Hack

Software para HP 49G/50G

versión 1.0

mantenido por

Gustavo Portales

www.gaak.org

Contenidos

Importante 3
Introducción 3
Requerimientos – Instalación 3
Funcionamiento 4
Los Comandos 4
  COERCE 5
  XRCL 5
  XSTO 6

BZ 7
RF 8
SYS 8
DCOD 9
COD 10
OBJFIX 10
VARS2 11
PG 11
PGp 12
OB~LD 12
OB→ 12
→DIR 13
→XLIB 13
→BAK 14
→ID 14
BY 15
PMEM 15
PFREE 16
ITYPE 18
DTEMP 19
CDHD 20
GKMAP 20
ADDR 20
FOB 20
SC 20
D→LIB 20
L→DIR 21
LBCRC 21
GP 21
RINFO 21
RCFG 22
RMSG 22
REXT 23
USE 23
USED 23
USAG 23
STOX 26
GRX2 27
UP 27
WKEY 27
→FLASH 28
XMENU 29
AsmHack 29
ABOUTHACK 29
Programación 30
Agradecimientos 30

Importante

Este software se proporciona "tal como está" con la esperanza de que sea útil y se encuentra sujeto a cambio sin previo aviso. GaaK no ofrece garantía alguna con respecto a este software, incluidas, pero sin limitarse a ellas, garantías implícitas de comerciabilidad o idoneidad para un uso concreto; ni se hará responsable ante cualquier persona por daños especiales, colaterales, accidentales o consecuentes relacionados o causados por este software.

Introducción

Hack para HP 49G/50G, basado en "Hack" para HP 48 escrito por "Mika Heiskanen", es una colección de utilidades de diferentes autores. Este paquete es completamente compatible con 49G/50G.
Hack tiene entre sus más destacadas cualidades:

- Escáner de Memoria, con capacidad de mapear bancos de memoria.
- Extensión a comandos de uso común como `STO`, `RCL`, `PURGE`.
- Amplia gama de herramientas para la manipulación de bibliotecas; como la exclusiva forma de instalarlas evitando el reinicio clásico después del proceso.
- Analizador de teclado para programadores desde UserRPL hasta Machine Language.
- Los más famosos compresores: `BZ` y `RF`.
- Navegador de Bancos de Memoria.
- Informe para conocer los argumentos necesarios requeridos por determinado comando, analizador de estructuras internas de los objetos, manejo del directorio oculto, entre otros.

Hack está pensado para programadores que conocen los temas relacionados, no use las herramientas contenidas en este paquete si no tiene el conocimiento necesario.

Requerimientos – Instalación

Digite `VERSION` y presione ENTER para obtener información acerca de la versión de ROM actual en su máquina. Se necesita como mínimo de una calculadora (o emulador) HP 49G con ROM `1.18` ó HP 50G con ROM `2.00`.

Transfiera el archivo <hack.hp> de su computadora a la calculadora e instale la biblioteca en el puerto de su preferencia. Para instalar se recomienda presionar ← FILES , luego busque la biblioteca y muévala al puerto 2 (Flash). Para terminar el proceso presione simultáneamente [ON F3].

Funcionamiento

Para acceder al menú de opciones de la biblioteca Hack, realice cualquiera de los siguientes pasos:

- Digite `1219 MENU` y presione ENTER.
- Presione ↱ LIB y encuentre la etiqueta de menú "Hack ", presione finalmente F#.
- Presione APPS y encuentre la opción "Happy Hacking", luego presione ENTER.

Los Comandos

Aunque los programadores usamos el modo RPN la mayoría de veces, no se dejó de lado al modo Algebraico. Cada uno de los comandos puede ser usado en modo algebraico; por ejemplo el ejecutar `1219 COERCE` en modo RPN es lo mismo a ejecutar `COERCE(1219)` en modo algebraico. Tenga en cuenta que el objeto del nivel 1 puede ser llamado con `ANS(1)` en modo algebraico; así, podríamos ejecutar `BY(ANS(1))` para obtener el tamaño en Bytes del primer objeto en la pila.
Los comandos a continuación descritos son los disponibles en el paquete Hack, enumerados en orden para hacerlos de esta manera comandos soportados (invariables). En caso de existir futuras versiones o actualizaciones de Hack, se asegura el orden de los siguientes comandos:

Hack

1 COERCE	2 XRCL	3 XSTO	4 BZ	5 RF	6 SYS
7 DCOD	8 COD	9 OBJFIX	10 VARS2	11 PG	12 PGp
13 OB~LD	14 OB→	15 →DIR	16 →XLIB	17 →BAK	18 →ID
19 BY	20 PMEM	21 PFREE	22 ITYPE	23 DTEMP	24 CDHD
25 GKMAP	26 ADDR	27 FOB	28 SC	29 D→LIB	30 L→DIR
31 LBCRC	32 GP	33 RINFO	34 RCFG	35 RMSG	36 REXT
37 USE	38 USED	39 USAG	40 STOX	41 GRX2	42 UP
43 WKEY	44 →FLASH	45 XMENU	46 AsnHack	47 ABOUTHACK	

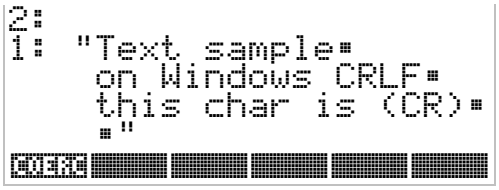
Las excepciones de comandos soportados son los dos últimos, ellos estarán siempre al final de la lista (si es que se añadieran otros comandos en el futuro, estarían después de `XMENU`).

Cada uno de los comandos requiere combinaciones de argumentos distintos para su operación, revise los siguientes detalles proporcionados.
Note también que los diversos argumentos requeridos pueden ser ingresados en listas, por ejemplo el comando COERCE podría operar con el argumento ‹ 1219 '4+5' #4C3h›.
En los detalles proporcionados a continuación para cada comando, es posible que encuentre al final algo como: ← Proceso Inverso; esto le describirá el proceso inverso.

COERCE

El primer comando del paquete Hack. Éste es un conversor de objetos pensado en sustituir y ampliar la gama de opciones del comando →NUM. Este comando requiere un argumento en la pila:

- Coloque un número entero y COERCE lo convertirá en real, similar a →NUM.
- Coloque un número real y COERCE lo convertirá en un número binario de sistema (SB), similar al comando R~SB.
- Coloque un simbólico (vector o matriz) y COERCE evaluará cada una de las componentes, similar a →NUM. Por ejemplo COERCE convertirá al vector [4 '7*3'] en [4. 21.].
- Coloque un número entero binario y COERCE lo convertirá en SB, por ejemplo # 5h → ¤ 5h.
- Coloque un SB y COERCE lo convertirá en entero, por ejemplo ¤ 5h → 5.
- Coloque un número real extendido y COERCE lo convertirá en número real simple. ← SYS.
- Coloque un número complejo extendido y COERCE lo convertirá en número complejo simple. ← SYS.
- Coloque un carácter y COERCE colocará este carácter en una cadena de caracteres.
- Coloque una cadena de caracteres (texto) y COERCE removerá los finales de línea que se ven como un molesto carácter al final de cada línea. Esto ocurre constantemente con textos creados en Windows que usa como fin de línea a CRLF. COERCE reemplazará el carácter CR por un espacio.



- Finalmente para cualquier otro argumento no especificado antes, COERCE aplicará →NUM, por ejemplo para expresiones algebraicas generadas en el editor de ecuaciones, como '4+5'.

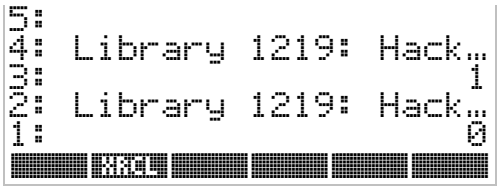
COERCE puede ser asignado en una tecla de usuario mediante el comando AsnHack.

XRCL

Basado en el comando RCL (en inglés "Recall"). XRCL requiere un argumento en el nivel 1 de la pila:

- Coloque un objeto comprimido ($bz) y XRCL devolverá el objeto original (descomprimido). Para esto se llama internamente al comando BZ.
- Coloque el nombre de una variable (global o local) y XRCL devolverá el objeto almacenado en ella, similar a RCL.
- Coloque PICT y XRCL devolverá el gráfico del PICTURE, similar a RCL.

- Coloque un puntero ROM (en inglés "ROM pointer") y XRCL devolverá el objeto contenido en él. (Use el comando →XLIB para obtener un puntero ROM). Por ejemplo el programa contenido en el comando →H se obtendría ejecutando: 256 0 →XLIB XRCL.
- Coloque una lista conteniendo:
  – { HOME CASDIR VX }, para obtener en el nivel 1 la variable independiente del CAS.
  – { « 1 2 + » }, para obtener en el nivel 1 el programa « 1 2 + ».
  – { COERCE }, para obtener en el nivel 1 el programa contenido en el comando COERCE.
  El primer caso tiene la forma { path }, donde "path" es la ruta a un objeto, similar a RCL.
  El segundo caso tiene la forma { seco }, donde "seco" es un programa.
  El tercer caso tiene la forma { romptr }, donde "romptr" es un puntero ROM explicado en el ítem anterior. El mismo resultado se obtendría con 1219 0 →XLIB XRCL.
- Coloque una dirección (hxs ó SB) y XRCL devolverá el objeto alojado en dicha dirección, similar al comando A→. Por ejemplo #33C2Dh XRCL.
- Coloque un puntero de acceso (Extended Ptr) y XRCL devolverá el objeto relacionado.
- Coloque un número entero o real y XRCL devolverá en el nivel 1 la cantidad de objetos alojados en el puerto (si el valor numérico fue menor a 3) y los nombres (o rutas) de los objetos son mostrados en los niveles sucesivos como se muestra en el primer gráfico para el caso del puerto 1. Para el resto de valores numéricos (mayores a 2) se considerará como identificadores de bibliotecas (lid), para estos casos XRCL buscará la biblioteca en todos los puertos y colocará la biblioteca y el puerto de ubicación encontrados. El segundo gráfico muestra los resultados para la búsqueda del lid 1219 y nos dice que la biblioteca 1219 se encuentra alojada en los puertos 0 y 1.



- Coloque un objeto etiquetado y XRCL se comportará de forma similar a RCL, con la ventaja de que para un lid se reconocerá a un: entero, real, bint, hxs, como por ejemplo: :&:#1219d.
- Coloque un puntero Flash (en inglés "Flash pointer") y XRCL devolverá el objeto contenido en él. (Para crear punteros Flash, refiérase al comando →FLASH).

XRCL puede ser asignado en una tecla de usuario mediante el comando AsnHack.

XSTO

Basado en el comando STO (en inglés "Store"). XSTO requiere dos argumentos en la pila:

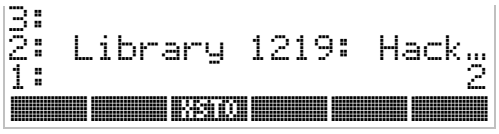
- Coloque cualquier objeto en el nivel 2 y el nombre de una variable (global o local) en el nivel 1 y XSTO intentará almacenar el objeto en la variable, similar a STO.

- Coloque un gráfico en el nivel 2 y PICT en el nivel 1 y XSTO almacenará el gráfico en PICTURE, similar a STO.

- Coloque un objeto algebraico en el nivel 2 y un simbólico en el nivel 1 y XSTO se comportará de forma similar a STO. El ejemplo de la derecha asume la existencia de una matriz 7x8 almacenada en la variable freq, y colocará 'X+9' en la fila 2, columna 3 de la matriz.

- Coloque cualquier objeto en el nivel 2 y un objeto etiquetado en el nivel 1 y XSTO guardará el objeto del nivel 2 en el destino indicado en el nivel 1. Por ejemplo: LCD→ :0:LCD1 XSTO almacenará el gráfico de la pantalla actual de su calculadora en la variable LCD1 del puerto 0, de forma similar a STO. Lo diferente ocurre cuando el nivel 1 y el nivel 2 son el mismo objeto etiquetado, en estos casos el objeto mencionado será copiado a su directorio actual (mayormente HOME). El gráfico de la derecha indica lo necesario para hacer una copia en HOME de la variable LCD1 almacenada en el puerto 0. Esto aplica también a bibliotecas (:0:1219 DUP XSTO copiará la biblioteca 1219 instalada en el puerto 0 a la variable L1219 en HOME).

- Coloque un objeto Backup en el nivel 2 y el número de un puerto en el nivel 1 para almacenar el contenido del objeto Backup en el puerto indicado, similar a STO. (Para crear objetos Backup refiérase al comando →BAK).
- Finalmente, coloque una biblioteca en el nivel 2 y el número de un puerto de destino (0, 1, 2) en el nivel 1 para instalar dicha biblioteca en el puerto mencionado. Con XSTO olvídese de ejecutar ATTACH o de reiniciar después de cada instalación como ocurre con el comando STO. Después de ejecutar XSTO su biblioteca estará lista para usar. (Es posible que antes de terminar el proceso observe el mensaje "Weird Config", ello sucederá cuando la variable interna $CONFIG de la biblioteca instalada no sea común, siendo considerados como forma común a los que usan xATTACH, TOSRRP, XEQSETLIB).

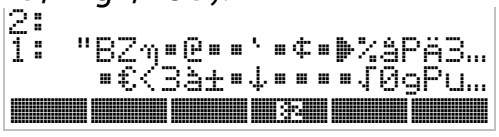


XSTO puede ser asignado en una tecla de usuario mediante el comando AsnHack.

BZ

El famoso compresor/descompresor de la serie 48 que perduró y ahora en la serie 49 el descompresor viene incluido en la ROM. (Serie 49 abarca a los modelos: 48gII, 49G, 49g+, 50G).

El uso es sencillo, sólo coloque en el nivel 1 de la pila un objeto (cualquiera) y al presionar [▦] éste será comprimido obteniendo como resultado una cadena de caracteres (objeto comprimido) similar al mostrado en la imagen de la derecha ($bz). Durante el proceso de compresión una animación con los indicadores le dirán que la máquina está trabajando (si presiona [ON] el proceso será interrumpido).



Al terminar el proceso de compresión usted verá en pantalla los siguientes datos relacionados:

- Tamaño en Bytes antes y después de la compresión: Size: 1098 → 339.5.
- Porcentaje de compresión (valores negativos indican reducción): %CH: -69.08.
- Tiempo de compresión expresado en segundos: Time: 4.599.

El descompresor (también disponible desde BZU) requiere una $bz en el nivel 1 de la pila (como en la imagen antes mostrada) y al presionar [R/S] éste devolverá el objeto original (descomprimido).

RF

El comando RF (en inglés "Redundance Fighter") es un compresor/descompresor diseñado para trabajar en condiciones de poca memoria.

Coloque un objeto (cualquiera) en el nivel 1 de la pila y ejecute RF para comprimirlo, obteniendo un objeto comprimido similar al mostrado en la imagen ($rf). Durante el proceso de compresión una animación con los indicadores le dirán que la máquina está trabajando. Cuando el proceso de compresión termina verá los siguientes datos (similar a BZ):

- Tamaño en Bytes antes y después de la compresión: Size: 1098 → 365.
- Porcentaje de compresión (valores negativos indican reducción): %CH: -66.76.
- Tiempo de compresión expresado en segundos: Time: 5.072.

El descompresor (también disponible desde RFU) requiere una $rf en el nivel 1 de la pila (como en la imagen antes mostrada) y al presionar [R/S] éste devolverá el objeto original (descomprimido).

SYS

Ampliación para el comando SYSEVAL, que permite evaluar apuntando directamente a direcciones de memoria (muy parecido a la programación SysRPL). También tiene un conversor para objetos de punto flotante:

- Precaución: La siguiente descripción debe ser usada con cautela en el uso de direcciones.
  Coloque una dirección de memoria (entero binario o SB) y SYS se comportará de forma similar a SYSEVAL, por ejemplo con #340C6 SYS obtendrá la versión de ROM de su calculadora.
  Note que SYS puede usarse con →FLASH para evaluar desde Flash; por ejemplo para obtener el gráfico usado en los títulos de navegadores a pantalla completa, ejecute:
  - En ROM 1.18; #5F534h « SYS » 2 0 →FLASH EVAL.
  - En ROM 1.19-6; #63081h « SYS » 2 0 →FLASH EVAL.
  - En ROM 2.09; #63C9Ah « SYS » 2 0 →FLASH EVAL.

    Estos ejemplos fueron extraídos desde el FPTR 2 2F (en SysRPL) y las direcciones se evalúan necesariamente con FPTR 2 0.

    Esto traducido a SysRPL sería: :: #63C9A ' xSYS FPTR 2 0 ;
- Coloque un número entero o real y SYS lo convertirá en real extendido (y viceversa).
- Coloque un número complejo simple y SYS lo convertirá en complejo extendido (y viceversa).
- Coloque una matriz (o vector) de reales y SYS convertirá cada uno de los números reales en reales extendidos (y viceversa).
  Note que "todos" los elementos deben ser del mismo tipo (reales o reales extendidos). Para la conversión de un vector (o matriz) real a real extendido se recomienda usar la forma COERCE SYS; por ejemplo [4 5 '6+9'] COERCE SYS.

DCOD

Comando que requiere un objeto (de cualquier tipo con prólogo) en el nivel 1 y devolverá una cadena de caracteres hexa ($dcod) que representa al objeto antes mencionado, similar a →H. La diferencia entre DCOD y →H es que $dcod mostrará estructuras RPL para los siguientes tipos objetos:

- Programas simples, con prólogo #02D9D.
- Listas, con prólogo #02A74.
- Expresiones algebraicas, con prólogo #02AB8.
- Objetos unidad, con prólogo #02ADA.

El prólogo son los primeros cinco nibbles de la estructura hexa de un objeto. Existen objetos que no tienen prólogo como por ejemplo los External.

Ejemplos

Objeto	→	$dcod	@ comentario
"+"	→	C2A2070000B2	@ "+"
« 1 → X 'X+1' »	→	D9D20 9B983 6B372 39083 D6E201085 8BA20 D6E201085 6B372 85B93 B2130 4D983 B2130	@ :: @ x<< @ Z1_ @ xALG→ @ LAM X @ SYMBOL @ LAM X @ Z1_ @ x+ @ ; @ x>> @ ;
« "HOLA" DIR END # 2E139h SYS »	→	D9D20 9B983 C2A20D000084F4C414 69A20FF700000 E4A2051000931E200000000000 29E203C4500 4D983 B2130	@ :: @ x<< @ "HOLA" @ DIR ENDDIR @ # 2E139h @ xSYS (ROMPTR 4C3 005) @ x>> @ ;
1_s	→	ADA20 C49F2 C2A207000037 777D2 B2130	@ UNIT @ %1 @ "s" @ umEND @ ;

La entrada SysRPL "MakeDir/StdLabel_" tiene como dirección #2E139h usado en el tercer ejemplo, si usted desea ver la estructura interna de esta entrada SysRPL, ejecute: #2E139h XRCL DCOD.

COD

Si usted vio la descripción de uso del comando DCOD, entenderá fácilmente el funcionamiento de COD (en inglés "Code"), que codificará una $dcod y obtendrá el objeto original:

$dcod	→	Objeto
C2A207000004	→	"@"
ADA20 C49F2 C2A207000004 777D2 B2130	→	1_@

Para casos de error, COD devolverá la posición del primer carácter defectuoso. Supongamos un $dcod de la forma: "C2A2070000X4", al ejecutar el comando COD usted obtendrá un mensaje de error de sintaxis inválida y en el nivel 1 de la pila el número 11, que es la posición del primer carácter erróneo "X". Use el editor de textos de su calculadora para editar la $dcod; busque y ejecute la opción [TOOL] seguido de [GOTO] (en inglés "Go to position"), ingrese 11 y presione [ENTER] para ir a la posición del primer carácter defectuoso. Esto es realmente útil para $dcod grandes, en donde los errores no se ven a simple vista como en este caso (X no es valor hexa).

Advertencia

No intente usar una $dcod con el comando →H.

OBJFIX

Comando para fijar posibles errores ocurridos en la transferencia de un objeto… Cada objeto tiene en la PC una cabecera de la siguiente forma:

- Serie 48: "HPHP48-x<objeto>" (48SX, 48GX).
- Serie 49: "HPHP49-x<objeto>" (48gII, 49G, 49g+, 50G).

… esta cabecera no es necesaria en las calculadoras, sólo son válidas en la PC.

Los errores de transferencia ocurren mayormente cuando se intenta transferir un objeto de la serie 48 a una calculadora de la serie 49 (ob48 → 49). En estos casos la transferencia conseguirá el objeto completo (cadena de caracteres incluida la cabecera), entonces OBJFIX se encargará de remover la cabecera para obtener sólo el objeto. Tenga presente también que no todos los objetos de la serie 48 se ejecutan correctamente en la serie 49, por ejemplo los programas, bibliotecas, directorios.

Por ejemplo, asuma que ya se transfirió a una calculadora de la serie 49 un gráfico que se encuentra almacenado en la variable 'GRF48', al llamar el objeto a la pila usted verá algo similar a la imagen adjunta. Para corregir el error ejecute: 'GRF48' OBJFIX y obtendrá el objeto deseado (en la variable original).

Note que OBJFIX verificará los cinco primeros caracteres de la cadena, esto es: "HPHP4", pensado para compatibilidad con las dos series.

VARS2

La segunda versión del comando VARS, con VARS2 conseguirá todas las variables contenidas en un directorio, incluyendo a las variables ocultas y nulas.
Por ejemplo presione [VAR] y busque e ingrese al sub-directorio CASDIR (o cualquier otro) y verá luego el menú con los nombres de las variables contenidas, como por ejemplo: MODULO, REALASSUME PERIOD. En este punto, ejecute 1219 "" →ID XSTO y verá que todas las variables serán ocultadas (pero siguen presentes, no se preocupe). Si ejecuta luego el comando VARS, éste le devolverá una lista vacía (supuestamente no hay variables por estar ocultas), en cambio VARS2 si le devolverá las variables contenidas en el directorio actual, incluyendo las ocultas y nulas. Para comprobar, asuma que VARS2 le devolvió: { MODULO REALASSUME PERIOD VX EPS }, aparentemente hay cinco variables pero al ejecutar SIZE (a esta lista) obtendrá 6, esto es porque hay una variable nula que no se ve a simple vista (puede usar →S2 para ver la estructura SysRPL de esta lista). Puede también usar ORDER para mover la variable nula, y ella ocultará sólo a las variables siguientes. Para remover la variable nula que oculta el resto de variables ejecute "" →ID PG.

Advertencia

No! usar VARS2 en HOME combinado con PG (o mejor dicho ejecutar HOME VARS2 PG), ello borraría el directorio oculto que (entre otras cosas) contiene la configuración del teclado.

PG

Sin lugar a dudas el reemplazo por excelencia para el comando PURGE por tener un nombre pequeño y por su gran potencia con diversos tipos de objetos:

- Coloque PICT y PG borrará el gráfico del PICTURE, similar a PURGE.
- Coloque un objeto etiquetado y PG se comportará de forma similar a PURGE con la excepción para bibliotecas. Olvídese en estos casos de ejecutar :2:1219 DUP DETACH PURGE, con PG bastará con ejecutar :2:1219 PG.
- Coloque el nombre de una variable (global) y PG borrará su contenido (incluyendo nombres de directorios), fusionando de esta manera los comandos PURGE y PGDIR. Inténtelo ejecutando por ejemplo: 'CASDIR' PG.
- Coloque una lista con los diversos tipos de objetos que soporta PG para borrar todos ellos en un solo paso. Por ejemplo: { CASDIR 1219 STARTED } PG. Esto es usado por ejemplo en el comando PGp (lo obtenido por PVARS DROP es una lista con todos los nombres de variables en un puerto). Para listas, el comando PG simplemente es re-evaluado para cada uno de los objetos (uno a uno en el orden listado).
- Coloque un número entero o real y PG asumirá que es un lid (lid: identificador de biblioteca), luego detectará el puerto de instalación de la biblioteca en mención, le aplicará un DETACH y finalmente la removerá (ejecutando después un ATTACH para posibles bibliotecas con el mismo

lid en otro puerto)… Por ejemplo; asuma la biblioteca 1219 instalada en el puerto 0 y en el puerto 2 (como se sabe las bibliotecas le dan preferencia al puerto menor, en este caso la biblioteca operativa es la instalada en el puerto 0), entonces al ejecutar `1219 PG` se detectará este puerto para la biblioteca 1219 (vía GP) y removerá la biblioteca 1219 del puerto 0 finalizando con la habilitación de la biblioteca 1219 del puerto 2.

`PG` puede ser asignado en una tecla de usuario mediante el comando `AsnHack`.

PGp

Con un nombre tan pequeño pero de gran potencia, el comando `PGp` (en inglés "Purge Port") borrará todos los objetos de un puerto, incluyendo la biblioteca Hack (que es desde donde se llama a PGp). Para que esto suceda sólo se debe colocar el número del puerto (0, 1, 2) en el nivel 1 de la pila y luego ejecutar `PGp` (por ejemplo `2 PGp`). Observe que antes de empezar la tarea, PGp le preguntará si está seguro (como confirmación usted debe indicar que "Si").
Se recomienda ejecutar antes `CLEAR MEM DROP` para remover objetos temporales y evitar mensajes de error como "Objeto en uso".
Note también que PGp no trabajará para el puerto 3 en la 50G ya que PGp es equivalente a ejecutar `2 PVARS DROP PG` (actualmente `PVARS` no trabaja con el puerto 3).

OB~LD

Simplemente convierte el objeto del nivel 1 de la pila en Datos de biblioteca (objeto LD, en inglés "Library Data") y viceversa. `OB~LD` usa un identificador que permitirá ignorar a otros LD que no hayan sido generados con este comando de Hack. Los objetos a convertir deberán tener prólogo (refiérase a `DCOD` para obtener información acerca de prólogos).

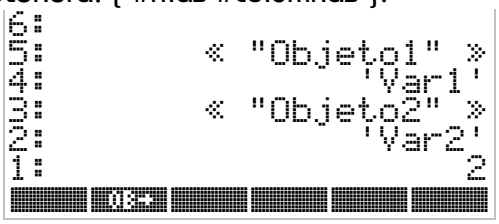
OB→

Comando que requiere un objeto compuesto para devolver cada una de las componentes en la pila:

- Coloque un programa simple y descompondrá el objeto devolviendo en el nivel 1 el número de elementos. ← `→PRG`.
- Coloque un objeto algebraico y `OB→` devolverá el objeto descompuesto con el número de elementos en el nivel 1 de la pila. ← `→ALG`. Observe la diferencia con `OBJ→` :
  - `'4+5' OB→` devuelve como resultado `4 5 + 3.`
  - `'4+5' OBJ→` devuelve como resultado `4 5 2. +`
- Coloque un vector (o matriz) y `OB→` devolverá cada una de las componentes en la pila y el número de elementos en el nivel 1. Por ejemplo: `257 RMSG OB→` devolverá todas las cadenas

de mensajes de la biblioteca 257 en la pila y en el nivel 1: `« 24 »` (asumiendo ROM 2.09); que indica 24 elementos. Para el caso de matrices usted obtendrá: { #filas #columnas }.

- Coloque un directorio y `OB→` devolverá en el nivel 1 de la pila el número de variables del directorio seguido de cada una de ellas con sus respectivos objetos, como se aprecia en la imagen de la derecha. ← `→DIR`.



- Coloque una biblioteca y `OB→` intentará convertirla en directorio, creándolo de la forma L1219 (asumiendo un Lid 1219). Si el proceso fue satisfactorio presione [VAR] para ver las variables de su nuevo directorio, basado en la biblioteca ingresada al inicio.

También, se agregó el soporte a los siguientes tipos de objetos:

- Coloque un nombre global y `OB→` convertirá esto en cadena de caracteres. ← `→ID`.
- Coloque un puntero ROM y `OB→` devolverá los dos números enteros (Lid, cmd). ← `→XLIB`.
- Coloque un puntero Flash y `OB→` devolverá los dos números enteros (Bid, FPTRid). ← `→FLASH`.
- Coloque un puntero de acceso (`Extended Ptr`) y `OB→` devolverá el objeto relacionado.
- Coloque un objeto de reserva y obtendrá el objeto original en el nivel 2 y el nombre del objeto de reserva en el nivel 1. ← `→BAK`.
- Coloque un objeto LD y `OB→` aplicará el comando `OB~LD`.
- Coloque un Lid (identificador de biblioteca) entero o real y `OB→` buscará la biblioteca en cada uno de los puertos (de 0 a 2) hasta encontrarla, la colocará en la pila y le aplicará `OB→` para intentar visualizarla como directorio.

→DIR

Así como se pueden generar listas (`→LIST`), programas (`→PRG`), objetos algebraicos (`→ALG`)... ahora es posible con `→DIR` (en inglés "To Directory") generar directorios a partir de los datos colocados en la pila con sus respectivos nombres de variables.
La imagen de la derecha propone la forma para crear el directorio llamado EJEMP de `2` variables (en orden):

- VAR01 → `« "Objeto1" »`.
- VAR02 → `« "Objeto2" »`.

En el ejemplo se usa la forma Var01 ó Var02 sólo para describir el orden en que serán creadas las variables del directorio, usted puede usar cualquier nombre y la cantidad de variables que desee.

→XLIB

Comando necesario para crear punteros ROM. La imagen de la derecha muestra el resultado de ejecutar: `#4C3h 9 →XLIB`. Así, este comando requiere dos números, primero un Lid (identificador

de biblioteca: #4C3h), seguido del número de un comando (9). Las combinaciones de los números para obtener el mismo puntero ROM pueden ser:

- Dos enteros o reales, por ejemplo: 1219 9 →XLIB. ← OB→.
- Dos enteros binarios, por ejemplo: #4C3h #9d →XLIB.
- Cualquiera de los dos anteriores: #4C3h 9 →XLIB, ó 1219 #9h →XLIB.

Los punteros ROM muestran un nombre asociado (si es que lo tienen; en el ejemplo VARS2), caso contrario se mostrarán como ROMPTR 4C3 98 ó XLIB 1219 152 (asumiendo 1219 152 →XLIB).

Ejemplos

Proceso:	→	... desde →XLIB:
APPS	→	161 6 →XLIB EVAL
MODE	→	161 7 →XLIB EVAL
↰ FILES	→	162 6 →XLIB EVAL
↰ MTRW	→	162 18 →XLIB EVAL
↱ LIB	→	163 43 →XLIB EVAL
↱ TIME	→	163 34 →XLIB EVAL

Los ejemplos mostrados son simplemente algunos basados en procesos del teclado.

→BAK

Comando que requiere dos argumentos en la pila, en el nivel 2 cualquier objeto y en el nivel 1 un nombre. Al ejecutar →BAK un objeto de reserva (en inglés "Backup object") será creado. ← OB→. Luego, éste objeto de reserva puede almacenarse en uno de los puertos (por ejemplo con 2 XSTO).
Los objetos de reserva se utilizan para copiar datos del directorio HOME a un puerto de memoria, con el propósito de preservar su contenido para uso futuro (por eso se llaman objetos de reserva).
Cuando se crea un objeto de respaldo la calculadora obtiene un valor llamado CRC (verificación de redundancia cíclica, conocido también como checksum). Este valor se basa en los datos binarios del objeto en interés y se almacena junto con el objeto de reserva para usarse como dato de verificación, que comprobará la integridad del objeto de reserva.
En operaciones que involucran a un objeto de reserva, la calculadora recalcula el valor CRC y lo compara con el valor original (almacenado en el objeto de reserva), si se encuentra una discrepancia la calculadora advierte que los datos pueden estar corruptos: "Invalid Card Data".

→ID

Coloque en el nivel 1 de la pila una cadena de caracteres y →ID la convertirá en un nombre de variable (global), permitiendo de esta manera guardar objetos hasta en nombres reservados y soportar espacios. La imagen muestra un ejemplo, a continuación otros:

- Ejecute `1.5 "3/2" →ID XSTO` para almacenar en una variable con caracteres especiales "/".
- Ejecute `1219 "lid Hack" →ID XSTO` para almacenar en una variable con espacios.
- Ejecute `« 5000 0.02 BEEP » "BEEP" →ID XSTO` para almacenar el programa de sonido en la variable 'BEEP', ahora si usted ejecuta `BEEP` desde la línea de comandos se ejecutará este programa.
- Ejecute `"" →ID EVAL` y presione [VAR] para ver el directorio oculto (ejecute `UP` para salir).
- Presione [VAR] y busque e ingrese al sub-directorio CASDIR (o cualquier otro) y verá enseguida el menú con los nombres de las variables contenidas, como por ejemplo: [KOUU], [KEIN] [KEIO]. En este punto, ejecute `1219 "" →ID XSTO` y verá que todas las variables serán ocultadas (pero siguen presentes, no se preocupe). Para remover la variable nula que oculta el resto de variables ejecute `"" →ID PG`. (Para detalles similares refiérase al comando `VARS2`).

El máximo de caracteres soportados por `→ID` es 127.

BY

Probablemente el más simple de los comandos de Hack, coloque cualquier objeto en el nivel 1 y `BY` (en inglés "Bytes") devolverá su tamaño en Bytes.
Valores relacionados:

- 1 Kilobyte = 1024 Bytes… 1 Byte = 2 nibbles… 1 Byte = 1 Octeto… 1 nibble = 1 cuarteto.
- Número de nibbles = Número de Bytes multiplicado por 2: `BY 2 *`. Esto es similar a `→H SIZE`.

Ejemplo para conseguir el tamaño en Bytes del directorio actual: `#8D5Ah SYS BY "Size" →TAG`, donde #8D5Ah es la dirección de la entrada SysRPL "CONTEXT@".

PMEM

Use `PMEM` (en inglés "Port Memory") para obtener los Bytes libres en un puerto válido. Los puertos válidos son `0:IRAM`, `1:ERAM`, `2:FLASH`, `3:SD` (en caso de estar insertada una tarjeta SD en la 50G). Ingrese en el nivel 1 de la pila el número del puerto (por ejemplo `2`) y luego presione [PMEM]. El resultado obtenido en este caso será el mismo que al ejecutar `2 PVARS SWAP DROP`. En el caso del puerto 0 (`0 PMEM`) se diferencia del comando `MEM` en que éste realiza un GARBAGE antes.

Recuerde que `PMEM` devolverá siempre el valor en Bytes. Si desea convertir a Kilobytes (KB) debe dividir los Bytes entre 1024. El gráfico de la derecha es el resultado del siguiente programa: `2 PMEM DUP 1024 / IP R→I`, donde el nivel 2 muestra el valor en Bytes (obtenido por `PMEM`) y el nivel 1 muestra su equivalente en Kilobytes (usados en [←] FILES).

Como ejemplo final ejecute el siguiente programa y obtendrá la memoria libre "total" de la máquina:
`{0 1 2} PMEM ΣLIST DUP "Bytes" →TAG SWAP 1024 / IP R→I "KB" →TAG`.

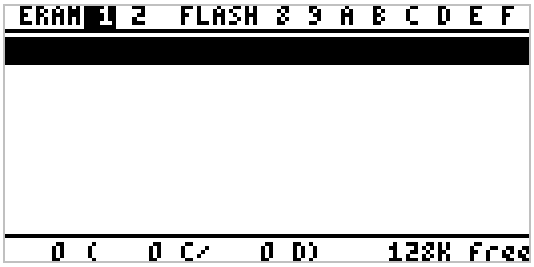
PFREE

Uno de los comandos más interesantes de Hack, con PFREE (en inglés "Port Free") obtendrá el contenido de los puertos 1 y 2 divididos en sus respectivos segmentos de memoria:

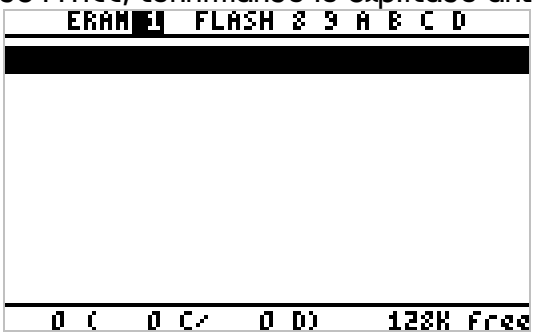
- Puerto 1; conocido también como puerto "ERAM", se subdivide en:
  - 49G: Segmentos ERAM1 y ERAM2 (cada uno de 128KB: 2*128KB=256KB)
  - 50G: Segmento ERAM1 (sólo un segmento: 1*128KB=128KB)
- Puerto 2; conocido también como puerto "FLASH", se subdivide en bancos de memoria por pertenecer al segmento ROM (en inglés "Read-Only Memory"):
  - 49G: Bancos de usuario disponibles del 8 al 15 (8 bancos: 8*128KB=1024KB=1MB).
  - 50G: Bancos de usuario disponibles del 8 al 13 (6 bancos: 6*128KB=768KB)

Cada segmento siempre es de 128KB, es por eso que no es posible instalar bibliotecas o almacenar objetos mayores a 128KB en ninguno de los puertos.

A continuación se muestra una posible pantalla inicial de PFREE, confirmando lo explicado antes:



49G



50G

Funcionamiento

Las imágenes anteriores muestran pantallas de PFREE vacías. Observe ahora la imagen de la derecha, es una pantalla PFREE (virtual) capturada de una 50G y tiene tres partes:

- En la primera fila (cabecera) se muestran los bancos o segmentos de memoria disponibles, tanto de ERAM como Flash. El segmento de memoria seleccionado se muestra resaltado (en el gráfico ERAM1).
- En la fila final se muestra un resumen del segmento de memoria seleccionado actualmente (ERAM1). Se divide en cuatro partes que han sido ligeramente resaltadas con barras horizontales:
  - Número total de objetos encontrados en el segmento de memoria seleccionado (16 objetos en ERAM1).
  - Cantidad de objetos con CRC defectuoso (de los 16 objetos, 2 tienen mal CRC).
  - Cantidad de objetos que han sido removidos (de los 16 objetos, 4 han sido borrados).
  - Cantidad de memoria libre en el segmento (29 Kilobytes libres en ERAM1).
- Las filas intermedias (punteadas en la imagen) muestran los objetos e información relacionada a ellos:

- Tipo de objeto (1219, MFNT, GROB, CODE, INTG…). Observe que para bibliotecas se muestra su Lid ("identificador de biblioteca" como 1219, 275, 995…).
- Dos caracteres especiales son usados enseguida para indicar al objeto como "borrado" o con "CRC defectuoso", para ello se usa "D" y "C" respectivamente.
- Nombre del objeto (ó el título si es que es una biblioteca). Si esto es muy grande, se mostrará "…" al final.
- Tamaño del objeto en Bytes o Kilobytes.

Características de PFREE

- De diseño:
  - PFREE mostrará los tamaños en Kilobytes si el valor de Bytes es mayor a 9999. para ello agregará el símbolo "K".
  - PFREE trabaja siempre con el tipo de caracteres pequeños (Minifont).
  - El máximo de filas intermedias varía de acuerdo al modelo, en 49G el máximo es de 8 mientras que en la 50G es de 10.
- De funcionalidad:
  - Las teclas disponibles son:

◀	→ Selecciona el segmento de memoria previo.
▶	→ Selecciona el segmento de memoria siguiente.
▲	→ Selecciona el objeto previo.
▼	→ Selecciona el objeto siguiente.
STO▶	→ Coloca el objeto seleccionado en la pila.
ON	→ Termina PFREE (también con ◀).

    La opción STO▶ funcionará con cualquier tipo de objeto incluso con objetos borrados ("D") de alguno de los bancos Flash. Después de colocar el objeto la aplicación PFREE terminará.
  - Note que el seleccionar un nuevo segmento de memoria (usando ◀, ▶) por primera vez puede ser un poco lento (más sensible en 49G), debido a que PFREE necesita escanear todo el segmento (o banco) de memoria y verificar el CRC de los objetos.

Tipos de objeto en PFREE

Usados en la primera columna de las filas intermedias, los tipos de objetos son:

PFREE	Objeto	Prólogo	→Nombre
%	número real	02933	DOREAL
C%	número complejo	02977	DOCMP
"…"	cadena de caracteres	02A2C	DOCSTR
[]	vector	029E8	DOARRY
[[]]	matriz	02686	DOMATRIX
LIST	lista	02A74	DOLIST
ID	identificador (global)	02E48	DOIDNT
LAM	identificador (local)	02E6D	DOLAM
PROG	programa	02D9D	DOCOL
ALG	simbólico	02AB8	DOSYMB
FPTR	Puntero Flash	026AC	DOFLASHP

HXS	número entero binario	02A4E	DOHXS
GROB	objeto gráfico	02B1E	DOGROB
TAG	objeto etiquetado	02AFC	DOTAG
UNIT	objeto unidad	02ADA	DOEXT
XLIB	nombre XLIB	02E92	DOROMP
BINT	entero binario de sistema	02911	DOBINT
DIR	directorio	02A96	DORRP
%%	número real extendido	02955	DOEREL
C%%	número complejo extendido	0299D	DOECMP
LK[]	arreglo enlazado	02A0A	DOLNKARRY
CHAR	carácter	029FB	DOCHAR
CODE	código	02DCC	DOCODE
1219	biblioteca (Hack)	02B40	DOLIB
LDAT	datos de biblioteca	02B88	DOEXT0
ACPT	puntero extendido	02BAA	DOACPTR
FONT	tipo de caracteres	02BCC	DOEXT2
MFNT	tipo de caracteres pequeños	026FE	DOMINIFONT
EXT3	reservado 3	02BEE	DOEXT3
EXT4	reservado 4	02C10	DOEXT4
INTG	número entero	02614	DOINT
?!?	otro	?????	DO?????

ITYPE

Con ITYPE (en inglés "Internal Type"), conseguirá el tipo de cualquier objeto, "similar" a TYPE. Vea en la siguiente tabla los tipos de objetos y las diferencias de ITYPE con TYPE:

Abrev.	Objeto	Dispatch	Prólogo	ITYPE	TYPE	Ejemplo
ob	cualquier tipo de objeto	any		¤ 0h	27.	FlashPtr
%	número real	real	02933	¤ 1h	0.	1219.
C%	número complejo	cmp	02977	¤ 2h	1.	(53.,90.)
$	cadena de caracteres	str	02A2C	¤ 3h	2.	"Hack"
[]	vector o matriz	arry	FFFFF	¤ 4h	29.	[1 2 3]
{}	lista	list	02A74	¤ 5h	5.	{1 2 3}
id	identificador (global)	id	02E48	¤ 6h	6.	'Hack'
lam	identificador (local)	lam	02E6D	¤ 7h	7.	'Hack'
seco	programa	seco	02D9D	¤ 8h	8.	« 8 2 * »
symb	simbólico	symb	02AB8	¤ 9h	9.	X+9
alg	algebraico	sym	00000	¤ Ah	10.	
hxs	número entero binario	hxs	02A4E	¤ Bh	11.	# 1219d
grb	objeto gráfico	grob	02B1E	¤ Ch	12.	Graphic 131 x 64
tag	objeto etiquetado	TAGGED	02AFC	¤ Dh	13.	:2:1219
unit	objeto unidad	unitob	02ADA	¤ Eh	14.	4.599_s
rptr	nombre XLIB	rompointer	02E92	¤ Fh	15.	COERCE
SB, #	entero binario de sistema	bint_	02911	¤ 1Fh	20.	¤ 1219d
rrp	directorio	rrp_	02A96	¤ 2Fh	15.	DIR é END

%%	número real extendido	ereal_	02955	¤ 3Fh	21.	1.219E3
C%%	número complejo extendido	ecmp_	0299D	¤ 4Fh	22.	(5.3E1,9E1)
lnk[]	arreglo enlazado	linkdarray_	02A0A	¤ 5Fh	23.	Linked Array
chr	carácter	char	029FB	¤ 6Fh	24.	@
code	código	code_	02DCC	¤ 7Fh	25.	Code
lib	biblioteca	lib_	02B40	¤ 8Fh	16.	Library 1219: Hack…
bak	objeto de reserva	backup_	02B62	¤ 9Fh	17.	Backup Hack
LD	datos de biblioteca	libdat_	02B88	¤ AFh	26.	Library Data
acptr	puntero extendido	accptr_	02BAA	¤ BFh	27.	Extented Ptr
font	tipo de caracteres	font_	02BCC	¤ CFh	30.	Ft8_0:SYSTEM 8
mfnt	tipo de caracteres pequeños	mfnt_	026FE	¤ DFh	27.	MiniFont 00
			02C10	¤ EFh		
z	número entero	zint_	02614	¤ FFh	28.	1219

El comando TYPE se basa en procesos de verificación de Dispatch, mientras que ITYPE accede a un bloque de memoria en la ROM donde se encuentra un listado completo con los tipos "reales" permitidos, ello lo hace más rápido.

Excepciones:

- ITYPE no diferencia vectores y matrices:
  - El primer ejemplo tiene el prólogo #02686h (=MATRIX)
  - El segundo ejemplo tiene el prólogo #029E8h (=ARRY)
  - El tercer ejemplo tiene el prólogo #029E8h (=ARRY)

Ejemplo:	→	ITYPE	TYPE
[30 90 '7*8']	→	¤ 4h	29.
[30. 90. 56.]	→	¤ 4h	3.
[(3. 4.) (4. 5.)]	→	¤ 4h	4.

- El Dispatch (despacho) "sym" es genérico y puede abarcar varios objetos, por ejemplo nombres globales, objetos simbólicos, etc. Algo similar ocurre con el dispatch real, que puede abarcar a los números enteros. Revise esto comparando por ejemplo el comando EXPAND, éste le dirá que acepta sym, arry, real, cmp; pero usted puede ingresar enteros (1219), reales (1219.), nombres globales ('VARIABLE'), objetos simbólicos (X+9), vectores y complejos.
- Objetos como los punteros Flash no tienen un dispatch asociado, por eso deberán tratarse como objetos "any". Un ejemplo SysRPL sería:

```
::
  CK1&Dispatch
  any
  ::
    DUPTYPEFLASHPTR? ITE
    "Is Flash"
    "Is any object"
  ;
;
```

DTEMP

Usado para obtener en la pila los objetos temporales usados en eventos recientes. DTEMP (en inglés "Dump Temporal objects") ignorará objetos SB (números binarios del sistema "BINTs").
Por ejemplo, DTEMP usado desde la línea de comandos después de un reinicio (#0 SYS) obtendrá en el nivel 2 de la pila una lista conteniendo todos los Lids activados en dicho reinicio (Lid: identificador de biblioteca).

Note que para objetos `Extended Ptr`, necesitará ejecutar `XRCL` para ver su contenido.

Para remover estos objetos de la memoria temporal debe ejecutarse un GARBAGE, que es un vaciado de basura. Intente este nuevo ejemplo y verá los cambios en la cantidad de objetos conseguidos: `#5F42h SYS DTEMP`, donde #5f42h es el equivalente a "`GARBAGE`" desde SysRPL.

CDHD

El comando `CDHD` (en inglés "Current Directory into Hidden Directory"), hará que el directorio oculto sea el directorio actual configurado. Los siguientes son procesos relacionados a la administración del directorio actual:

- Presionar [VAR] para ver las variables (ocultas) en el menú.
- Comandos como `VARS`, `VARS2` obtendrán las variables (ocultas) de forma tradicional.
- Comandos como `UP` ó `UPDIR` terminarán `CDHD`.

El proceso completo para ver el directorio oculto sería: `CDHD 31.1 KEYEVAL`.

GKMAP

ADDR

Con `ADDR` (en inglés "Address") usted obtendrá la dirección en memoria de cualquier objeto colocado en el nivel 1 de la pila. El comando `ADDR` es lo mismo que `→A`.

FOB

SC

D→LIB

L→DIR

LBCRC

En ocasiones, objetos de respaldo y bibliotecas con CRC defectuoso requieren recalcular su checksum para un correcto funcionamiento. (Refiérase a →BAK para mayores detalles relacionados al checksum).
Coloque en el nivel 1 de la pila una biblioteca o un objeto backup y LBCRC devolverá el mismo objeto con CRC recalculado.

GP

Ingrese en el nivel 1 un lid (identificador de biblioteca) y GP (en inglés "Get Port") devolverá el puerto de alojamiento para dicha biblioteca. Por ejemplo 1219 GP devolverá 2 (asumiendo que Hack se encuentre instalada en el puerto 2).
GP es usado en el comando PG para saber de donde borrar la biblioteca; por ejemplo 1219 PG primero reconocería que es desde el puerto 2 que se llama a la biblioteca 1219 y finalmente aplicaría el proceso a :2:1219.
Suponiendo el caso de que la biblioteca 1219 se encuentre alojada en los tres puertos (0, 1 y 2), sólo una de ellas es la operativa y en todos los casos es la alojada en el puerto menor, en este caso el puerto 0. Afortunadamente GP "no" se basa en esta información, GP revisa una entrada llamada "ROMPTAB", que es donde se encuentra el listado de todas las bibliotecas "operativas".

RINFO

El comando RINFO (en inglés "Recall Information") obtendrá información de bibliotecas instaladas. Se requiere para ello un Lid (identificador de biblioteca) en el nivel 1 de pila y este comando devolverá:

- Nivel 6: el Lid ingresado (número entero).
- Nivel 5: el título de la biblioteca (nombre global).
- Nivel 4: el tamaño en Bytes (número entero o real).
- Nivel 3: el hash (número entero binario).
- Nivel 2: el link (número entero binario).
- Nivel 1: el Lid de la siguiente biblioteca (número entero).

Cada uno de los niveles devueltos por RINFO estarán etiquetados para mejor entendimiento (en la imagen se asume el resultado de ejecutar 1219 RINFO); excepto:

- Nivel 5: si el título es nulo se mostrará "NoName".
- Nivel 3: si no hay hash se mostrará "NoHash".
- Nivel 2: si no hay link se mostrará "NoLink".
- Nivel 1: si no hay próximo Lid se mostrará "NoNextLid", ejemplo para el Lid 1792.

Note que RINFO trabaja tanto para bibliotecas del sistema (`1 RINFO`) como también para bibliotecas de usuario (`1219 RINFO`), donde el área de Lids de usuario es de 258 a 1791.

RCFG

Éste es un extractor de los objetos de configuración ($CONFIG) de las bibliotecas. `RCFG` (en inglés "Recall Configuration") requiere un Lid (identificador de biblioteca) e intentará extraer y colocar en la pila el contenido configurativo de la biblioteca.

$CONFIG

Variable interna de las bibliotecas que contiene mayormente un programa que habilita la biblioteca en cada reinicio. Cuando la calculadora se reinicia se verifica (entre otras cosas) y evalúa cada una de las variables $CONFIG de las bibliotecas instaladas, por ello la forma estándar de $CONFIG debe ser: `« 1219 ATTACH »`, donde 1219 es el Lid de la biblioteca Hack. Esto también puede ser expresado en SysRPL como: `:: 1219 XEQSETLIB` (ó `TOSRRP`) `;`... sin embargo, muchos usuarios optan por personalizar esta variable para que al reiniciar aparezcan mensajes de bienvenida u otros.
Si el contenido de $CONFIG de alguna de las bibliotecas es defectuoso o comete algún error, su HP no reiniciará correctamente y se verá forzado a iniciar el sistema a modo de prueba de fallas, impidiendo para esto el reconocimiento de todas las bibliotecas de usuario presionando durante el reinicio ◀.

Como ejemplo veamos el programa configurativo de la biblioteca 256, para ello ejecute: `256 RCFG`:

```
::
  BINT86
  SysITE
  ::
    # 100
    TOSRRP
  ;
  NOP
;
```

... donde se habilitará la biblioteca 256 (mediante `#100 TOSRRP`) sólo si el flag -86 está activado.

RMSG

Éste es un extractor de las tablas de mensaje ($MESSAGE) de las bibliotecas. `RMSG` (en inglés "Recall Messages") requiere un Lid (identificador de biblioteca) e intentará extraer y colocar en la pila un vector conteniendo todos los mensajes usados por la biblioteca.

$MESSAGE

Variable interna de las bibliotecas que contiene un listado de todos los mensajes usados. Estos mensajes son usados la mayoría de veces como mensajes de error, pero no siempre tienen esa función. Cada uno de los mensajes tendrá un identificador según su orden de ubicación, por ejemplo si

su biblioteca es la 999 (#3E7h) y tiene un $MESSAGE como: { "Primer Mensaje" "Segundo Mensaje" }, entonces después de instalada podrá llamar el segundo mensaje ejecutando desde SysRPL:
`:: #3E702 JstGETTHEMSG ;.`

Ejemplo para ver el listado de mensajes de la biblioteca 257: `257 RMSG OB→ EVAL →LIST:`
{ "Invalid File" "Too Many" "Unknown Instruction" "Invalid Field" ...}
... donde la biblioteca 257 es conocida como MASD y contiene el comando ASM.

REXT

USE

USED

USAG

Con `USAG` (en inglés "Usage") puede obtener la siguiente información relacionada a un comando:

- Cantidad de argumentos requeridos.
- Tipo de argumentos requeridos.
- Tipo del comando (funciones o aplicaciones RPN/ALG).
- Informe extra para algunas funciones (como las trigonométricas).

Con el soporte a los tres idiomas pre-definidos en la máquina, es posible visualizar los mensajes en inglés, francés y español. Debe mencionarse que lastimosamente no todos los mensajes son traducidos, ello no depende de USAG, depende de la ROM.

¿Cómo trabaja USAG?

USAG tiene un motor que examina internamente la estructura inicial de un comando que la mayoría de veces (y la forma correcta) debe ser un `CKn` ó `CKn&Dispatch`, donde n es un número entre cero y cinco. Observe la siguiente tabla que muestra algunos ejemplos:

- `CK0`; comandos que no requieren argumentos como: `DEPTH`, `FONT→`, `CLEAR`, `MEM`.
- `CK1`; comandos que requieren un argumento de tipo no especificado como: `NEWOB`, `→STR`, `DUP`.
- `CK2`; comandos que requieren dos argumentos de tipo no especificado como: `SAME`, `SWAP`.

- CK2&Dispatch; comandos que requieren dos argumentos de tipo especificado. Por ejemplo el comando BEEP tiene la siguiente estructura:

```
::
  CK2&Dispatch
  2REAL
  DOBEEP
;
```

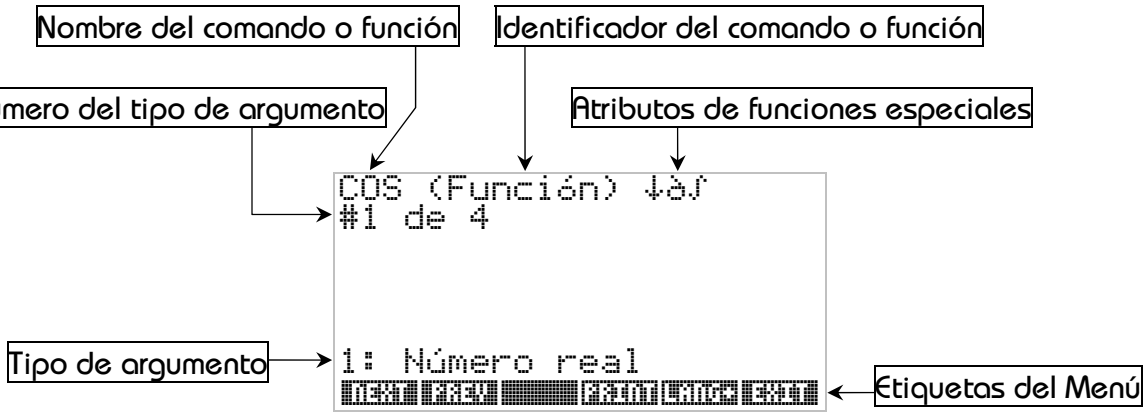
  … entonces para el caso del comando BEEP se requerirá dos argumentos en la pila y ambos deberán ser reales.

Argumentos requeridos

Se requiere sólo un argumento en el nivel 1 de la pila y debe ser una lista conteniendo un comando como se aprecia en la imagen de la derecha. Presione luego [EVAL].

Funcionamiento

Asumiendo que usted colocó en el nivel 1 de la pila { COS }, entonces USAG mostrará una ventana como la siguiente:



Detalles de cada parte de una ventana USAG:

- Nombre del comando o función: nombre del comando ingresado en el nivel 1.
- Identificador del comando o función: mostrará información del tipo de comando (recuerde que los comandos ó son funciones ó son aplicaciones RPN/ALG). Los posibles valores entonces son: (RPN/ALG) o (Función).
- Atributos de funciones especiales: son caracteres que describen los tres atributos adicionales de funciones especiales:
  "↓" le indica que la función tiene una inversa (en el caso de COS es ACOS)
  "à" le indica que la función tiene una derivada (en el caso de COS es -SIN)
  "∫" le indica que la función tiene una antiderivada (en el caso de COS es SIN)
  (esta información está basada en la estructura de comandos de la biblioteca 002 de la serie 48, ya que en la serie 49 la estructura de comandos de la biblioteca 002 es diferente).
- Número del tipo de argumento: le dirá la cantidad total de combinaciones y la actual mostrada. Esto es mostrado siempre en la segunda línea de la pantalla.

- Tipo de Argumento: las próximas líneas de la pantalla mostrarán los tipos de argumento requeridos por el comando y sus correspondientes niveles en la pila (en modo RPN).
- Etiquetas del Menú: las opciones para ver la combinación siguiente (NEXT) o anterior (PREV) funcionarán normalmente hasta llegar al límite (el paso del primero al último o del último al primero ocasionarán un sonido).
  Con PRINT (en 49G) será posible imprimir la pantalla actual a una impresora serial conectada.
  El idioma usado puede ser cambiado presionando LANG. (Inglés, Francés, Español). Finalmente cuando desee terminar presione EXIT.

Características de USAG

- De diseño:
  - USAG trabaja siempre con FONT8. En versiones anteriores USAG no modificaba la fuente actual (sea 6 ó 7), pero se detectaron fuentes que no muestran correctamente los caracteres de atributos como ↓, à, ♪, ó las vocales tildadas usadas en español.
  - USAG deshabilita el flag -72 y al terminar la aplicación restaura su estado original. Esto es vital para no mostrar los mensajes usando MiniFont, que reduciría la estética USAG.
- De funcionalidad:
  - La opción PRINT sólo se encontrará disponible en 49G, debido a temas de velocidad y protocolos de transferencia (Serial/USB) no se consigue compatibilidad con la 50G.
  - Vea algunas equivalencias para las opciones mostradas en el Menú:

    NEXT → ▶
    PREV → ◀
    LANG → +/-
    EXIT → ON

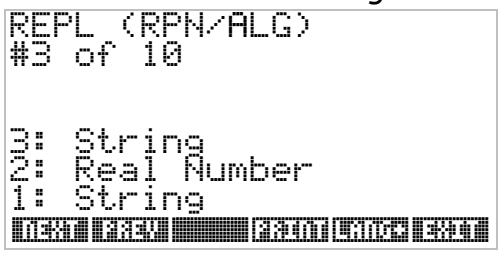
    Por tratarse de una aplicación que también se distribuye por separado del paquete Hack, USAG tiene habilitada la tecla HIST, que mostrará información extra (About).
  - El funcionamiento de USAG depende de una estructura común usada por la mayoría de los comandos HP. Si un comando no tiene estructura común (mencionada antes) USAG no será muy útil, por ejemplo, USAG aplicado al comando →Q sólo muestra "Any", lo que no es de gran ayuda. En la gran mayoría de casos USAG será exacto, pero es posible que usted necesite revisar la Guía de Usuario.

Ejemplos

REPL mostrado en español

REPL mostrado en francés

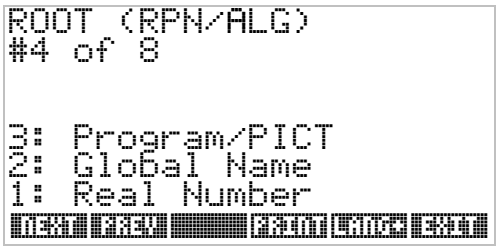
REPL mostrado en inglés



Caso: comandos que no requieren argumentos o de un solo argumento no muestran la interfaz usada para el resto de casos.

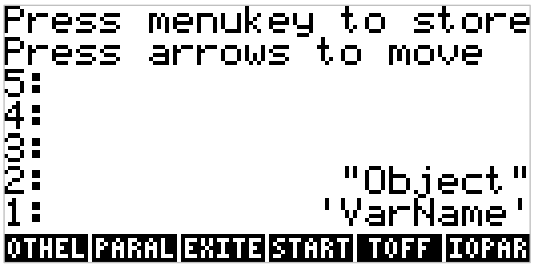
GXOR mostrando "PICT"

Caso: cuando vea "Program/PICT" usted tendrá que deducir cual de los dos es, ya que USAG no pudo encontrar la respuesta exacta.



STOX

Comando que permite almacenar el objeto del nivel 2 de la pila en la variable del nivel 1, en una posición específica en el menú de variables de su directorio actual. La imagen de la derecha muestra la interfaz de usuario de STOX; para validar deberá presionar una de las seis teclas del menú: F#, y el objeto será guardado en la posición especificada.



Las teclas disponibles son:

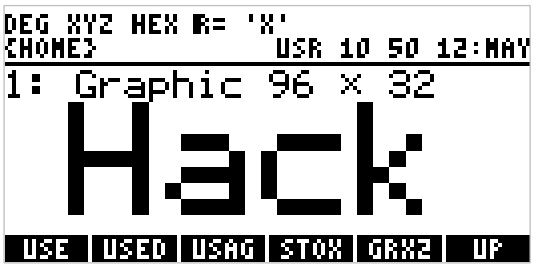
▶	→	Mostrar la siguiente página del menú (también NXT).
◀	→	Mostrar la página previa del menú.
▲	→	Mostrar la primera página del menú (también VAR).
▼	→	Mostrar la última página del menú.
ON	→	Cancelar y salir de STOX.

Recuerde que cada página de menú contiene siempre 6 variables.

Suponiendo que en el ejemplo mostrado antes se presionó F5, entonces el nuevo menú de variables será como se aprecia en la imagen de la derecha, donde 'VarName' se encuentra en la posición de la tecla F5.

STOX puede ser asignado en una tecla de usuario mediante el comando AsnHack.

GRX2



Comando que trabaja con objetos gráficos (Grobs). Su misión es duplicar las dimensiones del Grob del nivel 1 (aplicar Zoom 2x). El gráfico de la derecha se obtuvo al ejecutar el siguiente programa con FONT8 (Zoom 4x = Escala 400%):
`"Hack" 2 →GROB GRX2 GRX2`, que es equivalente a un Zoom 4x del gráfico original obtenido con `→GROB`.
`GRX2` es aplicable también a objetos Grey, conseguidos por ejemplo al ejecutar `GREY 131 64`.

El siguiente es un programa ejemplo que mostrará un reloj grande (gracias a `GRX2`) en la pantalla:

```
DO
 TIME #2F381h SYS 2 →GROB GRX2 →LCD 0.1 WAIT
UNTIL KEY END
DROP
```

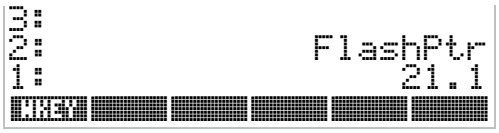
... donde #2F381h es la dirección de la entrada soportada en SysRPL "`TOD>t$`".

UP

Basado en el comando `UPDIR` (en inglés "Up Directory"), ambos hacen lo mismo, suben de nivel la ruta de directorios. Supongamos que la ruta de su directorio actual es HOME/Dir01/Dir02, al ejecutar `UP` o `UPDIR` su directorio actual será HOME/Dir01.
Lo interesante que ocurre con `UP` es que al configurar el nuevo nivel de la ruta de directorios, también configura la página del menú que contiene al directorio abandonado. Es decir, imagine que tiene en HOME 18 variables y que la última es CASDIR, al ejecutar desde la línea de comandos `CASDIR` hará que CASDIR sea su directorio actual, su nueva ruta sería entonces HOME/CASDIR... en este punto ejecute `UP` y verá que el menú es mostrado en la tercera página (ejecute `RCLMENU` para comprobar, debería obtener 2.03)... Pero porqué en la tercera página?... la tercera página se debe a que el menú muestra 6 variables por vez, entonces la 18ava variable se encontrará en la tercera página.

`UP` puede ser asignado en una tecla de usuario mediante el comando `AsnHack`.

WKEY



Comando inspirado en dos de los comandos de la biblioteca Keyman, el comando `WKEY` (en inglés "Wait for Key") espera que se presione una tecla y devolverá el proceso actual contenido en ella, respetando el flag relacionado a teclas de usuario (flag -62).
El gráfico muestra el resultado que devolvió `WKEY` después de presionar la tecla APPS, en donde el nivel 2 es el proceso y el nivel 1 es el código equivalente de la tecla en mención. El código de una tecla tiene dos números que describen la fila y la columna en el teclado, y un valor decimal que se describe en la siguiente tabla:

.1	función principal		
.2	tecla combinada con ↰	.21	simultáneamente con ↰
.3	tecla combinada con ↱	.31	simultáneamente con ↱

.4	tecla combinada con [ALPHA]	.41	simultáneamente con [ALPHA]
.5	tecla combinada con [ALPHA] [↰]	.51	simultáneamente con [ALPHA] [↰]
.6	tecla combinada con [ALPHA] [↱]	.61	simultáneamente con [ALPHA] [↱]

Así; la tecla 21.1 hace referencia a [APPS], mientras que 21.3 hará referencia a [↱] [BEGIN].
El valor entero 21 quiere decir fila 2, columna 1. De igual forma 105 será fila 10, columna 5.
Máquinas con ROM superior a 1.18 reconocen combinaciones simultáneas como la tecla 23.21 que hace referencia a la combinación simultánea de [↰] [i].

En el teclado el máximo de filas es 10 y de columnas es 6, siendo la última tecla la 105.

`WKEY` puede ser asignado en una tecla de usuario mediante el comando `AsnHack`.

→FLASH

Comando necesario para crear punteros flash. La imagen de la derecha muestra el resultado de ejecutar: `2 #80h →FLASH`. Así, este comando requiere dos números; primero un Bid (donde Bid es el identificador de banco de memoria: `2`), seguido del identificador de puntero flash (`#80h`).

```
2:
1:                                  FlashPtr
```

Las combinaciones de los números para obtener el mismo puntero flash pueden ser:

- Dos enteros o reales, por ejemplo: `2 128 →XLIB`. ← `OB→`.
- Dos enteros binarios, por ejemplo: `#2h #80h →XLIB`.
- Cualquiera de los dos anteriores: `#2d 128 →XLIB`, ó `2 #128d →XLIB`.

Ejemplos

Proceso:	→	… desde →FLASH:
[APPS]	→	2 128 →FLASH EVAL
[MODE]	→	4 79 →FLASH EVAL
[MODE] [▒▒▒]	→	2 10 →FLASH EVAL
[MODE] [▒▒▒]	→	4 85 →FLASH EVAL
[MODE] [▒▒▒]	→	4 80 →FLASH EVAL
[CAT] / [↱] [CAT]	→	2 133 →FLASH EVAL
[↰] [FILES]	→	4 103 →FLASH EVAL
[↰] [MTRW]	→	1 745 →FLASH EVAL
[↱] [CHARS]	→	1 846 →FLASH EVAL
[↱] [TIME]	→	2 155 →FLASH EVAL
[↱] [TIME] (ALARMS)	→	2 68 →FLASH EVAL
[↱] [TIME] (SET ALARM)	→	2 69 →FLASH EVAL
[↱] [TIME] (SET T&D)	→	2 70 →FLASH EVAL
[↱] [NUM.SLV] (EQUATION)	→	2 50 →FLASH EVAL

Los ejemplos mostrados son simplemente algunos basados en procesos del teclado.

XMENU

AsnHack

Comando para asignar algunas herramientas de Hack de uso frecuente en teclas de usuario. Para validar la asignación acepte el mensaje mostrado por AsnHack y ello configurará las siguientes teclas de usuario:

Teclado:	Código:	Proceso:
→ F#	10.31 … 16.31	Reemplazo para conseguir el proceso contenido en F#.
← UPDIR	31.2	Reemplaza al comando UPDIR por UP.
STO▸	32.1	Reemplaza el comando STO por XSTO.
← RCL	32.2	Reemplaza el comando RCL por XRCL.
→ CUT	32.31	Asigna WKEY. (ver WKEY)
← DEL	45.2	Reemplaza un CLEAR innecesario por PG.
← DEF	93.21	Asigna STOX. (ver WKEY)
→ →NUM	105.3	Reemplaza el comando →NUM por COERCE.

Para una explicación acerca de códigos de tecla (en combinación simultánea) refiérase al comando WKEY, que le explicará el funcionamiento de teclas como 32.31, 10.31, 16.31.

Note que para que estas teclas asignadas funcionen se requiere el flag -62 activado (-62 SF), es por eso que AsnHack lo activará al realizar la asignación.

ABOUTHACK

Usado simplemente para ver créditos, formas de contacto (correo electrónico), información de versión y fecha de desarrollo de Hack.

La versión de Hack será expresada como 1.0 en caso de ser la versión final, y las versiones previas (versiones beta) serán expresadas como ß5A; donde "ß5" hace referencia a la quinta compilación (Build 5 ó quinta versión Beta), y "A" (valor hexa) hace referencia a la versión en desarrollo (1.0).
En conclusión; ß5A puede ser leído como: Quinta compilación (beta) de la versión 1.0.
Cuando las versiones beta llegan a su fin (después de las pruebas necesarias) surge la versión final.

El formato de la fecha de desarrollo es día.mes.año; por ejemplo 05.06.07 hace referencia al 5 de Junio del 2007.

Programación

Agradecimientos

Hack fue escrito por Gustavo Portales pero existen muchas personas involucradas que contribuyeron de alguna manera en la elaboración de este paquete, entre los destacados se menciona:

- Mika Heiskanen; por escribir la versión para 48, base para esta versión 49G/50G.
- Rick Grevelle; por sus excelentes utilidades para 48, como SC.
- Detlef Mueller; por sus excelentes utilidades para 48, como LIB→.
- Mario Mikocevic; por sus valiosas herramientas personales para 48.
- Andre Schoorl; por mantener y aportar en varias aplicaciones de Mika para 48, entre ellas Hack.
- Hewlett-Packard; por escribir la versión original de USAG para 48.
- Cyrille de Brebisson; por Extable, Debug2, entre muchos más para 49G/50G.
- Jurjen N. Bos; por el incomparable Nosy para 49G/50G.
- Thomas Rast; por las mejoras en Extable y por escribir PortBrowser para 49G.
- Wolfgang Rautenberg; por su utilidad Keyman para 49G/50G.
- Peter Geelhoed; por ser co-autor de Emacs y sus utilidades como LibEx para 49G. Especial agradecimiento por su apoyo en mis inicios como programador.
- Carsten Dominik; por Emacs para 49G/50G.
- Eric Rechlin; por mantener hpcalc.org y por ayudarme en varios temas relacionados.
- William Graves; por Debug4x, basado en Debug2.
- Christoph Giesselink; por el único Emu48.
- Joseph K. Horn; por donarme una flamante 50G.
- Edwin Córdoba; por permitirme participar y ganar mi 50G.

El orden no tiene orden, es decir, el último pudo ser el primero y el primero pudo ser el último en la lista, simplemente así se me ocurrió. Pero el agradecimiento es igual para todos.
Thank you very much!, including <comp.sys.hp48> and <www.adictoshp.org>.