UNIVERSIDAD AUTONOMA DE NUEVO LEON
FACULTAD DE INGENIERIA MECANICA Y ELECTRICA
DIVISION ESTUDIOS DE POST-GRADO

EL COMPILADOR INTERACTIVO FIFTH

T E S I S

En opción al Grado de Maestro en Ciencias de la Ingeniería
Eléctrica con especialidad en Electrónica

P R E S E N T A
FRANCISCO JAVIER DE LA GARZA SALINAS

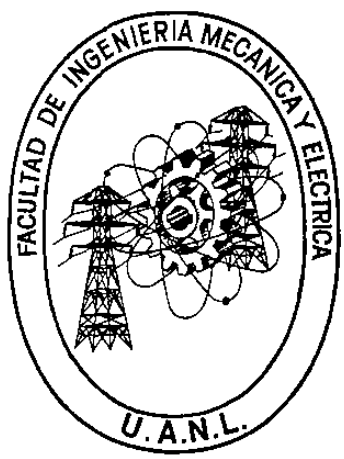
SAN NICOLAS DE LOS GARZA, N. L. DIC. DE 1997.

EL COMPILADOR INTERACTIVO FIFTH

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MACANICA Y ELECTRICA

DIVISION DE ESTUDIOS DE POST-GRADO



EL COMPILADOR INTERACTIVO FIFTH

TESIS

EN OPCION AL GRADO DE MAESTRO EN CIENCIAS DE LA

INGENIERIA ELECTRICA CON ESPECIALIDAD EN ELECTRONICA

QUE PRESENTA

FRANCISCO JAVIER DE LA GARZA SALINAS

SAN NICOLAS DE LOS GARZA, N.L. 5 DE DICIEMBRE DE 1997

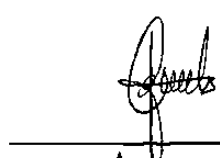
UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MACANICA Y ELECTRICA

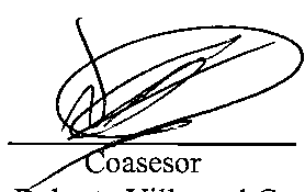
DIVISION DE ESTUDIOS DE POST-GRADO

Los miembros del Comité de tesis recomendamos que la presente tesis "El compilador interactivo Fifth" realizada por el Ing. Francisco Javier de la Garza Salinas sea aceptada como opción para obtener el grado de Maestro en Ciencias de la Ingeniería Eléctrica con especialidad en Electrónica.

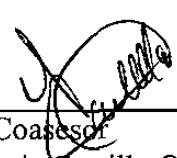
El Comité de Tesis



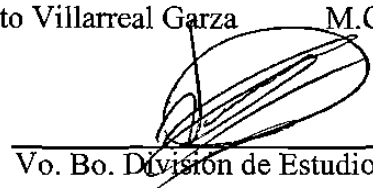
Asesor
M.C. Luis Manuel Camacho Velázquez



Coasesor
M.C. Roberto Villarreal Garza



Coasesor
M.C. José Luis Castillo Ocañas



Vo. Bo. División de Estudios de Postgrado
M.C. Roberto Villarreal Garza

San Nicolás de los Garza, N.L., a 5 de Diciembre de 1997

A mis Padres poy ayudarme a llegar al lugar en donde estoy,

A Lety por ser compañera de mis sueños,

A Javy y Susy por regalarme su tiempo,

Y a mis compañeros por alentarme a seguir adelante

GRACIAS

Prólogo

El uso de los microprocesadores a principios de los años 70's marcó el inicio de la nueva era en la computación. Las expectativas que creó esta nueva tecnología hicieron que se abriera un gran mercado y, gracias a esto, los costos se fueran hacia abajo. Apple Computer inició la carrera de las PC, que muy pronto muchas compañías seguirían. Surgió entonces el termino compatibilidad, utilizado en equipo de cómputo para identificar a máquinas capaces de compartir elementos, especialmente software. Sin embargo, el problema de compatibilidad no tiene nada que ver con programación, se trata del diseño mismo del microprocesador. Cada instrucción desarrollada en un lenguaje de alto nivel, como Basic, es transformada a código máquina. Si por alguna razón se requiere cambiar de sistema, se necesita otro compilador.

La idea de que un compilador funcione para varios microprocesadores no es nueva, pero como fabricante tiene sus propios métodos de diseño, diferentes manejos de datos y de E/S, esta tarea no es sencilla. Los requisitos para un lenguaje que pueda ser llevado fácilmente de un microprocesador a otro sería que las instrucciones básicas que lo forman sean pocas y que el manejo de los datos sea sencilla. Fifth cumple no sólo con esta característica, si no que además, es posible trabajar con varios microprocesadores simultáneamente convirtiéndose en una herramienta ideal para el manejo de procesos en tiempo real.

1. Síntesis

Los principios y las bases de la compilación interactiva, el manejo de un lenguaje de programación sencillo y transportable a diversos microprocesadores y su aplicación en la PC se describen en la presente tesis. El sistema de programación descrito es un compilador interactivo para el lenguaje Fifth. El sistema soporta a varias familias de microcontroladores y de procesadores digitales de señal, puede ser utilizado como sistema operativo y de desarrollo. Con la capacidad interactiva se pueden programar aplicaciones con varios microprocesadores sin la necesidad de emuladores. El sistema es capaz de modelar un sistema digital como un conjunto de microprocesadores elementales, que se estructuran en clases de un mismo tipo.

Inicialmente se describe brevemente los alcances del trabajo en el capítulo 2. En el capítulo 3 se describe el concepto de un compilador interactivo y se inicia la descripción y funcionamiento del sistema Fifth. La forma de trabajo del compilador, las estructuras de datos y la programación en el sistema se describen en el capítulo 4 y tratando de que las explicaciones sean lo más claras posibles se describe el uso del sistema en el capítulo 5. En el capítulo 6 se describen detalles técnicos tanto de programación como de construcción del compilador. Debido a que el sistema está basado en una PC, en el capítulo 7 se hace referencia a las funciones especiales que están disponibles. El sistema Fifth es de gran ayuda para la programación en tiempo real y para ello cuenta con interrupciones y la posibilidad de ejecutar procesos paralelos que se explican en el capítulo 8. Finalmente la construcción de programas ejecutables y de mejoras al sistema por parte del usuario se explican en el capítulo 9.

2. Introducción

El objetivo de esta tésis es hacer una descripción de un compilador interactivo y de la programación en ambiente multiprocesador. Esta tésis fue desarrollada en base a mi estancia en la Universidad Técnica de Hamburgo, como parte de mis estudios de especialización y trabajo como investigador asociado en el departamento de Técnica Informática VI de la misma Universidad.

El diseño de sistemas cada vez más complicados requiere de herramientas que cumplan diversos objetivos como: facilidad de programación, aplicación a diversas situaciones y código rápido y compacto. Esta tesis tiene como objetivo fundamental describir el diseño de un compilador interactivo, basando los ejemplos y aplicaciones.

Los avances en electrónica digital, especialmente en el campo de los microprocesadores, han exigido que los diseñadores de electrónica busquen el máximo nivel 'state of the art', esta tarea se ha dificultado debido a que el surgimiento de un nuevo microprocesador, con nuevas características y funcionalidad, implique una modificación en los programas ya escritos. Un lenguaje de programación con pocas funciones elementales, permite la migración hacia otros sistemas sin mucha dificultad.

La metodología de presentación de la tesis consta de una explicación sobre un concepto o instrucción para observar su uso posteriormente por medio de ejemplos. A pesar de que las explicaciones tratan de ser claras, la experiencia en el área de microprocesadores facilita la lectura.

3. Introducción al Sistema Fifth

El sistema Fifth es un compilador interactivo para el lenguaje Fifth, está integrado por un editor y diversas funciones del sistema operativo que lo convierten en un sencillo ambiente de programación. Para aplicaciones que requieran del uso de instrucciones especializadas se cuenta con lenguaje ensamblador para el microprocesador 80X86. De igual forma que se realizan programas en BASIC o Forth, en Fifth pueden ser desarrollados o ejecutados programas para la PC. Fifth soporta el uso de los puertos seriales y puede hacer uso del coprocesador matemático y los gráficos de alta resolución (CGA, EGA O VGA). Fifth permite además el uso de interrupciones y procesos paralelos para aplicaciones en tiempo real. Además, el sistema Fifth puede ser ampliado en distintas formas y es capaz de generar programas para otros microprocesadores distintos al de la PC.

La implementación actual del sistema Fifth, descrita en esta tesis, requiere una PC compatible con MS-DOS y necesita 384 KB de RAM. El sistema Fifth para la PC consta de los siguientes programas:

IPU.EXE	Compilador y ambiente de programación.
KONF.COM	Programa de configuración.
SWFPT.TXT	Programa con funciones de punto flotante.
PC.COM	Programa para hacer una segunda PC el sistema destino de Fifth.
PC.TXT	Definición del modo para el sistema destino de Fifth.

Al igual que con cualquier otro sistema, se debe hacer una copia de seguridad de estos archivos antes de su uso. Para una mayor comodidad de trabajo los archivos se copian en un subdirectorio del disco duro.

El sistema Fifth se inicia con la instrucción: "IPU" y de manera general con: "IPU Comando" en donde "Comando" es una instrucción de Fifth, tal y como se explicará en los siguientes capítulos. Esta instrucción es traducida y ejecutada, después de lo cual se activa el editor para otras instrucciones. Por ejemplo, se inicia con: IPU LT NOMBRE el texto NOMBRE.TXT es cargado en el área de textos del editor, y con IPU INCLUDE NOMBRE QUIT el programa es cargado, traducido y finalmente se abandona el IPU. Instrucciones de este tipo pueden ser utilizadas por archivos de lotes.

El editor del sistema Fifth sirve para editar el texto de los programas, así como para efectuar algunas instrucciones. Se explica en detalle en el capítulo 5. El editor de Fifth, con su estructura especial, soporta la programación interactiva consistente en que varias líneas sean compiladas y ejecutadas al presionar una tecla y después puedan ser editadas nuevamente. A pesar que los programas pueden ser escritos en otro editor y posteriormente ser traducidos con el IPU, es importante acostumbrarse a utilizar el editor de Fifth, ya que cuenta con algunas funciones especiales útiles con la programación interactiva.

Las instrucciones en el sistema Fifth son editadas y transferidas al sistema para su procesamiento al presionar la tecla F1. Después de ejecutar una tarea el sistema regresa automáticamente al modo de edición. Con el comando "QUIT" se abandona el IPU.

Con el programa KONF.COM pueden ser modificados algunos parámetros de operación, como la velocidad de transmisión del puerto serie o el tipo de monitor usado. El programa se inicia con el comando 'KONF' y después pregunta por el nombre del sistema Fifth que se quiere modificar y otros datos. 'KONF' puede ser utilizado por versiones modificadas del 'IPU'. 'KONF' es explicado en detalle en el capítulo 4.8. Algunas otras modificaciones son posibles con el comando 'BUILD' descrito en el capítulo 9. 'KONF' es un ejemplo de un programa desarrollada en Fifth para la PC que hace uso de entrada y salida en archivos del disco.

4. Fifth

Esta sección contiene una introducción al lenguaje Fifth. Se recomienda al lector acompañar la lectura con sus correspondientes prácticas en el sistema Fifth. Para este propósito se encuentran algunos ejemplos al final de cada sección. Esta introducción concentra en las estructuras básicas del lenguaje. La información detallada sobre datos técnicos de programación se encuentra en el capítulo 6. Las funciones especiales para la implementación en la PC se describen en el capítulo 7.

Fifth es un lenguaje muy sencillo y por lo tanto fácil de aprender. Las instrucciones básicas son pocas y sólo es posible utilizar un tipo lógico de datos. Fifth permite ser ampliado de acuerdo a necesidades específicas con funciones y estructuras de control y datos especiales. El lenguaje permite el acceso a todas las localidades de memoria de E/S de la computadora permitiendo una programación cercana a la máquina.

El sistema Fifth es interactivo. Los programas en Fifth no necesitan ser editados y compilados en un sólo bloque, ya que esta tarea puede ser realizada y aprobada paso a paso. Todos los elementos del programa pueden ser ejecutados interactivamente y probada con parámetros propios. Esta característica interactiva se conserva aún cuando el sistema se extiende a otros microprocesadores conectados a la PC. Los programas desarrollados interactivamente son, dada su construcción y la estructura de control usada, bien estructurados y traducidos por el compilador en un óptimo código máquina.

De los lenguajes de programación comunes, Fifth puede compararse con Forth, pero también tiene algunas similitudes con C y otros lenguajes funcionales como Lisp. Fifth se utiliza para trabajos con gráficas, programación en tiempo real, control y recolección de datos. Se ha consolidado como un lenguaje de cómoda implementación y existe en forma ampliada como cross-compilador para distintos microprocesadores. Además, Fifth puede ser utilizado para programar sistemas con varios microprocesadores comunicándose entre sí. Algunos de los microprocesadores programables en Fifth son:

Familia 8031,6303	Microcontroladores de 8 bits
Z (1) 80, 6809,6586	Microprocesador estándar de 8 bits
Familia 80 X 86	Microprocesador estándar de 16 bits
V25,78310,80C166	Microcontroladores de 16 bits
Familia 68000	Microprocesador estándar de 16 bits
Familia de Transputers	Microprocesador RISC de 16 y 32 bits
RTX 2000/1	Microprocesador de 16 bits orientador a Forth/Fifth
TMS34010/20	Microprocesadores gráficos
DSP56001, DSP32C	Procesadores de señal

El sistema Fifth ha tenido un desarrollo continuo en estrecho contacto con usuarios profesionales. Los usuarios pueden ayudar a través de sugerencia y crítica constructiva a hacer de Fifth una herramienta aún más productiva.

4.1 Programación en conceptos

Una de las características esenciales de Fifth es la programación en conceptos. Los conceptos son acciones, poco o muy complejas, de la computadora. Los bloques de un programa son definidos por una serie de conceptos conocidos y reunidos bajo un nuevo concepto. Formalmente los conceptos son una palabra o un grupo de palabras unidas. Fifth permite la construcción de clases, en donde conceptos semejantes pueden ser agrupados, de manera similar a la organización de los archivos en un directorio del disco. El concepto de una clase está formado por: el nombre de la clase y un identificador dentro de la misma clase. Para simplificar la representación utilizaremos sólo conceptos de una sola palabra que sirven como nombres para un bloque de programa.

Técnicamente visto un concepto es un bloque de un programa de Fifth, una subrutina ejecutable. Más adelante se muestra que las estructuras de control para realizar interacciones son manejadas como conceptos dentro del texto del programa.

Sólo algunas estructuras sirven para realizar las uniones entre conceptos, lo que crea un lenguaje de programación funcional.

Durante el desarrollo de un programa, los subprogramas reciben un nombre para posteriormente ser ejecutados referenciándolos. Supongamos que 'lectura', 'cálculo' y 'despliegue' son los nombres de tres subprogramas, entonces durante la ejecución del 'programa' deberán colocarse en serie.

```
Lectura Cálculo Despliegue
```

Las palabras (conceptos) de esta parte de programa son las llamadas instrucciones en otros lenguajes de alto nivel. Aquí no existen símbolos separadores entre las instrucciones; como tampoco son necesarios los paréntesis de argumentos, las palabras son trabajadas como en una oración, las palabras son separadas por un espacio o por el final de un renglón. En casos particulares se puede prescindir del espacio entre palabras, como cuando la segunda palabra inicia con el símbolo 'o'. También se evita el espacio cuando las palabras forman una unidad. Los nombres de las clases y selectores de conceptos unidos son seleccionados de tal manera que no sea necesario un símbolo separador entre ellos.

A partir de bloques antiguos pueden ser formados nuevos subprogramas más complejos y recibir un nombre. Una definición de este tipo tiene la forma:

```
N1 N2 N3

...

>: NN
```

En donde NN es el nombre del conjunto de subprogramas N1,N2 y N3 con el cual puede ser ejecutado posteriormente; en otras palabras NN es el nombre del nuevo concepto. El comando '>:' sirve como terminación del texto anterior y asigna el nuevo nombre. Por lo tanto, el nombre de la función nueva aparece al final del texto referenciado. La palabra '>:' indica el final de una definición. El inicio de una definición no tiene una marca especial. El comienzo de una definición es el inicio del texto o el final de la definición anterior.

Los nombres pueden ser de hasta 63 caracteres y su significado puede ser seleccionado de forma que ayuden al buen entendimiento del programa. No se hace diferencia alguna entre mayúsculas y minúsculas. Los nombres pueden contener algunos símbolos especiales, para facilitar su uso se colocaron en el editor una serie de teclas ALT (al presionar ALT se muestran estos caracteres). Ciertos símbolos (ver cap. 6.1) sólo pueden estar al inicio ('.') o al final ('.') de una palabra. Los números, el símbolo '%' y los símbolos básicos como '+', '-', 'and' o '>:' están prohibidos como nuevos nombres. El mismo nombre puede repetirse varias veces pero es la última definición para ese modo de compilación la válida.

La definición de un subprograma con una sola instrucción es la unidad mínima permitida por el sistema Fifth que trabaja interactiva e incrementalmente. Los subprogramas editados pueden tener tantas líneas como sean necesarias. Durante el trabajo con el sistema Fifth puede ser probada cada nueva definición y así conseguir que cada función que se agrega no contenga errores y en caso de que estos existan, eliminarlos en las primeras etapas. Naturalmente un programa puede ser editado y compilado como un conjunto completo. Pero para la etapa de pruebas se recomienda proseguir en pasos.

Normalmente un programa en Fifth está formado por muchas definiciones, como la anterior. El compilador no limita al programa en cuanto a su formato. Para mantener una sencilla localización se sugiere colocar el símbolo '>:' al inicio de un renglón y no incluir más instrucciones en la misma línea. El editor de Fifth apoya esta convención a través de un modo especial de inspección, en el cual, sólo se muestran las líneas que inician con

'>:' y facilita la localización y posicionamiento en el subprograma deseado. Entre las definiciones debe de haber al menos dos líneas en blanco. En cualquier lugar pueden usarse líneas en blanco para aumentar la claridad de lectura del programa.

La técnica de programación, por medio de la cual los elementos son colocados sucesivamente formando funciones más complejas, es conocido como método 'Bottom-Up'. Este método obliga a la estructuración del programa en una solución del programa construida en etapas, lo que requiere un análisis 'Top-Down' del problema. Fifth obliga a una programación estrictamente 'Bottom-Up'. Cada nueva definición sólo puede hacer uso de funciones definidas anteriormente en el programa. El hecho de que no existen referencias hacia adelante es también una garantía de que el programa crece a través de nuevas definiciones y de que pueda ser creado inmediatamente código ejecutable. Las definiciones de un programa muy grande irán siendo cada vez más abstractas, ya que tras de un concepto se irán ocultando detalles. Por ejemplo, la última definición podría construir la función total de un programa:

Lectura Cálculo Despliegue

>: Análisis_De_Señal

Podría estar al inicio de las especificaciones para un programa de 'Análisis_De_Señal" y estar lista durante el diseño del programa para trabajar con la sintaxis de Fifth.

El sistema Fifth contiene muchos elementos predefinidos a partir de los cuales pueden formarse subprogramas. El comando 'LIST' muestra los nombres definidos con anterioridad. La ejecución de los nuevos subprogramas no tiene diferencia sintáctica alguna con la ejecución de las funciones ya definidas, las funciones nuevas pueden considerarse como extensiones del lenguaje Fifth.

Las palabras 'LIST' y '>:' son ejemplos de instrucciones de control que no pueden ser compiladas como nombres de elementos de un programa, sino que controlan al sistema o el trabajo del compilador.

En caso de que un texto de programa sea transferido al sistema por medio de la tecla F1 y que este no termine como definición, será ejecutado inmediatamente después de su compilación. Así, tenemos un modo de interpretación especial para la ejecución interactiva de programas o una instrucción de ejecución explícita. Para ejecutar una función FUNCION definida con anterioridad ejecutamos:

FUNCION F1

Ejemplo: Con el siguiente texto

"¡Hola!"

>: h

se define la función 'h' que muestra en pantalla el texto "Hola". los textos comprendidos entre dobles comillas, incluso de varias líneas, son interpretados en Fifth como instrucciones de despliegue de textos. Intente ejecutar los siguientes comandos, al final de cada renglón presione la tecla F1 para transferir al sistema. La tecla Return se utiliza sólo para editar textos de varias líneas. Los dos puntos ".." indican que el texto siguiente hasta el final del renglón es un comentario. Los comentarios sirven aquí como explicación y no es necesario teclearlos.

"Hola" ..se ejecuta inmediatamente

"Hola" >:h ..define "h"

h

h h h

h h h "¡Puedo Fifth!" >:hh ..utiliza h

List

hh

4.2 Parámetros y resultados

Los subprogramas en Fifth pueden trabajar con varios parámetros y entregar uno o más resultados. Los subprogramas sin resultados son procedimientos que pudieran tener parámetros, los que regresan resultados son conocidos como funciones en otros lenguajes de programación. En adelante se nombrarán ambos casos como funciones de Fifth.

Los parámetros y resultados de las funciones de Fifth son datos de cierto tamaño dependiente de la aplicación. Estos son colocados automáticamente en una pila (stack) diseñada para este propósito, de manera similar en la que se colocan libros en una pila (última entrada-primera salida). Una función con N parámetros, los toma de la pila y también coloca ahí los posibles resultados.

Los parámetros pueden ser números constantes en distintos formatos (ver Cap. 6.1), términos en forma general, estar dentro del programa o ser el resultado de funciones anteriores (anidación de funciones). Aquí anteceden los parámetros al llamado de la función (notación postfija), como también corresponde a la secuencia de ejecución. Las funciones no requieren de paréntesis para los parámetros, si no que opera con los datos colocados con anterioridad en la pila.

El llamado a una función con dos parámetros y un resultado tiene la siguiente forma:

```
97 103 NN
```

en donde el resultado de la función 'NN' se coloca sobre la pila. Ambos números constantes son colocados durante la ejecución del programa en la pila y hasta aquí son totalmente independientes del llamado a la función 'NN', posteriormente serán tomados de la pila y trabajados por la función.

Para describir una función de Fifth deben ser especificados el número y significado de los parámetros y resultados. En adelante se usará la siguiente notación:

X Y Z FUNC → R S T

en donde X, Y y Z son los parámetros tomados de la pila por FUNC y R, S y T los resultados entregados Z es el último parámetro colocado sobre la pila antes de la ejecución de FUNC y con ello el elemento superior de la pila. El elemento superior sobre la pila es el llamado 'Top on stack' (TOS) y al siguiente elemento se le conoce como 'Next on stack' (NOS). Después de la ejecución de la función, T es el TOS y S el NOS. La línea también puede ser vista como una fórmula de sustitución: la serie 'X Y Z FUNC' es equivalente a 'R S T', sólo en lo referente a los datos sobre la pila. La cantidad de parámetros y resultados de una definición no debe ser especificado (a excepción de los OBJETOS ver 4.5) ya que esta información no es necesaria. Los siguientes casos especiales aparecen a menudo:

X1 . . . Xn	FUN →	.. Procedimiento sin resultados
X1 . . . Xn	FUN → Y	.. Función con un resultado
X1 . . . Xn	FUN → Y	.. Seudovariable

El tipo del dato en la pila es el tipo estándar utilizado por Fifth para parámetros y resultados. El tipo es dependiente de la aplicación y se orientan al tamaño de datos con los que opera el microprocesador usado o con el tipo de aplicación. Los microprocesadores pequeños utilizan sólo un byte, mientras que para algunos procesadores de señal programados con Fifth utilizan números de punto fijo de 24 + 24 bits. El uso de un tipo estándar de datos no debe ser visto como una limitante del lenguaje. Bajo un aumento considerable en la velocidad del sistema y una eficiencia en el manejo de memoria se puede representar la implementación en donde el tipo y el dato están integrados.

La implementación en la PC utiliza datos de 16 bits que pueden ser interpretados como enteros en el rango - 32768 a 32767. Como parámetros pueden ser utilizados número de punto flotante que serían convertidos en enteros, a menos que sean almacenados en

variables del mismo tipo. Los datos de 16 bits pueden también ser interpretados como direcciones de funciones o de estructuras de datos.

Junto al llamado de funciones, los términos son el segundo elemento más importante de un programa en Fifth. Los términos describen a los datos que serán colocados sobre la pila. Así puede estar formado un programa con series de llamados a funciones y términos. En su forma más sencilla, los términos son constantes, que pueden ser escritos en distintos formatos (decimal, hexadecimal, etc.) o que a partir de la declaración de una constante (ver Cap. 6.1) son también introducidos de manera simbólica. De forma general los términos son formados por constantes, variables y los operadores aritméticos y lógicos:

+ - * /

AND OR EOR

así como por comparadores:

= <= < <> >= >

En caso que el resultado de una comparación sea falso, se almacena un '0' en la pila. En caso que la comparación sea verdadera, se almacena un '-1'.

Para describir los términos se puede utilizar la notación algebraica convencional. Un término se escribiría:

2 * x + 1 < 443 and z < 5.

Las operaciones tienen distintas prioridades. La multiplicación y la división tiene mayor prioridad que la suma y la resta, y éstas a su vez, tienen mayor prioridad que los comparadores y por último están las operaciones lógicas. Para las operaciones de comparación es válida la siguiente regla:

$x \leq y \leq z$ es equivalente a

$x \leq y$ and $y \leq z$

En los términos pueden ser utilizadas las operaciones '²' (cuadrado) y '|' (Valor absoluto), en la forma: x^2 ó $|x|$. donde puede ser usado el tipo de escritura matemática acostumbrada.

Para obtener otra prioridad en el orden de ejecución de las operaciones o para llamar funciones dentro de un término se pueden utilizar los corchetes '[' y ']'

3 * [x + 2] ó 2 + [x y FUNC]

Los elementos colocados en la parte superior de la pila, puede ser referenciado en la formulación de un nuevo término igual que una variable con los nombres 'TOS' y 'NOS': 3 * TOS ≤ 76 and x < NOS

El inicio y final de un término no requiere una marca especial. Un símbolo de operación y el siguiente dato corresponden a un mismo término, mientras que un término nuevo inicia con un símbolo que no sea de operación.

Además de la notación algebraica, puede ser utilizada la notación postfija, en donde las operaciones trabajan como funciones especiales con dos parámetros y un resultado en el cual el concepto de los términos se limita a constantes y variables. El término del primer ejemplo se escribiría como:

2 x * 1 + 443 ≤ z 5 < and.

La selección del tipo de notación utilizada se realiza por medio de las instrucciones 'UPN' y ' ALG'. El tipo de notación no debe ser cambiado en medio de un programa. El compilador se encuentra inicialmente en el tipo UPN. La escritura algebraica es más sencilla de leer en operaciones complicadas, pero ocasiona el manejo especial de las operaciones con respecto al llamado de funciones y un acceso complicado a la pila. La notación UPN es preferida por la mayoría de los programadores de Fifth. La posibilidad de elección muestra que el tipo de notación de un lenguaje orientado a pilas como Fifth

no es forzadamente UPN y no debe ser visto como parte esencial de la definición del lenguaje. En lo que respecta al código generado no existe ninguna diferencia. Las operaciones x^2 y |x| son válidos también en UPN.

Como se mencionó anteriormente, en la implementación para la PC (y en la mayoría de las otras) pueden ser utilizados números de punto flotante y variables en los términos. En la implementación para la PC se utiliza el coprocesador matemático para evaluar los términos de punto flotante. En caso de que no exista un coprocesador, el archivo SWFPT.TXT debe ser compilado antes para que las operaciones de punto flotante se realicen. Las conversiones necesarias dentro de un término se realizan automáticamente. La división '/', en caso de estar incluida, es una operación de punto flotante. Las funciones 'SIN', 'COS', 'LN', 'EXP', 'ATAN', 'SQRT' son operaciones especiales de punto flotante, que toman sin necesidad de conversión un número de punto flotante como parámetro y arrojan como resultado un valor de punto flotante. En algunos casos, en caso de ser necesario, ocurrirá alguna conversión de TOS y NOS al formato estándar. Si permanecen otros términos de punto flotante además de TOS y NOS se reportará un error de conversión. La conversión de un término de punto flotante puede realizarse con la instrucción explícita F>I. Aún cuando el tipo de dato transferido este fijo al formato de enteros, se puede al menos trabajar dentro de un término con números de punto flotante, los cuales tienen un formato totalmente distintos en memoria.

Para manipular a los elementos superiores de la pila de datos existen funciones de almacenamiento, que permiten el uso libre de los parámetros en una función:

DUP .. Duplica el elemento superior de la pila

OVER .. Duplica el segundo elemento superior de la pila

SWAP .. Intercambia los dos elementos superiores de la pila

RM .. Retira el elemento superior de la pila

RMN .. Retira el segundo elemento superior de la pila

Utilizando la función 'DUP', un programa puede utilizar varias veces el valor del TOS como parámetro para una función. La secuencia ' OVER OVER ' duplica al par de elementos superiores de la fila. 'RMN' equivale a ' SWAP RM '. Se puede utilizar, por ejemplo, cuando se ha formado un nuevo término utilizando el 'TOS' y se desea retirar al antiguo. Si se utiliza la notación 'UPN', 'TOS' y 'NOS' son equivalentes a 'DUP' y 'OVER'. Otras funciones básicas son:

x NOT	→ y	.. Complemento en bits del TOS
x - -	→ y	.. Cambio de signo
x ABS	→ y	.. Valor absoluto, como I X I
x y/R	→ p q	.. División entera, q cociente, p residuo
x SPLIT	→ m l	.. Parte en dos a x, m parte alta, l la baja
m l PACK	→ x	.. Inverso a SPLIT
x y MAX	→ z	.. Máximo
x y MIN	→ z	.. Mínimo

Para mostrar los términos o resultados de funciones se utiliza el signo de interrogación '?' como un parámetro y sin resultados. Se pueden elegir entre distintos formatos para '?' (ver 7.2). Ejemplo:

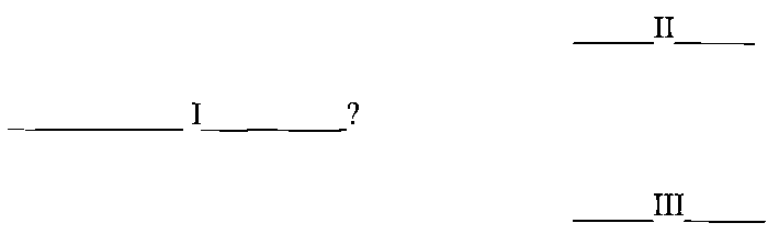
Las siguientes instrucciones muestran el uso de las funciones aritméticas y el despliegue numérico. Cada ejemplo crea sus propios parámetros, es decir, no utilizan parámetros colocados anteriormente en la pila. Los ejemplos utilizan la notación UPN.

F#2	..Elige formato decimal de despliegue
3	..Ningún efecto aparente

```
3 ?

3 4 ?

3 4 ? ?

3 ? 4 ?

70 33 * ?

12 12 * 16 - ?

12 * 16 - >: f

12 f ?

12 f 5 f + ?

/r rm >: mod          ..Función módulo

63 17 mod ?

3 5 / ?                    ..Requiere coprocesador o SWFPT

2 π * ?

2 sqrt ?
```

4.3 Estructuras de control

Fifth utiliza una estructura elemental para la ramificación condicional de un programa basado en un parámetro tomado de la pila. Esto permite que el programa tenga la alternativa de tomar dos caminos distintos:



La formulación en Fifth es: ...I... c(.. II ..)q ... III ...

Después de I se ejecuta II en caso que el parámetro tomado por 'c(' sea diferente de '0'. En caso contrario, se ejecuta III. '0' representa el valor lógico falso y todo valor distinto de '0' representa verdadero, normalmente es parámetro lógico resultado de una operación de comparación como '=' o '≤'. En las ramas II y III pueden aparecer otras condiciones de tal forma que puede ser construido un árbol de decisiones.

La estructura c(..)q es una ramificación abierta con distintas salidas. Toda salida de una función condicionada conduce al final del llamado de la función.

Para utilizar una estructura de control como condiciones o estructuras de interacción en sólo una parte de la función, para retornar a un punto común después de efectuar varias condiciones, existe en Fifth una segunda estructura con un par de paréntesis '(' y ')'. La parte de una función colocada entre paréntesis se comporta como una función independiente sin nombre, la cual se ejecuta al iniciar el paréntesis y cuyas salidas conducen al cierre del paréntesis.

En I (c (II)q) III) IV

el programa regresa, incondicionalmente después de la ejecución de II o III a IV. El efecto de la condición limitada es idéntico a la estructura IF.. THEN .. ELSE.

Para el caso particular de esta sencilla condición, el compilador de Fifth permite, por razones de compatibilidad con versiones anteriores la formulación equivalente

I if (II) else (III) IV

donde se puede prescindir de la parte 'else' y el término de control condicional puede seguir al 'if'.

En funciones complicadas y de varias líneas debe mantenerse la claridad del texto del programa considerando la anidación de los paréntesis c(..)q y (..), de forma que el paréntesis que abre esté sobre el que cierra. El paréntesis que abre debe ser el primer caracter del renglón correspondiente. El editor ayuda a través de la tabulación automática del cursor al introducir un '(' al inicio de un renglón y manteniendo el tabulador al confirmar con la tecla Return. La tecla '←' retrocede una posición del tabulador cuando el cursor se encuentra al inicio de una línea tabulada, esto es, bajo la apertura de un paréntesis.

Para generar estructuras de control para lazos o para influir de manera especial sobre una parte del programa, Fifth utiliza un concepto muy sencillo. Estas son realizadas como funciones especiales de Fifth y pueden incluso ser redefinidas por el usuario. Una función 'K:' controla el resto del programa que sigue a su uso.

, K: .. Programa controlado.. >: NN

Las funciones de control terminan con ':' para hacer más claro que influye sobre la parte siguiente del programa. La terminación ':' no requiere de separación con la siguiente palabra y sólo puede aparecer al final de una palabra. El programa influido por una función de control puede bifurcarse y contener otras estructuras de control.

Para limitar el efecto de las funciones de control a una parte de programa, éste debe estar encerrado entre paréntesis '(' y ')':

... I ...(IIa .. K: .. IIb ..) .. III...

Las partes I, IIa, IIb y III son ejecutadas una tras otra, en donde la influencia de 'K:' se limita a IIb.

A las palabras claves de Fifth pertenecen las estructuras de control 'IT:', 'IT-:' y 'REP:' para ciclos fijos y variables. 'IT:' es una función de iteración que toma un parámetro numérico de la pila y ejecuta la parte siguiente del programa ese número de veces, además el respectivo valor de repetición se coloca en la pila para ser utilizado por el programa. Este valor va de 0 a N-1, siendo N el número de iteraciones. El comando 100 IT: ? muestra en pantalla los números 0,1, .. 99. Si se debe limitar el efecto de la iteración a una parte del programa se utiliza:

... (100 IT: ?) ...

En caso que el programa no utilice el valor de la iteración, éste puede ser retirado de la pila con la instrucción 'RM'.

'IT-:' es similar a 'IT:' sólo que el valor de la iteración va en sentido contrario, es decir de N-1 a 0.

'REP:' es una iteración indeterminada. En donde la parte siguiente del programa se repetirá hasta que exista una condición de ruptura con el comando 'BR' (BREAK). Aquí no existe una variable con el valor de la iteración.

'BR' termina también una iteración determinada. En caso que 'BR' sea dependiente a una condición de terminación al inicio de la iteración, se obtiene el equivalente a un WHILE, en cambio, al final de la iteración se tendría una estructura REPEAT .. UNTIL.

Además de las estructuras de iteración, Fifth permite la formulación recursiva de funciones. Debido a que el nombre de la función aparece hasta el final de la misma, existe una instrucción especial '!R' para el llamado recursivo de la función. La definición de un función recursiva debe contener una condición con una salida no recursiva.

Como un elemento en la definición de una nueva función de control o en la implementación de tablas con saltos existe la función indirecta '!', '!' toma de la pila de datos una dirección y ejecuta con ella un salto a un subprograma. Naturalmente se debe estar seguro que esta dirección corresponde realmente a una función.

El siguiente ejemplo ilustra el uso de las estructuras de control en Fifth. La función estándar 'CR' utilizada en dos de los ejemplos, produce un cambio de línea en la pantalla y 'WAIT' espera hasta que sea presionada una tecla. La función 'WAIT' regresa un '-1' si se presionó la barra espaciadora, en cualquier otro caso regresa '0'.

```
c ("verdadero")Q "falso" >:pl          ..Función lógica


4 pl

0 pl

4 7 = pl

5 6 + 11 = pl

10 it: 3*20 + ?                        ..Modifica variable de ciclo

rep: "tecla?" cr wait c(br)q   .. Repite hasta "barra espaciadora"

rep: wait c (br )q "de nuevo!" cr      .. DO WHILE "No barra espaciadora"

dup 0 = c ( rm )q

swap over mod !r >:ggt         ..Algoritmo recursivo ggt

1391 776 ggt?                          ..1391, 776 son (teilerfremd)
```

4.4 Estructuras elementales

Los programas en Fifth pueden utilizar otras estructuras de datos además de los implícitos de la pila de datos. En esta sección se considera a las variables y arreglos como estructuras de datos elementales, así como la posibilidad de utilizar datos como apuntadores de dirección. En secciones siguientes (4.5 y 4.6) se describen las variables locales para funciones y procesos, y en 4.7 se habla sobre la programación orientada a

objetos y la construcción de estructuras complejas (Registros) a partir de estructuras elementales. Fifth permite además, definir nuevas estructuras de datos sin relación con las ya existentes.

Las variables y arreglos, además de servir para formular algoritmos complejos, permiten la comunicación con ciertas direcciones físicas de la computadora y así trabajar estrechamente con la máquina.

Las estructuras de datos, como las variables y los arreglos, deben ser definidos en Fifth antes de su utilización. Estas estructuras no tienen un valor determinado después de su declaración. Para que contengan un valor inicial es necesario definir una función de inicialización y efectuarla.

En general las estructuras de datos son objetos almacenables, que pueden ser leídos o escritos. Fifth diferencía a las operaciones de escritura por la colocación del operador de escritura ">>" o "»" antes del nombre de la estructura de datos.

Las variables son lugares de almacenamiento para datos sencillos. Una variable local 'NN' se declara utilizando la instrucción 'VAR' de la siguiente forma: VAR NN en donde se reserva (pero no se inicializa) automáticamente el lugar de memoria en el área de trabajo. El contenido de una variable 'NN' puede ser leído y colocado en la pila de datos o sobreescrito con el valor superior de la pila (TOS) el cual será retirado. Las instrucciones correspondientes son:

```
NN          .. Lee contenido de 'NN' y lo coloca en la pila

>>NN        .. Escribe TOS en 'NN'
```

De forma similar son declarados y manejados los arreglos de datos lineales y de dos dimensiones. En su declaración debe especificarse el número de elementos, el espacio en memoria es reservado pero no inicializado. La declaración del arreglo lineal de longitud 33 sería:

ARRAY 33 NN

la de una matriz de 10 x 33 elementos:

ARRAY 10,33 KK.

Los elementos de estos arreglos pueden ser leídos y escritos como variables, en donde el índice correspondiente al elemento es tomado como parámetro de la pila. El índice se coloca frente al nombre del arreglo dentro de corchetes. En caso que el corchete no contenga el índice, éste será tomado como un parámetro extra de la pila. Los índices inician en 0. Las instrucciones de lectura y escritura del elemento 17 del arreglo NN y 3, 12 de KK serían:

```
NN [17]          ..Lectura, podría ser 17 NN []

KK  [3 12]       ..o: KK[3,12], 3 12 KK[]I, 3 KK[12]

>> NN[17]        ..Escritura, podría ser 17 >> NN[]

>> KK [3 12]     ..o: >> KK[3,12], 3 12 >> K[]
```

Arreglos lineares y de dos dimensiones con bytes son declarados de manera análoga con la instrucción 'ARRAY:B' y de igual manera manejados:

ARRAY.B 80 NN

En la implementación para PC se encuentran disponibles también variables y arreglos de punto flotante, los cuales son creados con 'VAR.F' y 'ARRAY.F' respectivamente y manejados de manera normal.

Para inicializar todos los elementos de un arreglo a un valor no es necesario realizar un lazo. En su lugar se puede formular la asignación de una constante al arreglo, en donde se escribe el nombre del arreglo sin corchetes. La expresión: $20 >> NN inicializa un arreglo de bytes al valor $20 (espacio). En general se considera en Fifth a los términos e instrucciones en donde aparecen los nombres de arreglos sin corchetes como operaciones

vectoriales. En este contexto los arreglos son también estructuras elementales en Fifth y el lenguaje puede manejar fácilmente las operaciones vectoriales características de los procesadores digitales de señal. Ejemplos de la sintaxis de operaciones vectoriales sobre los arreglos lineares U,V y W del mismo tamaño, que no están implementados en la PC, serían:

```
U 4 * V + >> W          .. Suma de vectores

U V * >> W              ..Multiplicación de componentes.
```

En la implementación vectorial existe el operador 'Σ' que calcula la suma de los componentes de un vector. El producto escalar de estos dos vectores se calcula con:

```
U V * Σ
```

La combinación '* Σ' puede ser utilizada para encontrar el producto de una matriz por un vector. El compilador de Fifth elimina el trabajo de realizar las iteraciones correspondientes y con ello se tiene la posibilidad de producir un código optimizado.

Los arreglos pueden ser almacenados y leídos con comandos sencillos en disco (ver cap. 7.4). Para almacenar y leer el arreglo NN de un disco con el nombre UVW.DAT se utiliza:

```
SDF NN UVW              ..almacenar

LDF NN UVW              ..leer
```

Dado que sólo pueden ser almacenados datos sencillos en la pila como parámetros de funciones y no grandes estructuras de datos, es necesario poder transferir sus apuntadores, es decir, la dirección de estos datos. Si 'NN' es una estructura de datos o bien una función, su dirección sería conocida como '%NN'. El operador '%' permite también un desplazamiento (offset) constante a esta dirección en la forma: %3 NN ?

El acceso directo a la memoria se realiza con la instrucción 'MM', la cual toma una dirección como parámetro de la pila de datos. Las operaciones de escritura y lectura de un dato estándar en la dirección $1233 son:

```
$1233 MM            ..lectura

$1233 >> MM         ..escritura
```

De manera similar trabajan las operaciones 'MM.B' para bytes, 'MM.X' para bytes en las direcciones de E/S y 'MM.F' para números de punto flotante. Estas instrucciones deben ser utilizadas con cuidado, ya que un error en una dirección puede causar la destrucción de información importante para la computadora y con ello bloquearla.

Mientras que sólo existen arreglos globales como estructuras predefinidas en Fifth que pueden ser utilizadas eficientemente en cualquier microprocesador, con 'MM', 'MM.B' y 'MM.F' pueden ser definidas funciones de acceso complejas o estructuras de datos dinámicas. Las estructuras de registro discutidas en el Cap. 4.7 hacen uso implícito de 'MM'.

Los elementos de la pila de datos pueden ser trabajados como un arreglo lineal haciendo uso de la palabra especial 'STK', cuyo elemento 0 es el TOS:

```
STK[0]      ..o 0 STK[], 0 STK es equivalente a DUP

STK[1]      ..es equivalente a OVER
```

Las variables (al igual que otras estructuras de datos) aparecen en el texto del programa como instrucciones, en donde se encuentran reunidas dos funciones de acceso bajo un concepto superior. El concepto es seleccionado por '>>', formando las funciones de lectura y escritura. Fifth permite también que dos subprogramas, en donde uno produce un valor al igual que una lectura y el otro requiere de un valor como en una escritura, sean unidos bajo un mismo nombre y sean diferenciados por el operador '>>'. Un ejemplo de este tipo sería la seudovariable SIO, con la cual es posible comunicarse a

través del puerto COM1 de la computadora. La operación de recepción toma un byte del puerto serie y lo deposita como resultado en la pila de datos. La operación de transmisión toma el byte como parámetro de la pila.

```
SIO          ..recepción

>> SIO       ..transmisión
```

De esta forma es muy sencillo y cómodo generar funciones complejas de comunicación por el puerto serie.

Otras dos seudovariables son 'USP' y 'FADR'. 'USP' lee el apuntador lógico de la pila de datos, es decir el número de parámetros que contiene. Con la siguiente secuencia se eliminarán 8 datos de la pila: USP 8 - >> USP.

Con la ayuda de la instrucción 'USP ?' puede ser comprobada la correcta utilización de la pila. Con 'USP' podría ser definida una función de prueba que muestre todos los elementos de la pila.

```
usp it: stk[] ? >:Stack?
```

'FADR' lee y retira la dirección de regreso de la función llamada de la pila del programa. Por el contrario '>> FADR' transfiere de la pila de datos a la del programa. 'FADR' es muy útil en la definición de nuevas estructuras de control y puede ser utilizada también para almacenar datos en la pila del programa.

Ejemplo:

```
VAR aa, bb, cc              ..declaración de 3 variables

7 >> aa 10 >> bb 5 >> cc    ..valores iniciales

aa ?

bb aa - cc * ?              ..Notación UPN
```

```
Array 256 t

0 >> t                          ..valor inicial

t [10] ?

128 it: dup >> t[]

t[10] ?

T[12] bb * >> aa

aa ?
```

El siguiente programa ordena los 128 elementos del arreglo t por tamaño. El algoritmo utilizado es el de la burbuja (Bubble sort). El valor de la variable de iteración se almacena en 'i' para hacer más claro el proceso.

```
Var i

128 it-: it: >> i               ..iteración anidada

t[i] t[i 1 +] <= c( )q          ..en ese caso no hacer nada

t[i] t[i 1 +] >> t[i] >> t[i 1 +]  ..intercambiar valores

>:sort
```

La siguiente función de terminal ilustra la programación con el puerto serie. En el ciclo REP: se monitorea el estado de recepción del teclado y puerto serie con las funciones 'RCNT' y 'KEY' (ver Cap. 7). El código de la tecla es enviado por el puerto y los datos recibidos se muestran en pantalla con la función 'OUT'. Dado que el ciclo REP: no contiene ninguna instrucción 'BR', el programa sólo puede ser interrumpido presionando Ctrl-Brk. En una aplicación práctica de este programa, el puerto serie debe ser configurado para trabajar con interrupciones de manera que no existan pérdidas en la recepción de datos:

```
rep: rcnt c(sio out)q key c(>> sio)q >:terminal
```

4.5 Objetos y variables locales

Los objetos son funciones de Fifth, cuya definición especifica el número de parámetros que toma de la pila, así como la cantidad de datos producidos como resultado. La forma de esta especificación es: OBJ n m en donde 'n' es el número de parámetros y 'm' el de resultados. n y m pueden tener un valor máximo de 10. El valor de n y m pueden ser '-1', que indica un número variable de parámetros o resultados. En este caso, el número real de parámetros o resultados estaría en TOS, ya sea antes o después del llamado de la función. Los objetos con parámetros y resultados variables no están implementados para todos los tipos de microprocesadores.

Los objetos cuentan con otras características que los distinguen de las funciones declaradas normalmente. Por ejemplo, la definición de un objeto 'NN' con 3 parámetros y 2 resultados tiene la forma:

```
OBJ 3 2

..

.. programa

..

>: NN
```

y la función 'NN' puede ser llamada normalmente después mencionando su nombre en el texto del programa. No importa cuantos datos coloque el programa 'NN' en la pila, sólo el valor especificado de resultados serán transferidos al programa que efectuó el llamado y estos son los que se encuentren en la parte superior de la pila al finalizar 'NN'. Los valores restantes son retirados automáticamente de la pila. El programa 'NN' podría, por ejemplo, colocar el par de resultados sin necesidad de retirar los parámetros.

Además, para una función declarada como objeto se coloca el inicio de la pila de tal forma que, al iniciar sólo contiene el número especificado de parámetros. Al inicio del llamado de 'NN' sólo hay 3 parámetros en la pila. A los elementos de esta pila pueden dárseles nombres iniciando con el inferior, esto es el primer parámetro y servir como variables locales del objeto. El ordenamiento de los nombres para los elementos de la pila sigue a la especificación del objeto en la forma:

OBJ n m , X1, X2, X3,.....Xk

Los k elementos inferiores son nombrados secuencialmente con 'X1'...'Xk'. Normalmente se elige k=n (número de parámetros) y las variables locales son ocupadas desde el inicio con los parámetros de la función. Xn nombra el último parámetro de la pila, y la secuencia de nombres corresponde exactamente a la forma en la que los parámetros son creados por el programa. Si k>n, entonces Xn+1 ...Xk son variables locales extra, para las que se reserva espacio en la pila pero sin valor inicial. Ellas deben ser inicializadas dentro de la función objeto por medio de una operación de escritura. Las variables locales son escritas y leídas como las globales, por ejemplo:

```
X1          ..El valor de X1 se hace TOS

>> X1       ..TOS se escribe en X1
```

La utilización de objetos y variables locales permite formular programas más sencillos y fáciles de leer. En muchos casos es posible colocar los resultados secuencialmente como términos en las variables locales. Las variables locales son retiradas automáticamente de la pila antes de ser transferidos los resultados. Por lo general sólo se escriben algunas definiciones de función muy pequeñas o críticas en el tiempo sin especificación de objeto y variables locales.

Los objetos juegan un papel central, cuando se utiliza Fifth en la programación de sistemas con varios microprocesadores. Fifth permite programar sistemas con varios microprocesadores que se comunican por un puerto serie. También aquí las funciones de Fifth son compiladas para el microprocesador especificado por la instrucción MODE y

pueden, generalmente, sólo ser llamadas por éste. Por ejemplo, la PC tiene el Modo 1 y con el puerto serie está unido un subsistema en el Modo 2. Después de la instrucción MODE 2, el compilador (una versión extendida del IPU para programar el subsistema) transforma la definición de una función en código máquina para el microprocesador del subsistema y envía este código a través de un puerto, de forma que pueda ser ejecutada allá. Las funciones declaradas como objetos para un microprocesador pueden ser llamados también por otro microprocesador. El llamado es transformado de tal forma que el microprocesador envía a través de un protocolo los parámetros requeridos por la función objeto. Es decir, el llamado de objetos realiza automáticamente la comunicación entre varios microprocesadores. En un supuesto caso, los objetos definidos en el modo 2 pueden ser ejecutados desde un programa definido en el modo 1 para la PC. Una utilización muy común en sistemas de multiprocesadores es declarar las funciones de entrada y salida de la PC como objetos. Estos pueden ser utilizados después por otros microprocesadores.

Varios microprocesadores en un sistema multiprocesador pueden utilizar distintos tipos de datos. Para la comunicación sobre llamadas a objetos es necesario realizar una conversión. Ejemplo:

El siguiente ejemplo muestra el ejemplo de la limpieza automática de la pila en los objetos y da una nueva definición de 'ggt'.

```
5 7 >:tf

obj 0 1 5 7 >:to                ..mismo programa en 'tf' y 'to'

list

1 tf ? ?

1 to ? ?                        ..'to' deja sólo 7 como resultado

obj 2 1, x,y y 0 = c( x )q y x y / r rm !r >: ggt
```

```
1391 776    ggt    ?
```

Las funciones de Fifth pueden tener varios datos como resultado. En las siguientes dos definiciones en notación algebraica pueden ser realizadas operaciones sobre parejas de datos. Se trata de las operaciones complejas de suma y multiplicación.

```
alg

obj 4 2, re1, im1, re2, im2

re1 + re2                    ..Parte real del resultado

im1 + im2                    ..Parte imaginaria

>:c+

obj 4 2, re1, im1, re2, im2

re1 * re2 - im1 * im2              ..Parte real

re1 * im2 + im1 * re2              ..Parte imaginaria

>:C*
```

Para experimentar en una configuración de multiprocesadores, pueden ser utilizados los programas ejemplo PC.COM y PC.TXT que permiten unir al IPU-PC con una segunda PC a través del puerto serie y tener a ésta como 'Sistema Destino'. En la segunda PC se ejecuta PC.COM para crea un ambiente ejecutable en donde pueden ser corridas funciones de Fifth. Al comenzar se inicializa el puerto COM1 y la PC entra en una función de comunicación, en donde se toma el código del IPU-PC y también se puede ejecutar. En la IPU-PC debe cargarse y compilarse el archivo PC.TXT el cual extiende al compilador de manera que los programas en el modo 2 se producen y envían a la otra PC. Los programas y comandos del modo 2 corren después automáticamente en la otra PC. Se puede verificar este efecto con una instrucción del tipo: 100 it: ?, pero también

pueden definirse objetos en el modo 2 y ejecutarlos desde un programa en modo 1 y así, por ejemplo, producir una gráfica en la pantalla de la otra computadora.

4.6 Procesos paralelos

Por su completa sencillez Fifth es uno de los pocos lenguajes de programación con capacidad de trabajar con procesos paralelos así como con microprocesadores que trabajan en paralelo. Los procesos paralelos son un instrumento muy importante en la programación en tiempo real y son parte importante de Fifth en aplicaciones con otros sistemas. Pero los procesos paralelos pueden ser utilizados sólo en la PC y con ello representan una etapa superior de la programación estructurada. Los procesos paralelos son discutidos con detenimiento en el capítulo 8. Esta sección contiene sólo un pequeño avance.

Los procesos, como se utilizan en Fifth, son programas que se ejecutan al mismo tiempo con su propia pila de datos y direcciones de regreso, entre los cuales puede ser conmutado el microprocesador. Son las correspondientes corutinas de lenguajes de programación como Módula II. Sólo las pilas y un tipo de variable local, todavía por discutir, son distintos en los procesos de Fifth. Por el contrario, utilizan la misma memoria para los programas y accesan a los mismos datos globales, de forma que distintos procesos se pueden comunicar también entre sí a través de estructuras de datos globales.

Los procesos tienen distintos niveles de prioridad y son declarados con la función de control 'p#', al cual sigue un símbolo con el grado de prioridad deseado. En la implementación para la PC existen dos prioridades, 'L' (baja, 'low') y 'H' (alta, 'high'). La creación de un proceso nuevo con su propia pila de programas y datos se realiza, para la prioridad inferior, llamando a la función de control 'p#l' y para la prioridad superior con 'p#h'. Esta función antecede al bloque de programa que debe convertirse en proceso:

```
p#l
```

..

.. Programa

..

>:NN

Esta parte del programa termina como de costumbre con el símbolo de definición '>:' o con el cierre de un paréntesis en caso de una definición integrada. La instrucción 'p#l' hace que el siguiente programa inicie con una nueva pila.

'p#' tiene una variante en donde el nivel de prioridad, así como un número seleccionable de parámetros son tomados del proceso creador. En este caso se agrega un dato numérico que indica la cantidad de parámetros que se transfiere al 'proceso hijo', como 'p#5'. Por ejemplo, con un programa en la forma: 10 it: p#1 ... en cada ciclo de la iteración se crea un nuevo proceso, todos ejecutan el mismo programa pero inician con valores distintos de TOS; el número de iteración tomado del proceso creador. El creador del proceso hijo puede, por medio de una función de control especial, esperar su terminación. (Ver capítulo 8.3).

La conmutación del microprocesador entre los distintos procesos se efectúa ya sea a través de instrucciones explícitas en el programa o en base a interrupciones por hardware. Todos los procesos inicializados por medio de 'p#' están listos para ser ejecutados, sin embargo, en un tiempo específico sólo se efectúa un proceso mientras los restantes se encuentran en una lista de espera hasta que son llamados por el microprocesador. La conmutación de un proceso a otro de la misma prioridad en la lista se efectúa por medio de la instrucción 'SWITCH'.

Con esto puede explicarse una aplicación importante de los procesos paralelos. Cuando un programa efectúa procesos de entrada/salida y debe esperar una condición específica, es posible agregar en el ciclo de espera el comando 'SWITCH' para brindar a otros procesos la posibilidad de utilizar el tiempo de espera.

Si el microprocesador es conmutado continuamente entre distintos procesos parecería que los programas se ejecutan al mismo tiempo.

Por ejemplo, existe la posibilidad de conmutar al microprocesador entre los procesos de más baja prioridad con la señal de reloj del sistema de la PC y así cada proceso tiene una parte igual del tiempo de cálculo. Normalmente los programas que se ejecutan simultáneamente no son independientes entre sí, sino que deben comunicarse unos con otros, por lo que un proceso debe esperar la entrega de datos de otro proceso. Para el acoplamiento de este tipo de procesos existe una estructura de control especial en Fifth: Puntos de sincronización y líneas de espera (Ver Cap. 8).

Dado que los procesos de Fifth utilizan un mismo código de programa y que existe la posibilidad que una función de Fifth se ejecute por medio de varios procesos, se requiere que las variables auxiliares para cada proceso sean distintas. Esto afecta al lugar de almacenamiento de la pila de datos y también a las variables locales de los objetos. Por lo tanto existe un tipo de variable que se declara con 'VAR.P' (Variable de Proceso) que se utiliza igual que las variables discutidas en 4.4 después de la declaración:

```
VAR.P A,B,C
```

A, B y C pueden ser utilizadas como cualquier variable. Sin embargo, el lugar de almacenamiento en memoria es distinto entre los procesos. Por lo tanto, las variables de proceso no pueden ser utilizadas para intercambiar datos. Para lo cual después de cambiar el microprocesador a otro proceso lo valores de estas variables permanecen inalterados, aún cuando hayan sido sobreescritos por otro proceso.

Existe un límite en el número de variables-P que pueden utilizarse y éstas son creadas después de cada declaración, es decir, después de la declaración:

```
VAR.P X,Y,Z
```

X representa la misma variable-P que A etc. Si se desea que X,Y y Z sean variables-P distintas, puede utilizarse un índice explícito que haga que X esté en seguida de C (Ver 6.1):

```
VAR.P %2 C X,Y,Z
```

Cuando se ejecutan varios procesos paralelos en la PC es posible realizar los despliegues en pantalla de tal forma que cada uno tenga un área de las 16 posibles. La separación se efectúa con la función 'SSCRN' (Ver Cap. 7.2). Con la instrucción 'n WND' se elige a la ventana n para los siguientes desplegados. 'WND' utiliza una variable-P reservada que sirve como apuntador a la ventana utilizada por el proceso.

4.7 Programación orientada a objetos

En el Cap. 4.1 se explicó que los conceptos de un programa en Fifth pueden ser organizados en clases. Una clase tiene un nombre y cada elemento es seleccionado por medio de otra palabra como se manejó en las declaraciones de 'VAR' y 'ARRAY'.

Antes de poder definir a los elementos de una clase es necesario definir la clase. Esto se realiza con la palabra 'CLASS' y el nombre de la nueva clase. Un ejemplo sería:

```
CLASS XPL
```

La definición de los elementos de la clase XPL, digamos las variables U,V y una función F sería:

```
VAR XPL U, XPL V        ó

...     >:XPL F
```

Estos elementos son utilizados después como 'XPL U', ' >> XPL U' ó 'XPL F'. En la lista de símbolos mostrada por el comando 'LIST' aparece sólo el nombre 'XPL' sin sus elementos. Los elementos pueden ser vistos utilizando el comando 'LIST XPL'. El espacio entre el nombre de la clase y el del selector puede omitirse si se eligen los que

inician con punto '.'. Normalmente se reúnen los nombres de funciones y estructuras de datos con características comunes, por ejemplo todas las definiciones de un módulo, y se elige un nombre para la clase que representa esas características. Los nombres de las clases no pueden ser a su vez elemento de otra.

Una nueva clase puede tomar la definición de una clase ya existente para especializarla o simplemente modificarla. En la declaración aparece el nombre de la clase anterior entre paréntesis:

```
Class (General) Especial
```

La clase 'Especial' toma todas las definiciones existentes de la clase 'General', incluso aquellas que se declaren más adelante'. Pero si se define un concepto con el mismo nombre en 'Especial' y en 'General', entonces el concepto que se utiliza es el definido en 'Especial', siempre y cuando pueda ser utilizado en el modo de compilación actual.

Una aplicación importante de las clases consiste en reunir las funciones de acceso a un tipo determinado de estructura de datos y sus operaciones correspondientes. Estas funciones necesitan como parámetro un apuntador, que marque la posición de la estructura de datos que se manipula con las funciones se conoce como objeto de esa clase. El apuntador a un objeto puede ser utilizado como parámetros o resultado de una función de Fifth y ser almacenado en una variable. Si 'a' es un elemento de la clase XYZ y U es una variables que contiene el apuntador a un objeto de esta clase, se utilizaría de la siguiente forma:

```
U XYZ .a
```

Fifth utiliza una sintaxis muy sencilla para que una variable U contenga un apuntador a un objeto de la clase XYZ. Esta sería:

```
VAR (XYZ) U
```

Después de esta declaración, cada vez que se mencione U se cumplirá la operación de lectura para la variable apuntada y además se puede utilizar automáticamente un selector de la clase XYZ. El uso de '.a' se simplifica a

U.a

En caso que aparezca U sin selector de la clase, sólo se compilará la operación de lectura de U. Como en cualquier otro caso, U puede obtener un valor de la pila.

Los selectores de la clase XYZ son examinados después de indicada una variable, pero sólo cuando esta es del mismo 'tipo de clase'. De esta forma se crea, además del orden en las clases, una protección contra un llamado equivocado de un elemento de una clase en un contexto erróneo.

De la misma manera pueden unirse en las clases las variables P y locales. Las variables locales, que pueden ser utilizadas únicamente por el tipo de datos estándar de la implementación, pueden contener también 'tipos de clase'.

4.8 Programas modulares y compilación

Los programas de Fifth están estructurados en definiciones de funciones en forma jerárquica. Esto no impide, que grandes partes de programas puedan ser independientes entre sí, donde una parte no utilice funciones de la otra y no sea hasta más tarde cuando otro programa utilice las funciones de ambos. El compilador de Fifth no genera archivos objeto independientes a partir de estos módulos, si no que compila como sistema interactivo código del programa ejecutable. La combinación de módulos puede ser llevada a cabo sólo en el nivel de texto con la instrucción 'INCLUDE'. La lista de símbolos de los nombres definidos en un programa hace invisible a una construcción modular del programa de este tipo.

Los nombres de las variables locales de un objeto son válidas sólo en la definición actual, donde substituyen a una posible definición anterior. Los nombres restantes en un programa, así como los nombres de funciones, variables, arreglos y variables-P son

solamente válidos para el mismo nivel lexical, esto es, las funciones no son anidadas como en Pascal, en donde dentro de un procedimiento pueden ser definidas nuevas funciones auxiliares y estructuras de datos. Para un nombre definido en este nivel es válida la última definición del texto del programa actual. La definición de objetos y las instrucciones para el compilador son independientes y por tanto siempre válidas. El 'diccionario' del compilador en donde se reúnen las anteriores y las nuevas definiciones de conceptos, consta también de listas traslapadas ordenadas según el modo de compilación.

En un programa muy grande, formado quizá por varios archivos de texto, son definidas cientos de palabras que serán difíciles de observar. Es por tanto deseable, poder 'olvidar' los nombres de funciones auxiliares y estructuras de datos que sólo se utilizan para definir una o más funciones y después ya no son necesarias. Es decir, en una parte del programa se construye una interfase modular de donde ciertas funciones se extraen para ser utilizadas en el resto del programa, mientras otras sólo se utilizan temporalmente y por tanto ya no son necesarias.

El olvido de los símbolos definidos se realiza con la función del compilador 'H<', a la cual seguiría una lista con los nombres de los módulos que se conservan:

H< N1, N2, N3 ...

'H<' retira todos los símbolos a excepción de los contenidos en la lista. Los símbolos retirados que tenían anteriormente otro significado lo retoman. La última definición efectuada antes de la instrucción 'H<' permanece aún sin encontrarse en la lista. El símbolo '%' puede estar al final de la lista de interfase para identificar el inicio de las definiciones del usuario. La instrucción

H< %

borra todas las definiciones de la tabla de símbolos a excepción de la última. Después de los datos

... Función ...

>:g

... Función auxiliar 1 ...

>:a1

... Función auxiliar 2 ...

>:a2

... Función ...

>:a

H< g, a

permanecen sólo 'g' y 'a'. Las funciones 'a1' y 'a2' son retiradas y en su caso retoman su significado anterior, sin que esto afecte la ejecución de 'a'. Las definiciones sólo podrán ser utilizadas por 'a' y posteriormente eliminadas y no serán listadas con el comando 'LIST'.

Las posibilidades de controlar la lista de símbolos con 'H<' y el uso de clases son útiles en programas extensos. También existe la compilación condicionada para obtener variantes del mismo texto del programa. La compilación condicionada se controla por medio de símbolos que normalmente se declaran como constantes al inicio de un programa (ver 6.1), en la forma

const 1 version_1

La parte del programa condicionada se coloca entre las palabras 'CC' y 'CE', en donde el parámetro de control se coloca como dirección constante después de 'CC':

CC %version_1

... texto condicionado ...

CE

En caso que el parámetro de control sea una dirección 0 ó que no se coloque, el texto se tomará como un comentario. El uso de 'CC' sin parámetros puede ser usado para 'eliminar' partes del programa de forma sencilla. 'CC' y 'CE' se utilizan como parejas de paréntesis.

5. El editor

5.1 Generalidades

Toda la información es introducida al sistema a través de un editor de texto y cuando el sistema regresa de efectuar alguna función, vuelve al modo de edición. El editor de Fifth utiliza un espacio de memoria de 65K para almacenar el texto. El texto de un programa de Fifth es ilimitado con la inclusión de archivos.

Dentro del editor de texto se muestra una línea de estado y dos líneas de ayuda. En la línea de estado se muestra X el área de texto editado, que se explica en la siguiente sección. Y es el número de renglón en esa área. ZPOS es la posición (hexadecimal) de la dirección de inicio del cursor del texto en el segmento de almacenamiento del texto, TLEN es la longitud total del texto. NOMBRE es el nombre del archivo utilizado por una operación en el disco. Finalmente N es el modo actual de compilación (ver 6.2).

Level X Zeile Y ZPOS/TLEN Text NAME Mode N

El uso del editor se realiza con teclas de movimiento del cursor (flechas) y teclas de funciones. Los renglones de ayuda muestran las teclas de función utilizables. Presionando las teclas Alt o Ctrl se muestran sus funciones correspondientes. Un conjunto de la tecla Alt tiene símbolos especiales que son permitidos en el texto. Alt-F3..F6 tienen símbolos de líneas que permiten el uso de gráficos sencillos. Las líneas de estado y ayuda pueden ser retiradas de la pantalla con Alt-F10 y así poder utilizar la pantalla completa para la edición, presionando Alt-F10 aparecerán nuevamente.

El editor trabaja en un modo de autoinserción, para facilitar la edición de textos con formato de programa. Al colocar un paréntesis en un renglón se coloca un tabulador en las siguientes líneas del programa.

El editor permite hasta 132 caracteres por línea. Debido a que en la mayoría de los modos de video sólo se pueden observar 80 caracteres, el editor mueve el área de texto hacia la derecha o izquierda dependiendo de la posición del cursor. Los símbolos que no son visibles en la pantalla son indicados.

El sistema Fifth puede ser configurado para diversos modos de video por medio del programa KONF. En el modo EGA se muestran 43 renglones. También existe un manejador de video que puede mostrar 80 renglones en monitor VGA. Los variados modos de video son una ventaja para el programador que puede tener mejor visión del programa. Diversos modos gráficos, donde también se puede editar, se seleccionan con GR.CGA, GR.EGA (43 renglones), GR.H (Hércules, 43 renglones) y GR.VGA (60 renglones). En estos modos se cuenta con instrucciones para colocar puntos y líneas en la pantalla. El modo de texto puede restablecerse con el comando 'TX'. El manejador de página completa existe también en el modo gráfico. En todos los modos de video se muestran 80 caracteres por línea, a excepción del modo Hércules con 90 caracteres.

5.2 Funciones básicas

Las teclas del cursos tienen las funciones habituales. Otras funciones básicas son las siguientes:

Tab Mueve el cursor un tabulador

Home Coloca el cursor al inicio del renglón

End Coloca el cursor al final del renglón

PgDn Pasa una pantalla hacia adelante

PgUp Pasa una pantalla hacia atrás

Ctrl-Home Abre la ventana de edición a toda la pantalla.

Ctrl-End Coloca el cursor al final del texto editado

Return	Coloca el cursor al inicio del siguiente renglón y coloca un renglón vacío.
Ctrl-Return	Coloca un 'CR/LF' en la posición actual del cursor.
Ins	Inserta un espacio en la posición del cursor.
BackSpace	Elimina el caracter anterior al cursor.
Del	Elimina el caracter sobre el que se encuentra el cursor.
F8 ó Ctrl-Y	Elimina el renglón actual.

Después de editar un texto se puede trabajar en tres maneras distintas:

F1	Después de ejecutarlo se edita un nuevo texto.
F2	Después de ejecutarlo el texto se conserva.
Alt-F1	Los resultados de la compilación son enviados a la impresora.

La segunda forma permite modificar en repetidas ocasiones un texto y observar el resultado.

El texto se conserva también cuando se detecta un error. En un 'Error de sintaxis' se coloca el cursor al inicio de la palabra equivocada.

La tercer forma sirve para imprimir información sobre la compilación como símbolos y sus respectivas direcciones.

5.3 La pila de texto

El editor tiene una estructura especial diseñado especialmente para el lenguaje Fifth y la programación interactiva. La memoria contiene no sólo un texto, si no una serie de partes de texto separadas en lo que se conoce como pila de texto. El texto situado en la parte inferior (dirección 1) sirve para almacenar el texto fuente de las definiciones de funciones ya compiladas y la parte superior del texto (a partir de la dirección TLOC

hasta TEND) contiene los datos que siguen en ser procesados. Abajo pueden existir (a partir de la dirección BLOC) partes del texto sin procesar.

…..	)*….)*…..	)*…..	)….
1	BLOC	TLOC	TEND
Definiciones anteriores	Sin procesar	Texto actual	Final

El editor sirve normalmente para trabajar el texto actual, pero se puede seleccionar otra parte del texto. En la línea de estado se encuentra el valor Level-X que indica en que sección del texto se encuentra el editor. En la parte superior X=1, hacia abajo X=2 y en el área del texto fuente bajo BLOC X=0.

Conservar todas las definiciones en un texto fuente es una función automática del editor. Existen dos tipos de datos, los comandos ejecutados inmediatamente y las definiciones de nuevas palabras. En los primeros después de la ejecución con la tecla F1 es retirado el texto. En los últimos el texto se copia después de su compilación en la parte del texto, de forma que todo el texto fuente de todas las definiciones es reunido. La definición de funciones independientes deben ser editadas completamente antes de su procesamiento, es decir, no pueden trabajarse por medio de F1 renglón por renglón. Es recomendable que en lugar de compilar un texto fuente completo, éste se edite función por función. Al final el texto se encuentra completo gracias a la conservación automática de funciones.

Algunas funciones de control trabajan sobre el texto fuente:

F5 Coloca el cursor al inicio de la última definición introducida y permite su edición.

F6 Regresa al espacio del texto actual.

F7 Retira el texto a partir del renglón actual y lo copia al área actual de trabajo.

La última función de control sirve para corregir grandes definiciones y repetirlas.

Todas las funciones de edición disponibles para el texto actual, pueden ser utilizadas en el texto fuente tras presionar F5. Las funciones propias del texto actual para compilación y ejecución aquí no están disponibles.

La posibilidad de colocar varias partes de texto en la pila sirve para suspender la edición de un texto largo y ejecutar un comando como podría ser 'LIST' o ejecutar una función. También se tiene un modo de edición especial para comandos. La pila de datos puede ser utilizada sistemáticamente para desarrollar un programa Top-Down, editando primero las funciones principales del programa y referenciarlas posteriormente en nuevas funciones. El compilador trabaja primero sobre la parte superior del texto, que contiene las definiciones de funciones, y posteriormente interpreta la función principal.

Para abrir una nueva área de texto en el actual y regresar después de su ejecución al texto actual se cuenta con dos funciones:

F3 Conserva el texto actual y abre un nuevo texto.

F1 Con un texto vacío regresa al texto anterior.

La función F3 no funciona en un texto vacío.

Los textos colocados anteriormente en la pila no pueden ser trabajados por el compilador, pero pueden ser pasados al texto actual y ahí editados.

F4 Cambia el editor a la siguiente sección de texto.

F6 Regresa al texto actual.

Los textos encontrados con la tecla F4 pueden ser editados como el texto actual. Pero las funciones de compilación y ejecución no están disponibles.

Finalmente es posible dividir el texto a partir de algún renglón deseado en diferentes secciones y enviar una a la pila de texto. De manera similar se pueden unir las dos partes superiores de la pila de texto en una.

F9 Parte el texto actual y coloca el resto del texto en la pila.

F9 realiza las siguientes modificaciones en el texto (C es la posición del cursor).

…)*	… Sección 1 … C …	Sección 2	…) …
TLOC			TEND
…)*	… Sección 2)*	Sección 1	… C)
	TLOC		TEND

F9 al inicio del texto actual es equivalente a F3 y para un texto vacío funciona como F1. F9 permite trabajar un gran programa parte por parte, colocando a gusto en la pila partes del texto.

5.4 Otras funciones del editor

Otras funciones del editor sirve para borrar, copiar, mover bloques de texto, buscar y reemplazar.

Alt-F8 Borra el texto actual a partir de la posición del cursor.

A diferencia de la función F8, que sólo borra un renglón, con Alt-F8 se puede borrar una sección completa de texto. Para efectuar esta operación se pide la confirmación. Para borrar un bloque de texto que no llegue hasta el final del texto, se puede colocar el resto con F9 en la pila, borrar la sección que no se desea y con F9 recuperar la sección de la pila.

F10 Copia el renglón al final del texto.

Utilizando repetidamente la tecla F10 se puede copiar una sección al final del texto. F10 es también útil cuando se deben repetir en varias ocasiones un mismo renglón.

Otros comandos para manipulación de bloques y renglones se muestran al presionar las teclas Ctrl-K.

Se muestran todas las opciones disponibles que se pueden seleccionar al presionar otra tecla. Los comandos con bloques sirven para copiar (C), mover (V), o borrar (Y). Antes debe ser definido el inicio del bloques (B) y su final (K). Los renglones incluidos dentro del bloque son señalados en la orilla derecha de la pantalla con el símbolo '<<'. Las opciones C, V y Y pueden ser utilizadas cuando se hayan definido un inicio y final del bloque. El inicio y final de un bloque debe encontrarse en la misma sección del texto pero un bloque puede copiarse o moverse hacia otra sección de la pila.

Tras utilizar una función de borrado con F8, Alt-F8 y Ctrl-K-Y se brinda la posibilidad de recuperar el texto con el uso de Ctrl-U.

La función de búsqueda del editor de Fifth sirve para encontrar pequeños textos.

Ctrl-Q Permite la búsqueda de un texto.

El dato . Se puede editar el texto o borrar la línea completa con F8 y se termina con la teclas Enter, '→', Ctrl-PgUp o Ctrl-PgDn. Ctrl-PgUp y Ctrl-PgDn efectúan inmediatamente la búsqueda.

El proceso de búsqueda se efectúa de acuerdo a las convenciones de Fifth. No hay distinción entre minúsculas y mayúsculas y ciertos símbolos sólo se pueden encontrar al inicio o final de una palabra (ver Cap. 6.1). El número de espacios entre los símbolos no se toma en cuenta. Una búsqueda puede realizarse hacia arriba o abajo de la posición actual del cursor. Las palabras buscadas se deben encontrar en el orden indicado dentro de un sólo renglón.

Ctrl-PgDn Búsqueda en el texto actual.

Ctrl-L Búsqueda a partir de la posición actual del texto.

Ctrl-PgUp Búsqueda hacia atrás.

Los comentarios no son tomados en cuenta durante la búsqueda. Una búsqueda en un programa grande puede requerir incluso varios segundos, si se desea interrumpir el proceso se puede hacer con Ctrl-Brk como cualquier programa de Fifth. El cursor se coloca en el renglón que se había alcanzado.

Para reemplazar una palabra por otra en el texto o una frase se utiliza la función.

Ctrl-→ Modificar un texto.

Los espacios iniciales y finales de la frase no se toman en cuenta al hacer la substitución, pero los contenidos en el texto, así como las mayúsculas y minúsculas se conservan.

Adicionalmente, las funciones normales de edición están disponibles:

Up Flecha hacia arriba, sube un renglón

Down Flecha hacia abajo, siguiente renglón

PgUp Una pantalla hacia atrás

PgDn Una pantalla hacia adelante

Ctrl-Home Inicio del texto

Ctrl-End Fin del texto

6. Detalles técnicos de programación

6.1 Números y nombres de funciones.

%, CONST, LIST, ULIST, NEW, H

El sistema Fifth divide todo texto en palabras. Como separadores de palabras son válidos SP (espacio) y CR (fin de línea).

, % " [ ] ∅ |

Los siguientes símbolos son siempre inicio de línea:

. $ ²)

y los siguientes se consideran el final:

(: ' #

Los siguientes datos son equivalentes:

MEM$4700 MEM $4700

Todos los datos son convertidos automáticamente a mayúsculas antes de que el compilador trabaje sobre ellos. Las palabras pueden estar formadas hasta por 63 caracteres.

La secuencia '..' denota el final de una línea. Por tanto, lo que siga se toma como comentario. El texto del comentario se conserva aún después de la compilación.

Los números son reconocidos con los siguientes formatos:

u..vw cifras decimales, también -u..vw

$x..yz cifras hexadecimales, ej. $1ffe

'a código numérico de la letra

%Nombre Dirección de Nombre

En el último formato la dirección de una función o estructura se busca en el diccionario de Fifth. También se puede tener un desplazamiento de esa dirección:

%1 Nombre

Además existe un formato para introducir números complejos, el cual no es utilizado en la implementación para la PC. El formato es el siguiente:

xxx j yyy

donde xxx y yyy son números decimales enteros de punto flotante. La parte real es xxx y yyy la imaginaria. 'j', al igual que '%', tiene un significado fijo y por tanto no puede utilizarse como nombre de nuevas funciones.

Los número pueden utilizarse también en la declaración de constantes. El formato es el siguiente:

constante nnn Nombre

donde 'Nombre' se convierte en el nombre del número 'nnn', el cual, puede ser utilizado como se explicó anteriormente. La declaración de constantes, al igual que con las variables, puede extenderse con el uso de coma. El utilizar el nombre de una constante es equivalente a escribir el número correspondiente. Si 'Nombre' es una constante simbólica, entonces '%Nombre' tiene el mismo valor.

Las palabras definidas por el usuario pueden ser eliminadas del diccionario con ayuda de la instrucción 'NEW'. La instrucción

NEW Nombre

elimina la definición más reciente de 'Nombre' y todas las anteriores. Así mismo, esta instrucción coloca la palabra 'Nombre' como la palabra de búsqueda para la función del editor Ctrl-PgUp/PgDn, de manera que la definición de 'Nombre' pueda encontrarse fácil y rápidamente, presionando las teclas F5, Ctrl-Home y Ctrl-PgDn. A menudo se corrige la definición y el resto del programa se transfiere al área de trabajo actual (F7) para ser compilado nuevamente.

La instrucción 'NEW' sin nombre elimina todas las palabras definidas por el usuario.

La instrucción 'H<' explicada en el capítulo 4.8 también elimina elementos de la lista de símbolos, pero no inicializa los apuntadores de código y datos, de forma que el código de las funciones eliminadas permanece y no será sobreescrito por las siguientes definiciones. Los nombres siguientes a la función 'H' pueden ser precedidos del modo de trabajo en caso de que el mismo nombre sea utilizado en distintos modos. Un ejemplo sería:

```
h< XYZ, 0 NN, 1 NN, UVW, %
```

Con el comando 'LIST' se muestran los símbolos almacenados en el diccionario de Fifth, su modo de trabajo y los apuntadores del programa y datos correspondientes. Cada palabra es mostrada antecediendo el tipo de dato:

n	Función del modo n, n=0-9, A, B, C, D, E, F.
n:xy	Objeto del modo n con x argumentos y y resultados.
n*	Función compilada del modo n, n=0-9, A, B, C, D, E, F.
M*	Función compilada para todos los modos.
M-	Componente de un registro.
n=	Variable del tipo palabra o byte para el modo n.

n+ Estructura de datos con parámetro de longitud en modo n

n### Variable de proceso o puerto en el modo n.

(c) Nombre de una clase.

N Constante.

M Palabra implementada del vocabulario básico.

X Palabra no implementada en el modo seleccionado.

'LIST' muestra normalmente la información en la pantalla e iniciando con las palabras definidas por el usuario. Al presionar la barra espaciadora se muestran las palabras implementadas en el sistema y finalmente las palabras básicas del diccionario de Fifth. Si se presiona cualquier otra tecla termina el comando. Se puede obtener un listado en impresora presionando Alt-F1. 'LIST' seguido del nombre de una clase lista los elementos correspondientes.

'LIST' tiene una variante: 'ULIST', que lista todas las palabras definidas por el usuario incluyendo todos los nombres en las clases seguidos de sus direcciones correspondientes. Los nombres que aún no han sido utilizados están marcados con una 'u' en la dirección. Así pueden encontrarse definiciones inútiles en programas largos. La instrucción 'ULIST nombre' envía a la impresora la lista de símbolos con 'nombre' como cabeza de página.

6.2 Funciones compiladas y funciones del compilador

KF!, ORIG, ESR, BYTE, WORD, MODE, SET, CF:, RDPAR,

ZEXT, ZTYP, BYTE0, WORD0, M(), TOKEN, BTBL, WTBL

Fifth es un compilador de un sólo paso, el texto del programa es leído en forma secuencial palabra por palabra y convertido a código máquina del microprocesador correspondiente. Las funciones especificadas en el texto no se ejecutan inmediatamente,

sino que son traducidas en instrucciones máquina, típicamente llamados a subrutinas. De cada definición de una función surge una subrutina en código máquina que se almacena en la memoria de programas del microprocesador objetivo. Las funciones objeto son compiladas de la misma forma. El código necesario para manejar las pilas no es parte del subprograma, éste es incluido al momento de la ejecución. Las instrucciones ejecutadas inmediatamente, no contenidas en la definición de una función, son compiladas del mismo modo, sólo que el subprograma creado es ejecutado inmediatamente por el microprocesador objetivo.

Las funciones de control son traducidas como llamados a funciones normales. El compilador se encarga de colocar en la pila del programa la dirección donde se debe continuar al terminar la sección del programa. Esto se realiza únicamente con las funciones cuyo nombre termina en ':'. Esta 'norma' puede ser forzada utilizando la instrucción del compilador 'KF!'. Por ejemplo, en la siguiente secuencia se lee la dirección del programa siguiente al cierre del paréntesis:

```
... (kf! fadr)...
```

El lugar de memoria dentro del microprocesador objetivo en donde se colocan los nuevos subprogramas, es determinado por una variable del compilador, la cuál es identificada con 'PC' al ejecutar el comando 'LIST'. Después de cada definición se incrementa esta variable con la longitud del subprograma de manera que, no sea sobreescrito con los siguientes datos. Esta variable no sufre cambio con los programas que se ejecutan inmediatamente ya que el código sólo es utilizado una vez y después se puede sobreescribir. El apuntador de código puede ser modificado a un cierto valor con la instrucción 'ORIG'. La siguiente instrucción hará que el código de la próxima función sea grabado en la dirección $1234:

```
ORIG $1234
```

La dirección debe ser un lugar de memoria destinado a recibir el código del programa a través del sistema. La dirección debe elegirse de manera que no sobreescriba código

anterior. En la PC, el valor inicial del apuntador de código no debe reducirse ya que se destruiría parte del programa del propio compilador.

El rango de memoria utilizado por la PC es de 65KB debido a la segmentación del espacio de memoria, lo cual limita el tamaño total del programa compilado. Las direcciones básicas utilizadas por el compilador son de 32 bits, de manera que, para microprocesadores de 32 bits no existe esta limitante. Para microprocesadores con gran espacio de memoria las estructuras de datos son almacenadas en un segmento distinto al código del programa. En una configuración de multiprocesadores se utiliza un espacio de memoria individual para cada microprocesador.

Los subprogramas que se encuentran almacenados con anterioridad en un sistema (en un EPROM) pueden ser llamados como funciones de Fifth declarándose como funciones externas con la instrucción 'ESR'. Se debe proporcionar la dirección inicial del subprograma de la siguiente manera:

```
esr $1234 nombre
```

La ejecución del subprograma se efectúa con la palabra 'Nombre'. La declaración de un subprograma externo no modifica el apuntador del código. También se podría especificar como un objeto 'obj n m'.

Ciertas instrucciones no son permitidas dentro de la definición de una función o dentro de instrucciones compiladas. Las variables y estructuras de datos no pueden ser declaradas dentro de instrucciones compiladas.

Las funciones de Fifth pueden contener código en lenguaje ensamblador. Esto puede efectuarse con ayuda de las instrucciones o en forma de bytes o palabras con las instrucciones 'BYTE' y 'WORD'. Estas instrucciones colocan los bytes o palabras constantes dentro del código del programa.

```
BYTE        $13,$44,$71,$96
```

WORD $7893,$962

Para facilitar la introducción de tablas de símbolos, la instrucción 'BYTE' acepta el formato alfanumérico.

BYTE 'C','A','S','A',0 .. Es equivalente a

BYTE "CASA",0

Los números de punto flotante son escritos con 'WORD' en el formato de 32 bits de la IEEE aún cuando la aplicación sea de tan sólo 16 bits.

El compilador de Fifth puede cambiar el microprocesador destino con la instrucción

MODE n .. n= 0 .. 15

La instrucción 'MODE' no puede ser utilizada dentro de la definición de una función. El sistema básico trabaja con los modos 0 y 1. El modo 1 es el normal para los programas de Fifth que se ejecutan en la PC. En el modo 0 son definidas nuevas funciones para la generación de código y funciones básicas del compilador. De esta forma el compilador de Fifth puede ser ampliado. En el modo 0 se cuentan con la mayoría de las posibilidades del modo 1 y a través de llamados a objetos pueden ser utilizadas funciones de otros modos, sin embargo, las definiciones de objetos en el modo 0 no pueden ser utilizadas por otros modos. Los modos 0 y 1 utilizan el microprocesador 80x86 de la PC pero utilizan distintos segmentos para sus respectivos códigos. De esta forma, el segmento de la PC para el código del usuario no es utilizado por el compilador. En general, cada modo utiliza su propio apuntador para código y datos. A diferencia del modo 1, el modo 0 utiliza una aritmética entera en el rango de 0 a 65,535. La división es aquí una operación entera.

En todos los modos del compilador pueden ser definidas funciones que en los listados se marcan con 'M*' o 'n*'. Estas son instrucciones que controlan el procesamiento de textos por el compilador. A diferencia de las funciones compiladas que sólo son utilizadas

cuando el código generado por el compilados las manda llamar, las instrucciones del compilador se ejecutan inmediatamente durante el análisis del texto. Algunos ejemplos de estas instrucciones son 'VAR', '>>', '>', 'OBJ' y 'ORIG'. 'SET' es una función del compilador mediante la cual, puede modificarse una variable del modo 0 durante la compilación.

```
SET %A $1234        .. coloca $1234 en A
```

El compilador de Fifth permite definir nuevas funciones del compilador para cada modo. La definición de una instrucción del compilador se diferencía de la definición normal de una función por la instrucción 'CF:'. Por ejemplo:

```
CF:
```

se coloca al inicio de una definición, mientras el compilador se encuentra en modo n, la instrucción del compilador definida se puede utilizar inclusive en el modo n. Las instrucciones del compilador son, por defecto, funciones del modo 0, cuyo código reside en el segmento de código del modo 0 en la PC y puede utilizar funciones del modo 0 en su definición. También pueden ser definidas funciones del compilador que sean efectivas de inmediato en todos los modos. Estas se definen en el modo 0 con un doble 'CF:', es decir 'CF:CF:'.

Las funciones definidas en modos distintos pueden tener el mismo nombre. Sin embargo, las funciones objeto deben tener un nombre único, pues la nueva definición sobreescribiría a la anterior. Las palabras pertenecientes al vocabulario básico al igual que '%' y 'j' no están disponibles como nombres que puedan ser utilizados. Las funciones que no pertenecen al vocabulario básico y que son independientes del modo utilizado pueden ser redefinidas de manera que para un modo tenga un significado particular.

En el modo 0 hay una gran cantidad de funciones especiales disponibles para ser utilizadas en la definición de funciones del compilador. Las tres funciones de mayor uso son: 'RDPAR', 'BYTE0' y 'WORD0'.

'RDPAR' verifica si la siguiente palabra en el texto del programa es una constante y coloca el resultado de la operación en la pila. Si se trata de una constante el resultado es 0. La variable 'ZTYP' contiene un valor diferente de 0 si la operación se realizó correctamente y 'ZTEXT' contiene el byte superior del dato de 32 bits. Con datos de punto flotante 'ZTYP' contiene el exponente decimal elevado $8000 y la mantisa se produce como un número entero de 32 bits. Para otros tipos de datos ZTYP=1 (decimal), 2 (hexadecimal), 3 (ASCII), 4 (dirección) ó 5 (hexadecimal 32 bits).

'BYTE0' y 'WORD0' toman un parámetro de la pila y lo colocan como bytes o palabras en el código compilado, son las versiones de 'BYTE' y 'WORD'. La siguiente instrucción del compilador definida en el modo 1 coloca el byte $fb en el código del programa cada vez que 'sti' aparece en el programa. Debido a que las funciones del compilador específicas para un modo, colocan una serie de bytes en el código del programa, también se les conoce como macros.

```
CF: $fb byte0 >:sti
```

Las funciones del compilador, como la anterior, que colocan una serie de bytes pueden definirse más fácilmente con 'm('. Esta instrucción traduce a código el texto del programa hasta el cierre del paréntesis. La siguiente función es equivalente a la anterior:

```
CF: m( byte $fb) >:sti
```

Dentro de los paréntesis del macro el compilador se encuentra en el modo para el cual es definida la función. 'm(' también puede ser utilizada en definiciones de subprogramas del modo 0. Se puede especificar explícitamente para que modo se desea compilar.

```
M( #2 ... )          .. para modo 2
```

Los parámetros leídos del texto del programa pueden ser reunidos y clasificados como símbolos. Para esto, puede ser utilizada la función 'TOKEN' del modo 0 seguida de una lista de símbolos separados por coma. La función 'TOKEN' toma una palabra del texto

del programa, la busca en la lista y entrega como resultado la posición de la palabra en la lista y como TOS $ffff si la búsqueda fue exitosa, 0 en caso contrario.

La implementación de tablas en el código con acceso a bytes se facilita con la función 'BTBL'. Esta función es seguida de una lista de constantes.

```
Btbl 1,2,4,8,16,0,0,0 >:c-tab
```

Los elementos de la lista pueden ser direccionados y leídos con un índice. En unión con 'TOKE' pueden realizarse ensambladores de mnemónicos de manera sencilla. 'TOKEN' produce el índice de una palabra en un tabla de símbolos y con 'BTBL' se leen los códigos ordenados. De manera análoga se puede colocar una tabla de palabras de 16 bits con 'WTBL'. 'WTBL' y 'BTBL' están implementadas también en el modo 1 al igual que 'TOKEN' y 'RDPAR' en unión con la función 'INP1'.

6.3 Estructuras de datos

VAR, ARRAY, D+, D-, D--, D?, E*, E/R, SL, SR, D>F, F>D,

DMEM, MSEG:, DSEG, CSEG, PORT, ALLOC, ALLOC+,

DS, CPSR, RWC, AEXT, CPASR, PFX, CP'

Las variables globales y los arreglos tienen las siguientes variantes:

VAR	ó	ARRAY	.. Tipo de datos estándar
VAR.B	ó	ARRAY.B	.. Byte
VAR.F	ó	ARRAY.F	.. Punto flotante
VAR.D	ó	ARRAY.D	.. Doble palabra

Las variables de doble palabra mueven dos palabras cuando trabajan con la pila. Si se coloca una palabra doble en la pila la de menor peso irá a la cabeza. En

implementaciones de 8, 16 ó 32 bits son de 16, 32 y 64 bits respectivamente. Pueden contener números enteros del doble de largo o parejas de valores. Al tipo de variable VAR.D le puede ser dado un tipo de clase al igual que a las variables estándar, donde se coloca en paréntesis el nombre de la clase antes del nombre de la variable. El tipo palabra doble es soportado por varias operaciones aritméticas enteras con doble precisión, lo cual no ocurre para el tipo punto flotante. Las variables y arreglos de palabras dobles no pueden ser utilizadas como elementos de un registro. Las operaciones aritméticas sobre palabras dobles son 'D+', 'D-', cambio de signo 'D—' y desplegado 'D?' (ver 7.2). La operación de multiplicación 'E*' forma una palabra doble a partir de dos palabras. 'E*' es una operación sin signo. 'E/R' divide una palabra doble entre una palabra y entrega una palabra como residuo y una palabra doble como cociente. Además existen dos operaciones de corrimiento sobre palabras dobles: 'SR' y 'SL'. Estas operaciones requieren tres parámetros, la palabra doble y el número de corrimientos. El resultado es la palabra doble con la cantidad de corrimientos correspondiente. Las palabras dobles que contienen enteros de doble largo pueden ser convertidos a punto flotante y viceversa con las instrucciones 'D>F' y 'F>D'.

El espacio para una estructura de datos puede elegirse automáticamente o indicarse explícitamente. La definición de una variables ABC sería:

```
VAR $1234 ABC    .. Explícitamente

VAR ABC          .. Automáticamente
```

Los ARRAY's y BARRAY's serían definidos

```
ARRAY 13 $1234 ABC    .. Explícitamente

ARRAY 13 ABC          .. Automáticamente
```

Normalmente al inicio de un programa se definen una gran cantidad de variables y estructuras de datos. Para abreviar puede colocarse una lista de nombre separados por coma después de la instrucción 'VAR'. El comando 'VAR' es substituido por el comando

de repetición ','. También 'ARRAY' puede ser seguido de una lista de nombres con su correspondiente información de longitud.

```
ARRAY 13 abc, 2 coef, 4 $1000 s-tab
```

También para los componentes de estructuras de registros declarados con '.VAR' y '.ARRAY' puede colocarse explícitamente el desplazamiento a partir de la dirección inicial del registro.

```
.ARRAY 8 22 DATA        .. Componente del arreglo con desplazamiento 22
```

De esta forma puede darse otra interpretación de los datos sobre componentes ya definidos. El compilador no verifica si el componente definido se encuentra dentro de los límites del arreglo.

Para la separación automática de espacio de una estructura de datos, el compilador utiliza un apuntador que contiene la dirección que puede ser utilizada y se incrementa con cada nueva declaración. Este apuntador puede ser colocado en un valor determinado utilizando la instrucción 'DMEM'.

```
Dmem $1800
```

De esta forma se colocan las estructuras de datos en una región seleccionada, lo que puede ser importante en sistemas que no cuenten con memoria completa.

Debido a la segmentación de memoria en la PC, el apuntador de datos utiliza en los modos 0 y 1 sólo 65 KB. Para aplicaciones que deban operar con muchos datos, este espacio puede ser muy pequeño. Por esta razón con la dirección explícita para archivos y estructuras de datos, puede ser agregada una dirección de 32 bits. La mitad superior de la palabra (16 bits) es utilizada, en las estructuras de datos, como la dirección relativa al segmento de datos estándar. En las implementaciones de 32 bits se utiliza completo el espacio de memoria con el apuntador de datos.

Con la función de acceso a memoria 'MM.z' (ver 4.4), en implementaciones de 16 bits como en la PC, todo el espacio de memoria está disponible. En el modo 1 utiliza 'mm' una dirección de 16 bits tomada de la pila como parámetro, pero también puede ser cambiado a cualquier segmento de 65 KB con la función 'MSEG:', cuya dirección absoluta se toma de la pila. El efecto de 'MSEG:' se limita al subprograma siguiente. Las direcciones de los segmentos de datos y código pueden conocerse con 'DSEG' y 'CSEG' respectivamente.

El compilador realiza los accesos a los componentes de registros con un apuntador de dirección y con ayuda de las operaciones estándar 'MM', 'MM.B' y 'MM.F'. En sistemas de 16 bits con espacio de memoria de 32 bits debe elegirse el segmento apropiado con 'MSEG:' a menos que se utilice el segmento estándar y que sea ahí donde estén localizados los registros.

Algunos tipos de microprocesadores, como el de la PC, utilizan un espacio de memoria propio para los dispositivos de E/S. Fifth permite el acceso a estas direcciones de E/S de tipo byte a través de un tipo propio de variable 'PORT', además de la opción 'mm.x'. Los puertos son declarados y utilizados como una variable, pero se requiere incluir en su declaración la dirección correspondiente. Algunos puertos importantes de la PC pueden ser declarados con:

```
PORT $20 IRQCTL, $42 CTR2
```

Fifth permite, además de la posibilidad de definir tipos de registros, definir tipos de datos formados por diversas variables o tipos especiales de arreglos que no pertenecen necesariamente a las estructuras elementales. A pesar que los registros ofrecen la funcionalidad requerida por la mayoría de las aplicaciones y que las construcciones normales del lenguaje deben ser preferidas, es importante contar con mecanismos que ayuden a trabajar con microprocesadores especiales, funciones especiales de hardware o tipos de direccionamiento a través de tipos de datos propios con una generación especial de código. En la definición de estos nuevos tipos de datos debe ser establecido claramente, como deben realizarse las operaciones correspondientes de lectura y

escritura y como debe generar el compilador el código del nuevo tipo cuando se utilice la estructura de datos. La definición de nuevos tipos de datos se realizan con ayuda de 'ALLOC', 'ALLOC+' y 'DS'. Cuando las instrucciones de lectura y escritura del nuevo tipo se realizan como subprogramas, el código generado está formado por llamados a esas subrutinas para lo cual ayuda la función 'CPSR' del compilador.

Cada estructura de datos del nuevo tipo tiene una dirección conocida por el compilador, la cual es utilizada por las operaciones de lectura y escritura como un parámetro numérico. La colocación del parámetro numérico en la pila de datos durante la ejecución, es efectuada por la instrucción del compilador 'CP'NN'. La definición de un nuevo tipo de dato tiene normalmente la siguiente estructura:

```
CF:
nnn ALLOC          .. Cada estructura utiliza nnn bytes
DS                 .. Nombramiento de la estructura
CP'NN
RWC IF (%Instrucción Lectura) ELSE (%Instrucción Escritura)
CPSR
                   .. El código compilado tiene la forma
                   .. 1) Dirección constante instrucción de lectura
                   .. 2) Dirección constante instrucción de escritura
>:NOMBRE
```

Los primeros dos renglones definen lo que debe hacer el compilador cuando se declare una estructura del nuevo tipo: generar una dirección con ayuda del apuntador de datos y reserva nnn bytes. 'DS' facilita el registro del nombre de la estructura en la tabla de

símbolos del compilador. El resto de la definición son las instrucciones que debe llamar el compilador, cuando sea encontrado el nombre de la estructura en el texto del programa. El programa encuentra la dirección correspondiente a la tabla de símbolos como parámetro en la pila. La parte de mayor peso de la palabra de la dirección de 32 bits se localiza en la variable 'AEXT'. La función 'RWC' pregunta si se encuentra un '>>' frente al nombre de la estructura, es decir, si se compila una operación de lectura o escritura.

'ALLOC' permite nuevamente la opción de almacenamiento explícito. Las estructuras del nuevo tipo pueden ser definidas en alguna de las siguientes formas:

```
NOMBRE $1234 ABC      .. Almacenamiento explícito

NOMBRE ABC            .. Almacenamiento automático
```

Si se desea poder declarar varias estructuras del mismo tipo separándolas con coma al igual que en la declaración de variables, es necesario colocar la palabra 'KOMMA' al inicio de la definición del tipo.

'ALLOC' tiene la variante 'ALLOC+' que conserva en el diccionario el parámetro de longitud. Con 'ALLOC+' pueden ser cargadas y almacenadas estructuras previamente definidas con las operaciones de disco 'LDF' y 'SDF' (ver 7.4). La instrucción del compilador encuentra ambos parámetros de longitud en las variables del sistema 'DISP' y 'OP'.

Las instrucciones 'RWC IF ... CPSR' pueden ser formuladas más cortas con ayuda de 'CPSAR' (compilación alternativa de subrutinas).

```
%InstrucciónLectura %InstrucciónEscritura cpsar
```

'CPSAR' se utiliza también para crear una seudovariable a partir de un par de subprogramas. Por ejemplo, a partir de dos funciones 'recibir' y 'enviar' se puede definir la variable 'serie':

```
CF: %recibir %enviar cpsar >:serie
```

Las instrucciones del compilador en la definición de un tipo pueden ser más complicadas que en el ejemplo anterior y pedir, por ejemplo, más parámetros del texto del programa para seleccionar un campo correspondiente a la estructura. El siguiente ejemplo muestra la definición de un tipo complejo de doble palabra para el modo 1, tal y como se realizaría a partir de un registro. Las palabras componentes se seleccionan con los símbolos '.re' e '.im'. Las direcciones de las palabras componentes son calculadas a partir de la dirección inicial de la palabra doble. Los componentes pueden entonces ser compiladas en la dirección calculada como variables normales, para lo cual sirve la función 'CP'SV'. Si no se elige ningún componente aparecerá un mensaje de error.

```
mode 0

token .re, .im >:select

mode 1

CF:

4 alloc                 .. Espacio para dos palabras

ds

select

c( 2 * +                .. Agregar desplazamiento

cp'sv

)q

err "ERROR"             .. Mensaje de error

>:c-var
```

Para estructuras del tipo 'c-var' la definición debe ser seguida del componente. Por ejemplo, para mostrar la parte real de la estructura 'z' se escribe:

```
z.re ?
```

Las estructuras de datos pueden tener, como los arreglos, un argumento opcional entre corchetes cuando al inicio de la instrucción de generación de código se coloca 'PFX'. 'PFX' genera como resultado 0 si el llamado a la estructura no tiene argumento, de lo contrario el resultado es -1.

Las instrucciones del compilador 'CP'NN' y 'CP'SV' son casos especiales de la función del compilador 'CP", con la cual pueden ser compilador elementos básicos del diccionario (palabras marcadas con M) así como estructuras básicas del lenguaje. La palabra deseada se escribe después de 'CP" como podría ser 'CP'+'. La siguiente función crea los códigos máquina para sumar una constante con el primer elemento de la pila (TOS).

```
CP'NN CP'+
```

Las instrucciones 'CP" deben colocarse en el orden descrito (postfijo). La serie 'CP'+ CP'NN' tiene un significado totalmente distinto, aún cuando se esté utilizando la notación infija en el texto del programa. 'CP" trabaja dependiente del contexto y efectúa las posibles optimizaciones. La variante 'CP'FC' se compila igual que 'CPSR' en un llamado a una función (parte alta de la palabra de la dirección en la variable FEXT), pero realiza las conversiones necesarias. El llamado a objetos deben ser compilados con 'CP'OB' y contener el modo de operación así como el número de parámetros y resultados.

6.4 Comunicación entre microprocesadores

KFL, KMOVE, BMOVE, #

En configuraciones extensas del sistema Fifth con varios modos de compilación y microprocesadores, la PC se encarga de las funciones de entrada y salida. En caso que un microprocesador del sistema requiera un servicio del modo 1, se le deberá enviar una petición.

El único soporte de comunicación entre los microprocesadores del sistema Fifth lo constituye el llamado a objetos. Estos llamados son transferidos por el sistema automáticamente sin necesidad de programación especial. Para poder ejecutar una operación del modo 1 desde un microprocesador del subsistema ésta debe declararse como objeto. Este mecanismo es válido también para llamados al microprocesador del modo 1 desde un microprocesador del subsistema y en el caso de subsistemas con varios microprocesadores, para llamados entre ellos.

La función '?' para despliegue en pantalla se efectúa para los subsistemas a través de un llamado a un objeto del modo 1. Para desplegar libremente una cadena alfanumérica el compilador crea automáticamente las funciones de despliegue del modo 1 e introduce los llamados a objetos en el código del subsistema.

Para subsistemas con datos de 1 byte se utiliza un protocolo simplificado, el cual, no contempla el llamado de objetos del modo 1 desde el submicroprocesador (pero sí en sentido contrario) y maneja los desplegados directamente en la consola.

La comunicación se lleva a cabo según protocolos estándar en forma serial sobre los puertos de comunicación ('vínculo'). En un sistema con varios microprocesadores la comunicación no se realiza con un sólo puerto de la PC. Por ello, son contemplados hasta 8 vínculos. Para cada vínculo se determina un protocolo, así como rutinas de envío y recepción de bytes. Las rutinas de envío y recepción son funciones de Fifth configurados libremente con un parámetro y resultado respectivamente. En el caso más sencillo, las rutinas de envío y recepción trabajan con el puerto serie. Existe la posibilidad de omitir el protocolo estándar y crear nuevas funciones de Fifth para el manejo de llamados a objetos.

El vínculo de comunicación con un microprocesador es establecido durante la instalación del modo de compilación correspondiente utilizando el comando del sistema 'KFL' y normalmente una aplicación no lo modifica.

KFL N, N=0..7 ó -1

Con el comando 'KFL' puede ser especificado además el protocolo de comunicación así como las rutinas de envío y recepción.

KFL N, P, %r, %e

Donde N es el vínculo deseado (0..7), P el tipo de protocolo y %r, %e las direcciones de las rutinas. En lugar de direcciones se pueden colocar los símbolos SIO.1 .. SIO.4 para utilizar las rutinas estándar (ver 7.2). Los valores que puede tener P son:

P=0	Protocolo de 16 bits
P=1	Protocolo de 8 bits
P=4	Segundo protocolo de 16 bits

El parámetro utilizado por el vínculo es válido para todos los modos que pueden utilizar este vínculo.

El uso más frecuente de 'KFL' es:

kfl -1

Esta instrucción suprime las rutinas de comunicación del compilador para ese modo de forma que en lugar de ser utilizado el espacio de memoria para código del microprocesador objetivo se utilice un segmento de la PC, el llamado segmento de código cruzado, donde se coloca el código compilado en lugar de utilizar el sistema objetivo real. Naturalmente no existe la posibilidad de ejecución del código. Los comandos 'MEM', 'LDF' y 'SDF' para inspeccionar , editar, cargar y almacenar el código compilado se conservan. Los segmentos de código cruzado son normalmente el

segmento de datos del modo 0/1, pero pueden ser colocados relativos al segmento de datos incluyendo un parámetro.

```
Kfl -1, $2000          .. 128K sobre el segmento de datos
```

El segmento de datos del modo 1 sólo puede ser utilizado en el rango 0..$bfff ya que arriba de $c000 residen los datos del sistema y las pilas. Para sistemas objetivo que requieran el rango superior debe seleccionarse otro segmento de código cruzado, como $f800, en donde se elimina el rango de direcciones para código $8000..$ffff del microprocesador objetivo con el rango $0000..$7fff del segmento de datos del modo 1.

6.5 Funciones monitor

MEM

Para inspeccionar la memoria de la PC y de los subsistemas conectados existe una función:

```
MEM $1234
```

Lista el contenido de 16 lugares de memoria a partir de la dirección indicada. El número puede ser también substituido por %NOMBRE. Tras presionar la barra espaciadora se muestran los siguientes 16 espacios de memoria. Con la tecla '-' puede posicionarse el cursos bajo un valor. Con presionar dos valores, éstos son introducidos en la memoria.

'MEM' funciona siempre sobre el modo de trabajo del subsistema correspondiente, es decir, en el modo n se muestra la memoria del microprocesador correspondiente. Tras confirmar el valor dado con la tecla '-', éste es enviado a la memoria del subsistema objetivo.

En una dirección de 32 bits se toman los 16 bits iniciales como la dirección absoluta del segmento. Si se introduce la dirección en forma simbólica como %NOMBRE, 'MEM' utilizará el segmento en donde se encuentra almacenada la función o estructura 'NOMBRE'.

6.6 Errores durante la ejecución

El código generado por el compilador no contiene por sí sólo instrucciones para detectar errores durante la ejecución. Es decir, no existe un mecanismo para controlar los límites de arreglo o pila o señalar una división entre 0. Sólo llamados a objetos en direcciones equivocadas de la PC y daños en protocolo de comunicación estándar 0 se muestran y generan la interrupción del programa.

Los errores de ejecución pueden detectarse fácilmente con el desarrollo interactivo y en las etapas iniciales de un programa. Pero un programa de aplicación puede controlar la interrupción del programa y el desplegado de un mensaje de error con la función 'ERR' del modo 1. 'ERR' se coloca como función de control antes del mensaje de error de la rutina generada como lo muestra el siguiente ejemplo:

```
(err "Error en USP = " USP ? )
```

La PC regresará después de la interrupción al modo de edición. Las funciones que se ejecutan en los subsistemas no se interrumpen.

También las teclas Ctrl-Brk generan a través de una interrupción un error de ejecución y puede así provocarse la terminación de un programa.

7. Funciones especiales en la PC

Además de las funciones y estructuras básicas de Fifth explicadas en el capítulo 4 y de las funciones y características generales vistas en el capítulo 6, el sistema cuenta con funciones y estructuras de datos para la programación en la PC. Dentro de éstas se encuentran técnicas científicas para aplicaciones con varios microprocesadores, programas para comunicación con los puertos serie y paralelo, operaciones en disco y la capacidad de utilizar las gráficas de alta resolución y el coprocesador matemático.

7.1 Soporte de puertos

SIO, RCNT, BAUD

El compilador de Fifth puede hacer uso de los puertos de impresión y serie (RS232) de la PC con rutinas integradas. Con el programa KONF.COM se puede configurar si se trabaja por medio de una terminal y el tamaño de las pilas de envío y recepción.

Los puertos serie pueden ser configurados para la comunicación automática con otros subsistemas o bien pueden ser utilizados directamente por una aplicación del usuario.

SIO	.. lee un byte del puerto serie
>> SIO	.. envía un byte colocado en la pila
1200 BAUD	.. reinicializa el puerto serie
RCNT	.. lee la cantidad de bytes en la pila de recepción

Las funciones mostradas son válidas para el puerto COM1. Para utilizar los otros puertos se tendría que colocar '.2', '.3' ó '.4' al final de 'SIO', de tal forma que 'SIO' y 'SIO.1' son equivalentes.

Los puertos utilizan 8 bits de datos con un bit de inicio y otro final, ningún bit de paridad ni líneas de control. Si el compilador de Fifth es almacenado con el comando 'BUILD' (ver Cap. 9.2) después de utilizar 'BAUD.x dato', cuando se entre nuevamente al compilador la inicialización del puerto se efectuará automáticamente. Los puertos seleccionados con el programa 'KONF' son también incializados cuando arranca el programa. Los puertos referidos con 'KONF' o 'BAUD.x' son inicializados después de cada interrupción con Ctrl-Brk. Al configurar un puerto con 'KONF' existe la opción de transmitir la señal DTR por el conector DB25 cada vez que se inicializa enviando un pulso negativo, el cual puede ser utilizado por el subsistema, por ejemplo, para inicializar (reset) al microprocesador con las teclas Ctrl-Brk.

Aún para la comunicación automática con un subsitema, las instrucciones 'SIO.x' y '>> SIO.x' esperan hasta que la operación de comunicación sea efectuada. El ciclo de espera puede ser interrumpido presionando Ctrl-Brk. Cuando se utilizan procesos en paralelo el tiempo de espera es utilizado parcialmente por otros procesos (ver 8.4).

7.2 Funciones de salida

OUT, CR, SPS, >>O:, >>PRN:, >>TEXT,

WND, SSCRN, CHXY, CS, ATTRIB, VPAGE,

GR, GXY, RXY, PKT, VKT, PXY, GBYTES

Las funciones estándar para salida de datos que pueden ser utilizadas en todos los modos como '?' y 'D?' así como "..." son funciones de la PC.

La función '?' para desplegar una cantidad puede utilizar un formato seleccionable, fijado con la instrucción 'F#n'.

n = 0 Hexadecimal $xxxx

n = 1 Con signo

n = 2	Posicionado en un campo de 16 espacios
n = 3	Bytes xx
n = 5	Sin signo
n = 6	Posicionado en un campo de 16 espacios

La función de despliegue de palabras dobles 'D?' es utilizada por sistemas de 32 bits para efectuar la operación '?'. Los formatos utilizados son:

n = 0	Hexadecimal $xxxx.xxxx
n = 1	Con signo
n = 2	Posicionado en un campo de 16 espacios
n = 3	Igual a n = 1

De igual manera se muestran números de punto flotante, el formato hexadecimal es la representación binaria del número.

Caracteres individuales se muestran con la función 'OUT'. Las funciones 'CR' (retorno de carro), 'SPS' (n espacios en blanco) y "..." utilizan la función 'OUT'.

Los datos generados en una función pueden ser enviados a una rutina de salida distinta a la estándar con la instrucción:

```
%RutinaDeSalida      >>O:
```

'>>O:' toma la dirección de la rutina de salida de la pila de datos. El cambio de la salida trabaja como función de control y es válida hasta el final del subprograma que contiene '>>O:'. El siguiente ejemplo muestra como es posible enviar un texto por el puerto serie en lugar de mostrarlo en pantalla.

```
>> sio.1 >:send

%send >>O:

"Pila de datos = " usp ? cr

>:send-usp
```

La rutina a la que se transfiere el despliegue de la información puede contener a su vez instrucciones de salida. El ejemplo anterior se puede modificar para enviar datos al puerto serie y a la pantalla.

```
dup >> sio.1 out >:send
```

Variantes especiales de '>>O:' son '>>PRN:' y para el modo 0 '>>TEXT:', que envían los datos a la impresora o al almacén temporal de textos respectivamente. La información desviada a la memoria de texto es colocada al final del texto que se encuentra en la parte superior de la pila de textos. Por ejemplo, si se efectúa después de presionar la tecla F3 una llamada a la función del modo 0 como la siguiente, coloca una tabla de senos al final del texto editado.

```
>>text:



#1

"WORD"

( 64 it:

        dup 7 and 0 = if(cr)

        128 / pi * sin 32767 * f>i f#1 ? ","

)
```

"

32767

>:tabla

"

>:generador_tabla_seno

La función del modo 0 llama para la generación de su tarea al modo 1 con '#1' y puede, por tanto, hacer uso de las funciones de punto flotante. Si no existe un coprocesador matemático se debe utilizar el archivo SWFPT.TXT.

Los textos que no son desviados se envían a la pantalla de la terminal. La rutina de la terminal puede cambiarse. El modo de texto es configurado con 'KONF' utilizando la instrucción 'TX' y es válido al iniciar el sistema. Con 'GR.CGA', GR.EGA', 'GR.VGA' o 'GR.H' se cambia el modo gráfico correspondiente a las normas CGA, EGA, VGA o Hércules. En los modos EGA, VGA y Hércules se utilizan 43 ó 60 líneas de texto.

La salida a pantalla puede modificarse con algunas instrucciones de control:

n WND selecciona como salida para los siguientes datas a la ventana n, n <= 15.

n k SSCRN parte la ventana seleccionada en el renglón n para n>0 ó columna -n para n<0. La parte superior o izquierda conserva su número de ventana, mientras la otra parte recibe un nuevo número. Entre ambas ventanas aparece una línea divisora. Si n=0 la ventana k será una copia de la ventana actual, se corre su frontera baja o derecha sin generar una línea divisora.

x y CHXY define la posición del cursor donde aparecerá el siguiente texto en pantalla. (0,0) es la esquina superior izquierda de la ventana.

CS Borra la pantalla actual.

n ATTRIB define el color del texto para TX, EGA y VGA.

n VPAGE define la hoja visible en la pantalla.

Después de inicializar un manejador de video con TX o GR.XXX las primeras 16 posibles ventanas, en las distintas hojas de pantalla disponibles, son iniciadas, de manera que puedan ser utilizadas. Inicialmente la ventana 0 es seleccionada al igual que las primeras hojas de pantalla.

Los valores del atributo corresponden a la convención del BIOS, por ejemplo, '$17 ATTRIB' significa en modo texto letras blancas en fondo azul. Para el modo gráfico, el color de fondo utilizado por CS, es el Byte de mayor peso del parámetro atributo.

Existen funciones para dibujar puntos y líneas en el modo gráfico:

x y GXY Define una coordenada absoluta en (x,y) ((0,0) es la esquina superior izquierda).

dx dy RXY Define una coordenada relativa a la última posición.

PKT Coloca un punto en la coordenada actual.

dx dy VKT Escribe un vector en la coordenada actual.

x y PXY Escribe un vector hasta la nueva posición (x,y).

El rango de coordenadas posible depende del modo gráfico utilizado y del tamaño de la ventana.

En el siguiente ejemplo son calculadas las coordenadas de una figura 'Lissojous' y utilizadas para dibujar:

```
gr.vga

320 240 gxy
```

```
1384 it:

        dup 20 / sin 200 * 320 +

        swap 24 / sin 200 * 240 +

        pxy
```

Como elemento para una impresora de puntos con una gráfica puede ser utilizada la función 'GBYTES'. Al igual que 'CHXY', requiere de dos coordenadas como parámetro y toma los 64 puntos localizados en la columna y los coloca como 8 bytes en la posición %78 WBUF. El siguiente ejemplo muestra el uso de GBYTES para imprimir una pantalla EGA.

```
.. Impresión de una línea de gráfico (8 puntos de columna)

prn:    .. Número de línea en la pila

27 out '* out' 4 out

128 out 2 out .. 80 puntos/pulgada, 640 G-Bytes

(80 it: over gbytes 8 it: 78 + wbuf[] out) rm

27 out 'J out 24 out $d out     .. 24/180 pulgadas de corrimiento

>:imprime_línea_gráfica

.. Función de impresión

43 it: imprime_línea_gráfica >:ImprimeEGA
```

7.3 Instrucciones interactivas

INP1, WORT, EOL, RDPAR, .FP, LINE:, INP

Los programas de Fifth son ejecutados interactivamente utilizando su nombre y anteponiendo sus parámetros. Durante la ejecución, las funciones pueden requerir datos del usuario. Para leer el teclado se utiliza la función 'KEY' del modo 1. Esta devuelve como resultado un cero si no se presionó una tecla o un -1 en la pila y el código de la tecla. La siguiente línea espera a que se presione una tecla y muestra el código correspondiente:

```
rep: key c(?)q
```

Con la función INP1 se lee una línea completa que puede ser editada con la tecla BS (Back space) y termina cuando se presiona una tecla sin símbolo. INP1 tiene como parámetro la longitud máxima de la línea y regresa como resultado el código de la tecla con lo que se finalizó. Los datos aparecen en pantalla en la posición del cursor actual, que puede modificarse con 'CHXY'. La secuencia

```
0 INP1
```

lee sólo un símbolo

```
-1 INP1
```

retorna la edición en el lugar que se dejó sin inicializar el almacén temporal de datos. Una línea de máximo 5 caracteres que sólo pueda terminar con la tecla 'ENTER' se puede implementar de la siguiente manera:

```
5 rep: inp1 $d = c( br )q -1
```

El texto leído puede ser cómodamente analizado con las funciones 'WORT', 'EOL', 'TOKEN' y 'RDPAR'. 'WORT' retira, según la convención de Fifth, una palabra del texto y lo coloca en el arreglo de bytes 'WBUF' convertido en mayúsculas. La longitud de la

palabra está en 'WLEN'. 'EOL' prueba si se a alcanzado el final del texto. 'TOKEN' es utilizado, como se describió en 6.3, también en el modo 1 para definir símbolos de búsqueda que se utilizarán para buscar en la línea de texto. Con RDPAR se lee un número, donde se prohibe la introducción de %NOMBRE como constante simbólicas. Se puede pedir un número de la siguiente manera:

```
"Dame un número: " 5 inp1 rm rdpar
```

Los números de punto flotante son convertidos al formato de 32 bits de la IEEE. La palabra de menor peso se coloca en la pila mientras que la de mayor peso se coloca en la pila mientras que la de mayor peso es colocada en la variable 'ZEXT'. En este caso, el número aún no se encuentra disponible como operador de punto flotante. Primero se debe utilizar la función de conversión '.FP', la cual puede ser utilizada junto con 'RDPAR'.

```
10 inp1 rdpar.fp
```

La línea de texto puede ser llenada también por la salida de un programa. Existe la función de control 'LINE:', la cual envía el subprograma que le prosigue al almacén temporal.

Los datos más complicados pueden ser leídos de forma sencilla en el modo 0 con un llamado controlado por el programa al editor y compilador. El comando correspondiente sería:

```
inp !
```

causa que el texto sea leído a través del editor de textos y traducido por el compilador de Fifth. 'INP' entrega como resultado la dirección inicial del código del programa y '!' se encarga de ejecutarlo. El texto introducido puede ser cualquier combinación de funciones de Fifth. Las instrucciones se tratan como un subprograma y pueden utilizar datos anteriormente colocados en la pila o dejar ahí nuevos valores. 'INP' no verifica si las instrucciones del usuario producen un determinado número de datos. En caso que se

requiera una verificación de este tipo, el programa maestro debe preguntar por el valor de la pila antes y después de efectuar la tarea. Los números introducidos con 'INP' pueden ser formulados como operaciones complejas, ya que todos los datos son compilados y ejecutados como un programa.

INP sólo puede ejecutarse a partir de un programa iniciado en el modo 0, el cual puede hacer uso de otros modos a través de llamadas a objetos. Ejecutando la función 'SETMD' con el modo deseado como parámetro, el compilador se cambia a este modo antes de llamar a INP, compila para ese microprocesador y le envía el código generado. Así, no existe limitante, para ejecutar el programa desde el inicio en el modo 0. Además, las funciones del modo 0 no pueden declararse como objetos para ser utilizadas por otros modos. Para poder solicitar datos compilados por llamado a función en cualquier programa del modo 1, este se inicia como un proceso (ver Cap. 8) y se inicia un ciclo, donde se espera un punto de sincronización hasta que se requiera un dato en el proceso del modo 1. El programa transfiere la dirección de ejecución recibida en una variable de memoria.

El editor de Fifth puede ser ejecutado en el modo 0 también sólo (sin compilador), para realizar únicamente funciones de procesamiento de texto o del compilador. La instrucción es 'EDIT'. 'EDIT' es controlado por el valor de la variable 'CFLG'. Este valor debe ser 0 si se desea iniciar un nuevo texto o se desea eliminar el anterior, -1 si se desea editar el texto actual y en $ff cuando se edita un nuevo texto pero se desea conservar el anterior en memoria. 'EDIT' termina al presionar las teclas F1 o F2 y deja un valor en CFLG, si es F1 un 0 y con F2 un -1. La dirección inicial del texto en el segmento se encuentra en TLIN. La línea de texto en la dirección 'TLIN', puede ser tomado del segmento de texto con la función 'GETLINE' y la coloca en el arreglo de bytes 'LINE' y su longitud en la variable 'LEND'. 'EDIT' requiere a 'GETLINE'. Las funciones del modo 0 para análisis de texto como 'TOKEN' o 'RDPAR' llaman a 'GETLINE'. Analizan el contenido de 'LINE' en la posición del apuntador 'LPTR' y dejan la posición inicial de la palabra examinada en 'WANF'. El último renglón de texto tiene una longitud de 1 y contiene el símbolo $1a.

Como otra posibilidad para la introducción de datos se soporta el uso del ratón de la PC, siempre y cuando se cuente con uno y su manejador este cargado. El ratón es inicializado al entrar al sistema y con cada Ctrl-Brk.

La función 'MAUS' arroja 3 resultados: la coordenada x, y y estado de las teclas del ratón (el bit 0 para botón izquierdo, bit 1 botón derecho). La coordenada inicial y el rango de coordenadas del ratón se determinan con '>> MAUS'. Por ejemplo, para definir una posición del ratón en una ventana gráfica de ancho 320 y alto 240, se define el rango de coordenadas correspondientes y la posición inicial.

```
0 0 320 240 >> MAUS
```

Con ayuda de las funciones gráficas se puede definir un cursor gráfico y mostrar la posición actual del ratón.

```
0 -10 rxy 0 20 vkt -10 -10 rxy 20 0 vkt -10 0 rxy >:gcu
```

Un cursor gráfico definido así tiene la ventaja sobre los que ofrece el BIOS, que sólo es válido en su ventana. El siguiente ejemplo muestra una función para dibujar en la pantalla con ayuda del ratón:

```
var x, y


$87 attrib gcu

(       rep:

        maus dup 1 and c( br )q

        >> fadr

        over x = over y = and
```

```
        fadr swap
        c( rm rm rm ptq )q
        swap >> y swap >> x x swap y swap
        br
)
gcu 7 attrib
>:cum


gr.vga
100 100 over over gxy over over >> y >> x 640 480 >> maus
rep:
cum dup 1 and
c( br rm rm rm )q              .. Termina con botón izquierdo
c( gxy )q pxy          .. Dibuja con botón derecho
>:pm
```

En la función auxiliar 'CUM' se dibuja un cursor en la posición actual con ayuda de las funciones anteriores, pues el ratón es solicitado hasta existir en las variables x y y y entonces debe ser borrado. El llamado a 'PTQ' en 'CUM' se ocupa de que el microprocesador pueda hacer otras actividades entre solicitudes del ratón (ver Cap. 8).

Algunos manejadores de ratón utilizan una frecuencia elevada (54Hz en vez de lo normal 18Hz). En estos casos los timers para procesos colocados por el sistema se activarán más frecuentemente (ver Cap. 8) y las funciones de tono tocarán las melodías más rápido.

Finalmente, se debe recordar, todos los puertos de entrada/salida de la PC pueden ser accesados directamente desde Fifth y programadas sus funciones de entrada y salida para tareas específicas.

7.4 Operaciones en disco

LDF, SDF, (C)FILE

Los programas de Fifth pueden hacer uso de dos tipos de operaciones en disco, los comandos cortos 'LDF' y 'SDF' y las funciones estándar como 'OPEN', 'READ', etc.

Los comandos cortos 'LDF' y 'SDF' sirven para almacenar o leer espacios de memoria de hasta 65 KB de longitud en disco. El rango de memoria se especifica con dirección y longitud tras 'LDF' o 'SDF', o simbólicamente, después de ser declarado en el modo correspondiente como 'ARRAY' o 'BARRAY'.

```
barray 32768 ABC
```

'LDF' (Load Data File) y 'SDF' (Store Data File) son funciones del modo 1. La sintaxis de los comandos es:

```
LDF ABC archivo     .. Carga del disco

SDF ABC archivo     .. Escribe en el disco
```

En caso que el arreglo ABC esté definido en varios modos, se puede especificar el modo deseado:

```
LDF 1 ABC archivo   .. Carga el arreglo ABC del modo 1
```

Los archivos de disco tienen la extensión '.DAT', si no se especifica otra. Se pueden incluir los directorios necesarios junto con el nombre del archivo. Las especificaciones del archivo tras 'LDF' y 'SDF' son constantes, pueden ser modificadas después de la definición.

En lugar del arreglo se puede especificar directamente el modo, dirección inicial y longitud en bytes.

```
LDF 2 $1000 0 c:\f\rs80      .. Carga el archivo RS80.DAT en el modo 2
                             .. dirección $1000
```

Si en 'LDF' se especifica una longitud de 0 o no se utiliza, se carga el archivo completo (max. 65 KB).

Por el contrario las funciones estándar derivadas de funciones del DOS permiten trabajar con archivos de cualquier longitud, cuyo nombre no se compila como constante, si no que se crea a través de un programa. Las operaciones de lectura y escritura utilizan un segmento de datos de Fifth, pero pueden ser utilizadas en otro segmento o modo utilizando la combinación de funciones 'BMOVE' y 'KMOVE'.

Estas operaciones de disco son reunidas en la clase 'FILE'. La mayoría de estas funciones requieren como parámetro el manejador de archivo del DOS, el cual se produce con una operación 'OPEN' o 'CREATE'.

```
FILE NAME            .. Toma el nombre del área de trabajo.

FILE OPEN → h l      .. Abre el archivo con nombre en WBUF.

l<>0                 .. Operación exitosa, h=manejador de archivo del DOS.

l=0                  .. No se abrió el archivo, h=código de error del DOS.

FILE CREATE → h l .. Crea el archivo con nombre en WBUF.
```

a l h FILE READ → rl .. Carga l bytes del archivo, manejador h en el segmento de

.. datos a partir de la dirección a, rl = número real de bytes leídos.

a l h FILE WRITE → wl .. Escribe l bytes a partir de dirección a del segmento de datos

.. en el archivo con manejador h, wl = cantidad de bytes escritos.

ha la h FILE POS .. Coloca el apuntador del archivo h en $ha.la.

h FILE LEN → ha la .. Calcula la longitud del archivo h, el apuntador de archivos

.. se coloca al final del archivo.

h FILE CLOSE .. Cerrar el archivo con manejador h.

FILE DELETE .. Borra el archivo con nombre en WBUF.

Los códigos generados por 'OPEN' son iguales a los del DOS. Varios archivos pueden estar abiertos simultáneamente. El número máximo de archivos abiertos es determinado por la configuración del DOS. La posición del archivo, donde se efectúa 'READ' y 'WRITE', es colocado en 0 después de 'OPEN' o 'CREATE' y aumenta de acuerdo al número de bytes escritos o leídos, de manera que se efectúa una escritura o lectura secuencial. Tras llamar a 'LEN', a través de 'WRITE' el apuntador se coloca al final del archivo. Cada archivo utilizado para escritura debe ser cerrado, de manera que todos los datos se almacenen. En el ejemplo siguiente, al archivo 'EX.TXT' se le incluyen 128 bytes del arreglo 'DBUF'. El nombre del archivo es fijo, pero pudiera también ser solicitado con la función 'INP1'.

```
var (file) extxt
```

```
(>line: "EX.TXT")     .. Nombre del archivo

file name             .. Tomar la línea de texto

open rm >> extxt      .. extxt toma el manejador

extxt len rm rm       .. se coloca al final del archivo

%dbuf 128 extxt write rm

extxt close
```

7.5 Varios

RTIME, RND, TON

Para medir el tiempo de ejecución de un programa se tiene la función 'RTIME', la cual debe anteceder a un programa del modo 1 que se desee medir. Después de la ejecución se muestra el tiempo en milisegundos. Pueden ser medidos tiempos hasta de 54 ms. Los programas cortos pueden aumentar la resolución del tiempo, una iteración:

```
RTIME 100 it: rm FUN
```

en donde se mide también la ejecución de la estructura de iteración.

RTIME puede ser utilizado también para medir la ejecución en otro sistema, en donde se llama a la función a ser medida como objeto y se compensa el tiempo necesario para la comunicación por el puerto serie, es decir, el tiempo de ejecución de un objeto 'vacío'. La comunicación a través de puertos serie asíncronos genera un tiempo de seguridad dependiente del equipo para cada medición.

Para simular eventos aleatorios se tiene un seudogenerador de números aleatorios. 'RND' coloca un valor aleatorio en la pila.

```
10000 it: rm rnd split gxy pkt .. Punto aleatorio

rnd 32767                    .. Número aleatorio en [-1,1]
```

'RND' utiliza un registro de corrimiento circular de 127 pasos, el cual es rotado 16 veces en cada ocasión.

Además, la bocina de la PC puede ser programada para ejecutar melodías con la función 'TON'. Se pueden generar señales acústicas, el programa ejecutor no tiene que esperar la serie de tonos. Junto a 'TON' se coloca la lista de notas con su longitud 1 a 8 en unidades de 1/9. Los nombres de las notas son:

p (pausa), C, Cis, D, Es, E, F, Fis, G, As, A, B, H

c, cis, d, es, e, f, fis, g, as, b, h, c', cis', d', es', e', f'

Ejemplo:

TON c 3, p 1, g 3, p 1, G 3, p 1, c 8

8. Interrupciones y procesos paralelos

El sistema Fifth permite trabajar con interrupciones y procesos paralelos en la PC, de manera que se puede utilizar la PC para programación en tiempo real. La posibilidad de programar interrupciones y utilizar procesos paralelos es compatible a muchos sistemas externos programables en Fifth, de manera que se puedan operar sistemas en tiempo real separados. Después de la breve discusión en el Cap. 4.6, en este capítulo se explican detalladamente los mecanismos de Fifth para trabajar con procesos paralelos.

8.1 Interrupciones

IRQ#

Una rutina de interrupción es una rutina de servicio ejecutada por una interrupción del hardware. Debido a que una interrupción por hardware se puede presentar en cualquier punto del programa que se está ejecutando, debe ser almacenada toda la información importante para el programa antes de efectuar la rutina de interrupción y al finalizar restablecer los valores. Las interrupciones del 80x86 de la PC están numeradas y la dirección de inicio de las rutinas de servicio correspondientes tienen una dirección establecida, de donde pueden ser leídos al ocurrir una interrupción. Para convertir un programa de Fifth en una rutina de servicio para cierta interrupción se requieren instrucciones para el almacenamiento y restablecimiento, así como inicialización y la dirección de inicio debe colocarse en su lugar de memoria correspondiente. Todo esto se realiza con la palabra especial 'IRQ#', la cual se coloca como función de control antes del texto de la rutina de servicio, seguida del número de la interrupción:

```
irq#8
        ...
        ... rutina de servicio
        ...
>:int8
```

Como la mayoría de los sistemas basados en el 8086, la PC utiliza un controlador de interrupciones i8259 con 8 entradas para interrupciones por hardware, 8 a 15 para la PC. Programando el registro del controlador (Dirección E/S $21) pueden ser activadas o desactivadas individualmente las entradas. La interrupción se activa cuando su bit correspondiente es 1. Esto no significa que se efectuará una interrupción, esto dependerá de la programación del circuito conectado a la entrada de la interrupción. El controlador debe recibir en cada rutina de interrupción un comando EOI ($20 en la dirección E/S $20).

Mientras se ejecuta una rutina de interrupción el microprocesador no podrá atender a otra. Para que la computadora reaccione rápido a las interrupciones es necesario mantener pequeñas estas rutinas. No se deben incluir funciones del sistema operativo como entrada y salida de datos. Para acciones complicadas iniciadas por hardware se utilizan procesos de alta prioridad, los cuales pueden ser activados por pequeñas rutinas de interrupción.

Fifth instala su propia rutina de interrupción para el reloj (timer) del sistema, el cual puede activar funciones definidas por el usuario cada vez que ocurra una interrupción del timer. La función definida por el usuario contiene una instrucción especial de espera y es iniciado como proceso, como se describe en la siguiente sección. También existen rutinas establecidas para los puertos serie de manera que no se requiere una programación adicional.

8.2 Procesos paralelos

P#, PID, SWITCH, MTSWI, NSWI:, BRK:

Un proceso es una serie de tareas que el microprocesador debe realizar. Un proceso inicia con la primer instrucción del programa y termina cuando el microprocesador ejecuta la última instrucción. En un período de tiempo determinado se describe un proceso a través del lugar del programa alcanzado y del uso de memoria incluyendo los registros del microprocesador y las pilas de datos y direcciones de retorno. Un proceso puede 'congelarse' conservando estos datos y liberar al microprocesador para que pueda ejecutar otro programa. Esto sucede, por ejemplo, cuando ocurre una interrupción. El proceso 'congelado' puede continuar a partir del punto alcanzado restableciendo los registros y pilas correspondientes del microprocesador.

La serie de instrucción que se efectúan durante una interrupción son un proceso, el cual inicia con la primer instrucción de la rutina de interrupción y termina tras su ejecución. Mientras la rutina de interrupción es ejecutada, el proceso suspendido es 'congelado' y será continuado al termino de la rutina. El microprocesador sólo puede ejecutar un proceso a la vez, pero pueden existir varios procesos congelados. La actividad del microprocesador puede desviarse hacia un proceso suspendido, congelando el proceso actual. La desviación, llamada también cambio de contexto, requiere que el microprocesador modifique sus apuntadores de pila a la pila del nuevo proceso.

En muchas aplicaciones se utiliza únicamente un programa principal y sus rutinas de interrupción sin que existan otros procesos. Como se explicó en el Cap. 4.6, existen ventajas al utilizar más de un proceso. Por ejemplo, en caso de ocurrir una situación de espera en el proceso actual, podría existir un proceso alternativo que continúe utilizando la capacidad del microprocesador. Otra aplicación surge cuando el microprocesador se cambia rápidamente entre varios procesos. Se efectúan entonces varios programas independientes casi simultáneamente (multitarea).

Para poder trabajar con varios procesos simultáneamente se requiere reservar para cada uno su propio espacio de memoria para las pilas. El proceso que se encuentra actualmente activo no debe modificar los datos importantes de los procesos congelados. Este es el caso durante la ejecución de una rutina de interrupción. Al inicio de la rutina se inicializa el espacio de memoria reservada para la pila de datos.

Para definir un programa de Fifth como proceso alterno con espacio propio para pilas, se ejecuta la función de control 'P#n' (n=0..10):

```
p#0

...

... Texto del programa

...

>:nomrep
```

El llamado a 'nomrep' provoca que se inicialice una nueva pila para el siguiente programa y que el proceso congelado conserve la suya. La frontera superior de la pila perteneciente al programa identifica al proceso y es conocida también como identificador del proceso. El identificador de un proceso puede ser conocido ejecutando la función 'PID', aunque su valor normalmente no es conocido. El proceso creado con 'P#0' no recibe ningún parámetro del programa que lo activa. Si después de 'P#' se coloca un número distinto de 0, esa será la cantidad de parámetros que se tomen del proceso que lo activa. La ejecución de un proceso que inicia con 'P#n' donde n=0..10 se asemeja a un objeto con n parámetros sin resultado. Al igual que en los objetos, n puede ser -1, en cuyo caso el número de parámetros se encuentra en la parte superior de la pila.

La posibilidad de colocar un nivel de prioridad en procesos sin transferencia de parámetros se mostró en el Cap. 4.6. Los procesos de Fifth aparecen con distintos niveles de prioridad: alta y baja. Mientras un proceso con alta prioridad esté listo para ejecución

y no se encuentre en situación de espera, tiene preferencia sobre todos los procesos de baja prioridad. Los procesos creados con 'P#n' tienen siempre el nivel de prioridad del proceso que los ejecuta. Para seleccionar un nivel de prioridad independiente, se cuenta con las funciones de control 'P#L' y 'P#H'. Con 'P#L' se crea un proceso de baja prioridad y con 'P#H' uno de alta. Los procesos creados con 'P#L' o 'P#H' no reciben parámetros. El primer valor de la pila al inicio del proceso es igual al PID del programa que lo ejecutó. Cuando un proceso de baja prioridad genere con 'P#H' un proceso de alta prioridad, éste tendrá prioridad sobre su generador. El generador podrá continuar cuando el nuevo proceso termine o esté en situación de espera.

Para congelar un proceso activo y generar un cambio de contexto hacia otro proceso, existe la función especial 'SWITCH'. En caso que sólo existan dos procesos (1 y 2) y que hallan alcanzado la siguiente parte del programa:

(1) "---" SWITCH "+++" SWITCH "***"

(2) "..." SWITCH "..." SWITCH "..."

y 1 esté activo y 2 congelado, en la pantalla aparecería:

---...+++...***

Para cambiar entre procesos de baja prioridad, además de utilizar la instrucción 'SWITCH', se puede efectuar desde un proceso de alta prioridad con la instrucción 'MTSWI'. El sistema Fifth en la PC utiliza un proceso de alta prioridad, el cual es activado por breves períodos de tiempo con la interrupción del reloj del sistema. Este puede ser configurado con el programa KONF para que cada vez que se active ejecute una instrucción 'MTSWI'. Debido a que la interrupción del reloj ocurre periódicamente, todos los procesos de baja prioridad reciben el mismo tiempo de atención del microprocesador simulando un proceso paralelo. Una parte de un programa activo de baja prioridad es suspendido por la interrupción del reloj y así permanece hasta que toque su turno en la cola de procesos en espera o se ejecute una instrucción 'SWITCH'. En caso que largas suspensiones en una parte del programa sean indeseables, se puede

utilizar la función 'NSWI:' la cual impide el cambio de proceso hasta que termine la tarea actual.

Comúnmente las funciones utilizan además de las pilas algunas variables de memoria. En caso que una misma función sea ejecutada por distintos procesos en un cambio de contexto un valor calculado en un proceso sería destruido por otro. Para evitar este problema, se utiliza el tipo de variable 'VAR.P' (variable de proceso). En la implementación para la PC hay 48 variables de proceso de 16 bits, situadas en el espacio de memoria PID...PID+$5f. Cada proceso tiene un conjunto privado de variables P que cambian junto al cambio de contexto.

Con Ctrl-Brk todos los procesos son terminados incluyendo la compilación y es retomada la interrupción inicial del sistema. Los puertos serie son reinicializados. La rutina de terminación (Break) puede inicializar aplicaciones específicas del usuario, se formula la función deseada, se antepone la palabra 'BRK:' y se ejecuta para su instalación.

8.3 Sincronización

RCLT:, SCPKT, I!

Los procesos congelados, listos para ser ejecutados y de distintos niveles de prioridad se encuentran en distintas listas de espera. La instrucción 'SWITCH' dentro del proceso actual causa que éste se coloque al final de la lista de espera de su nivel de prioridad y su lugar es tomado por el proceso que encabeza la lista. En caso que la lista esté vacía, 'SWITCH' no tendrá efecto ya que el proceso colocado al final de la pila será también el único y se activará nuevamente. Además del proceso que está activo y de los procesos listos para ser ejecutados que se encuentran en la lista de espera, pueden existir otros procesos congelados en estado de espera sin estar listos para ser ejecutados.

Los procesos de cualquier nivel de prioridad pueden ser congelados en estado de espera de distintas formas, en donde no se colocan en la lista de procesos listos para ejecución,

sino que se utilizan puntos de sincronización, listas de espera y la instrucción de resecuencialización 'RCLT:'. 'RCLT:' es una función de control que ocasiona que la sección del programa se ejecute y entonces espere hasta que todos los procesos hermanos que contienen 'P#n' hayan terminado. 'RCLT:' toma un parámetro de la pila, el cual controla el paralelismo para la siguiente sección del programa

```
( 2 RCLT:

    ...

    ( P#0 ... )

    ( P#0 ... )

    ( P#0 ... )

... )
```

El parámetro para 'RCLT:' indica el número de procesos hijos que pueden ser construidos en la sección del programa controlada por 'RCLT:'. Cuando se alcanza este número, las siguientes secciones del programa que contengan 'P#n' o sean llamados a funciones se consideran como subprogramas con n parámetros sin resultados. Por tanto, en nuestro ejemplo, sólo se construirán dos procesos hijos. El proceso creador al encontrar el paréntesis final podrá sólo continuar cuando hayan terminado todos los procesos hijos. Si se elige un parámetro muy grande para 'RCLT:', se crearán todos los procesos de la sección del programa, siempre que exista suficiente espacio de memoria (cada proceso en la PC requiere 512 bytes del segmento de datos). Por el contrario, con el parámetro 0 todos los procesos siguientes que inicien con 'P#n' se ejecutarán como subprogramas. Con la función de control 'RCLT:' es posible calcular los elementos de un arreglo con procesos paralelos, aún cuando se encuentren en distintos microprocesadores.

Cuando se tienen que comunicar procesos paralelos para intercambiar datos, es necesaria su sincronización. Para ello, Fifth cuanta con la estructura 'SCPKT' (punto de sincronía). Los puntos de sincronía son declarados igual que las variables:

```
SCPKT a,b,c
```

y deben ser inicializados antes de su uso con una instrucción de escritura:

```
>> a
```

La instrucción de inicialización no toma un parámetro de escritura de la pila. La instrucción de lectura 'a' ocasiona que el proceso se congele hasta que otro proceso efectúe una lectura del mismo punto de sincronía. Entonces ambos procesos pueden continuar, siempre y cuando el proceso congelado esté listo para ejecución y el actual no tenga un nivel de prioridad menor. El siguiente ejemplo muestra como un proceso envía datos hacia otro, donde existe un punto de sincronía 'a' que indica que el receptor está listo para recibir datos y además señala la validez de los datos.

Proceso envía:

```
.. generación de datos
a                       .. espera al receptor
                        .. coloca los datos en memoria
a                       .. reporta la validez
...
```

Proceso receptor: ...

```
a                       .. listo para recibir
a                       .. espera el envío
```

.. toma los datos

Los puntos de sincronía se pueden utilizar también para reactivar un proceso en espera por una interrupción. Para ello se requiere colocar la instrucción de lectura al final de la rutina de interrupción. Un proceso activado de esta manera se asemeja a una continuación de una rutina de interrupción. Mientras se efectúa el proceso las interrupciones están nuevamente activas.

Existe un punto de sincronía especial del sistema utilizado cuando se efectúa un proceso paralelo a la edición y/o compilación. El proceso utiliza la función 'I!', la cual contiene el punto de sincronía y simultáneamente causa que aparezca una 'I' parpadeante en la línea de estado del editor. De esta manera se notifica al usuario del sistema, se efectúan las acciones necesarias y se libera al proceso ejecutando interactivamente la función 'I!'.

8.4 Filas de espera, despliegue de datos y comunicación

QUEUE, PTQ, LISTEN

Cuando varios procesos requieren acceso a un recurso de la computadora como monitor, puerto serie o coprocesador matemático se requiere construir una fila de espera. Fifth cuenta con la estructura especial 'QUEUE'. Las filas son declaradas e inicializadas al igual que los puntos de sincronía:

```
QUEUE       L:      .. Declaración


>>L:                .. Inicialización
```

La sección del programa utiliza a la fila como una función de control y a través de ella hará uso de los recursos.

```
... (L: ... operaciones del puerto serie ... ) ...
```

El proceso llamado entra en la fila de espera y se activa a la salida del siguiente proceso en espera.

Las funciones de uso de pantalla del sistema pueden ser utilizadas por varios procesos. El sistema se ejecuta como un proceso de baja prioridad colocado en la fila de procesamiento, el programador edita sus datos que son después traducidos y ejecutados. Este proceso requiere de la pantalla. Para que otro proceso paralelo pueda mostrar datos, se puede dividir la pantalla y generar una ventana propia. Pero en caso que dos procesos requieran hacer uso de la misma pantalla se tendría que utilizar una fila de espera.

Al incio, el sistema genera automáticamente una fila de espera especial, que forma parte de la función especial 'PQT', la cual hace que el proceso espera hasta la siguiente interrupción del reloj del sistema. El reloj de la PC tiene una frecuencia de 1/18.2 Hz.

El siguiente ejemplo muestra como procesos de alta prioridad sincronizados con la fila del reloj pueden mostrar datos en la ventana 0 independientemente, mientras se edita en la ventana 15. A través del reloj se asegura que el proceso sólo requiere momentáneamente del microprocesador el proceso incrementa una variable y muestra su valor.

```
var x
p·h                    .. inicio del proceso paralelo
rep:
        ptq            .. instrucción de espera del reloj
        x 1 + >> x
        x ?
>:muestra_x
```

Otro apoyo para los procesos paralelos se encuentra implementado en las rutinas del puerto serie 'SIO.x' y '>> SIO.x', siempre que éstas se encuentren configurados para ser controladas por interrupciones. En caso que una de estas rutinas sea llamada por un proceso de mayor prioridad y ocurra una situación de espera, el proceso se congela y continuará hasta que ocurra una interrupción del puerto serie de manera que, el tiempo de espera pueda ser utilizado por otro proceso. Esto es válido también en el llamado a objetos de otro subsistema.

Si se utiliza un protocolo de comunicación estándar con un subsistema de Fifth (ver 6.4), los llamados a objetos se manejan automáticamente con una fila de espera. De esta manera los procesos pueden hacer llamados a objetos hacia cualquier sistema conectado. El protocolo estándar 0 brinda además una ayuda extra. Si N es el canal más solicitado, con la instrucción 'N LISTEN' se inicia una función de comunicación que maneja los requerimientos de objetos con un proceso paralelo de alta prioridad (con PID configurable). En caso que un sistema externo requiera hacer uso de la pantalla, se puede dividir la pantalla y crear una zona propia que sirva para este propósito. La comunicación de Fifth a través de llamados a objetos armoniza con el uso de procesos paralelos en la PC y subsitemas. A la inversa es posible que un proceso paralelo realice un llamado a un objeto de otro microprocesador y ambos trabajen en paralelo.

9. Archivos COM y mejoras al sistema

Los programas compilados por el usuario en el modo 1 pueden ser almacenados como archivos COM. También es posible, almacenar como mejoras al sistema las nuevas funciones del modo 0, incluyendo las que afectan al compilador.

9.1 Creación de archivos COM

BUILD1

Para almacenar los programas compilados en Fifth como archivos COM, existe la instrucción especial BUILD1. Se utiliza en el modo 1 de la siguiente forma:

comando BUILD1 Nombre

El 'comando' es cualquier dato compilable en el modo 1, el cual se convertirá con BUILD1 en subprograma e instrucción inicial del archivo NOMBRE.COM. Normalmente se coloca el llamado a una función definida por el usuario. NOMBRE puede ser antecedido por una ruta en disco. Al ejecutar NOMBRE se ejecuta el comando y se termina el programa. La ejecución del programa puede ser abortada, igual que en el modo interactivo, presionando Ctrl-Brk.

BUILD1 almacena sólo el código almacenado en el segmento correspondiente del modo 1. El editor de texto y el compilador no quedan disponibles para el programa de aplicación. Debido a que la función de comunicación, las operaciones en disco para datos y programa son funciones del modo 1, el archivo COM puede proveer datos y código a varios microprocesadores obteniendo una aplicación multiprocesador.

'BUILD1' permite el uso de un parámetro numérico, que indica la dirección del segmento de datos que se utilizará. Si se utiliza el siguiente parámetro

..... BUILD1 $8000 NAME

al ejecutarse el archivo '.COM' el segmento de datos se colocará lo más junto posible al código del programa para utilizar el mínimo de memoria. El segmento de código se sitúa relativo al segmento de datos a diferencia de lo que sucede durante la compilación interactiva, es por eso que no se pueden utilizar arreglos en direcciones del código. Los arreglos de constantes pueden construirse fácilmente con las instrucciones 'BTBL' y 'WTBL'.

El siguiente ejemplo muestra como construir un programa de terminal 'TERM.COM' (ver 4.4).

```
rep: rcnt1 c( sio1 out )q key c( >> sio1 )q build1 term1
```

Otros ejemplos de funciones desarrolladas con 'BUILD1' son los programas 'KONF.COM' y 'PC.COM'.

Para leer y analizar la línea de comandos del DOS desde un programa COM se coloca la línea en la memoria de datos de Fifth y se utilizan funciones como 'RDPAR', 'TOKEN', etc.

9.2 Mejorando el compilador

BUILD, INI, SETMD, DEFMD

Versiones mejoradas del compilador se pueden almacenar como archivos EXE con el comando BUILD del modo 0. BUILD almacena todo el código del modo 0, 1 y del diccionario. Además, BUILD guarda las instrucciones anteriores al comando BUILD como subprograma y los convierte en el programa de inicialización de la nueva versión. BUILD es utilizado de manera similar a BUILD1 de la siguiente manera:

```
...

... Comando de inicialización

...
```

build NOMBRE

en donde "NOMBRE" será el nombre del nuevo programa EXE. Después de ser almacenado automáticamente se ejecuta el nuevo programa de inicialización. El comando de inicialización puede ser cualquier comando del modo 0. Puede ser el llamado a una funciones definida con anterioridad o contener un ciclo sin fin, de manera que la ejecución del archivo EXE es exacta a la ejecución de este comando. Después de realizar el comando de inicialización o de interrumpirlo con Ctrl-Brk entra el sistema en el modo de edición. El comando 'LIST' muestra todas las palabras definidas antes de ejecutar la instrucción 'BUILD', las cuales permanecen como elementos para desarrollo de nuevos programas. El nuevo diccionario esta protegido contra el uso de 'NEW', que sólo afectará a las nuevas funciones definidas en el sistema.

En caso de que el nuevo compilador requiera de variables del sistema con un valor predefinido, se deben almacenar sus valores en el programa de inicialización.

El programa de inicialización puede ser ejecutado con el nombre 'INI' del modo 0. En caso que se desee utilizar el programa de incialización de la versión actual al crear una nueva, se debe colocar el comando 'INI' antes de la instrucción 'BUILD'. Otra función utilizada con frecuencia con este comando es 'SETMD', la cual, toma un número de la pila y coloca al compilador en este modo.

'BUILD' almacena sólo las nuevas palabras definidas y el código correspondientes de los modos 0 y 1, pero no el código para subsistemas de otros modos. En caso de que estos se hayan almacenado como archivos de datos, se puede utilizar la instrucción 'LDF' en el comando de inicialización. De esta manera el nuevo compilador puede automáticamente alimentar a otros microprocesadores y ejecutar una aplicación o iniciar la programación interactiva.

La creación de nuevas versiones del sistema tiene muchas aplicaciones. Un uso muy sencillo consiste en configurar los puertos de comunicación. También existe la

posibilidad de crear grandes sistemas sin necesidad de trabajar constantemente con el texto completo del programa.

Una importante utilización de la capacidad de creación de nuevas versiones, consiste en poder obtener un nuevo modo de programación y las funciones de Fifth correspondientes para la generación de código para un nuevo microprocesador. La incialización de un nuevo modo de compilación se realiza con la instrucción 'DEFMD', teniendo como canal de comunicación con el nuevo sistema a uno de los puertos serie.

La implementación en un nuevo microprocesador requiere la creación de una gran cantidad de funciones para generación de código y un conocimiento preciso del nuevo sistema y del modo de trabajo de Fifth. Por el contrario es muy sencillo, utilizar las funciones de un modo ya definido en otro. Por ejemplo, para tener otro sistema en el modo 2 del microprocesador 8086, que tiene la misma programación que el modo 1 se ejecuta:

```
DEFMD 2 -1  .. -N toma la generación de código del modo N
```

En este ejemplo se debe modificar la función '? ', que en el modo 1 se utiliza como llamado a un subprograma, para convertirla en un llamado a un objeto del modo 1, cuando no se tenga la capacidad de desplegar datos de manera local. Para el nuevo modo se toma sólo el código generado para el diccionario básico, como lo mostraba el comando 'LIST'.

10. Conclusiones y recomendaciones

Fifth, a pesar de ser un lenguaje de programación poco conocido, cumple con los requisitos de un lenguaje de alto nivel sin perder el nivel de control de un lenguaje ensamblador. Con la descripción aquí dada se pretende brindar un panorama general del lenguaje y sus diversos campos de aplicación.

Fifth nació como un lenguaje de programación sencillo y de fácil manejo y es la forma en que debe permanecer. No está diseñado para trabajar en ambientes sofisticados como Windows, pero esto no es una desventaja, es una característica fundamental del lenguaje. Tratarlo de llevar a estos ambientes de programación haría que desapareciera su estructura sencilla, como ocurrió con C que se convirtió en un lenguaje de programación confuso, dando paso o otro tipo de lenguajes más aptos para la plataforma Windows.

En esta tesis se trataron los aspectos básicos del lenguaje tratando de mostrar la herramienta. Se sugiere como complemento a este estudio la profundización en la programación en paralelo y en tiempo real. Existe una gran cantidad de literatura en las dos áreas y en ambos temas es posible escribir estudios especializados.

Bibliografía

Dettman, Terry R.

DOS Programmers reference

QUE

1988

Mayer-Lindenberg, Frederik

Die Fifth Programmiersprache

1991

Mayer-Lindenberg, Frederik

A short overview of Fith

Reporte técnico

1997

Swan, Tom

Mastering Turbo Assembler

Hayden Books

1989

The application engineering staff of Analog Devices, DSP Division

Digital Signal Processing Applications

Prentice Hall

1990

Glosario de términos

Apuntador: Tipo especial de variable que en contiene la dirección de alguna variable o dispositivo. Hacia esa dirección que contiene se dice que apunta.

Arreglo: Conjunto de variables del mismo tipo que tienen un sólo nombre. Cada una de estas variables se conoce como elemento del arreglo y cada elemento se diferencía por un número (índice del arreglo).

Clase: Conjunto de términos que tienen un objetivo similar y se agrupan bajo un nombre para mayor claridad.

Comando: Instrucción del lenguaje que puede ser ejecutada.

Compilador: Inerprete de un lenguaje de computación que traduce a lenguaje máquina para que el microprocesador pueda ejecutar las instrucciones.

Concepto: Es la definición de un elemento del lenguaje de programación.

Coprocesador matemático: Circuito integrado que trabaja en conjunto con el microprocesador. Es el encargado de efectuar las operaciones matemáticas de alto nivel.

Estructura de control: Se utiliza para manejar el flujo que sigue un programa durante su ejecución. Existen diversas estructuras de control que se pueden resumir en condiciones y repeticiones.

Función: Es una sección del programa que tiene una tarea específica. Se conoce también como subprograma en otros lenguajes de programación.

IPU: (Interaktiv Programmierumgebung) Ambiente de programación interactivo.

Microprocesador: Circuito integrado con la capacidad de ser reprogramado para ejecutar series de operaciones sin necesidad de modificar el circuito impreso.

MS-DOS: Sistema operativo de disco de MicroSoft. Sistema operativo utilizado en las computadoras compatibles.

Notación algebráica: Estilo de escritura utilizado comunmente en álgebra y matemáticas.

Notación postfija: Estilo de escritura en donde se colocan primero los datos y posteriormente las operaciones que se efectúan con ellos.

Número de punto flotante: Número real que contiene una parte fracional y se almacena siguiendo el estándar de la IEEE para estos datos.

Número entero: Número real que no contiene parte fraccional.

Objeto: Es una función especial basada en la transparencia entre diferentes microprocesadores.

Palabra: Término utilizado para representar 32 bits de datos ó dos bytes.

Parámetro: Define un elemento utilizado por un programa para enviar un valor a una subrutina.

Pila de datos: Espacio utilizado para almacenar parámetros, resultados y datos en general. El concepto de pila se utiliza para designar que la última entrada es la primer salida.

Sistema multiprocesador: Arreglo de varios microprocesadores conectados entre sí con la finalidad de utilizar sus características particulares y potencia de cómputo para lograr un objetivo común.

Sistema objetivo: Sistema para el cual se crea el programa actual.

Subprograma: Es una sección del programa que es ejecutada en una o más ocaciones por el programa principal. Se utiliza cuando una sección del código del programa se utiliza más de una vez o para dar mayor claridad al programa.

Término: Son los datos que se colocan en la pila. Pueden ser constantes, variables o el resultado de la ejecución de alguna función.

Variable local: Tipo especial de variable que existe sólamente dentro de la función en que es definida.

Variable: Es una localidad de memoria referenciada con un nombre propio que puede cambiar su contenido o valor almacenado.

Resúmen autobiográfico

Nombre:	Francisco Javier de la Garza Salinas
Nombre de los padres:	Francisco Javier de la Garza Pérez
	Hermila Salinas Villarreal
Lugar y fecha de nacimiento:	Monterrey, N.L.
	2 de diciembre de 1966
Grado de escolaridad:	Ingeniero en Control y Computación
	Facultad de Ingeniería Mecánica y Eléctrica
	Universidad Autónoma de Nuevo León
Especialidad:	Sistemas Distribuidos
	Universidad Técnica de Hamburgo, Alemania
Docencia:	Facultad de Ingeniería Mecánica y Eléctrica
	Universidad Autónoma de Nuevo León
	1988 a la fecha

Laboral:	Facultad de Ingeniería Civil
	Universidad Autónoma de Nuevo León
	Departamento de Sistemas. 1988 a 1993.
	Nylon de México, S.A. de C.V.
	Manetnimiento eléctrico/instrumentos 1995-1996.
	Controles y Válvulas, S.A. de C.V.
	Sistemas de automatización. 1996 a la fecha.
Grado que deseo obtener:	Maestro en Ciencias de la Ingeniería Eléctrica
	con Especialidad en Electrónica
Nombre de la tesis:	El compilador interactivo Fifth