B)
 12 (* 0C INR C *) INR (C)
 13 (* 0D DCR C *) DCR (C)
 14 (* 0E MVI C *) MVI (C , BIEN)
 15 (* 0F RRC *) RRC

```

```

16 (* 10 *) BIEN = FALSE
17 (* 11 LXI D *) LXI (D , BIEN)
18 (* 12 STAX D*) STAX (D , BIEN)
19 (* 13 INX D *) INX (D)
20 (* 14 INR D *) INR (D)
21 (* 15 DCR D *) DCR (D)
22 (* 16 MVI D *) MVI (D , BIEN)
23 (* 17 RAL *) RAL
24 (* 18 ... *) BIEN = FALSE
25 (* 19 DAD D *) DAD (D)
26 (* 1A LDAX D*) LDAX (D , BIEN)
27 (* 1B DCX D *) DCX (D)
28 (* 1C INR E *) INR (E)
29 (* 1D DCR E *) DCR (E)
30 (* 1E MVI E *) MVI (E , BIEN)
31 (* 1F RAR *) RAR
32 (* 20 RIM *) BIEN = FALSE
33 (* 21 LXI H *) LXI (H , BIEN)
34 (* 22 SHLD *) SHLD (BIEN)
35 (* 23 INX H *) INX (H)
36 (* 24 INR H *) INR (H)
37 (* 25 DCR H *) DCR (H)
38 (* 26 MVI H *) MVI (H , BIEN)
39 (* 27 DAA *) DAA
40 (* 28 ... *) BIEN = FALSE
41 (* 29 DAD H *) DAD (H)
42 (* 2A LHLD *) LHLD (BIEN)
43 (* 2B DCX H *) DCX (H)
44 (* 2C INR L *) INR (L)
45 (* 2D DCR L *) DCR (L)
46 (* 2E MVI L *) MVI (L , BIEN)
47 (* 2F CMA *) CMA
48 (* 30 SIM *) BIEN = FALSE
49 (* 31 LXI SP*) LXI (SP HI , BIEN)
50 (* 32 STA *) STA (BIEN)
51 (* 33 INX SP*) INX (SP HI)
52 (* 34 INR M *) INR M (BIEN)
53 (* 35 DCR M *) DCR M (BIEN)
54 (* 36 MVI M *) MVI M (BIEN)
55 (* 37 STC *) STC
56 (* 38 ... *) BIEN = FALSE
57 (* 39 DAD SP*) DAD (SP HI)
58 (* 3A LDA *) LDA (BIEN)
59 (* 3B DCX SP*) DCX (SP HI)
60 (* 3C INR A *) INR (ACC)
61 (* 3D DCR A *) DCR (ACC)
62 (* 3E MVI A *) MVI (ACC , BIEN)
63 (* 3F CMC *) CMC
64 (* 40 MOV B , B *) CPU (. B .) = CPU (. B .)
65 (* 41 MOV B , C *)
66 CPU (. B .) = CPU (. C .)
67 (* 42 MOV B , D *)
68 CPU (. B .) = CPU (. D .)
69 (* 43 MOV B , E *)
70 CPU (. B .) = CPU (. E .)

```

```

68 (* 44 MOV B , H *)
 CPU (. B .) = CPU (. H .)
69 (* 45 MOV B , L *)
 CPU (. B .) = CPU (. L .)
70 (* 46 MOV B , M *) MOV R M(B , BIEN)
71 (* 47 MOV B , A *)
 CPU (. B .) = CPU (. ACC .)
72 (* 48 MOV C , B *)
 CPU (. C .) = CPU (. B .)
73 (* 49 MOV C , C *)
 CPU (. C .) = CPU (. C .)
74 (* 4A MOV C , D *)
 CPU (. C .) = CPU (. D .)
75 (* 4B MOV C , E *)
 CPU (. C .) = CPU (. E .)
76 (* 4C MOV C , H *)
 CPU (. C .) = CPU (. H .)
77 (* 4D MOV C , L *)
 CPU (. C .) = CPU (. L .)
78 (* 4E MOV C , M *) MOV R M(C , BIEN)
79 (* 4F MOV C , ACC *)
 CPU (. C .) = CPU (. ACC .)
80 (* 50 MOV D , B *)
 CPU (. D .) = CPU (. B .)
81 (* 51 MOV D , C *)
 CPU (. D .) = CPU (. C .)
82 (* 52 MOV D , D *) CPU (. D .) = CPU (. D .)
83 (* 53 MOV D , E *)
 CPU (. D .) = CPU (. E .)
84 (* 54 MOV D , H *)
 CPU (. D .) = CPU (. H .)
85 (* 55 MOV D , L *)
 CPU (. D .) = CPU (. L .)
86 (* 56 MOV D , M *) MOV R M(D , BIEN)
87 (* 57 MOV D , A *)
 CPU (. D .) = CPU (. ACC .)
88 (* 58 MOV E , B *)
 CPU (. E .) = CPU (. B .)
89 (* 59 MOV E , C *)
 CPU (. E .) = CPU (. C .)
90 (* 5A MOV E , D *)
 CPU (. E .) = CPU (. D .)
91 (* 5B MOV E , E *) CPU (. E .) = CPU (. E .)
92 (* 5C MOV E , H *)
 CPU (. E .) = CPU (. H .)
93 (* 5D MOV E , L *)
 CPU (. E .) = CPU (. L .)
94 (* 5E MOV E , M *) MOV R M(E , BIEN)
95 (* 5F MOV E , A *)
 CPU (. E .) = CPU (. ACC .)
96 (* 60 MOV H , B *)
 CPU (. H .) = CPU (. B .)
97 (* 61 MOV H , C *)
 CPU (. H .) = CPU (. C .)
98 (* 62 MOV H , D *)

```

```

CPU (. H .) = CPU (. D .)
99 (* 63 MOV H , E *)
CPU (. H .) = CPU (. E .)
100 (* 64 MOV H , H *) CPU (. H .) = CPU (. H .)
101 (* 65 MOV H , L *)
CPU (. H .) = CPU (. L .)
102 (* 66 MOV H , M *) MOV R M(H , BIEN)
103 (* 67 MOV H , A *)
CPU (. H .) = CPU (. ACC .)
104 (* 68 MOV L , B *)
CPU (. L .) = CPU (. B .)
105 (* 69 MOV L , C *)
CPU (. L .) = CPU (. C .)
106 (* 6A MOV L , D *)
CPU (. L .) = CPU (. D .)
107 (* 6B MOV L , E *)
CPU (. L .) = CPU (. E .)
108 (* 6C MOV L , H *)
CPU (. L .) = CPU (. H .)
109 (* 6D MOV L , L *) CPU (. L .) = CPU (. L .)
110 (* 6E MOV L , M *) MOV R M(L , BIEN)
111 (* 6F MOV L , A *) CPU (. L .) = CPU (. ACC .)
112 (* 70 MOV M , B *) MOV M R (B , BIEN)
113 (* 71 MOV M , C *) MOV M R (C , BIEN)
114 (* 72 MOV M , D *) MOV M R (D , BIEN)
115 (* 73 MOV M , E *) MOV M R (E , BIEN)
116 (* 74 MOV M , H *) MOV M R (H , BIEN)
117 (* 75 MOV M , L *) MOV M R (L , BIEN)
118 (* 76 HALT *) FINAL = TRUE
119 (* 77 MOV M , A *) MOV M R (ACC , BIEN)
120 (* 78 MOV A , B *) CPU (. ACC .) = CPU (. B .)
121 (* 79 MOV A , C *) CPU (. ACC .) = CPU (. C .)
122 (* 7A MOV A , D *) CPU (. ACC .) = CPU (. D .)
123 (* 7B MOV A , E *) CPU (. ACC .) = CPU (. E .)
124 (* 7C MOV A , H *) CPU (. ACC .) = CPU (. H .)
125 (* 7D MOV A , L *) CPU (. ACC .) = CPU (. L .)
126 (* 7E MOV A , M *) MOV R M (ACC , BIEN)
127 (* 7F MOV A , A *) CPU (. ACC .) = CPU (. ACC .)
128 (* 80 ADD B *) ADD (B)
129 (* 81 ADD C *) ADD (C)
130 (* 82 ADD D *) ADD (D)
131 (* 83 ADD E *) ADD (E)
132 (* 84 ADD H *) ADD (H)
133 (* 85 ADD L *) ADD (L)
134 (* 86 ADD M *) ADD M (BIEN)
135 (* 87 ADD A *) ADD (ACC)
136 (* 88 ADC B *) ADC (B)
137 (* 89 ADC C *) ADC (C)
138 (* 8A ADC D *) ADC (D)
139 (* 8B ADC E *) ADC (E)
140 (* 8C ADC H *) ADC (H)
141 (* 8D ADC L *) ADC (L)
142 (* 8E ADC M *) ADC M (BIEN)
143 (* 8F ADC A *) ADC (ACC)
144 (* 90 SUB B *) SUB (B)

```

|     |              |                            |
|-----|--------------|----------------------------|
| 145 | (* 91 SUB C  | *) SUB ( C )               |
| 146 | (* 92 SUB D  | *) SUB ( D )               |
| 147 | (* 93 SUB E  | *) SUB ( E )               |
| 148 | (* 94 SUB H  | *) SUB ( H )               |
| 149 | (* 95 SUB L  | *) SUB ( L )               |
| 150 | (* 96 SUB M  | *) SUB M ( BIEN )          |
| 151 | (* 97 SUB A  | *) SUB ( ACC )             |
| 152 | (* 98 SBB B  | *) SBB ( B )               |
| 153 | (* 99 SBB C  | *) SBB ( C )               |
| 154 | (* 9A SBB D  | *) SBB ( D )               |
| 155 | (* 9B SBB E  | *) SBB ( E )               |
| 156 | (* 9C SBB H  | *) SBB ( H )               |
| 157 | (* 9D SBB L  | *) SBB ( L )               |
| 158 | (* 9E SBB M  | *) SBB M ( BIEN )          |
| 159 | (* 9F SBB A  | *) SBB ( ACC )             |
| 160 | (* 40 ANA B  | *) ANA ( B )               |
| 161 | (* 41 ANA C  | *) ANA ( C )               |
| 162 | (* 42 ANA D  | *) ANA ( D )               |
| 163 | (* 43 ANA E  | *) ANA ( E )               |
| 164 | (* 44 ANA H  | *) ANA ( H )               |
| 165 | (* 45 ANA L  | *) ANA ( L )               |
| 166 | (* 46 ANA M  | *) ANA M ( BIEN )          |
| 167 | (* 47 ANA A  | *) ANA ( ACC )             |
| 168 | (* 48 XRA B  | *) XRA ( B )               |
| 169 | (* 49 XRA C  | *) XRA ( C )               |
| 170 | (* 4A XRA D  | *) XRA ( D )               |
| 171 | (* 4B XRA E  | *) XRA ( E )               |
| 172 | (* 4C XRA H  | *) XRA ( H )               |
| 173 | (* 4D XRA L  | *) XRA ( L )               |
| 174 | (* 4E XRA M  | *) XRA M ( BIEN )          |
| 175 | (* 4F XRA A  | *) XRA ( ACC )             |
| 176 | (* B0 ORA B  | *) ORA ( B )               |
| 177 | (* B1 ORA C  | *) ORA ( C )               |
| 178 | (* B2 ORA D  | *) ORA ( D )               |
| 179 | (* B3 ORA E  | *) ORA ( E )               |
| 180 | (* B4 ORA H  | *) ORA ( H )               |
| 181 | (* B5 ORA L  | *) ORA ( L )               |
| 182 | (* B6 ORA M  | *) ORA M ( BIEN )          |
| 183 | (* B7 ORA A  | *) ORA ( ACC )             |
| 184 | (* B8 CMP B  | *) CMP ( B )               |
| 185 | (* B9 CMP C  | *) CMP ( C )               |
| 186 | (* BA CMP D  | *) CMP ( D )               |
| 187 | (* BB CMP E  | *) CMP ( E )               |
| 188 | (* BC CMP H  | *) CMP ( H )               |
| 189 | (* BD CMP L  | *) CMP ( L )               |
| 190 | (* BE CMP M  | *) CMP M ( BIEN )          |
| 191 | (* BF CMP A  | *) CMP ( ACC )             |
| 192 | (* C0 RNZ    | *) RN FLAG ( ZERO , BIEN ) |
| 193 | (* C1 POP B  | *) POP ( B , BIEN )        |
| 194 | (* C2 JNZ    | *) JN FLAG ( ZERO , BIEN ) |
| 195 | (* C3 JMP    | *) JUMP ( BIEN )           |
| 196 | (* C4 CNZ    | *) CN FLAG ( ZERO , BIEN ) |
| 197 | (* C5 PUSH B | *) PUSH ( B , BIEN )       |
| 198 | (* C6 ADI    | *) ADI ( BIEN )            |
| 199 | (* C7 RST 0  | *) RST ( 0 , BIEN )        |

|     |                |                             |
|-----|----------------|-----------------------------|
| 200 | (* C8 RZ       | *) R FLAG ( ZERO , BIEN )   |
| 201 | (* C9 RET      | *) RET ( BIEN )             |
| 202 | (* CA JZ       | *) J FLAG ( ZERO , BIEN )   |
| 203 | (* CB ...      | *) BIEN = FALSE             |
| 204 | (* CC CZ       | *) C FLAG ( ZERO , BIEN )   |
| 205 | (* CD CALL     | *) CALL ( BIEN )            |
| 206 | (* CE ACI      | *) ACI ( BIEN )             |
| 207 | (* CF RST 1    | *) RST ( 1 , BIEN )         |
| 208 | (* D0 RNC      | *) RN FLAG ( CARRY , BIEN ) |
| 209 | (* D1 POP D    | *) POP ( D , BIEN )         |
| 210 | (* D2 JNC      | *) JN FLAG ( CARRY , BIEN ) |
| 211 | (* D3 OUT      | *) OUT ( BIEN )             |
| 212 | (* D4 CNC      | *) CN FLAG ( CARRY,BIEN )   |
| 213 | (* D5 PUSH D   | *) PUSH ( D,BIEN )          |
| 214 | (* D6 SUI      | *) SUI ( BIEN )             |
| 215 | (* D7 RST 2    | *) RST (2 , BIEN)           |
| 216 | (* D8 RC       | *) R FLAG (CARRY ,BIEN )    |
| 217 | (* D9 ...      | *) BIEN = FALSE             |
| 218 | (* DA JC       | *) J FLAG (CARRY , BIEN )   |
| 219 | (* DB IN       | *) INP (BIEN )              |
| 220 | (* DC CC       | *) C FLAG ( CARRY , BIEN )  |
| 221 | (* DD ...      | *) BIEN =FALSE              |
| 222 | (* DE SBI      | *) SBI (BIEN )              |
| 223 | (* DF RST 3    | *) RST (3 ,BIEN)            |
| 224 | (* E0 RPO      | *) RN FLAG (PARITY,BIEN)    |
| 225 | (* E1 POP H    | *) POP (H,BIEN)             |
| 226 | (* E2 JPO      | *) JN FLAG (PARITY,BIEN)    |
| 227 | (* E3 XTHL     | *) XTHL(BIEN)               |
| 228 | (* E4 CPO      | *) CN FLAG (PARITY,BIEN)    |
| 229 | (* E5 PUSH H   | *) PUSH (H,BIEN)            |
| 230 | (* E6 ANI      | *) ANI (BIEN)               |
| 231 | (* E7 RST 4    | *) RST (4,BIEN )            |
| 232 | (* E8 RPE      | *) R FLAG(PARITY,BIEN)      |
| 233 | (* E9 PCHL     | *) PCHL                     |
| 234 | (* EA JPE      | *) J FLAG (PARITY,BIEN)     |
| 235 | (* EB XCHG     | *) XCHG                     |
| 236 | (* EC CPE      | *) C FLAG (PARITY,BIEN)     |
| 237 | (* ED ...      | *) BIEN =FALSE              |
| 238 | (* EE XRI      | *) XRI(BIEN)                |
| 239 | (* EF RST 5    | *) RST (5,BIEN)             |
| 240 | (* F0 RP       | *) RN FLAG(SIGNE,BIEN)      |
| 241 | (* F1 POP PSW  | *) POP ( ACC , BIEN )       |
| 242 | (* F2 JP       | *) JN FLAG( SIGNE , BIEN )  |
| 243 | (* F3 DI       | *) BIEN =FALSE              |
| 244 | (* F4 CP       | *) CN FLAG(SIGNE,BIEN)      |
| 245 | (* F5 PUSH PSW | *) PUSH (ACC,BIEN)          |
| 246 | (* F6 ORI      | *) ORI(BIEN)                |
| 247 | (* F7 RST 6    | *) RST(6,BIEN)              |
| 248 | (* F8 RM       | *) R FLAG(SIGNE,BIEN)       |
| 249 | (* F9 SPHL     | *) SPHL                     |
| 250 | (* FA JM       | *) J FLAG(SIGNE,BIEN)       |
| 251 | (* FB EI       | *) BIEN =FALSE              |
| 252 | (* FC CM       | *) C FLAG(SIGNE,BIEN)       |
| 253 | (* FD ...      | *) BIEN =FALSE              |
| 254 | (* FE CPI      | *) CPI(BIEN)                |

```
255 (* FF RST 7 *) RST(7,BIEN)
 OTHERWISE BIEN = FALSE
```

END

END

FUNCTION FIN PASO A PASO BOOLEAN

VAR

CAR STRING ( 1 )

CONST

CARFINAL = '.'

BEGIN

READLN ( CAR )

FIN PASO A PASO = ( CAR = CARFINAL )

END

BEGIN

BIEN = TRUE FINAL = FALSE

CMS ( 'CLRSCRN' , RC )

TECL DIRECCION ( HEXDIRHI , HEXDIRLO , DIRHI , DIRLO , BIEN )

CPU ( . PC HI . ) = DIRHI

CPU ( . PC LC . ) = DIRLO

STATUS = BUSQUEDA CONTADOR = 0

VER CPU ( OUTPUT , BIEN )

VER PAGINA ( HEXDIRHI , DIRHI )

REPEAT

REPEAT

CONTADOR = CONTADOR + 1

CASE STATUS OF

BUSQUEDA BEGIN

FASE BUSQUEDA ( BIEN )

IF BIEN

THEN STATUS = EJECUTAR

ELSE BEGIN

WRITELN ( 'ERROR EN BUSQUEDA ' )

DEMORA

STATUS = FIN

END

END

EJECUTAR BEGIN

FASE EJECUCION ( BIEN , FINAL )

CONTADOR = CONTADOR + 1

IF BIEN

THEN BEGIN

IF FINAL

THEN STATUS = FIN

ELSE STATUS = BUSQUEDA

END

ELSE BEGIN

WRITELN ( 'ERROR EN EJECUCION 1 ' )

DEMORA

STATUS = FIN

END

VER CPU ( OUTPUT , BIEN )

DIRHI = CPU ( . PC HI . )

ENTEROHEXA ( DIRHI , HEXDIRHI , BIEN )

VER PAGINA ( HEXDIRHI , DIRHI )

END

```

 FIN WRITELN ('FIN EXECUTE ')
 END
 UNTIL (CONTADOR = 1000) OR (STATUS = FIN) OR (PASO A PASO)
 IF (CONTADOR = 1000) AND
 RESPUESTA ('EJECUTE MIL OPERACIONES . ME DETENGO ')
 THEN STATUS = FIN
 ELSE CONTADOR = 0
 UNTIL (STATUS = FIN) OR (PASO A PASO AND FIN PASO A PASO)
 VER CPU (OUTPUT , BIEN)
 DEMORA
END
PROCEDURE MAIN
CONST
 CENTRADO = 50
VAR
 BIEN , FINAL BOOLEAN
 OPCION STRING (1)
BEGIN
 PASO A PASO = FALSE ACARREO = FALSE MEDIO ACARREO = FALSE
 TERMIN (INPUT) TERMOUT (OUTPUT)
 BIEN = TRUE
 FINAL = FALSE
 MEM INICIALIZACION
 CPU INICIALIZACION
 REPEAT
 CMS ('CLRSCRN' , COM CMS)
 WRITELN ('0' FIN ' CENTRADO)
 WRITELN ('1' EDICION DE MEMORIA' CENTRADO)
 WRITELN ('2' CPU INICIALIZACION' CENTRADO)
 WRITELN ('3' GUARDO CPU ' CENTRADO)
 WRITELN ('4' RECUPERO CPU ' CENTRADO)
 WRITELN ('5' INICIALIZO MEMORIA' CENTRADO)
 WRITELN ('6' GUARDO MEMORIA ' CENTRADO)
 WRITELN ('7' RECUPERO MEMORIA ' CENTRADO)
 WRITELN ('8' EXECUTE ' CENTRADO)
 WRITELN ('9' VER PAGINA ' CENTRADO)
 WRITELN ('A' VER CPU ' CENTRADO)
 WRITELN ('B' EDICION DE CPU ' CENTRADO)
 WRITE ('C' PASO A PASO ' CENTRADO)
 IF PASO A PASO
 THEN WRITELN ('SI')
 ELSE WRITELN ('NO')
 WRITELN ('BIEN ' CENTRADO , BIEN)
 READ (OPCION)
 CASE OPCION (. 1 .) OF
 '0' FINAL = TRUE
 '1' MEM EDITAR (BIEN)
 '2' CPU INICIALIZACION
 '3' CPU BACKUP ACTUALIZACION (BIEN)
 '4' CPU BACKUP RECUPERACION (BIEN)
 '5' MEM INICIALIZACION
 '6' MEM BACKUP ACTUALIZACION (BIEN)
 '7' MEM BACKUP RECUPERACION (BIEN)
 '8' EXECUTE (BIEN)
 '9' VER MEMORIA (BIEN)

```

```
'A','A' BEGIN
 VER CPU (OUTPUT , BIEN)
 DEMORA
 END
'B','B' CPU EDICION (BIEN)
'C','C' PASO A PASO = NOT PASO A PASO
 OTHERWISE WRITELN ('ERROR')
 END
IF NOT BIEN THEN FINAL = TRUE
UNTIL FINAL
END
BEGIN
 MAIN
END .
```

## MEMBER TIPOS

|                         |              |             |              |          |          |
|-------------------------|--------------|-------------|--------------|----------|----------|
| FUNCTION ESTBYTE        |              |             |              |          | TIP00010 |
| ( UNDISTINTIVO          | DISTINTIVO ) | BOOLEAN     | EXTERNAL     |          | TIP00020 |
| FUNCTION ESTWORD        |              |             |              |          | TIP00030 |
| ( UNDISTINTIVO          | DISTINTIVO ) | BOOLEAN     | EXTERNAL     |          | TIP00040 |
| PROCEDURE HEXATWORD     |              |             |              |          | TIP00050 |
| ( UNHEXA                | DISTINTIVO   | VAR UNTWORD | TWORD )      | EXTERNAL | TIP00060 |
| PROCEDURE TWORDHEXA     |              |             |              |          | TIP00070 |
| ( UNTWORD               | TWORD        | VAR UNHEXA  | DISTINTIVO ) | EXTERNAL | TIP00080 |
| PROCEDURE HEXATBYTE     |              |             |              |          | TIP00090 |
| ( UNHEXA                | DISTINTIVO   | VAR UNTBYTE | TBYTE )      | EXTERNAL | TIP00100 |
| PROCEDURE TBYTEHEXA     |              |             |              |          | TIP00110 |
| ( UNTBYTE               | TBYTE        | VAR UNHEXA  | DISTINTIVO ) | EXTERNAL | TIP00120 |
| FUNCTION ESDEC          |              |             |              |          | TIP00130 |
| ( UN DEC                | DISTINTIVO ) | BOOLEAN     | EXTERNAL     |          | TIP00140 |
| FUNCTION ESOC           |              |             |              |          | TIP00150 |
| ( UN OCT                | DISTINTIVO ) | BOOLEAN     | EXTERNAL     |          | TIP00160 |
| FUNCTION ESBIN          |              |             |              |          | TIP00170 |
| ( UN BIN                | DISTINTIVO ) | BOOLEAN     | EXTERNAL     |          | TIP00180 |
| FUNCTION PASARHEXATBYTE |              |             |              |          | TIP00190 |
| ( LA PALABRA            | DISTINTIVO   | VAR VALOR   | TBYTE )      |          | TIP00200 |
| BOOLEAN                 | EXTERNAL     |             |              |          | TIP00210 |
| FUNCTION PASARHEXATWORD |              |             |              |          | TIP00220 |
| ( LA PALABRA            | DISTINTIVO   | VAR VALOR   | TWORD )      |          | TIP00230 |
| BOOLEAN                 | EXTERNAL     |             |              |          | TIP00240 |

/ /

```

MEMBER TABLA
FUNCTION DETECTAR IDENTIFICADOR
(CONST LA PALABRA DISTINTIVO
 VAR UN ARREGLO SIMBOLOS ARREGLO SIMBOLOS
 CONST EL TOPE SIMBOLOS INDICEIDENTIFICADOR
 VAR I INDICEIDENTIFICADOR)
 BOOLEAN EXTERNAL
FUNCTION INSERTAR IDENTIFICADOR
(VAR EL ARREGLO SIMBOLOS ARREGLO SIMBOLOS
 VAR EL INDICE SIMBOLOS INDICEIDENTIFICADOR
 CONST LA PALABRA DISTINTIVO
 BOOLEAN EXTERNAL
FUNCTION ASIGNAR ETIQUETA
(ESTEVALOR TWORD
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR)
 BOOLEAN EXTERNAL
FUNCTION PROCESAR ETIQUETA
(CONST ETIQUETA DISTINTIVO
 VAR VALOR TWORD
 BOOLEAN EXTERNAL
FUNCTION ASIGNAR TBYTE
(ESTEVALOR TBYTE
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR)
 BOOLEAN EXTERNAL
FUNCTION ASIGNAR TWORD
(ESTEVALOR TWORD
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR)
 BOOLEAN EXTERNAL
FUNCTION ASIGNAR DEC
(ESTEVALOR DISTINTIVO
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR)
 BOOLEAN EXTERNAL
FUNCTION ASIGNAR CCT
(ESTEVALOR DISTINTIVO
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR)
 BOOLEAN EXTERNAL
FUNCTION ASIGNAR BIN
(ESTEVALOR DISTINTIVO
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR)
 BOOLEAN EXTERNAL
FUNCTION ASIGNAR ALFABETICO
(CONST ESTEVALOR DISTINTIVO
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR)
 BOOLEAN EXTERNAL
FUNCTION INS IDENTIFICADOR
(CONST UNIDENTIFICADOR DISTINTIVO
 UNACLASEDECLARACION CLASEDECLARACION
 VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR)
 BOOLEAN EXTERNAL
INCLUDE EXTERNOS(PROCTIPO)
/ /

```

TAB00010  
 TAB00020  
 TAB00030  
 TAB00040  
 TAB00050  
 TAB00060  
 TAB00070  
 TAB00080  
 TAB00090  
 TAB00100  
 TAB00110  
 TAB00120  
 TAB00130  
 TAB00140  
 TAB00150  
 TAB00160  
 TAB00170  
 TAB00180  
 TAB00190  
 TAB00200  
 TAB00210  
 TAB00220  
 TAB00230  
 TAB00240  
 TAB00250  
 TAB00260  
 TAB00270  
 TAB00280  
 TAB00290  
 TAB00300  
 TAB00310  
 TAB00320  
 TAB00330  
 TAB00340  
 TAB00350  
 TAB00360  
 TAB00370  
 TAB00380  
 TAB00390  
 TAB00400  
 TAB00410  
 TAB00420  
 TAB00430  
 TAB00440  
 TAB00450  
 TAB00460  
 TAB00470  
 TAB00480  
 TAB00490

MEMBER PROCTIPO  
TYPE

|                                |         |          |          |
|--------------------------------|---------|----------|----------|
| INCLUDE ESTRUCTURA (RESOLVE)   |         |          | PR000010 |
| FUNCTION PROC ETIQUETA         |         |          | PR000020 |
| ( VAR LA PALABRA DISTINTIVO    |         |          | PR000030 |
| VAR UN RESOLVENTE RESOLVENTE ) | BOOLEAN | EXTERNAL | PR000040 |
| FUNCTION PROC ESPECIAL         |         |          | PR000050 |
| ( VAR LA PALABRA DISTINTIVO    |         |          | PR000060 |
| VAR UN RESOLVENTE RESOLVENTE ) | BOOLEAN | EXTERNAL | PR000070 |
| FUNCTION PROC WORD             |         |          | PR000080 |
| ( VAR LA PALABRA DISTINTIVO    |         |          | PR000090 |
| VAR UN RESOLVENTE RESOLVENTE ) | BOOLEAN | EXTERNAL | PR000100 |
| FUNCTION PROC BYTE             |         |          | PR000110 |
| ( VAR LA PALABRA DISTINTIVO    |         |          | PR000120 |
| VAR UN RESOLVENTE RESOLVENTE ) | BOOLEAN | EXTERNAL | PR000130 |
| FUNCTION PROC NUMBYTE          |         |          | PR000140 |
| ( VAR LA PALABRA DISTINTIVO    |         |          | PR000150 |
| VAR UN RESOLVENTE RESOLVENTE ) | BOOLEAN | EXTERNAL | PR000160 |
| FUNCTION PROC NUMWORD          |         |          | PR000170 |
| ( VAR LA PALABRA DISTINTIVO    |         |          | PR000180 |
| VAR UN RESOLVENTE RESOLVENTE ) | BOOLEAN | EXTERNAL | PR000190 |
|                                |         |          | PR000200 |

FILE EXTERNOS MACLIB 41 VM/SP RELEASE 5 \*\*\* U.N.P.S.J.B \*\*\* PUT 8801+

MEMBER FASE

PROCEDURE ENSAMBLADOR

( VAR ENTRADA , AUXILIAR TEXT ) ENTRY

FAS00010

FAS00020

# APENDICE IV

IMPRESION PROGRAMA ENSAMBLADOR

```

MEMBER ARCHIVO
(******)
(* ARCHIVO
SEGMENT ARCHIVO

PROCEDURE CMS (CONST S STRING VAR RC INTEGER) EXTERNAL
VAR RC INTEGER

PROCEDURE LEER NOMBRE ARCHIVO
(VAR CMS NOMBRE ,
 FILE NOMBRE STRING
 VAR BIEN BOOLEAN) ENTRY
VAR
 F NAME , F TYPE , F MODE ALPHA
 LINE STRING (80) (* LINEA DE PANTALLA *)
 POS , LONG LINE 0 .. 80
BEGIN
 WRITELN ('NOMBRE , TIPO Y DISCO DEL ARCHIVO ')
 READ (LINE) POS = 1 BIEN = TRUE
 LONG LINE = LENGTH (LINE)
 IF POS = LONG LINE
 THEN
 TOKEN (POS , LINE , F NAME)
 ELSE BIEN = FALSE
 IF POS = LONG LINE
 THEN
 TOKEN (POS , LINE , F TYPE)
 ELSE BIEN = FALSE
 IF POS = LONG LINE
 THEN
 TOKEN (POS , LINE , F MODE)
 ELSE BIEN = FALSE
 IF BIEN
 THEN
 BEGIN
 WRITESTR (CMS NOMBRE ,
 'STATE ' , F NAME 9 ,
 F TYPE 9 ,
 F MODE 9)
 WRITESTR (FILE NOMBRE ,
 'NAME = ' , F NAME 8 , ' ' ,
 F TYPE 8 , ' ' ,
 F MODE 8)
 CMS NOMBRE = COMPRESS (CMS NOMBRE)
 FILE NOMBRE = COMPRESS (FILE NOMBRE)
 END
 END
END
PROCEDURE VERIFICAR ARCHIVO
(CONST CMS NOMBRE STRING
 VAR BIEN BOOLEAN) ENTRY
BEGIN
 CMS (CMS NOMBRE , RC)
 BIEN = (RC = 0)
END
(* FIN ARCHIVO

```

```

*) ARC00010
*) ARC00020
ARC00030
ARC00040
ARC00050
ARC00060
ARC00070
ARC00080
ARC00090
ARC00100
ARC00110
ARC00120
ARC00130
ARC00140
ARC00150
ARC00160
ARC00170
ARC00180
ARC00190
ARC00200
ARC00210
ARC00220
ARC00230
ARC00240
ARC00250
ARC00260
ARC00270
ARC00280
ARC00290
ARC00300
ARC00310
ARC00320
ARC00330
ARC00340
ARC00350
ARC00360
ARC00370
ARC00380
ARC00390
ARC00400
ARC00410
ARC00420
ARC00430
ARC00440
ARC00450
ARC00460
ARC00470
ARC00480
ARC00490
ARC00500
ARC00510
ARC00520
ARC00530
*) ARC00540

```



```

MEMBER ENSAMBLA
PROGRAM ENSAMBLA (INPUT , OUTPUT , ENTRADA , AUXILIAR , SALIDA)
PROCEDURE ENSAMBLADOR
 (VAR ENTRADA , AUXILIAR TEXT) EXTERNAL
FUNCTION FINAL
 (VAR AUXILIAR , SALIDA TEXT) BOOLEAN EXTERNAL
PROCEDURE LEER NOMBRE ARCHIVO
 (VAR CMS NOMBRE ,
 FILE NOMBRE STRING
 VAR BIEN BOOLEAN) EXTERNAL
PROCEDURE VERIFICAR ARCHIVO
 (CONST CMS NOMBRE STRING
 VAR BIEN BOOLEAN) EXTERNAL
PROCEDURE PRINCIPAL (CONST FILEENTRADA , FILSALIDA STRING)
VAR
 ENTRADA , AUXILIAR , SALIDA TEXT
BEGIN
 RESET (ENTRADA , FILEENTRADA)
 REWRITE (AUXILIAR , 'NAME=AUXILIAR.ENS.41')
 ENSAMBLADOR (ENTRADA , AUXILIAR)
 CLOSE (ENTRADA)
 CLOSE (AUXILIAR)
 RESET (AUXILIAR , 'NAME=AUXILIAR.ENS.41')
 REWRITE (SALIDA , FILSALIDA)
 WRITELN ('BIEN = ' , FINAL (AUXILIAR , SALIDA))
 CLOSE (ENTRADA)
 CLOSE (AUXILIAR)
 CLOSE (SALIDA)
END
VAR
 CMSENTRADA , CMSSALIDA , FILEENTRADA , FILSALIDA STRING (80)
 BIEN BOOLEAN
BEGIN
 LEER NOMBRE ARCHIVO (CMSENTRADA , FILEENTRADA , BIEN)
 LEER NOMBRE ARCHIVO (CMSSALIDA , FILSALIDA , BIEN)
 VERIFICAR ARCHIVO (CMSENTRADA , BIEN)
 IF BIEN THEN PRINCIPAL (FILEENTRADA , FILSALIDA)
END .
/ /

```

```

ENS00010
ENS00020
ENS00030
ENS00040
ENS00050
ENS00060
ENS00070
ENS00080
ENS00090
ENS00100
ENS00110
ENS00120
ENS00130
ENS00140
ENS00150
ENS00160
ENS00170
ENS00180
ENS00190
ENS00200
ENS00210
ENS00220
ENS00230
ENS00240
ENS00250
ENS00260
ENS00270
ENS00280
ENS00290
ENS00300
ENS00310
ENS00320
ENS00330
ENS00340
ENS00350
ENS00360
ENS00370

```

```

MEMBER FASE
SEGMENT FASE
INCLUDE ESTRUCTURA (DISTINTIVO)
INCLUDE ESTRUCTURA (ESTTIPOS)
INCLUDE EXTERNOS (TIPOS)
INCLUDE ESTRUCTURA (ESTTABLA)
INCLUDE ESTRUCTURA (TABLA)
INCLUDE EXTERNOS (TABLA)
PROCEDURE ENSAMBLADOR
(VAR ENTRADA , AUXILIAR TEXT) ENTRY
(*****)
(* DESDE AQUI A NUEVO AVISO ES GENERAL PARA TODO LOS ENSAMBLADORES *)
CONST
 LONGLINEA = 80 (* LONGITUD LINEA DE TEXTO *)
TYPE
 LINEA = STRING (80) (* LINEA DE TEXTO PARA ENSAMBLAR *)
(*
 REGLAS DEL ENSAMBLADOR
*)
STATIC
 LA LINEA LINEA
 LA LONGITUD LINEA 0 .. LONGLINEA
 PCS 1 .. LONGLINEA + 1
 LA PALABRA DISTINTIVO
 TRADUCCION LINEA
VALUE
 LA LINEA = ' '
 LA LONGITUD LINEA = 0
 PCS = 1
 LA PALABRA = ' '
STATIC
 ELORIGEN , PUNTERO TWORD
 HEXAORIGEN , HEXAPUNTERO DISTINTIVO
VALUE
 ELORIGEN = 0 PUNTERO = 0
 HEXAORIGEN = '0000' HEXAPUNTERO = '0000'
FUNCTION ERROR
(VAR ARCHIVO TEXT CONST MEMERROR STRING) BOOLEAN
BEGIN
 WRITELN (ARCHIVO , LA LINEA)
 IF POS 1 THEN
 WRITELN (ARCHIVO , ' ' POS - 1)
 ELSE
 WRITELN (ARCHIVO , ' ')
 WRITELN (ARCHIVO , MEMERROR)
 ERROR = FALSE
END
CONST
 ERRPALABRA = 'FALTA PALABRA'
 ERRLINEA = 'FIN INESPERADO DE LINEA'
 ERRSALTOLINEA = 'FIN INESPERADO DE ARCHIVO FUENTE'
 ERRPUNTOYCOMA = 'SE ESPERABA '
 ERRCONMEMONICO = 'SE ESPERABA UN MEMONICO'
 ERRSENTENCIA = 'SENTENCIA IRRECONOCIBLE'
 ERRCUERPO = 'FALTA FIN DEL PROGRAMA'

```

FAS00010  
 FAS00020  
 FAS00030  
 FAS00040  
 FAS00050  
 FAS00060  
 FAS00070  
 FAS00080  
 FAS00090  
 FAS00100  
 FAS00110  
 FAS00120  
 FAS00130  
 FAS00140  
 FAS00150  
 FAS00160  
 FAS00170  
 FAS00180  
 FAS00190  
 FAS00200  
 FAS00210  
 FAS00220  
 FAS00230  
 FAS00240  
 FAS00250  
 FAS00260  
 FAS00270  
 FAS00280  
 FAS00290  
 FAS00300  
 FAS00310  
 FAS00320  
 FAS00330  
 FAS00340  
 FAS00350  
 FAS00360  
 FAS00370  
 FAS00380  
 FAS00390  
 FAS00400  
 FAS00410  
 FAS00420  
 FAS00430  
 FAS00440  
 FAS00450  
 FAS00460  
 FAS00470  
 FAS00480  
 FAS00490  
 FAS00500  
 FAS00510  
 FAS00520  
 FAS00530  
 FAS00540

```

ERRIDENTIFICADOR = 'IDENTIFICADOR MAL FORMADO O NO DECLARADO'
ERRDECLARACION = 'DECLARACION MAL FORMADA'
ERRUNDECLARACIONES = 'NO SE ENCUENTRA EL INICIO DEL PROGRAMA'
ERRDEFORIGEN = 'DEFINICION DE ORIGEN ERRONEA'
ERRORIGEN = 'ORIGEN ERRONEO'
ERRENCABEZADO = 'ENCABEZADO ERRONEO O NO ENCONTRADO'
ERRPROGRAMA = 'PROGRAMA CON ERRORES'
ERROP = 'NO SE ENCUENTRA MEMONICO'
ERRETIQUETA = 'ETIQUETA MAL DECLARADA'
PROCEDURE PASARAMAYUSCULA
VAR I INTEGER
BEGIN
 FOR I = 1 TO LENGTH (LA PALABRA) DO
 LA PALABRA (. I .) =
 CHR (ORD (LA PALABRA (. I .)) - ORD ('A') + ORD ('A'))
 END
FUNCTION ESLITERAL BOOLEAN
VAR AUX BOOLEAN CAR CHAR
BEGIN
 CAR = LA PALABRA (. 1 .)
 AUX = (CAR IN (. 'A' .. 'Z' .)) OR (CAR IN (. 'a' .. 'z' .))
 IF AUX THEN PASARAMAYUSCULA
END
FUNCTION NUEVALINEA BOOLEAN
BEGIN
 READ (ENTRADA , LA LINEA)
 POS = 1 LA LONGITUD LINEA = LENGTH (LA LINEA)
 NUEVALINEA = (LA LONGITUD LINEA = 1) OR
 ERROR (AUXILIAR , ERRLINEA)
END
FUNCTION SALTOLINEA BOOLEAN
BEGIN
 SALTOLINEA = (NOT EOF (ENTRADA) AND NUEVALINEA) OR
 ERROR (AUXILIAR , ERRSALTOLINEA)
END
FUNCTION OBTENERPALABRA BOOLEAN
VAR BUFFER ALPHA
BEGIN
 TOKEN (POS , LA LINEA , BUFFER)
 WRITESTR (LA PALABRA , BUFFER)
 OBTENERPALABRA = (LA PALABRA = ' ') OR
 ERROR (AUXILIAR , ERRPALABRA)
END
FUNCTION NUEVAPALABRA BOOLEAN
BEGIN
 NUEVAPALABRA = ((POS = LA LONGITUD LINEA) AND
 SALTOLINEA AND
 NUEVALINEA AND
 OBTENERPALABRA) OR
 OBTENERPALABRA
END
(*
PUNTOYCOMAYCOMENTARIO = ' '
*)
FUNCTION PUNTOYCOMAYCOMENTARIO BOOLEAN

```

FAS00550  
 FAS00560  
 FAS00570  
 FAS00580  
 FAS00590  
 FAS00600  
 FAS00610  
 FAS00620  
 FAS00630  
 FAS00640  
 FAS00650  
 FAS00660  
 FAS00670  
 FAS00680  
 FAS00690  
 FAS00700  
 FAS00710  
 FAS00720  
 FAS00730  
 FAS00740  
 FAS00750  
 FAS00760  
 FAS00770  
 FAS00780  
 FAS00790  
 FAS00800  
 FAS00810  
 FAS00820  
 FAS00830  
 FAS00840  
 FAS00850  
 FAS00860  
 FAS00870  
 FAS00880  
 FAS00890  
 FAS00900  
 FAS00910  
 FAS00920  
 FAS00930  
 FAS00940  
 FAS00950  
 FAS00960  
 FAS00970  
 FAS00980  
 FAS00990  
 FAS01000  
 FAS01010  
 FAS01020  
 FAS01030  
 FAS01040  
 FAS01050  
 FAS01060  
 FAS01070  
 FAS01080  
 FAS01090

```

BEGIN
 PUNTOYCOMAYCOMENTARIO = ((LA PALABRA = ' ') AND SALTOLINEA) OR
 ERROR (AUXILIAR , ERRPUNTOYCOMA)
END
(*
 CONMEMONICO = OP , PUNTOYCOMAYCOMENTARIO
*)
INCLUDE PROGSEG (OP)
FUNCTION CONMEMONICO BOOLEAN
BEGIN
 CONMEMONICO = (OP AND PUNTOYCOMAYCOMENTARIO) OR
 ERROR (AUXILIAR , ERRCONMEMONICO)
END
(*
 CONETIQUETA = DISTINTIVO CONMEMONICO
*)
FUNCTION CONETIQUETA BOOLEAN
VAR
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR
BEGIN
 CONETIQUETA =
 PROCESAR ETIQUETA (LA PALABRA , PUNTERO)
 AND CONMEMONICO
END
(*
 SENTENCIA = CONETIQUETA CONMEMONICO
*)
FUNCTION SENTENCIA BOOLEAN
BEGIN
 SENTENCIA = CONMEMONICO OR CONETIQUETA
 WRITELN (AUXILIAR , TRACUCCION)
END
(*
 FIN = 'FIN'
*)
FUNCTION FIN BOOLEAN
BEGIN
 FIN = (LA PALABRA = 'FIN')
END
(*
 CUERPO = *(SENTENCIA) FIN
*)
FUNCTION CUERPO BOOLEAN
BEGIN
 WHILE SENTENCIA DO
 CUERPO = FIN OR ERROR (AUXILIAR , ERRCUERPO)
 END
FUNCTION IDENTIFICADOR
 (UNACLASEDECLARACION CLASEDECLARACION
 VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR) BOOLEAN
BEGIN
 IDENTIFICADOR =
 ESLITERAL AND
 INS IDENTIFICADOR
 (LA PALABRA , UNACLASEDECLARACION , UNINDICEIDENTIFICADOR) OR

```

FAS01100  
 FAS01110  
 FAS01120  
 FAS01130  
 FAS01140  
 FAS01150  
 FAS01160  
 FAS01170  
 FAS01180  
 FAS01190  
 FAS01200  
 FAS01210  
 FAS01220  
 FAS01230  
 FAS01240  
 FAS01250  
 FAS01260  
 FAS01270  
 FAS01280  
 FAS01290  
 FAS01300  
 FAS01310  
 FAS01320  
 FAS01330  
 FAS01340  
 FAS01350  
 FAS01360  
 FAS01370  
 FAS01380  
 FAS01390  
 FAS01400  
 FAS01410  
 FAS01420  
 FAS01430  
 FAS01440  
 FAS01450  
 FAS01460  
 FAS01470  
 FAS01480  
 FAS01490  
 FAS01500  
 FAS01510  
 FAS01520  
 FAS01530  
 FAS01540  
 FAS01550  
 FAS01560  
 FAS01570  
 FAS01580  
 FAS01590  
 FAS01600  
 FAS01610  
 FAS01620  
 FAS01630  
 FAS01640

```

 ERROR (AUXILIAR , ERRIDENTIFICADOR)
END
(*
 UNTBYTE = 'TBYTE' DISTINTIVO TBYTE
*)
FUNCTION UNTBYTE BOOLEAN
VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR
 VALOR TBYTE
BEGIN
 UNTBYTE = (LA PALABRA = 'TBYTE') AND
 NUEVAPALABRA AND
 IDENTIFICADOR (CTBYTE , UNINDICEIDENTIFICADOR) AND
 NUEVAPALABRA AND
 PASARHEXATBYTE (LA PALABRA , VALOR) AND
 ASIGNAR TBYTE (VALOR , UNINDICEIDENTIFICADOR)
END
(*
 UNTWORD = 'TWORD' DISTINTIVO TWORD
*)
FUNCTION UNTWORD BOOLEAN
VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR
 VALOR TWORD
BEGIN
 UNTWORD = (LA PALABRA = 'TWORD') AND
 NUEVAPALABRA AND
 IDENTIFICADOR (CTWORD , UNINDICEIDENTIFICADOR) AND
 NUEVAPALABRA AND
 PASARHEXATWORD (LA PALABRA , VALOR) AND
 ASIGNAR TWORD (VALOR , UNINDICEIDENTIFICADOR)
END
(*
 UNDEC = 'DEC' DISTINTIVO DEC
*)
FUNCTION UNDEC BOOLEAN
VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR
BEGIN
 UNDEC = (LA PALABRA = 'DEC') AND
 NUEVAPALABRA AND
 IDENTIFICADOR (CDEC , UNINDICEIDENTIFICADOR) AND
 NUEVAPALABRA AND
 ESDEC (LA PALABRA) AND
 ASIGNAR DEC (LA PALABRA , UNINDICEIDENTIFICADOR)
END
(*
 UNOCT = 'OCT' DISTINTIVO OCT
*)
FUNCTION UNOCT BOOLEAN
VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR
BEGIN
 UNOCT = (LA PALABRA = 'OCT') AND
 NUEVAPALABRA AND
 IDENTIFICADOR (COCT , UNINDICEIDENTIFICADOR) AND
 NUEVAPALABRA AND
 ESOC (LA PALABRA) AND
 ASIGNAR OCT (LA PALABRA , UNINDICEIDENTIFICADOR)
END

```

FAS01650  
 FAS01660  
 FAS01670  
 FAS01680  
 FAS01690  
 FAS01700  
 FAS01710  
 FAS01720  
 FAS01730  
 FAS01740  
 FAS01750  
 FAS01760  
 FAS01770  
 FAS01780  
 FAS01790  
 FAS01800  
 FAS01810  
 FAS01820  
 FAS01830  
 FAS01840  
 FAS01850  
 FAS01860  
 FAS01870  
 FAS01880  
 FAS01890  
 FAS01900  
 FAS01910  
 FAS01920  
 FAS01930  
 FAS01940  
 FAS01950  
 FAS01960  
 FAS01970  
 FAS01980  
 FAS01990  
 FAS02000  
 FAS02010  
 FAS02020  
 FAS02030  
 FAS02040  
 FAS02050  
 FAS02060  
 FAS02070  
 FAS02080  
 FAS02090  
 FAS02100  
 FAS02110  
 FAS02120  
 FAS02130  
 FAS02140  
 FAS02150  
 FAS02160  
 FAS02170  
 FAS02180  
 FAS02190

```

(*) UNBIN = 'BIN' DISTINTIVO BINARIO
*)
FUNCTION UNBIN BOOLEAN
VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR
BEGIN
 UNBIN = (LA PALABRA = 'BIN') AND
 NUEVAPALABRA AND
 IDENTIFICADOR (CBIN , UNINDICEIDENTIFICADOR) AND
 NUEVAPALABRA AND ESBIN (LA PALABRA) AND
 ASIGNAR BIN (LA PALABRA , UNINDICEIDENTIFICADOR)
END
(*)
 UNALFA = 'ALFA' DISTINTIVO ALFA
*)
FUNCTION UNALFABETICO BOOLEAN
VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR
BEGIN
 UNALFABETICO = (LA PALABRA = 'ALFA') AND
 NUEVAPALABRA AND
 IDENTIFICADOR (CALFABETICO , UNINDICEIDENTIFICADOR) AND
 NUEVAPALABRA AND
 ASIGNAR ALFABETICO (LA PALABRA , UNINDICEIDENTIFICADOR)
END
(*)
 UNAETIQUETA = 'ETIQUETA' DISTINTIVO
*)
FUNCTION UNAETIQUETA BOOLEAN
VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR
 VALOR TWORD
BEGIN
 UNAETIQUETA = (LA PALABRA = 'ETIQUETA') AND
 NUEVAPALABRA AND
 IDENTIFICADOR (CETIQUETA , UNINDICEIDENTIFICADOR) AND
 NUEVAPALABRA AND
 ((PASARHEXATWORD (LA PALABRA , VALOR) AND
 ASIGNAR ETIQUETA (VALOR , UNINDICEIDENTIFICADOR))
 OR (LA PALABRA = ' '))
END
(*)
 DECLARACION = UNTBYTE UNTWORD UNDEC UNOCT UNBIN
 UNALFA UNETIQUETA
*)
FUNCTION DECLARACION BOOLEAN
BEGIN
 DECLARACION = NUEVAPALABRA AND
 (UNTBYTE OR UNTWORD OR UNDEC OR UNOCT OR UNBIN OR
 UNALFABETICO OR UNAETIQUETA OR
 ERROR (AUXILIAR , ERRODECLARACION))
END
(*)
 INICIO = 'INICIO'
*)
FUNCTION INICIO BOOLEAN
BEGIN

```

FAS02200  
 FAS02210  
 FAS02220  
 FAS02230  
 FAS02240  
 FAS02250  
 FAS02260  
 FAS02270  
 FAS02280  
 FAS02290  
 FAS02300  
 FAS02310  
 FAS02320  
 FAS02330  
 FAS02340  
 FAS02350  
 FAS02360  
 FAS02370  
 FAS02380  
 FAS02390  
 FAS02400  
 FAS02410  
 FAS02420  
 FAS02430  
 FAS02440  
 FAS02450  
 FAS02460  
 FAS02470  
 FAS02480  
 FAS02490  
 FAS02500  
 FAS02510  
 FAS02520  
 FAS02530  
 FAS02540  
 FAS02550  
 FAS02560  
 FAS02570  
 FAS02580  
 FAS02590  
 FAS02600  
 FAS02610  
 FAS02620  
 FAS02630  
 FAS02640  
 FAS02650  
 FAS02660  
 FAS02670  
 FAS02680  
 FAS02690  
 FAS02700  
 FAS02710  
 FAS02720  
 FAS02730  
 FAS02740

```

INICIO = (LA PALABRA = 'INICIO')
END
(*)
DECLARACIONES = *(DECLARACION PUNTOYCOMAYCOMENTARIO) INICIO
*)
FUNCTION DECLARACIONES BOOLEAN
VAR
UNDECLARACION , ELINICIO , ELERROR BOOLEAN
BEGIN
WHILE DECLARACION DO
DECLARACIONES = INICIO OR ERROR (AUXILIAR , ERRUNDECLARACIONES)
END
(* DEFINICION VARIABLES GLOBALES *)
FUNCTION COMPUTARORIGEN BOOLEAN
BEGIN
HEXATWORD (HEXAORIGEN , ELORIGEN)
HEXAPUNTERO = HEXAORIGEN
PUNTERO = ELORIGEN
COMPUTARORIGEN = TRUE
END
FUNCTION RELATIVIZARORIGEN BOOLEAN
BEGIN
WRITELN (AUXILIAR , 'ORIGEN RELATIVO')
RELATIVIZARORIGEN = TRUE
END
FUNCTION ABSOLUTO BOOLEAN
BEGIN
ABSOLUTO = ESTWORD (LA PALABRA) AND COMPUTARORIGEN
END
FUNCTION RELATIVO BOOLEAN
BEGIN
RELATIVO = (LA PALABRA = 'RELATIVO') AND RELATIVIZARORIGEN
END
(*)
DEFORIGEN = ABSOLUTO RELATIVO
*)
FUNCTION DEFORIGEN BOOLEAN
BEGIN
DEFORIGEN = ABSOLUTO OR RELATIVO OR
ERROR (AUXILIAR , ERRDEFORIGEN)
END
(*)
ORIGEN = 'ORIGEN'
*)
FUNCTION ORIGEN BOOLEAN
BEGIN
ORIGEN = (LA PALABRA = 'ORIGEN') OR
ERROR (AUXILIAR , ERRORIGEN)
END
(*)
ENCABEZADO = ORIGEN DEFORIGEN PUNTOYCOMAYCOMENTARIO
*)
FUNCTION ENCABEZADO BOOLEAN
BEGIN
ENCABEZADO = (ORIGEN AND DEFORIGEN AND PUNTOYCOMAYCOMENTARIO) OR

```

```

FAS02750
FAS02760
FAS02770
FAS02780
FAS02790
FAS02800
FAS02810
FAS02820
FAS02830
FAS02840
FAS02850
FAS02860
FAS02870
FAS02880
FAS02890
FAS02900
FAS02910
FAS02920
FAS02930
FAS02940
FAS02950
FAS02960
FAS02970
FAS02980
FAS02990
FAS03000
FAS03010
FAS03020
FAS03030
FAS03040
FAS03050
FAS03060
FAS03070
FAS03080
FAS03090
FAS03100
FAS03110
FAS03120
FAS03130
FAS03140
FAS03150
FAS03160
FAS03170
FAS03180
FAS03190
FAS03200
FAS03210
FAS03220
FAS03230
FAS03240
FAS03250
FAS03260
FAS03270
FAS03280
FAS03290

```

( ERROR ( AUXILIAR , ERRENCABEZADO ) )

```

END
(*
PROGRAMA = ENCABEZADO DECLARACIONES CUERPO .
*)
FUNCTION PROGRAMA BOOLEAN
BEGIN
PROGRAMA = (ENCABEZADO AND DECLARACIONES AND CUERPO) OR
ERROR (AUXILIAR , ERRPROGRAMA)
END
(* HASTA AQUI TODO ES INDEPENDIENTE DEL MICROPROCESADOR *)
INCLUDE CMS
VAR RC INTEGER
BEGIN
CMS ('CLRSCRN' , RC)
WRITELN ('ENSAMBLANDO ...')
WRITELN ('PASADA 1 = ' , PROGRAMA)
END
/ /

```

FAS03300  
FAS03310  
FAS03320  
FAS03330  
FAS03340  
FAS03350  
FAS03360  
FAS03370  
FAS03380  
FAS03390  
FAS03400  
FAS03410  
FAS03420  
FAS03430  
FAS03440  
FAS03450  
FAS03460  
FAS03470

```

MEMBER FINAL
TYPE LINEA = STRING (80)
FUNCTION PONER ETIQUETA
 (VAR ENTRA , SALE LINEA POS INTEGER) BOOLEAN
VAR
 UN ALPHA ALPHA
 I INDICE IDENTIFICADOR
 UN VALOR
 UNA ETIQUETA CISTINTIVO
 J INTEGER
 ESPOSIBLE BOOLEAN
BEGIN
 J = POS
 TOKEN (POS , ENTRA , UN ALPHA)
 WITH TABLA ETIQUETA DO
 BEGIN
 ESPOSIBLE =
 (DETECTAR IDENTIFICADOR
 (UNA ETIQUETA , UN ARREGLO SIMBOLOS , ELINDICE , I) AND
 LAS ETIQUETAS ASIGNADAS (. I .))
 IF ESPOSIBLE
 THEN
 BEGIN
 TWORDHEX4 (UN ARREGLO VALOR (. I .) , UN VALOR)
 WRITESTR (SALE , SUBSTR (ENTRA , 1 , I) , UN VALOR)
 END
 END
 PONER ETIQUETA = ESPOSIBLE
END
FUNCTION FINAL
 (VAR AUXILIAR , SALIDA TEXT) BOOLEAN ENTRY
VAR
 OTRA LINEA ,
 UNA LINEA LINEA
 POS INTEGER
 AUX , BIEN BOOLEAN
 UN ALPHA ALPHA
BEGIN
 WHILE NOT EOF (AUXILIAR) DO
 BEGIN
 READLN (AUXILIAR , UNA LINEA)
 POS = 1
 REPEAT
 TOKEN (POS , UNA LINEA , UN ALPHA)
 UNTIL (POS LENGTH (UNA LINEA)) OR (UN ALPHA = ' ')
 AUX = ((UN ALPHA = ' ') AND
 PONER ETIQUETA (UNA LINEA , OTRA LINEA , POS)) OR
 (POS LENGTH (UNA LINEA))
 BIEN = BIEN AND AUX
 WRITELN (SALIDA , OTRA LINEA)
 END
 FINAL = BIEN
END
/ /

```

```

FIN00010
FIN00020
FIN00030
FIN00040
FIN00050
FIN00060
FIN00070
FIN00080
FIN00090
FIN00100
FIN00110
FIN00120
FIN00130
FIN00140
FIN00150
FIN00160
FIN00170
FIN00180
FIN00190
FIN00200
FIN00210
FIN00220
FIN00230
FIN00240
FIN00250
FIN00260
FIN00270
FIN00280
FIN00290
FIN00300
FIN00310
FIN00320
FIN00330
FIN00340
FIN00350
FIN00360
FIN00370
FIN00380
FIN00390
FIN00400
FIN00410
FIN00420
FIN00430
FIN00440
FIN00450
FIN00460
FIN00470
FIN00480
FIN00490
FIN00500
FIN00510
FIN00520

```

```

MEMBER MEMONICO
FUNCTION BUSCAR MEMONICO
 (CONST UN MEMONICO DISTINTIVO
 VAR CODIGO DISTINTIVO) BOOLEAN
CONST
 TUPEMEMONICO = 255
TYPE
 TABLAMEMONICO = ARRAY (. 0 .. 255 .) OF DISTINTIVO
STATIC
 LOS MEMONICOS TABLAMEMONICO
VALUE
 LOS MEMONICOS (. 0 .) = 'NOP
 LOS MEMONICOS (. 1 .) = '
 LOS MEMONICOS (. 2 .) = 'RIM
 LOS MEMONICOS (. 3 .) = 'SIM
 LOS MEMONICOS (. 4 .) = 'MOVBB
 LOS MEMONICOS (. 5 .) = 'MOVDB
 LOS MEMONICOS (. 6 .) = 'MOVHB
 LOS MEMONICOS (. 7 .) = 'MOVMB
 LOS MEMONICOS (. 8 .) = 'ADDB
 LOS MEMONICOS (. 9 .) = 'SUBB
 LOS MEMONICOS (. 10 .) = 'ANAB
 LOS MEMONICOS (. 11 .) = 'ORAB
 LOS MEMONICOS (. 12 .) = 'RNZ
 LOS MEMONICOS (. 13 .) = 'RNC
 LOS MEMONICOS (. 14 .) = 'RPO
 LOS MEMONICOS (. 15 .) = 'RP
 LOS MEMONICOS (. 16 .) = 'LXIBN
 LOS MEMONICOS (. 17 .) = 'LXIDN
 LOS MEMONICOS (. 18 .) = 'LXIHN
 LOS MEMONICOS (. 19 .) = 'LXISPN
 LOS MEMONICOS (. 20 .) = 'MOVBC
 LOS MEMONICOS (. 21 .) = 'MOVDC
 LOS MEMONICOS (. 22 .) = 'MOVHC
 LOS MEMONICOS (. 23 .) = 'MOVMC
 LOS MEMONICOS (. 24 .) = 'ADDC
 LOS MEMONICOS (. 25 .) = 'SUBC
 LOS MEMONICOS (. 26 .) = 'ANAC
 LOS MEMONICOS (. 27 .) = 'ORAC
 LOS MEMONICOS (. 28 .) = 'POPB
 LOS MEMONICOS (. 29 .) = 'POPD
 LOS MEMONICOS (. 30 .) = 'POPH
 LOS MEMONICOS (. 31 .) = 'POPPSW
 LOS MEMONICOS (. 32 .) = 'STAXB
 LOS MEMONICOS (. 33 .) = 'STAXD
 LOS MEMONICOS (. 34 .) = 'SHLDADR
 LOS MEMONICOS (. 35 .) = 'STAADR
 LOS MEMONICOS (. 36 .) = 'MOVBD
 LOS MEMONICOS (. 37 .) = 'MOVDD
 LOS MEMONICOS (. 38 .) = 'MOVHD
 LOS MEMONICOS (. 39 .) = 'MOVMD
 LOS MEMONICOS (. 40 .) = 'ADD
 LOS MEMONICOS (. 41 .) = 'SUBD
 LOS MEMONICOS (. 42 .) = 'ANAD
 LOS MEMONICOS (. 43 .) = 'ORAD

```

|                        |            |   |          |
|------------------------|------------|---|----------|
| LOS MEMONICOS (. 44 .) | = 'JNZADK  | ' | MEM00550 |
| LOS MEMONICOS (. 45 .) | = 'JNCADR  | ' | MEM00560 |
| LOS MEMONICOS (. 46 .) | = 'JPDADR  | ' | MEM00570 |
| LOS MEMONICOS (. 47 .) | = 'JPADR   | ' | MEM00580 |
| LOS MEMONICOS (. 48 .) | = 'INXB    | ' | MEM00590 |
| LOS MEMONICOS (. 49 .) | = 'INXD    | ' | MEM00600 |
| LOS MEMONICOS (. 50 .) | = 'INXH    | ' | MEM00610 |
| LOS MEMONICOS (. 51 .) | = 'INXSP   | ' | MEM00620 |
| LOS MEMONICOS (. 52 .) | = 'MOVBE   | ' | MEM00630 |
| LOS MEMONICOS (. 53 .) | = 'MOVDF   | ' | MEM00640 |
| LOS MEMONICOS (. 54 .) | = 'MOVHE   | ' | MEM00650 |
| LOS MEMONICOS (. 55 .) | = 'MOVME   | ' | MEM00660 |
| LOS MEMONICOS (. 56 .) | = 'ADDE    | ' | MEM00670 |
| LOS MEMONICOS (. 57 .) | = 'SUBE    | ' | MEM00680 |
| LOS MEMONICOS (. 58 .) | = 'ANAE    | ' | MEM00690 |
| LOS MEMONICOS (. 59 .) | = 'ORAE    | ' | MEM00700 |
| LOS MEMONICOS (. 60 .) | = 'JMPADR  | ' | MEM00710 |
| LOS MEMONICOS (. 61 .) | = 'OUTADR  | ' | MEM00720 |
| LOS MEMONICOS (. 62 .) | = 'XTHL    | ' | MEM00730 |
| LOS MEMONICOS (. 63 .) | = 'DI      | ' | MEM00740 |
| LOS MEMONICOS (. 64 .) | = 'INRB    | ' | MEM00750 |
| LOS MEMONICOS (. 65 .) | = 'INRD    | ' | MEM00760 |
| LOS MEMONICOS (. 66 .) | = 'INRH    | ' | MEM00770 |
| LOS MEMONICOS (. 67 .) | = 'INRM    | ' | MEM00780 |
| LOS MEMONICOS (. 68 .) | = 'MOVBH   | ' | MEM00790 |
| LOS MEMONICOS (. 69 .) | = 'MOVDB   | ' | MEM00800 |
| LOS MEMONICOS (. 70 .) | = 'MOVHH   | ' | MEM00810 |
| LOS MEMONICOS (. 71 .) | = 'MOVMH   | ' | MEM00820 |
| LOS MEMONICOS (. 72 .) | = 'ADDB    | ' | MEM00830 |
| LOS MEMONICOS (. 73 .) | = 'SUBB    | ' | MEM00840 |
| LOS MEMONICOS (. 74 .) | = 'ANAB    | ' | MEM00850 |
| LOS MEMONICOS (. 75 .) | = 'ORAB    | ' | MEM00860 |
| LOS MEMONICOS (. 76 .) | = 'CNZADR  | ' | MEM00870 |
| LOS MEMONICOS (. 77 .) | = 'CNCADR  | ' | MEM00880 |
| LOS MEMONICOS (. 78 .) | = 'CPDADR  | ' | MEM00890 |
| LOS MEMONICOS (. 79 .) | = 'CPADR   | ' | MEM00900 |
| LOS MEMONICOS (. 80 .) | = 'DCRB    | ' | MEM00910 |
| LOS MEMONICOS (. 81 .) | = 'DCRD    | ' | MEM00920 |
| LOS MEMONICOS (. 82 .) | = 'DCRH    | ' | MEM00930 |
| LOS MEMONICOS (. 83 .) | = 'DCRM    | ' | MEM00940 |
| LOS MEMONICOS (. 84 .) | = 'MOVBI   | ' | MEM00950 |
| LOS MEMONICOS (. 85 .) | = 'MOVBL   | ' | MEM00960 |
| LOS MEMONICOS (. 86 .) | = 'MOVHL   | ' | MEM00970 |
| LOS MEMONICOS (. 87 .) | = 'MOVML   | ' | MEM00980 |
| LOS MEMONICOS (. 88 .) | = 'ADDL    | ' | MEM00990 |
| LOS MEMONICOS (. 89 .) | = 'SUBL    | ' | MEM01000 |
| LOS MEMONICOS (. 90 .) | = 'ANAL    | ' | MEM01010 |
| LOS MEMONICOS (. 91 .) | = 'ORAL    | ' | MEM01020 |
| LOS MEMONICOS (. 92 .) | = 'PUSHB   | ' | MEM01030 |
| LOS MEMONICOS (. 93 .) | = 'PUSHD   | ' | MEM01040 |
| LOS MEMONICOS (. 94 .) | = 'PUSHH   | ' | MEM01050 |
| LOS MEMONICOS (. 95 .) | = 'PUSHPSW | ' | MEM01060 |
| LOS MEMONICOS (. 96 .) | = 'MVIBN   | ' | MEM01070 |
| LOS MEMONICOS (. 97 .) | = 'MVIDN   | ' | MEM01080 |
| LOS MEMONICOS (. 98 .) | = 'MVIHN   | ' | MEM01090 |

|                         |          |          |
|-------------------------|----------|----------|
| LOS MEMONICOS (. 99 .)  | = 'MVIMN | MEM01100 |
| LOS MEMONICOS (. 100 .) | = 'MOVBM | MEM01110 |
| LOS MEMONICOS (. 101 .) | = 'MOVDM | MEM01120 |
| LOS MEMONICOS (. 102 .) | = 'MOVHM | MEM01130 |
| LOS MEMONICOS (. 103 .) | = 'HLT   | MEM01140 |
| LOS MEMONICOS (. 104 .) | = 'ADDM  | MEM01150 |
| LOS MEMONICOS (. 105 .) | = 'SUBM  | MEM01160 |
| LOS MEMONICOS (. 106 .) | = 'ANAM  | MEM01170 |
| LOS MEMONICOS (. 107 .) | = 'ORAM  | MEM01180 |
| LOS MEMONICOS (. 108 .) | = 'ADIN  | MEM01190 |
| LOS MEMONICOS (. 109 .) | = 'SUIN  | MEM01200 |
| LOS MEMONICOS (. 110 .) | = 'ANIN  | MEM01210 |
| LOS MEMONICOS (. 111 .) | = 'ORIN  | MEM01220 |
| LOS MEMONICOS (. 112 .) | = 'RIC   | MEM01230 |
| LOS MEMONICOS (. 113 .) | = 'RAI   | MEM01240 |
| LOS MEMONICOS (. 114 .) | = 'DAA   | MEM01250 |
| LOS MEMONICOS (. 115 .) | = 'STC   | MEM01260 |
| LOS MEMONICOS (. 116 .) | = 'MOVBA | MEM01270 |
| LOS MEMONICOS (. 117 .) | = 'MOVDA | MEM01280 |
| LOS MEMONICOS (. 118 .) | = 'MOVHA | MEM01290 |
| LOS MEMONICOS (. 119 .) | = 'MOVMA | MEM01300 |
| LOS MEMONICOS (. 120 .) | = 'ADDA  | MEM01310 |
| LOS MEMONICOS (. 121 .) | = 'SUBA  | MEM01320 |
| LOS MEMONICOS (. 122 .) | = 'ANAA  | MEM01330 |
| LOS MEMONICOS (. 123 .) | = 'ORAA  | MEM01340 |
| LOS MEMONICOS (. 124 .) | = 'RST0  | MEM01350 |
| LOS MEMONICOS (. 125 .) | = 'RST2  | MEM01360 |
| LOS MEMONICOS (. 126 .) | = 'RST4  | MEM01370 |
| LOS MEMONICOS (. 127 .) | = 'RST6  | MEM01380 |
| LOS MEMONICOS (. 128 .) | = '      | MEM01390 |
| LOS MEMONICOS (. 129 .) | = '      | MEM01400 |
| LOS MEMONICOS (. 130 .) | = '      | MEM01410 |
| LOS MEMONICOS (. 131 .) | = '      | MEM01420 |
| LOS MEMONICOS (. 132 .) | = 'MOVCB | MEM01430 |
| LOS MEMONICOS (. 133 .) | = 'MOVEB | MEM01440 |
| LOS MEMONICOS (. 134 .) | = 'MOVLB | MEM01450 |
| LOS MEMONICOS (. 135 .) | = 'MAAB  | MEM01460 |
| LOS MEMONICOS (. 136 .) | = 'ADCB  | MEM01470 |
| LOS MEMONICOS (. 137 .) | = 'SBBB  | MEM01480 |
| LOS MEMONICOS (. 138 .) | = 'XRAB  | MEM01490 |
| LOS MEMONICOS (. 139 .) | = 'CMPB  | MEM01500 |
| LOS MEMONICOS (. 140 .) | = 'RZ    | MEM01510 |
| LOS MEMONICOS (. 141 .) | = 'RC    | MEM01520 |
| LOS MEMONICOS (. 142 .) | = 'RPE   | MEM01530 |
| LOS MEMONICOS (. 143 .) | = 'RM    | MEM01540 |
| LOS MEMONICOS (. 144 .) | = 'DADB  | MEM01550 |
| LOS MEMONICOS (. 145 .) | = 'DADD  | MEM01560 |
| LOS MEMONICOS (. 146 .) | = 'DADH  | MEM01570 |
| LOS MEMONICOS (. 147 .) | = 'DADSP | MEM01580 |
| LOS MEMONICOS (. 148 .) | = 'MOVCC | MEM01590 |
| LOS MEMONICOS (. 149 .) | = 'MOVFC | MEM01600 |
| LOS MEMONICOS (. 150 .) | = 'MOVLC | MEM01610 |
| LOS MEMONICOS (. 151 .) | = 'MOVAC | MEM01620 |
| LOS MEMONICOS (. 152 .) | = 'ADCC  | MEM01630 |
| LOS MEMONICOS (. 153 .) | = 'SBBB  | MEM01640 |

|                         |            |   |          |
|-------------------------|------------|---|----------|
| LOS MEMONICOS (. 154 .) | = 'XRAC    | ' | MEM01650 |
| LOS MEMONICOS (. 155 .) | = 'CMPC    | ' | MEM01660 |
| LOS MEMONICOS (. 156 .) | = 'RET     | ' | MEM01670 |
| LOS MEMONICOS (. 157 .) | = '        | ' | MEM01680 |
| LOS MEMONICOS (. 158 .) | = '        | ' | MEM01690 |
| LOS MEMONICOS (. 159 .) | = 'SPHL    | ' | MEM01700 |
| LOS MEMONICOS (. 160 .) | = 'LD4XB   | ' | MEM01710 |
| LOS MEMONICOS (. 161 .) | = 'LD4XD   | ' | MEM01720 |
| LOS MEMONICOS (. 162 .) | = 'LHLDADR | ' | MEM01730 |
| LOS MEMONICOS (. 163 .) | = 'LD4ADR  | ' | MEM01740 |
| LOS MEMONICOS (. 164 .) | = 'MOVCD   | ' | MEM01750 |
| LOS MEMONICOS (. 165 .) | = 'MOVED   | ' | MEM01760 |
| LOS MEMONICOS (. 166 .) | = 'MOVL0   | ' | MEM01770 |
| LOS MEMONICOS (. 167 .) | = 'MOVAD   | ' | MEM01780 |
| LOS MEMONICOS (. 168 .) | = 'ADCD    | ' | MEM01790 |
| LOS MEMONICOS (. 169 .) | = 'SBBD    | ' | MEM01800 |
| LOS MEMONICOS (. 170 .) | = 'XRAD    | ' | MEM01810 |
| LOS MEMONICOS (. 171 .) | = 'CMPD    | ' | MEM01820 |
| LOS MEMONICOS (. 172 .) | = 'JZADR   | ' | MEM01830 |
| LOS MEMONICOS (. 173 .) | = 'JCADR   | ' | MEM01840 |
| LOS MEMONICOS (. 174 .) | = 'PCHL    | ' | MEM01850 |
| LOS MEMONICOS (. 175 .) | = 'JMADR   | ' | MEM01860 |
| LOS MEMONICOS (. 176 .) | = 'DCXB    | ' | MEM01870 |
| LOS MEMONICOS (. 177 .) | = 'DCXD    | ' | MEM01880 |
| LOS MEMONICOS (. 178 .) | = 'DCXH    | ' | MEM01890 |
| LOS MEMONICOS (. 179 .) | = 'DCXSP   | ' | MEM01900 |
| LOS MEMONICOS (. 180 .) | = 'MOVCE   | ' | MEM01910 |
| LOS MEMONICOS (. 181 .) | = 'MOVEE   | ' | MEM01920 |
| LOS MEMONICOS (. 182 .) | = 'MOVLE   | ' | MEM01930 |
| LOS MEMONICOS (. 183 .) | = 'MOVAE   | ' | MEM01940 |
| LOS MEMONICOS (. 184 .) | = 'ADCE    | ' | MEM01950 |
| LOS MEMONICOS (. 185 .) | = 'SBBE    | ' | MEM01960 |
| LOS MEMONICOS (. 186 .) | = 'XRAE    | ' | MEM01970 |
| LOS MEMONICOS (. 187 .) | = 'CMPE    | ' | MEM01980 |
| LOS MEMONICOS (. 188 .) | = '        | ' | MEM01990 |
| LOS MEMONICOS (. 189 .) | = 'JNADR   | ' | MEM02000 |
| LOS MEMONICOS (. 190 .) | = 'JPEADR  | ' | MEM02010 |
| LOS MEMONICOS (. 191 .) | = 'PI      | ' | MEM02020 |
| LOS MEMONICOS (. 192 .) | = 'INRC    | ' | MEM02030 |
| LOS MEMONICOS (. 193 .) | = 'INRE    | ' | MEM02040 |
| LOS MEMONICOS (. 194 .) | = 'INRL    | ' | MEM02050 |
| LOS MEMONICOS (. 195 .) | = 'INRA    | ' | MEM02060 |
| LOS MEMONICOS (. 196 .) | = 'MOVCH   | ' | MEM02070 |
| LOS MEMONICOS (. 197 .) | = 'MOVEH   | ' | MEM02080 |
| LOS MEMONICOS (. 198 .) | = 'MOVLH   | ' | MEM02090 |
| LOS MEMONICOS (. 199 .) | = 'MOVAH   | ' | MEM02100 |
| LOS MEMONICOS (. 200 .) | = 'ADCH    | ' | MEM02110 |
| LOS MEMONICOS (. 201 .) | = 'SSBH    | ' | MEM02120 |
| LOS MEMONICOS (. 202 .) | = 'ARAH    | ' | MEM02130 |
| LOS MEMONICOS (. 203 .) | = 'CMPH    | ' | MEM02140 |
| LOS MEMONICOS (. 204 .) | = 'CZADR   | ' | MEM02150 |
| LOS MEMONICOS (. 205 .) | = 'CCADR   | ' | MEM02160 |
| LOS MEMONICOS (. 206 .) | = 'XCHG    | ' | MEM02170 |
| LOS MEMONICOS (. 207 .) | = 'CMADR   | ' | MEM02180 |
| LOS MEMONICOS (. 208 .) | = 'DCRC    | ' | MEM02190 |

|                                    |          |
|------------------------------------|----------|
| LOS MEMONICOS (. 209 .) = 'DCRE    | MEM02200 |
| LOS MEMONICOS (. 210 .) = 'DCRL    | MEM02210 |
| LOS MEMONICOS (. 211 .) = 'DCRA    | MEM02220 |
| LOS MEMONICOS (. 212 .) = 'MOVCL   | MEM02230 |
| LOS MEMONICOS (. 213 .) = 'MOVFL   | MEM02240 |
| LOS MEMONICOS (. 214 .) = 'MOVLL   | MEM02250 |
| LOS MEMONICOS (. 215 .) = 'MOVAL   | MEM02260 |
| LOS MEMONICOS (. 216 .) = 'ADCL    | MEM02270 |
| LOS MEMONICOS (. 217 .) = 'SBBL    | MEM02280 |
| LOS MEMONICOS (. 218 .) = 'XRAL    | MEM02290 |
| LOS MEMONICOS (. 219 .) = 'CML     | MEM02300 |
| LOS MEMONICOS (. 220 .) = 'CALLADR | MEM02310 |
| LOS MEMONICOS (. 221 .) = '        | MEM02320 |
| LOS MEMONICOS (. 222 .) = 'CPEADR  | MEM02330 |
| LOS MEMONICOS (. 223 .) = '        | MEM02340 |
| LOS MEMONICOS (. 224 .) = 'MVICN   | MEM02350 |
| LOS MEMONICOS (. 225 .) = 'MVIEN   | MEM02360 |
| LOS MEMONICOS (. 226 .) = 'MVILN   | MEM02370 |
| LOS MEMONICOS (. 227 .) = 'MVIAN   | MEM02380 |
| LOS MEMONICOS (. 228 .) = 'MOVCM   | MEM02390 |
| LOS MEMONICOS (. 229 .) = 'MOVFM   | MEM02400 |
| LOS MEMONICOS (. 230 .) = 'MOVLM   | MEM02410 |
| LOS MEMONICOS (. 231 .) = 'MOVAM   | MEM02420 |
| LOS MEMONICOS (. 232 .) = 'ADCM    | MEM02430 |
| LOS MEMONICOS (. 233 .) = 'SBBM    | MEM02440 |
| LOS MEMONICOS (. 234 .) = 'XRAM    | MEM02450 |
| LOS MEMONICOS (. 235 .) = 'CMPM    | MEM02460 |
| LOS MEMONICOS (. 236 .) = 'ACIN    | MEM02470 |
| LOS MEMONICOS (. 237 .) = 'SBIN    | MEM02480 |
| LOS MEMONICOS (. 238 .) = 'XRIN    | MEM02490 |
| LOS MEMONICOS (. 239 .) = 'CPIN    | MEM02500 |
| LOS MEMONICOS (. 240 .) = 'RRC     | MEM02510 |
| LOS MEMONICOS (. 241 .) = 'RAR     | MEM02520 |
| LOS MEMONICOS (. 242 .) = 'CMA     | MEM02530 |
| LOS MEMONICOS (. 243 .) = 'CMC     | MEM02540 |
| LOS MEMONICOS (. 244 .) = 'MOVCA   | MEM02550 |
| LOS MEMONICOS (. 245 .) = 'MOVEA   | MEM02560 |
| LOS MEMONICOS (. 246 .) = 'MOVLA   | MEM02570 |
| LOS MEMONICOS (. 247 .) = 'MOVAA   | MEM02580 |
| LOS MEMONICOS (. 248 .) = 'ADCA    | MEM02590 |
| LOS MEMONICOS (. 249 .) = 'SSBA    | MEM02600 |
| LOS MEMONICOS (. 250 .) = 'XRAA    | MEM02610 |
| LOS MEMONICOS (. 251 .) = 'CMPA    | MEM02620 |
| LOS MEMONICOS (. 252 .) = 'RST1    | MEM02630 |
| LOS MEMONICOS (. 253 .) = 'RST3    | MEM02640 |
| LOS MEMONICOS (. 254 .) = 'RST5    | MEM02650 |
| LOS MEMONICOS (. 255 .) = 'RST4    | MEM02660 |

## STATIC

HEXADECIMAL ARRAY (. 0 .. 15 .) OF CHAR  
VALUE

|                           |          |
|---------------------------|----------|
| HEXADECIMAL (. 0 .) = '0' | MEM02700 |
| HEXADECIMAL (. 1 .) = '1' | MEM02710 |
| HEXADECIMAL (. 2 .) = '2' | MEM02720 |
| HEXADECIMAL (. 3 .) = '3' | MEM02730 |
| HEXADECIMAL (. 4 .) = '4' | MEM02740 |

|                                                  |          |
|--------------------------------------------------|----------|
| HEXADECIMAL (. 5 .) = '5'                        | MEM02750 |
| HEXADECIMAL (. 6 .) = '6'                        | MEM02760 |
| HEXADECIMAL (. 7 .) = '7'                        | MEM02770 |
| HEXADECIMAL (. 8 .) = '8'                        | MEM02780 |
| HEXADECIMAL (. 9 .) = '9'                        | MEM02790 |
| HEXADECIMAL (. 10 .) = 'A'                       | MEM02800 |
| HEXADECIMAL (. 11 .) = 'B'                       | MEM02810 |
| HEXADECIMAL (. 12 .) = 'C'                       | MEM02820 |
| HEXADECIMAL (. 13 .) = 'D'                       | MEM02830 |
| HEXADECIMAL (. 14 .) = 'E'                       | MEM02840 |
| HEXADECIMAL (. 15 .) = 'F'                       | MEM02850 |
| VAR                                              | MEM02860 |
| I , J , K INTEGER                                | MEM02870 |
| ENCONTRADO BOOLEAN                               | MEM02880 |
| BEGIN                                            | MEM02890 |
| ENCONTRADO = FALSE K = -1                        | MEM02900 |
| REPEAT                                           | MEM02910 |
| K = K + 1                                        | MEM02920 |
| ENCONTRADO = LOS MEMONICOS (. K .) = UN MEMONICO | MEM02930 |
| UNTIL ( K = 256 ) OR ENCONTRADO                  | MEM02940 |
| ENCONTRADO                                       | MEM02950 |
| THEN                                             | MEM02960 |
| BEGIN                                            | MEM02970 |
| I = K DIV 16                                     | MEM02980 |
| J = K MOD 16                                     | MEM02990 |
| CODIGO (. 1 .) = HEXADECIMAL (. I .)             | MEM03000 |
| CODIGO (. 2 .) = HEXADECIMAL (. J .)             | MEM03010 |
| END                                              | MEM03020 |
| ELSE CODIGO = '***'                              | MEM03030 |
| BUSCAR MEMONICO = ENCONTRADO                     | MEM03040 |
| END                                              | MEM03050 |
| / /                                              |          |

|                                                  |                                  |          |
|--------------------------------------------------|----------------------------------|----------|
| MEMBER OP                                        |                                  | OP 00010 |
| FUNCTION OP                                      | BOOLEAN                          | OP 00020 |
| VAR                                              |                                  | OP 00030 |
| EL MEMONICO , ARGUMENTO , BUSQUEDA               | DISTINTIVO                       | OP 00040 |
| LOS ARGS                                         | ARRAY (. 1 .. 8 .) OF DISTINTIVO | OP 00050 |
| UN ARG                                           | DISTINTIVO                       | OP 00060 |
| LOS RESOLVENTES                                  | ARRAY (. 1 .. 8 .) OF RESOLVENTE | OP 00070 |
| UN RESOLVENTE                                    | RESOLVENTE                       | OP 00080 |
| I                                                | INTEGER                          | OP 00090 |
| FIN                                              | BOOLEAN                          | OP 00100 |
| FUNCTION ESIDENTIFICADOR                         | BOOLEAN                          | OP 00110 |
| VAR AUXESIDENTIFICADOR                           | BOOLEAN                          | OP 00120 |
| BEGIN (* ESIDENTIFICADOR *)                      |                                  | OP 00130 |
| AUXESIDENTIFICADOR =                             |                                  | OP 00140 |
| ESLITERAL AND                                    |                                  | OP 00150 |
| (PROC ETIQUETA ( LA PALABRA , UN RESOLVENTE ) OR |                                  | OP 00160 |
| PROC ESPECIAL ( LA PALABRA , UN RESOLVENTE ) OR  |                                  | OP 00170 |
| PROC WORD ( LA PALABRA , UN RESOLVENTE ) OR      |                                  | OP 00180 |
| PROC BYTE ( LA PALABRA , UN RESOLVENTE ) )       |                                  | OP 00190 |
| IF AUXESIDENTIFICADOR THEN                       |                                  | OP 00200 |
| LOS RESOLVENTES (. I .) = UN RESOLVENTE          |                                  | OP 00210 |
| ESIDENTIFICADOR = AUXESIDENTIFICADOR             |                                  | OP 00220 |
| END (* ESIDENTIFICADOR *)                        |                                  | OP 00230 |
| FUNCTION ESNUMERO                                | BOOLEAN                          | OP 00240 |
| VAR AUXESNUMERO                                  | BOOLEAN                          | OP 00250 |
| BEGIN (* ESNUMERO *)                             |                                  | OP 00260 |
| AUXESNUMERO =                                    |                                  | OP 00270 |
| ( LENGTH ( LA PALABRA ) 1 ) AND                  |                                  | OP 00280 |
| ( LA PALABRA (. 1 .) = '\$' ) AND                |                                  | OP 00290 |
| (                                                |                                  | OP 00300 |
| PROC NUMBYTE ( LA PALABRA , UN RESOLVENTE )      |                                  | OP 00310 |
| OR                                               |                                  | OP 00320 |
| PROC NUMWORD ( LA PALABRA , UN RESOLVENTE )      |                                  | OP 00330 |
| )                                                |                                  | OP 00340 |
| IF AUXESNUMERO THEN                              |                                  | OP 00350 |
| LOS RESOLVENTES (. I .) = UN RESOLVENTE          |                                  | OP 00360 |
| ESNUMERO = AUXESNUMERO                           |                                  | OP 00370 |
| END (* ESNUMERO *)                               |                                  | OP 00380 |
| FUNCTION FILTRAR ARGUMENTOS                      | BOOLEAN                          | OP 00390 |
| BEGIN                                            |                                  | OP 00400 |
| FILTRAR ARGUMENTOS =                             |                                  | OP 00410 |
| ( ESIDENTIFICADOR                                |                                  | OP 00420 |
| OR                                               |                                  | OP 00430 |
| ESNUMERO                                         |                                  | OP 00440 |
| OR                                               |                                  | OP 00450 |
| ( ( LA PALABRA = ' , ' ) AND                     |                                  | OP 00460 |
| NUEVAPALABRA AND                                 |                                  | OP 00470 |
| ( LA PALABRA ' , ' ) AND                         |                                  | OP 00480 |
| FILTRAR ARGUMENTOS )                             |                                  | OP 00490 |
| OR                                               |                                  | OP 00500 |
| ( LA PALABRA = ' ' ) )                           |                                  | OP 00510 |
| FIN = ( LA PALABRA = ' ' )                       |                                  | OP 00520 |
| END                                              |                                  | OP 00530 |
| FUNCTION CODIFICAR MEMONICO                      | BOOLEAN                          | OP 00540 |
| VAR J                                            | INTEGER                          |          |

|                                                            |          |
|------------------------------------------------------------|----------|
| CODIGO. DISTINTIVO                                         | OP 00550 |
| INCLUDE PROGSEG (MEMONICO)                                 | OP 00560 |
| BEGIN                                                      | OP 00570 |
| FOR J = 1 TO I DO                                          | OP 00580 |
| WRITESTR ( ARGUMENTO , ARGUMENTO , LOS ARGS ( . I . ) )    | OP 00590 |
| WRITESTR ( BUSQUEDA , EL MEMONICO , ARGUMENTO )            | OP 00600 |
| CODIFICAR MEMONICO = BUSCAR MEMONICO ( BUSQUEDA , CODIGO ) | OP 00610 |
| WRITESTR ( TRADUCCION , CODIGO , ' ' )                     | OP 00620 |
| END                                                        | OP 00630 |
| FUNCTION CODIFICAR ARGUMENTO BOOLEAN                       | OP 00640 |
| VAR J INTEGER                                              | OP 00650 |
| LONG INSTRUCCION TWORD                                     | OP 00660 |
| BEGIN                                                      | OP 00670 |
| LONG INSTRUCCION = 1                                       | OP 00680 |
| FOR J = 1 TO I DO                                          | OP 00690 |
| WITH LOS RESOLVENTES ( . J . ) DO                          | OP 00700 |
| BEGIN                                                      | OP 00710 |
| WRITESTR( TRADUCCION , TRADUCCION , VALOR )                | OP 00720 |
| LONG INSTRUCCION = LONG INSTRUCCION + LONGITUD             | OP 00730 |
| END                                                        | OP 00740 |
| PUNTERO = PUNTERO + LONG INSTRUCCION                       | OP 00750 |
| CODIFICAR ARGUMENTO = PUNTERO ( 64 * 1024 )                | OP 00760 |
| END                                                        | OP 00770 |
| VAR AUX BOOLEAN                                            | OP 00780 |
| BEGIN (* OP *)                                             | OP 00790 |
| I = 0                                                      | OP 00800 |
| WRITESTR ( EL MEMONICO , LA PALABRA )                      | OP 00810 |
| REPEAT                                                     | OP 00820 |
| AUX = NUEVAPALABRA AND FILTRAR ARGUMENTOS                  | OP 00830 |
| IF AUX AND NOT FIN                                         | OP 00840 |
| THEN                                                       | OP 00850 |
| BEGIN                                                      | OP 00860 |
| I = I + 1                                                  | OP 00870 |
| LOS ARGS ( . I . ) = UN ARG                                | OP 00880 |
| END                                                        | OP 00890 |
| UNTIL FIN OR ( I = 8 )                                     | OP 00900 |
| OP = AUX AND FIN AND ( I = 8 ) AND                         | OP 00910 |
| CODIFICAR MEMONICO AND CODIFICAR ARGUMENTO                 | OP 00920 |
| END (* OP *)                                               | OP 00930 |

MEMBER PARAMOP

FUNCTION ESIDENTIFICADOR BOOLEAN

BEGIN (\* ESIDENTIFICADOR \*)

ESIDENTIFICADOR =

ESLITERAL AND

(PROC ETIQUETA ( LA PALABRA , UN ARG , LOS RESOLVENTES (. I .) ) OR

PROC ESPECIAL ( LA PALABRA , UN ARG , LOS RESOLVENTES (. I .) ) OR

PROC WORD ( LA PALABRA , UN ARG , LOS RESOLVENTES (. I .) ) OR

PROC BYTE ( LA PALABRA , UN ARG , LOS RESOLVENTES (. I .) ) OR

PROC BIN ( LA PALABRA , UN ARG , LOS RESOLVENTES (. I .) ) OR

PROC OCT ( LA PALABRA , UN ARG , LOS RESOLVENTES (. I .) ) OR

PROC DEC ( LA PALABRA , UN ARG , LOS RESOLVENTES (. I .) ) OR

PROC ALFABETICO ( LA PALABRA , UN ARG , LOS RESOLVENTES (. I .) )

END (\* ESIDENTIFICADOR \*)

FUNCTION ESNUMERO BOOLEAN

BEGIN (\* ESNUMERO \*)

ESNUMERO =

( LENGTH ( LA PALABRA ) 1 ) AND

( LA PALABRA (. 1 .) = '\$' ) AND

(

( ESTBYTE ( LA PALABRA ) AND

PROC NUMBYTE ( LA PALABRA , UN ARG , LOS RESOLVENTES (. I .) )

OR

( ESTWORD ( LA PALABRA ) AND

PROC NUMWORD ( LA PALABRA , UN ARG , LOS RESOLVENTES (. I .) )

)

END (\* ESNUMERO \*)

/ /

PAR00010

PAR00020

PAR00030

PAR00040

PAR00050

PAR00060

PAR00070

PAR00080

PAR00090

PAR00100

PAR00110

PAR00120

PAR00130

PAR00140

PAR00150

PAR00160

PAR00170

PAR00180

PAR00190

PAR00200

PAR00210

PAR00220

PAR00230

PAR00240

PAR00250

PAR00260

MEMBER PROCTIPO

|                                                       |          |
|-------------------------------------------------------|----------|
| TYPE                                                  | PRG00010 |
| INCLUDE ESTRUCTURA (RESOLVE)                          | PRG00020 |
| FUNCTION PROC ETIQUETA                                | PRG00030 |
| ( VAR LA PALABRA DISTINTIVO                           | PRG00040 |
| VAR UN RESOLVENTE RESOLVENTE )                        | PRG00050 |
| BOOLEAN ENTRY                                         | PRG00060 |
| VAR I INDICEIDENTIFICADOR                             | PRG00070 |
| BEGIN                                                 | PRG00080 |
| WITH UN RESOLVENTE , TABLA ETIQUETA DO                | PRG00090 |
| BEGIN                                                 | PRG00100 |
| IF DETECTAR IDENTIFICADOR                             | PRG00110 |
| ( LA PALABRA ,                                        | PRG00120 |
| UN ARREGLO SIMBOLO ,                                  | PRG00130 |
| ELINDICE , I )                                        | PRG00140 |
| THEN BEGIN                                            | PRG00150 |
| CLASE = CETIQUETA                                     | PRG00160 |
| WRITESTR ( UN ARG , 'ADR' )                           | PRG00170 |
| ASIGNADO = LAS ETIQUETAS ASIGNADAS ( . I . )          | PRG00180 |
| IF ASIGNADO                                           | PRG00190 |
| THEN TWORDHEXA ( UN ARREGLO VALOR ( . I . ) , VALOR ) | PRG00200 |
| ELSE WRITESTR ( VALOR , ' ' , I )                     | PRG00210 |
| PROC ETIQUETA = TRUE                                  | PRG00220 |
| END                                                   | PRG00230 |
| ELSE BEGIN                                            | PRG00240 |
| CLASE = CERROR                                        | PRG00250 |
| WRITESTR ( UN ARG , '**' )                            | PRG00260 |
| WRITESTR ( VALOR , '**' )                             | PRG00270 |
| PROC ETIQUETA = FALSE                                 | PRG00280 |
| END                                                   | PRG00290 |
| END                                                   | PRG00300 |
| FUNCTION PROC ESPECIAL                                | PRG00310 |
| ( VAR LA PALABRA DISTINTIVO                           | PRG00320 |
| VAR UN RESOLVENTE RESOLVENTE )                        | PRG00330 |
| BOOLEAN ENTRY                                         | PRG00340 |
| STATIC                                                | PRG00350 |
| ESPECIALES ARREGLO SIMBOLOS                           | PRG00360 |
| ELINDICE INDICEIDENTIFICADOR                          | PRG00370 |
| VALUE                                                 | PRG00380 |
| ESPECIALES ( . 0 . ) = ''                             | PRG00390 |
| ESPECIALES ( . 1 . ) = 'ACC'                          | PRG00400 |
| ESPECIALES ( . 2 . ) = 'B'                            | PRG00410 |
| ESPECIALES ( . 3 . ) = 'C'                            | PRG00420 |
| ESPECIALES ( . 4 . ) = 'D'                            | PRG00430 |
| ESPECIALES ( . 5 . ) = 'E'                            | PRG00440 |
| ESPECIALES ( . 6 . ) = 'H'                            | PRG00450 |
| ESPECIALES ( . 7 . ) = 'L'                            | PRG00460 |
| ESPECIALES ( . 8 . ) = 'M'                            | PRG00470 |
| ESPECIALES ( . 9 . ) = 'PSW'                          | PRG00480 |
| ELINDICE = 9                                          | PRG00490 |
| VAR I INDICEIDENTIFICADOR                             | PRG00500 |
| BEGIN                                                 | PRG00510 |
| WITH UN RESOLVENTE DO                                 | PRG00520 |
| IF DETECTAR IDENTIFICADOR                             | PRG00530 |
| ( LA PALABRA , ESPECIALES , ELINDICE , I )            | PRG00540 |
| THEN                                                  |          |

|                                                     |          |
|-----------------------------------------------------|----------|
| BEGIN                                               | PR000550 |
| CLASE = CESPECIALES                                 | PR000560 |
| WRITESTR ( UN ARG , LA PALABRA )                    | PR000570 |
| LONGITUD = 0                                        | PR000580 |
| PROC ESPECIAL = TRUE                                | PR000590 |
| END                                                 | PR000600 |
| ELSE                                                | PR000610 |
| BEGIN                                               | PR000620 |
| CLASE = CERROR                                      | PR000630 |
| WRITESTR ( UN ARG , '***' )                         | PR000640 |
| WRITESTR ( VALOR , '***' )                          | PR000650 |
| PROC ESPECIAL = FALSE                               | PR000660 |
| END                                                 | PR000670 |
| END                                                 | PR000680 |
| FUNCTION PROC WORD                                  | PR000690 |
| ( VAR LA PALABRA DISTINTIVO                         | PR000700 |
| VAR UN RESOLVENTE RESOLVENTE )                      | PR000710 |
| BOOLEAN ENTRY                                       | PR000720 |
| VAR I INDICEIDENTIFICADOR                           | PR000730 |
| BEGIN                                               | PR000740 |
| WITH UN RESOLVENTE , TABLA WORD DO                  | PR000750 |
| IF DETECTAR IDENTIFICADOR                           | PR000760 |
| ( LA PALABRA , UN ARREGLO SIMBOLOS , ELINDICE , I ) | PR000770 |
| THEN                                                | PR000780 |
| BEGIN                                               | PR000790 |
| CLASE = CTWORD                                      | PR000800 |
| WRITESTR ( UN ARG , 'N' )                           | PR000810 |
| TWORDHEXA ( UN ARREGLO VALOR ( . I . ) , VALOR )    | PR000820 |
| LONGITUD = 2                                        | PR000830 |
| PROC WORD = TRUE                                    | PR000840 |
| END                                                 | PR000850 |
| ELSE                                                | PR000860 |
| BEGIN                                               | PR000870 |
| CLASE = CERROR                                      | PR000880 |
| WRITESTR ( UN ARG , '***' )                         | PR000890 |
| WRITESTR ( VALOR , '***' )                          | PR000900 |
| PROC WORD = FALSE                                   | PR000910 |
| END                                                 | PR000920 |
| END                                                 | PR000930 |
| FUNCTION PROC BYTE                                  | PR000940 |
| ( VAR LA PALABRA DISTINTIVO                         | PR000950 |
| VAR UN RESOLVENTE RESOLVENTE )                      | PR000960 |
| BOOLEAN ENTRY                                       | PR000970 |
| VAR I INDICEIDENTIFICADOR                           | PR000980 |
| BEGIN                                               | PR000990 |
| WITH UN RESOLVENTE , TABLA BYTE DO                  | PR001000 |
| IF DETECTAR IDENTIFICADOR                           | PR001010 |
| ( LA PALABRA , UN ARREGLO SIMBOLOS , ELINDICE , I ) | PR001020 |
| THEN                                                | PR001030 |
| BEGIN                                               | PR001040 |
| CLASE = CTWORD                                      | PR001050 |
| WRITESTR ( UN ARG , 'N' )                           | PR001060 |
| TBYTEHEXA ( UN ARREGLO VALOR ( . I . ) , VALOR )    | PR001070 |
| LONGITUD = 1                                        | PR001080 |
| PROC BYTE = TRUE                                    | PR001090 |
| END                                                 |          |
| ELSE                                                |          |

|                                 |          |
|---------------------------------|----------|
| BEGIN                           | PR001100 |
| CLASE = CERROR                  | PR001110 |
| WRITESTR ( UN ARG , '***' )     | PR001120 |
| WRITESTR ( VALOR , '***' )      | PR001130 |
| PROC BYTE = FALSE               | PR001140 |
| END                             | PR001150 |
| END                             | PR001160 |
| FUNCTION PROC NUMBYTE           | PR001170 |
| ( VAR LA PALABRA DISTINTIVO     | PR001180 |
| VAR UN RESOLVENTE RESOLVENTE )  | PR001190 |
| BOOLEAN ENTRY                   | PR001200 |
| VAR AUX BOOLEAN                 | PR001210 |
| BEGIN                           | PR001220 |
| AUX = ESTBYTE ( LA PALABRA )    | PR001230 |
| WITH UN RESOLVENTE DO           | PR001240 |
| IF AUX                          | PR001250 |
| THEN                            | PR001260 |
| BEGIN                           | PR001270 |
| CLASE = CNUMBYTE                | PR001280 |
| WRITESTR ( UN ARG , 'N' )       | PR001290 |
| WRITESTR ( VALOR , LA PALABRA ) | PR001300 |
| LONGITUD = 1                    | PR001310 |
| END                             | PR001320 |
| ELSE                            | PR001330 |
| BEGIN                           | PR001340 |
| CLASE = CERROR                  | PR001350 |
| WRITESTR ( UN ARG , '***' )     | PR001360 |
| WRITESTR ( VALOR , '***' )      | PR001370 |
| PROC NUMBYTE = FALSE            | PR001380 |
| END                             | PR001390 |
| END                             | PR001400 |
| FUNCTION PROC NUMWORD           | PR001410 |
| ( VAR LA PALABRA DISTINTIVO     | PR001420 |
| VAR UN RESOLVENTE RESOLVENTE )  | PR001430 |
| BOOLEAN ENTRY                   | PR001440 |
| VAR AUX BOOLEAN                 | PR001450 |
| BEGIN                           | PR001460 |
| AUX = ESTWORD ( LA PALABRA )    | PR001470 |
| WITH UN RESOLVENTE DO           | PR001480 |
| IF AUX                          | PR001490 |
| THEN                            | PR001500 |
| BEGIN                           | PR001510 |
| CLASE = CNUMWORD                | PR001520 |
| WRITESTR ( UN ARG , 'N' )       | PR001530 |
| WRITESTR ( VALOR , LA PALABRA ) | PR001540 |
| LONGITUD = 1                    | PR001550 |
| END                             | PR001560 |
| ELSE                            | PR001570 |
| BEGIN                           | PR001580 |
| CLASE = CERROR                  | PR001590 |
| WRITESTR ( UN ARG , '***' )     | PR001600 |
| WRITESTR ( VALOR , '***' )      | PR001610 |
| PROC NUMWORD = FALSE            | PR001620 |
| END                             |          |
| END                             |          |
| / /                             |          |

```

MEMBER TABLA
SEGMENT TABLA
 INCLUDE ESTRUCTURA (DISTINTIVO)
 INCLUDE ESTRUCTURA (ESTTIPOS)
 INCLUDE ESTRUCTURA (ESTTABLA)
 INCLUDE EXTERNOS (TIPOS)
TYPE
 INCLUDE ESTRUCTURA (TABLA)
STATIC
 TABLA DEC ,
 TABLA OCT ,
 TABLA BIN ,
 TABLA ALFABETICO TABLADISTINTIVO
 TABLA BYTE TABLATBYTE
 TABLA ETIQUETA ,
 TABLA WORD TABLATWORD
VALUE
 TABLA DEC.ELINDICE = 0
 TABLA OCT.ELINDICE = 0
 TABLA BIN.ELINDICE = 0
 TABLA ALFABETICO.ELINDICE = 0
 TABLA BYTE.ELINDICE = 0
 TABLA ETIQUETA.ELINDICE = 0
 TABLA WORD.ELINDICE = 0
TYPE
 ETIQUETAS ASIGNADAS = ARRAY (. INDICEIDENTIFICADOR .) OF BOOLEAN
STATIC
 LAS ETIQUETAS ASIGNADAS ETIQUETAS ASIGNADAS
FUNCTION DETECTAR IDENTIFICADOR
 (CONST LA PALABRA DISTINTIVO
 VAR UN ARREGLO SIMBOLOS ARREGLO SIMBOLOS
 CONST EL TOPE SIMBOLOS INDICEIDENTIFICADOR
 VAR I INDICEIDENTIFICADOR)
 BOOLEAN ENTRY
BEGIN
 UN ARREGLO SIMBOLOS (. AUSENTE .) = LA PALABRA
 I = EL TOPE SIMBOLOS
 WHILE (UN ARREGLO SIMBOLOS (. I .) LA PALABRA) DO I = I - 1
 DETECTAR IDENTIFICADOR = (I AUSENTE)
END
FUNCTION INSERTAR IDENTIFICADOR
 VAR EL ARREGLO SIMBOLOS ARREGLO SIMBOLOS
 VAR EL INDICE SIMBOLOS INDICEIDENTIFICADOR
 CONST LA PALABRA DISTINTIVO)
 BOOLEAN ENTRY
VAR
 ES POSIBLE INSERTAR BOOLEAN
BEGIN
 ES POSIBLE INSERTAR = (EL INDICE SIMBOLOS TOPEIDENTIFICADOR)
 IF ES POSIBLE INSERTAR
 THEN
 BEGIN
 EL INDICE SIMBOLOS = EL INDICE SIMBOLOS + 1
 EL ARREGLO SIMBOLOS (. EL INDICE SIMBOLOS .) = LA PALABRA
 END

```

TAB00010  
 TAB00020  
 TAB00030  
 TAB00040  
 TAB00050  
 TAB00060  
 TAB00070  
 TAB00080  
 TAB00090  
 TAB00100  
 TAB00110  
 TAB00120  
 TAB00130  
 TAB00140  
 TAB00150  
 TAB00160  
 TAB00170  
 TAB00180  
 TAB00190  
 TAB00200  
 TAB00210  
 TAB00220  
 TAB00230  
 TAB00240  
 TAB00250  
 TAB00260  
 TAB00270  
 TAB00280  
 TAB00290  
 TAB00300  
 TAB00310  
 TAB00320  
 TAB00330  
 TAB00340  
 TAB00350  
 TAB00360  
 TAB00370  
 TAB00380  
 TAB00390  
 TAB00400  
 TAB00410  
 TAB00420  
 TAB00430  
 TAB00440  
 TAB00450  
 TAB00460  
 TAB00470  
 TAB00480  
 TAB00490  
 TAB00500  
 TAB00510  
 TAB00520  
 TAB00530  
 TAB00540

```

INSERTAR IDENTIFICADOR = ES POSIBLE INSERTAR
END
FUNCTION ASIGNAR ETIQUETA
(ESTEVALOR TWORD
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR) BOOLEAN ENTRY
VAR
 ASIGNAR BOOLEAN
BEGIN
 WITH TABLA ETIQUETA DO
 BEGIN
 ASIGNAR = (UNINDICEIDENTIFICADOR AUSENTE) AND
 (UNINDICEIDENTIFICADOR = ELINDICE)
 UN ARREGLO VALOR (. UNINDICEIDENTIFICADOR .) = ESTEVALOR
 LAS ETIQUETAS ASIGNADAS (. UNINDICEIDENTIFICADOR .) = TRUE
 END
END
FUNCTION ASIGNAR DEC
(ESTEVALOR DISTINTIVO
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR) BOOLEAN ENTRY
VAR
 ASIGNAR BOOLEAN
BEGIN
 WITH TABLA DEC DO
 BEGIN
 ASIGNAR = (UNINDICEIDENTIFICADOR AUSENTE) AND
 (UNINDICEIDENTIFICADOR = ELINDICE)
 UN ARREGLO VALOR (. UNINDICEIDENTIFICADOR .) = ESTEVALOR
 END
END
FUNCTION ASIGNAR TBYTE
(ESTEVALOR TBYTE
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR) BOOLEAN ENTRY
VAR
 ASIGNAR BOOLEAN
BEGIN
 WITH TABLA BYTE DO
 BEGIN
 ASIGNAR = (UNINDICEIDENTIFICADOR AUSENTE) AND
 (UNINDICEIDENTIFICADOR = ELINDICE)
 UN ARREGLO VALOR (. UNINDICEIDENTIFICADOR .) = ESTEVALOR
 END
END
FUNCTION ASIGNAR TWORD
(ESTEVALOR TWORD
 UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR) BOOLEAN ENTRY
VAR
 ASIGNAR BOOLEAN
BEGIN
 WITH TABLA WORD DO
 BEGIN
 ASIGNAR = (UNINDICEIDENTIFICADOR AUSENTE) AND
 (UNINDICEIDENTIFICADOR = ELINDICE)
 UN ARREGLO VALOR (. UNINDICEIDENTIFICADOR .) = ESTEVALOR
 END
END

```

TAB00550  
 TAB00560  
 TAB00570  
 TAB00580  
 TAB00590  
 TAB00600  
 TAB00610  
 TAB00620  
 TAB00630  
 TAB00640  
 TAB00650  
 TAB00660  
 TAB00670  
 TAB00680  
 TAB00690  
 TAB00700  
 TAB00710  
 TAB00720  
 TAB00730  
 TAB00740  
 TAB00750  
 TAB00760  
 TAB00770  
 TAB00780  
 TAB00790  
 TAB00800  
 TAB00810  
 TAB00820  
 TAB00830  
 TAB00840  
 TAB00850  
 TAB00860  
 TAB00870  
 TAB00880  
 TAB00890  
 TAB00900  
 TAB00910  
 TAB00920  
 TAB00930  
 TAB00940  
 TAB00950  
 TAB00960  
 TAB00970  
 TAB00980  
 TAB00990  
 TAB01000  
 TAB01010  
 TAB01020  
 TAB01030  
 TAB01040  
 TAB01050  
 TAB01060  
 TAB01070  
 TAB01080  
 TAB01090

|                                                               |          |
|---------------------------------------------------------------|----------|
| FUNCTION ASIGNAR CCT                                          | TAB01100 |
| ( ESTEVALOR DISTINTIVO                                        | TAB01110 |
| UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR ) BOOLEAN ENTRY     | TAB01120 |
| VAR                                                           | TAB01130 |
| ASIGNAR BOOLEAN                                               | TAB01140 |
| BEGIN                                                         | TAB01150 |
| WITH TABLA OCT DO                                             | TAB01160 |
| BEGIN                                                         | TAB01170 |
| ASIGNAR = ( UNINDICEIDENTIFICADOR AUSENTE ) AND               | TAB01180 |
| ( UNINDICEIDENTIFICADOR = ELINDICE )                          | TAB01190 |
| UN ARREGLO VALOR ( . UNINDICEIDENTIFICADOR . ) = ESTEVALOR    | TAB01200 |
| END                                                           | TAB01210 |
| EAD                                                           | TAB01220 |
| FUNCTION ASIGNAR BIN                                          | TAB01230 |
| ( ESTEVALOR DISTINTIVO                                        | TAB01240 |
| UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR ) BOOLEAN ENTRY     | TAB01250 |
| VAR                                                           | TAB01260 |
| ASIGNAR BOOLEAN                                               | TAB01270 |
| BEGIN                                                         | TAB01280 |
| WITH TABLA BIN DO                                             | TAB01290 |
| BEGIN                                                         | TAB01300 |
| ASIGNAR = ( UNINDICEIDENTIFICADOR AUSENTE ) AND               | TAB01310 |
| ( UNINDICEIDENTIFICADOR = ELINDICE )                          | TAB01320 |
| UN ARREGLO VALOR ( . UNINDICEIDENTIFICADOR . ) = ESTEVALOR    | TAB01330 |
| END                                                           | TAB01340 |
| END                                                           | TAB01350 |
| FUNCTION ASIGNAR ALFABETICO                                   | TAB01360 |
| ( CONST ESTEVALOR DISTINTIVO                                  | TAB01370 |
| UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR ) BOOLEAN ENTRY     | TAB01380 |
| VAR                                                           | TAB01390 |
| ASIGNAR BOOLEAN                                               | TAB01400 |
| BEGIN                                                         | TAB01410 |
| WITH TABLA ALFABETICO DO                                      | TAB01420 |
| BEGIN                                                         | TAB01430 |
| ASIGNAR = ( UNINDICEIDENTIFICADOR AUSENTE ) AND               | TAB01440 |
| ( UNINDICEIDENTIFICADOR = ELINDICE )                          | TAB01450 |
| UN ARREGLO VALOR ( . UNINDICEIDENTIFICADOR . ) = ESTEVALOR    | TAB01460 |
| END                                                           | TAB01470 |
| END                                                           | TAB01480 |
| FUNCTION INS ETIQUETA                                         | TAB01490 |
| ( CONST ESTEIDENTIFICADOR DISTINTIVO                          | TAB01500 |
| VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR ) BOOLEAN       | TAB01510 |
| BEGIN                                                         | TAB01520 |
| WITH TABLA ETIQUETA DO                                        | TAB01530 |
| BEGIN                                                         | TAB01540 |
| INS ETIQUETA =                                                | TAB01550 |
| INSERTAR IDENTIFICADOR                                        | TAB01560 |
| ( UN ARREGLO SIMBOLOS , ELINDICE , ESTEIDENTIFICADOR )        | TAB01570 |
| UNINDICEIDENTIFICADOR = ELINDICE                              | TAB01580 |
| LAS ETIQUETAS ASIGNADAS ( . UNINDICEIDENTIFICADOR . ) = FALSE | TAB01590 |
| END                                                           | TAB01600 |
| END                                                           | TAB01610 |
| FUNCTION INS DEC                                              | TAB01620 |
| ( CONST ESTEIDENTIFICADOR DISTINTIVO                          | TAB01630 |
| VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR ) BOOLEAN       | TAB01640 |

```

BEGIN
 WITH TABLA DEC DO
 BEGIN
 INS DEC =
 INSERTAR IDENTIFICADOR
 (UN ARREGLO SIMBOLOS , ELINDICE , ESTEIDENTIFICADOR)
 UNINDICEIDENTIFICADOR = ELINDICE
 END
END
FUNCTION INS TBYTE
 (CONST ESTEIDENTIFICADOR DISTINTIVO
 VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR) BOOLEAN
BEGIN
 WITH TABLA BYTE DO
 BEGIN
 INS TBYTE =
 INSERTAR IDENTIFICADOR
 (UN ARREGLO SIMBOLOS , ELINDICE , ESTEIDENTIFICADOR)
 UNINDICEIDENTIFICADOR = ELINDICE
 END
END
FUNCTION INS TWORD
 (CONST ESTEIDENTIFICADOR DISTINTIVO
 VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR) BOOLEAN
BEGIN
 WITH TABLA WORD DO
 BEGIN
 INS TWORD =
 INSERTAR IDENTIFICADOR
 (UN ARREGLO SIMBOLOS , ELINDICE , ESTEIDENTIFICADOR)
 UNINDICEIDENTIFICADOR = ELINDICE
 END
END
FUNCTION INS OCT
 (CONST ESTEIDENTIFICADOR DISTINTIVO
 VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR) BOOLEAN
BEGIN
 WITH TABLA OCT DO
 BEGIN
 INS OCT =
 INSERTAR IDENTIFICADOR
 (UN ARREGLO SIMBOLOS , ELINDICE , ESTEIDENTIFICADOR)
 UNINDICEIDENTIFICADOR = ELINDICE
 END
END
FUNCTION INS BIN
 (CONST ESTEIDENTIFICADOR DISTINTIVO
 VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR) BOOLEAN
BEGIN
 WITH TABLA BIN DO
 BEGIN
 INS BIN =
 INSERTAR IDENTIFICADOR
 (UN ARREGLO SIMBOLOS , ELINDICE , ESTEIDENTIFICADOR)
 UNINDICEIDENTIFICADOR = ELINDICE
 END
END

```

```

TAB01650
TAB01660
TAB01670
TAB01680
TAB01690
TAB01700
TAB01710
TAB01720
TAB01730
TAB01740
TAB01750
TAB01760
TAB01770
TAB01780
TAB01790
TAB01800
TAB01810
TAB01820
TAB01830
TAB01840
TAB01850
TAB01860
TAB01870
TAB01880
TAB01890
TAB01900
TAB01910
TAB01920
TAB01930
TAB01940
TAB01950
TAB01960
TAB01970
TAB01980
TAB01990
TAB02000
TAB02010
TAB02020
TAB02030
TAB02040
TAB02050
TAB02060
TAB02070
TAB02080
TAB02090
TAB02100
TAB02110
TAB02120
TAB02130
TAB02140
TAB02150
TAB02160
TAB02170
TAB02180
TAB02190

```

```

END
END
FUNCTION INS ALFABETICO
 (CONST ESTEIDENTIFICADOR DISTINTIVO
 VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR) BOOLEAN
BEGIN
 WITH TABLA ALFABETICO DO
 BEGIN
 INS ALFABETICO =
 INSERTAR IDENTIFICADOR
 (UN ARREGLO SIMBOLOS , ELINDICE , ESTEIDENTIFICADOR)
 UNINDICEIDENTIFICADOR = ELINDICE
 END
END
FUNCTION INS IDENTIFICADOR
 (CONST UNIDENTIFICADOR DISTINTIVO
 UNACLASEDECLARACION CLASEDECLARACION
 VAR UNINDICEIDENTIFICADOR INDICEIDENTIFICADOR)
 BOOLEAN ENTRY
BEGIN
 CASE UNACLASEDECLARACION OF
 CETIQUETA INS IDENTIFICADOR =
 INS ETIQUETA (UNIDENTIFICADOR ,
 UNINDICEIDENTIFICADOR)
 CDEC INS IDENTIFICADOR =
 INS DEC (UNIDENTIFICADOR ,
 UNINDICEIDENTIFICADOR)
 COCT INS IDENTIFICADOR =
 INS OCT (UNIDENTIFICADOR ,
 UNINDICEIDENTIFICADOR)
 CBIN INS IDENTIFICADOR =
 INS BIN (UNIDENTIFICADOR ,
 UNINDICEIDENTIFICADOR)
 CALFABETICO INS IDENTIFICADOR =
 INS ALFABETICO(UNIDENTIFICADOR ,
 UNINDICEIDENTIFICADOR)
 CTBYTE INS IDENTIFICADOR =
 INS TBYTE (UNIDENTIFICADOR ,
 UNINDICEIDENTIFICADOR)
 CTWORD INS IDENTIFICADOR =
 INS TWORD (UNIDENTIFICADOR ,
 UNINDICEIDENTIFICADOR)
 OTHERWISE
 END
END
FUNCTION PROCESAR ETIQUETA
 (CONST ETIQUETA DISTINTIVO
 VAR VALOR TWORD) BOOLEAN ENTRY
VAR I INDICEIDENTIFICADOR
BEGIN
 WITH TABLA ETIQUETA DO
 PROCESAR ETIQUETA =
 (DETECTAR IDENTIFICADOR
 (ETIQUETA , UN ARREGLO SIMBOLOS , ELINDICE , I) OR
 INS IDENTIFICADOR (ETIQUETA , CETIQUETA , I)) AND

```

TAB02200  
 TAB02210  
 TAB02220  
 TAB02230  
 TAB02240  
 TAB02250  
 TAB02260  
 TAB02270  
 TAB02280  
 TAB02290  
 TAB02300  
 TAB02310  
 TAB02320  
 TAB02330  
 TAB02340  
 TAB02350  
 TAB02360  
 TAB02370  
 TAB02380  
 TAB02390  
 TAB02400  
 TAB02410  
 TAB02420  
 TAB02430  
 TAB02440  
 TAB02450  
 TAB02460  
 TAB02470  
 TAB02480  
 TAB02490  
 TAB02500  
 TAB02510  
 TAB02520  
 TAB02530  
 TAB02540  
 TAB02550  
 TAB02560  
 TAB02570  
 TAB02580  
 TAB02590  
 TAB02600  
 TAB02610  
 TAB02620  
 TAB02630  
 TAB02640  
 TAB02650  
 TAB02660  
 TAB02670  
 TAB02680  
 TAB02690  
 TAB02700  
 TAB02710  
 TAB02720  
 TAB02730  
 TAB02740

FILE PROGSEG MACLIB A1 VM/SP RELEASE 5 \*\*\* U.N.P.S.J.B \*\*\* PUT 8801+

ASIGNAR ETIQUETA ( 1 , VALOR )

END

INCLUDE PROGSEG (PROCTIPO)

INCLUDE PROGSEG (FINAL)

/ /

TAB02750

TAB02760

TAB02770

TAB02780

MEMBER TIPOS

SEGMENT TIPOS

INCLUDE ESTRUCTURA (DISTINTIVO)

INCLUDE ESTRUCTURA ( ESTTIPOS)

STATIC

DIGITOSHEXADECIMALES ,

DIGITOSDECIMALES ,

DIGITOSOCTALES ,

DIGITOSBINARIOS SET OF CHAR

VALUE

DIGITOSHEXADECIMALES =

( '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' , '8' , '9' ,  
 'A' , 'B' , 'C' , 'D' , 'E' , 'F'  
 . )

DIGITOSDECIMALES =

( '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' , '8' , '9' . )

DIGITOSOCTALES =

( '0' , '1' , '2' , '3' , '4' , '5' , '6' , '7' . )

DIGITOSBINARIOS =

( '0' , '1' . )

FUNCTION ESTBYTE ( UNDISTINTIVO DISTINTIVO ) BOOLEAN ENTRY

VAR

I , LON INTEGER

OK BOOLEAN

BEGIN

LON = LENGTH ( UNDISTINTIVO )

OK = ( LON = LONGTBYTE )

FOR I = 1 TO LON DO

OK = OK AND ( UNDISTINTIVO (.I.) IN DIGITOSHEXADECIMALES )

ESTBYTE = OK

END

FUNCTION ESTWORD ( UNDISTINTIVO DISTINTIVO ) BOOLEAN ENTRY

VAR

I , LON INTEGER

OK BOOLEAN

BEGIN

LON = LENGTH ( UNDISTINTIVO )

OK = ( LON = LONGTWORD )

FOR I = 1 TO LON DO

OK = OK AND ( UNDISTINTIVO (.I.) IN DIGITOSHEXADECIMALES )

ESTWORD = OK

END

PROCEDURE HEXATWORD

( UNHEXA DISTINTIVO VAR UNTWORD TWORD ) ENTRY

VAR

AUX TWORD

I 1 .. LONGTWORD

BEGIN

AUX = 0

FOR I = 1 TO LONGTWORD DO

BEGIN

AUX = AUX \* BASE HEXA

CASE UNHEXA (. I .) OF

'0'

'1' AUX = AUX + 1

TIP00010  
 TIP00020  
 TIP00030  
 TIP00040  
 TIP00050  
 TIP00060  
 TIP00070  
 TIP00080  
 TIP00090  
 TIP00100  
 TIP00110  
 TIP00120  
 TIP00130  
 TIP00140  
 TIP00150  
 TIP00160  
 TIP00170  
 TIP00180  
 TIP00190  
 TIP00200  
 TIP00210  
 TIP00220  
 TIP00230  
 TIP00240  
 TIP00250  
 TIP00260  
 TIP00270  
 TIP00280  
 TIP00290  
 TIP00300  
 TIP00310  
 TIP00320  
 TIP00330  
 TIP00340  
 TIP00350  
 TIP00360  
 TIP00370  
 TIP00380  
 TIP00390  
 TIP00400  
 TIP00410  
 TIP00420  
 TIP00430  
 TIP00440  
 TIP00450  
 TIP00460  
 TIP00470  
 TIP00480  
 TIP00490  
 TIP00500  
 TIP00510  
 TIP00520  
 TIP00530  
 TIP00540

|           |                  |             |              |          |
|-----------|------------------|-------------|--------------|----------|
| '2'       | AUX              | =           | AUX + 2      | TIP00550 |
| '3'       | AUX              | =           | AUX + 3      | TIP00560 |
| '4'       | AUX              | =           | AUX + 4      | TIP00570 |
| '5'       | AUX              | =           | AUX + 5      | TIP00580 |
| '6'       | AUX              | =           | AUX + 6      | TIP00590 |
| '7'       | AUX              | =           | AUX + 7      | TIP00600 |
| '8'       | AUX              | =           | AUX + 8      | TIP00610 |
| '9'       | AUX              | =           | AUX + 9      | TIP00620 |
| 'A','A'   | AUX              | =           | AUX + 10     | TIP00630 |
| 'B','B'   | AUX              | =           | AUX + 11     | TIP00640 |
| 'C','C'   | AUX              | =           | AUX + 12     | TIP00650 |
| 'D','D'   | AUX              | =           | AUX + 13     | TIP00660 |
| 'E','E'   | AUX              | =           | AUX + 14     | TIP00670 |
| 'F','F'   | AUX              | =           | AUX + 15     | TIP00680 |
| OTHERWISE | AUX              | =           | 0            | TIP00690 |
| END       |                  |             |              | TIP00700 |
| END       |                  |             |              | TIP00710 |
| UNTWORD   | =                | AUX         |              | TIP00720 |
| END       |                  |             |              | TIP00730 |
| PROCEDURE | TWORDHEXA        |             |              | TIP00740 |
| (         | UNTWORD          | TWORD       | VAR UNHEXA   | TIP00750 |
|           |                  |             | DISTINTIVO ) | TIP00760 |
| ENTRY     |                  |             |              | TIP00770 |
| VAR       |                  |             |              | TIP00780 |
| I         | INTEGER          |             |              | TIP00790 |
| BEGIN     |                  |             |              | TIP00800 |
| FOR I     | =                | LONGTWORD   | DOWNT0 1 DO  | TIP00810 |
| BEGIN     |                  |             |              | TIP00820 |
| CASE (    | UNTWORD MOD BASE | HEXA )      | OF           | TIP00830 |
| 0         | UNHEXA (. I .)   | =           | '0'          | TIP00840 |
| 1         | UNHEXA (. I .)   | =           | '1'          | TIP00850 |
| 2         | UNHEXA (. I .)   | =           | '2'          | TIP00860 |
| 3         | UNHEXA (. I .)   | =           | '3'          | TIP00870 |
| 4         | UNHEXA (. I .)   | =           | '4'          | TIP00880 |
| 5         | UNHEXA (. I .)   | =           | '5'          | TIP00890 |
| 6         | UNHEXA (. I .)   | =           | '6'          | TIP00900 |
| 7         | UNHEXA (. I .)   | =           | '7'          | TIP00910 |
| 8         | UNHEXA (. I .)   | =           | '8'          | TIP00920 |
| 9         | UNHEXA (. I .)   | =           | '9'          | TIP00930 |
| 10        | UNHEXA (. I .)   | =           | 'A'          | TIP00940 |
| 11        | UNHEXA (. I .)   | =           | 'B'          | TIP00950 |
| 12        | UNHEXA (. I .)   | =           | 'C'          | TIP00960 |
| 13        | UNHEXA (. I .)   | =           | 'D'          | TIP00970 |
| 14        | UNHEXA (. I .)   | =           | 'E'          | TIP00980 |
| 15        | UNHEXA (. I .)   | =           | 'F'          | TIP00990 |
| OTHERWISE | UNHEXA (. I .)   | =           | '*'          | TIP01000 |
| END       |                  |             |              | TIP01010 |
| UNTWORD   | =                | UNTWORD DIV | BASE HEXA    | TIP01020 |
| END       |                  |             |              | TIP01030 |
| END       |                  |             |              | TIP01040 |
| PROCEDURE | HEXATBYTE        |             |              | TIP01050 |
| (         | UNHEXA           | DISTINTIVO  | VAR UNTBYTE  | TIP01060 |
|           |                  |             | TBYTE )      | TIP01070 |
| ENTRY     |                  |             |              | TIP01080 |
| VAR       |                  |             |              | TIP01090 |
| AUX       | TBYTE            |             |              |          |
| I         | 1 ..             | LONGTBYTE   |              |          |
| BEGIN     |                  |             |              |          |

|                                                       |          |
|-------------------------------------------------------|----------|
| AUX = 0                                               | TIP01100 |
| FOR I = 1 TO LONGTBYTE DO                             | TIP01110 |
| BEGIN                                                 | TIP01120 |
| AUX = AUX * BASE HEXA                                 | TIP01130 |
| CASE UNHEXA (. I .) OF                                | TIP01140 |
| '0'                                                   | TIP01150 |
| '1'   AUX = AUX + 1                                   | TIP01160 |
| '2'   AUX = AUX + 2                                   | TIP01170 |
| '3'   AUX = AUX + 3                                   | TIP01180 |
| '4'   AUX = AUX + 4                                   | TIP01190 |
| '5'   AUX = AUX + 5                                   | TIP01200 |
| '6'   AUX = AUX + 6                                   | TIP01210 |
| '7'   AUX = AUX + 7                                   | TIP01220 |
| '8'   AUX = AUX + 8                                   | TIP01230 |
| '9'   AUX = AUX + 9                                   | TIP01240 |
| 'A', 'A'   AUX = AUX + 10                             | TIP01250 |
| 'B', 'B'   AUX = AUX + 11                             | TIP01260 |
| 'C', 'C'   AUX = AUX + 12                             | TIP01270 |
| 'D', 'D'   AUX = AUX + 13                             | TIP01280 |
| 'E', 'E'   AUX = AUX + 14                             | TIP01290 |
| 'F', 'F'   AUX = AUX + 15                             | TIP01300 |
| OTHERWISE AUX = 0                                     | TIP01310 |
| END                                                   | TIP01320 |
| UNTBYTE = AUX                                         | TIP01330 |
| END                                                   | TIP01340 |
| PROCEDURE TBYTEHEXA                                   | TIP01350 |
| ( UNTBYTE   TBYTE   VAR UNHEXA   DISTINTIVO )   ENTRY | TIP01360 |
| VAR                                                   | TIP01370 |
| I   INTEGER                                           | TIP01380 |
| BEGIN                                                 | TIP01390 |
| FOR I = LONGTBYTE DOWNT0 1 DO                         | TIP01400 |
| BEGIN                                                 | TIP01410 |
| CASE ( UNTBYTE MOD BASE HEXA ) OF                     | TIP01420 |
| 0   UNHEXA (. I .) = '0'                              | TIP01430 |
| 1   UNHEXA (. I .) = '1'                              | TIP01440 |
| 2   UNHEXA (. I .) = '2'                              | TIP01450 |
| 3   UNHEXA (. I .) = '3'                              | TIP01460 |
| 4   UNHEXA (. I .) = '4'                              | TIP01470 |
| 5   UNHEXA (. I .) = '5'                              | TIP01480 |
| 6   UNHEXA (. I .) = '6'                              | TIP01490 |
| 7   UNHEXA (. I .) = '7'                              | TIP01500 |
| 8   UNHEXA (. I .) = '8'                              | TIP01510 |
| 9   UNHEXA (. I .) = '9'                              | TIP01520 |
| 10   UNHEXA (. I .) = 'A'                             | TIP01530 |
| 11   UNHEXA (. I .) = 'B'                             | TIP01540 |
| 12   UNHEXA (. I .) = 'C'                             | TIP01550 |
| 13   UNHEXA (. I .) = 'D'                             | TIP01560 |
| 14   UNHEXA (. I .) = 'E'                             | TIP01570 |
| 15   UNHEXA (. I .) = 'F'                             | TIP01580 |
| OTHERWISE UNHEXA (. I .) = '*'                        | TIP01590 |
| END                                                   | TIP01600 |
| UNTBYTE = UNTBYTE DIV BASE HEXA                       | TIP01610 |
| END                                                   | TIP01620 |
|                                                       | TIP01630 |
|                                                       | TIP01640 |

```

END
FUNCTION PASARHEXATBYTE
 (LA PALABRA DISTINTIVO VAR VALOR TBYTE) BOOLEAN ENTRY
VAR
 AUX BOOLEAN
BEGIN
 AUX = ESTBYTE (LA PALABRA)
 IF AUX THEN HEXATBYTE (LA PALABRA , VALOR)
 PASARHEXATBYTE = AUX
END
FUNCTION PASARHEXATWORD
 (LA PALABRA DISTINTIVO VAR VALOR TWORD) BOOLEAN ENTRY
VAR
 AUX BOOLEAN
BEGIN
 AUX = ESTWORD (LA PALABRA)
 IF AUX THEN HEXATWORD (LA PALABRA , VALOR)
 PASARHEXATWORD = AUX
END
FUNCTION ESDEC (UN DEC DISTINTIVO) BOOLEAN ENTRY
VAR
 OK BOOLEAN
 I INTEGER
BEGIN
 OK = TRUE
 FOR I = 1 TO LENGTH (UN DEC) DO
 OK = OK AND (UN DEC (. I .) IN DIGITOSDECIMALES)
 END
 FUNCTION ESOC (UN OCT DISTINTIVO) BOOLEAN ENTRY
 VAR
 OK BOOLEAN
 I INTEGER
 BEGIN
 OK = TRUE
 FOR I = 1 TO LENGTH (UN OCT) DO
 OK = OK AND (UN OCT (. I .) IN DIGITOSOCIALES)
 END
 FUNCTION ESBIN (UN BIN DISTINTIVO) BOOLEAN ENTRY
 VAR
 OK BOOLEAN
 I INTEGER
 BEGIN
 OK = TRUE
 FOR I = 1 TO LENGTH (UN BIN) DO
 OK = OK AND (UN BIN (. I .) IN DIGITOSBINARIOS)
 END
 END
 END
 / /

```

TIP01650  
 TIP01660  
 TIP01670  
 TIP01680  
 TIP01690  
 TIP01700  
 TIP01710  
 TIP01720  
 TIP01730  
 TIP01740  
 TIP01750  
 TIP01760  
 TIP01770  
 TIP01780  
 TIP01790  
 TIP01800  
 TIP01810  
 TIP01820  
 TIP01830  
 TIP01840  
 TIP01850  
 TIP01860  
 TIP01870  
 TIP01880  
 TIP01890  
 TIP01900  
 TIP01910  
 TIP01920  
 TIP01930  
 TIP01940  
 TIP01950  
 TIP01960  
 TIP01970  
 TIP01980  
 TIP01990  
 TIP02000  
 TIP02010  
 TIP02020  
 TIP02030  
 TIP02040  
 TIP02050  
 TIP02060  
 TIP02070  
 TIP02080  
 TIP02090  
 TIP02100

## MEMBER ARCHIVO

```

(*****) ARC00010
(* ARCHIVO *) ARC00020
SEGMENT ARCHIVO ARC00030
 ARC00040
PROCEDURE CMS (CONST S STRING VAR RC INTEGER) EXTERNAL ARC00050
VAR RC INTEGER ARC00060
 ARC00070
PROCEDURE LEER NOMBRE ARCHIVO ARC00080
 (VAR CMS NOMBRE , ARC00090
 FILE NOMBRE STRING ARC00100
 VAR BIEN BOOLEAN) ENTRY ARC00110
VAR ARC00120
 F NAME , F TYPE , F MODE ALPHA ARC00130
 LINE STRING (80) (* LINEA DE PANTALLA *) ARC00140
 POS , LONG LINE 0 .. 80 ARC00150
BEGIN ARC00160
 WRITELN ('NOMBRE , TIPO Y DISCO DEL ARCHIVO ') ARC00170
 READ (LINE) POS = 1 BIEN = TRUE ARC00180
 LONG LINE = LENGTH (LINE) ARC00190
 IF POS = LONG LINE ARC00200
 THEN ARC00210
 TOKEN (POS , LINE , F NAME) ARC00220
 SE BIEN = FALSE ARC00230
 IF POS = LONG LINE ARC00240
 THEN ARC00250
 TOKEN (POS , LINE , F TYPE) ARC00260
 ELSE BIEN = FALSE ARC00270
 IF POS = LONG LINE ARC00280
 THEN ARC00290
 TOKEN (POS , LINE , F MODE) ARC00300
 ELSE BIEN = FALSE ARC00310
 IF BIEN ARC00320
 THEN ARC00330
 BEGIN ARC00340
 WRITESTR (CMS NOMBRE , ARC00350
 'STATE ' , F NAME 9 , ARC00360
 F TYPE 9 , ARC00370
 F MODE 9) ARC00380
 WRITESTR (FILE NOMBRE , ARC00390
 'NAME = ' , F NAME 8 , ' . ' , ARC00400
 F TYPE 8 , ' . ' , ARC00410
 F MODE 8) ARC00420
 CMS NOMBRE = COMPRESS (CMS NOMBRE) ARC00430
 FILE NOMBRE = COMPRESS (FILE NOMBRE) ARC00440
 END ARC00450
 END ARC00460
 END ARC00470
PROCEDURE VERIFICAR ARCHIVO ARC00480
 (CONST CMS NOMBRE STRING ARC00490
 VAR BIEN BOOLEAN) ENTRY ARC00500
BEGIN ARC00510
 CMS (CMS NOMBRE , RC) ARC00520
 BIEN = (RC = 0) ARC00530
END ARC00540
(* FIN ARCHIVO

```

FILE PROGSEG MAQLIB 41 VM/SP RELEASE 5 \*\*\* U.N.P.S.J.B \*\*\* PUT 8801+

(\*\*\*\*\*  
/ /

ARC00550

```

MEMBER ENSAMBLA
PROGRAM ENSAMBLA (INPUT , OUTPUT , ENTRADA , AUXILIAR , SALIDA)
PROCEDURE ENSAMBLADOR
 (VAR ENTRADA , AUXILIAR TEXT) EXTERNAL
FUNCTION FINAL
 (VAR AUXILIAR , SALIDA TEXT) BOOLEAN EXTERNAL
PROCEDURE LEER NOMBRE ARCHIVO
 (VAR CMS NOMBRE ,
 FILE NOMBRE STRING
 VAR BIEN BOOLEAN) EXTERNAL
PROCEDURE VERIFICAR ARCHIVO
 (CONST CMS NOMBRE STRING
 VAR BIEN BOOLEAN) EXTERNAL
PROCEDURE PRINCIPAL (CONST FILEENTRADA , FILSALIDA STRING)
VAR
 ENTRADA , AUXILIAR , SALIDA TEXT
BEGIN
 RESET (ENTRADA , FILEENTRADA)
 REWRITE (AUXILIAR , 'NAME=AUXILIAR.ENS.41')
 ENSAMBLADOR (ENTRADA , AUXILIAR)
 CLOSE (ENTRADA)
 CLOSE (AUXILIAR)
 RESET (AUXILIAR , 'NAME=AUXILIAR.ENS.41')
 WRITE (SALIDA , FILSALIDA)
 WRITELN ('BIEN = ' , FINAL (AUXILIAR , SALIDA))
 CLOSE (ENTRADA)
 CLOSE (AUXILIAR)
 CLOSE (SALIDA)
END
VAR
 CMSENTRADA , CMSSALIDA , FILEENTRADA , FILSALIDA , STRING (80)
 BIEN BOOLEAN
BEGIN
 LEER NOMBRE ARCHIVO (CMSENTRADA , FILEENTRADA , BIEN)
 LEER NOMBRE ARCHIVO (CMSSALIDA , FILSALIDA , BIEN)
 VERIFICAR ARCHIVO (CMSENTRADA , BIEN)
 IF BIEN THEN PRINCIPAL (FILEENTRADA , FILSALIDA)
END
/ /

```

ENS00010  
 ENS00020  
 ENS00030  
 ENS00040  
 ENS00050  
 ENS00060  
 ENS00070  
 ENS00080  
 ENS00090  
 ENS00100  
 ENS00110  
 ENS00120  
 ENS00130  
 ENS00140  
 ENS00150  
 ENS00160  
 ENS00170  
 ENS00180  
 ENS00190  
 ENS00200  
 ENS00210  
 ENS00220  
 ENS00230  
 ENS00240  
 ENS00250  
 ENS00260  
 ENS00270  
 ENS00280  
 ENS00290  
 ENS00300  
 ENS00310  
 ENS00320  
 ENS00330  
 ENS00340  
 ENS00350  
 ENS00360  
 ENS00370

## MEMBER TABLA

ARREGLO SIMBOLOS = ARRAY (. INDICEIDENTIFICADOR .) OF DISTINTIVO

TAB000010

TABLADISTINTIVO = RECORD

TAB000020

UN ARREGLO SIMBOLOS ARREGLO SIMBOLOS

TAB000030

UN ARREGLO VALOR ARRAY (. INDICEIDENTIFICADOR .) OF DISTINTIVO

TAB000040

ELINDICE INDICEIDENTIFICADOR

TAB000050

END

TAB000060

TABLATBYTE = RECORD

TAB000070

UN ARREGLO SIMBOLOS ARREGLO SIMBOLOS

TAB000080

UN ARREGLO VALOR ARRAY (. INDICEIDENTIFICADOR .) OF TBYTE

TAB000090

ELINDICE INDICEIDENTIFICADOR

TAB000100

END

TAB000110

TABLATWORD = RECORD

TAB000120

UN ARREGLO SIMBOLOS ARREGLO SIMBOLOS

TAB000130

UN ARREGLO VALOR ARRAY (. INDICEIDENTIFICADOR .) OF TWORD

TAB000140

ELINDICE INDICEIDENTIFICADOR

TAB000150

END

TAB000160

MEMBER RESOLVE

RESOLVENTE = RECORD

RES00010

CLASE CLASEDECLARACION

RES00020

UN ARG

RES00030

VALOR DISTINTIVO

RES00040

ASIGNADO BOOLEAN

RES00050

LONGITUD TWCRO

RES00060

END

RES00070

/ /

MEMBER ESTTIPOS

|                           |          |
|---------------------------|----------|
| CCNST                     | EST00010 |
| TOPETBYTE = 255           | EST00020 |
| LONGTBYTE = 2             | EST00030 |
| TOPETWORD = 64 * 1024 - 1 | EST00040 |
| LONGTWORD = 4             | EST00050 |
| BASE HEXA = 16            | EST00060 |
| TYPE                      | EST00070 |
| TBYTE = 0 .. TOPETBYTE    | EST00080 |
| TWORD = 0 .. TOPETWORD    | EST00090 |

/ /

MEMBER ESTTABLA

CCNST

AUSENTE = 0

TOPEIDENTIFICADOR = 255

TYPE

INDICEIDENTIFICADOR =AUSENTE .. TOPEIDENTIFICADOR

CLASEDECLARACION = ( CTBYTE , CTWORD ,  
CDEC , COCT , CBIN ,  
CALFABETICO , CETIQUETA ,  
CESPECIALES ,  
CNUMBYTE , CNUMWORD , CERROR )

EST00010  
EST00020  
EST00030  
EST00040  
EST00050  
EST00060  
EST00070  
EST00080  
EST00090  
EST00100

FILE ESTRUCTU MACLIB 41 VM/SP RELEASE 5 \*\*\* U.N.P.S.J.B \*\*\* PUT 8801+

MEMBER DISTINTI

TYPE

DISTINTIVO = STRING ( 16 )

DIS00010

DIS00020

/ /