HEWLETT PACKARD

RPLMAN

Guia de Programación RPL

Sobre este Documento:

El presente documento es una edición no oficial de la “Guia de Programación RPL” de Hewlett-Packard (RPLMAN.doc) traducida al español. El documento fuente utilizado está disponible en la página web de Eric Rechlin, específicamente en la siguiente dirección:

www.hpcalc.org/hp48/docs/programming/rplmanes.zip

El documento original RPLMAN.doc fue escrito en la época de las calculadoras HP48S/SX por los expertos que desarrollaron el sistema, y acompañaba a una serie de herramientas para PC suministradas por Hewlett-Packard para crear programas de su potente calculadora. Estas herramientas llamadas “HP 48 Software Development Tools” tambien están disponibles en la red:

www.hpcalc.org/hp48/pc/programming/tools.zip

En la actualidad, este documento sigue siendo de interes para los programadores, principalmente porque explica cómo funciona el sistema operativo RPL, tema sobre el cual no hay mucha información disponible en la red.

En esta edición intentamos, fundamentalmente, hacer más cómoda la lectura, a fin de motivar el estudio de su valioso contenido.

Marcos Navarro
ing.mnavarro@hotmail.com
ENERO, 2012

NOTAS - N.T.> significa Nota del Traductor
- No se traducen las palabras que forman parte del RPL del Sistema

CONTENIDO

GUIA DE PROGRAMACION RPL

1. Introducción

La calculadora HP 48 se diseñó para ser un bloc matemático personalizable para ser usado por estudiantes y profesionales de campos técnicos. En muchos aspectos es un descendiente de la HP 41, proporcionando una capacidad de computación mucho más amplia y sofisticada que la HP 41 pero conservando su orientación RPN/una-función-por-tecla.

La HP 48 usa la, así llamada, arquitectura Saturno, así denominada por el nombre del código de la CPU original diseñada para el ordenador de mano HP 71B. También usa un sistema operativo/lenguaje a la medida llamado RPL, que se diseñó para proporcionar capacidades de matemática simbólica, ejecutándose desde ROM en un entorno con RAM limitada (todavía hoy, es el único sistema simbólico que se puede ejecutar en ROM). La combinación de hardware y firmware especializados hace relativamente difícil el desarrollo de software de aplicaciones para la HP48 y, en consecuencia, la HP48 no esta situada como un vehículo fundamental para aplicaciones externas. La orientación del producto y de su lenguaje de programación del usuario es hacia la personalización sencilla por el usuario principal.

A pesar de estas barreras, el precio y la configuración física de la HP48 la hacen una plataforma deseable para muchos desarrolladores de software, especialmente aquellos que quieren llegar a los clientes de los mercados normales de la HP48. El lenguaje de usuario es adecuado para programas sencillos, pero para sistemas elaborados, la deliberada protección de errores y otros gastos pueden dar lugar a penalizaciones substanciales comparado con los programas que usan todo la gama de llamadas del sistema.

En este documento, proporcionaremos una descripción del diseño y las convenciones del lenguaje RPL. Este material debería proporcionar suficientes detalles para permitir la creación de programas RPL y otros objetos, usando las herramientas asociadas de compilación para IBM-PC. Se incluye documentación de un gran número de objetos del RPL del sistema que son utilidades útiles para el desarrollo de programas.

2. Principios de RPL

El siguiente material esta extraído de "RPL: Un Lenguaje de Control Matemático" por W.C. Wickes, publicado en "Entornos de Programación", Instituto para la Investigación de Forth Aplicado, Inc., 1988)

2.1 Orígenes

En 1984, se inició un proyecto en la división de Hewlett-Packard en Corvallis para desarrollar el software de un nuevo sistema operativo para el desarrollo de una línea de calculadoras y dar soporte a una nueva generación de hardware y software. Anteriormente todas las calculadoras HP se implementaron enteramente en lenguaje ensamblador, un proceso que se iba haciendo cada vez más pesado e ineficiente a medida que aumentaba la memoria de las calculadoras. Los objetivos para el nuevo sistema operativo fueron los siguientes:

- Proporcionar control de la ejecución y manejo de la memoria, incluyendo memoria conectable;
- Proporcionar un lenguaje de programación para un rápido desarrollo de prototipos y aplicaciones;
- Para soportar un variedad de calculadoras de negocio y técnicas;
- Para ejecutarse idénticamente en RAM y ROM;
- Para minimizar el uso de memoria, especialmente RAM;
- Para ser transportable a varias CPU's;
- Para ser extensible; y
- Para soportar operaciones de matemática simbólica.

Se tuvieron en cuenta varios lenguajes y sistemas operativos ya existentes pero ninguno cumplía con todos los objetivos del diseño. Por consiguiente se desarrolló un nuevo sistema, el cual mezcla la interpretación entrelazada de Forth con el enfoque funcional de Lisp. El sistema operativo resultante, conocido de modo no oficial como RPL (de Reverse-Polish Lisp), hizo su primera aparición pública en junio de 1986 en la calculadora Business Consultant 18C. Más tarde, RPL ha sido la base de las calculadoras HP-17B, HP-19B, HP-27S, HP-28C y HP-28S y HP 48S y HP 48SX. La HP-17B, 18C y la 19B se diseñaron para aplicaciones de negocios; ellas y la calculadora científica HP-27S ofrecen una lógica de cálculo "algebraica" y el sistema operativo subyacente es invisible para el usuario. Las familias HP 28/HP 48 de calculadoras científicas usan una lógica RPN y muchas de la facilidades del sistema operativo están disponibles directamente como comandos de la calculadora.

2.2 Control Matemático

Los objetivos oficiales del sistema operativo listados arriba se fueron mezclando, a lo largo del ciclo de desarrollo del RPL, con un objetivo menos formal de creación de un lenguaje de control matemático que extendería la facilidad de uso y la naturaleza interactiva de una calculadora al reino de las operaciones de la matemática simbólica. En este contexto, una calculadora se distingue de un ordenador por:

- tamaño muy compacto;
- "encendido instantáneo" --sin calentamiento o carga de software;
- teclas dedicadas para las operaciones comunes en lugar de teclados qwerty.
- "acción instantánea" cuando se pulsa una tecla de función.

La HP-28, que fue desarrollada por el mismo equipo que creó el sistema operativo RPL, fue la primera realización de este objetivo de fondo; la HP 48 es la última y más madura implementación.

Buena parte del diseño del RPL se puede derivar a partir de considerar la manera en que se evalúan ordinariamente las expresiones matemáticas. Considera, por ejemplo, la expresión

1+2seno(3x)+4

Como sabe cualquier entusiasta del RPN, la expresión aquí escrita no se corresponde con en su orden de izquierda a derecha con el orden en el que una persona o una máquina podría realmente llevar a cabo el cálculo. Por ejemplo, la primera suma se tiene que posponer hasta que se ejecuten varios pasos. Reescribiendo la expresión en la forma RPN, obtenemos una representación que también es ejecutable en el orden escrito:

1 2 3 x * seno * + 4 +

Para traducir esta secuencia a un lenguaje de control, necesitamos formalizar varios conceptos. Primero, usamos el término genérico objeto para referirnos a cada paso de la secuencia, tales como 1, 2, o seno. Incluso en este sencillo ejemplo, hay tres clases de objetos:

1. Objetos de Datos. La ejecución de un objeto tal como 1, 2 o 3 en el ejemplo, simplemente devuelve el valor del objeto.

2. Nombres. El símbolo x debe ser el nombre de algún otro objeto; cuando se ejecuta x, el objeto nombrado substituye al símbolo.

3. Procedimientos. Los objetos tales como *, seno y + representan operaciones matemáticas, que se aplican, por ejemplo, a objetos de datos para crear nuevos objetos de datos.

El concepto de objeto está estrechamente ligado al concepto de ejecución, en el que se puede pensar como la activación de un objeto. Un objeto individual se caracteriza por su tipo de objeto, que determina su acción cuando se ejecuta y su valor, que lo distingue de otro objeto del mismo tipo.

La evaluación de una expresión en estos términos se convierte en la ejecución secuencial de una serie de objetos (los objetos que representan la forma RPN de una expresión). Son necesarias dos construcciones para hacer la ejecución coherente: una pila de objetos y un puntero del intérprete. La primera construcción proporciona un lugar del cual los objetos procedimiento pueden tomar sus argumentos y al cual pueden devolver sus objetos resultado. Una pila LIFO (Last In, First Out = Ultimo en Entrar, Primero en Salir) como la usada en Forth es ideal para este propósito y se incluye una pila así en RPL. El puntero del intérprete no es más que un contador de programa que indica el siguiente objeto a ejecutar. El puntero del intérprete se debe distinguir del contador de programa de la CPU, que indica la siguiente instrucción de la CPU.

Una expresión matemática considerada como una secuencia de objetos sugiere una clasificación adicional de objetos en atómicos o compuestos. Un objeto atómico es un objeto que no se puede separar en objetos independientes; son ejemplos un sencillo objeto de datos como 1 o 2 o quizás un objeto como * o + que se implementan normalmente en lenguaje máquina. Un objeto compuesto es una colección de otros objetos. En Forth, una palabra secundaria es un ejemplo de objeto compuesto. RPL proporciona al menos tres tipos de objetos compuestos: secundarios, que son procedimientos definidos como una secuencia de objetos sin restricción; simbólicos, que son secuencias de objetos que deben equivaler lógicamente a expresiones algebraicas; y listas, que contienen objetos reunidos para cualquier propósito lógico que no sea la ejecución secuencial.

El punto final en esta breve derivación de las matemáticas-a-RPL es observar que la definición de objetos compuestos nos lleva al concepto de interpretación entrelazada y de una pila de retornos. Esto es, es fácil imaginar en el ejemplo, que el nombre del objeto x podría representar un objeto compuesto que a su vez representa otra expresión. En ese caso, se podría esperar que la ejecución de x haga saltar al puntero del intérprete a la secuencia de objetos referenciada por x, mientras que la posición del objeto que sigue a x en la original se almacena de modo que la ejecución pueda luego volver allí. Este proceso debe poderse repetir indefinidamente, por lo que RPL proporciona una pila LIFO para los objetos de retorno.

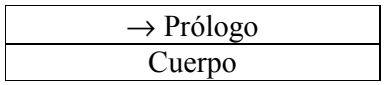
La introducción precedente podría, en algunos aspectos, haber sido también una introducción para la derivación de Forth, si se ignoran cuestiones de aritmética de punto flotante versus enteros. En concreto, ambos sistemas usan la interpretación entrelazada y una pila de datos LIFO para el intercambio de objetos. Sin embargo, hay varias diferencias importantes entre Forth y RPL:

- RPL soporta la ejecución entrelazada tanto indirecta como directa de una manera completamente uniforme.
- RPL soporta la colocación dinámica de sus objetos.
- El código RPL es, en general, completamente relocalizable.

2.3 Definiciones Formales

Esta sección presentará las definiciones abstractas de RPL que son independientes de cualquier CPU o implementación concretas.

La estructura fundamental en RPL es el objeto. Cualquier objeto está formado por un par: la dirección del prólogo y el cuerpo del objeto.



Las dos partes están contiguas en memoria con la parte de la dirección del prólogo en la memoria inferior. La dirección del prólogo es la de una rutina en código máquina que ejecuta el objeto; el cuerpo son datos usados por el prólogo. Los objetos se clasifican por tipo; cada tipo se asocia con un prólogo único. Así los prólogos sirven para el doble propósito de ejecutar un objeto y de identificar su tipo.

Un objeto o es atómico o es compuesto. Un objeto compuesto o es nulo o no es nulo; un compuesto que no es nulo tiene una cabeza que es un objeto y una cola que es un compuesto.

Además de ejecutarse, todos los objetos RPL se pueden copiar, comparar, incluir en objetos compuestos y saltarse. La última propiedad implica que la longitud en memoria de cualquier objeto está predeterminada o se puede calcular a partir del objeto. En los objetos atómicos tales como los número reales, el tamaño es fijo. En los objetos atómicos más complicados como una formación numérica, el tamaño se puede calcular a partir de las dimensiones de la formación que se almacenan en el cuerpo del objeto formación. (Las formaciones RPL no son compuestos -- los elementos no tienen prólogos individuales y por tanto, no son objetos). Los objetos compuestos pueden incluir un campo longitud o pueden terminar con un objeto marcador.

Un puntero es una dirección en el espacio de memoria de la CPU y puede ser un puntero de posición o un puntero de objeto. Un puntero de posición direcciona cualquier parte de la memoria, mientras que un puntero de objeto apunta a un objeto, específicamente al puntero de posición del prólogo al principio de un objeto.

RPL requiere, además de los contadores de programa de la CPU, cinco variables para su funcionamiento fundamental:

- El puntero del intérprete I.
- El puntero del objeto actual O.
- El puntero de la pila de datos D.
- El puntero de la pila de retornos R.
- La cantidad de memoria libre M.

En la definición más general de RPL, I es un puntero de objeto que apunta al objeto compuesto que está en la cima de una pila de objetos compuestos llamada "runstream". R apunta al resto de la pila "runstream". En las implementaciones prácticas, esta definición se simplifica permitiendo que I apunte a cualquier objeto embebido en un compuesto, mientras que R es un puntero de posición que apunta a la cima de una pila de punteros de objeto, cada uno de los cuales apunta a un objeto embebido.

En RPL es fundamental que los objetos se puedan ejecutar directa o indirectamente con resultados equivalentes. Esto significa que un objeto se puede representar en cualquier sitio por un puntero al objeto así como también por el objeto mismo.

2.4 Ejecución

La ejecución de un objeto RPL consiste en la ejecución por la CPU del prólogo del objeto, donde el código del prólogo puede acceder al cuerpo del objeto mediante el puntero de objeto O. La ejecución del puntero de objeto es la ejecución por la CPU del destinatario del puntero. Esta ejecución interpretativa se controla por el interpretador interno o bucle interno, quién determina la secuencia de objetos/punteros de objeto a ejecutar.

Los objetos RPL se ordenan por sus propiedades generales de ejecución en tres clases:

- Los objetos que simplemente se devuelven ellos mismos a la pila de datos cuando se ejecutan se llaman objetos de la clase datos. Son ejemplos los números reales, las cadenas de caracteres y las formaciones.

- Los objetos que sirven de referencia a otros objetos se llaman objetos de la clase identificador. RPL define tres tipos de objetos de clase identificador: identificador (nombre global), identificador temporal (nombre local) y puntero ROM (nombre XLIB)

- Los objetos que contienen cuerpos a los que puede pasar el flujo de la ejecución se llaman objetos de la clase procedimiento. Hay tres tipos de objetos de la clase procedimiento: programas (también llamados "secundarios" o una "definiciones colon" en la terminología Forth), objetos código y objetos código primitiva.

El bucle interno RPL y el diseño de prólogos permite la ejecución directa e indirecta de objetos indistintamente (nota: hay una solicitud de patente por los conceptos descritos a continuación). El bucle interno consta del siguiente pseudo-código:

```
O = [I]
I = I + delta
PC = [O] + delta
```

donde [x] significa el contenido de la dirección x y "delta" es el tamaño de una dirección de memoria. Este bucle es el mismo que en Forth, excepto que la ejecución de la CPU salta a [O]+delta en vez de a [O]. Esto es así porque todos los prólogos de RPL empiezan con su propia dirección, lo que es la característica que hace posible la ejecución directa además de la indirecta. Los prólogos se parecen a esto:

```
PROLOG   ->PROLOG                              Auto-dirección
         IF O + delta != PC THEN GOTO REST     Test para ejecución directa
         O = I - delta                         Corrige O
         I = I + len                           Corrige I
REST     (resto del prólogo)
```

Aquí "len" es el tamaño (longitud) del cuerpo del objeto.

Cuando se ejecuta un objeto directamente, el bucle interno no pone O ni I correctamente. Sin embargo, el prólogo sabe si se está ejecutando directamente comparando la dirección del PC con O y puede actualizar las dos variables en consecuencia. El prólogo es también responsable de preservar la interpretación entrelazada incluyendo un retorno al bucle interno al finalizar.

Esta interpretación flexible es intrínsecamente más lenta que la ejecución sólo-indirecta (como Forth), debido al gasto necesario para hacer la comprobación de directo/indirecto. En las implementaciones prácticas del RPL, es posible desplazar casi por completo este gasto al caso de la ejecución directa, de modo que la penalización en la ejecución del caso indirecto sea despreciable, incluyendo objetos primitivas en lenguaje ensamblador que nunca se ejecutan directamente. El truco está en reemplazar el último paso del bucle interno con PC = [O] al modo Forth y, para los prólogos de los objetos ejecutables directamente, reemplazar la auto-dirección al principio de cada prólogo con una sección de código ejecutable de tamaño igual a "delta". Los códigos compilados de esta sección deben también ser la dirección de un meta-prólogo que maneje el caso de la ejecución directa. En las implementaciones con la CPU Saturno, la sección de código consiste en la instrucción M=M-1 (decrementar la memoria disponible es común a todos los prólogos de objetos ejecutables directamente) más una instrucción NOP de relleno para alcanzar el tamaño "delta".

La virtud de la ejecución directa es que permite el manejo sencillo de los objetos sin nombre que se crean durante la ejecución. En el curso de las manipulaciones algebraicas simbólicas es común crear, usar y descartar cualquier número de resultados temporales intermedios; la necesidad de compilar y almacenar estos objetos con alguna forma de nombre para la referencia indirecta y luego descompilarlos para recuperar memoria, haría inmanejable todo el proceso. En RPL tales objetos se pueden colocar en la pila, copiar, embeber en objetos compuestos, ejecutar y borrar. Por ejemplo, un objeto compuesto que representa la expresión x + y se puede añadir a un segundo objeto que representa 2z, devolviendo el objeto resultante x + y + 2z; más aun, cualquiera de estos objetos se podrían incluir en un objeto programa para llevar a cabo la adición repetidamente.

Aunque RPL es fundamentalmente un lenguaje sufijo sin sintaxis en el que los procedimientos toman sus argumentos de la pila y devuelven los resultados a la pila, proporciona operaciones que operan en el "runstream" para proporcionar operaciones prefijo y para permitir alteraciones en la ejecución entrelazada normal. La primera entre las operaciones en el "runstream" es la operación de encomillar, que toma el siguiente objeto del "runstream" y lo sube (coloca) en la pila de datos para posponer su ejecución. El propósito de esta operación es similar al QUOTE de Lisp, pero toma su nombre RPL, ' (tick), de su equivalente Forth. RPL también tiene operaciones para subir y bajar objetos de la pila de retornos. (Sin embargo, los parámetros de los bucles DO loop no se almacenan en la pila de retornos, sino que se usa en su lugar un entorno especial).

2.4.1 EVAL Un objeto en la pila de datos se puede ejecutar indirectamente mediante la palabra RPL EVAL, que baja un objeto de la pila y ejecuta su prólogo. El objeto del sistema EVAL se debe distinguir del comando del RPL de Usuario EVAL. Este último es equivalente al EVAL del sistema excepto por las listas, los objetos simbólicos y los objetos etiquetados. Con un objeto etiquetado el EVAL de Usuario ejecuta el objeto contenido en el cuerpo del objeto etiquetado. Con las listas y los simbólicos, el EVAL de Usuario los envía a la palabra del sistema COMPEVAL, que ejecuta el objeto como si fuera un programa (ver más adelante).

2.4.2 Objetos de la Clase Datos Los tipos de objetos en esta clase son:

Objeto Entero Binario (nota: el entero binario del RPL de Usuario es realmente un objeto cadena hex en términos del RPL del sistema).
Objeto Real
Objeto Real Extendido
Objeto Complejo
Objeto Complejo Extendido
Objeto Formación
Objeto Formación Encadenada
Objeto Cadena de Caracteres
Objeto Cadena Hex
Objeto Carácter
Objeto Gráfico
Objeto Unidad
Objeto Lista
Objeto Simbólico ("objeto algebraico")
Objeto Datos de Biblioteca
Objeto Directorio
Objeto Etiquetado
Objeto Externo

Todos los objetos de la clase de datos tienen la propiedad que, cuando se ejecutan, simplemente se colocan ellos mismos en la cima de la pila de datos.

2.4.3 Objetos de la Clase Identificador Los tipos de objetos en esta clase son:

Objeto Puntero ROM (nombre XLIB)
Objeto Identificador (nombre global)
Objeto Identificador Temporal (nombre local)

Los objetos en la clase identificador comparten la propiedad de servir para proporcionar referencias para otros objetos. Los objetos identificadores representan la resolución de las variables globales y los Objetos Punteros ROM representan la resolución de los comandos almacenados en bibliotecas. Los objetos identificadores temporales, por otro lado, proporcionan referencias para los objetos temporales en entornos temporales.

La ejecución de un objeto puntero ROM (por el prólogo DOROMP) conlleva localizar y luego ejecutar la parte del objeto ROM-WORD referenciada. La no localización es una condición de error.

La ejecución de un objeto identificador (por un prólogo DOIDNT) conlleva localizar y luego ejecutar la parte del objeto variable global referenciada. La no localización devuelve el objeto identificador mismo.

La ejecución de un objeto identificador temporal (por el prólogo DOLAM) conlleva localizar el objeto temporal referenciado y subirlo a la pila de datos. La no localización es una condición de error.

2.4.4 Objetos de la Clase Procedimiento Los tipos de objeto en esta clase son:

Objeto Código
Objeto Código Primitiva
Objeto Programa

Los objetos de la clase procedimiento comparten la propiedad de la ejecutabilidad, esto es, la ejecución de un objeto de la clase procedimiento implica pasar el control al procedimiento ejecutable o al código asociado con el objeto.

Los objetos código contienen secuencias en lenguaje ensamblador para su ejecución directa por la CPU, pero por lo demás son objetos normales, relocalizables. Los objetos código primitiva no tienen ningún prólogo en el sentido usual; el campo dirección del prólogo apunta directamente al cuerpo del objeto que contiene una secuencia en lenguajese pueden ejecutar nunca directamente. Cuando se ejecuta un objeto código o un objeto código primitiva, se pasa el control (ajustando el PC) al conjunto de instrucciones en lenguaje máquina contenidas en el cuerpo del objeto. En el caso de un objeto código primitiva, esta transferencia de control la hace el mecanismo de ejecución (EVAL o el bucle interno) mismo. En el caso de un objeto código, el prólogo pasa el control poniendo el PC al principio de la sección de lenguaje máquina contenida en el cuerpo del objeto. Observa de nuevo que el prólogo de un objeto código primitiva (que es su cuerpo) no necesita contener la lógica para comprobar si la ejecución es directa o indirecta (ni contener código para actualizar I o O) ya que, por definición, nunca se ejecuta directamente.

La ejecución de un programa es la ejecución secuencial de los objetos y punteros de objeto que comprenden el cuerpo del programa. La ejecución es entrelazada por cuanto los objetos en un programa pueden ser a su vez secundarios o punteros a secundarios. Cuando el bucle interno se encuentra un programa embebido lo ejecuta antes de reanudar la ejecución del actual.

El final de un programa está marcado con el objeto SEMI (de "semicolon" (punto y coma en inglés) -- un ";" es el delimitador de terminación reconocido por el compilador RPL para marcar el fin de la definición de un programa). La ejecución de SEMI baja el puntero de objeto de la cima de la pila de retornos y continúa la ejecución en ese punto.

2.4.5 Esquivar Objeto y SEMI Una de las premisas básicas del RPL es que cualquier objeto RPL que se pueda ejecutar directamente (lo que incluye todos los tipos de objetos excepto los objetos código primitiva) deben ser esquivables, o sea, deben tener una estructura que permita esquivarlos. El salto de objetos se da por todo el sistema RPL pero sobre todo durante la ejecución directa por el bucle interno cuando el puntero del intérprete I debe apuntar al siguiente objeto después del que se está ejecutando directamente.

Existen tanto objetos RPL como utilidades para realizar este función de saltarse objetos. Además, los objetos se tienen que esquivar a sí mismos cuando se ejecutan directamente. El mecanismo de esquivar los objetos atómicos es sencillo y claro ya que el tamaño del objeto o bien se sabe o es fácilmente calculable. En el caso de los objetos compuestos (programa, lista, unidad, simbólico) el tamaño no se puede calcular fácilmente y la función de esquivar aquí es algo más complicada, usándose una recursión implícita. Estos objetos compuestos no llevan una información conocida del tamaño o que sea fácilmente calculable y por tanto deben tener un delimitador en la cola, a saber, un puntero de objeto al objeto código primitiva SEMI. Observa que SEMI desempeña una función explícita en el objeto programa (el objeto compuesto clase procedimiento); en los objetos compuestos de la clase datos (listas, unidades y objetos simbólicos), sólo sirve como delimitador de la cola.

2.4.6 Punteros RPL Un puntero es una dirección y puede ser o bien un puntero de posición (de localización) o un puntero de objeto. Un puntero de posición direcciona cualquier segmento del mapa de memoria mientras que un puntero de objeto direcciona específicamente un objeto. Observa que, por ejemplo, la parte de dirección del prólogo de un objeto es un puntero de posición.

2.5 Manejo de la Memoria

La uniformidad de la ejecución directa e indirecta no solo significa que los objetos así como los punteros de objeto se pueden incluir en el flujo de la ejecución, sino también que los punteros de objeto pueden reemplazar lógicamente a los objetos. En particular, las pilas RPL de datos y de retornos son explícitamente pilas de punteros de objeto. Esto significa, por ejemplo, que un objeto en la pila de datos se puede copiar (con el DUP p.ej.) al costo de solo "delta" bytes de memoria, independientemente del tamaño del objeto. Es más, las duplicación y otras operaciones similares de la pila son muy rápidas.

Desde luego, los objetos referenciados en las pilas deben existir en algún lugar de la memoria. Muchos, incluyendo todos los objetos del sistema que proporcionan el manejo del sistema y un lenguaje de aplicación, están definidos en ROM y se pueden referenciar con un puntero sin ninguna otra implicación a tener en cuenta. Los objetos creados en RAM pueden existir en dos lugares. Aquellos que no tienen nombre se almacenan en un área de objetos temporales, donde se mantienen mientras estén referenciados por un puntero en cualquier sitio del sistema (esto implica que si un objeto temporal se mueve, se deben actualizar todos los punteros que haya apuntando a él). Nombrar un objeto es almacenarlo como un par con un campo de nombre en una lista encadenada llamada el área de objetos de usuario. Estos objetos se mantienen indefinidamente, hasta que se borran explícitamente o son reemplazados. Un objeto con nombre se accede por medio de un objeto identificador, que consiste en un objeto con un campo de nombre como su cuerpo. La ejecución de un identificador provoca la búsqueda, en el área de objetos de usuario, de un objeto almacenado con el mismo nombre, que entonces se ejecuta.

Esta resolución en tiempo de ejecución es de por sí más lenta que la resolución durante el tiempo de compilación usada por los objetos ROM, pero permite un sistema dinámico y flexible en el que no importa el orden en que se compilan los objetos.

El proceso de nombrar objetos almacenándolos con nombres en el área de objetos de usuario se ve aumentado por la existencia de los entornos locales, en los que los objetos se pueden unir a nombres (variables lambda) que son locales al procedimiento que se ejecuta actualmente. La unión se abandona cuando el proceso completa su ejecución. Esta característica simplifica las manipulaciones complicadas de la pila permitiendo que los objetos de la pila tengan nombres y se referencien entonces por el nombre dentro del alcance de un procedimiento de definición.

RPL permite que cualquier objeto almacenado en el área de objetos del usuario se pueda borrar sin corromper nada del sistema. Esto requiere ciertas convenciones de diseño:

- Cuando se almacena un objeto RAM en le área de objetos de usuario, se almacena una copia nueva del objeto, no un puntero al objeto.

- Los punteros a objetos RAM no están permitidos en los objetos compuestos. Cuando se crea un objeto compuesto a partir de objetos de la pila, los objetos RAM se copian y embeben directamente en el compuesto. Cuando un objeto almacenado se representa por su nombre en un compuesto, es el objeto identificador el que se embebe, no un puntero de posición como en Forth.

- Si un objeto almacenado está siendo referenciado por cualquier puntero en las pilas en el momento en que se borra, se copia al área de objetos temporales y se actualizan en consecuencia todos los punteros que le apuntaban. Esto significa que la memoria asociada con un objeto no se recupera hasta que se elimine la última referencia a él.

El uso de objetos temporales con múltiples referencias significa que un objeto temporal puede que no se borre de la memoria inmediatamente cuando se elimine solo una referencia a él. En las actuales implementaciones de RPL, no se realiza ninguna recuperación de memoria hasta que el sistema se queda sin memoria (M=0), en cuyo momento se borran todos los objetos no referenciados del área de objetos temporales. El proceso, llamado "recogida de basura" puede durar bastante tiempo, por lo que la ejecución del RPL no procede de modo uniforme.

De la discusión anterior se ve claro que RPL no es tan rápido en general como Forth debido a los gastos extra en la interpretación y al esquema tan elaborado de manejo de memoria. Mientras que la máxima velocidad de ejecución es siempre deseable, el diseño del RPL enfatiza su papel como un lenguaje de control matemático interactivo en el que la flexibilidad, facilidad de uso y la capacidad de manipulación de información de procedimiento son de la máxima importancia. En muchos casos, estos atributos del RPL conducen a una resolución más rápida de los problemas que en Forth, que se ejecuta más rápido pero es más difícil de programar.

RPL también proporciona objetos que son intermedios entre aquellos que están fijos en ROM y aquellos que se mueven en la RAM. Una biblioteca es una colección de objetos, organizados en una estructura permanente que permite una resolución en el tiempo de análisis y en el tiempo de ejecución por medio de tablas incluidas en la biblioteca. Un nombre XLIB es un objeto de la clase identificador que contiene un número de biblioteca y un número de un objeto dentro de la biblioteca. La ejecución de un nombre XLIB ejecuta el objeto almacenado. Las identidades y la posición de las bibliotecas se determinan en la configuración del sistema. Una biblioteca en particular se puede asociar con su propio directorio RAM, de modo que, por ejemplo, una biblioteca puede contener fórmulas permanentes para las que se mantienen en RAM los valores de sus variables.

2.6 RPL de Usuario y RPL del Sistema

No hay ninguna diferencia fundamental entre el lenguaje de programación de la HP 48, al que llamaremos "RPL de usuario" y el "RPL del sistema" en el que está implementada la funcionalidad de la HP 48. Los programas en el lenguaje de usuario son ejecutados por el mismo bucle interno del intérprete que los programas del sistema, con la misma pila de retornos. La pila de datos que se muestra en la HP 48 es la misma que la usada por los programas del sistema. La diferencia entre el RPL de usuario y el RPL del sistema es solo una cuestión de amplitud: el RPL de usuario es un subconjunto del RPL del sistema. El RPL de usuario no proporciona acceso directo a todos los tipos de objetos de la clase de datos disponibles; el uso de los procedimientos incorporados está limitado a aquellos proporcionados como comandos.

Un "comando" es un objeto de la clase procedimiento almacenado en una biblioteca, junto con una cadena de texto que sirve de nombre del comando. El nombre se usa para compilar y descompilar el objeto. Cuando el analizador de la línea de comandos encuentra un texto en la línea de comandos igual al nombre de un comando, compila un puntero de objeto si el comando está en una biblioteca en la ROM permanente de la HP 48. De lo contrario compila el correspondiente nombre XLIB. También, los objetos comando incorporados están precedidos en ROM por un campo de seis nibbles que es el cuerpo de un nombre XLIB. Cuando el descompilador encuentra un puntero de objeto, busca este campo en el ROM al frente del objeto; si encuentra un campo válido, usa entonces la información que hay allí para localizar el texto del nombre del comando a mostrar. Si no, descompila el propio objeto.

Los comandos se distinguen de otros objetos procedimiento por ciertas convenciones en su diseño. Estructuralmente, todos los comandos son objetos programa, siendo el primer objeto dentro de ellos uno de los objetos "de despacho" (de envío) del sistema CK0, CK1&Dispatch, CK2&Dispatch, CK3&Dispatch, CK4&Dispatch y CK5&Dispatch (ver la sección 13). CK0, que lo usan los comandos sin ningún argumento, puede ir seguido de cualesquiera objetos adicionales. CK1&Dispatch .. CK&Dispatch deben ir seguidos de una secuencia de pares de objetos; el primero de cada par identifica una combinación de tipos de argumentos de la pila y el segundo especifica el objeto a ejecutar para cada combinación correspondiente. El último par está seguido del objeto marcador de fin de programa (SEMI).

Las otras convenciones para los objetos comando gobiernan su comportamiento. En concreto, deberían:

- eliminar cualquier objeto temporal de la pila, retornando sólo los resultados especificados;

- hacer cualquier comprobación de márgenes necesaria para asegurar que no ocurran errores que puedan causar desastres;

- restaurar los modos de la HP48 a sus estados originales, a menos que el comando sea específicamente para cambiar un modo.

El gasto que implican estas convenciones estructurales y de comportamiento impone una pequeña penalización en la ejecución. Sin embargo, la principal ventaja del RPL del sistema sobre el RPL del usuario, la velocidad de ejecución, viene simplemente del conjunto más grande de procedimientos disponibles en el RPL del sistema, del acceso a la rápida aritmética binaria y del control mejorado sobre los recursos del sistema y del flujo de ejecución.

2.7 Programación en RPL del Sistema

Escribir programas en RPL del sistema no es diferente, en principio, que en RPL de usuario; la diferencia está en la sintaxis y en la amplitud del compilador. Para el RPL de usuario, el compilador es el ENTER de la línea de comandos, cuya lógica está documentada en los manuales del propietario. Para el RPL del sistema desarrollado en un PC, el compilador tiene varias partes. El análogo inmediato del analizador de la línea de comandos es el programa RPLCOMP, que analiza y traduce el texto del código fuente al lenguaje ensamblador del Saturno. (La sintaxis usada por RPLCOMP se describe en RPLCOMP.DOC). La salida de RPLCOMP se pasa al programa ensamblador SASM, que genera el código objeto ensamblado. El programa SLOAD resuelve las referencias de símbolos en la salida de SASM, devolviendo finalmente el código ejecutable adecuado para cargarlo en la HP 48. Se pueden recoger los objetos individuales en un directorio de la HP 48 que se vuelve a transferir de nuevo al PC, donde el programa USRLIB puede transformar el directorio en un objeto biblioteca. (Sería deseable la creación de una biblioteca directamente en el PC, pero el programa para hacer esto no está disponible aun).

A modo de ilustración, considera un hipotético proceso del desarrollo de un proyecto que dará lugar a un objeto biblioteca construido con la herramienta USRLIB. La biblioteca contendrá un solo comando, BASKET, que calcula los factores de tejido de una cesta según varios parámetros de entrada. BASKET se debe diseñar con la estructura descrita anteriormente para los comandos. Además, supón que BASKET llama a otros programas que no serán accesibles al usuario. Para conseguir esto, se compilan los objetos en el PC, luego se cargan en la HP 48 en un directorio común, almacenados como BASKET, B1, B2, ... , donde las últimas variables contienen las subrutinas. El directorio se carga al PC, donde se le aplica USRLIB con la directiva que dice que B1, B2, ... han de ser "ocultos".

No hay ninguna obligación de que el programa producido con el compilador RPL se deba presentar en un objeto biblioteca - si toda la aplicación se puede escribir dentro de un solo programa, mucho mejor. A medida que los programas crecen más allá de un cierto nivel razonable de complejidad, esto se hace más difícil y el enfoque de un objeto biblioteca con múltiples variables es más fácil de manejar.

1. Crea el fichero fuente en el PC usando tu editor de texto favorito. El nombre del fichero del programa fuente deberá tener la extensión ".s", tal como "prog.s". Usa el compilador RPLCOMP.EXE para producir el fichero ensamblador del Saturno "prog.a".

2. Usa el ensamblador del Saturno SASM.EXE para ensamblar el programa y producir un fichero de salida "prog.o"

3. Usa el cargador del Saturno SLOAD.EXE para resolver las llamadas de tu programa al sistema operativo de la HP 48. Los ficheros de salida de SLOAD.EXE pueden tener cualquier nombre, pero se usa a menudo la extensión ".ol".

4. Carga el fichero final (-usa transferencia binaria!) en la HP 48 y prueba el funcionamiento de tu código.

5. Carga el directorio que contiene uno o más objetos al PC y usa USRLIB.EXE para convertirlo en una biblioteca.

2.8 Programa de Ejemplo en RPL

Para familiarizarse con el proceso de producir un programa escrito en RPL interno, considera el siguiente ejemplo, al que llamaremos TOSET.

2.8.1 El Fichero Fuente

Este programa elimina los objetos duplicados de una lista descomponiendo la lista en una serie de objetos en la pila, creando una nueva lista vacía y colocando los objetos de la pila en la nueva lista si son únicos.

```
* ( {lista} --> {lista}' )
ASSEMBLE
        NIBASC  /HPHP48-D/
RPL
::
  CK1NOLASTWD                   ( *Req. 1 argumento* )
  CK&DISPATCH0 list
    ::
      DUPNULL{}? ?SEMI          ( *Sale si la lista está vacía* )
      INNERCOMP                 ( objn ... obj1 #n )
      reversym                  ( obj1 ... objn #n )
      NULL{} SWAP               ( obj1 ... objn {} #n )
      ZERO_DO (DO)
        SWAP                    ( obj1 ... objn-1 {} objn )
        apndvarlst              ( obj1 ... objn-1 {}' )
      LOOP
   ;
;
```

La primera línea es un comentario que muestra las condiciones de entrada y salida del programa. Los comentarios se denotan por un asterisco (*) en la primera columna o entre paréntesis. Cada programador tiene su propio estilo para los comentarios. El estilo mostrado aquí es que los objetos se muestran con el nivel uno de la pila a la derecha. El texto se incluye entre asteriscos.

La secuencia

```
ASSEMBLE
        NIBASC  /HPHP48-D/
RPL
```

es un comando para el ensamblador que incluye la cabecera para la transferencia binaria de datos desde el PC a la HP48. Se incluye aquí por simplicidad, pero podría ser incluida desde otro fichero por el cargador.

El primer comando, CK1NOLASTWD, exige que la pila contenga al menos un objeto y borra la posición de la ram que almacena el nombre del comando actual. Esto es importante porque no querrás atribuir los errores que se puedan producir en este programa a la última función que generó un error.

El segundo comando, CK&DISPATCH0, lee una estructura de la forma

```
tipo acción
tipo acción
...
```

para decidir que acción tomar basándose en el TYPE (tipo) del objeto presentado. Si el tipo de objeto en el nivel 1 no tiene una entrada en la tabla, se generará el error "Bad Argument Type" ("Tipo de Argumento Incorrecto"). En este ejemplo, sólo se acepta un tipo de argumento, una lista, y la acción correspondiente es un secundario. Para saber más de los comandos de chequeo de argumentos, ver el capítulo "Chequeo de Argumentos"

El comando DUPNULL{}? devuelve la lista y una bandera TRUE/FALSE (Cierto/Falso) que indica si la lista esta o no vacía. El comando ?SEMI sale del secundario si la bandera es TRUE (Cierto).

El comando INNERCOMP es una forma interna de la palabra de usuario LIST->. El número de objetos se devuelve al nivel uno como un entero binario (ver el capítulo "Enteros Binarios").

El comando "reversym" invierte el orden de #n objetos en la pila. Se usa aquí teniendo en cuenta el orden en que son colocados los objetos en una lista por "apndvarlst" que se describe luego.

El comando ZERO_DO inicia un bucle contador. Este bucle procesará cada objeto de la lista original. El comando (DO) le dice a RPLCOMP que este es el principio del bucle, si no el comando LOOP se señalaría como "sin emparejar".

El comando "apndvarlst" añade un objeto a una lista si y sólo si ese objeto no aparece ya en la lista.

El comando LOOP termina el bucle. Para saber más de los comandos de bucle, ver el capítulo "Estructuras de Control de Bucles"

2.8.2 Compilar el Programa Para compilar el programa para la HP 48, seguir estos pasos:

1. Almacena el código ejemplo en el fichero TOSET.S

2. RPLCOMP TOSET.S TOSET.A

Este comando compila el fuente RPL y produce un fichero fuente en ensamblador del Saturno.

3. SASM TOSET.A

Este comando ensambla el fichero fuente del Saturno para producir los ficheros TOSET.L y TOSET.O

4. El fichero TOSET.M es un fichero de control del cargador que se parece a esto:

```
TITLE Example           <-- Especifica el título del listado
OUTPUT TOSET            <-- Especifica el fichero de salida
LLIST TOSET.LR          <-- Especifica el fichero de listados
SUPPRESS XREF           <-- Suprime las referencias cruzadas
SEARCH ENTRIES.O        <-- Lee las entradas de la HP 48
REL TOSET.O             <-- Carga TOSET.o
END
```

Crea el fichero TOSET.M e invoca el cargador:

SLOAD -H TOSET.M

Comprueba el fichero TOSET.LR para ver si hay errores. Una referencia no resuelta (unresolved reference) apunta normalmente a un comando mal escrito. Carga ahora el fichero TOSET en la HP 48 y dale una oportunidad!

Entra la lista { 1 2 2 3 3 3 4 }, evalúa TOSET y deberías obtener { 1 2 3 4 }.

3. Estructuras de los Objetos

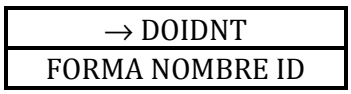
Este capítulo proporciona información adicional acerca de algunos tipos de objetos RPL soportados por la HP 48. Aunque la información es relevante principalmente para la programación en código máquina, un conocimiento de la estructura de los objetos puede, a menudo, ayudar a entender las consecuencias en la ejecución y en la eficacia de la programación en RPL.

A menos que se diga explícitamente lo contrario, todos los campos definidos específicamente dentro del cuerpo de un objeto se supone que son de 5 nibbles, el ancho de las direcciones de la CPU.

3.1 Tipos de Objetos

3.1.1 Objeto Identificador

Un objeto identificador es atómico, tiene el prólogo DOIDNT y un cuerpo que es una "forma Nombre ID".



Dirección del Prólogo
Cuerpo

Objeto Identificador

Una "forma Nombre ID" es una secuencia de caracteres precedida por un campo contador de caracteres de un byte.

Los objetos identificadores son, entre otras cosas, la resolución en tiempo de compilación de las variables globales.

3.1.2 Objeto Identificador Temporal

Un objeto identificador temporal es atómico, tiene el prólogo DOLAM y un cuerpo que es una forma Nombre ID

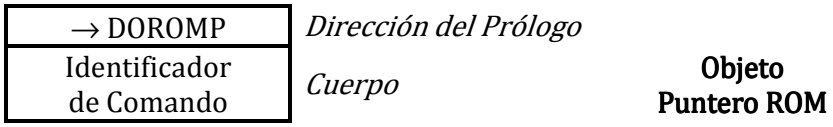
Dirección del Prólogo
Cuerpo

Objeto Identificador Temporal

Los objetos identificadores temporales proporcionan referencias con nombres para los objetos temporales unidos a los identificadores en la lista formal de parámetros de una estructura de variables temporales.

3.1.3 Objeto Puntero ROM

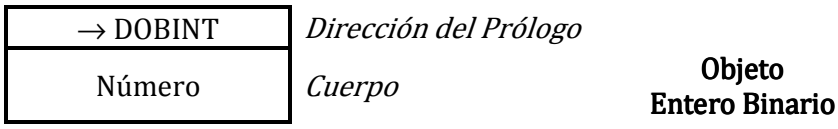
Un objeto puntero ROM, o nombre XLIB, es atómico, tiene el prólogo DOROMP y un cuerpo que es un identificador ROM-WORD.



Los objetos punteros ROM son la resolución en tiempo de compilación de los comandos en bibliotecas móviles. Un identificador identificador de comando es un par de campos de 12 bits: el primer campo es un número de ID de biblioteca y el segundo campo es el número ID del comando dentro de la biblioteca.

3.1.4 Objeto Entero Binario

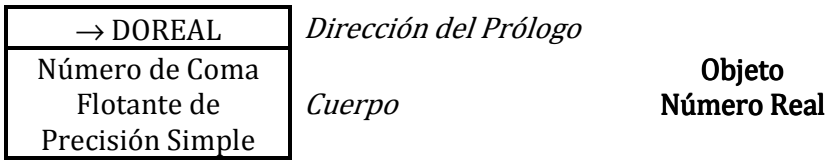
Un objeto entero binario es atómico, tiene el prólogo DOBINT y un cuerpo que es un número de 5 nibbles.



El uso de este tipo de objetos es para representar enteros binarios cuya precisión es equivalente a una dirección de memoria.

3.1.5 Objeto Número Real

Un objeto número real es atómico, tiene el prólogo DOREAL y un cuerpo que es un número en coma flotante de precisión simple (o, abreviando, un número real).



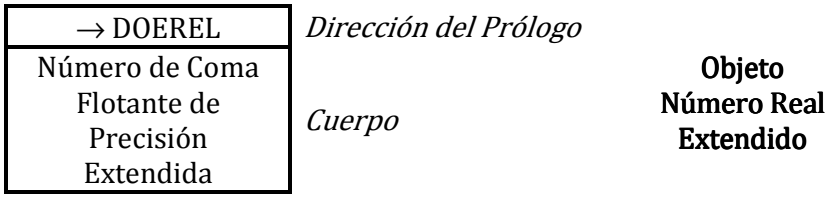
Uno de los usos de este tipo de objetos es representar números de coma flotante empaquetados (ocho bytes) en un sistema Saturno y, en esta aplicación, el cuerpo del objeto consiste en 16 nibbles BCD como sigue:

```
(mem baja)      EEEMMMMMMMMMMMMMS
```

donde S es el signo numérico (0 para no negativo y 9 para negativo), MMMMMMMMMMM son los 12 dígitos de la mantisa con una coma implícita entre el primero y segundo dígitos y el primer dígito distinto de cero si el número no es cero y EEE es el exponente en forma de complemento a diez (-500 < EEE < 500).

3.1.6 Objeto Número Real Extendido

Un objeto número real extendido es atómico, tiene el prólogo DOEREL y un cuerpo que es un número de coma flotante de precisión extendida (o, abreviando, real extendido).



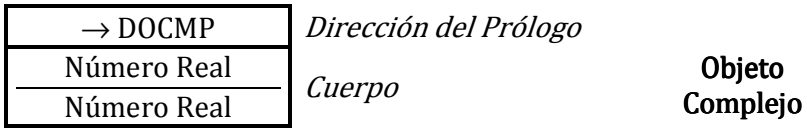
Uno de los usos de este tipo de objetos es representar números de coma flotante sin empaquetar (10.5 bytes) en un sistema Saturno y, en este aplicación, el cuerpo del objeto puede consistir en 21 nibbles BCD como sigue:

```
(mem baja)      EEEEEMMMMMMMMMMMMMMS
```

donde S es el signo numérico (0 para no negativo, 9 para negativo), MMMMMMMMMMMMMMM es una mantisa de 15 dígitos con una coma decimal implícita entre el primer y el segundo dígitos y el primero distinto de cero si el número no es cero y EEEEE es el exponente en forma de complemento a diez (-50000 < EEEEE < 50000).

3.1.7 Objeto Número Complejo

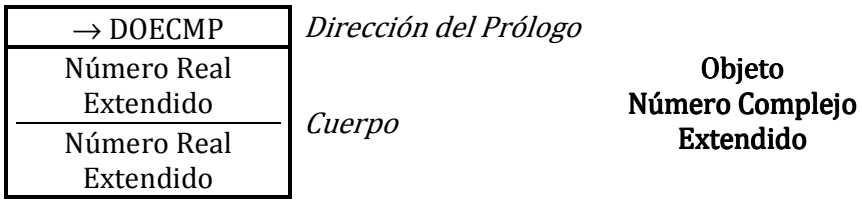
Un objeto número complejo es atómico, tiene el prólogo DOCMP y un cuerpo que es un par de números reales.



Este tipo de objetos se usan para representar números complejos de precisión simple, donde la parte real se interpreta como el primer número real del par.

3.1.8 Objeto Número Complejo Extendido

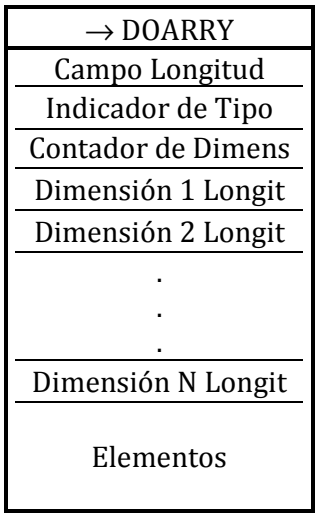
Un objeto número complejo extendido es atómico, tiene el prólogo DOECMP y un cuerpo que es un par de números reales extendidos.



Este tipo de objetos se usa para representar números complejos de precisión extendida de la misma manera que un objeto complejo.

3.1.9 Objeto Formación

Un objeto Formación es atómico, tiene el prólogo DOARRY y un cuerpo que es una colección de los elementos de la formación. El cuerpo también incluye un campo longitud (indicando la longitud del cuerpo), un indicador de tipo (indicando el tipo de objetos de sus elementos), un campo contador de dimensiones y campos de longitud para cada dimensión.

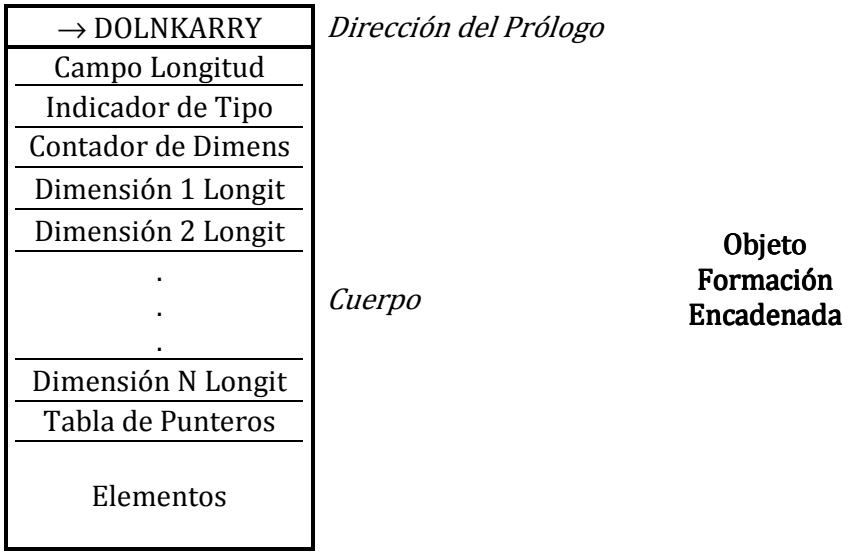


Los elementos de la formación son cuerpos de objetos del mismo tipo de objeto. El indicador de tipo es una dirección de prólogo (piensa en esta dirección de prólogo como si se aplicara a cada elemento de la formación).

El "OPTION BASE" de la formación es siempre 1. Una formación nula se designa teniendo algún límite de dimensión el valor cero. Todos los elementos de un objeto formación están siempre presentes como se indica por la información de la dimensionalidad y se ordenan en memoria por elorden lexicográfico de los índices de la formación.

3.1.10 Objeto Formación Encadenada

Un objeto formación encadenada es atómico, tiene el prólogo DOLNKARRY y un cuerpo que es una colección de los elementos de la formación. El cuerpo también incluye un campo longitud (que indica la longitud del cuerpo), un indicador de tipo (que indica el tipo de objeto de los elementos), un campo contador de dimensiones, campos longitud para cada dimensión y una tabla de punteros cuyo contenido son offsets auto-relativos hacia adelante a los elementos de la formación; los elementos de la tabla de punteros están ordenados en memoria por el orden lexicográfico de los índices de la formación.

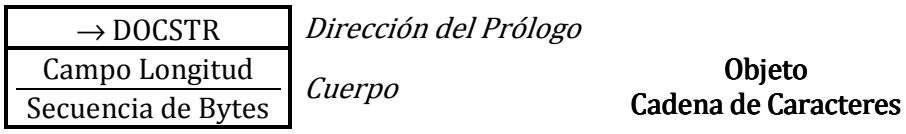


Los elementos de la formación son cuerpos de objetos del mismo tipo de objeto. El indicador de tipo es una dirección de prólogo (piensa en esta dirección de prólogo como si se aplicara a cada elemento de la formación).

El "OPTION BASE" de la formación encadenada es siempre 1. Una formación encadenada nula es aquella que tiene alguno de sus límites de dimensión igual a cero. No se supone ningún orden de los elementos de un objeto formación encadenada y ni siquiera se supone su presencia; la ausencia de un elemento en una dimensión asignada se indica por el valor cero ocupando el elemento correspondiente de la tabla de punteros.

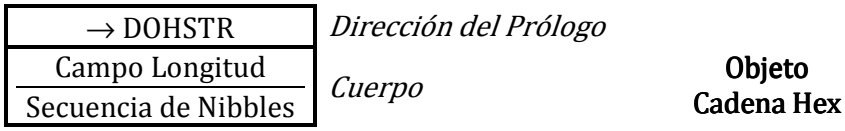
3.1.11 Objeto Cadena de Caracteres

Un objeto cadena de caracteres es atómico, tiene el prólogo DOCSTR y un cuerpo que es una cadena de caracteres (una secuencia de bytes). El cuerpo también incluye un campo longitud (que indica la longitud del cuerpo).



3.1.12 Objeto Cadena Hexadecimal (Hex)

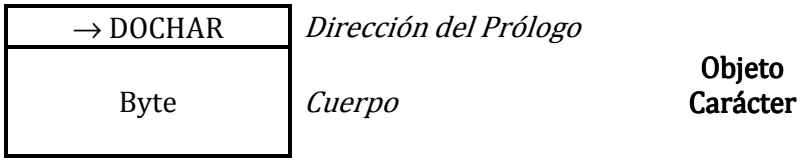
Un objeto cadena Hex es atómico, tiene el prólogo DOHSTR y un cuerpo que es una secuencia de nibbles. El cuerpo también incluye un campo longitud (que indica la longitud del cuerpo).



Un uso típico de este tipo de objetos es un búfer o tabla. Los objetos cadena hex de 16 o menos nibbles se usan para representar objetos enteros binarios en RPL de usuario.

3.1.13 Objeto Carácter

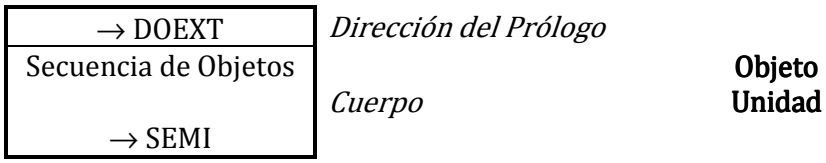
Un objeto carácter es atómico, tiene el prólogo DOCHAR y un cuerpo que es un sólo byte.



Este objeto se usa para representar cantidades de un solo byte, tales como caracteres ASCII o ROMAN8.

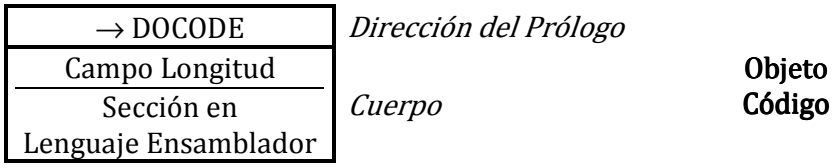
3.1.14 Objeto Unidad

Un objeto unidad es compuesto, tiene el prólogo DOEXT y un cuerpo que es una secuencia que consiste en un número real seguido de cadenas de nombres de unidades, caracteres prefijo, operadores de unidades y potencias de números reales, delimitado por la cola por un puntero a SEMI.



3.1.15 Objeto Código

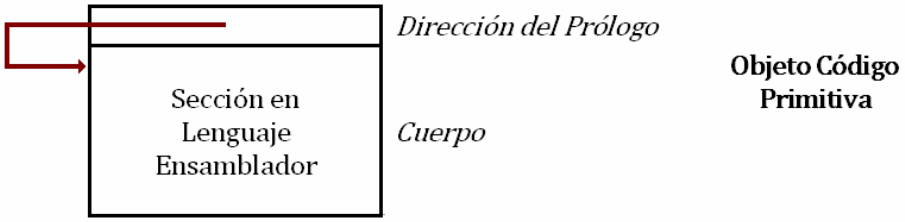
Un objeto código es atómico, tiene el prólogo DOCODE y un cuerpo que es una sección en lenguaje ensamblador. El cuerpo también incluye un campo longitud (que indica la longitud del cuerpo). Cuando se ejecuta, el prólogo coloca el contador de programa del sistema en la sección en lenguaje ensamblador dentro del cuerpo.



Las principales aplicaciones de este tipo de objeto son los procedimientos en lenguaje ensamblador que se pueden embeber directamente en los objetos compuestos o existir en RAM.

3.1.16 Objeto Código Primitiva

Un objeto código primitiva es un caso especial de objeto código, usado para representar el código de las primitivas en las bibliotecas incorporadas. El prólogo de un objeto código primitiva es su cuerpo, que es una sección en lenguaje ensamblador; así, cuando se ejecuta, el cuerpo se ejecuta a sí mismo.

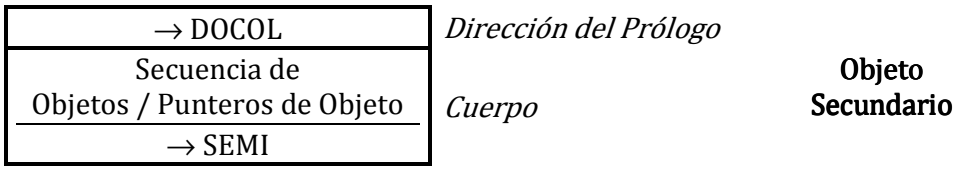


El propósito principal de este tipo de objetos es la ejecución más rápida de los objetos código en las bibliotecas incorporadas, esto es, estos objetos se ejecutan sin el nivel extra inherente a la ejecución de un prólogo separado. Sin embargo, su estructura implica que (1) sólo pueden existir en las bibliotecas incorporadas (nunca en RAM ni en bibliotecas móviles) ya que el cuerpo debe estar en una dirección fija, (2) no se pueden saltar y (3) no pueden existir en cualquier situación en la que pueda ser necesario esquivar un objeto, tales como un elemento de una formación o un objeto dentro de cualquier objeto compuesto.

Observa que este tipo de objetos es una excepción al esquema de clasificación de tipos de objetos presentada al principio de este documento. Sin embargo, un objeto es un objeto código primitiva si y sólo si su dirección del prólogo es igual a la dirección del objeto más 5. Además, los prólogos de este tipo de objetos (o sea, los cuerpos de los objetos) no necesitan contener ninguna lógica para comprobar si la ejecución es directa o indirecta ya que, por definición, no se pueden ejecutar directamente.

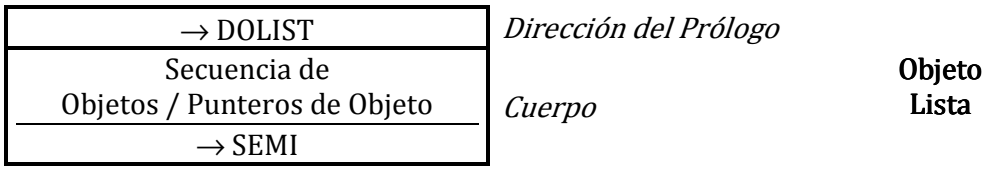
3.1.17 Objeto Programa

Un objeto programa (secundario) es compuesto, tiene el prólogo DOCOL y un cuerpo que es una secuencia de objetos y punteros de objeto, el último de los cuales es un puntero de objeto que apunta al objeto código primitiva SEMI.



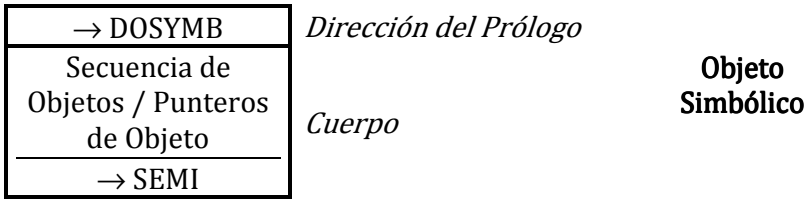
3.1.18 Objeto Lista

Un objeto lista es compuesto, tiene el prólogo DOLIST y un cuerpo que es una secuencia de objetos y punteros de objeto, el último de los cuales es un puntero de objeto que apunta al objeto código primitiva SEMI.



3.1.19 Objeto Simbólico

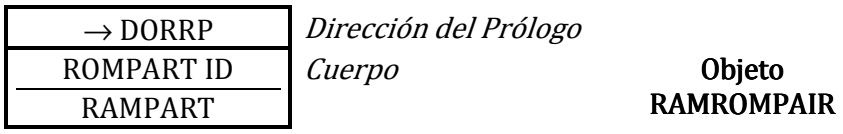
Un objeto simbólico es compuesto, tiene el prólogo DOSYMB y un cuerpo que es una secuencia de objetos y punteros de objeto, el último de los cuales es un puntero de objeto que apunta al objeto código primitiva SEMI.



Este tipo de objetos se usa para representar objetos simbólicos para aplicaciones de matemática simbólica.

3.1.20 Objeto Directorio

Un objeto directorio (RAMROMPAIR) es atómico, tiene el prólogo DORRP y un cuerpo que consiste en un número ID de biblioteca y una RAMPART (PARTE RAM) (lista encadenada de variables--pares objeto/nombre)

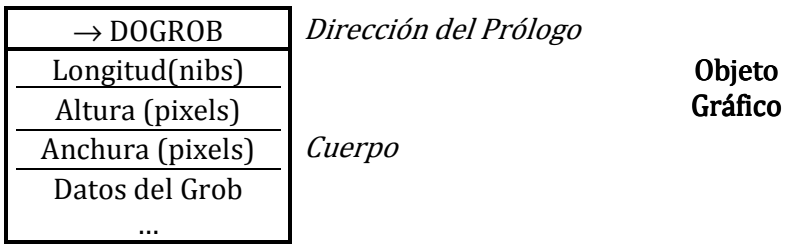


3.1.21 Objeto Gráfico

Un objeto gráfico es atómico, tiene el prólogo DOGROB y un cuerpo que consiste en lo siguiente:

- Un campo de 5 nibbles de longitud para los datos que siguen.
- Una cantidad de cinco nibbles que describe la altura del gráfico en pixels.
- Una cantidad de cinco nibbles que describe la anchura del gráfico en pixels.
- Los datos.

La dimensión real de las filas en nibbles (W) es siempre un número par por razones del hardware, así que cada fila de datos de pixels está completada con 0 a 7 bits de datos desperdiciados.



Los nibbles de datos comienzan en la esquina superior izquierda del objeto gráfico y siguen de izquierda a derecha y de arriba hacia abajo. Cada fila de datos de pixels está completada según se necesite hasta obtener un número par de nibbles por fila. Así, la anchura en nibbles W se determina por:

```
W=CEIL(Anchura en pixels)/8
```

Los bits de cada nibble se escriben en orden inverso, de modo que el pixel mostrado más a la izquierda se representa en un nibble por el bit menos significativo del nibble.

3.2 Terminología y Abreviaciones

En los diagramas de la pila que se usarán en el lo que resta de este documento, se usan los siguientes símbolos para representar los diferentes tipos de objetos: (N.T.> no los traduzco en los diagramas)

```
ob .......... Cualquier Objeto
id .......... Objeto Identificador
lam ......... Objeto Identificador Temporal
romptr ...... Objeto Puntero ROM
# ............Objeto Entero Binario
% ........... Objeto Real
%% .......... Objeto Real Extendido
C% .......... Objeto Complejo
C%% ......... Objeto Complejo Extendido
arry ........ Objeto Formación
lnkarry ..... Objeto Formación Encadenada
$ ........... Objeto Cadena de Caracteres
hxs ......... Objeto Cadena Hex
chr ......... Objeto Carácter
ext ......... Objeto Externo
code ........ Objeto Código
primcode .... Objeto Código Primitiva
:: .......... Objeto Secundario
{} .......... Objeto Lista
symb ........ Objeto Simbólico
comp ........ Cualquier Objeto Compuesto (lista,secundario,simbólico)
rrp ......... Objeto Directorio
tagged ...... Objeto Etiquetado
flag ........ TRUE/FALSE (Cierto/Falso)
```

(TRUE y FALSE denotan las partes de objeto de las ROM-WORDs (palabras ROM) incorporadas que tienen estos nombres. Las direcciones de estos objetos (o sea, sus representaciones en la pila de datos) se interpretan por las estructuras de control RPL como el valor apropiado de veracidad. Ambos objetos son objetos código primitiva que, cuando se ejecutan, se colocan ellos mismos en la pila de datos).

Además de la notación arriba indicada, es útil conocer alguna terminología adicional.

ELEMENTO:
Un ELEMENTO de un objeto compuesto es cualquier objeto o puntero de objeto en el cuerpo de un objeto compuesto.

NUCLEO:

de una cadena de caracteres: el núcleo de un objeto cadena de caracteres son los caracteres del cuerpo.

de una cadena hex: el núcleo de un objeto cadena hex es la secuencia de nibbles del cuerpo.

de un compuesto: el núcleo de un objeto compuesto es la secuencia de elementos del cuerpo sin incluir el puntero de objeto final que apunta a SEMI.

LONGITUD:

de una cadena de caracteres: la longitud de un objeto cadena de caracteres es el número de caracteres del núcleo.

de una cadena hex: la longitud de un objeto cadena hex es el número de nibbles del núcleo.

de un compuesto: la longitud de un objeto compuesto es el número de elementos del núcleo.

NULO:

cadena de caracteres: un objeto cadena de caracteres nula es una cuya longitud es cero.

cadena hex: un objeto cadena hex nulo es uno cuya longitud es cero.

compuesto: un objeto compuesto nulo es uno cuya longitud es cero.

INTERNO:

un interno de un objeto compuesto es cualquier objeto en el núcleo del objeto compuesto o lo apuntado por cualquier puntero de objeto en el núcleo del objeto compuesto.

(A menudo nos referimos a un objeto compuesto diciendo que contiene un tipo específico de objeto, por ejemplo, "una lista de enteros binarios"; lo que realmente se quiere decir es que los internos del núcleo son todos de este tipo de objeto).

4. Enteros Binarios

Los enteros binarios internos tienen un tamaño fijo de 20 bits y son el tipo usado más a menudo por los contadores, bucles, etc. Los enteros binarios ofrecen ventajas de tamaño y velocidad.

NOTA: Los enteros binarios a nivel de usuario se implementan como cadenas hex, así que el objeto de usuario #247d es en realidad una cadena hex y no se debería confundir con un entero binario cuyo prólogo es DOBINT.

4.1 Enteros Binarios Incorporados

El compilador RPLCOMP interpreta un número decimal en un fichero fuente como una directiva que genera un objeto entero binario - usando un prólogo y un cuerpo. Los enteros binarios incorporados pueden accederse con sólo un puntero de objeto. Por ejemplo, " 43 " (sin las comillas) en el fichero fuente genera un objeto binario:

```
CON(5)   =DOBINT
CON(5)   43
```

El objeto ocupa cinco bytes, pero se puede reemplazar por la palabra "FORTYTHREE" (Cuarenta y tres), que es un punto de entrada soportado que generaría el siguiente código:

```
CON(5)   =FORTYTHREE
```

Un escollo a tener en cuenta en la convención de cómo se nombran los enteros binarios es la diferencia que hay entre las entradas FORTYFIVE (Cuarenta y cinco) y FOURFIVE (Cuatro Cinco). En el primer caso, el valor decimal es 45 pero el último es el decimal 69 (hex 45). Los nombres como 2EXT e IDREAL, donde los valores no son obvios, se usan conjuntamente con la familia de comandos CK&Dispatch de chequeo de argumentos. Los nombres para la familia de los CK&Dispatch se equiparan a los mismos lugares que otros enteros binarios. Esto se ha hecho por legibilidad. Por ejemplo, la palabra SEVENTEEN (diecisiete), para el decimal 17, tiene los nombres 2REAL y REALREAL equiparados a la misma posición. Una "d" o "h" al final de nombres tales como BINT_122d o BINT80h (N.T.> BINT viene de B_inary INT_eger = Entero binario) indican la base asociada con el valor.

Las palabras como ONEONE (UnoUno), ZEROONE (CeroUno), etc. ponen más de un entero binario en la pila. Esto se indica con un pequeño diagrama de pila entre paréntesis, tal como (--> #1 #1) para ONEONE.

Las entradas soportadas para los enteros binarios se listan debajo con el valor hex entre paréntesis cuando sea necesario:

N.T.> A la izquierda, el valor decimal. Separados con | si son dos o más números diferentes dichos de seguido (CuatroCinco) --> 4|5

```
  238  2EXT (#EE)        49  FORTYNINE        161  SYMREAL (#A1)
  204  2GROB (#CC)       41  FORTYONE         170  SYMSYM (#AA)
   85  2LIST (#55)       47  FORTYSEVEN       208  TAGGEDANY (#D0)
   17  2REAL (#11)       46  FORTYSIX          10  TEN
  273  3REAL (#111)      43  FORTYTHREE        13  THIRTEEN
 2563  Attn# (#A03)      42  FORTYTWO          30  THIRTY
  253  BINT253            4  FOUR              38  THIRTYEIGHT
  255  BINT255d          69  FOURFIVE          35  THIRTYFIVE
   64  BINT40h           14  FOURTEEN          34  THIRTYFOUR
  128  BINT80h           67  FOURTHREE         39  THIRTYNINE
  192  BINTC0h           66  FOURTWO           31  THIRTYONE
  115  BINT_115d         40  FOURTY            37  THIRTYSEVEN
  116  BINT_116d         97  IDREAL (#61)      36  THIRTYSIX
  122  BINT_122d        337  INTEGER337        33  THIRTYTHREE
  130  BINT_130d         82  LISTCMP (#52)     32  THIRTYTWO
  131  BINT_131d         87  LISTLAM (#57)      3  THREE
   65  BINT_65d          81  LISTREAL (#51)    12  TWELVE
   91  BINT_91d    1048575  MINUSONE(#FFFFF)  20  TWENTY
   96  BINT_96d           9  NINE              28  TWENTYEIGHT
 3082  Connecting(#C0A)  19  NINETEEN          25  TWENTYFIVE
    8  EIGHT              1  ONE               24  TWENTYFOUR
   18  EIGHTEEN         100  ONEHUNDRED        29  TWENTYNINE
   80  EIGHTY           1|1  ONEONE(--> #1 #1) 21  TWENTYONE
   81  EIGHTYONE         30  REALEXT (#1E)     27  TWENTYSEVEN
   11  ELEVEN            16  REALOB (#10)      26  TWENTYSIX
   14  EXT (#E)         256  REALOBOB (#100)   23  TWENTYTHREE
 3584  EXTOBOB (#E00)    17  REALREAL (#11)    22  TWENTYTWO
  225  EXTREAL (#E1)     26  REALSYM (#1A)      2  TWO
  234  EXTSYM  (#EA)    240  ROMPANY (#F0)    131  XHI
   15  FIFTEEN            7  SEVEN            130  XHI-1 (#82)
   50  FIFTY             17  SEVENTEEN          0  ZERO
   58  FIFTYEIGHT        70  SEVENTY          0|0  ZEROZERO (--> #0 #0 )
   55  FIFTYFIVE         74  SEVENTYFOUR    0|0|1  ZEROZEROONE (--> #0 #0 #1 )
   54  FIFTYFOUR         79  SEVENTYNINE    0|0|2  ZEROZEROTWO (--> #0 #0 #2 )
   59  FIFTYNINE          6  SIX            0|0|0  ZEROZEROZERO (--> #0 #0 #0)
   51  FIFTYONE          16  SIXTEEN          103  char (#6F)
   57  FIFTYSEVEN        60  SIXTY              6  id (#6)
   56  FIFTYSIX          68  SIXTYEIGHT         6  idnt (#6)
   53  FIFTYTHREE        64  SIXTYFOUR        773  infreserr (#305)
   52  FIFTYTWO          61  SIXTYONE        2563  intrptderr (#a03)
    5  FIVE              63  SIXTYTHREE         5  list (#5)
   84  FIVEFOUR          62  SIXTYTWO         771  ofloerr (#303)
   86  FIVESIX          158  SYMBUNIT (#9E)     1  real (#1)
   83  FIVETHREE        174  SYMEXT (#AE)       8  seco (#8)
   40  FORTY            166  SYMID (#A6)        3  str (#3)
   48  FORTYEIGHT       167  SYMLAM (#A7)      10  sym (#A)
   45  FORTYFIVE        160  SYMOB (#A0)        9  symb (#9)
   44  FORTYFOUR
```

4.2 Manipulación de Enteros Binarios

4.2.1 Funciones Aritméticas

```
#*              ( #2 #1 --> #2*#1 )
#+              ( #2 #1 --> #2+#1 )
#+-1            ( #2 #1 --> #2+#1-1 )
#-              ( #2 #1 --> #2-#1 )
#-#2/           ( #2 #1 --> (#2-#1)/2 )
#-+1            ( #2 #1 --> (#2-#1)+1 )
#/              ( #2 #1 --> #resto #cociente )
#1+             ( # --> #+1 )
#1+'            ( # --> #+1 y ejecuta ' )
#1+DUP          ( # --> #+1 #+1 )
#1-             ( # --> #-1 )
#10*            ( # --> #*10 )
#10+            ( # --> #+10 )
#12+            ( # --> #+12 )
#2*             ( # --> #*2 )
#2+             ( # --> #+2 )
#2-             ( # --> #-2 )
#2/             ( # --> FLOOR(#/2) )
#3+             ( # --> #+3 )
#3-             ( # --> #-3 )
#4+             ( # --> #+4 )
#4-             ( # --> #-4 )
#5+             ( # --> #+5 )
#5-             ( # --> #-5 )
#6*             ( # --> #*6 )
#6+             ( # --> #+6 )
#7+             ( # --> #+7 )
#8*             ( # --> #*8 )
#8+             ( # --> #+8 )
#9+             ( # --> #+9 )
#MAX            ( #2 #1 --> MAX(#2,#1) )
#MIN            ( #2 #1 --> MIN(#2,#1) )
2DUP#+          ( #2 #1 --> #2 #1 #1+#2 )
DROP#1-         ( # ob --> #-1 )
DUP#1+          ( # --> # #+1 )
DUP#1-          ( # --> # #-1 )
DUP3PICK#+      ( #2 #1 --> #2 #1 #1+#2 )
OVER#+          ( #2 #1 --> #2 #1+#2 )
OVER#-          ( #2 #1 --> #2 #1-#2 )
ROT#+           ( #2 ob #1 --> ob #1+#2 )
ROT#+SWAP       ( #2 ob #1 --> #1+#2 ob )
ROT#-           ( #2 ob #1 --> ob #1-#2 )
ROT#1+          ( # ob ob' --> ob ob' #+1 )
ROT+SWAP        ( #2 ob #1 --> #1+#2 ob )
SWAP#-          ( #2 #1 --> #1-#2 )
SWAP#1+         ( # ob --> ob #+1 )
SWAP#1+SWAP     ( # ob --> #+1 ob )
SWAP#1-         ( # ob --> ob #-1 )
SWAP#1-SWAP     ( # ob --> #-1 ob )
SWAPOVER#-      ( #2 #1 --> #1 #2-#1 )
```

4.2.2 Funciones de Conversión

```
COERCE          ( % --> # )  Si %<0 entonces # es 0
                             Si %>FFFFF entonces #=FFFFF
COERCE2         ( %2 %1 --> #2 #1 ) Ver COERCE
COERCEDUP       ( % --> # # ) Ver COERCE
COERCESWAP      (ob % --> # ob )
UNCOERCE        ( # --> % )
UNCOERCE%%      ( # --> %% )
UNCOERCE2       ( #2 #1 --> %2 %1 )
```

5. Constantes Carácter

Las siguientes palabras son útiles para la interconversión entre objetos carácter y otros tipos de objetos:

```
CHR>#           ( chr --> # )
#>CHR           ( # --> chr )
CHR>$           ( chr --> $ )
```

Las siguientes constantes carácter y cadena están soportadas:

```
CHR_# CHR_* CHR_+ CHR_, CHR_- CHR_. CHR_/ CHR_0 CHR_1 CHR_2
CHR_3 CHR_4 CHR_5 CHR_6 CHR_7 CHR_8 CHR_9 CHR_: CHR_; CHR_<
CHR_= CHR_> CHR_A CHR_B CHR_C CHR_D CHR_E CHR_F CHR_G CHR_H
CHR_I CHR_J CHR_K CHR_L CHR_M CHR_N CHR_O CHR_P CHR_Q CHR_R
CHR_S CHR_T CHR_U CHR_V CHR_W CHR_X CHR_Y CHR_Z CHR_a CHR_b
CHR_c CHR_d CHR_e CHR_f CHR_g CHR_h CHR_i CHR_j CHR_k CHR_l
CHR_m CHR_n CHR_o CHR_p CHR_q CHR_r CHR_s CHR_t CHR_u CHR_v
CHR_w CHR_x CHR_y CHR_z

CHR_00 (hex 0) CHR_...  CHR_DblQuote    CHR_-> CHR_<<
CHR_>> CHR_Angle CHR_Deriv CHR_Integral CHR_LeftPar
CHR_Newline  CHR_Pi CHR_RightPar CHR_Sigma CHR_Space
CHR_UndScore CHR_[  CHR_]  CHR_{ CHR_}  CHR_<= CHR_>=
CHR_<>

$_R<<           ( $ "R\80\80" "R<ángulo><ángulo>" )
$_R<Z           ( $ "R\80Z"   "R<ángulo>Z"        )
$_XYZ           ( $ "XYZ"                         )
$_<<>>          ( $ "ABBB"                        )
$_{}            ( $ "{}"                          )
$_[]            ( $ "[]"                          )
$_''            ( $ "''"                          )
$_::            ( $ "::"                          )
$_LRParens      ( $ "()"                          )
$_2DQ           ( $ """""                         )
$_ECHO          ( $ "ECHO"                        )
$_EXIT          ( $ "EXIT"                        )
$_Undefined     ( $ "Undefined"                   )
$_RAD           ( $ "RAD"                         )
$_GRAD          ( $ "GRAD"                        )
NEWLINE$        ( $ "\0a"                         )
SPACE$          ( $ " "                           )
```

6. Cadenas Hex y de Caracteres

6.1 Cadenas de Caracteres

Las siguientes palabras están disponibles para manipular cadenas de caracteres:
N.T.> Cuando se dice "añade" se entiende que es por la derecha. Cuando sea por la izquierda se indicará expresamente.

```
&$              ( $1 $2 --> $3 )
                    Añade $2 a $1
!append$        ( $1 $2 --> $3 )
                    Lo mismo que &$, excepto que intentará la
                    concatenación "in situ" si no hay suficiente memoria
                    para la nueva cadena y el objetivo está en tempob
                    (área de memoria de objetos temporales)
$>ID            ( $nombre --> Id )
                    Convierte el objeto cadena en un objeto nombre
&$SWAP          ( ob $1 $2 --> $3 ob )
                    Añade $2 a $1 y luego intercambia (swap) en la pila
                    el resultado con ob
1-#1-SUB$       ( $ # --> $' )
                    Donde $' = caracteres 1 hasta #-1 de $
>H$             ( $ chr --> $' )
                    Añade chr a $ por la izquierda
>T$             ( $ chr --> $' )
                    Añade chr a $
AND$            ( $1 $2 --> $1 AND $2 )
                    AND (y) lógico Bit a Bit entre dos cadenas
APPEND_SPACE    ( $ --> $' )
                    Añade un espacio a $
Blank$          ( # --> $  )
                    Crea una cadena de # espacios
CAR$            ( $ --> chr | $ )
                    Devuelve el primer carácter de $ o NULL$ si $ es nulo
CDR$            ( $ --> $' )
                    $' es $ menos el primer carácter. Devuelve NULL$ si $
                    es nulo
COERCE$22       ( $ --> $' )
                    Si $ tiene más de 22 caracteres trunca a 21
                    caracteres y añade "..."
DECOMP$         ( ob --> $ )
                    Descompila el objeto para mostrarlo en la pila
DO>STR          ( ob --> $ )
                    Versión interna de ->STR
DROPNULL$       ( ob --> NULL$ )
                    Elimina el objeto (Drop) de la pila y devuelve una
                    cadena de longitud cero
DUP$>ID         ( $nombre --> $nombre Id )
                    Duplica (Dup) y convierte el objeto cadena en un
                    objeto nombre.
DUPLEN$         ( $ --> $ #longitud )
                    Devuelve $ y su longitud
DUPNULL$?       ( $ --> $ flag )
                    Devuelve TRUE si $ es de longitud cero
EDITDECOMP$     ( ob --> $ )
                    Descompila el objeto para editarlo
JstGETTHEMESG   ( # --> $ )
                    Obtiene mensaje número # de la tabla de mensajes
```

```
ID>$            ( ID --> $nombre )
                    Convierte el objeto nombre en una cadena
LAST$           ( $ # --> $' )
                    Devuelve los últimos # caracteres de $
LEN$            ( $ --> #longitud )
                    Devuelve la longitud de $
NEWLINE$&$      ( $ --> $' )
                    Añade "\0a" a $ (un salto de línea)
NULL$           ( --> $ )
                    Devuelve una cadena vacía
NULL$?          ( $ --> flag )
                    Devuelve TRUE si $ es de longitud cero
NULL$SWAP       ( ob --> $ ob )
                    Intercambia (swap) una cadena vacía con ob
NULL$TEMP       ( --> $ )
                    Crea una cadena vacía en TEMPOB (área de objetos
                    temporales)
OR$             ( $1 $2 --> $3 )
                    OR (o) lógico Bit a Bit entre dos cadenas
OVERLEN$        ( $ ob --> $ ob #longitud )
                    Devuelve la longitud del $ del nivel 2
POS$            ( $buscar_en $a_buscar #inicio --> #posición )
                    Devuelve la #posición (#0 si no lo encuentra) de
                    $a_buscar dentro de $buscar_en empezando por la
                    posición #inicio de $buscar_en y buscando de
                    izquierda a derecha
POS$REV         ( $buscar_en $a_buscar #inicio --> #posición )
                    Devuelve la #posición (#0 si no lo encuentra) de
                    $a_buscar dentro de $buscar_en empezando por la
                    posición #inicio de $buscar_en y buscando de derecha
                    a izquierda (N.T.>#inicio=1 es primer carácter izq)
PromptIdUtil    ( id ob -> $ )
                    Devuelve una cadena de la forma "id: ob"
SEP$NL          ( $ --> $2 $1 )
                    Divide $ por donde haya un carácter "salto de línea"
SUB$            ( $ #inicio #fin --> $' )
                    Devuelve una subcadena de $
SUB$1#          ( $ #pos --> # )
                    Devuelve un entero binario con el valor del carácter
                    en la posición #pos de $
SUB$SWAP        ( ob $ #inicio #fin --> $' ob )
                    Devuelve una subcadena de $ y hace SWAP con ob
SWAP&$          ( $1 $2 --> "$2$1" )
                    Añade $1 a $2
TIMESTR         ( %fecha %hora --> "WED 03/30/90 11:30:15A" )
                    Devuelve la cadena con la hora y fecha (como la
                    palabra de usuario TSTR)
XOR$            ( $1 $2 --> $3 )
                    XOR (o exclusivo) lógico Bit a Bit entre dos cadenas
a%>$            ( % --> $ )
                    Convierte % en $ usando el modo actual de mostrar
                    números en pantalla
a%>$,           ( % --> $ )
                    Convierte % en $ usando el modo actual de mostrar
                    números en pantalla. (Igual que a%>$ pero sin comas)
palparse        ( $ --> ob TRUE         )
                ( $ --> $ #pos $' FALSE )
                    Analiza la cadena y la convierte en un objeto y
                    devuelve TRUE o devuelve la posición de error y FALSE
```

6.2 Cadenas Hex

N.T.> Cuando digo añade, no es una suma sino que se concatena, como en las cadenas de caracteres.

```
#>%             ( hxs --> % )
                    Convierte el hxs en real
%>#             ( % --> hxs )
                    Convierte el real en hxs
&HXS            ( hxs1 hxs2 --> hxs3 )
                    Añade hxs2 a hxs1
2HXSLIST?       ( { hxs1 hxs2 } --> #1 #2 )
                    Convierte una lista de dos hxs en dos enteros
                    binarios. Produce el error "Valor del Argumento
                    Incorrecto" si la entrada es inválida
HXS#HXS         ( hxs1 hxs2 --> %flag )
                    Devuelve %1 si hxs1 <> hxs2, si no, %0
HXS>#           ( hxs --> # )
                    Convierte los 20 bits inferiores de hxs en entero
                    binario
HXS>$           ( hxs --> $ )
                    Hace hxs>$, luego le añade el carácter de la base
HXS>%           ( hxs --> % )
                    Convierte la cadena hex en un número real
HXS<HXS         ( hxs1 hxs2 --> %flag )
                    Devuelve %1 si hxs<hxs2, si no, %0
HXS>HXS         ( hxs1 hxs2 --> %flag )
                    Devuelve %1 si hxs1>hxs2, si no, %0
HXS>=HXS        ( hxs1 hxs2 --> %flag )
                    Devuelve %1 si hxs1>=hxs2, si no, %0
HXS<=HXS        ( hxs1 hxs2 --> %flag )
                    Devuelve %1 si hxs1<=hxs2, si no, %0
LENHXS          ( hxs --> #longitud )
                    Devuelve el número de nibbles en hxs
NULLHXS         ( --> hxs )
                    Devuelve una cadena hex de longitud cero
SUBHXS          ( hxs #m #n --> hxs' )
                    Devuelve la subcadena
```

Los enteros binarios del RPL de usuario son realmente cadenas hex. Las siguientes palabras suponen cadenas hex de 64 o menos bits y devuelven el resultado de acuerdo con el ancho de palabra actual:

```
bit/            ( hxs1 hxs2 --> hxs3 )
                    Divide hxs1 por hxs2
bit%#/          ( % hxs --> hxs' )
                    Divide % por hxs y devuelve hxs
bit#%/          ( hxs % --> hxs' )
                    Divide hxs por % y devuelve hxs
bit*            ( hxs1 hxs2 --> hxs3 )
                    Multiplica hxs1 por hxs2
bit%#*          ( % hxs --> hxs' )
                    Multiplica % por hxs y devuelve hxs
bit#%*          ( hxs % --> hxs' )
                    Multiplica hxs por % y devuelve hxs
bit+            ( hxs1 hxs2 --> hxs3 )
                    Suma hxs1 a hxs2
```

```
bit%#+          ( % hxs --> hxs' )
                    Suma % a hxs y devuelve hxs
bit#%+          ( hxs % --> hxs' )
                    Suma hxs a % y devuelve hxs
bit-            ( hxs1 hxs2 --> hxs3 )
                    Resta hxs2 de hxs1
bit%#-          ( % hxs --> hxs' )
                    Resta % de hxs y devuelve hxs
bit#%-          ( hxs % --> hxs' )
                    Resta hxs de % y devuelve hxs
bitAND          ( hxs1 hxs2 --> hxs3 )
                    AND (y) lógico bit a bit
bitASR          ( hxs --> hxs' )
                    Desplazamiento aritmético a la derecha de un bit
bitOR           ( hxs1 hxs2 --> hxs3 )
                    OR (o) lógico bit a bit
bitNOT          ( hxs1 hxs2 --> hxs3 )
                    NOT (no) lógico bit a bit
bitRL           ( hxs --> hxs' )
                    Rotación a la izquierda de un bit
bitRLB          ( hxs --> hxs' )
                    Rotación a la izquierda de un byte
bitRR           ( hxs --> hxs' )
                    Rotación a la derecha de un bit
bitRRB          ( hxs --> hxs' )
                    Rotación a la derecha de un byte
bitSL           ( hxs --> hxs' )
                    Desplazamiento a la izquierda de un bit
bitSLB          ( hxs --> hxs' )
                    Desplazamiento a la izquierda de un byte
bitSR           ( hxs --> hxs' )
                    Desplazamiento a la derecha de un bit
bitSRB          ( hxs --> hxs' )
                    Desplazamiento a la derecha de un byte
bitXOR          ( hxs1 hxs2 --> hxs3 )
                    XOR (o exclusivo) lógico bit a bit
```

Control del Tamaño de Palabra:

WORDSIZE	(--> #)	Devuelve el tamaño de palabra de los enteros binarios de usuario.
dostws	(# -->)	Almacena el tamaño de palabra binaria.
hxs>$	(hxs --> $)	Convierte la cadena hex a una cadena de caracteres usando el modo de pantalla y el tamaño de palabra actuales.

7. Números Reales

Los números reales se escriben con % y los números reales extendidos se escriben con %%.

7.1 Reales Incorporados

Los siguientes números reales y reales extendidos están incorporados:

```
%%.1    %%4    %-8          %11    %21    %5
%%.4    %%5    %-9          %12    %22    %6
%%.5    %%60   %-MAXREAL    %13    %23    %7
%%0     %%7    %-MINREAL    %14    %24    %8
%%1     %-2    %.1          %15    %25    %MAXREAL
%%10    %-3    %.5          %16    %26    %MINREAL
%%12    %-4    %0           %17    %27    %PI
%%2     %-5    %1           %180   %3     %e
%%2PI   %-6    %10          %2     %360   %-1
%%3     %-7    %100         %20    %4
```

7.2 Funciones de Números Reales

En los diagramas de pila que siguen a continuación, %1 y %2 se refieren a dos números reales diferentes, NO a los números reales uno y dos.

```
%%*             ( %%1 %%2 -->  %%3 )
                  Multiplica dos reales extendidos
%%*ROT          ( ob1 ob2 %%1 %%2 --> ob2 %%3 ob1 )
                  Multiplica dos reales extendidos,
                  luego hace ROT
%%*SWAP         ( ob %%1 %%2 --> %%3 ob )
                  Multiplica dos reales extendidos,
                  luego hace SWAP
%%*UNROT        ( ob1 ob2 %%1 %%2 --> %%3 ob1 ob2 )
                  Multiplica dos reales extendidos,
                  luego hace UNROT
%%+             ( %%1 %%2 --> %%3 )
                  Suma dos reales extendidos
%%-             ( %%1 %%2 --> %%3 )
                  Resta
%%ABS           ( %% --> %%' )
                  Valor absoluto
%%ACOSRAD       ( %% --> %%' )
                  Arco-coseno usando radianes
%%ANGLE         ( %%x %%y --> %%ángulo )
                  Angulo usando el modo actual de ángulos de %%x %%y
%%ANGLEDEG      ( %%x %%y --> %%ángulo )
                  Angulo usando grados sexagesimales de %%x %%y
```

```
%%ANGLERAD      ( %%x %%y --> %%ángulo )
                    Angulo en radianes de %%x %%y
%%ASINRAD       ( %% --> %%' )
                    Arco-seno con radianes
%%CHS           ( %% --> %%' )
                    Cambiar signo
%%COS           ( %% --> %%' )
                    Coseno
%%COSDEG        ( %% --> %%' )
                    Coseno con grados sexagesimales
%%COSH          ( %% --> %%' )
                    Coseno hiperbólico
%%COSRAD        ( %% --> %%' )
                    Coseno con radianes
%%EXP           ( %% --> %%' )
                    e^x
%%FLOOR         ( %% --> %%' )
                    El mayor entero <= x
%%H>HMS         ( %% --> %%' )
                    Horas decimales a hh.mmss
%%INT           ( %% --> %%' )
                    Parte entera
%%LN            ( %% --> %%' )
                    ln(x)
%%LNP1          ( %% --> %%' )
                    ln(x+1)
%%MAX           ( %%1 %%2 --> %%3 )
                    Devuelve el mayor de dos %%s
%%P>R           ( %%radio %%ángulo --> %%x %%y )
                    Conversión de polar a rectangular
%%R>P           ( %%x %%y --> %%radio %%ángulo )
                    Conversión de rectangular a polar
%%SIN           ( %% --> %%' )
                    Seno
%%SINDEG        ( %% --> %%' )
                    Seno con grados sexagesimales
%%SINH          ( %% --> %%' )
                    Seno hiperbólico
%%SQRT          ( %% --> %%' )
                    Raíz cuadrada
%%TANRAD        ( %% --> %%' )
                    Tangente con radianes
%%^             ( %%1 %%2 --> %%3 )
                    Exponencial
%+              ( %1 %2 --> %3 )
                    Suma
%+SWAP          ( ob %1 %2 --> %3 ob )
                    Suma, luego SWAP
%-              ( %1 %2 --> %3 )
                    Resta
%1+             ( % --> %+1 )
                    Suma uno
%1-             ( % --> %-1 )
                    Resta uno
```

```
%>#             ( % --> hxs )
                    Convierte el real en entero binario
%>%%            ( % --> %% )
                    Convierte el real en real extendido
%%>%            ( %% --> % )
                    Convierte el real extendido en real
%>%%-           ( %1 %2 --> %%3 )
                    Convierte dos % en %% y luego los resta
%>%%1           ( %x --> %% )
                    Convierte % en %% y luego lo invierte (1/x)
%>%%ANGLE       ( %x %y --> %%ángulo )
                    Angulo en el modo actual de ángulos
%>%%SQRT        ( % --> %% )
                    Convierte % en %% y luego raíz cuadrada (sqrt(x))
%>%%SWAP        ( ob % --> %% ob )
                    Convierte % en %% y luego hace SWAP
%>C%            ( %real %imaginaria --> C% )
                     Conversión de real en complejo
%>HMS           ( % --> %hh.mmss )
                    Horas decimales a hh.mmss
%ABS            ( % --> %' )
                    Valor absoluto
%ABSCOERCE      ( % --> # )
                    Valor absoluto y luego lo convierte en #
%ACOS           ( % --> %' )
                    Arco-coseno
%ACOSH          ( % --> %' )
                    Arco-coseno hiperbólico
%ALOG           ( % --> %' )
                     10^x
%ANGLE          ( %x %y --> %ángulo )
                    Angulo con el modo actual de ángulos de %x %y
%ASIN           ( % --> %' )
                    Arco-seno
%ASINH          ( % --> %' )
                    Arco-seno hiperbólico
%ATAN           ( % --> %' )
                    Arco-tangente
%ATANH          ( % --> %' )
                    Arco-tangente hiperbólica
%CEIL           ( % --> %' )
                    El siguiente entero más grande
%CH             ( %1 %2 --> %3 )
                    Variación porcentual
%CHS            ( % --> %' )
                    Cambio de signo
%COMB           ( %m %n -> %COMB(m,n) )
                    Combinaciones de m elementos tomados de n en n
%COS            ( % --> %' )
                    Coseno
%COSH           ( % --> %' )
                    Coseno hiperbólico
%D>R            ( % --> %' )
                    Grados sexagesimales a radianes
%EXP            ( % --> %' )
                    e^x
%EXPM1          ( % --> %' )
                    e^x-1
```

```
%EXPONENT       ( % --> %' )
                    Devuelve el exponente
%FACT           ( % --> %! )
                    Factorial
%FLOOR          ( % --> %' )
                    El mayor entero <= x
%FP             ( % --> %' )
                    Parte Fraccionaria
%HMS+           ( %1 %2 --> %3 )
                    Suma de HH.MMSS
%HMS-           ( %1 %2 --> %3 )
                    Resta de HH.MMSS
%HMS>           ( % --> %' )
                    Convierte hh.mmss a horas decimales
%IP             ( % --> %' )
                    Parte entera
%IP>#           ( % --> # )
                    IP(ABS(x)) convertido en #
%LN             ( % --> %' )
                    ln(x)
%LNP1           ( % --> %' )
                    ln(x+1)
%LOG            ( % --> %' )
                    Logaritmo base 10
%MANTISSA       ( % --> %' )
                    Devuelve la mantisa
%MAX            ( %1 %2 --> % )
                    Devuelve el mayor de dos reales
%MAXorder       ( %1 %2 --> %mayor %menor )
                    Ordena dos números
%MIN            ( %1 %2 --> % )
                    Devuelve el menor de dos reales
%MOD            ( %1 %2 --> %3 )
                    Devuelve %1 MOD %2
%NFACT          ( % --> %' )
                    Factorial
%NROOT          ( %1 %2 --> %3 )
                    Raíz enésima
%OF             ( %1 %2 --> %3 )
                    Devuelve el porcentaje de %1 que es %2
%PERM           ( %m %n --> %PERM(%m,%n) )
                    Devuelve las variaciones de %m elementos tomados de
                    %n en %n
%POL>%REC       ( %x %y --> %radio %ángulo )
                    Conversión rectangular a polar
%R>D            ( %radianes --> %grados )
                    Radianes a grados sexagesimales
%RAN            ( --> %aleatorio )
                    Número aleatorio
%RANDOMIZE      ( %semilla --> )
                    Actualiza la semilla de números aleatorios usando el
                    reloj del sistema si %semilla es %0
%REC>%POL       ( %radio %ángulo --> %x %y )
                    Conversión de polar a rectangular
%SGN            ( % --> %' )
                    Signo: devuelve -1, 0 o 1 dependiendo del signo del
                    argumento
%SIN            ( % --> %' )
                    Seno
```

```
%SINH           ( % --> %' )
                    Seno hiperbólico
%SPH>%REC       ( %r %th %ph --> %x %y %z )
                    Conversión de coordenadas esférica a rectangulares
%SQRT           ( % --> %' )
                    Raíz cuadrada
%T              ( %1 %2 --> %3 )
                    Porciento total
%TAN            ( % --> %' )
                    Tangente
%TANH           ( % --> %' )
                    Tangente hiperbólica
%^              ( %1 %2 --> %3 )
                    Exponencial
2%%>%           ( %%1 %%2 --> %1 %2 )
                    Conversión real extendido a real
2%>%%           ( %1 %2 --> %%1 %%2 )
                    Conversión real a real extendido
C%>%            ( C% --> %real %imag )
                    Conversión complejo a real
DDAYS           ( %fecha1 %fecha2 --> %diferencia )
                    Días entre dos fechas en formato DMY (Día/Mes/Año)
DORANDOMIZE     ( % --> )
                    Actualiza semilla del generador de números aleatorios
RNDXY           ( %número %posiciones --> %número' )
                    Redondea %número a %posiciones
TRCXY           ( %número %posiciones --> %número' )
                    Trunca %número a %posiciones
SWAP%>C%        ( %imaginario %real --> C% )
                    Conversión real a complejo
```

8. Números Complejos

Los números complejos se representan por C% y los números complejos extendidos por C%%

8.1 Números Complejos Incorporados

```
C%0             (0,0)
C%1             (1,0)
C%-1            (-1,0)
C%%1            (%%1,%%0)
```

8.2 Palabras de Conversión

```
%>C%            ( %real %imag --> C% )
%%>C%%          ( %%real %%imag --> C%% )
%%>C%           ( %%real %%imag --> C% )
C%>%            ( C% --> %real %imag )
C%%>%%          ( C%% --> %%real %%imag )
C%%>C%          ( C%% --> C% )
C%>%%C          ( C% --> %%real %%imag )
C%>%%SWAP       ( C% --> %%imag %%real )
C>Im%           ( C% --> %imag )
C>Re%           ( C% --> %real )
```

8.3 Funciones de Complejos

```
C%1/            ( C% --> C%' )
                        Inverso
C%ABS           ( C% --> % )
                        Devuelve SQRT(x^2+y^2) de (x,y)
C%ACOS          ( C% --> C%' )
                        Arco-coseno
C%ALOG          ( C% --> C%' )
                        Antilogaritmo base 10
C%ARG           ( C% --> %)
                        Devuelve ANGULO(x,y) de (x,y)
C%ASIN          ( C% --> C%' )
                        Arco-seno
C%ATAN          ( C% --> C%' )
                        Arco-tangente
C%C^C           ( C%1 C%2 --> C%3 )
                        Potencia
C%CHS           ( C% --> C%' )
                        Cambio de signo
C%%CHS          ( C%% --> C%%' )
                        Cambio de signo
C%CONJ          ( C% --> C%' )
                        Conjugado
C%%CONJ         ( C%% --> C%%' )
                        Conjugado
```

```
C%COS           ( C% --> C%' )
                        Coseno
C%COSH          ( C% --> C%' )
                        Coseno hiperbólico
C%EXP           ( C% --> C%' )
                        e^z
C%LN            ( C% --> C%' )
                        Logaritmo natural
C%LOG           ( C% --> C%' )
                        Logaritmo base 10
C%SGN           ( C% --> C%' )
                        Devuelve (x/SQRT(x^2+y^2),y/SQRT(x^2+y^2)
C%SIN           ( C% --> C%' )
                        Seno
C%SINH          ( C% --> C%' )
                        Seno hiperbólico
C%SQRT          ( C% --> C%' )
                        Raíz cuadrada
C%TAN           ( C% --> C%' )
                        Tangente
C%TANH          ( C% --> C%' )
                        Tangente hiperbólica
```

9. Formaciones

La notación [array] representa una formación real o compleja. [arry%] y [arryC%] representan una formación real y compleja respectivamente. {dims} significa una lista con las dimensiones de la formación, que puede ser o bien { #cols } o bien { #filas #cols }.

A menos que se indique lo contrario, las siguientes palabras NO comprueban las condiciones fuera de límites (p.ej. elementos especificados que no se encuentran dentro del margen de la formación actual).

```
ARSIZE          ( [array] --> #elementos )
                ( [array] --> {dims} )
GETATELN        ( # [array] --> ob TRUE )
                ( # [array] --> FALSE ) (no hay tal elemento)
MAKEARRY        ( {dims} ob --> [array] )
                 Crea una formación no-encadenada que tiene el mismo
                 tipo de elementos que ob. Todos los elementos se
                 inicializan con ob.
MATCON          ( [arry%] % --> [arry%]' )
                ( [arryC%] C% --> [arryC%]' )
                 Pone todos los elementos de la formación a % o C%
MATREDIM        ( [array] {dims} --> [array]' )
MATTRN          ( [array] --> [array]' )
MDIMS           ( [1-D array] --> #m FALSE )
                ( [2-D array] --> #m #n TRUE )
MDIMSDROP       ( [2-D array] --> #m #n )
                 -No usar MDIMSDROP con un vector!
OVERARSIZE      ( [array] ob --> [array] ob #elementos )
PULLREALEL      ( [arry%] # --> [arry%] % )
PULLCMPEL       ( [arryC%] # --> [arryC%] C% )
PUTEL           ( [arry%] % # --> [arry%]' )
                ( [arryC%] C% # --> [arryC%] )
PUTREALEL       ( [arry%] % # --> [arry%]' )
PUTCMPEL        ( [arryC%] C% # --> [arryC%]' )
```

10. Objetos Compuestos

Las palabras descritas en este capítulo se usan para manipular objetos compuestos - principalmente listas y secundarios. En la siguiente notación, el término "comp" se refiere a cualquier objeto compuesto. El término "#n" se refiere al número de objetos en un objeto compuesto y el término "#i" se refiere al índice de un objeto dentro de un compuesto. El término "flag" se refiere a TRUE o FALSE.

```
&COMP           ( comp comp' --> comp'' )
                 comp se concatena a comp'
2Ob>Seco        ( ob1 ob2 --> :: ob1 ob2 ; )
::N             ( obn ... ob1 #n --> :: obn ... ob1 ; )
::NEVAL         ( obn ... ob1 #n --> ? )
                 Hace ::N y luego evalúa el secundario
>TCOMP          ( comp ob --> comp' )
                 Se añade ob al final de comp
CARCOMP         ( comp --> ob )
                ( comp --> comp )
                 Devuelve el primer objeto del núcleo del compuesto.
                 Devuelve un comp nulo si el compuesto suministrado es
                 nulo.
CDRCOMP         ( comp --> comp' )
                ( comp --> comp )
                 Devuelve el núcleo del compuesto menos el primer
                 objeto. Devuelve un comp nulo si el compuesto
                 suministrado es nulo.
DUPINCOMP       ( comp --> comp obn ... ob1 #n )
DUPLENCOMP      ( comp --> comp #n )
DUPNULLCOMP?    ( comp --> comp flag ) TRUE si comp es nulo.
DUPNULL{}?      ( {list} --> {list} flag ) TRUE si {list} es nula.
EQUALPOSCOMP    ( comp ob --> #pos | #0 )
                 Devuelve el índice del primer objeto en comp que
                 coincide (EQUAL) con ob (ver NTHOF también)
Embedded?       ( ob1 ob2 --> flag )
                 Devuelve TRUE si ob2 está embebido en, o es igual
                 que, ob1; si no, devuelve FALSE.
INCOMPDROP      ( comp --> obn ... ob1 )
INNERCOMP       ( comp --> obn ... ob1 #n )
INNERDUP        ( comp --> obn ... ob1 #n #n )
LENCOMP         ( comp --> #n )
NEXTCOMPOB      ( comp #offset --> comp #offset' ob TRUE )
                ( comp #offset --> comp FALSE )
                 #offset es el offset en nibbles desde el principio
                 del comp hasta el enésimo objeto en el mismo.
                 Devuelve un nuevo #offset' y el siguiente objeto si
                 éste no es SEMI, si es SEMI devuelve comp y FALSE.
                 Usar #5 como #offset al principio del comp, para
                 saltarse el prólogo.
NTHCOMDDUP      ( comp #i --> ob ob )
NTHCOMPDROP     ( comp #i --> ob )
NTHELCOMP       ( comp #i --> ob TRUE )
                ( comp #i --> FALSE )
                 Devuelve FALSE si #i está fuera de margen.
NTHOF           ( ob comp --> #i | #0 ) Igual que SWAP EQUALPOSCOMP.
NULL::          ( --> :: ; ) (Devuelve un secundario nulo)
NULL{}          ( --> { } )  (Devuelve una lista Nula)
```

```
ONE{}N          ( ob --> { ob } )
Ob>Seco         ( ob --> :: ob ; )
POSCOMP         ( comp ob pred --> #i | #0 )
                 Si el objeto especificado "cuadra" con algún elemento
                 especificado del compuesto, donde "cuadrar" se define
                 como que el predicado (pred) especificado devuelve
                 TRUE cuando se aplica a algún elemento del compuesto
                 y al objeto ob, entonces POSCOMP devuelve el índice
                 de izquierda a derecha del elemento dentro del
                 compuesto o cero. Por ejemplo, para encontrar el
                 primer real menor que 5 en una lista de reales:

                              :: {list} 5 ' %< POSCOMP ;

PUTLIST         ( ob #i {list} --> {list}' ) (Supone que 0<#i<=#n)
SUBCOMP         ( comp #m #n --> comp' ) (Devuelve un subcompuesto)
                  IF #m > #n THEN comp' es nulo
                  IF #m=0    THEN #m se pone a 1
                  IF #n=0    THEN #n se pone a 1
                  IF #m > LONGitud(comp) THEN comp' es nulo
                  IF #n > LONG(comp) THEN #n se pone a LONG(comp)
SWAPINCOMP      ( comp obj --> obj obn ... ob1 #n )
THREE{}N        ( ob1 ob2 ob3 --> { ob1 ob2 ob3 } )
TWO{}N          ( ob1 ob2 --> { ob1 ob2 } )
{}N             ( obn ... ob1 #n --> {list} )
apndvarlst      ( {list} ob --> {list}' )
                 Añade ob a la lista si ob no se encuentra dentro de
                 la lista
matchob?        ( ob comp --> ob TRUE )
                ( ob comp --> FALSE )
                 Determina si ob es igual (EQUAL) a algún elemento del
                 comp
```

11. Objetos Etiquetados

Están disponibles las siguientes palabras para manipular objetos etiquetados. Recuerda que un objeto puede tener etiquetas múltiples.

```
%>TAG           ( ob % --> tagged )
                     Etiqueta ob con %

>TAG            ( ob $ --> tagged )
                     Etiqueta ob con $

ID>TAG          ( ob id/lam --> tagged )
                     Etiqueta ob con id

STRIPTAGS       ( tagged --> ob )
                     Elimina todas las etiquetas

STRIPTAGSl2     ( tagged ob' --> ob ob' )
                     Quita las etiquetas del objeto etiquetado del nivel 2

TAGOBS          ( ob $ --> tagged )
                ( ob1... obn {$1 ... $n} --> tagged_1 ... tagged_n)
                     Etiqueta un objeto, o varios objetos si hay una lista
                     de etiquetas en el nivel 1


USER$>TAG       ( ob $ --> tagged )
                     Etiqueta ob con $ (válido hasta 255 caracteres)
```

12. Objetos de Unidades

Cuando se comparan los objetos de unidad para ver si son consistentes dimensionalmente, se puede extraer una cadena hex, llamada "cadena de cantidad", usando la palabra U>NCQ. Esta cadena de cantidad contiene información acerca de que unidades hay y se puede comparar directamente con otra cadena de cantidad. Si coinciden las cadenas de cantidad, se puede decir que los dos objetos de unidad son dimensionalmente consistentes. U>NCQ también devuelve dos números reales extendidos que consisten en el número y un factor de conversión a las unidades básicas.

```
U>NCQ           ( unidad --> n%% cf%% qhxs )
                 Devuelve el número, el factor de conversión y la
                 cadena de cantidad hex (qhxs)
UM=?            ( unidad1 unidad2 --> %flag )
                 Devuelve %1 si dos obs unidades son iguales
UM#?            ( unidad1 unidad2 --> %flag )
                 Devuelve 1% si unidad1 <> unidad2
UM<?            ( unidad1 unidad2 --> %flag )
                 Devuelve %1 si unidad1 < unidad2
UM>?            ( unidad1 unidad2 --> %flag )
                 Devuelve %1 si unidad1 > unidad2
UM<=?           ( unidad1 unidad2 --> %flag )
                 Devuelve %1 si unidad1 <= unidad2
UM>=?           ( unidad1 unidad2 --> %flag )
                 Devuelve %1 si unidad1 >= unidad2
UM>U            ( % unidad --> unidad' )
                 Reemplaza la parte numérica de un objeto unidad
UM%             ( unidad %porcentaje --> unidad' )
                 Devuelve un porcentaje de un objeto unidad
UM%CH           ( unidad1 unidad2 --> % )
                 Devuelve la diferencia porcentual
UM%T            ( unidad1 unidad2 --> % )
                 Devuelve la fracción porcentual
UM+             ( unidad1 unidad2 --> unidad3 )
                 Suma
UM-             ( unidad1 unidad2 --> unidad3 )
                 Resta
UM*             ( unidad1 unidad2 --> unidad3 )
                 Multiplicación
UM/             ( unidad1 unidad2 --> unidad3 )
                 División
UM^             ( unidad1 unidad2 --> unidad3 )
                 Potencia
UM1/            ( unidad --> unidad' )
                 Inverso
UMABS           ( unidad --> unidad' )
                 Valor absoluto
UMCHS           ( unidad --> unidad' )
                 Cambio de signo
UMCONV          ( unidad1 unidad2 --> unidad1' )
                 Convierte unidad1 a las unidades de unidad2
UMCOS           ( unidad --> unidad' )
                 Coseno
UMMAX           ( unidad1 unidad2 --> unidad? )
                 Devuelve la mayor de unidad1 y unidad2
```

```
UMMIM           ( unidad1 unidad2 --> unidad? )
                 Devuelve la menor de unidad1 y unidad2
UMSI            ( unidad --> unidad' )
                 Convierte a las unidades básicas SI
UMSIN           ( unidad --> unidad' )
                 Seno
UMSQ            ( unidad --> unidad' )
                 Cuadrado
UMSQRT          ( unidad --> unidad' )
                 Raíz cuadrada
UMTAN           ( unidad --> unidad' )
                 Tangente
UMU>            ( unidad --> % unidad' )
                 Devuelve la parte numérica y la parte normalizada de
                 unidad de un objeto de unidad
UMXROOT         ( unidad1 unidad2 --> unidad3 )
                 unidad1^1/unidad2
UNIT>$          ( unidad --> $ )
                 Descompila un objeto unidad a una cadena de texto
```

13. Variables Temporales y Entornos Temporales

Una de las características implementadas en RPL es la capacidad de crear variables temporales ("variables locales", "variables lambda") cuyos nombres los da el programador y que se pueden destruir fácilmente cuando no se necesitan más. Estas variables temporales sirven para varios propósitos importantes. Primero de todo, se pueden usar para eliminar las manipulaciones de la pila dentro de un programa, lo que hace la tarea de seguir la pila (rastrearla) mucho más fácil y facilita la depuración. Además, son esenciales para la implementación de programas que toman un número indefinido de parámetros de la pila y quieren salvar uno o más de esos parámetros.

Las variables temporales se referencian con objetos identificadores temporales ("nombres locales") y la unión entre un objeto identificador temporal y su valor está soportada por estructuras en memoria llamadas entornos temporales. (Esto es el análogo RPL de la "unión lambda" en Lisp).

Los entornos temporales se apilan en orden cronológico. Esto permite que el programador pueda crear sus propias variables temporales "privadas" sin que exista la posibilidad que interfieran con aquellas creadas por otros. Cuando se ejecuta un objeto identificador temporal, se realiza una búsqueda a través de la pila de entornos temporales, empezando en el creado más recientemente y yendo hacia atrás por los entornos temporales anteriores, si hiciera falta. Cuando se produce una coincidencia entre el objeto identificador temporal que se está ejecutando y un objeto identificador temporal en uno de los entornos temporales, se sube a la pila de datos el objeto unido a ese identificador. La ejecución de un objeto identificador temporal sin ninguna unión es una condición de error.

Los procesos de crear un entorno temporal y asignarle valores iniciales a sus variables temporales se consiguen simultáneamente con el objeto suministrado BIND. BIND espera una lista de objetos identificadores temporales en la cima de la pila de datos y al menos tantos objetos (excluyendo la lista) en la pila como objetos identificadores temporales haya en la lista. BIND creará entonces un entorno temporal y unirá cada objeto identificador temporal de la lista con un objeto de la pila, eliminando ese objeto de la pila.

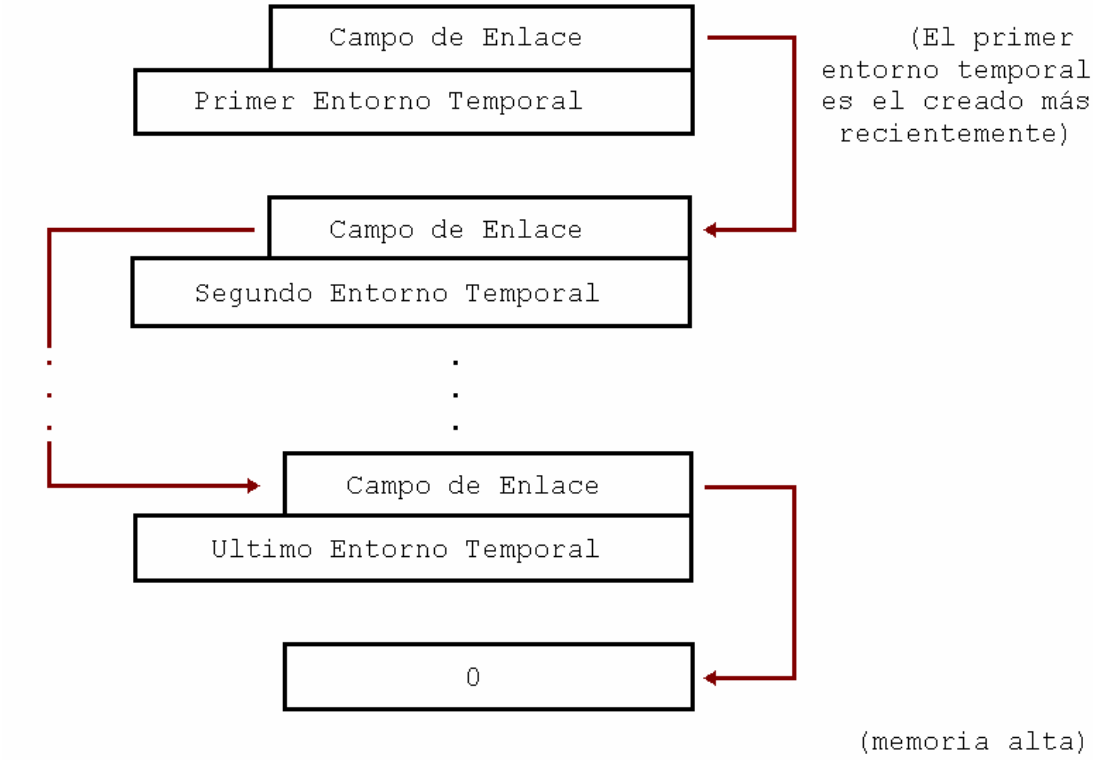
La ejecución posterior de cualquier objeto identificador temporal de la lista devolverá el objeto unido a él. El valor unido a un objeto identificador temporal se puede cambiar usando STO exactamente del mismo modo que un valor "unido" a un objeto identificador (nombre global).

La disolución de un entorno temporal se consigue con el objeto suministrado ABND (contracción de "abandonar"). ABDN elimina el entorno temporal que está en la cima de la pila de entornos temporales. No se pueden eliminar variables temporales individuales de un entorno temporal; se tiene que abandonar todo el entorno temporal.

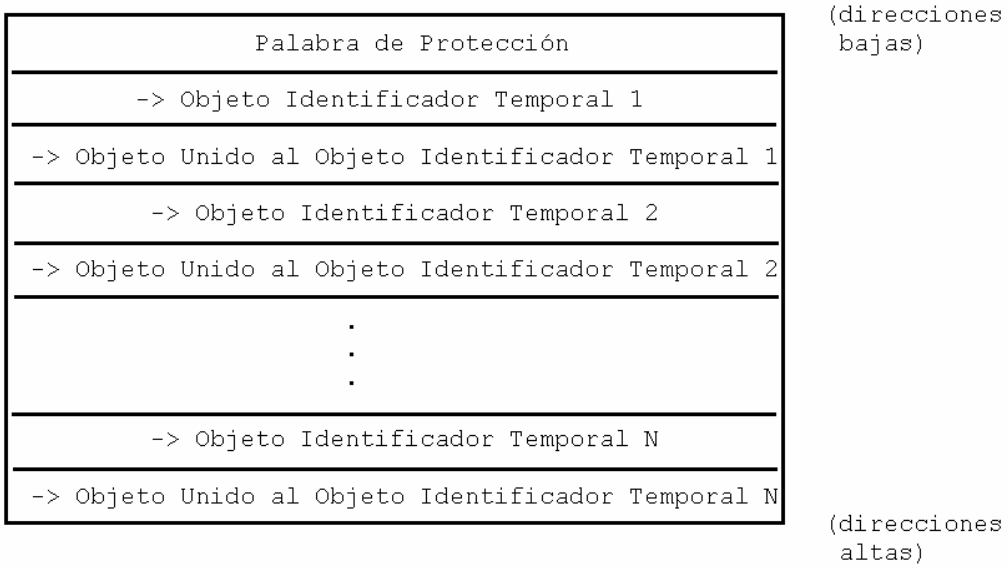
Observa que el compilador RPL no comprueba si hay un ABND asociado con cada BIND. Puedes incluir los dos en un mismo programa o los puedes poner en programas separados, como prefieras, sin ninguna otra restricción que las exigencias de la buena práctica de la programación estructurada. Esto también quiere decir que debes recordar incluir el ABND en algún punto, si no, puedes dejar entornos innecesarios en la memoria después de terminar la ejecución del programa. (En RPL de usuario no tienes tal libertad. La palabra estructural -> tiene BIND incorporado en ella y el analizador de la línea de comandos exige que haya el >> o el ' correspondiente que incluye el ABND.)

13.1 Estructura del Area de Entornos Temporales

A continuación se muestra la estructura del área de entornos temporales.



Cada entorno temporal consiste en una palabra de protección (el cuerpo de un objeto entero binario) que se usa en el control de errores, seguida por una secuencia de uno o más pares de punteros de objeto. El primer puntero de objeto en cada par es la dirección de un objeto identificador temporal y el segundo puntero de objeto en cada par es la dirección del objeto unido a ese objeto identificador temporal. Todos los punteros de objeto en un entorno temporal son actualizables. A continuación se muestra la estructura de cada entorno temporal dentro del área de entornos temporales:



13.2 Variables Temporales Con Nombre versus Sin Nombre

Las variables temporales normalmente se nombran con el correspondiente identificador temporal en la lista usada por BIND. Los nombres de la lista se usan en el mismo orden en que los objetos unidos aparecen en la pila -- el último identificador de la lista corresponde al objeto en el nivel 1, el identificador anterior al último corresponde al objeto en el nivel 2 y así sucesivamente. En el siguiente ejemplo, el entero binario ONE se une a Var1 y TWO se une a Var2

```
ONE TWO
{
  ' LAM Var1
  ' LAM Var2
}
BIND                ( Une ONE con la variable temporal Var1 y TWO
                      con la variable temporal Var2 )
...
LAM Var1            ( Llama a ONE desde Var1 )
...
LAM Var2            ( Llama a TWO desde Var2 )
...
' LAM Var1 STO      ( Almacena un nuevo objeto en Var1 )
...
ABND                ( Abandona el entorno temporal )
```

Los identificadores temporales pueden contener cualquier carácter de texto excepto que no deberías empezar los nombres con ' o # ya que tales nombres están reservados para los programas incorporados en ROM. Por razones parecidas, se recomienda que uses nombres que no entren en conflicto con los nombres generados por el usuario; un modo fácil de asegurarse de esto es incluir un carácter ilegal, tal como uno de los delimitadores de objetos, en tus nombres.

Si NO HAY NINGUNA POSIBILIDAD que se cree otro entorno temporal encima del entorno que vas a crear, se pueden usar los nombres nulos para ahorrar memoria. Hay varias palabras de utilidades que te permiten acceder a variables locales en el entorno superior mediante su número de posición, que es más rápido que la resolución ordinaria de un nombre. Por ejemplo, el ejemplo anterior sería así:

```
::
  ONE TWO
  { NULLLAM NULLLAM }
  BIND                ( Une ONE y TWO a dos variables temporales con
                        nombres nulos )
  ...
  2GETLAM             ( Llama a ONE desde la primera variable )
  ...
  1GETLAM             ( Llama a TWO desde la última variable )
  ...
  2PUTLAM             ( Almacena un nuevo objeto en la primera
                        variable )
  ...
  ABND                ( Abandona el entorno temporal )
;
```

La numeración comienza con la última variable temporal (o sea, en el mismo orden que el número del nivel de la pila).

13.3 Palabras Proporcionadas para las Variables Temporales

Se proporcionan las siguientes palabras para trabajar con variables temporales. El término "lamob" se usa en este caso para indicar un objeto llamado desde una variable temporal.

```
1ABNDSWAP       ( ob --> lamob ob )
                 Hace :: 1GETLAM ABND SWAP ;
1GETABND        ( --> lamob )
                 Hace :: 1GETLAM ABND ;
1GETLAM
 ...            ( --> ob )
22GETLAM         Devuelve el contenido del enésimo lam

1GETSWAP        ( ob --> lamob ob )
                 Hace :: 1GETLAM SWAP ;
1LAMBIND        ( ob --> )
                 Hace :: 1NULLLAM{} BIND ;
1NULLLAM{}      ( --> { NULLLAM } )
                 Devuelve una lista con un lam nulo
1PUTLAM
 ...            ( ob --> )
22PUTLAM         Almacena ob en el enésimo lam

2GETEVAL        ( --> ? )
                 Llama y evalúa el ob en el segundo lam
@LAM            ( id --> ob TRUE )
                ( id --> FALSE )
                 Llama lam por nombre, devuelve ob y TRUE si id
                 existe; si no, FALSE
ABND            ( --> )
                 Abandona el entorno de variables tempor. de la cima
BIND            ( ob ... { id ... } --> )
                 Crea un nuevo entorno de variables temporales
CACHE           ( obn ... ob1 n lam --> )
                 Salva n objetos más la cuenta n en un entorno
                 temporal, estando unido cada objeto al mismo
                 identificador lam. El último par tiene la cuenta.
DUMP            ( NULLLAM --> ob1..obn n )
                 DUMP es esencialmente el inverso de CACHE, PERO: SOLO
                 funciona cuando el nombre cacheado es NULLLAM y
                 SIEMPRE hace una recolección de basura.
DUP1LAMBIND     ( ob --> ob )
                 Hace DUP y luego 1LAMBIND
DUP4PUTLAM      ( ob --> ob )
                 Hace DUP y luego 4PUTLAM
DUPTEMPENV      ( --> )
                 Duplica el entorno temporal de la cima y borra la
                 palabra de protección.
GETLAM          ( #n --> ob )
                 Devuelve el objeto de la enésima variable temporal
NULLLAM         ( --> NULLLAM )
                 Nombre de variable temporal nulo
PUTLAM          ( ob #n --> )
                 Almacena ob en la enésima variable temporal
STO             ( ob id --> )
                 Almacena ob en la variable global/temp. con nombre.
STOLAM          ( ob id --> )
                 Almacena ob en la variable temporal con nombre.
```

13.4 Sugerencias para la Codificación

La característica DEFINE del compilador RPL se puede usar para combinar la legibilidad de las variables con nombre con la velocidad y eficacia de las variables con nombres nulos. Por ejemplo:

```
DEFINE RclCodigo  1GETLAM
DEFINE StoCodigo  1PUTLAM
DEFINE RclNombre  2GETLAM
DEFINE StoNombre  2PUTLAM
::
  ...
  { NULLLAM NULLLAM }
  BIND                ( Une dos objetos a las variables temporales 1
                        y 2 ambas con nombres nulos )
  ...
  RclCodigo           ( Llama el contenido de la última variable )
  ...
  RclNombre           ( Llama el contenido de la primera variable )
  ...
  StoCodigo           ( Almacena un objeto en la primera variable )
  ...
  ABND                ( Abandona el entorno temporal )
;
```

Si se va a usar un gran número de variables temporales sin nombre, he aquí un truco que ahorra código:

Reemplaza:

```
...
{
  NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM
  NULLLAM NULLLAM NULLLAM NULLLAM
} BIND
...
```

Con:

```
NULLLAM TWENTYFOUR NDUPN
TWENTYFOUR {}N BIND
```

El primer método gasta 67.5 bytes mientras que el último método gasta 15 bytes, -con lo que se ahorran 52.5 bytes!

También puedes usar TWENTYFOUR ' NULLAM CACHE, que es más corto aún y no requiere crear la lista de identificadores nulos en tempob. Observa, sin embargo, que CACHE añade una variable temporal extra (para mantener la cuenta), de modo que todos los números de posición de las variables difieren en uno con respecto a los métodos anteriores.

14. Chequeo de Argumentos

Cualquier objeto programa que un usuario puede ejecutar directamente debería asegurarse que están presentes el número y tipo correctos de argumentos para evitar problemas. Si en último término el objeto va a ser un comando de biblioteca, entonces debería seguir la convención de la estructura de los comandos (ver sección 2.6):

```
:: CK0 ... ; para comandos con 0 argumentos o

:: CK<n>&Dispatch tipo1 acción1
                  tipo2 acción2
                        ...
                  tipon acciónn
;
```

para comandos con <n> argumentos, donde tipoi es un código de tipo y accióni es el correspondiente destino del despacho/envío (o sea, acción correspondiente a realizar) para esa combinación de tipos o

```
:: CKN ... ;   para comandos que toman un número de argumentos
               especificado por un número real en el nivel 1
               (como PICK o ->LIST).
```

CK<n>&Dispatch es realmente una combinación de CK<n> y CK&DISPATCH1. Hay unos pocos comandos incorporados (p.ej. TYPE) que usan las dos palabras en vez de la forma combinada, pero todas las funciones algebraicas deben usar CK<n>&Dispatch ya que estas palabras también sirven para identificar el número de argumentos usados por una función.

Si un objeto no va a ser un comando de biblioteca, entonces debería tener la siguiente estructura:

```
:: CK0NOLASTWD ... ; en los programas con 0 argumentos o

:: CK<n>NOLASTWD  CK<n>&DISPATCH1   type1 action1
                                    type2 action2
                                       ...
                                    typen actionn
;
```

para programas con <n> argumentos o

```
:: CKNNOLASTWD ... ;
```

para programas que toman tantos argumentos como se especifica en el nivel 1

14.1 Número de Argumentos

Las siguientes palabras verifican que haya 0-5 argumentos en la pila y emiten el mensaje de error "Faltan Argumentos" si no es así.

```
CK0, CK0NOLASTWD              No se requiere ningún argumento
CK1, CK1NOLASTWD              Se requiere un argumento
CK2, CK2NOLASTWD              Se requieren dos argumentos
CK3, CK3NOLASTWD              Se requieren tres argumentos
CK4, CK4NOLASTWD              Se requieren cuatro argumentos
CK5, CK5NOLASTWD              Se requieren cinco argumentos
```

Cada palabra CK<n>... "marca" la pila por debajo del <ené>simo argumento y si está activada la capacidad de recuperación de argumentos, guarda una copia de los <n> argumentos en el área de salvar últimos argumentos. Si se produce un error que se controla por el controlador de errores del bucle externo, entonces se borra la pila hasta el nivel marcado (esto elimina cualquier objeto descarriado que no hubiera sido puesto allí por el usuario). Si está activado el sistema de recuperación de argumentos, entonces se reponen en la pila los argumentos guardados.

Cualquier CK<n> también anota el comando en el que se ejecuta y de nuevo lo hace por el controlador de errores del bucle externo, que usa el nombre del comando como parte del mensaje de error que se muestra. Un CK<n> sólo se debería usar en los comandos de biblioteca y debe ser el primer objeto del programa del comando. CK<n>NOLASWD no anota el comando y se puede usar en cualquier punto. Sin embargo, no es en general una buena idea ejecutar estas palabras excepto:

- al principio de un objeto ejecutado por el usuario o
- inmediatamente después de la ejecución de cualquier procedimiento de usuario.

Los procedimientos de usuario sólo se deberían ejecutar cuando la pila solo contiene objetos de usuario; el CK<n>NOLASTWD (usualmente CK0NOLASTWD) se ejecuta inmediatamente después del procedimiento de usuario para actualizar la marca de guardar la pila para proteger los resultados del procedimiento en la pila. Esto normalmente se hace conjuntamente con 0LASTOWDOB!, que borra la salvaguarda del comando hecha por el último CK<n> ejecutado dentro del procedimiento de usuario, de modo que ese comando no se identifique como el culpable de cualquier error posterior. Son palabras útiles para estos propósitos:

```
AtUserStack     que es :: CK0NOLASTWD 0LASTOWDOB! ;
CK1NoBlame      que es :: 0LASTOWDOB! CK1NOLASTWD ;
```

Los análogos de CK<n> y CK<n>NOLASTWD para los objetos que toman un número de argumentos especificados por la pila son CKN y CKNNOLASTWD. Ambas palabras comprueban que haya un número real en el nivel 1 de la pila y luego comprueban si hay ese número de objetos adicionales en la pila. La pila se marca en el nivel 2 y sólo se restaura el número real con LAST ARG.

14.2 Despachar según el Tipo de Argumento

Las palabras CK&DISPATCH1 y CK&DISPATCH0 proporcionan un mecanismo de despachar-por-tipo (las palabras CK<n>&Dispatch incluyen el mismo mecanismo, de modo que la siguiente discusión se les aplica también a ellas), que proporciona una bifurcación directa de acuerdo con los tipos de objeto de hasta cinco argumentos a la vez. Cada palabra va seguida por un número indefinido de pares de objetos. Cada par consiste en un entero binario o un puntero de objeto a un entero binario, seguido por cualquier objeto o puntero de objeto (el uso exclusivo de punteros de objeto garantiza el despacho más rápido):

```
        ...
    CK&DISPATCH1    #tipo1 acción1
                    #tipo2 acción2
                        ...
                    #tipon acción3
;
```

La secuencia de pares de objeto debe terminar con un SEMI (;).

CK&DISPATCH1 procede así: Por cada tipoi, desde el tipo1 hasta el tipon, si tipoi coincide con la configuración de la pila entonces se ejecuta la accióni, descartándose el resto de las palabras que contiene CK&DISPATCH1. Si no se encuentra ninguna coincidencia, se informa del error "Tipo de Argumento Incorrecto".

Si se hace una pasada completa por la tabla sin ninguna coincidencia, CK&DISPATCH1 hace una segunda pasada por la tabla, eliminando esta vez las etiquetas que pudieran haber en los objetos de la pila y se comparan los objetos que así quedan con los tipos requeridos.

La palabra CK&DISPATCH0 no realiza el segundo pase que quita las etiquetas. Esta palabra solo se debería usar cuando es importante encontrar un objeto etiquetado. El comportamiento general de la HP 48 es considerar las etiquetas como algo auxiliar de lo etiquetado y por tanto se debe usar CK&DISPATCH1 en la mayoría de los casos.

El entero binario tipoi se codifica nominalmente así:

```
#nnnnn
 |||||
 ||||+-- Tipo de argumento del nivel 1
 |||+--- Tipo de argumento del nivel 2
 ||+---- Tipo de argumento del nivel 3
 |+----- Tipo de argumento del nivel 4
 +------ Tipo de argumento del nivel 5
```

Cada "n" es un dígito hexadecimal que representa un tipo de objeto, como se muestra en la tabla de abajo. Así #00011 representa dos números reales; #000A0 indica un objeto de la clase simbólico (symb, id o lam) en el nivel 2 y cualquier tipo de objeto en el nivel 1. También hay tipos de objetos de dos dígitos, que terminan en F; por tanto, usar cualquiera de estos reduce el número total de argumentos que se pueden codificar en un solo entero de tipoi. Por ejemplo, #13F4F representa un número real en el nivel 3, un número real extendido en el nivel 2 y un complejo extendido en el nivel 1.

La siguiente tabla muestra los valores de los dígitos hex para cada tipo de argumento. La columna "# nombre" muestra el nombre del puntero de objeto del entero binario correspondiente que se puede usar para una función de un solo argumento. El capítulo de "Enteros Binarios" contiene una lista de los enteros binarios incorporados que se pueden usar en diversas combinaciones comunes de dos argumentos.

Valor	Argumento	# nombre	TYPE del Usuario
0	Cualquier Objeto	any	
1	Número Real	real	0
2	Número Complejo	cmp	1
3	Cadena de Caracteres	str	2
4	Formación	arry	3,4
5	Lista	list	5
6	Nombre Global	idnt	6
7	Nombre Local	lam	7
8	Secundario	seco	8
9	Simbólico	symb	9
A	Clase Simbólico	sym	6,7,9
B	Cadena Hex	hxs	10
C	Objeto Gráfico	grob	11
D	Objeto Etiquetado	TAGGED	12
E	Objeto Unidad	unitob	13
0F	Puntero ROM		14
1F	Entero Binario		20
2F	Directorio		15
3F	Real Extendido		21
4F	Complejo Extendido		22

5F	Formación Encadenada	23
6F	Carácter	24
7F	Objeto Código	25
8F	Biblioteca	16
9F	Backup	17
AF	Biblioteca de Datos	26
BF	Objeto externo1	27
CF	Objeto externo2	28
DF	Objeto externo3	29
EF	Objeto externo4	30

14.3 Ejemplos

Los comandos incorporados y otras palabras proporcionan buenos ejemplos del esquema de comprobar-y-despachar. A continuación la definición del comando de usuario STO:

```
:: CK2&Dispatch
    THIRTEEN  XEQXSTO                  ( 2:cualquier ob 1:ob etiquetado )
    SIX       :: STRIPTAGSl2 ?STO_HERE ;( 2:cualquiera   1:id )
    SEVEN     :: STRIPTAGSl2 STO ;     ( 2:cualquiera   1:lam )
    NINE      :: STRIPTAGSl2 SYMSTO ;  ( 2:cualquiera   1:symb )
    # 000c8   PICTSTO                  ( 2:grob         1:programa [PICT] )
    # 009f1   LBSTO                    ( 2:backup ob    1:número real )
    # 008f1   LBSTO                    ( 2:biblioteca   1:número real )
;
```

Como quiera que STO es un comando, empieza con CK2&Dispatch, que verifica que haya al menos dos argumentos presentes, los salva y salva también el comando STO para el control de errores, luego despacha (envía) a uno de los objetos acción listados en la tabla de despachos. Si el objeto del nivel uno está etiquetado, STO envía a la palabra XEQSTO. Si es un nombre global (id), STO ejecuta :: STRIPTAGSl2 ?STO_HERE ;, que está embebido directamente dentro del programa STO. Y así sucesivamente hasta la última opción, que es un despacho (envío) a LBSTO cuando los argumentos son una biblioteca en el nivel 2 y un número real en el nivel 1.

El comando TYPE proporciona un ejemplo de despacho que está situado en un punto distinto del inicio del comando. TYPE es un comando, pero su contador de argumentos y su despachador según el tipo de los argumentos están separados de modo que la última parte puede ser llamada por otras palabras del sistema que no quieren marcar la pila:

```
::
  CK1
  :: CK&DISPATCH0
        real             %0
        cmp              %1
        str              %2
        arry             XEQTYPEARRY
        list             %5
        id               %6
        lam              %7
        seco             TYPESEC ( 8, 18, o 19 )
        symb             %9
        hxs              %10
        grob             % 11
        TAGGED           % 12
        unitob           % 13
        rompointer       % 14
        THIRTYONE ( # )  % 20
        rrp              % 15
        # 3F ( %% )      % 21
        # 4F ( C%% )     % 22
        # 5F ( LNKARRY ) % 23
        # 6F ( CHR )     % 24
        # 7F ( CODE )    % 25
        library          % 16
```

```
            backup          % 17
            # AF            % 26 ( Biblioteca de Datos )
            any             % 27 ( external )
        ;
        SWAPDROP
    ;
```

Aquí se usa CK&DISPATCH0 aunque CK&DISPATCH1 también funcionaría ya que los objetos etiquetados están listados explícitamente en la tabla de despachos (envíos). Observa también que el último tipoi es "any" (cualquiera) lo que significa que el tipo 27 se devuelve cuando el objeto es cualquier tipo no listado previamente.

El programa "interior" (el que empieza después de CK1) es el cuerpo de la palabra del sistema XEQTYPE.

15. Estructuras de Control de Bucles

Hay disponibles dos tipos de estructuras de bucles - los bucles indefinidos y los bucles definidos.

15.1 Bucles Indefinidos

Los bucles indefinidos se construyen combinando las siguientes palabras RPL:

```
BEGIN ( --> )
  Copia el puntero del intérprete (la variable RPL I) en la pila de
  retornos. También llamada IDUP

UNTIL ( flag --> )
  Si flag es TRUE, elimina el puntero de la cima de la pila de
  retornos, si no, copia ese puntero al puntero del intérprete.

WHILE ( flag --> )
  Si flag es TRUE, entonces no hace nada. Si no, elimina el primer
  puntero de la pila de retornos y hace que el puntero del
  intérprete se salte los dos siguientes objetos.

REPEAT ( --> )
  Copia el primer puntero de la pila de retornos al puntero del
  intérprete

AGAIN ( --> )
```

El bucle WHILE es un bucle indefinido:

```
BEGIN
  <cláusula test>
WHILE
  <objeto del bucle>
REPEAT
```

El bucle WHILE ejecuta <cláusula test> y si el resultado es la bandera del sistema TRUE, ejecuta el <objeto del bucle> y repite; si no, sale y sigue justo pasado el REPEAT. El bucle WHILE no se ejecuta nunca si la primera vez que se evalúa la <cláusula test> devuelve FALSE.

La acción que realiza WHILE precisa que el <objeto del bucle> sea un sólo objeto. Sin embargo, el compilador RPL combina automáticamente todos los objetos que haya entre WHILE y REPEAT en un sólo objeto de modo que

```
BEGIN
  <cláusula test>
WHILE
  ob1 ... obn
REPEAT
```

se compila realmente como

```
BEGIN
  <cláusula test>
WHILE
  :: ob1 ... obn ;
REPEAT
```

Otro bucle indefinido común es BEGIN...UNTIL:

```
BEGIN
  <cláusula del bucle>
UNTIL
```

Este bucle se ejecuta al menos una vez, al contrario que el bucle WHILE, que no ejecuta su objeto del bucle si el test inicial es falso. La palabra UNTIL espera una bandera (TRUE o FALSE).

El bucle BEGIN...AGAIN no tiene test:

```
BEGIN
  <cláusula del bucle>
AGAIN
```

Para que se termine este bucle se precisa que suceda un error o que se manipule directamente la pila de retornos.

15.2 Bucles Definidos

Los bucles definidos con un contador de bucles se consiguen en RPL por medio de los DO Loop. La palabra DO toma dos objetos enteros binarios de la pila y almacena el objeto de la cima como el índice y el otro como el valor de parada en un entorno DoLoop especial. DO también copia el puntero del intérprete en la pila de retornos. Los entornos DoLoop están apilados, de modo que se pueden anidar indefinidamente. El índice más interno (superior) se reclama (se llama) con INDEX@; el índice en el segundo entorno se reclama (se llama) con JINDEX@. El valor de parada más interno (superior) está disponible vía ISTOP@.

Los complementos de DO son LOOP y +LOOP. LOOP incrementa el valor delíndice del entorno DoLoop más interno (el entorno DoLoop de la cima de la pila de entornos DoLoop); luego, si el (nuevo) valor es mayor que o igual al valor de parada, LOOP elimina el puntero de la cima de la pila de retornos y elimina el entorno DoLoop más interno. Si no, LOOP actúa copiando el puntero de la cima de la pila de retornos al puntero del intérprete. La forma estándar de un DoLoop es

```
parada inicio DO <cláusula del bucle> LOOP,
```

que ejecuta <cláusula del bucle> para cada valor del índice desde inicio hasta parada-1.

+LOOP es parecido a LOOP excepto que toma un entero binario de la pila e incrementa el contador del bucle en esa cantidad en lugar de en 1.

15.2.1 Palabras Proporcionadas

Se proporcionan las siguientes palabras para usarlas con los bucles DO. Las palabras marcadas con * no se reconocen como especiales por el compilador RPL, de modo que deberías incluir directivas del compilador para evitar mensajes de advertencia. Por ejemplo, #1+_ONE_DO puede ir seguido de (DO) que, para el compilador, hace de pareja con el próximo LOOP pero que no genera ningún código compilado (Ver RPLCOMP.DOC)

```
#1+_ONE_DO *     ( #fin --> )
                  Equivalente a #1+ ONE DO; usada a menudo para
                  ejecutar un bucle #fin veces
DO               ( #fin #inicio --> )
                  Empieza el bucle DO
DROPLOOP *       ( ob --> )
                  Hace un DROP y luego LOOP
DUP#0_DO *       ( # --> # )
                  Empieza el bucle # ...#0 DO
DUPINDEX@        ( ob --> ob ob #índice )
                  Hace DUP, luego devuelve el valor del índice del
                  entorno DoLoop más interno.
ExitAtLOOP       ( --> )
                  Almacena cero en el valor de parada del entorno
                  DoLoop más interno.
INDEX@           ( --> #índice )
                  Devuelve el índice del entorno DoLoop más interno
INDEX@#-         ( # --> #' )
                  Resta a # el valor del índice del entorno DoLoop más
                  interno.
```

```
INDEXSTO        ( # --> )
                 Almacena # como el índice del entorno DoLoop más
                 interno.
ISTOP@          ( --> #stop )
                 Devuelve el valor de parada del entorno DoLoop más
                 interno
ISTOPSTO        ( # --> )
                 Almacena el nuevo valor de parada en el el entorno
                 DoLoop más interno.
JINDEX@         ( --> #índice )
                 Devuelve el índice del segundo entorno DoLoop
LOOP            ( --> )
                 Termina una estructura de bucle
NOT_UNTIL *     ( flag --> )
                 Termina una estructura de bucle
ONE_DO *        ( #fin --> )
                 Comienza un Bucle #1...#fin DO
OVERINDEX@      ( ob1 ob2 --> ob1 ob2 ob1 #índice )
                 Hace OVER y luego devuelve el valor del índice del
                 entorno DoLoop más interno.
SWAPINDEX@      ( ob1 ob2 --> ob2 ob1 #índice )
                 Hace SWAP y luego devuelve el valor del índice del
                 entorno DoLoop más interno.
SWAPLOOP *      ( ob1 ob2 --> ob2 ob1 )
                 Hace SWAP y luego LOOP
ZEROISTOPSTO    ( --> )
                 Almacena cero como el valor de parada del entorno
                 DoLoop más interno
ZERO_DO *       ( #fin --> )
                 Comienza el bucle DO desde #0 hasta #fin
toLEN_DO        ( {list} --> {list} )
                 Comienza el bucle DO con inicio = #1 y con el valor
                 de parada = #número-de-elementos-de-la-lista+1
```

15.2.2 Ejemplos

```
FIVE ZERO
DO
  INDEX@
LOOP
```

Devuelve los valores:

```
#00000 #00001 #00002 #00003 #00004
```

La siguiente secuencia muestra cada uno de los elementos (hasta 8) de una lista de cadenas de caracteres en una línea distinta de la pantalla.

```
DUPLENCOMP
ONE_DO (DO)
  DUP INDEX@ NTHCOMPDROP
  INDEX@ DISPN
LOOP
```

Una versión más compacta que utiliza toLEN_DO:

```
toLEN_DO (DO)
  DUP INDEX@ NTHCOMPDROP
  INDEX@ DISPN
LOOP
```

Otra versión ligeramente más rápida, pues evita repetidas extracciones de los elementos de la lista:

```
INNERCOMP
#1+_ONE_DO (DO)
  INDEX@ DISPN
LOOP
```

Esta versión muestra los elementos en orden inverso al de las versiones anteriores.

16. Generación y Captura de Errores

El subsistema RPL de control de errores se invoca ejecutando la palabra ERRJMP, o sea, cuando un objeto de la clase procedimiento desea generar un error, ejecuta ERRJMP (probablemente después de actualizar los valores de ERROR y ERRNAME). La mecánica de ERRJMP se describirá más tarde.

16.1 Captura: ERRSET y ERRTRAP

RPL proporciona objetos procedimiento con la capacidad de interceptar la ejecución del subsistema de control de errores, o sea, capturar un error generado por un objeto que es menor (está en un nivel inferior) según el orden del entrelazado. Esta capacidad está disponible a través de los objetos incorporados ERRSET y ERRTRAP usados del siguiente modo:

```
:: ... ERRSET <objeto sospechoso> ERRTRAP <objeto si-error> ... ;
```

Arriba, un error generado por el <objeto sospechoso> será capturado. <objeto si-error> denota el objeto que se ejecutará si el <objeto sospechoso> genera un error. El algoritmo exacto es: Si <objeto sospechoso> genera un error, entonces continúa la ejecución en <objeto si-error>; si no, continúa la ejecución más allá de <objeto si-error>.

La acción del <objeto si-error> es completamente flexible; cuando <objeto si-error> toma el control, puede examinar los valores de ERROR y de ERRNAME para determinar si le concierne o no el error actual. Si no, puede sencillamente reiniciar el subsistema ejecutando ERRJMP. Si le concierne, puede decidir controlar el error, o sea, borrar tanto ERROR como ERRNAME y NO reiniciar el subsistema. También puede desactivar la ejecución del resto del programa (quizás vía RDROP).

Observa que durante la ejecución (normal) del <objeto sospechoso>, hay en algún sitio del "runstream" un puntero de objeto al siguiente ERRTRAP.

16.2 Acción de ERRJMP

Cuando un procedimiento RPL quiere iniciar un error, ejecuta ERRJMP que es el subsistema de control de errores. ERRJMP se recorre el RUNSTREAM desde el puntero del intérprete I hacia arriba a través de la pila de retornos buscando una trampa de errores. Específicamente, ERRJMP elimina los cuerpos de programas pendientes del RUNSTREAM hasta que encuentra uno en el que su primer elemento es un puntero de objeto que direcciona (apunta) a ERRTRAP (este cuerpo de programa puede corresponder a un nivel de la pila de retornos así como también al puntero del intérprete I). Entonces se SALTA el puntero de objeto a ERRTRAP y continúa la ejecución más allá de él (en el <objeto si-error>).

Observa, por tanto, que ERRTRAP sólo se ejecuta si <objeto sospechoso> termina sin generar ningún error; en este caso, ERRTRAP, entre otras cosas, se SALTARA el <objeto si-error> y continuará la ejecución más allá de él.

Si un procedimiento no está meramente de paso por un error que él no inició, su invocación de ERRJMP debe ir precedida por la ejecución de ERRORSTO, que almacena un número de error en un sitio especial del sistema. ERROR@ devuelve el número de error almacenado, que lo pueden usar las trampas de error para determinar si quieren controlar un error determinado. El número de error se almacena y se devuelve como un entero binario; los 12 bits más significativos del número representan el ID de Biblioteca de la biblioteca que contiene el mensaje de error y el resto de los bits indican el número de error dentro de la tabla de mensajes de la biblioteca.

16.3 La Palabra de Protección

Cada entorno temporal y cada entorno DoLoop tienen una palabra de protección. La única razón de la existencia de esta palabra de protección es permitir al subsistema de control de errores distinguir los entornos temporales y DoLoop que ya existían cuando se activó la trampa de error de aquellos que se crearon después que se activó la trampa de error. Por ejemplo, considera lo siguiente:

```
::
  ...
  { NULLLAM } BIND
  ...
  TEN ZERO DO
    ERRSET ::
      ...
      { NULLLAM } BIND
      ...
      FIVE TWO DO
        <procedimiento>
      LOOP
      ABND
    ;
    ERRTRAP
      :: "Falló el Procedimiento" FlashMsg ;
  LOOP
  ...
  ABND
  ...
;
```

Si <procedimiento> genera un error, entonces este error será atrapado por la palabra o secundario a continuación de ERRTRAP. Sin embargo, los entornos temporales y DoLoop internos se deben borrar de modo que el procedimiento externo tenga disponible los parámetros DoLoop y las variables locales correctas. La palabra de protección sirve para apoyar esta función.

ERRSET incrementa la palabra de protección del entorno temporal más interno y del entorno DoLoop más interno. Estos entornos más internos tendrán por consiguiente un valor de la palabra de protección distinto de cero. (DO y BIND siempre inicializan la palabra de protección a cero).

ERRTRAP y ERRJMP borran los entornos temporales y DoLoop (desde el primero (más interno) hacia el último (más externo)) hasta que, en ambos casos, encuentran uno con la palabra de protección distinta de cero, que entonces se decrementa. Por tanto, tanto si ERRJMP ejecuta un <objeto si-error> o ERRTRAP sigue la ejecución a continuación de <objeto si-error>, solo estarán presentes los entornos temporales y DoLoop que ya existían cuando se activó ERRSET .

Observa especialmente que la palabra de protección es más que un simple interruptor, ya que permite un número prácticamente ilimitado de trampas de error anidadas.

El ejemplo anterior es realmente una trampa de error bastante pobre - el código debería determinar cual fue el error y tomar la acción correspondiente. Se puede usar la palabra ERROR@ para saber que error se produjo. Los número de error corresponden a los números de mensajes - ver la tabla de mensajes en el apéndice A del "HP 48, Manual de Referencia del Programador"

16.4 Palabras de Error

Se suministran las siguientes palabras para el control de errores:

```
ABORT           ( --> )
                 Hace ERRORCLR y ERRJMP
DO#EXIT         ( msj# --> )
                 Almacena un nuevo número de error y ejecuta ERRJMP;
                 también ejecuta AtUserStack
'ERRJMP         ( --> ERRJMP )
                 Pone el objeto ERRJMP en la pila
ERRBEEP         ( --> )
                 Genera un pitido de error
ERRJMP          ( --> )
                 Invoca al subsistema de control de errores
ERROR@          ( --> # )
                 Devuelve el número de error actual
ERRORCLR        ( --> )
                 Almacena cero como el número de error
ERROROUT        ( # --> )
                 Almacena un nuevo número de error y hace ERRJMP
ERRORSTO        ( # --> )
                 Almacena un nuevo número de error
ERRTRAP         ( --> )
                 Se salta el siguiente objeto del "runstream"
```

17. Test y Control

Este capítulo examina las palabras relacionadas con el control del flujo: bifurcaciones condicionales e incondicionales y las palabras de test asociadas.

17.1 Banderas y Tests

TRUE y FALSE son objetos incorporados que son reconocidos por las palabras test como banderas para tomar decisiones de bifurcación. Las siguientes palabras crean o combinan banderas:

```
AND ( flag1 flag2  --> flag )
     Si flag1 y flag2 son ambos TRUE entonces TRUE si no, FALSE.

FALSE ( --> FALSE )
     Pone la bandera FALSE en la pila

FALSETRUE       ( --> FALSE TRUE )

FalseFalse      ( --> FALSE FALSE )

OR ( flag1 flag2  --> flag )
     Si flag1 o flag2 (o ambos) es TRUE entonces TRUE si no, FALSE.

ORNOT           ( flag1 flag2 --> flag3 )
     O lógico seguido de un NO lógico.

NOT ( flag --> flag' )
     Si flag es TRUE entonces FALSE si no, TRUE.

NOTAND          ( flag1 flag2 --> flag3 )
     NO lógico y luego Y lógico.

ROTAND          ( flag1 ob flag2 --> ob flag3 )
     Hace ROT y luego un Y lógico.

TRUE            ( --> TRUE )
     Coloca en la pila la bandera TRUE

TrueFalse       ( --> TRUE FALSE )

TrueTrue        ( --> TRUE TRUE )

XOR             ( flag1 flag2  --> flag )
     Si tanto flag1 y flag2 son TRUE o FALSE entonces FALSE si no,
     TRUE.

COERCEFLAG      ( TRUE --> %1 )
                ( FALSE --> %0 )
     Convierte una bandera del sistema en una bandera número real.
```

17.1.1 Tests de Objetos en General

Las siguientes palabras comprueban el tipo de objeto y la igualdad:

```
EQ ( ob1 ob2 --> flag )
  Si los objetos ob1 y ob2 son el mismo objeto, o sea, ocupan
  físicamente el mismo espacio en la memoria, entonces TRUE, si no,
  FALSE.

EQUAL ( ob1 ob2 --> flag )
  donde ob1 y ob2 no son objetos código primitiva. Si los objetos ob1
  y ob2 son el mismo entonces TRUE, si no, FALSE (esta palabra es el
  equivalente en RPL del sistema del comando en RPL de usuario SAME)

2DUPEQ ( ob1 ob2 --> ob1 ob2 flag )
  Devuelve TRUE si ob1 y ob2 tienen la misma dirección física.

EQOR ( flag1 ob1 ob2 --> flag2 )
  Hace EQ y luego un O lógico.

EQUALOR ( flag1 ob1 ob2 --> flag2 )
  Hace EQUAL y luego un O lógico.

EQOVER ( ob1 ob2 ob3 --> ob1 flag ob1 )
  Hace EQ y luego OVER.

EQUALNOT ( ob1 ob2 --> flag )
  Devuelve FALSE si ob1 es igual a ob2.
```

Las siguientes palabras comprueban el tipo de un objeto. Las palabras de la forma TYPE...? tienen un diagrama de pila (ob --> flag); las de la forma DTYPE...? o DUPTYPE...? duplican primero el objeto (ob --> ob flag).

Palabras Test	Tipo de Objeto
TYPEARRY?	formación
DTYPEARRY?	
DUPTYPEARRY?	
TYPEBINT?	entero binario
DUPTYPEBINT?	
TYPECARRY?	formación compleja
TYPECHAR?	carácter
DUPTYPECHAR?	
TYPECMP?	número complejo
DUPTYPECMP?	
TYPECOL?	programa
DTYPECOL?	
DUPTYPECOL?	

```
TYPECSTR?               cadena
DTYPECSTR?
DUPTYPECSTR?

TYPEEXT?                unidad
DUPTYPEEXT?

TYPEGROB?               objeto gráfico
DUPTYPEGROB?

TYPEHSTR?               cadena hex
DUPTYPEHSTR?

TYPEIDNT?               identificador (nombre global)
DUPTYPEIDNT?

TYPELAM?                identificador temporal (nombre local)
DUPTYPELAM?

TYPELIST?               lista
DTYPELIST?
DUPTYPELIST?

TYPERARRY?              formación real

TYPEREAL?               número real
DTYPEREAL?
DUPTYPEREAL?

TYPEROMP?               puntero ROM (nombre XLIB)
DUPTYPEROMP?

TYPERRP?                Directorio
DUPTYPERRP?

TYPESYMB?               Simbólico
DUPTYPESYMB?

TYPETAGGED?             Etiquetado
DUPTYPETAG?
```

17.1.2 Comparaciones de Enteros Binarios

Las siguientes palabras comparan enteros binarios, devolviendo TRUE o FALSE. La igualdad se comprueba en el sentido de EQUAL (no EQ). Se tratan todos los enteros binarios como "sin signo". Algunas de estas palabras también están disponibles combinadas con las palabras de "casos" (ver más adelante).

```
#=      ( # #' --> flag )       TRUE si # = #'.

#<>     ( # #' --> flag )       TRUE si # <> #' (no igual).

#0=     ( # --> flag )          TRUE si # = 0
```

```
#0<>    ( # --> flag )          TRUE si # <> 0

#<      ( # #' --> flag )       TRUE si # < #'

#>      ( # #' --> flag )       TRUE si # > #'

2DUP#<  ( # #' --> # #' flag )  TRUE si # < #'

2DUP#=  ( # #' --> # #' flag )  TRUE si # = #'

DUP#0=  ( # --> # flag )        TRUE si # = #0

DUP#1=  ( # --> # flag )        TRUE si # = #1

DUP#0<> ( # --> # flag )        TRUE si # <> #0

DUP#1=  ( # --> # flag )        TRUE si # = #1

DUP#<7  ( # --> # flag )        TRUE si # < #7

DUP%0=  ( % --> % flag )        TRUE si % = %0

ONE#>   ( # --> flag )          TRUE si # > #1

ONE_EQ  ( # --> flag )          TRUE si # es ONE (1)

OVER#>  ( # #' --> # flag )     TRUE si # > #'

OVER#0= ( # ob --> # ob flag )  TRUE si # es #0

OVER#<  ( # #' --> # flag )     TRUE si # > #'

OVER#=  ( # #' --> # flag )     TRUE si # = #'

OVER#>  ( # #' --> # flag )     TRUE si # < #'
```

17.1.3 Tests de Números Decimales

Las siguientes palabras comparan números reales, reales extendidos y complejos, devolviendo TRUE o FALSE.

```
%<      ( % %' --> flag )       TRUE si % < %'

%<=     ( % %' --> flag )       TRUE si % <= %'

%<>     ( % %' --> flag )       TRUE si % <> %'

%=      ( % %' --> flag )       TRUE si % = %'

%>      ( % %' --> flag )       TRUE si % > %'

%>=     ( % %' --> flag )       TRUE si % >= %'

%0<     ( % --> flag )          TRUE si % < 0
```

```
%0<>     ( % --> flag )          TRUE si % <> 0

%0=      ( % --> flag )          TRUE si % = 0

%0>      ( % --> flag )          TRUE si % > 0

%0>=     ( % --> flag )          TRUE si % >= 0


%%0<=    ( %% %%' --> flag )     TRUE si %% <= %%'

%%0<>    ( %% --> flag )         TRUE si %% <> 0

%%0=     ( %% --> flag )         TRUE si %% = 0

%%0>     ( %% --> flag )         TRUE si %% > 0

%%0>=    ( %% --> flag )         TRUE si %% >= 0

%%>      ( %% %%' --> flag )     TRUE si %% > %%'

%%>=     ( %% %%' --> flag )     TRUE si %% >= %%'

%%<=     ( %% %%' --> flag )     TRUE si %% <= %%'


C%%0=    ( C%% --> flag )        TRUE si C%% = (%%0,%%0)

C%0=     ( C% --> flag )         TRUE si C% = (0,0)
```

17.2 Palabras que Operan en el "Runstream"

N.T.> El Runstream es la secuencia de palabras (comandos, etc.) que se ejecutan unas detrás de otras.

En muchos casos es deseable interrumpir el orden entrelazado normal de ejecución e insertar objetos adicionales o saltarse otros en el runstream. Para estos propósitos se proporcionan las siguientes palabras.

```
'  ( --> ob )
```

Este es al análogo RPL del QUOTE del Lisp y es uno de los objetos de control más fundamentales, permitiendo posponer la evaluación de un objeto. De modo más preciso, supone que el cuerpo en la cima del RUNSTREAM no es uno vacío, o sea, que el puntero del intérprete no apunta a un SEMI; y (1) Si el siguiente objeto en el runstream es un objeto, entonces sube este objeto a la pila de datos y mueve el puntero del intérprete al siguiente objeto; (2) Si el siguiente objeto es un puntero de objeto, entonces sube lo apuntado a la pila de datos y salta también al siguiente objeto. Como ejemplo, la evaluación de los secundarios

```
:: # 3 # 4 SWAP ;      y      :: # 3 # 4 ' SWAP EVAL ;
```

dan ambos el mismo resultado.

'R (--> ob)

Si el objeto apuntado por el puntero en la cima de la pila de retornos (o sea, el primer elemento en el segundo cuerpo en el runstream) es un objeto, entonces 'R sube este objeto a la pila de datos y adelanta el puntero al siguiente objeto del mismo compuesto. Si el puntero apunta a un puntero de objeto que no apunta a SEMI, entonces sube lo apuntado a la pila de datos y adelanta de modo similar el puntero de la pila de retornos. Si lo apuntado es SEMI, entonces si el primer elemento en el segundo cuerpo en el runstream es un puntero de objeto a SEMI, entonces sube un secundario nulo a la pila de datos y no adelante el puntero de la pila de retornos. 'R es útil para definir operadores prefijos. Por ejemplo, supón que PREFIXSTO se define como :: 'R STO ; Entonces la secuencia PREFIXSTO FRED ANOTHEROBJECT subiría primero FRED a la pila de datos y luego ejecutaría STO, después de lo cual la ejecución continúa en ANOTHEROBJECT.

ticR (--> ob TRUE | FALSE)

Esta palabra funciona de modo parecido a 'R, excepto que devuelveuna bandera para indicar si se ha alcanzado el final del compuesto de la cima de la pila de retornos. O sea, si el puntero de la cima de la pila de retornos apunta a un puntero de objeto a SEMI, entonces ticR baja la pila de retornos y devuelve solo FALSE. Si no, devuelve el siguiente objeto del compuesto y TRUE, mientras adelanta el puntero de la pila de retornos al siguiente objeto.

>R (:: -->)

Inserta el cuerpo de :: en el runstream justo debajo del de la cima. (O sea, sube un puntero al cuerpo de :: a la pila de retornos). Un ejemplo de su uso es

```
:: ' :: <foo> ; >R <bar> ;
```

que provocará que, cuando se ejecute, <bar> se ejecuta antes que <foo>.

R> (--> ::)

Crea un objeto programa a partir cuerpo del compuesto apuntado por el puntero de la cima de la pila de retornos y sube el programa a la pila de datos y baja la pila de retornos. Ejemplo:

```
:: :: R> EVAL <foo> ; <bar> ;
```

que cuando se ejecuta, causará que <bar> se ejecute antes que <foo>.

R@ (--> ::)

Lo mismo que R> excepto que la pila de retornos no se baja.

RDROP (-->)

Baja la pila de retornos.

IDUP (-->)

Duplica el cuerpo de la cima del runstream. (O sea, sube la variable RPL I a la pila de retornos).

COLA (-->)

Suponiendo que el puntero del intérprete está apuntando a un objeto distinto de SEMI, COLA elimina el resto del cuerpo del programa pasado el objeto y ejecuta el objeto. Esto permite una eficiente recursión por la cola; la eficiencia se gana porque COLA se puede usar para evitar un excesivo crecimiento de los retornos pendientes. Un ejemplo de su uso es en una definición de factorial:

```
fact:       :: { LAM x } BIND # 1 factpair ABND
            ;

factpair:   :: LAM x #0= ?SEMI
               LAM x #* LAM x #1- ' LAM x
               STO COLA factpair
            ;
```

En este ejemplo, la importancia de COLA es que esté justo antes de factpair en la definición de factpair. Si no se usara, el cálculo de n! requeriría n niveles de la pila de retornos, que, cuando se terminara el cálculo, simplemente se irían bajando todos (ya que sus cuerpos estarían vacíos). Con la inclusión de COLA, la definición usa un número máximo fijo de niveles, independientemente del argumento de la función.

?SEMI (flag -->)

Sale del programa en curso si la bandera es TRUE.

?SEMIDROP (ob TRUE -->) o (FALSE -->)

Elimina ob si la bandera es TRUE; sale del programa en curso si la bandera es FALSE.

?SKIP (flag -->)

Si la bandera es TRUE, se salta el siguiente objeto a continuación de ?SKIP.

NOT?SEMI (flag -->)

Sale del programa en curso si la bandera es FALSE.

17.3 If/Then/Else

La capacidad if/then/else fundamental del RPL se proporciona por medio de las palabras RPIT y RPITE:

```
RPITE   ( flag ob1 ob2 --> ? )
```

Si flag es TRUE entonces se eliminan flag y ob2 y se EVALúa ob1, si no, se eliminan flag y ob1 y se EVALúa ob2. La expresión RPL

```
' <foo> ' <bar> RPITE
```

es equivalente a la expresión FORTH

```
IF <foo> ELSE <bar> THEN
```

```
RPIT    ( flag ob --> ? )
```

Si flag es TRUE entonces se elimina flag y se EVALúa ob, si no, simplemente se eliminan flag y ob. La expresión RPL

```
' <foo> RPIT
```

es equivalente a la expresión FORTH

```
 IF <foo> THEN
```

Sin embargo, también están disponibles las versiones prefijo de estas palabras y se usan más a menudo que las versiones sufijo:

```
IT      ( flag -->  )
```

Si flag es TRUE entonces ejecuta el siguiente objeto en el runstream; si no, se salta ese objeto. Por ejemplo,

```
DUPTYPEREAL? IT :: %0 %>C% ;
```

convierte un número real en un número complejo; no hace nada si el argumento no es un número real.

```
ITE     ( flag -->  )
```

Si flag es TRUE entonces ejecuta el siguiente objeto del runstream y se salta el segundo objeto; si no, se salta el siguiente objeto y ejecuta el segundo. Por ejemplo,

```
DUPTYPELIST? ITE INNERCOMP ONE
```

toma una lista y la descompone en sus componentes, dejando la cuenta (de los elementos de la lista) en la pila; con cualquier otro tipo de argumento distinto de una lista, sube el entero binario #1 a la pila.

El inverso de IT es

```
?SKIP ( flag --> )
```

Si flag es TRUE, se salta el siguiente objeto del runstream; si no, lo ejecuta.

También hay un salto incondicional:

```
SKIP ( --> )
```

Se salta el siguiente objeto del runstream y continúa la ejecución más allá de él. La secuencia SKIP ; es un NOP (no operation, o sea, no hace nada).

Palabras Combinadas:

Palabra	Pila	Equivalente
#0=ITE	(# -->)	#0= ITE
#<ITE	(# -->)	#0< ITE
#=ITE	(# -->)	#= ITE
#>ITE	(# -->)	#> ITE
ANDITE	(flag flag' -->)	AND ITE
DUP#0=ITE	(# --> #)	DUP #0= ITE
EQIT	(ob1 ob2 -->)	EQ IT
EQITE	(ob ob' -->)	EQ ITE
DUP#0=IT	(# --> #)	DUP #0= IT
SysITE		(# -->)
UserITE		(# -->)

17.4 Palabras CASE

La palabra case (caso) es una combinación de ITE, COLA y SKIP. O sea, case toma una bandera de la pila; si es TRUE, case ejecuta el objeto que le sigue en el runstream mientras baja la pila de retornos al puntero del intérprete, descartando el resto del programa que sigue al objeto (como COLA). Si es FALSE, case se salta el siguiente objeto y continúa con el programa (como SKIP). Por ejemplo, el siguiente programa ejecuta objetos diferentes de acuerdo con el valor de un entero binario en la pila:

```
:: DUP #0= case ZEROCASE
   DUP ONE #= case ONECASE
   DUP TWO #= case TWOCASE
   ...
;
```

Hay varias palabras que contienen a "case" como parte de sus definiciones. El ejemplo anterior se puede escribir de modo más compacto usando OVER#=case:

```
:: ZERO OVER#=case ZEROCASE
   ONE OVER#=case ONECASE
   TWO OVER#=case TWOCASE
   ...
;
```

Lo que hacen las palabras que se listan más adelante está generalmente suficientemente claro a partir de sus nombres. Los nombres tienen (hasta) tres partes: una parte inicial, luego "case" y luego una parte final. La parte inicial indica que se hace antes de la acción "case", o sea, "xxxcase..." es equivalente a "xxx case...". Las palabras que tienen una parte final después de "case" son de dos tipos. En un tipo, la parte final indica el objeto mismo ejecutado condicionalmente, o sea, "...caseyyy" es equivalente a "...case yyy". En el otro tipo, la parte final es una palabra o palabras que se incorporan al siguiente objeto. caseDROP y casedrop son del primer y segundo tipo respectivamente. caseDROP es equivalente a case DROP; casedrop es como case con un DROP incorporado en el siguiente objeto. O sea,

```
(-?!)
```

Palabras que hacen COLA o SKIP al siguiente objeto:

```
#=casedrop     ( # # --> )
               ( # #' --> # )
   Se debería llamar OVER#=casedrop

%1=case        ( % --> )

%0=case        ( % --> flag )

ANDNOTcase     ( flag1 flag2 --> )

ANDcase        ( flag1 flag2 --> )

case2drop      ( ob1 ob2 TRUE --> )
               ( FALSE --> )

casedrop       ( ob TRUE --> )
               ( FALSE --> )

DUP#0=case     ( # --> # )

DUP#0=csedrp   ( # --> # ) # <> #0
               ( # -->   ) # = #0

EQUALNOTcase   ( ob ob' --> )

EQUALcase      ( ob ob' --> )
```

```
EQUALcasedrp  ( ob ob' ob' --> )
              ( ob ob' ob'' --> ob )

EQcase        ( ob1 ob2 --> )

NOTcase       ( flag --> )

NOTcasedrop   ( ob FALSE --> )
              ( TRUE --> )

ORcase        ( flag1 flag2 --> )

OVER#=case    ( # #' --> # )
```

Palabras case que o salen o continúan con el siguiente objeto:

```
caseDoBadKey  ( flag --> ) Sale vía DoBadKey

caseDrpBadKey ( ob TRUE --> ) Sale vía DoBadKey
              ( FALSE --> )

case2DROP     ( ob1 ob2 TRUE --> )
              ( FALSE --> )

caseDROP      ( ob TRUE --> )
              ( FALSE --> )

caseFALSE     ( TRUE --> FALSE )
              ( FALSE --> )

caseTRUE      ( TRUE --> TRUE )
              ( FALSE --> )

casedrpfls    ( ob TRUE --> FALSE )
              ( FALSE --> )

case2drpfls   ( ob1 ob2 TRUE --> FALSE )
              ( FALSE --> )

casedrptru    ( ob TRUE --> TRUE )
              ( FALSE --> )

DUP#0=csDROP  ( #0 -->  )
              ( #  --> # ) # <> 0.
NOTcaseTRUE   ( FALSE --> TRUE )
              ( TRUE --> )
```

18. Operaciones de la Pila

Las palabras listadas en este capítulo realizan operaciones de la pila sencillas o múltiples.

```
2DROP           ( ob1 ob2 --> )
2DROP00         ( ob1 ob2 --> #0 #0 )
2DROPFALSE      ( ob1 ob2 --> FALSE )
2DUP            ( ob1 ob2 --> ob1 ob2 ob1 ob2 )
2DUP5ROLL       ( ob1 ob2 ob3 --> ob2 ob3 ob2 ob3 ob1 )
2DUPSWAP        ( ob1 ob2 --> ob1 ob2 ob2 ob1 )
2OVER           ( ob1 ob2 ob3 ob4 --> ob1 ob2 ob3 ob4 ob1 ob2 )
2SWAP           ( ob1 ob2 ob3 ob4 --> ob3 ob4 ob1 ob2 )
3DROP           ( ob1 ob2 ob3 --> )
3PICK           ( ob1 ob2 ob3 --> ob1 ob2 ob3 ob1 )
3PICK3PICK      ( ob1 ob2 ob3 --> ob1 ob2 ob3 ob1 ob2 )
3PICKOVER       ( ob1 ob2 ob3 --> ob1 ob2 ob3 ob1 ob3 )
3PICKSWAP       ( ob1 ob2 ob3 --> ob1 ob2 ob1 ob3 )
3UNROLL         ( ob1 ob2 ob3 --> ob3 ob1 ob2 )
4DROP           ( ob1 ob2 ob3 ob4 --> )
4PICK           ( ob1 ob2 ob3 ob4 --> ob1 ... ob4 ob1 )
4PICKOVER       ( ob1 ob2 ob3 ob4 --> ob1 ob2 ob3 ob4 ob1 ob4 )
4PICKSWAP       ( ob1 ob2 ob3 ob4 --> ob1 ob2 ob3 ob1 ob4 )
4ROLL           ( ob1 ob2 ob3 ob4 --> ob2 ob3 ob4 ob1 )
4UNROLL         ( ob1 ob2 ob3 ob4 --> ob4 ob1 ob2 ob3 )
4UNROLL3DROP    ( ob1 ob2 ob3 ob4 --> ob4 )
4UNROLLDUP      ( ob1 ob2 ob3 ob4 --> ob4 ob1 ob2 ob3 ob3 )
4UNROLLROT      ( ob1 ob2 ob3 ob4 --> ob4 ob3 ob2 ob1 )
5DROP           ( ob1 ... ob5 --> )
5PICK           ( ob1 ... ob5 --> ob1 ... ob5 ob1 )
5ROLL           ( ob1 ... ob5 --> ob2 ... ob5 ob1 )
5ROLLDROP       ( ob1 ... ob5 --> ob2 ... ob5 )
5UNROLL         ( ob1 ... ob5 --> ob5 ob1 ... ob4 )
6DROP           ( ob1 ... ob6 --> )
6PICK           ( ob1 ... ob6 --> ob1 ... ob6 ob1 )
6ROLL           ( ob1 ... ob6 --> ob2 ... ob6 ob1 )
7DROP           ( ob1 ... ob7 --> )
7PICK           ( ob1 ... ob7 --> ob1 ... ob7 ob1 )
7ROLL           ( ob1 ... ob7 --> ob2 ... ob7 ob1 )
8PICK           ( ob1 ... ob8 --> ob1 ... ob8 ob1 )
8ROLL           ( ob1 ... ob8 --> ob2 ... ob8 ob1 )
8UNROLL         ( ob1 ... ob8 --> ob8 ob1 ... ob7 )
DEPTH           ( ob1 ... obn ... --> #n )
DROP            ( ob --> )
DROPDUP         ( ob1 ob2 --> ob1 ob1 )
DROPFALSE       ( ob --> FALSE )
DROPNDROP       ( ... # ob ) elimina ob, luego DROP # objetos
DROPONE         ( ob --> #1 )
DROPOVER        ( ob1 ob2 ob3 --> ob1 ob2 ob1 )
DROPRDROP       ( ob --> )   DROP ob y baja 1 nivel de la pila de
                             retornos.
DROPROT         ( ob1 ob2 ob3 ob4 --> ob2 ob3 ob1 )
DROPSWAP        ( ob1 ob2 ob3 --> ob2 ob1 )
DROPSWAPDROP    ( ob1 ob2 ob3 --> ob2 )
DROPTRUE        ( ob --> TRUE )
DROPZERO        ( ob --> #0 )
DUP             ( ob --> ob ob )
DUP#1+PICK      ( ... #n --> ... #n obn )
DUP3PICK        ( ob1 ob2 --> ob1 ob2 ob2 ob1 )
```

```
DUP4UNROLL      ( ob1 ob2 ob3 --> ob3 ob1 ob2 ob3 )
DUPDUP          ( ob --> ob ob ob )
DUPONE          ( ob --> ob ob #1 )
DUPPICK         ( ... #n --> ... #n obn-1 )
DUPROLL         ( ... #n --> ... #n obn-1 )
DUPROT          ( ob1 ob2 --> ob2 ob2 ob1 )
DUPTWO          ( ob --> ob ob #2 )
DUPUNROT        ( ob1 ob2 --> ob2 ob1 ob2 )
DUPZERO         ( ob --> ob ob #2 )
N+1DROP         ( ob ob1 ... obn #n --> )
NDROP           ( ob1 ... obn #n --> )
NDUP            ( ob1 ... obn #n --> ob1 ... obn ob1 ... obn )
NDUPN           ( ob #n --> ob ... ob )
ONEFALSE        ( --> #1 FALSE )
ONESWAP         ( ob --> #1 ob )
OVER            ( ob1 ob2 --> ob1 ob2 ob1 )
OVER5PICK       ( v w x y z --> v w x y z y v )
OVERDUP         ( ob1 ob2 --> ob1 ob2 ob1 ob1 )
OVERSWAP        ( ob1 ob2 --> ob2 ob1 ob1 )
OVERUNROT       ( ob1 ob2 --> ob1 ob1 ob2 )
PICK            ( obn ... #n --> ... obn )
ROLL            ( obn ... #n --> ... obn )
ROLLDROP        ( obn ... #n --> ... )
ROLLSWAP        ( obn ... ob #n --> ... obn ob )
ROT             ( ob1 ob2 ob3 --> ob2 ob3 ob1 )
ROT2DROP        ( ob1 ob2 ob3 --> ob2 )
ROT2DUP         ( ob1 ob2 ob3 --> ob2 ob3 ob1 ob3 ob1 )
ROTDROP         ( ob1 ob2 ob3 --> ob2 ob3 )
ROTDROPSWAP     ( ob1 ob2 ob3 --> ob3 ob2 )
ROTDUP          ( ob1 ob2 ob3 --> ob2 ob3 ob1 ob1 )
ROTOVER         ( ob1 ob2 ob3 --> ob2 ob3 ob1 ob3 )
ROTROT2DROP     ( ob1 ob2 ob3 --> ob3 )
ROTSWAP         ( ob1 ob2 ob3 --> ob2 ob1 ob3 )
SWAP            ( ob1 ob2 --> ob2 ob1 )
SWAP2DUP        ( ob1 ob2 --> ob2 ob1 ob2 ob1 )
SWAP3PICK       ( ob1 ob2 ob3 --> ob1 ob3 ob2 ob1 )
SWAP4PICK       ( ob1 ob2 ob3 ob4 --> ob1 ob2 ob4 ob3 ob4 )
SWAPDROP        ( ob1 ob2 --> ob2 )
SWAPDROPDUP     ( ob1 ob2 --> ob2 ob2 )
SWAPDROPSWAP    ( ob1 ob2 ob3 --> ob3 ob1 )
SWAPDROPTRUE    ( ob1 ob2 --> ob2 TRUE )
SWAPDUP         ( ob1 ob2 --> ob2 ob1 ob1 )
SWAPONE         ( ob1 ob2 --> ob2 ob1 #1 )
SWAPOVER        ( ob1 ob2 --> ob2 ob1 ob2 )
SWAPROT         ( ob1 ob2 ob3 --> ob3 ob2 ob1 )
SWAPTRUE        ( ob1 ob2 --> ob2 ob1 TRUE )
UNROLL          ( ... ob #n --> ob ... )
UNROT           ( ob1 ob2 ob3 --> ob3 ob1 ob2 )
UNROT2DROP      ( ob1 ob2 ob3 --> ob3 )
UNROTDROP       ( ob1 ob2 ob3 --> ob3 ob1 )
UNROTDUP        ( ob1 ob2 ob3 --> ob3 ob1 ob2 ob2 )
UNROTOVER       ( ob1 ob2 ob3 --> ob3 ob1 ob2 ob1 )
UNROTSWAP       ( ob1 ob2 ob3 --> ob3 ob2 ob1 )
ZEROOVER        ( ob --> ob #0 ob )
reversym        ( ob1 ... obn #n --> obn ... ob1 #n )
```

19. Operaciones de Memoria

Las palabras que se presentan en este capítulo manipulan directorios, variables y ram del sistema.

19.1 Memoria Temporal

N.T.>TEMPOB es el área de memoria de objetos temporales

La palabra de usuario NEWOB crea una nueva copia de un objeto en la memoria temporal. Hay unas cuantas variaciones internas en este tema:

```
CKREF           ( ob --> ob' )
                 Si ob está en TEMPOB, no está embebido en un objeto
                 compuesto y no está referenciado entonces no hace
                 nada. Si no, copia ob en TEMPOB y devuelve la copia.

INTEMNOTREF?    ( ob --> ob flag )
                 Si el objeto inicial no está en el área TEMPOB, no
                 está embebido en un objeto compuesto y no está
                 referenciado, devuelve ob y TRUE, si no, devuelve ob
                 y FALSE.

TOTEMPOB        ( ob --> ob' )
                 Copia ob en TEMPOB y devuelve un puntero al nuevo ob.
```

19.2 Variables y Directorios

El fundamento en RPL del sistema de las palabras de usuario STO y RCL son las palabras STO, CREATE y @:

```
CREATE ( ob id --> )
     Crea en el directorio actual una PALABRA-RAM (RAM-WORD) con ob como
     su parte objeto y la FORMA de NOMBRE (NAME FORM) del id como su parte
     nombre. Se produce un error si ob es o contiene el directorio actual
     ("Recursión de Directorio"). Se supone que ob no es un objeto código
     primitiva.

STO ( ob id --> )
    ( ob lam --> )
     En el caso lam, el identificador temporal lam se vuelve a unir al
     nuevo ob. La unión es al primer objeto identificador temporal que
     empareja (casa) con lam en el área de Entornos Temporales (buscando
     desde el primer entorno temporal (el más interno) hacia el último).
     Se devuelve un error si lam no está unido a nada. En el caso de id,
     STO intenta emparejar id con la parte nombre de una variable global.
     Si no lo consigue, STO crea en el directorio actual una variable con
     ob como su parte objeto y la "forma de nombre" a partir del
     identificador como su parte nombre. Si se empareja (se resuelve),
     entonces el ob reemplaza la parte de objeto de la variable resuelta.
     Si cualquier puntero de objeto del sistema actualizable hace
     referencia a la parte objeto de la variable resuelta, entonces la
     parte objeto se coloca en el área de objetos temporales antes de
     reemplazarla y se ajustan todos los punteros de objeto actualizables
     afectados para que se refieran a la copia de la parte de objeto en el
     área de objetos temporales. En el caso del id, STO supone que ob no
     es un objeto código primitiva.
```

```
@   ( id --> ob TRUE )
    ( id --> FALSE )
    ( lam --> ob TRUE )
    ( lam --> FALSE )
     En el caso lam, @ intenta emparejar lam a la parte objeto
     identificador temporal de una unión en el área de Entornos Temporales
     (buscando desde la primera área de entornos temporales (la más
     interna) hacia la última). Si lo consigue, entonces el objeto unido
     al lam se devuelve junto con una bandera TRUE; si no, se devuelve una
     bandera FALSE. En el caso id, @ intenta emparejar id con la parte
     nombre de una variable global, comenzando en el directorio actual y
     subiendo por los directorios padre si fuera necesario. Si no se
     consigue resolver, entonces se devuelve una bandera FALSE. Si se
     consigue resolver, se devuelve la parte objeto de la variable
     resuelta junto con una bandera TRUE.
```

Una dificultad al usar STO y @ es que no hacen ninguna distinción con los comandos incorporados; con SIN como su argumento (objeto), STO copiará alegremente el cuerpo entero de SIN en una variable. Luego @ llamaría (a la pila) ese programa indescompilable. Por esta razón, es mejor usar SAFESTO y SAFE@, que funcionan como STO y @ excepto que convierten automáticamente los cuerpos ROM en nombres XLIB (SAFESTO) y al revés de nuevo (SAFE@).

Las extensiones adicionales son:

```
?STO_HERE ( ob id --> )
          ( ob lam --> )
     Esto es la versión del sistema del STO de usuario. Es igual que
     SAFESTO, excepto que con las variables globales, a) almacena sólo en
     el directorio actual; y b) no sobrescribirá un directorio almacenado.

SAFE@_HERE ( id --> ob TRUE )
           ( id --> FALSE )
           ( lam --> ob TRUE )
           ( lam --> FALSE )
     Igual que SAFE@, excepto que con variables globales la búsqueda se
     restringe al directorio actual.
```

Otras palabras relacionadas:

```
  PURGE    ( id --> )     Elimina la variable especificada por id; no
                          hace ninguna comprobación de tipo en el
                          objeto almacenado.

  XEQRCL   ( id --> ob )  Igual que SAFE@ con las variables globales,
                          pero produce un error si la variable no
                          existe.

  XEQSTOID ( ob id --> ) Nombre alternativo de ?STO_HERE
```

19.2.1 Directorios. Un directorio (abreviado "rrp" de su nombre original "ramrompair" (parramrom) es un objeto cuyo cuerpo contiene un lista encadenada de variables globales--objetos con nombre referenciados con nombres globales. El cuerpo también contiene un número ID de biblioteca que asocia ("attaches", "asigna") un objeto biblioteca con el directorio de modo que los comandos de la biblioteca siguen a las variables del directorio en el orden de búsqueda en la compilación de nombres.

Un directorio puede estar "enraizado", o sea, almacenado en una variable global (que puede estar dentro del cuerpo de otro directorio), en cuyo caso sus nombres de variables están disponibles para la compilación. El directorio concreto en el que empieza la búsqueda para la resolución de un nombre se llama "directorio actual" o "directorio contexto"; este directorio se especifica por el contenido de una posición RAM del sistema. Un directorio no enraizado (en el tempob o en un puerto, por ejemplo), no se debería seleccionar nunca como eldirectorio contexto. Ni puede haber ninguna referencia dentro de un directorio en tempob; un directorio no es un objeto compuesto, de modoque la "recolección de basura" (G.C) no puede funcionar correctamente si existen tales referencias. Por esta razón, no se puede eliminar un directorio referenciado internamente con PURGE--usa XEQPGDIR en su lugar.

Además del contexto, otra posición de la RAM del sistema identifica el directorio "señal de stop", que es el que actúa como el punto final en la búsqueda de la resolución de un nombre así como el directorio contexto es el punto de partida. Usando la señal de stop, puedes restringir la búsqueda en la resolución de un nombre a un margen específico; sin embargo, deberías usar trampas de error para asegurar que la señal de stop se reinicializa al directorio "home" cuando sucede un error.

El directorio "home" ("sysramrompair") es el directorio por defecto tanto para el contexto como para el señal de stop. No es un directorio normal pues nunca puede estar no-enraizado (o sea, siempre está enraizado) y contiene unas estructuras adicionales que no tienen los directorios ordinarios (tales como asignaciones múltiples de bibliotecas y tablas alternativas de mensajes y "hash" de comandos).

Un directorio es un objeto de la clase datos por lo que la ejecución de un directorio simplemente lo devuelve a la pila. Sin embargo, la ejecución de un nombre global tiene la propiedad que ejecutando el nombre de un directorio enraizado (almacenado) hace de ese directorio el directorio actual en lugar de ejecutar el directorio mismo.

Hay disponibles las siguientes palabras para la manipulación de directorios:

```
CONTEXT! ( rrp --> )
     Almacena como directorio actual un puntero a un directorio enraizado

CONTEXT@ ( --> rrp )
     Llama a la pila el directorio contexto actual.

CREATEDIR       ( id --> )
     Crea un objeto directorio en el directorio actual.
```

```
DOVARS  ( --> { id1 id2 ... } )
     Devuelve la lista de los nombres de variables del directorio actual.

HOMEDIR ( --> )
     Hace de HOME el directorio actual.

PATHDIR ( --> { HOME dir dir ... } )
      Devuelve el camino actual.

UPDIR   ( --> )
     Convierte en contexto el directorio padre del actual.

XEQORDER ( { id1 id2 ... } --> )
     Ordena el directorio actual.

XEQPGDIR ( id --> )
     Elimina un directorio mientras respeta las convenciones de
     referencia/recolección de basura.
```

19.3 El Directorio Oculto

Hay un directorio sin nombre y oculto al principio del directorio "home", que contiene las definiciones de teclas del usuario y la información de las alarmas. Los programas de aplicaciones pueden usar también este directorio. Sin embargo, recuerda que el usuario no tiene modo alguno de detectar o eliminar variables de este directorio, de modo que una aplicación debería o bien eliminar tales variables antes de terminar o proporcionar un comando que le permita al usuario eliminar variables específicas del directorio oculto.

Estas palabras proporcionan capacidades de almacenamiento, llamada y eliminación en el directorio oculto:

```
PuHiddenVar  ( id --> )
     Elimina la variable oculta de nombre id.

RclHiddenVar ( id --> ob TRUE )
             ( id --> FALSE   )
     Llama (@) a una variable oculta.

StoHiddenVar    ( ob id --> )
     Almacena ob en una variable oculta.
```

19.4 Utilidades Adicionales de Memoria

```
GARBAGE  ( --> )
     Fuerza una recolección de basura.

MEM( --> # )
     Devuelve (a la pila) la cantidad de memoria libre (no se fuerza una
     recolección de basura)

OCRC ( ob --> #nibbles checksum(hxs) )
     Devuelve el tamaño del objeto en nibbles y una cadena hex con el
     checksum

getnibs ( DATOShxs DIRECCIONhxs --> DATOShxs' )
     Versión interna en RPL de PEEK

putnibs ( DATOShxs DIRECCIONhxs --> )
     Versión interna en RPL de POKE
```

20. Manejo de la Pantalla y Gráficos

La mayoría de los comandos gráficos en RPL de usuario se dirigen a la pantalla de gráficos, que es el objeto gráfico visible en el entorno del trazador (dibujo de funciones). Sin embargo, la "pantalla de texto", el grob visible en el entorno de la pila estándar tiene las mismas propiedades que la pantalla de gráficos y deben usarla los programas de las aplicaciones para mostrar gráficos cuando sea posible, para dejar la pantalla de gráficos como un recurso "propiedad" del usuario. El Escritor de Ecuaciones (EquationWriter) lo hace así, por ejemplo, como también lo hace la tarjeta HP de la Biblioteca de Ecuaciones HP82211A (HP Solve Equation Library)

20.1 Organización de la Pantalla

La ram del sistema de la HP 48 contiene tres objetos gráficos dedicados usados para mostrar información.

Puntero		Grob	Posición
HARDBUFF2	→	etiquetas de menús	(Mem Baja)
ABUFF	→	grob de texto	
GBUFF	→	grob de gráficos	(Mem Alta)

El grob de texto y el grob de gráficos se pueden agrandar y se pueden desplazar.

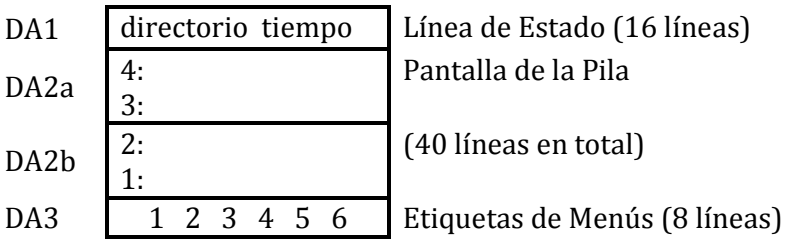
La palabra TOADISP hace visible el grob de texto; TOGDISP pone en el LCD (pantalla de cristal líquido) el grob gráfico.

Las siguientes palabras son útiles para devolver a la pila los grobs de pantalla:

```
ABUFF      ( --> grob de texto )
GBUFF      ( --> grob gráfico )
HARDBUFF   ( --> HBgrob )
           Devuelve a la pila el grob de texto o gráfico que se esté
           mostrando actualmente.
HARDBUFF2  ( --> grob de menú )
HBUFF_X_Y  ( --> HBgrob #x1 #y1 )
```

Un puntero ram llamado VDISP indica qué grob se está mostrando actualmente en la pantalla. VDISP puede apuntar al grob de texto o al grob gráfico. VDISP no se puede acceder directamente - la palabra HARDBUFF devuelve el grob de pantalla actual a la pila (ver más adelante). Recuerda que ABUFF y GBUFF solo devuelven punteros, de modo que si se llama al grob para modificarlo y devolverlo posteriormente al usuario, se debería usar TOTEMPOB para crear una copia única en la memoria temporal.

Desde el punto de vista del usuario, la pantalla de texto se organiza en tres regiones y la convención interna de numeración de estas áreas queda reflejada en muchas de las palabras de control de la pantalla (ver "Control de las Areas de la Pantalla" más adelante). Las áreas de la pantalla se numeran 1, 2 y 3. Las letras "DA" de "Display Area" (Area de Pantalla) se encuentran en los nombres de algunas palabras de control de la pantalla.



El área de la pantalla 2 está en realidad dividida en las áreas 2a y 2b, una distinción usada muy a menudo por la línea de la línea de comandos. La frontera entre 2a y 2b se puede mover, pero el tamaño total de las áreas 1, 2 y 3 es fijo.

20.2 Preparación de la Pantalla

Dos palabras establecen el control sobre la pantalla de texto. Estas palabras son RECLAIMDISP y ClrDA1IsStat.

La palabra RECLAIMDISP realiza las siguientes acciones:

- Asegura que el grob de texto es la pantalla actual
- Borra la pantalla de texto
- Si es necesario, cambia el tamaño del grob de texto a las medidas estándar (131 de ancho por 56 de alto)

RECLAIMDISP se parece mucho a la palabra de usuario CLLCD, excepto que CLLCD no cambia el tamaño del grob de texto.

La palabra ClrDA1IsStat suspende la actualización del reloj en la pantalla y es opcional. Si se va a solicitar alguna entrada al usuario usando palabras tales como WaitForKey o un bucle externo parametrizado (ver "Control del Teclado"), entonces se continuará actualizando el reloj y puede estropear la imagen en pantalla.

Se puede encontrar un ejemplo del uso de ClrDA11IsStat en la aplicación Tabla Periódica, dónde un usuario puede entrar una fórmula molecular. La palabra WaitForKey se usa para obtener las teclas pulsadas y ClrDA1IsStat evita que el reloj sobreimpresione la cuadrícula de la Tabla Periódica en la pantalla.

Si la pantalla del menú no es necesaria, la palabra TURNMENUOFF eliminará DA3 de la pantalla y aumentará el grob de texto hasta 131x64. La palabra complementaria de ésta es TURNMENUON que restaura los menús en la pantalla.

Un esqueleto simplificado para un secundario de una aplicación que puede ser invocada por el usuario final y usa la pantalla de texto podría ser más o menos así:

```
::
  ClrDA1IsStat          ( *Suspende la actualización del reloj en la
                          pantalla* )
  RECLAIMDISP           ( *Asegura y borra la pantalla alfa* )
  TURNMENUOFF           ( *Quita las teclas de menú* )

  < aplicación >

  ClrDAsOK   -\         ( *Le dice a la 48 que redibuje el LCD* )
    -o-        > Escoge una
  SetDAsTemp -/         ( *Congela todas las áreas de la pantalla* )
;
```

20.3 Control del Refresco de la Pantalla

Cuando se termina una aplicación o se vuelve al bucle externo del sistema para esperar que se entre algo por teclado, hay disponibles varias versiones internas de la palabra de usuario FREEZE para controlar la pantalla y hay una palabra que asegura que se redibujarán ciertas o todas las áreas de la pantalla:

```
SetDA1Temp      Congela el área de pantalla 1
SetDA2aTemp     Congela el área de pantalla 2a
SetDA2bTemp     Congela el área de pantalla 2b
SetDA3Temp      Congela el área de pantalla 3
SetDAsTemp      Congela todas las áreas de la pantalla
ClrDAsOK        Redibuja toda la pantalla cuando termina el programa
```

Todavía hay más variaciones en este tema - ver el capítulo "Control del Teclado" para más información.

20.4 Borrar la Pantalla

Las siguientes palabras se pueden usar para borrar toda la pantalla o una parte del HARDBUFF. Recuerda que HARDBUFF se refiere al grob mostrado actualmente, que es o el grob de texto o el grob gráfico.

```
BLANKIT         ( #filainicio #filas --> )
                 Borra #filas empezando en #filainicio
BlankDA12       ( --> )
                 Borra desde la fila 0 hasta la 56
BlankDA2        ( --> )
                 Borra desde la fila 16 hasta la 56
CLEARVDISP      ( --> )
                 Pone a cero todo el HARDBUFF
Clr16           ( --> )
                 Borra las primeras 16 filas de pixels
Clr8            ( --> )
                 Borra las primeras 8 filas de pixels
Clr8-15         ( --> )
                 Borra desde la fila 8 de pixels hasta la 15
```

20.5 Control de los Indicadores

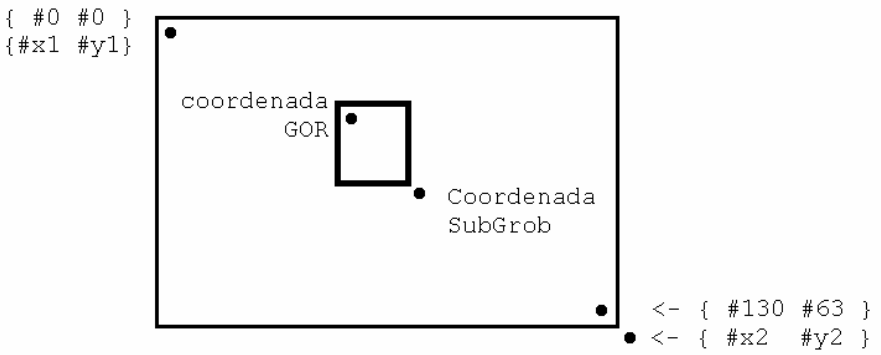
Las siguientes palabras controlan los indicadores de desplazamiento a la izquierda, desplazamiento a la derecha y alfa. Es poco probable que una aplicación tenga que controlarlos directamente y el mal uso de estas palabras puede conducir a pantallas confusas después de que la aplicación haya terminado.

```
ClrAlphaAnn     Apaga el indicador alfa
ClrLeftAnn      Apaga el indicador de desplazamiento izquierda
ClrRightAnn     Apaga el indicador de desplazamiento derecha
SetAlphaAnn     Enciende el indicador alfa
SetLeftAnn      Enciende el indicador de desplazamiento izquierda
SetRightAnn     Enciende el indicador de desplazamiento derecha
```

20.6 Coordenadas de la Pantalla

El pixel superior izquierdo de la pantalla tiene las coordenadas x=0 e y=0, que son las mismas que las coordenadas del pixel de usuario { #0 #0 }. Las coordenadas del pixel inferior derecho son x=130 e y=63.

NOTA: los subgrobs se toman desde la coordenada superior izquierda hasta el pixel debajo y a la derecha de la esquina inferior derecha. Los términos #x1 e #y1 se refieren al pixel superior izquierdo de una subárea, mientras que #x2 e #y2 se refieren al pixel debajo y a la derecha de la esquina inferior derecha.



20.6.1 Coordenadas de la Ventana

Las siguientes rutinas devuelven HARDBUFF y las coordenadas para las porciones de la pantalla en la forma adecuada para una subsiguiente llamada a SUBGROB. Los términos #x1 e #y1 se refieren a la esquina superior izquierda de la ventana en el grob mostrado actualmente. Si se ha desplazado el grob, -éstas no serán #0 #0!.

Si HARDBUFF ha sido desplazado, algunas palabras de pantalla pueden no ser apropiadas para usarse, ya que dependen de que la esquina superior izquierda de la pantalla sea #0 #0. Al LCD se le llama entonces la "ventana" y los términos #x1 e #y1 se referirán a las coordenadas del pixel de la esquina superior izquierda de la ventana. La palabra HBUFF_X_Y devuelve HARDBUFF y estas coordenadas de la ventana. La palabra WINDOWCORNER devuelve solo las coordenadas de la ventana. Las palabras DISPROW1* y DISPROW2*, mencionadas más adelante, funcionan en relación a la esquina de la ventana.

```
Rows8-15        ( --> HBgrob #x1 #y1+8 #x1+131 #y1+16 )
TOP16           ( --> HBgrob #x1 #y1 #x1+131 #y1+16 )
TOP8            ( --> HBgrob #x1 #y1 #x1+131 #y1+8 )
WINDOWCORNER    ( --> #x #y )
                Devuelve las coordenadas del pixel de la esquina superior
                izquierda de la ventana.
```

La palabra Save16 llama a TOP16 y a SUBGROB para producir un grob que consiste en las 16 primeras filas de la pantalla actual:

```
Save16          ( --> grob )
```

En la HP 48 no existen las palabras equivalentes para las primeras 8 filas ni para las filas 8 hasta la 15, pero se pueden escribir como se indica a continuación:

```
:: TOP8 SUBGROB ; ( --> grob ) ( *Salva las primeras 8 filas* )
:: TOP8-15 SUBGROB ; ( --> grob ) ( *Salva las filas 8 a 15* )
```

20.7 Mostrar Texto

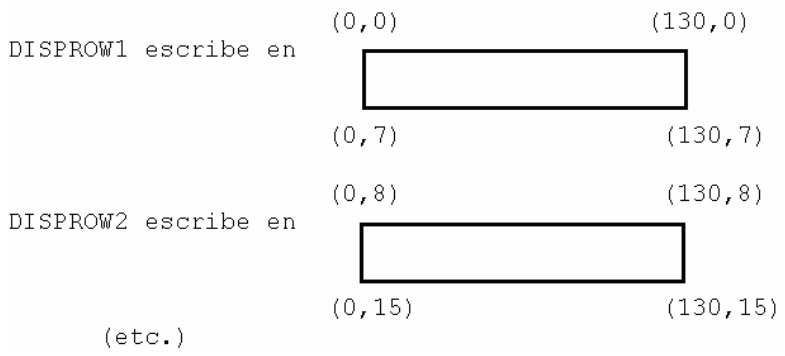
Hay tres fuentes (tipos de letras) disponibles en la HP 48, que se distinguen por el tamaño. La fuente más pequeña es de anchura variable; la mediana y la grande son de anchura fija.

Las palabras que se describen más adelante muestran texto usando las fuentes mediana y grande en áreas específicas. El uso de las fuentes pequeñas y otras posiciones opcionales para las fuentes mediana y grande se deben hacer en los gráficos, lo que se describe más tarde.

20.7.1 Areas Estándar para Mostrar Texto

Cuando la pantalla actual es el grob de texto y no ha sido desplazada, se pueden usar las siguientes palabras para mostrar texto en la fuente mediana (5x7). Las cadenas largas se truncan a 22 caracteres con puntos suspensivos al final (...) y las cadenas con menos de 22 caracteres se rellenan con espacios (por la derecha)

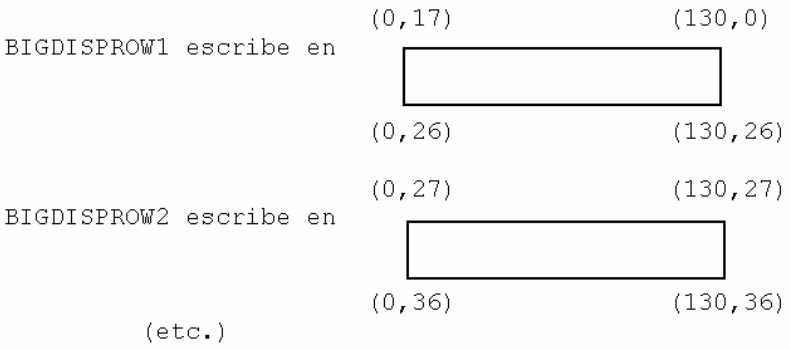
```
DISPROW1        ( $ --> )
DISPROW2        ( $ --> )
DISPROW3        ( $ --> )
DISPROW4        ( $ --> )
DISPROW5        ( $ --> )
DISPROW6        ( $ --> )
DISPROW7        ( $ --> )
DISPN           ( $ #fila --> )
DISP5x7         ( $ #inicio #max )
```



La palabra DISP5x7 se puede usar para mostrar una cadena que se extiende por más de una línea de la pantalla. La cadena debe tener embebidos "retornos de carro" (chr: #13d) para indicar donde se ha de pasar a la siguiente línea de la pantalla. Si un segmento de línea es mayor de 22 caracteres, se truncará y se mostrará con puntos suspensivos al final (...). La cadena se muestra empezando en la fila #inicio con #max filas.

Las siguientes palabras se pueden usar para mostrar texto en la fuente grande (5x9). Las cadenas largas se truncan a 22 caracteres con puntos suspensivos (...) al final y las cadenas con menos de 22 caracteres se rellenan con espacios en blanco (por la derecha).

```
BIGDISPROW1     ( $ --> )
BIGDISPROW2     ( $ --> )
BIGDISPROW3     ( $ --> )
BIGDISPROW4     ( $ --> )
BIGDISPN        ( $ #fila --> )
```



20.7.2 Mensajes Temporales

A veces es conveniente mostrar una advertencia y luego devolver la pantalla a su estado previo. Hay varias técnicas y herramientas disponibles para esto. El modo más fácil de hacerlo es con la palabra FlashWarning. El código de FlashWarning se parece a este:

```
FlashWarning            ( $ --> )
::
  ERRBEEP               ( *Produce un pitido de error* )
  Save16                ( *Salva las 16 filas de pixels superiores* )
  SWAP DISPSTATUS2      ( *Muestra la advertencia* )
  VERYVERYSLOW          ( *Pausa de unos 3 segundos* )
  Restore16             ( *Restaura las 16 filas superiores* )
;
```

Se pueden hacer variaciones alrededor de FlashWarning usando palabras como TOP16 o una versión de la sugerida arriba que salva menos filas. El ejemplo que sigue salva las 8 filas superiores y muestra un mensaje de una línea durante unos 0.6 segundos:

```
::
  TOP8 SUBGROB          ( *Salva las 8 filas superiores* )
  SWAP DISPROW1*        ( *Muestra el mensaje* )
  VERYSLOW VERYSLOW     ( *Pausa corta* )
  Restore8              ( *Restaura las 8 filas superiores* )
;
```

NOTA: Es importante usar DISPROW1* y DISPROW2* en vez de DISPROW1 y DISPROW2 si existe la posibilidad que HARDBUFF haya sido desplazado. No existen las palabras correspondientes para otras líneas de la pantalla.

20.8 Objetos Gráficos

La siguiente sección presenta herramientas para crear, manipular y mostrar objetos gráficos.

20.8.1 Advertencias

He aquí dos advertencias:

1. El término "operación tipo-explosiva" se refiere a una operación que se lleva a cabo directamente sobre un objeto sin hacer antes una copia. La convención de los nombres para las palabras que realizan este tipo de operación hace que tengan a menudo un signo de admiración para indicar una operación "explosiva", como por ejemplo GROB! o GROB!ZERO.

   Debes recordar dos cosas cuando uses operaciones "explosivas":

   - Ya que se modifica el propio objeto, cualquier puntero en la pila que se refiera a ese objeto apuntará ahora a un objeto modificado. Se puede usar la palabra CKREF para asegurar que un objeto es único.

   - Estas operaciones no comprueban errores, por lo que los parámetros inadecuados o fuera de margen pueden corromper la memoria sin posibilidad de recuperación.

2. En la práctica, es mejor usar la palabra XYGROBDISP para colocar un grob en el grob de pantalla. La palabra XYGROBDISP es de naturaleza conservadora - si el gráfico a colocar en HARDBUFF sobrepasa los límites de HARDBUFF, se aumenta el grob HARDBUFF para acomodar al nuevo grob.

20.8.2 Herramientas Gráficas

Las siguientes palabras crean o modifican objetos gráficos:

```
$>BIGGROB       ( $ --> grob ) ( fuente 5x9 )
$>GROB          ( $ --> grob ) ( fuente 5x7 )
$>grob          ( $ --> grob ) ( fuente 3x7 )
DOLCD>          ( --> 64x131grob )
GROB!           ( grob1 grob2 #col #fila --> )
                 Almacena grob1 en grob2. -Es una palabra
                 tipo-explosiva sin chequeo de errores!
GROB!ZERO       ( grob #x1 #y1 #x2 #y2 --> grob' )
                 Pone a cero una sección rectangular del grob. NOTA:
                 operación tipo-explosiva
GROB!ZERODRP    ( grob #x1 #y1 #x2 #y2 --> )
                 Pone a cero una sección rectangular de un grob. NOTA:
                 -Operación tipo-explosiva!
GROB>GDISP      ( grob --> )
                 Almacena el grob en el grob gráfico.
HARDBUFF        ( --> HBgrob (el grob de pantalla actual) )
HEIGHTENGROB    ( grob #filas --> )
                 Añade #filas al grob, a menos que el grob sea nulo.
                 NOTA: -Se supone es el grob de texto o el gráfico!
INVGROB         ( grob --> grob' )
                 Invierte los bits de datos del grob - tipo-explosiva
LINEOFF         ( #x1 #y1 #x2 #y2 --> )
                 Borra los pixels en una línea del grob de texto.
                 Nota: #x2 debe ser > #x1 (usar ORDERXY#)
LINEOFF3        ( #x1 #y1 #x2 #y2 --> )
                 Borra los pixels en una línea del grob gráfico.
                 Nota: #x2 debe ser > #x1 (usar ORDERXY#)
LINEON          ( #x1 #y1 #x2 #y2 --> )
                 Activa los pixels en una línea del grob de texto.
                 Nota: #x2 debe ser > #x1 (usar ORDERXY#)
LINEON3         ( #x1 #y1 #x2 #y2 --> )
                 Activa los pixels en una línea del grob gráfico.
                 Nota: #x2 deber ser > #x1 (usar ORDERXY#)
MAKEGROB        ( #alto #ancho --> grob )
ORDERXY#        ( #x1 #y1 #x2 #y2 --> #x1 #y1 #x2 #y2 )
                 Ordena dos puntos para dibujar una línea
PIXOFF          ( #x #y --> )
                 Borra un pixel en el grob de texto
PIXOFF3         ( #x #y --> )
                 Borra un pixel en el grob gráfico
PIXON           ( #x #y --> )
                 Activa un pixel en el grob de texto
PIXON?          ( #x #y --> flag )
                 Devuelve TRUE si el pixel del grob de texto está
                 activado.
PIXON?3         ( #x #y --> flag )
                 Devuelve TRUE si el pixel del grob gráfico está
                 activado
PIXON3          ( #x #y --> )
                 Activa un pixel en el grob gráfico
SUBGROB         ( grob #x1 #y1 #x2 #y2 --> subgrob )
Symb>HBuff      ( symb --> )
                 Muestra symb en el HARDBUFF en el formato del
                 Escritor-de-Ecuaciones. Puede que aumente HARDBUFF,
                 así que, ejecutar RECLAIMDISP después.
```

```
TOGLINE       ( #x1 #y1 #x2 #y2 --> )
               Cambia el estado (encendido/apagado) de los pixels de
               una línea del grob de texto.
TOGLINE3      ( #x1 #y1 #x2 #y2 --> )
               Cambia el estado (encendido/apagado) de los pixels de
               una línea del grob gráfico.
```

NOTA: - #x2 debe ser mayor que #x1 para dibujar la línea!

20.8.3 Dimensiones de los Grobs

Las siguientes palabras devuelven o verifican información sobre el tamaño:

```
CKGROBFITS      ( grob1 grob2 #n #m --> grob1 grob2' #n #m )
                 Trunca grob2 si no cabe en grob1
DUPGROBDIM      ( grob --> grob #alto #ancho )
GBUFFGROBDIM    ( --> #alto #ancho )
                 Devuelve las dimensiones del grob gráfico
GROBDIM         ( grob --> #alto #ancho )
GROBDIMw        ( grob --> #ancho )
```

20.8.4 Grobs Incorporados

Las siguientes palabras se refieren a los grobs incorporados:

```
BigCursor       Cursor 5x9 (recuadro hueco)
CROSSGROB       Símbolo "+" 5x5
CURSOR1         Cursor de Inserción 5x9 (flecha)
CURSOR2         Cursor Sobreescribir 5x9 (recuadro sólido)
MARKGROB        Símbolo "X" 5x5
MediumCursor    Cursor 5x7 (recuadro hueco)
SmallCursor     Cursor 3x5 (recuadro hueco)
```

20.8.5 Utilidades para Mostrar Menús

Las etiquetas de menú son grobs de 8 filas de alto y 21 pixels de ancho. Las columnas de las etiquetas de menú en HARDBUFF2 son:

```
           Decimal
           -------
ZERO          0   Tecla de menú 1
TWENTYTWO    22   Tecla de menú 2
# 0002C      44   Tecla de menú 3
# 00042      66   Tecla de menú 4
# 00058      88   Tecla de menú 5
# 0006E     110   Tecla de menú 6
```

La rutina DispMenu.1 vuelve a mostrar el menú actual; la rutina DispMenu vuelve a mostrar el menú actual y también llama a SetDA3Valid para "congelar" la línea de menú de la pantalla.

Las siguientes palabras convierten objetos a etiquetas de menú y muestran las etiquetas en el número de columna dado:

```
Grob>Menú       ( #col 8x21grob --> )
                  Muestra un grob de 8x21 (-solo!)
Id>Menú         ( #col Id --> )
                 Llama al Id y muestra la etiqueta estándar o la
                 etiqueta de directorio, dependiendo del contenido de
                 Id.
Seco>Menú       ( #col seco --> )
                 Evalúa el secundario y usa el resultado para hacer y
                 mostrar la etiqueta de menú apropiada.
Str>Menú        ( #col $ --> )
                 Hace y muestra una etiqueta de menú estándar
```

Las siguientes palabras convierten cadenas de texto en las diferentes clases de grobs de teclas de menús disponibles:

```
MakeBoxLabel    ( $ --> grob )  Recuadro con una bala (cuadradito) en
                                su interior.
MakeDirLabel    ( $ --> grob )  Recuadro con una barra de directorio
MakeInvLabel    ( $ --> grob )  Etiqueta blanca (Solucionador)
MakeStdLabel    ( $ --> grob )  Etiqueta negra (estándar)
```

20.9 Desplazar la Pantalla

Están disponibles las siguientes palabras para desplazar la pantalla:

```
SCROLLDOWN      ( *Desplaza un pixel hacia abajo con repetición* )
SCROLLLEFT      ( *Desplaza un pixel a la izquierda con repetición* )
SCROLLRIGHT     ( *Desplaza un pixel a la derecha con repetición* )
SCROLLUP        ( *Desplaza un pixel hacia arriba con repetición* )

JUMPBOT         ( *Mueve la ventana al extremo inferior del grob* )
JUMPLEFT        ( *Mueve la ventana al extremo izquierdo del grob* )
JUMPRIGHT       ( *Mueve la ventana al extremo derecho del grob* )
JUMPTOP         ( *Mueve la ventana al extremo superior del grob* )
```

Las palabras SCROLL* comprueban si se mantienen pulsadas sus teclas cursor (flecha) correspondientes y repiten su acción hasta que se alcanza el extremo del grob o se suelta la tecla.

El siguiente ejemplo ilustra una serie de operaciones con gráficos y el uso del bucle externo parametrizado que proporciona desplazamiento para el usuario.

N.T.> POL = Bucle Externo Parametrizado.

```
*------------------------------------------------------
*
* Incluye el fichero de cabecera KEYDEFS.H que define palabras como
* kcUpArrow (códigodeteclaFlechaArriba) para números de teclas físicas
*
INCLUDE KEYDEFS.H
*
* Incluye los ocho caracteres necesarios para la carga binaria
*
ASSEMBLE
        NIBASC  /HPHP48-D/
RPL
*
* Empieza el secundario
*
::
  RECLAIMDISP           ( *Reclama la pantalla alfa* )
  ClrDA1IsStat          ( *Desactiva temporalmente el reloj* )
*                       ( *Pruébalo eliminando ClrDA1IsStat* )
  ZEROZERO              ( #0 #0 )
  150 150 MAKEGROB      ( #0 #0 150x150grob )
  XYGROBDISP            (  )
  TURNMENUOFF           ( *Apaga la línea de menús* )
*
* Dibuja líneas diagonales.  Recuerda que LINEON precisa que #x2>#x1!
*
  ZEROZERO              ( #x1 #y1 )
  149 149               ( #x1 #y1 #x2 #y2 )
  LINEON                ( *Dibuja la línea* )
  ZERO 149              ( #x1 #y1 )
  149 ZERO              ( #x1 #y1 #x2 #y2 )
  LINEON                ( *Dibuja la línea* )
*
* Pone el texto
*
  HARDBUFF
  75 50 "SCROLLING"     ( HBgrob 75 150 "SCROLLING" )
  150 CENTER$3x5        ( HBgrob )
  75 100 "EXAMPLE"      ( HBgrob 75 100 "EXAMPLE" )
  150 CENTER$3x5        ( HBgrob )
  DROPFALSE             ( FALSE )
  { LAM Exit } BIND     ( *Une la bandera de salida del POL* )
  ' NOP                 ( *No hace nada en pantalla*  )
  ' ::                  ( *Controlador de las teclas físicas*   )
    kpNoShift #=casedrop
      ::
         kcUpArrow    ?CaseKeyDef
                          :: TakeOver SCROLLUP ;
```

```
            kcLeftArrow  ?CaseKeyDef
                             :: TakeOver SCROLLLEFT ;
            kcDownArrow  ?CaseKeyDef
                             :: TakeOver SCROLLDOWN ;
            kcRightArrow ?CaseKeyDef
                             :: TakeOver SCROLLRIGHT ;
            kcOn         ?CaseKeyDef
                             :: TakeOver
                                TRUE ' LAM Exit STO ;
            kcRightShift   #=casedrpfls
            DROP 'DoBadKeyT
          ;
        kpRightShift #=casedrop
          ::
            kcUpArrow    ?CaseKeyDef
                             :: TakeOver JUMPTOP ;
            kcLeftArrow  ?CaseKeyDef
                             :: TakeOver JUMPLEFT ;
            kcDownArrow  ?CaseKeyDef
                             :: TakeOver JUMPBOT ;
            kcRightArrow ?CaseKeyDef
                             :: TakeOver JUMPRIGHT ;
            kcRightShift #=casedrpfls
            DROP 'DoBadKeyT
          ;
        2DROP 'DoBadKeyT
      ;
      TrueTrue              ( *Banderas de control de Teclas* )
      NULL{}                ( *Ninguna Tecla de menú ("blanda") aquí*  )
      ONEFALSE              ( *primera fila, no suspender* )
      ' LAM Exit            ( *Condición de salida de la Aplicación* )
      ' ERRJMP              ( *Controlador de errores* )
      ParOuterLoop          ( *Ejecuta el ParOuterLoop (el POL)* )
      TURNMENUON            ( *Restaura la fila de menús* )
      RECLAIMDISP           ( *Borra y devuelve la pantalla a su tamaño* )
;
```

El código de arriba, si se almacena en un fichero llamado SCROLL.S se puede compilar así:

```
RPLCOMP SCROLL.S
SASM SCROLL.A
SLOAD -H SCROLL.M
```

Este ejemplo también supone que el fichero KEYDEFS.H está en el mismo directorio o que el fichero fuente se ha modificado para reflejar la ubicación de KEYDEFS.H. El fichero de control de la carga SCROLL.M se parece a esto:

```
OU SCROLL
LL SCROLL.LR
SU XR
SE ENTRIES.O
RE SCROLL.O
```

El fichero final, SCROLL, se puede cargar en modo binario a la HP 48 para probarlo.

Cuando se está ejecutando SCROLL, las teclas de flecha (cursor) desplazan la pantalla y las teclas de flecha desplazadas a la derecha mueven la ventana al extremo correspondiente. La tecla [ATTN] termina el programa.

Para más detalles acerca del ParOuterLoop, ver el capítulo "Control del Teclado".

21. Control del Teclado

Un programa que precisa entradas del usuario por teclado puede escoger entre tres técnicas básicas disponibles en el RPL interno, listadas en orden de complejidad creciente:

1. Esperar una pulsación de tecla individual, luego decidir que hacer con ella.
2. Llamar a la forma interna de INPUT.
3. Establecer un Bucle Externo Parametrizado (POL) para controlar un entorno de aplicación completo.

Las siguientes secciones discuten el esquema de numeración interno de las teclas y cada una de las tres estrategias anteriores de procesamiento de las teclas.

21.1 Ubicación de las Teclas

La palabra de usuario WAIT devuelve un número real que es está codificado de la forma rc.p dónde:

```
r = La fila de la tecla
c = La columna de la tecla
p = El plano de desplazamiento
```

p	Planos Principales	p	Planos Alfa
0 or 1	Sin desplazar	4	Alpha
2	Despla-izquierda	5	Alfa despla-izquierda
3	Despla-derecha	6	Alfa despla-derecha

Internamente, las posiciones de las teclas se representan con dos enteros binarios: #KeyCode (#CódigoTecla), que define una tecla física y #Plane (#Plano), que define el plano de desplazamiento.

El fichero KEYDEFS.H, suministrado con el compilador RPL, define los siguientes términos para los planos de las teclas:

```
DEFINE  kpNoShift        ONE
DEFINE  kpLeftShift      TWO
DEFINE  kpRightShift     THREE
DEFINE  kpANoShift       FOUR
DEFINE  kpALeftShift     FIVE
DEFINE  kpARightShft     SIX
```

N.T.> Dónde: kp = Plano de la Tecla
A = Alfa activado
LeftShift = Desplazado a la izquierda
RightShift = Desplazado a la derecha
NoShift = Sin desplazar

Las teclas se numeran internamente de 1 a 49 empezando en la esquina superior izquierda del teclado. Las definiciones principales de las teclas también se dan en KEYDEFS.H. He aquí algunas de ellas:

```
DEFINE  kcMenuKey1       ONE
DEFINE  kcMenuKey2       TWO
DEFINE  kcMenuKey3       THREE
DEFINE  kcMenuKey4       FOUR
DEFINE  kcMenuKey5       FIVE
DEFINE  kcMenuKey6       SIX
DEFINE  kcMathMenu       SEVEN
DEFINE  kcPrgmMenu       EIGHT
DEFINE  kcCustomMenu     NINE
     ...
DEFINE  KcPlus           FORTYNINE
```

Se recomienda el uso de estas definiciones en el código fuente a fin de hacerlo más legible.

La traducción entre la numeración interna de las teclas y la numeración rc.p se puede llevar a cabo con dos palabras:

```
Ck&DecKeyLoc    ( %rc.p --> #CódigoTecla #Plano )
CodePl>%rc.p    ( #CódigoTecla #Plano --> %rc.p )
```

21.2 Esperar una Tecla

Si una aplicación necesita esperar a que se pulse una sola tecla, como por ejemplo una decisión del tipo si-no-attn, lo mejor es usar la palabra WaitForKey, que devuelve la tecla pulsada en formato completo (tecla + plano). WaitForKey también mantiene a la HP48 en un estado de bajo consumo hasta que se pulsa una tecla y controla los indicadores alfa y desplazamiento así como el procesado de alarmas.

Están disponibles las siguientes palabras:

```
CHECKKEY        ( --> #CódigoTecla TRUE )
                ( --> FALSE )
                 Devuelve, pero no la saca, la siguiente tecla en el
                 búfer.
FLUSHKEYS       ( --> )
                 Vacía el búfer de teclado
GETTOUCH        ( --> #CódigoTecla TRUE )
                ( --> FALSE )
                 Devuelve y saca la siguiente tecla del búfer.
KEYINBUFFER?    ( --> FLAG )
                 Devuelve TRUE si hay una tecla en el búfer de
                 teclado, si no, devuelve FALSE.
ATTN?           ( --> flag )
                 Devuelve TRUE si se ha pulsado [ATTN]
ATTNFLGCLR      ( --> )
                 Borra la bandera de la tecla attn (no saca la tecla
                 attn del búfer)
WaitForKey      ( --> #CódigoTecla #Plano )
                 Devuelve la siguiente tecla que se pulse en formato
                 completo (tecla + plano)
```

21.3 InputLine

La palabra InputLine es el núcleo de la palabra de usuario INPUT así como también del apuntador para solicitar nombres de ecuaciones (NEW). InputLine hace lo siguiente:

- Muestra el apuntador en el área de pantalla 2a,
- Activa los modos de entrada del teclado,
- Inicializa la línea de edición,
- Acepta la entrada del usuario hasta que se pulsa [ENTER] explícita o implícitamente,
- Analiza, evalúa o solamente devuelve la entrada del usuario en la línea de edición,
- Devuelve TRUE si se salió con Enter o FALSE si se abortó con Attn.

A la entrada la pila debe contener lo siguiente:

```
$Prompt     El apuntador que se mostrará durante la entrada del usuario
$EditLine   El contenido inicial de la línea de edición
CursorPos   La posición inicial del cursor en la línea de edición,
            especificada como un número carácter entero binario o con una
            lista de dos elementos que son los números enteros binarios
            fila y columna. En todos los números, #0 indica el final de la
            línea de edición, fila o columna.
#Ins/Rep    El modo inserción/sobreescritura inicial:
               #0    el modo ins/sob actual
               #1    modo inserción
               #2    modo sobreescritura
#Entry      El modo de entrada inicial:
               #0    modo de entrada actual más modo programa
               #1    entrada programa/inmediata
               #2    entrada programa/algebraica
#AlphaLock  El modo Alfa inicial:
               #0    modo alfa actual
               #1    bloqueo alfa activado
               #2    bloqueo alfa desactivado
ILMenu      El menú InputLine inicial en el formato especificado por el
            "ParOuterLoop"
#ILMenuRow  El número de fila inicial del menú InputLine en le formato
            especificado por el "ParOuterLoop"
AttnAbort?  Una bandera que especifica si pulsando Attn mientras existe
            una línea de edición no nula, abortaría el "InputLine" (TRUE)
            o simplemente borraría la línea de edición (FALSE)
#Parse      Como procesar la línea de edición resultante:
               #0    Devolver la línea de edición como una cadena de texto
               #1    Devolver la línea de edición como una cadena de texto
                     Y como un objeto analizado
               #2    Analizar y evaluar la línea de edición.
```

InputLine devuelve diferentes resultados, dependiendo del valor inicial de #Parse:

```
#Parse  Pila               Descripción
------  -----------------  ------------------------------
  #0    $EditLine TRUE     Línea de edición
  #1    $EditLine ob TRUE  Línea de edición y línea de edición analizada
  #2    Ob1 ... Obn TRUE   Objeto u objetos resultantes
        FALSE              Pulsada Attn para abortar la edición
```

21.3.1 Ejemplo de InputLine

La llamada ejemplo a InputLine mostrada a continuación le pide al usuario un nombre de variable. Si el usuario entra un nombre válido, se devuelven el nombre y TRUE, si no, se devuelve FALSE.

```
( --> Ob TRUE | FALSE )
::
  "Enter name:"  ( *Cadena del apuntador* )
  NULL$          ( *Ningún nombre por defecto* )
  ONEONE         ( *Posición inicial de la línea de edición y del
                 cursor* )
  ONEONE         ( *Modo Inserción y entrada programa/inmediata* )
  NULL{}         ( *Sin menú de edición* )
  ONE            ( *Fila del menú* )
  FALSE          ( *Attn borra la línea de edición* )
  ONE            ( *Devuelve la línea de edición y el ob analizado* )
  InputLine      ( ($editline ob TRUE) | (FALSE) )
  NOTcaseFALSE   ( *Sale si se pulsa Attn* )
  SWAP NULL$?    ( *Sale si la línea de edición está en blanco* )
  casedrop FALSE
  DUPTYPEIDNT?   ( *Comprueba si el ob es un id* )
  caseTRUE       ( *Sí, sale y TRUE* )
  DROPFALSE      ( *No, elimina el ob y FALSE* )
;
```

21.4 El Bucle Externo Parametrizado

En esta sección, el término "bucle externo parametrizado" se usa para referirse al uso de la palabra RPL "ParOuterLoop" o al uso combinado de las utilidades fundamentales que lo componen (descritas más adelante), todo lo cual se puede ver como palabras que toman el control del teclado y de la pantalla hasta que se cumple una condición especificada.

El bucle externo parametrizado, "ParOuterLoop", toma nueve argumentos en este orden:

AppDisplay	El objeto de actualización de la pantalla que se evalúa antes de la evaluación de cada tecla. "AppDisplay" debería controlar la actualización de la pantalla no controlada por las teclas mismas y también debería realizar un control especial de errores.
AppKeys	Las asignaciones de las teclas físicas, un objeto secundario con el formato que se describe más adelante.
NonAppKeyOK?	Una bandera que especifica si las teclas físicas no asignadas por la aplicación deben realizar sus acciones por defecto o ser canceladas.
DoStdKeys?	Una bandera que se usa conjuntamente con "NonAppKeyOK? que especifica si las teclas que no usa la aplicación usan las definiciones estándar de las teclas en lugar del procesado de teclas por defecto.
AppMenu	La especificación de las teclas de menú, un secundario o una lista con el formato especificado en el documento de asignaciones de teclas de menú, o FALSE
#AppMenuRow	El número de fila del menú inicial de la aplicación. En la mayoría de las aplicaciones, debe ser el entero binario uno.
SuspendOK?	Una bandera que especifica si cualquier comando de usuario que crearía un entorno suspendido y restauraría el bucle externo del sistema debería generar en su lugar un error o no.
ExitCond	Un objeto que se evalúa a TRUE cuando se va a salir del bucle externo o si no, FALSE. "ExitCond" se evalúa antes de cada actualización de la pantalla y de cada evaluación de tecla de la aplicación.
AppError	El objeto controlador de errores que se evalúa si se produce un error durante la parte del bucle externo parametrizado de evaluación de una tecla .

El bucle externo parametrizado mismo no devuelve ningún resultado. Sin embargo, cualquiera de las teclas usadas por la aplicación puede devolver resultados a la pila de datos o de cualquier manera que se quiera.

21.4.1 Utilidades del Bucle Externo Parametrizado

La palabra del bucle externo parametrizado "ParOuterLoop" consiste enteramente en llamadas (con el adecuado control de errores) a sus cuatro palabras de utilidades RPL, que son, en orden:

```
POLSaveUI   Salva el actual interface de usuario (UI) en un entorno
            temporal. No toma argumentos y no devuelve ningún
            resultado.

POLSetUI    Dispone (configura) el interface de usuario actual de
            acuerdo a los mismos parámetros que requiere el
            "ParOuterLoop". No devuelve ningún resultado.

POLKeyUI    Muestra, lee y evalúa teclas, controla errores y sale de
            acuerdo al interface de usuario especificado por
            "POLStetUI". No toma ningún argumento ni retorna ningún
            resultado.

POLRestoreUI  Restaura el interface de usuario salvado por
              "POLSaveUI" y abandona el entorno temporal. No toma
              ningún argumento ni devuelve ningún resultado.
```

(Además de las cuatro utilidades de arriba, se usa la utilidad "POLResUI&Err" para proteger el interface de usuario salvado en el caso de un error que no se controla dentro del bucle externo parametrizado. Véase "Operación del Bucle Externo Parametrizado" y "Control de Errores con las Utilidades", más adelante)

Estas utilidades las pueden usar las aplicaciones que requieren un mayor control sobre el interface de usuario. Por ejemplo:

- Para una ejecución óptima, una aplicación puede crear un entorno temporal con variables temporales con nombres nulos después de llamar a "POLSaveUI", luego acceder a las variables con nombres nulos "dentro" de "POLKeyUI", ya que solo "POLSaveUI" crea un entorno temporal del bucle externo parametrizado y solo "POLRestoreUI" accede al mismo entorno.

- Para evitar gastos innecesarios y que consumen tiempo, una aplicación que usa múltiples bucles externos parametrizados consecutivos (no anidados) puede llamar a "POLSaveUI" al principio de la aplicación, luego llamar a "POLSetUI" y a "POLKeyUI" múltiples veces a lo largo de la aplicación y finalmente llamar a "POLRestoreUI" al final de la aplicación.

21.4.2 Examen del Bucle Externo Parametrizado

El bucle externo parametrizado opera como se esboza a continuación.

```
("POLSaveUI")
Salva el interface del sistema o del usuario de la aplicación
actual

Si se produce un error

  ("POLSetUI")
   Configura el interface de usuario de la nueva aplicación

  ("POLKeyUI")
   Mientras "ExitCond" se evalúe a FALSE
      {
         Evalúa "AppDisplay"
         Si se produce un error
            Lee y evalúa una tecla
         Entonces
            Evalúa "AppError"
      }

Entonces

   Restaura el interface de usuario y ERRJMP

("POLRestoreUI")
Restaura el interface de usuario salvado
```

El bucle externo parametrizado crea un entorno temporal cuando salva el interface de usuario actual y abandona este entorno cuando restaura un interface de usuario salvado. Esto significa que las palabras que operan en el entorno temporal más reciente, tales como "1GETLAM", NO se deberían usar "dentro" del bucle externo parametrizado (o sea, en una definición de tecla o en el objeto de actualización de la pantalla de la aplicación) A MENOS QUE el entorno temporal deseado se cree DESPUES DE llamar a "POLSaveUI" y se abandone antes de llamar a "POLRestoreUI". En los entornos temporales creados antes de llamar al bucle externo parametrizado, las aplicaciones deberían crear y operar con variables temporales con nombre.

21.4.3 Controlar Errores con las Utilidades

Para asegurar que puede restaurar adecuadamente un interface de usuario si se produce un error dentro de una aplicación, el bucle externo parametrizado protege el interface de usuario salvado estableciendo una trampa de error inmediatamente después de su llamada a "POLSaveUI", como se muestra a continuación:

```
::
  POLSaveUI             ( salva el interface de usuario actual )
  ERRSET                ( prepara la restauración del interface de
                          usuario salvado en caso de error )
    ::
      POLSetUI          ( Configura el interface de usuario de la
                          aplicación )
      POLKeyUI          ( muestra, lee y evalúa )
    ;
  ERRTRAP               ( si error, entonces restaura el interface de
                          usuario salvado y da error )
    POLResUI&Err
  POLRestoreUI          ( restaura el interface de usuario salvado )
;
```

El propósito de la utilidad soportada "POLResUI&Err" es restaurar el interface de usuario salvado por "POLSaveUI" y luego dar error.

Las aplicaciones que usan las utilidades del bucle externo parametrizado en vez del "ParOuterLoop" TIENEN que incluir este mismo nivel de protección del interface de usuario salvado en el control de errores .

21.4.4 La Pantalla

No hay ninguna pantalla por defecto en el bucle externo parametrizado; la aplicación es la responsable de establecer la pantalla inicial y de actualizarla.

Hay dos maneras en que una aplicación puede actualizar la pantalla: con el parámetro del bucle externo "AppDisplay" o con las asignaciones de teclas. Por ejemplo, si el usuario pulsa la tecla flecha-derecha para mover un resalte de una columna de matriz a otra, la asignación de la tecla flecha-derecha puede o bien pasar información a "AppDisplay" (a menudo de modo implícito) para manejar el cambio o el objeto de asignación de la tecla puede cambiar él mismo la pantalla. Ambos métodos tienen sus ventajas en diferentes circunstancias.

21.4.5 Control de Errores

El parámetro de control de errores del bucle externo "AppError" es responsable de procesar cualquier error generado durante la evaluación de teclas dentro del bucle externo parametrizado. Si ocurre un error, se evalúa "AppError". "AppError" debería determinar el error específico y actuar en consecuencia. Si una aplicación no puede manejar algunos errores, entonces "AppError" se debe especificar como "ERRJMP".

21.4.6 Asignaciones de Teclas Físicas

Cualquier tecla de la HP 48, en cualquiera de los seis planos (sin desplazamiento, desplazada a la izquierda, desplazada a la derecha, alfa sin desplazar, alfa desplazada a la izquierda y alfa desplazada a la derecha) puede asignarse durante la ejecución del bucle externo parametrizado. El parámetro del bucle externo "AppKeys" especifica las teclas a asignar y sus nuevas asignaciones.

Si una tecla no está asignada por una aplicación y el parámetro del bucle externo "NonAppKeyOK?" es TRUE, entonces se produce el proceso estándar o por defecto, de acuerdo con el parámetro del bucle externo "DoStdKeys?". Por ejemplo, si el modo de teclas de usuario está activado y la tecla tiene una asignación de usuario, entonces se procesa la tecla de usuario si "DoStdKeys?" es FALSE o se procesa la tecla estándar si "DoStdKeys?" es TRUE. Si "NonAppKeyOK?" es FALSE, entonces todas las teclas que no son de la aplicación emiten un pitido de advertencia de tecla cancelada y no hacen nada más.

En general, NonAppKeyOK? debe ser FALSE para mantener un control total.

Las asignaciones de teclas de la aplicación se especifican con el objeto secundario "AppKeys" que se pasa al bucle externo parametrizado. El procedimiento debe tomar como sus argumentos un código de tecla y una especificación del plano y debe devolver la definición deseada de la tecla y TRUE si la aplicación define la tecla o FALSE si la aplicación no lo hace. Específicamente, el diagrama de la pila del procedimiento de asignación de teclas debe parecerse a esto:

```
( #CódigoTecla #Plano --> DefTecla TRUE )
( #CódigoTecla #Plano --> FALSE )
```

La definición de tecla resultante "DefTecla" será procesada por el controlador principal de teclas "DoKeyOb".

Las asignaciones de teclas de la aplicación especificadas como procedimientos tienen generalmente una lógica de la forma

```
Si #Plano NO está Desplazado  (o primer plano que nos interesa)
Entonces
  Procesa #CódigoTecla en el plano no desplazado
Si no
   Si #Plano está  DesplazadoIzquierda (o siguiente plano que nos
                                        interesa)
   Entonces
      Procesa #CódigoTecla en el plano desplazadoIzquierda
      ...
   Si no
      señal sin definición
```

Esto se puede implementar en RPL de la forma

```
kpNoShift   #=casedrop :: (procesa el plano nodesplazado) ;
kpLeftShift #=casedrop :: (procesa el plano desplaz-Izq.) ;
2DROP FALSE
```

Cada controlador de un plano tiene generalmente una lógica de la forma

```
Si #CódigoTecla es 7  (o el primer código de tecla que nos interesa)
Entonces
    Devuelve la definición del código de tecla 7 y TRUE
Si no
   Si #CódigoTecla es 20  (o el siguiente código de tecla que nos interesa)
   Entonces
      Devuelve la definición del código 20 y TRUE
   Si no
      Señal sin definición
```

Esto se puede implementar en RPL de la siguiente forma:

```
kcMathMenu ?CaseKeyDef :: TakeOver (procesa MTH) ;
kcTan      ?CaseKeyDef :: TakeOver (procesa TAN) ;
( todas las demás teclas )
DROP FALSE
```

Para ahorrar código y hacer las definiciones de teclas más legibles, la palabra de control de estructuras "?CaseKeyDef" substituye la porción

```
#=casedrop :: ' <DefTecla> TRUE ;
```

del código con

```
?CaseKeyDef <DefTecla>
```

Más específicamente, ?CaseKeyDef" se usa de la forma

```
 ... #CódigoTecla #TestCódigoTecla ?CaseKeyDef <DefTecla> ...

If "#CódigoTecla" es igual a "#TestCódigoTecla", entonces "?CaseKeyDef"
elimina "#CódigoTecla" y "#TestCódigoTecla", sube "DefTecla" y TRUE, y
sale del secundario que la llamó.  Si no, "?CaseKeyDef" elimina sólo a
"#TestCódigoTecla", se salta "DefTecla" y continúa.
```

21.4.7 Asignaciones de Teclas de Menús

Una aplicación puede especificar una asignación inicial cualquiera de las teclas de menú en cualquiera de los tres planos (normal, deszp.izq. y deszp.der.) para que se inicialice cuando se inicia el bucle externo parametrizado. El parámetro del bucle externo "AppMenu" especifica el objeto de inicialización (una lista o un secundario) para el menú de la aplicación o FALSE, indicando que se dejará intacto el menú actual. Cuando se sale del bucle externo parametrizado, se restaura automáticamente el menú anterior.

Si "AppMenu" es una lista nula, entonces se hace un conjunto de seis asignaciones de teclas nulas de menú. Si "AppMenuu" es FALSE, entonces se mantiene el menú presente cuando se llama al bucle externo parametrizado.

NOTA: las asignaciones de teclas físicas (teclas "duras"/hardkeys) tienen prioridad sobre las asignaciones de teclas de menú (teclas "blandas"/softkeys). Esto significa que el controlador de teclas físicas debe incluir la siguiente línea si se van a procesar teclas de menú:

```
DUP#<7 casedrpfls
```

El parámetro AppMenu toma la siguiente forma:

```
{
  Definición de la Tecla de Menú 1
  Definición de la Tecla de Menú 2
  ...
  Definición de la Tecla de Menú n
}
```

Donde cada definición de tecla de menú toma una de las cuatro formas siguientes:

```
NullMenuKey

{ ObjEtiqueta :: TakeOver (Acción) ; }

{ ObjEtiqueta {
              :: TakeOver (Acción Primaria) ;
              :: TakeOver (Acción Deszp.Izq.) ;
            }

{ ObjEtiqueta {
              :: TakeOver (Acción Primaria) ;
              :: TakeOver (Acción Desp.Izq.) ;
              :: TakeOver (Acción Desp.Der.) ;
            }
}
```

Un ObjEtiqueta puede ser cualquier objeto, pero normalmente es una cadena de caracteres o un grob de 8x21. Ver el ejemplo que hay más adelante que ilustra el uso de teclas de menú. La palabra NullMenuKey inserta una tecla de menú en blanco que pita cuando se pulsa.

21.4.8 Evitar Entornos Suspendidos

Una aplicación puede tener que permitir evaluar comandos y objetos de usuario arbitrarios, pero no querrá que el entorno actual sea suspendido por los comandos "HALT" o "PROMPT". Si el parámetro del bucle externo "SuspendOK?" es FALSE, entonces cualquier comando que suspendería el entorno genera el error "HALT not Allowed" (HALT no Permitido), permitiendo que "AppError" lo maneje. Si "SuspendOK?" es TRUE, entonces la aplicación debe estar preparada para controlar las consecuencias. Aquí los peligros son muchos y graves.

En todas la aplicaciones previsibles, "SupendOK?" debe ser FALSE.

21.4.9 Especificar una Condición de Salida

El parámetro "ExitCond" del bucle externo es un objeto que se evalúa a TRUE cuando se va a salir del bucle externo o a FALSE si no es así. "ExitCond" se evalúa antes de cada evaluación de tecla.

21.4.10 Ejemplo de ParOuterLoop

```
*------------------------------------------------------

*
* Incluye el fichero cabecera KEYDEFS.H, que define palabras como
* kcUpArrow para números físicos de teclas.
*
INCLUDE KEYDEFS.H
*
* Incluye los ocho caracteres necesarios para la carga binaria
*
ASSEMBLE
        NIBASC  /HPHP48-D/
RPL
*
* Inicia el secundario
*
::
  RECLAIMDISP           ( *Llama la pantalla alfa* )
  ClrDA1IsStat          ( *Desactiva temporalmente el reloj* )
*                       ( *Pruébalo sin ClrDA1IsStat* )
  ZEROZERO              ( #0 #0 )
  150 150 MAKEGROB      ( #0 #0 150x150grob )
  XYGROBDISP            (  )
*
* Dibuja líneas diagonales.  Recuerda que LINEON precisa que
* - #x2>#x1 !
*
  ZEROZERO              ( #x1 #y1 )
  149 149               ( #x1 #y1 #x2 #y2 )
  LINEON                ( *Dibuja línea* )
  ZERO 149              ( #x1 #y1 )
  149 ZERO              ( #x1 #y1 #x2 #y2 )
  LINEON                ( *Dibuja línea* )
*
* Pone texto
*
  HARDBUFF
  75 50 "SCROLLING"     ( HBgrob 75 150 "SCROLLING" )
  150 CENTER$3x5        ( HBgrob )
  75 100 "EXAMPLE"      ( HBgrob 75 100 "EXAMPLE" )
  150 CENTER$3x5        ( HBgrob )
  DROPFALSE             ( FALSE )
  { LAM Exit } BIND     ( *Une la bandera de salida del POL* )
  ' DispMenu.1          ( *La Acción de Pantalla muestra el menú* )
  ' ::                  ( *Controlador de las teclas físicas*   )
    kpNoShift #=casedrop
      ::
         DUP#<7 casedrpfls ( *habilita las teclas de menús* )
         kcUpArrow    ?CaseKeyDef
                          :: TakeOver SCROLLUP ;
         kcLeftArrow  ?CaseKeyDef
                          :: TakeOver SCROLLLEFT ;
         kcDownArrow  ?CaseKeyDef
                          :: TakeOver SCROLLDOWN ;
```

```
            kcRightArrow ?CaseKeyDef
                              :: TakeOver SCROLLRIGHT ;
            kcOn         ?CaseKeyDef
                              :: TakeOver
                                 TRUE ' LAM Exit STO ;
            kcRightShift   #=casedrpfls
            DROP 'DoBadKeyT
          ;
        2DROP 'DoBadKeyT
      ;
      TrueTrue              ( *Banderas del control de teclado* )
      {
        { "TOP" :: TakeOver JUMPTOP ; }
        { "BOT" :: TakeOver JUMPBOT ; }
        { "LEFT" :: TakeOver JUMPLEFT ; }
        { "RIGHT" :: TakeOver JUMPRIGHT ; }
        NullMenuKey
        { "QUIT" :: TakeOver TRUE ' LAM Exit STO ; }
      }
      ONEFALSE              ( *Primera fila, no suspender* )
      ' LAM Exit            ( *Condición de salida de la Aplicación* )
      ' ERRJMP              ( *Controlador de Errores* )
      ParOuterLoop          ( *Ejecuta el ParOuterLoop* )
      RECLAIMDISP           ( *Cambia el tamaño y borra la pantalla* )
      SetDAsBAD             ( *Redibuja la Pantalla* )
    ;
```

Si se almacena el código anterior en un fichero llamado SCRSFKY.S se puede compilar así:

```
RPLCOMP SCRSFKY.S
SASM SCRSFKY.A
SLOAD -H SCRSFKY.M
```

Este ejemplo también supone que el fichero KEYDEFS.H o bien está en el mismo directorio o se ha modificado el fichero fuente para reflejar la ubicación de KEYDEFS.H. El fichero de control de la carga SCRSFKY.M se parece a esto:

```
OU SCRSFKY
LL SCRSFKY.LR
SU XR
SE ENTRIES.O
RE SCRSFKY.O
```

El fichero final, SCRSFKY, se puede cargar en modo binario en la HP 48 y probarse.

Cuando se está ejecutando SCRSFKY, las teclas de flecha desplazan la pantalla y las teclas de menú etiquetadas mueven la ventana al extremo correspondiente. La tecla [ATTN] finaliza la ejecución del programa.

22. Comandos del Sistema

Las siguientes palabras ponen, comprueban o controlan varias condiciones o modos del sistema.

```
ALARM?          ( --> flag )
                 Devuelve TRUE si se ha cumplido una alarma.
AtUserStack     ( --> )
                 Declara propiedad del usuario todos los objeto de la
                 pila.
CLKTICKS        ( --> hxs )
                 Devuelve una cadena hex de 13 nibbles que refleja el
                 número de marcas desde el 01/01/0000. Son 8192 marcas
                 por segundo.
ClrSysFlag      ( # --> )
                 Borra la bandera del sistema (#1 a #64)
ClrUserFlag     ( # --> )
                 Borra la bandera de usuario (#1 a #64)
DATE            ( --> %fecha )
                 Devuelve un número real con la fecha
DOBEEP          ( %frecuencia %duración --> )
                 Comando BEEP
DOBIN           ( --> )
                 Pone la base en modo BINario
DODEC           ( --> )
                 Pone la base en modo DECimal
DOENG           ( # --> )
                 Pone la pantalla en modo ENG con # (0-11) dígitos.
DOFIX           ( # --> )
                 Pone la pantalla en modo FIX con # (0-11) dígitos.
DOHEX           ( --> )
                 Pone la base en modo HEXadecimal
DOOCT           ( --> )
                 Pone la base en modo OCTal
DOSCI           ( # --> )
                 Pone la pantalla en modo SCI con # (0-11) dígitos
DOSTD           ( --> )
                 Pone la pantalla en modo STD
DPRADIX?        ( --> flag )
                 Devuelve TRUE  si la coma actual es .
                 Devuelve FALSE si la coma actual es ,
SETDEG          ( --> )
                 Pone el modo de ángulo en DEGREES (SEXAGESIMALES)
SETGRAD         ( --> )
                 Pone el modo de ángulo en GRADS (CENTESIMALES)
SETRAD          ( --> )
                 Pone el modo de ángulo en RADIANS (RADIANES)
SLOW            ( --> )
                 Retardo de 15 milisegundos
TOD             ( --> %time )
                 Devuelve la hora del día de la forma h.ms
TestSysFlag     ( # --> flag )
                 Devuelve TRUE si la bandera del sistema # está
                 activada
TestUserFlag    ( # --> flag )
                 Devuelve TRUE si la bandera de usuario # está
                 activada
VERYSLOW        ( --> )
                 Retardo de 300 milisegundos
VERYVERYSLOW    ( --> )
                 Retardo de 3 segundos
```

```
WORDSIZE        ( --> # )
                 Devuelve el ancho de palabra binario
dostws          ( # --> )
                 Almacena el ancho de palabra binario
dowait          ( %segundos --> )
                 Espera durante %segundos en estado de sueño ligero.
```