Allen-Bradley

Referencia del lenguaje BASIC

(Número de catálogo 1746-BAS)

Manual de referencia

Información importante para el usuario

El equipo de estado sólido tiene características de operación diferentes a las del equipo electromecánico. La publicación “Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control” (Publicación SGI-1.1) describe algunas diferencias importantes entre equipos de estado sólido y dispositivos electromecánicos cableados. Debido a estas diferencias y debido también a la amplia variedad de usos para los equipos de estado sólido, todas las personas responsables de la aplicación de este equipo deben asegurarse de que cada aplicación sea la correcta.

En ningún caso será Allen-Bradley Company responsable por daños indirectos o como consecuencia del uso o aplicación de este equipo.

Los ejemplos y diagramas mostrados en este manual tienen la única intención de ilustrar el texto. Debido a las muchas variables y requisitos asociados con cualquier instalación particular, Allen-Bradley Company no puede asumir responsabilidad u obligación por el uso real basado en los ejemplos y diagramas mostrados.

Allen-Bradley Company no asume responsabilidad por violación de patente alguna, con respecto al uso de información, circuitos, equipos o programas de software descritos en este manual.

Está prohibida la reproducción total o parcial del contenido de este manual sin el permiso por escrito de Allen-Bradley Company.

En este manual hacemos anotaciones para informarle de consideraciones de seguridad.

ATENCION: Identifica información sobre prácticas o circunstancias que pueden conducir a lesiones personales o la muerte, o a daños materiales o pérdidas económicas.

Las notas de “Atención” le ayudan a:

- identificar un peligro
- evitar un peligro
- reconocer las consecuencias

Importante: Identifica información especialmente importante para la aplicación y entendimiento correctos del producto.

Sírvase tomar nota de que en esta publicación se usa el punto decimal para separar la parte entera de la decimal de todos los números

Resumen de los cambios

La siguiente información resume los cambios en este manual desde la última impresión.

Información nueva

La siguiente tabla lista las secciones que documentan nuevas características e información adicional sobre características existentes, y muestra dónde encontrar esta información nueva.

Para obtener esta información nueva sobre	Vea el capítulo
Conjunto de documentación del módulo BASIC	Prefacio

Resumen de los cambios

Tabla de contenido
Manual de referencia del lenguaje BASIC SLC 550

Comandos BASIC

Llamadas en la línea de comando

Funciones de asignación

Funciones de control

Funciones de soporte de interrupción y control de ejecución

Capítulo 8

Funciones matemáticas y de conversión del backplane

Capítulo 9

Funciones del reloj/calendario

Capítulo 10

Funciones de estado

Capítulo 11

Funciones de salida

Capítulo 12

Funciones de entrada

Capítulo 13

Funciones de configuración

Funciones de cadena

Tabla de conversiones ASCII decimal/hexadecimal/octal

Guía de referencia rápida de comandos, instrucciones y CALLS BASIC

Prefacio

Lea este prefacio para familiarizarse con el resto del manual. Este prefacio incluye los siguientes temas:

- quién debe usar este manual
- el propósito de este manual
- términos y abreviaciones
- convenciones usadas en este manual
- Soporte de Allen–Bradley

Quién debe usar este manual

Use este manual si usted es responsable del diseño, instalación, programación o localización y corrección del fallos de sistemas de control que usan controladores lógicos compactos Allen-Bradley.

Usted debe tener un entendimiento básico de los productos SLC 500™. Debe entender los controladores programables y ser capaz de interpretar las instrucciones de lógica de escalera requeridas para controlar su aplicación. Si no fuera así, comuníquese con su representante local de Allen-Bradley para obtener información sobre los cursos de instrucción disponibles antes de usar este producto.

Propósito de este manual

El manual de Referencia del lenguaje BASIC es un manual que se usa cuando se programa el módulo BASIC. Este manual ha sido diseñado para fines de referencia solamente.

Tabla P.1
Contenido del manual

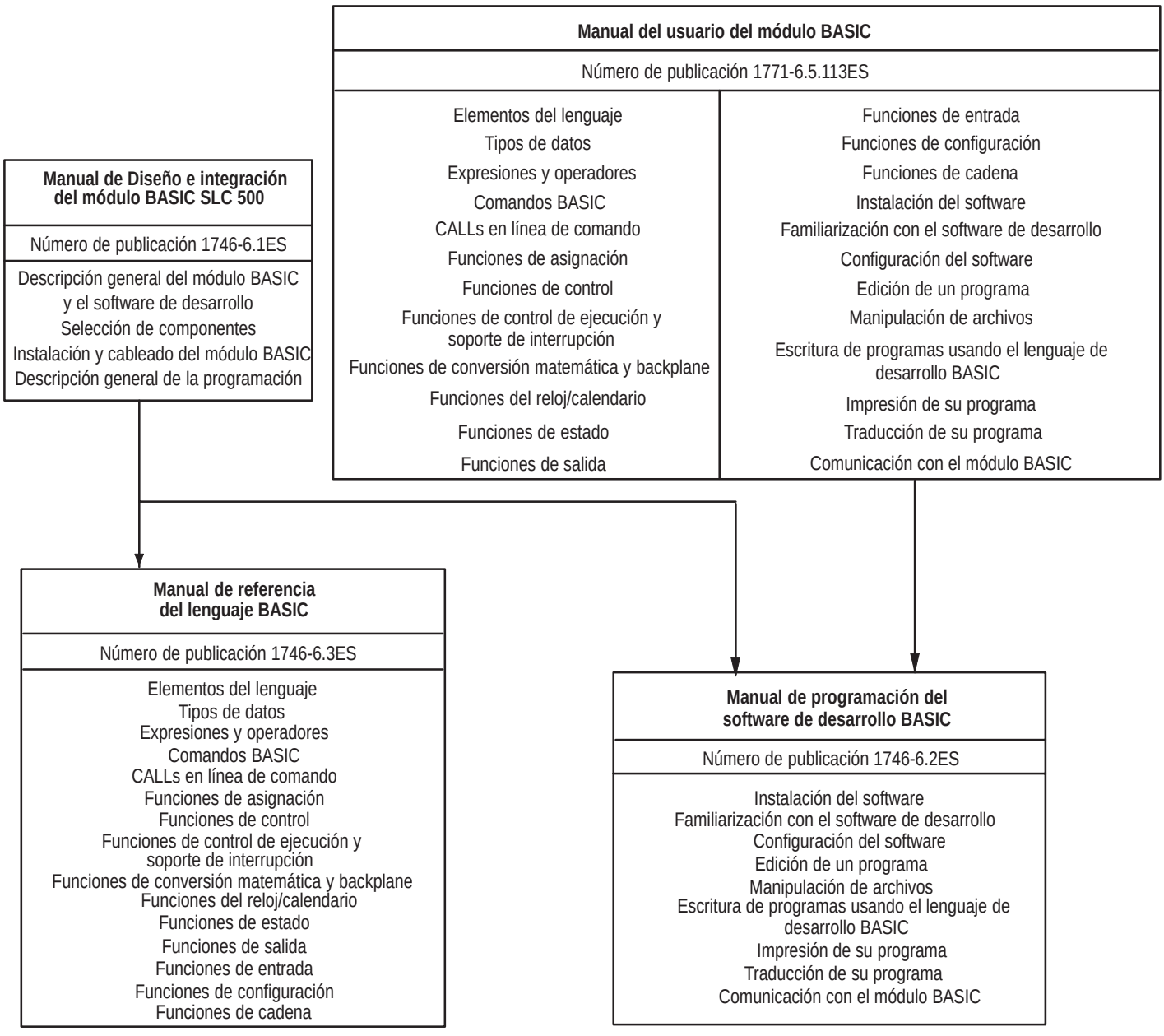
Capítulo	Título	Contenido
	Prefacio	Describe el propósito, antecedentes y alcance de este manual. También especifica la audiencia para la cual se ha diseñado este manual.
1	Elementos del lenguaje	Describe líneas del programa BASIC, números de líneas, instrucciones, comandos, operadores y longitud de líneas.
2	Tipos de datos	Describe e ilustra tipos de datos, nombres y tipos de variables.
3	Expresiones y operadores	Describe e ilustra operadores aritméticos, lógicos, de relación, trigonométricos, funcionales, logarítmicos, de cadena y de funciones especiales.
4	Comandos BASIC	Describe e ilustra comandos BRKPNT, CONT, [CTRL–C], Disabling and Enabling [CTRL–C], [CTRL–S], [CTRL–Q], EDIT, ERASE, IDLE, LIST, LIST@, LIST#, MODE, NEW, NULL, PROG, PROG1, PROG2, RAM, REM, REN, ROM, RROM, RUN, SNGLSTP, VER y XFER y CALLs 18 y 19.
5	Llamadas en la línea de comando	Describe e ilustra CALLs 73, 74, 77, 81, 82, 101, 103, 104, 109, 110 y 111.
6	Funciones de asignación	Describe e ilustra las funciones CLEAR, CLEARI, CLEARS, DATA, DIM, LET y RESTORE.
7	Funciones de control	Describe e ilustra las funciones CLOCK1, CLOCK0, DO–WHILE, DO–UNTIL, END, FOR–TO–(STEP)–NEXT, GOTO, IF–THEN–ELSE, NEXT y ON–GOTO.
8	Funciones de control de ejecución y soporte de interrupción	Describe e ilustra CALLs 16, 17, 20, 21, 26, 38, 70, 71, 72, y las funciones GOSUB, ONERR, ON–GOSUB, ONTIME, PUSH, POP, RETI, RETURN, STOP.
9	Funciones matemáticas y de conversión del backplane	Describe e ilustra CALLs 14, 15, 24 y 25.
10	Funciones del reloj/calendario	Describe e ilustra CALLs 40, 41, 42, 43, 44, 45, 46, 47, 48 y 52.
11	Funciones de estado	Describe e ilustra CALLs 36, 51, 55, 58, 59, 75, 80, 86, 87, 95, 97, 98, 108, 113, 120 y 121.
12	Funciones de salida	Describe e ilustra CALLs 23, 28, 29, 31, 37, 54, 57, 85, 91, 93, 94, 96, 112, 114, 115, 123, y las funciones PRINT, PH0., PH1. y ST@.
13	Funciones de entrada	Describe e ilustra CALLs 22, 27, 29, 35, 53, 56, 84, 90, 92, 117, 118, 122 y las funciones GET, INPL, INPS, INPUT, LD@, y READ.

Capítulo	Título	Contenido
14	Funciones de configuración	Describe e ilustra CALLS 30, 78, 99, 105, 119 y la función MODE.
15	Funciones de cadena	Describe e ilustra CALLs 60, 61, 62, 63, 64, 65, 66, 67, 68 y la función STRING
Apéndice A	Tabla de conversión decimal/hexadecimal/ octal/ASCII	Lista las conversiones decimal/hexadecimal/octal/ASCII.
Apéndice B	Guía de referencia rápida de comandos, instrucciones y CALLS BASIC	Lista los diversos comandos, instrucciones y CALLs necesarios para la programación BASIC.

Conjunto de documentación del módulo BASIC

El conjunto de documentación del módulo BASIC está organizado en manuales, de acuerdo a las tareas que usted debe realizar. Estas tareas incluyen el diseño, la integración y la programación de su módulo BASIC. La Figura P.1 muestra los documentos del conjunto de documentación del módulo BASIC y la información contenida en cada manual.

Figura P.1
Conjunto de documentación del módulo BASIC



Documentación relacionada

Los siguientes documentos contienen información adicional respecto a los productos SLC™ and PLC[Allen–Bradley. Para obtener una copia, comuníquese con la oficina o distribuidor local de Allen–Bradley.

Tabla P.2
Documentos relacionados

Para obtener	Lea este documento	Número de documento
Una descripción general de los productos de la familia SLC 500	Descripción general del sistema SLC 500	1747–2.30ES
Una descripción de cómo instalar y usar su controlador programable SLC 500 modular	Manual de instalación y operación de los controladores programables tipo hardware modular	1747–6.2ES
Una descripción de cómo instalar y usar su controlador programable SLC 500 compacto	Installation and Operation Manual for Fixed Hardware Style Programmable Controllers	1747–NI001
Un manual de procedimientos para personal técnico que usa APS para desarrollar aplicaciones de control	Manual del usuario del Software de Programación Avanzada (APS) Allen–Bradley	1747–6.4ES
Un manual de referencia que contiene información sobre datos del archivo de estado, el conjunto de instrucciones y localización y corrección de fallos de APS	Manual de referencia del Software de Programación Avanzada (APS) Allen–Bradley	1747–6.11ES
Información detallada sobre conexión a tierra y cableado de los controladores programables Allen-Bradley	Pautas de conexión a tierra y cableado del controlador programable de Allen-Bradley	1770–4.1ES
Una descripción sobre las diferencias importantes entre controladores programables de estado sólido y dispositivos electromecánicos cableados	Application Considerations for Solid–State Controls	SGI–1.1
Un artículo sobre tamaños y tipos de cables para conectar a tierra equipos eléctricos	National Electrical Code	Publicado por la Asociación Nacional de Protección contra Incendios de Boston, MA.
Una lista completa de la documentación de Allen-Bradley actual, incluyendo instrucciones para hacer pedidos. También indica si los documentos están disponibles en CD–ROM o en diferentes idiomas	Allen–Bradley Publication Index	SD499
Un glosario de términos y abreviaciones de automatización industrial	Glosario de automatización industrial Allen-Bradley	AG-7.1ES
Una descripción de cómo instalar y usar un módulo que actúa como puente entre redes DH-485 y dispositivos que requieren el protocolo DF1	DH–485/RS–232C Interface Module User's Manual	1747–NU001
Una descripción de cómo integrar su módulo 1746–BAS BASIC Allen–Bradley en su sistema de control SLC	Manual de Diseño e integración del módulo BASIC SLC 500	1746–6.1ES
Una descripción de cómo programar su módulo 1746–BAS BASIC Allen–Bradley usando el software de desarrollo BASIC	Manual de programación del software de desarrollo BASIC	1746–6.2ES
Una descripción de cómo instalar y usar el terminal de mano	Hand–Held Terminal User Manual	1747–NP002

Términos y abreviaciones

Los siguientes términos y abreviaciones son específicos para este producto. Para obtener un listado completo de la terminología de Allen-Bradley, consulte el Glosario de automatización industrial de Allen-Bradley, número de publicación AG–7.1ES.

- **BASIC** — el lenguaje de programación BASIC-52
- **DH-485** — protocolo de comunicación de red
- **dispositivo de la consola** — el dispositivo conectado al puerto de programación del módulo BASIC. Este dispositivo se usa como interface entre el usuario y el programa BASIC.
- **EPROM** — Memoria programable y borrable de sólo lectura
- **EEPROM** — Memoria eléctricamente programable y borrable de sólo lectura
- **Módulo BASIC** — el módulo BASIC SLC 500 (No. de catálogo 1746-BAS)
- **módulo de memoria** — EEPROM o UVPROM del módulo BASIC
- **MTOP** — valor de control del sistema que tiene la última dirección de memoria válida
- **puerto programación** — puerto usado para programar el módulo BASIC. Cualquiera de los puertos PRT1 o DH485 del módulo BASIC puede usarse como puerto de programación.
- **RAM** — Memoria de acceso aleatorio
- **ROM** — Memoria de sólo lectura, se refiere al espacio de memoria del módulo de memoria opcional (EEPROM o UVPROM)
- **RS-232/423** — interface de comunicaciones en serie
- **RS-422** — interface de comunicación diferencial
- **RS-485** — interface de comunicación de red
- **SCADA** — Control de supervisión y adquisición de datos
- **software de desarrollo BASIC** — software de desarrollo BASIC (No. de catálogo 1747-PBASE)
- **SLC 500** — controlador compacto y modular SLC 500
- **UVPROM** — Memoria programable y borrable por rayos ultravioleta, de sólo lectura
- **variable escalar** — una variable con un solo valor

Convenciones usadas en este manual

En este manual se usan las siguientes convenciones::

- Las listas con viñetas como ésta proporcionan información, no pasos de procedimientos.
- Las listas numeradas proporcionan pasos secuenciales o información jerárquica.
- El tipo de letra *cursiva* se usa para enfatizar.
- El texto en **`este tipo de letra`** indica palabras o frases que usted debe escribir.
- Los nombres de las teclas son iguales a los nombres mostrados y aparecen en negrita, mayúsculas entre corchetes (por ejemplo, **`[ENTER]`**).
- **[expr]** - Denota una expresión usada con un comando, operador o instrucción del sistema.
- **[ln num]** - Denota un número de línea usado con un comando, operador o instrucción del sistema.
- **[var]** - Denota una variable usada con un comando, operador o instrucción del sistema.

Soporte de Allen-Bradley

Allen-Bradley ofrece servicios de soporte a nivel mundial, con más de 75 oficinas de ventas/soporte, 512 distribuidores autorizados y 260 integradores de sistemas autorizados ubicados en los Estados Unidos, además de los representantes de Allen-Bradley en los principales países del mundo.

Soporte local para productos

Comuníquese con su representante local de Allen-Bradley para:

- soporte de ventas y pedidos
- instrucción técnica sobre productos
- soporte de garantía
- convenios de servicio de soporte

Ayuda técnica para productos

Si necesita comunicarse con Allen-bradley para obtener ayuda técnica, primero sírvase revisar la información en el capítulo apropiado. Luego llame al representante local de Allen-Bradley.

Sus preguntas o comentarios sobre este manual

Si encuentra algún problema con este manual, por favor comuníquenoslo usando el Informe de problemas de publicación adjunto.

Si tiene alguna sugerencia para que este manual pueda ser de mayor utilidad para usted, por favor comuníquese con nosotros a la siguiente dirección:

Allen-Bradley Company, Inc.
Automation Group
Technical Communication, Dept. J602V, T121
P.O. Box 2086
Milwaukee, WI 53201–2086

Capítulo 1

Elementos del lenguaje

Este capítulo le proporciona los elementos de un programa BASIC. Estos elementos incluyen BASIC:

- números de línea
- instrucciones, comandos y operadores
- longitud de línea

Conjunto de caracteres

Los programas BASIC están compuestos de un grupo de líneas de programa BASIC. Cada línea de programa BASIC está compuesta de un grupo de caracteres ASCII. Consulte el Apéndice A para obtener un listado completo de códigos de caracteres ASCII.

Línea de programa BASIC

Las líneas de programa BASIC constan de un número de línea BASIC e instrucciones y operadores BASIC. Las líneas de programa BASIC tienen el límite de longitud de línea BASIC.

Números de línea BASIC

Nos referimos a los números de línea BASIC así:

[ln num]

Los números de línea BASIC indican el orden en que se almacenan las líneas del programa en la memoria y también se usan como referencias cuando se ramifican y editan. Este número puede ser cualquier número entero del 1 al 65535. Normalmente se comienza numerando los programas BASIC con el número de línea 10 en incrementos de 10. Esto le permite añadir líneas adicionales posteriormente a medida que va trabajando en su programa.

Puesto que la computadora ejecuta las instrucciones en orden numérico, las líneas adicionales no necesitan aparecer en orden consecutivo en la pantalla. Si usted introduce la línea 35 después que la línea 40, la computadora siempre ejecutará la línea 35 después de la línea 30 y antes que la línea 40. Esta técnica evita tener que volver a introducir todo el programa si se olvidó de incluir una línea.

Importante: La primera línea de su programa debe ser un comentario.

Normalmente, los números de línea de un programa comienzan pareciéndose a la primera columna y terminan como la segunda columna siguiente:

```
#1          #2
10          5
20          7
30          10
40          15
50          20
60          30
70          35
80          40
.           .
.           .
.           .
```

Importante: El empleo de un número existente de línea causa que se pierda toda la información a que se refiere el número de línea original. Tenga cuidado cuando introduzca números en el modo de comando, ya que puede borrar accidentalmente algunas líneas del programa. Puede borrar una línea existente volviendo a escribirla sin que le siga información alguna y presionando [**RETURN**].

Instrucciones, comandos y operadores BASIC

Las líneas del programa BASIC constan de un número de línea BASIC e instrucciones y operadores BASIC. Dependiendo de la lógica de su programa, puede haber más de una instrucción en una línea. Si así fuera, cada una debe estar separada por el signo de dos puntos (:).

Longitud de línea BASIC

Una línea del programa BASIC siempre empieza con un número de línea y debe contener por lo menos un carácter, pero no más de 68 caracteres. Una línea del programa finaliza cuando usted presiona la tecla [**RETURN**].

Capítulo 2

Tipos de datos

Este capítulo le proporciona un método de definir o mostrar datos dentro del lenguaje de programación BASIC a través del uso de:

- tipos de datos
- variables

Tipos de datos

Los tipos de datos se dividen en tres secciones: pila de argumentos, tipos de datos elementales numéricos y de cadenas y datos de conversión del backplane.

Pila de argumentos

La pila de argumentos (pila A) almacena todas las constantes que el módulo BASIC está usando actualmente. Las operaciones tales como suma, resta, multiplicación y división siempre operan en los primeros dos números de la pila de argumentos y devuelven el resultado a la pila. La pila de argumentos tiene 203 bytes de longitud. Cada número de punto (coma) flotante colocado en la pila requiere 6 bytes de almacenamiento. La pila de argumentos puede contener hasta 33 números de punto (coma) flotante antes de tener un overflow.

Además, el comando PUSH guarda los datos en la pila de argumentos y el comando POP restaura los datos de la pila. Los comandos PUSH y POP generalmente se asocian con los CALL. PUSH y POP normalmente se asocian con los CALL. PUSH y POP son mecanismos que se usan para transferir información hacia y desde rutinas CALL.

PUSH hace una copia de la variable a la que se le aplica PUSH y luego coloca esa copia en la parte superior de la pila de argumentos. POP toma el valor de la parte superior de la pila de argumentos y lo copia a la variable a la que se le aplica POP.

Tipos de cadenas de datos

Una cadena es un carácter o grupo de caracteres almacenado en la memoria. Generalmente los caracteres almacenados en una cadena forman una palabra o una oración. Las cadenas le permiten usar caracteres en lugar de números. Las cadenas se muestran así:

$([expr])

El módulo BASIC usa variables de cadena de una sola dimensión, $([expr]). La dimensión de una variable de cadena (el valor [expr]) varía de 0 a 254. Esto significa que usted puede definir y manipular 255 cadenas diferentes en el módulo BASIC. Inicialmente no hay memoria asignada para cadenas. La memoria se asigna usando la instrucción STRING. Las cadenas se declaran y manipulan a través del operador $.

Cuando asigne memoria para una cadena, debe considerar los bytes superiores usados por BASIC para manipular cadenas. BASIC usa un byte superior por cadena declarada más un byte superior adicional.

Por ejemplo:

```
String 106,20
```

Asigna espacio para cinco cadenas de 20 bytes (100 bytes) e incluye cinco bytes superiores (1 por cadena) y un byte superior adicional.

En el módulo BASIC usted puede definir cadenas con la instrucción LET, la instrucción INPUT y con el operador ASC.

Ejemplo:

```
>10 STRING 106,20
>20 $(1)="THIS IS A STRING, "
>30 INPUT "WHAT'S YOUR NAME? - ",$(2)
>40 PRINT $(1),$(2)
>50 END

READY
>RUN

WHAT'S YOUR NAME? - FRED
THIS IS A STRING, FRED

READY
>
```

También puede asignar cadenas a cadenas con la instrucción LET.

Ejemplo:

```
LET $(2)=$(1)
```

Resultado: Asigna el valor de cadena en $(1) a la STRING $(2).

Tipos de datos numéricos

Hay dos tipos de datos numéricos diferentes:

- números enteros
- números en punto (coma) flotante

Usted puede introducir y ver números en cuatro formatos: entero, decimal, hexadecimal y exponencial.

Ejemplo:

```
129, 34.98, 0A6EH, 1.23456E+3
```

El módulo BASIC interpreta todos los números como números de punto (coma) flotante, excepto cuando realiza operaciones lógicas. Cuando realiza operaciones lógicas, el módulo BASIC convierte números en punto (coma) flotante a enteros, realiza la operación, luego convierte el resultado otra vez a punto (coma) flotante.

Números enteros

El módulo BASIC opera en enteros de 16 bits sin signo en un rango de 0 a 65535 ó 0FFFFH. Usted puede introducir todos los enteros en formato decimal o hexadecimal. Un número hexadecimal se indica colocando el carácter H después del número (ejemplo: 170H). Si el número hexadecimal empieza con A - F, entonces éste debe ser precedido por un cero (por ejemplo, usted debe introducir A567H como 0A567H.) Cuando un operador (tal como .AND. requiere un entero, el módulo BASIC trunca la fracción del número para que tenga el formato de entero. Los enteros se muestran así:

[integer]

Importante: El procesador SLC 500 opera en enteros de 16 bits con signo en un rango de -32768 a 32767. Si un valor entero mayor que 32767 se pasa al procesador desde el módulo BASIC, ese valor es interpretado como negativo por el procesador.

Números en punto (coma) flotante

En el módulo BASIC, todos los números se almacenan como números en punto (coma) flotante. Los números en punto (coma) flotante son números en los que el punto (coma) decimal “flota” dependiendo de los dígitos significativos de un número específico. El procesador considera la ubicación del punto (coma) decimal. Esto permite que el procesador almacene sólo los dígitos significativos de un valor, ahorrando así espacio de memoria.

Se puede representar el rango siguiente de números en el módulo BASIC:

+1E -127 to +.99999999 +127

Hay ocho dígitos significativos. Los números se redondean internamente para que tengan esta precisión.

Datos de conversión del backplane

El módulo BASIC se comunica con el procesador local a través del backplane de E/S SLC 500. Todos los datos comunicados hacia y desde el SLC 500 están en formato SLC 500. Los formatos SLC 500 son:

- Entero con signo de 16 bits (-32768 a 32767)
- Binario de 16 bits (0000000000000000 a 1111111111111111)

Importante: Todo entero mayor que 32767 es interpretado como número negativo por el procesador SLC 500.

Variables

Las variables que incluyen una expresión unidimensional [expr] son variables dimensionadas o en serie. Las variables que contienen una letra o una letra y un número son variables escalares. Todas las variables que se introducen con letra minúscula cambian a mayúscula. Las variables se muestran así:

[var]

El módulo BASIC asigna las variables de manera estática, lo cual significa que la primera vez que se usa una variable, el módulo BASIC asigna una porción de la memoria (8 bytes) específicamente para esa variable. Esta memoria no se puede desasignar de variable a variable. Esto significa que si ejecuta una instrucción (ejemplo: >10 Q = 3), usted no puede indicar al módulo BASIC que la variable Q ya no existe para liberar los 8 bytes de memoria que pertenecen a Q. Se puede borrar la memoria asignada a variables ejecutando una instrucción CLEAR. La instrucción CLEAR libera toda la memoria asignada a variables. Las variables se pueden reservar y volver a usar para ahorrar memoria.

Importante: El módulo BASIC requiere menos tiempo para encontrar una variable escalar porque no hay que evaluar ninguna expresión. Para ejecutar un programa lo más rápido posible, use variables unidimensionales solamente cuando sea necesario. Use variables escalares para variables intermedias y asigne el resultado final a una variable dimensionada. Además, coloque primero las variables usadas más frecuentemente. Las variables que se definen primero requieren menos tiempo para ser localizadas.

Nombres de variables

Las variables pueden representar valores numéricos o cadenas. Los nombres de variables sólo pueden tener ocho caracteres. El módulo BASIC compara el primero, el último y el número de caracteres en un nombre de variable con el primero, el último y el número de caracteres en otros nombres de variables para determinar si es un nombre de variable único. Los caracteres permitidos en un nombre de variable son letras, números y el punto decimal. También se permiten caracteres de declaración de tipo especial.

Una variable puede ser una letra (por ejemplo **`A, X, o I`**) seguida por:

- una expresión unidimensional, (ejemplo: **`J(4), G(A+6), I(10*SIN(X))`**)
- un número seguido por una expresión unidimensional (ejemplo: **`A1(8), P7(10*SIN(X)), W8(A + C`**)
- un número (0 a 9) o letra (ejemplo: **`AA, AC, XX, A1, X3, G8`**) excepto las combinaciones siguientes: **`CR, DO, IE, IF, IP, ON, PI, SP, TO, UI, UO`**

Importante: Las palabras reservadas (palabras ya usadas en funciones o instrucciones BASIC) no pueden usarse como nombres de variables.

Tipos de variables

Los caracteres de declaración de tipo indican lo que representa una variable. Se reconoce el carácter de declaración de tipo siguiente:

Carácter	Tipo de variable
$	Variable de cadena

El único otro tipo de variable válido es una variable en punto (coma) flotante. Las variables en punto (coma) flotante no requieren una declaración de tipo.

Capítulo 3

Expresiones y operadores

Este capítulo describe e ilustra cómo usted manipula y/o evalúa expresiones e instrucciones dentro del programa BASIC o la línea de comando. La Tabla 3.A lista los nemónicos correspondientes.

Tabla 3.A
Guía de referencia del capítulo

Si usted necesita	Use este mnemónico	Página
Valor absoluto	ABS( )	3-11
Devolver el valor entero del carácter ASCII.	ASC( )	3-13
Devolver el arco tangente del argumento.	ATN( )	3-10
Recuperar datos de la dirección de memoria especificada.	CBY( )	3-18
Contar el carácter ASCII convertido a valor.	CHR( )	3-16
Devolver el coseno del argumento.	COS( )	3-10
Recuperar o asignar datos hacia o desde la memoria de datos internos del módulo BASIC.	DBY( )	3-19
Hacer una prueba para determinar si el búfer de entradas está vacío.	EOF	3-17
“e” (2.7182818) A LA X	EXP( )	3-13
Hacer una prueba para determinar el número de bytes libres de la memoria RAM.	FREE	3-17
Obtener un entero	INT( )	3-11
Leer el número de bytes de memoria en el programa actualmente seleccionado.	LEN	3-18
Obtener un logaritmo natural	LOG( )	3-13
Leer la última dirección de memoria válida.	MTOP	3-18
Obtener el complemento a uno	NOT( )	3-11
Obtener PI-3.1415926	PI	3-12
Obtener un número aleatorio	RND	3-12
Obtener un signo	SGN	3-12
Devolver el seno del argumento	SIN( )	3-9
Obtener la raíz cuadrada	SQR( )	3-12

Si usted necesita	Use este mnemónico	Página
Devolver la tangente del argumento.	TAN( )	3-10
Recuperar y/o asignar el valor del reloj autónomo.	TIME	3-20
Recuperar y/o asignar datos hacia o desde la memoria de datos externos del módulo BASIC.	XBY( )	3-19
Sumar	+	3-5
Dividir	/	3-5
Elevar a la potencia	**	3-5
Multiplicar	*	3-5
Restar	–	3-6
Introducir un Y lógico	.AND.	3-8
Introducir un O lógico	.OR.	3-8
Introducir un O exclusivo lógico	.XOR.	3-8
Comunicación directa al puerto PRT1.	@	3-17
Comunicación directa al puerto PRT2.	#	3-17

Expresiones y operadores

Una expresión es una expresión matemática lógica que incluye operadores, constantes y variables. Hay ocho tipos de operadores que pueden actuar sobre una expresión:

- aritmético
- lógico
- de relación
- trigonométrico
- funcional
- logarítmico
- de cadena
- de función especial

Expresiones

Las expresiones son simples o complejas.

Ejemplo:

Expresión simple: `12*EXP(A)/100,H(1) + 55,`

o bien

Expresión compleja: `(SIN(A)*SIN(A)+COS(A)* COS(A)/2)`

Una variable autónoma [var] o constante [const] se considera también una expresión. Las expresiones se muestrn así:

[expr]

Operadores

Un operador ejecuta una operación definida sobre variables o constantes. Los operadores requieren uno o dos operandos. Los operadores típicos de dos operandos incluyen: ADD(+), SUBTRACT(-), MULTIPLY(*) y DIVIDE(/). Los operadores que requieren sólo un operando se conocen como operadores de un solo operando. Los operadores típicos de un solo operando son SIN, COS, y ABS.

Jerarquía de operadores

La jerarquía de operadores es el orden en que se ejecutan las operaciones en una expresión. Usted puede escribir expresiones complejas usando sólo un número pequeño de paréntesis. Para ilustrar la jerarquía de operadores, examine la siguiente ecuación:

4+3*2 = ?

En esta ecuación la multiplicación tiene prioridad sobre la suma. Por lo tanto, multiplique (3*2) y luego sume 4.

4+3*2 = 10

Cuando una expresión se escanea de izquierda a derecha, no se ejecuta ninguna operación hasta que se encuentre un operador de menor o igual prioridad. En el ejemplo, usted no puede ejecutar la suma hasta que la operación de multiplicación se complete porque la multiplicación tiene una prioridad más alta. Use paréntesis si tiene dudas acerca del orden de prioridades o para mejorar la legibilidad del programa. La prioridad de operadores de mayor a menor en el módulo BASIC es:

1. Operadores que usan paréntesis ( )

2. Exponenciación (**)

3. Negacion (-)

4. Multiplicación (*) y división (/)

5. Suma (+) y resta (-)

6. Expresiones de relación (=, <>, >, >=, <, <=).

7. Y lógico (.AND.)

8. O lógico (.OR.)

9. O exclusivo lógico (.XOR.)

Operadores aritméticos

El módulo BASIC tiene un juego completo de operadores aritméticos que están divididos en dos grupos: operadores de dos operandos y operadores de un operando.

La forma general de todas las instrucciones de dos operandos es:

(expr) OP (expr), donde OP es uno de los operadores aritméticos siguientes:

Suma (+)

Use el operador de suma para sumar la primera expresión a la segunda.

Ejemplo: `>PRINT 3+2`

Resultado: `5`

División (/)

Use el operador de división para dividir la primera expresión entre la segunda expresión.

Ejemplo: `>PRINT 100/5`

Resultado: `20`

Exponenciación (**)

Use el operador de exponenciación para elevar la primera expresión a la potencia de la segunda expresión. La potencia máxima a la que se puede elevar un número es 255.

Ejemplo: `>PRINT 2**3`

Resultado: `8`

Multiplicación (*)

Use el operador de multiplicación para multiplicar la primera expresión por la segunda.

Ejemplo: `>PRINT 3*3`

Resultado: `9`

Resta (–)

Use el operador de resta para restar la segunda expresión de la primera expresión.

Ejemplo: `>PRINT 9-6`

Resultado: `3`

Negación (–)

Use el operador de negación para cambiar una expresión positiva a una negativa.

Ejemplo: `>PRINT -(9+4)`

Resultado: `-13`

Overflow y división entre cero

Si se produce un overflow, underflow o división entre cero durante la evaluación de una expresión, el módulo BASIC genera mensajes de error y regresa al modo de comando. Consulte la operación ONERR en el capítulo 8 para obtener más información sobre cómo capturar estos errores.

El resultado máximo que se permite para cualquier cálculo es 0.99999999 E+127. Si se excede este número, el módulo BASIC genera el mensaje de error **`ERROR: ARITH. OVERFLOW`** y regresa al modo de comando.

El resultado más pequeño que se permite para cualquier cálculo es 0.99999999 E–128. Si se excede este número, el módulo BASIC genera el mensaje de error **`ERROR: ARITH. UNDERFLOW`** y regresa al modo de comando.

Si se intenta dividir cualquier número entre cero, el módulo BASIC genera el mensaje de error **`ERROR: DIVIDE BY ZERO`** y regresa al modo de comando.

```
>10 PRINT 9/0
>20 PRINT “PROGRAM SHOULD NOT GET HERE.”

READY
>RUN

ERROR: DIVIDE BY ZERO – IN LINE  10

10     PRINT 9/0
----------------X
READY
>
```

```
>10 PRINT 9.9E126*(2)
>

READY
>RUN

ERROR: ARITH. OVERFLOW – IN LINE  10

10     PRINT 9.9E126*(2)
------------------------X
READY
>
```

Operadores lógicos

El módulo BASIC tiene un juego completo de operadores lógicos que se dividen en dos grupos: operadores de dos operandos y operadores de un operando.

La forma general de todas las instrucciones de dos operandos es:

(expr) OP (expr), donde OP es uno de los operadores lógicos siguientes:

Estos operadores realizan operaciones lógicas BIT-WISE en los números entre 0 (0000H) y 65535 (0FFFFH) inclusive. Si el argumento está fuera de este rango, entonces el módulo BASIC genera un mensaje de error **ERROR: BAD ARGUMENT** y regresa al modo de comando. Todos los lugares decimales se truncan, no se redondean. Use la siguiente tabla para manipulaciones de bits en valores de 16 bits.

Tabla 3.B
Manipulación de bits en valores de 16 bits

X	Y	X .AND.Y	X .OR.Y	X .XOR.Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

.AND.

Use el operador lógico .AND. para intersectar lógicamente expresiones.

Ejemplo: `>PRINT 3.AND.2`

Resultado: `2`

.OR.

Use el operador lógico .OR. para unir expresiones lógicamente mediante el símbolo OR.

Ejemplo: `>PRINT 1.OR.4`

Resultado: `5`

.XOR.

Use el operador lógico .XOR. para hacer disyunciones exclusivas lógicas de expresiones.

Ejemplo: `>PRINT 7.XOR.6`

Resultado: `1`

Operadores de relación

Las expresiones de relación incluyen a los operadores =, < >, >, >=, <, and <=. En el módulo BASIC, las operaciones de relación generalmente se usan para probar una condición. Los operadores de relación del módulo BASIC devuelven un resultado de 65535 (0FFFFH) si la expresión de relación es falsa. El resultado regresa a la pila de argumentos. Debido a esto, es posible ver el resultado de una expresión de relación. Las expresiones de relación se muestran así:

[rel expr]

Ejemplos:

```
>PRINT 1=0
0
>PRINT 1>0
65535
>PRINT A<>A
0
>PRINT A=A
65535
```

Se pueden encadenar expresiones de relación con los operadores lógicos .AND., .OR., y .XOR. Esto hace posible probar una condición compleja con UNA instrucción.

Ejemplo:

```
>10 IF (A>E).AND.(A>C).OR.(A>D)THEN...
```

Además, puede usar el operador NOT([expr]).

Ejemplo:

```
>10 IF NOT(A>E).AND.(A>C)THEN...
```

Encadenando expresiones de relación con operadores lógicos, usted puede probar condiciones particulares con una instrucción.

Importante: Cuando use operadores lógicos para vincular expresiones de relación, debe asegurase de que las operaciones se realicen en la secuencia correcta. Si tiene dudas, use paréntesis.

Operadores trigonométricos

El módulo BASIC tiene un juego completo de operadores trigonométricos. Estos operadores son operadores de un operando.

SIN([expr])

Use el operador SIN para obtener el seno del argumento. El argumento se expresa en radianes. Los cálculos se aproximan hasta 7 dígitos significativos. El argumento debe estar entre ±200000.

Ejemplos:

```
>PRINT SIN(PI/4)     >PRINT SIN(0)
.7071067             0
```

COS([expr])

Use el operador COS para obtener el coseno del argumento. El argumento se expresa en radianes. Los cálculos se aproximan hasta 7 dígitos significativos. El argumento debe estar entre ±200000.

Ejemplos:

```
>PRINT COS(PI/4)     >PRINT COS(0)
.7071067             1
```

TAN([expr])

Use el operador TAN para obtener la tangente del argumento. El argumento se expresa en radianes. El argumento debe estar entre ±200000.

Ejemplos:

```
>PRINT TAN(PI/4)     >PRINT TAN(0)
1                    0
```

ATN([expr])

Use el operador ATN para obtener el arco tangente del argumento. El argumento se expresa en radianes. Los cálculos se ejecutan con 7 dígitos significativos. El operador ATN devuelve un resultado entre -PI/2 (3.1415926/2) y PI/2.

Ejemplos:

```
>PRINT ATN(PI)       >PRINT ATN(1)
1.2626272            .78539804
```

Comentarios sobre funciones trigonométricas

Los operadores SIN, COS y TAN usan una serie de Taylor para calcular la función. Estos operadores primero reducen el argumento a un valor entre 0 y PI/2. Esta reducción se realiza mediante la siguiente ecuación:

```
reduced argument=(user arg/PI - INT(user arg/PI) *PI
```

El argumento reducido de la ecuación anterior está entre 0 y PI. El argumento reducido luego se prueba para ver si es mayor que **PI/2**. Si lo es, entonces se resta de **PI** para obtener el valor final. Si no lo es, entonces el argumento reducido es el valor final.

Aunque este método de reducción de ángulo proporciona un medio simple y económico de generar los argumentos apropiados para una serie de Taylor, hay un problema de precisión asociado con esta técnica. El problema de precisión se nota cuando el argumento del usuario es grande (ejemplo **`greater than 1000`**). Esto se debe a que los dígitos significativos en la parte decimal (fracción) del argumento reducido se pierden en la expresión **`(user arg/PI - INT (user arg/PI))`**. Como regla general, mantenga los argumentos de las funciones trigonométricas tan pequeños como sea posible.

Operadores funcionales

El módulo BASIC tiene un juego completo de operadores funcionales. Estos operadores son operadores de un solo operando.

ABS([expr])

Use el operador ABS para obtener el valor absoluto de la expresión.

Ejemplos:

```
>PRINT ABS(5)        >PRINT ABS(-5)
5                    5
```

NOT([expr])

Use el operador NOT para obtener un complemento a uno de 16 bits de la expresión. La expresión debe ser un entero válido (ejemplo: **`between 0 and 65535 (0FFFFH) inclusive`**). Los números no enteros se truncan, no se redondean.

Ejemplos:

```
>PRINT NOT(65000)    >PRINT NOT(0)
535                  65535
```

INT([expr])

Use el operador INT para obtener la parte entera de la expresión.

Ejemplos:

```
>PRINT INT(3.7)      >PRINT INT(100.876)
3                    100
```

PI

PI es una constante almacenada. En el módulo BASIC PI se almacena como 3.1415926.

SGN([expr])

Use el operador SGN para obtener un valor de +1 si el argumento es mayor que cero, cero si el argumento es igual a cero y -1 si el argumento es menor que cero.

Ejemplos:

```
>PRINT SGN(52)   >PRINT SGN(0)   >PRINT SGN(-8)
1                0               -1
```

SQR([expr])

Use el operador SQR para obtener la raíz cuadrada del argumento. El argumento no puede ser menor que cero.

Ejemplos:

```
>PRINT SQR(9)   >PRINT SQR(45)   >PRINT SQR(100)
3               6.7082035        10
```

RND

Use el operador RND para obtener un número pseudo aleatorio entre 0 y 1 inclusive. El operador RND usa un valor de arranque binario de 16 bits y genera 65536 números pseudo aleatorios antes de repetir la secuencia. Los números generados está específicamente entre 0/65535 y 65535/65535 inclusive.

Importante: A diferencia de la mayoría de lenguajes BASIC, el operador RND en el módulo BASIC no requiere un argumento ni un argumento ficticio. Si se coloca un argumento después del operador RND, se produce un error de sintaxis.

Ejemplos:

```
>PRINT RND
.26771954
```

Operadores logarítmicos

El módulo BASIC tiene un juego completo de operadores logarítmicos. Estos operadores son operadores de un solo operando.

LOG([expr])

Use el operador LOG para obtener el logaritmo natural del argumento. El argumento debe ser mayor que 0. Este cálculo se ejecuta con 7 dígitos significativos.

Ejemplos:

```
>PRINT LOG(12)          >PRINT LOG(EXP(1))
2.484906                1
```

Si se necesitan logaritmos de base 10, puede usarse la siguiente expresión:

$$\log_{10}(x)=\log(x)/\log(10)$$

El logaritmo es natural.

EXP([expr])

Use el operador EXP para elevar el número (2.7182818) a la potencia del argumento.

Ejemplos:

```
>PRINT EXP(1)           >PRINT EXP(LOG(2))
2.7182818               2
```

Operadores de cadena

Dos operadores en el módulo BASIC pueden manipular CADENAS. Estos operadores son ASC( ) y CHR( ).

ASC([expr])

Use el operador ASC para obtener el valor entero del carácter ASCII entre paréntesis.

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PRINT ASC(a)
>20 PRINT ASC(A)


READY
>RUN

 65
 65

READY
>
```

La representación decimal del carácter ASCII A es 65. La representación decimal del carácter ASCII a es 97. Sin embargo, el módulo BASIC pone en mayúsculas todos los caracteres ASCII que no están entre comillas. De manera similar, no deben usarse los caracteres especiales ASCII cuyo valor decimal es mayor que 127.

Además, se pueden evaluar caracteres individuales en una cadena ASCII definida con el operador ASC.

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1) =“THIS IS A STRING”
>20 PRINT $(1)
>30 PRINT ASC($(1),1)
>40 END


READY
>RUN

THIS IS A STRING
84

READY
>
```

Cuando use el operador ASC tal como se muestra anteriormente, el $([expr]) denota a qué cadena se tiene acceso. La expresión después de la coma selecciona un carácter individual en la cadena. En el ejemplo anterior se selecciona el primer carácter en la cadena. La representación decimal para el carácter ASCII T es 84. La posición del carácter 0 en la cadena no es válida.

Ejemplo:

```
>NEW

>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1)="ABCDEFGHIKJL"
>20 FOR X = 1 TO 12
>30 PRINT ASC($(1),X),
>40 NEXT X
>50 END


READY
>RUN

65  66  67  68  69  70  71  72  73  75  74  76
READY
>
```

Los números impresos en el ejemplo anterior representan los caracteres ASCII de la A a la L.

Puede usar también el operador ASC para cambiar caracteres individuales en una cadena definida.

En general, el operador ASC le permite manipular caracteres individuales en una cadena. Un programa simple puede determinar si dos cadenas son idénticas.

Ejemplo:

```
>NEW

>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1) = "ABCDEFGHIJKL"
>20 PRINT $(1)
>30 ASC($(1),1) = 75 : REM DECIMAL EQUIVALENT OF K
>40 PRINT $(1)
>50 ASC($(1),2) = ASC($(1),3)
>60 PRINT $(1)

READY
>RUN

ABCDEFGHIJKL
KBCDEFGHIJKL
KCCDEFGHIJKL

READY
>
```

CHR([expr])

Use el operador CHR para convertir una expresión numérica a un carácter ASCII.

Ejemplo:

```
>PRINT CHR(65)
A
```

Al igual que el operador ASC, el operador CHR también selecciona caracteres individuales en una cadena ASCII definida.

Ejemplo:

```
>NEW

>1  REM EXAMPLE PROGRAM
>5  STRING 1000,40
>10 $(1) = “The BASIC Module”
>20 FOR I = 1 TO 16 : PRINT CHR($(1),I),: NEXT I


READY
>RUN

The BASIC Module
READY
>
```

En el ejemplo anterior, las expresiones contenidas dentro de los paréntesis que siguen al operador CHR tienen el mismo significado que las expresiones en el operador ASC.

A diferencia del operador ASC, usted *no puede* asignar un valor al operador CHR. Una instrucción tal como CHR ($(1),1)= H es NO VALIDA y genera un mensaje **`ERROR: BAD SYNTAX`**, haciendo que el módulo BASIC entre en el modo de comando. Use el operador ASC para cambiar un valor en una cadena, o use la rutina CALL de soporte - reemplazar cadena en una cadena.

Importante: Use la función CHR sólo en una instrucción de imprimir. El operador CHR no puede usarse en una instrucción de DATOS.

Operadores de funciones especiales

El módulo BASIC tiene un juego completo de operadores de funciones especiales. Estos operadores manipulan el hardware de E/S y las direcciones de memoria del módulo BASIC.

y @

Use los operadores # y @ para dirigir las comunicaciones. La comunicación se realiza a través del puerto PRT1 cuando se programa el operador @ y a través del puerto PRT2 cuando se programa el operador #. La ausencia de cualquiera de los operadores # o @ indica que la comunicación debe realizarse a través de un puerto de programación (puerto PRT1 o puerto DH485).

 # Ejemplo:

```
>10 A = GET#
```

Resultado: El siguiente carácter en el búfer de entradas PRT2 está asignado a la variable A.

@ Ejemplo:

```
>10 A = GET@
```

Resultado: El siguiente carácter en el búfer de entradas PRT1 está asignado a la variable A.

EOF

Use el operador EOF para probar un búfer de entradas vacío antes de ejecutar una función o instrucción de entrada. Esto evita que las instrucciones de entrada esperen indefinidamente en búfers de entradas vacíos. Use la instrucción EOF# para probar un búfer de entradas vacío para el puerto PRT2. Use la instrucción EOF@ para probar un búfer de entradas vacío para el puerto PRT1.

Ejemplo:

```
>10 REM EXAMPLE PROGRAM
>20 IF (NOT(EOF)) THEN A=GET
>30 REM IF BUFFER NOT EMPTY, READ SINGLE CHARACTER
```

FREE

Use el valor de control del sistema FREE para que le informe cuántos bytes de memoria RAM están disponibles para el usuario. Cuando el programa actualmente seleccionado está en la memoria RAM, la siguiente relación es verdadera:

```
FREE = MTOP - LEN - 511
```

LEN

Use el valor de control del sistema LEN para que le indique cuántos bytes de memoria ocupa el programa actualmente seleccionado. Esta es la longitud del programa y no incluye el tamaño de la memoria de cadenas ni el uso de memoria de variables y conjuntos. No se puede asignar un valor a LEN, sólo se puede leer. Un programa NULL (ejemplo **`no program`**) da un LEN de 1. El 1 representa el carácter final del archivo de programa..

Importante: El módulo BASIC no requiere ningún argumento ficticio para los valores de control del sistema.

MTOP

Use el operador MTOP para recuperar la última dirección de memoria válida en RAM que está disponible para el módulo BASIC. Después que se restablece, el módulo BASIC calcula la memoria externa y asigna la última dirección de memoria válida al valor de control del sistema MTOP. El módulo no usa una RAM externa más allá del valor asignado a MTOP. Si este valor no ha sido cambiado por CALL 77, entonces la última dirección BASIC válida es 5FFFH (24575).

Ejemplos:

```
>PRINT MTOP          or        PH0.MTOP
24575                          5FFFH
```

CBY([expr])

Use el operador CBY para recuperar los datos del programa o la posición de código de la dirección de memoria del módulo BASIC. No se puede asignar un valor a CBY, sólo se puede leer. El argumento pra el operador CBY debe ser un entero válido entre 0 y 65535 (0FFFFH). Si no es un entero válido, se produce un error de argumento.

Importante: El uso incorrecto de este operador puede causar un mal funcionamiento del módulo BASIC.

Ejemplo:

```
A = CBY(1000)
```

Resultado: El valor en el programa o la ubicación del código de la dirección de memoria 1000 es asignado a la variable A.

DBY([expr])

Use el operador DBY para recuperar o asignar datos hacia o desde la memoria interna de datos del módulo BASIC. El valor y el argumento en el operador DBY deben estar entre 0 y 255 inclusive. Esto se debe a que sólo hay 256 posiciones de memoria interna en el módulo BASIC y un byte sólo puede representar una cantidad entre 0 y 255 inclusive.

Importante: El uso incorrecto de este operador puede causar un mal funcionamiento del módulo BASIC.

Ejemplo:

```
A = DBY(B)
```

Resultado: El valor en la posición B de la memoria interna es asignado a la variable A. B debe estar entre 0 y 255.

```
DBY(250) = CBY(1000)
```

Resultado: El valor en el programa o la ubicación del código de la dirección de memoria 1000 es asignado a la posición 250 de la memoria interna.

XBY([expr])

Use el operador XBY para recuperar o asignar datos hacia o desde la memoria externa de datos del módulo BASIC. El argumento para el operador XBY debe ser un entero válido entre 0 y 65535 (0FFFFH). El valor asignado al operador XBY debe estar entre 0 y 255 inclusive. Si no fuera así, se producirá un error de argumento.

Importante: El uso incorrecto de este operador puede causar un mal funcionamiento del módulo BASIC.

Ejemplo:

```
A = XBY(0F000H)
```

Resultado: El valor en la posición de memoria externa 0F00H se asigna a la variable A.

```
XBY(4000H) = DBY(100)
```

Resultado: El valor en la posición de memoria interna 100 se asigna a la posición de memoria externa 4000H.

TIME (TIEMPO)

Use el operador TIME para recuperar o asignar un valor al reloj autónomo residente en el módulo BASIC. Después que se restablece, el tiempo es igual a 0. La instrucción CLOCK1 habilita el reloj autónomo. Cuando el reloj autónomo está habilitado, el operador de función especial TIME incrementa una vez cada 5 milisegundos. Las unidades de tiempo están en segundos.

Cuando se asigna un valor a TIME con una instrucción LET (ejemplo: **`TIME=100`**), sólo la parte entera de TIME cambia.

Ejemplo:

```
>CLOCK1              (enable FREE RUNNING CLOCK)

>CLOCK0              (disable FREE RUNNING CLOCK)

>PRINT TIME          (display TIME)
3.315

>TIME = 0            (set TIME = 0)

>PRINT TIME          (display TIME)
.315                 (only the integer is changed)
```

Se puede cambiar la parte fraccionaria de TIME manipulando el contenido de la posición de memoria interna 71 (47H). Esto puede hacerse usando una instrucción DBY(71). Observe que cada conteo en la posición de memoria interna 71 (47H) representa 5 milisegundos de tiempo.

Continuación del ejemplo:

```
>DBY(71) = 0         (fraction of TIME = 0)

>PRINT TIME
0

>DBY(71) = 3         (fraction of TIME = 3*5ms = 15 ms)

>PRINT TIME
1.5 E-2
```

Sólo la parte entera de TIME cambia cuando se asigna un valor. Esto le permite generar intervalos precisos de tiempo. Por ejemplo, si desea crear un reloj exacto de 12 horas: hay 43200 segundos en un período de 12 horas, de manera que se usa una instrucción ONTIME 43200, [ln num]. Cuando se interrumpe TIME, se ejecuta la instrucción TIME 0, pero no se le reasigna un valor al contador de milisegundos. Si el tiempo de espera de interrupción excede 5 milisegundos, el reloj permanece exacto.

Capítulo 4

Comandos BASIC

Este capítulo describe e ilustra palabras y expresiones que causan la ejecución de una función dentro del programa BASIC o la línea de comando. La Tabla 4.A lista los mnemónicos correspondientes.

Tabla 4.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Establecer el punto de interrupción del programa.	BRKPNT	4-2
Reactivar la función de interrupción **[CTRL–C]**.	CALL 18	4-6
Inhabilitar la función de interrupción **[CTRL–C]**.	CALL 19	4-6
Continuar después de una parada o **[CTRL–C]**.	CONT	4-4
Parar la ejecución y regresar al modo de comando.	CONTROL C	4-5
Volver a iniciar una lista después de **[CTRL–S]**.	CONTROL Q	4-8
Interrumpir un comando de lista.	CONTROL S	4-7
Editar una línea del programa BASIC.	EDIT	4-9
Borrar el último programa BASIC almacenado en la ROM por un comando PROG.	ERASE	4-10
Forzar al módulo BASIC para que pase al “modo de espera hasta interrupción”.	IDLE	4-10
Listar el programa al dispositivo de la consola.	LIST	4-11
Listar el programa a la impresora en serie.	LIST#	4-12
Listar el programa al dispositivo conectado al puerto PRT1.	LIST@	4-12
Establecer los parámetros de los puertos PRT1, PRT2 y DH485.	MODE	4-13
Borrar el programa almacenado en la RAM.	NEW	4-14
Establecer el conteo NULL después de un retorno de carro.	NULL	4-14
Guardar el programa actual en la EPROM.	PROG	4-15
Guardar la información de velocidad en baudios en la EPROM.	PROG1	4-16
Guardar la información de velocidad en baudios en la EPROM y ejecutar el programa después del restablecimiento.	PROG2	4-17

Si necesita	Use este mnemónico	Página
Llamar al modo RAM.	RAM	4-19
Especificar una línea de comentario u observación.	REM	4-19
Volver a numerar el programa BASIC.	REN	4-20
Seleccionar el modo ROM.	ROM	4-21
Seleccionar el modo ROM y ejecutar el programa seleccionado.	RROM	4-22
Ejecutar un programa.	RUN	4-23
Iniciar la ejecución de programa de un solo paso.	SNGLSTP	4-24
Verificar la versión de firmware del módulo BASIC.	VER	4-25
Transferir un programa de la ROM a la RAM.	XFER	4-26

BRKPNT

Objetivo:

Use el comando BRKPNT para establecer un punto de interrupción de programa en el número de línea especificado por este comando. La ejecución del programa se detiene justo antes del número de línea especificado por el comando BRKPNT. Si el número de línea es cero, se inhabilita el punto de interrupción. Después que se ha llegado al punto de interrupción se pueden examinar variables usando instrucciones de asignación. Continúe desde el punto de interrupción usando el comando CONT. Una vez que se ha alcanzado el punto de interrupción, éste se inhabilita. Para detenerse en el mismo lugar dos veces, establezca el punto de interrupción dos veces. El comando BRKPNT funciona sólo en los programas que se ejecutan desde RAM. No detiene los programas que se ejecutan desde ROM.

Syntaxis:

BRKPNT[ln num]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  D=0 : SU=0 : AV=0
>20  REM GET 100 DATUM POINTS
>30  FOR I=1 TO 100
>40  REM GET ANOTHER DATUM
>50  GOSUB 140
>60  REM SUM THE DATA
>70  GOSUB 170
>80  NEXT I
>90  REM AVERAGE THE DATA
>100 GOSUB 200
>110 REM PRINT RESULT
>120 PRINT "THE AVERAGE VALUE IS ",AV
>130 END
>140 REM THIS SUBROUTINE GENERATES RANDOM DATA
>150 D=RND
>160 RETURN
>170 REM THIS SUBROUTINE SUMS THE DATA
>180 SU=SU+D
>190 RETURN
>200 REM THIS SUBROUTINE AVERAGES THE DATA
>210 AV=SU/I
>220 RETURN

READY
>BRKPNT 160

Breakpoint enabled.

READY
>RUN

STOP – IN LINE  160
READY
>PRINT D,SU,AV
 .86042573  0  0

>D = .5

>PRINT D,SU,AV
 .5  0  0

>CONT

THE AVERAGE VALUE IS  .48060383

READY
>
```

CONT

Objetivo:

Use el comando CONT para continuar la ejecución de un programa que se detuvo con un comando **[CTRL–C]**, con BRKPNT o con una instrucción STOP. Si detuvo un programa presionando **[CTRL–C]** en el dispositivo de la consola o ejecutando una instrucción STOP, se puede continuar la ejecución del programa escribiendo CONT. Si presiona **[CTRL–C]** mientras **[CTRL–C]** está habilitado, se detendrá el programa. Use CONT para continuar. Entre el paro y el reinicio del programa usted puede mostrar o cambiar los valores de las variables. Sin embargo, no podrá CONTinuar si el programa se modifica durante el STOP o después de un error.

Importante: **[CTRL–C]** borra todos los búfers de entradas y salidas.

Sintaxis:

CONT

Ejemplo:

```
>NEW

>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 10000
>20 PRINT I
>30 NEXT I
>40 END

READY
>RUN

 1
 2
 3
 4
 5
 6
 7
 8
 9
 10 [Typed [Ctrl-C] here]

STOP – IN LINE 15
READY
>CONT

20
21
22
```

Control C

Objetivo:

Use el comando **[CTRL–C]** para detener la ejecución del programa actual y regresar el módulo BASIC al modo de comando. En algunos casos puede continuar la ejecución del programa usando CONT. Para obtener más información, vea la explicación de CONT.

Importante: **[CTRL–C]** borra todos los búfers de entradas y salidas.

Sintaxis:

[**CTRL-C**]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 10000
>20 PRINT I
>30 NEXT I
>40 END
>RUN
  1
  2
  3
  4
  5 - (CONTROL C typed)

STOP - IN LINE 20

READY
>PRINT I
  27

>I =10

>CONT

  10
  11
  12
```

Note que después de presionar **[CTRL–C]** e imprimir I, el valor de I es 27. El valor de I se incrementa varias veces antes de detectar **[CTRL–C]**.

CALL 18 - Cómo volver a habilitar la función de interrupción Control C

Objetivo:

Vuelva a habilitar la función de interrupción **[CTRL–C]** ejecutando CALL 18 en un programa del módulo BASIC o desde el modo de comando.

Importante: Cuando **[CTRL–C]** está inhabilitado, usted no puede detener la ejecución del programa a través de un comando BASIC. El desconectar y volver a conectar la alimentación eléctrica vuelve a habilitar la verificación de **[CTRL–C]** hasta que el programa inhabilite de nuevo **[CTRL–C]**. Para detener la ejecución del programa, usted debe desconectar y volver a conectar la alimentación eléctrica y presionar [**CTRL-C**] antes que se ejecute la línea que inhabilita **[CTRL–C]**.

Sintaxis:

CALL 18

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 19
.
>90 CALL 18
```

CALL 19 - Cómo inhabilitar la función de interrupción Control C

Objetivo:

Inhabilite la función de interrupción **[CTRL–C]** ejecutando CALL 19 en un programa del módulo BASIC o desde el modo de comando.

Cuando se ejecuta CALL 19, se inhabilita la función de interrupción de **[CTRL–C]** para las operaciones LIST y RUN. El desconectar y volver a conectar la alimentación eléctrica regresa la función **[CTRL–C]** a su operación normal si ésta se inhabilita desde el modo de comando.

Importante: **[CTRL–C]** está habilitado de manera predeterminada.

Sintaxis:

CALL 19

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 19
```

Control S

Objetivo:

Use el comando **[CTRL–s]** para interrumpir el desplazamiento de un programa BASIC durante la ejecución de un comando LIST. **[CTRL–s]** detiene la salida del puerto transmisor si un programa está en ejecución. En este caso XOFF (**[CTRL–s]**) funciona de la siguiente manera:

Importante: **[CTRL–s]** sólo funciona si usted ha habilitado el handshaking del software en el puerto de programación.

- XOFF sólo funciona con instrucciones PRINT.
- Cuando se recibe durante PRINT, la salida de datos se suspende inmediatamente, pero la ejecución del programa continúa.
- Cuando se recibe en cualquier otro momento, el programa continúa hasta que encuentra una instrucción PRINT. En este momento se suspende la salida de datos. El programa continúa llenando el búfer de salidas hasta que éste se llene.
- Se requiere XON (**[CTRL–Q]**) para continuar la operación de salida de datos.

Sintaxis:

[**CTRL-S**]

Ejemplo:

```
> LIST
1  REM EXAMPLE PROGRAM
10 A = 1
20 DO
[CTRL-S]
.
.
.
[CTRL-Q]
30 A = A+1
40 PRINT A
50 WHILE A < 20

READY
>
```

En este ejemplo, la salida se suspende cuando se presiona [**CTRL-S**]. La salida continúa después de presionar [**CTRL-Q**].

Control Q

Objetivo:

Use el comando **[CTRL–Q]** para reiniciar un comando LIST o una salida PRINT interrumpida por **[CTRL–s]**.

Sintaxis:

[**CTRL-Q**]

Ejemplo:

```
> LIST
1  REM EXAMPLE PROGRAM
10 A = 1
20 DO
[CTRL-S]
.
.
.
[CTRL-Q]
30 A = A+1
40 PRINT A
50 WHILE A < 20

READY
>
```

En este ejemplo, la salida se suspende cuando se presiona [**CTRL-S**]. La salida continúa después de presionar [**CTRL-Q**].

EDIT

Objetivo:

Use el comando EDIT para tener acceso al editor de la línea BASIC. Use este editor para editar una línea del programa actual en la RAM. La Tabla 4.B lista las operaciones del editor BASIC.

Tabla 4.B
Operaciones del editor BASIC

Operación	Función	Teclas
Mover	Use la operación Mover para controlar horizontalmente el cursor.	[**Space bar**] - mueve el cursor un espacio a la derecha. [**Backspace**] - mueve el cursor un espacio a la izquierda.
Reempla–zar	Use la operación Reemplazar para reemplazar el carácter en la posición actual del cursor.	Presione la tecla que corresponde al carácter que reemplazará el carácter en la posición actual del cursor.
Insertar	Use la operación Insertar para insertar texto en la posición actual del cursor. **Importante**: Cuando usted usa la operación Insertar, todo el texto a la derecha del cursor desaparece hasta que presione el segundo [**Ctrl-A**]. La longitud total de la línea es de 79 caracteres.	[**Ctrl-A**] **Importante:** Usted debe presionar un segundo [**Ctrl-A**] para terminar el comando Insertar.
Borrar	Use la operación Borrar para borrar el carácter en la posición actual del cursor.	[**Ctrl-D**]
Salir	Use la(s) operación(es) Salir para salir del editor, guardando o no los cambios.	[**Ctrl-Q**] - sale del editor y reemplaza la línea anterior con la línea editada. [**Ctrl-C**] - sale del editor sin guardar los cambios hechos en la línea.
Reescribir	Use la operación Reescribir para copiar la línea actual de texto e insertarla en la línea que sigue a la línea actual. El cursor se mueve al primer carácter en la línea nueva.	[**RETURN**]

Sintaxis:

EDIT

Ejemplo:

```
>EDIT 150
```

Resultado: Muestra la línea del programa número 150 para editar.

ERASE

Objetivo:

Use el comando ERASE para borrar el último programa BASIC almacenado en la EEPROM a través de un comando PROG.

Sintaxis:

ERASE

Ejemplo:

```
>ERASE

>ROM 13 ERASED
```

Resultado: Se borra el último programa almacenado en la EEPROM (ROM 13 en este ejemplo).

IDLE

Objetivo:

Use el comando IDLE para forzar al módulo BASIC a que entre al modo de espera hasta interrupción. La ejecución del programa se detiene hasta que haya una condición ONTIME. La interrupción ONTIME debe habilitarse antes de ejecutar el comando IDLE porque de lo contrario el módulo BASIC entrará a un modo de espera indefinida. **[CTRL–C]** sale del comando IDLE si **[CTRL–C]** está habilitado.

Sintaxis:

IDLE

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 TIME = 0
>20 CLOCK1
>30 ONTIME 2,70
>40 IDLE
>50 PRINT “END OF TEST!!!”
>60 END
>70 PRINT “TIMER INTERRUPT AT - ”,TIME,“SECONDS.”
>80 RETI

READY
>RUN

TIMER INTERRUPT AT - 2.005 SECONDS.
END OF TEST!!!
```

LIST

Objetivo:

Use el comando LIST para imprimir el programa al dispositivo de la consola. Los espacios se insertan después del número de línea y antes y después de las instrucciones. Esto ayuda en la depuración de los programas del módulo BASIC. Puede terminar el listado de un programa en cualquier momento presionando **[CTRL–C]** en el dispositivo de la consola. Puede interrumpir y continuar el listado usando **[CTRL–s]** y **[CTRL–Q]**.

Importante: [**CTRL-C**] termina el listado si la verificación de [**CTRL-C]** está habilitada. [**CTRL-S**] detiene el listado hasta que se presione [**CTRL-Q**] si el handshaking del software está habilitado.

Sintaxis:

LIST [ln num]

LIST [ln num] – [ln num]

La primera variación hace que el programa imprima desde el número de línea designado [ln num] hasta el final del programa. La segunda variación hace que el programa imprima desde el primer número de línea designado [ln num] hasta el segundo número de línea designado [ln num].

Importante: Es necesario separar los dos números de línea con un guión (–).

Ejemplo:

```
>LIST
1  REM EXAMPLE PROGRAM
10 PRINT “LOOP PROGRAM”
20 FOR I = 1 TO 3
30 PRINT I
40 NEXT I
50 END

READY
>LIST 30
1  REM EXAMPLE PROGRAM
30 PRINT I
40 NEXT I
50 END

READY
>LIST 20-40
1  REM EXAMPLE PROGRAM
20 FOR I = 1 TO 3
30 PRINT I
40 NEXT I
```

LIST@

Objetivo:

Use el comando LIST@ para imprimir el programa al dispositivo conectado al puerto PRT1. Todos los comentarios que se aplican al comando LIST se aplican al comando LIST@. Usted debe configurar los parámetros del puerto PRT1 de manera que coincidan con su dispositivo particular de lista. Los parámetros PRT1 (velocidad en baudios, paridad, bits de parada, etc) pueden establecerse usando el comando MODE.

Sintaxis:

LIST@

Ejemplos:

```
LIST@ :REM LISTS ALL LINES IN THE PROGRAM

LIST@10-20 :REM LISTS LINES 10 THROUGH 20
```

LIST#

Objetivo:

Use el comando LIST# para imprimir el programa al dispositivo conectado al puerto PRT2. Todos los comentarios que se aplican al comando LIST se aplican al comando LIST#. Usted debe configurar los parámetros del puerto PRT2 de manera que coincidan con su dispositivo particular de lista. Los parámetros PRT2 (velocidad en baudios, paridad, bits de parada, etc) pueden establecerse usando el comando MODE.

Sintaxis:

LIST#

Ejemplo:

Consulte los ejemplos de LIST@.

MODE

Objetivo:

Use el comando MODE para establecer los parámetros de puerto de los puertos PRT1, PRT2 y DH485. La Tabla 4.C lista los parámetros de puerto para los puertos PRT1 y PRT2.

Tabla 4.C
Parámetros de puerto PRT1 y PRT2

Parámetros de puerto	Selecciones	Valores predetermi–nados
Velocidad en baudios	300, 600, 1200, 2400, 4800, 9600, 19200	1200
arg1 (paridad)	Ninguno (N), par (E), impar (O)	N
arg2 (número de bits de datos)	7 u 8	8
arg3 (número de bits de parada)	1 ó 2	1
arg4 (handshaking)	No hay handshaking (N) Handhaking de software (S) Handhaking de hardware (H) Handhaking de hardware y software (B)	S
arg5 (tipo de almacenamiento)	Almacene la información en la ROM y RAM del usuario (E) Almacene la información en la RAM con batería de respaldo (R)	R

Importante: Si algún argumento (que no sea el nombre de puerto ni la velocidad en baudios) se deja en blanco, entonces ese argumento cambia al valor predeterminado especificado previamente para ese argumento.

La Tabla 4.D lista los parámetros de puerto para el puerto DH485.

Tabla 4.D
Parámetros de puerto DH485

Parámetros de puerto	Selecciones	Valores predetermi–nados
Velocidad en baudios	300, 600, 1200, 2400, 4800, 9600, 19200	19200
arg1 (dirección del nodo principal)	0 a 31	0
arg2 (dirección del nodo del módulo)	1 a 31	1
arg3 (dirección máxima de nodo)	1 a 31	31
arg4 (not used)		
arg5 (storage type)	Almacene la información en la ROM y RAM del usuario (E) Almacene la información en la RAM con batería de respaldo (R)	R

Importante: Si algún argumento (que no sea el nombre de puerto ni la velocidad en baudios) se deja en blanco, entonces ese argumento cambia al valor predeterminado especificado previamente para ese argumento.

Sintaxis:

MODE (nombre de puerto, velocidad en baudios, arg1, arg2, arg3, arg4, arg5)

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 MODE(DH485,19200,0,1,2,,R)
.
.
.
>25 MODE(PRT1,1200,N,8,,,)
```

Importante: La opción de tipo de almacenamiento E no puede usarse si MODE se usa como una instrucción.

NEW

Objetivo:

Use el comando NEW para borrar el programa actualmente almacenado en la RAM. Además, todas las variables están establecidas en CERO; todas las cadenas y todas las interrupciones llamadas de BASIC se borran. El reloj autónomo, la asignación de cadena y los valores internos indicadores de la pila no son afectados. El comando NEW se usa para borrar un programa y todas las variables en la RAM.

Sintaxis:

NEW

Ejemplo:

```
>NEW

>LIST

>READY
>
```

NULL

Objetivo:

Use el comando NULL para determinar cuántos caracteres NULL (00H) da el módulo BASIC después de un retorno de carro en una instrucción de impresión. Después de la inicialización este valor se establece en 0. La mayoría de impresoras contienen un búfer RAM que elimina la necesidad de dar caracteres NULL después de un retorno de carro.

Sintaxis:

NULL[integer]

PROG

Importante: Antes de intentar programar la EEPROM, lea las secciones PROG, PROG1 y PROG2 de este capítulo.

Objetivo:

Use el comando PROG para programar la EEPROM residente con el programa actual en la RAM. El módulo BASIC no puede programar UVPROM.

Importante: Asegúrese de haber seleccionado el programa que desea guardar antes de usar el comando PROG. Su módulo BASIC no copia automáticamente el programa RAM a la EEPROM. Si se produce un error durante la programación de la EEPROM, aparecerá el mensaje **`ERROR: Programming sequence failure`**.

Después de escribir **`PROG`**, el módulo BASIC muestra el número en el ARCHIVO EEPROM que el programa ocupa. La programación sólo toma unos segundos.

Sintaxis:

PROG

Ejemplo:

```
>LIST
1  REM EXAMPLE PROGRAM
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
40 END

READY
>PROG

ROM 12

Programming sequence successful.

READY
>ROM 12
```

```
>LIST
1  REM EXAMPLE PROGRAM
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
40 END

READY
>
```

Resultado: El programa que se acaba de colocar en la EEPROM es el doceavo programa almacenado.

Importante: Si excede el espacio disponible en la EEPROM, no puede continuar la programación hasta que la borre. Use el comando ERASE para borrar el último programa almacenado en la EEPROM. Asegúrese de usar CALL 81 o CALL 82 para determinar el espacio de memoria antes de programar su EEPROM. Consulte el capítulo 5 para obtener información sobre CALL 81 y 82.

PROG1

Importante: Antes de intentar programar una EEPROM, lea las secciones PROG, PROG1 y PROG2 de este capítulo.

Objetivo:

Use el comando PROG1 para programar la EEPROM residente con información de puerto para los tres puertos así como la información de almacenamiento MTOP. Al activar el módulo, el módulo BASIC lee esta información e inicializa MTOP y los tres puertos en serie. El mensaje de entrada al sistema se envía a la consola inmediatamente después que el módulo termina su secuencia de restablecimiento. Si cambia la velocidad en baudios del dispositivo de la consola, tiene que reprogramar la EEPROM para que el módulo sea compatible con la consola nueva. Reprograme cambiando el puerto apropiado o la información MTOP, luego ejecute PROG1 otra vez.

Sintaxis:

PROG1

Ejemplo:

```
READY
>PROG1

Programming sequence successful.
```

PROG2

Importante: Antes de intentar programar una EEPROM, lea las secciones PROG, PROG1 y PROG2 de este capítulo. Tome nota de que el comando PROG2 no transfiere el programa RAM a la EEPROM. El comando PROG2 habilita el primer programa en la EEPROM que va a ser cargado en cada puesta en marcha.

Objetivo:

Use el comando PROG2 de la misma manera que usaría el comando PROG1 excepto por lo siguiente: en lugar de entrar al sistema y entrar al modo de comando, el módulo inmediatamente empieza a ejecutar el primer programa almacenado en la EEPROM residente. De lo contrario, ejecuta el programa almacenado en la RAM.

Usted puede usar el comando PROG2 para ejecutar (RUN) un programa al momento de la puesta en marcha sin conectar a una consola. Guardar la información PROG2 es lo mismo que escribir una ROM 1, secuencia de comando RUN. Esta función también le permite escribir una secuencia especial de inicialización en BASIC y generar un mensaje de entrada para aplicaciones específicas. El comando PROG2 no altera el primer comando en el módulo de memoria.

Importante: El comando PROG2 no hace que el módulo BASIC se active al momento de la puesta en marcha si las comunicaciones predeterminadas PRT1 se han seleccionado a través de JW4. Consulte el capítulo 3 del Manual de diseño e integración del módulo BASIC SLC 500 (número de publicación 1746–6.1ES) para obtener más información.

La siguiente figura muestra la operación del módulo BASIC desde una condición de puesta en marcha usando PROG1 y PROG2, o RAM con batería de respaldo.

Sintaxis:

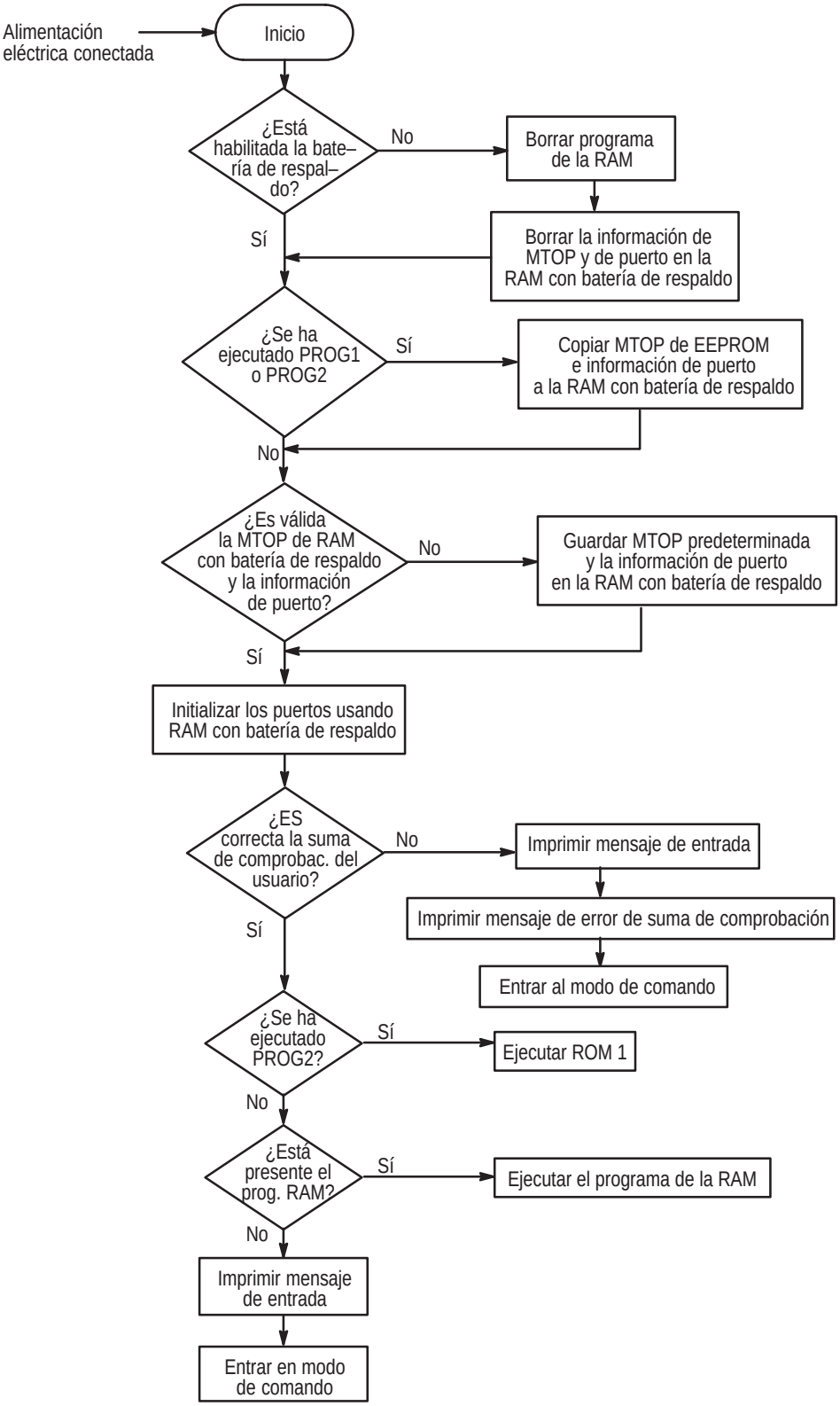
PROG2

Ejemplo:

```
READY
>PROG2

Programming sequence successful.
```

Figura 4.1
Operación de PROG1 o PROG2



RAM

Objetivo:

Use el comando RAM para indicarle al intérprete del módulo BASIC que seleccione el programa actual en la RAM. El programa actual se ve durante un comando LIST y se ejecuta cuando se escribe **RUN**.

Importante: El espacio de la RAM está limitado a 24 K bytes. Use la siguiente fórmula para calcular el espacio de RAM de usuario disponible.

RAM de usuario disponible = MTOP-H

H = LEN + S + 6*A + 8*V + 512

Donde:

LEN = valor de control del sistema que contiene la longitud del programa RAM actual

S = número de bytes asignados para cadenas (primer valor en la instrucción STRING)

A = suma de todos (dimensiones de la matriz +1)

V = suma de todos los nombres de variables usados (incluyendo cada nombre de matriz)

Sintaxis:

RAM

Ejemplo:

```
READY
>RAM
```

REM

Objetivo:

Use el comando REM para especificar una línea de comentario en un programa BASIC. Si se añaden líneas de comentarios a un programa, éste se entiende mejor. Las líneas del programa que empiezan con un comando REM no pueden terminar con el signo de dos puntos (:). Los comandos REM pueden colocarse después de un signo de dos puntos (:) en una línea del programa. Esto le permite colocar un comentario en cada línea.

Importante: Los comandos REM prolongan la ejecución del programa. Uselos de manera selectiva o colóquelos al final del programa, donde no afecten la velocidad de ejecución del mismo. No use comandos REM en lazos o subrutinas que se llaman frecuentemente.

Sintaxis:

REM

Ejemplo:

```
>10 REM THIS IS A COMMENT LINE
>20 NEW : REM THIS IS ALSO A COMMENT LINE
```

REN

Objetivo:

Use el comando RTEN para volver a numerar las líneas del programa.

Sintaxis:

REN[número nuevo],[número anterior],[incremento]

Ejemplos:

```
REN
```

Resultado: Vuelve a numerar el programa completo. El primer número de línea nuevo es 10. Los números de línea incrementan en valores de 10.

```
REN 20
```

Resultado: Vuelve a numerar el programa completo. El primer número de línea nuevo es 10. Los números de línea incrementan en valores de 20.

```
REN 300,50
```

Resultado: Vuelve a numerar el programa completo. El primer número de línea nuevo es 300. Los números de línea incrementan en valores de 50.

```
REN 1000,900,20
```

Resultado: Vuelve a numerar el programa desde la línea 900 en adelante. El número de línea 900 se convierte en número de línea 1000. Los números de línea siguientes incrementan en valores de 20.

ROM

Objetivo:

Use el comando ROM para indicarle al intérprete del módulo BASIC que seleccione el programa actual en la EEPROM o UVPROM. El programa actual se ve durante un comando LIST y se ejecuta cuando se escribe **RUN**.

Importante: Su módulo BASIC puede ejecutar y almacenar hasta 255 programas en la EEPROM, dependiendo del tamaño de los programas y la capacidad de la EEPROM. Los programas se almacenan en una cadena de secuencia llamada archivo EEPROM, en la EEPROM para su recuperación y ejecución.

Cuando usted introduce ROM [entero], el módulo BASIC selecciona ese programa en la memoria EEPROM y lo acepta como el programa actual. Si no se escribe un entero después del comando ROM (ejemplo: **ROM**) el módulo regresa de manera predeterminada a ROM 1. Puesto que los programas se almacenan en secuencia en la EEPROM, el entero que sigue al comando ROM selecciona el programa que el usuario desea ejecutar o listar. Si trata de seleccionar un programa que no existe (por ejemplo: escribe **ROM 8** y sólo hay 6 programas almacenados en la EEPROM) aparece el mensaje **ERROR: PROM MODE**.

El módulo no transfiere el programa de la EEPROM a la RAM cuando el modo ROM está seleccionado. Si trata de alterar un programa en el modo ROM escribiendo en un número de línea, aparece el mensaje **ERROR: PROM MODE**. El comando XFER le permite transferir un programa de la EEPROM a la RAM para fines de edición. No obtendrá un mensaje de erorr si trata de editar una línea del programa RAM.

Importante: Cuando se transfieren programas de la EEPROM a la RAM, se pierde el contenido previo de la RAM.

Puesto que el comando ROM *no* transfiere un programa a la RAM, es posible tener diferentes programas en la ROM y la RAM simultáneamente. Usted puede ir y volver entre los dos modos cuando está en el modo de comando. Si está en el modo de marcha, puede ir y volver usando los CALL 70, 71 y 72. También puede usar toda la memoria RAM para almacenamiento de variables si el programa se almacena en la EEPROM. El valor de control del sistema MTOP siempre se refiere a la RAM. El valor de control del sistema LEN se refiere al programa actualmente seleccionado en la RAM o la ROM.

Sintaxis:

ROM[integer]

Ejemplo:

```
READY
>ROM1
```

RROM

Objetivo:

Use el comando RROM para indicarle al intérprete del módulo BASIC que seleccione el programa en la EEPROM o UVPROM y luego ejecute el programa. Este comando es equivalente a escribir **ROM** y luego **RUN**.

Importante: Su módulo BASIC puede ejecutar y almacenar hasta 255 programas en la EEPROM, dependiendo del tamaño de los programas y la capacidad de la EEPROM. Los programas se almacenan en una cadena de secuencia llamada archivo EEPROM, en la EEPROM para su recuperación y ejecución.

Cuando usted introduce RROM [entero], el módulo BASIC selecciona ese programa en la memoria EEPROM, lo acepta como el programa actual y empieza la ejecución del programa. Si no se escribe un entero después del comando RROM (ejemplo: **RROM**) el módulo regresa de manera predeterminada a RROM 1. Puesto que los programas se almacenan en secuencia en la EEPROM, el entero que sigue al comando RROM selecciona el programa que el usuario desea ejecutar o listar. Si trata de seleccionar un programa que no existe (por ejemplo: escribe **RROM 8** y sólo hay 6 programas almacenados en la EEPROM) aparece el mensaje **ERROR: PROM MODE**.

El módulo no transfiere el programa de la EEPROM a la RAM cuando el modo ROM está seleccionado. Si trata de alterar un programa en el modo ROM escribiendo en un número de línea, aparece el mensaje **ERROR: PROM MODE**. El comando XFER le permite transferir un programa de la EEPROM a la RAM para fines de edición. No obtendrá un mensaje de error si trata de editar una línea del programa RAM.

Importante: Cuando se transfieren programas de la EEPROM a la RAM, se pierde el contenido previo de la RAM.

Como el comando RROM *no* transfiere un programa a la RAM, es posible tener diferentes programas en la ROM y la RAM simultáneamente. Usted puede ir y volver entre los dos modos cuando está en el modo de comando. Si está en el modo de marcha, puede ir y volver usando los CALL 70, 71 y 72. También puede usar toda la memoria RAM para almacenamiento de variables si el programa se almacena en la EEPROM. El valor de control del sistema MTOP siempre se refiere a la RAM. El valor de control del sistema LEN se refiere al programa actualmente seleccionado en la RAM o la ROM.

Sintaxis:

RROM[integer]

Ejemplo:

```
READY
>RROM
```

RUN

Objetivo:

Use el comando RUN para poner todas las variables a cero, borrar todas las interrupcioens creadas por BASIC y empezar la ejecución del programa con el primer número de línea del programa seleccionado. El comando RUN y la instrucción GOTO son los únicos que pueden colocar el intérprete del módulo BASIC en el modo de marcha desde el modo de comando. Termine la ejecución del programa en cualquier momento presionando **[CTRL–C]** en el dispositivo de la consola.

Variaciones: Algunos intérpretes BASIC permiten que un número de línea siga al comando RUN (ejemplo: **RUN 100**). El módulo BASIC no permite esta variación en el comando RUN.

La ejecución empieza con el primer número de línea. Para obtener una función similar al comando RUN [ln num], use la instrucción GOTO [ln num] en el modo directo. Vea la instrucción GOTO en el capítulo 7.

Sintaxis:

RUN

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 3
>20 PRINT I
>30 NEXT I
>40 END
>RUN

  1
  2
  3

READY
>
```

SNGLSTP

Objetivo:

Use el comando SNGLSTP para iniciar la ejecución de un programa de un solo paso. Si el número especificado por este comando es cero, la ejecución de un solo paso se inhabilita. Si el número no es cero, se establece un punto de interrupción en cada línea en el programa. Inicie el programa escribiendo el comando RUN. Después de cada parada, escriba **CONT** para ejecutar la siguiente línea. Puede inspeccionar variables o asignar variables en cada punto de interrupción. SNGLSTP funciona sólo en programas que se ejecutan desde la RAM. No detiene un programa que se ejecuta desde la EEPROM.

Sintaxis:

SNGLSTP[entero]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 5
>20 PRINT I
>30 NEXT I
>40 PRINT “PASSED FOR - NEXT LOOP”
>50 PRINT “THIS IS THE END”
>60 END

READY
>SNGLSTP 20

SINGLE STEP ENABLED

READY
>RUN

STOP - LINE 20
READY
>CONT

1

STOP - LINE 30
READY
>CONT

STOP - LINE 20
READY
>CONT

2
```

```
STOP - LINE 30
READY
>CONT


STOP - LINE 20
READY
>CONT

3

STOP - LINE 30
READY
>SNGLSTP 0

SINGLE STEP DISABLED

READY
>CONT

4
5
PASSED FOR - NEXT LOOP
THIS IS THE END

READY
>
```

VER

Objetivo:

Use el comando VER para imprimir el mensaje de entrada al módulo BASIC que muestra la versión actual del firmware.

Sintaxis:

VER

Ejemplo:

```
>VER
SLC 500 BASIC Module-Catalog Number 1746-BAS
Firmware release: 1.00
Allen-Bradley Company, Copyright 1991
All rights reserved

>
```

XFER

Objetivo:

Use el comando XFER para transferir el programa seleccionado actualmente en la ROM a la RAM y seleccionar el modo RAM. Después que se ejecuta el comando XFER, usted puede editar el programa de la misma manera que edita cualquier programa RAM.

Importante: El comando XFER borra todos los programas RAM existentes.

Sintaxis:

XFER

Ejemplo:

```
READY
>XFER
```

Capítulo 5

Llamadas en la línea de comando

Este capítulo describe e ilustra llamadas (CALL) que hacen que ocurra una función dentro del módulo BASIC. Estas llamadas no pueden ejecutarse dentro del programa BASIC *pero* se introducen en la línea de comando. La Tabla 5.A lista los nemónicos correspondientes.

Tabla 5.A
Guía de referencia del capítulo

CALL 73 - Inhabilitación de la RAM con batería de respaldo

Objetivo:

Use CALL 73 para inhabilitar la RAM con batería de respaldo. Cuando se ejecuta este CALL, se imprime el mensaje **`Battery Backup Disabled`** en el terminal principal. La inhabilitación de la RAM con batería de respaldo permite hacer un restablecimiento. Cuando se desconecta la alimentación eléctrica al módulo BASIC, el contenido de la RAM se destruye. Cuando se vuelve a conectar la alimentación, la RAM está vacía y la batería de respaldo está rehabilitada.

Sintaxis:

CALL 73

Ejemplo:

```
>CALL 73

Battery Backup Disabled

>REM TURNING POWER OFF, THEN BACK ON
```

CALL 74 - Habilitación de la RAM con batería de respaldo

Objetivo:

Use CALL 74 para habilitar la RAM con batería de respaldo. Cuando se ejecuta este CALL, se imprime el mensaje **`Battery Backup Enabled`** en el terminal principal. La RAM con batería de respaldo se habilita durante el encendido del módulo BASIC y permanece habilitada hasta que se ejecute un CALL 73 o hasta que la batería falle.

Sintaxis:

CALL 74

Ejemplo:

```
>CALL 74

Battery Backup Enabled
```

CALL 77 - Almacenamiento protegido de variables

Importante: Cambie CALL 77 desde el modo de comando solamente para asegurar una operación correcta.

Objetivo:

Use CALL 77 para reservar la parte superior de la memoria RAM para almacenamiento protegido de variables. Los valores se guardan si se invoca BATTERY-BACKUP. Los valores se guardan con el comando ST@ y se recuperan con el comando LD@. Cada variable almacenada requiere 6 bytes de espacio de almacenamiento.

Usted debe restar 6 veces el número de variables almacenadas desde MTOP reduciendo la memoria RAM disponible. Este valor se empuja (PUSH) a la pila como una nueva dirección MTOP. Todos los señaladores variables apropiados se reconfiguran. Haga esto sólo en el modo de comando.

Importante: No deje que la dirección ST@ escriba encima de la dirección MTOP. Esto podría alterar el valor de una variable o cadena. La selección MTOP más baja que puede establecerse es 4096 (1000H).

Importante: Call 77 desasigna toda la memoria de cadenas junto con el contenido de las cadenas. Por lo tanto, asegúrese de efectuar este CALL antes de ejecutar la instrucción de cadena.

Sintaxis:

PUSH [nueva dirección MTO]
CALL 77

Ejemplo: (para guardar 2 variables)

```
>PRINT MTOP
24575
>PRINT MTOP–12
24563
>PUSH 24563:REM NEW MTOP ADDRESS
>CALL 77
```

```
>1   REM EXAMPLE PROGRAM
>10  K = 678*PI
>20  L = 520
>30  PUSH K
>40  ST@ 24575 : REM STORE K IN PROTECTED AREA
>50  PUSH L
>60  ST@ 24569 : REM STORE L IN PROTECTED AREA
>70  REM TO RETRIEVE PROTECTED VARIABLES
>80  LD@ 24575 : REM REMOVE K FROM PROTECTED AREA
>90  POP K
>100 LD@ 24569 : REM REMOVE L FROM PROTECTED AREA
>110 POP L
>120 REM USE LD@ AFTER POWER LOSS AND BATTERY BACK–UP IS
USED
```

CALL 81 - Verificación y descripción del módulo de memoria del usuario

Objetivo:

Use CALL 81 para verificar el módulo de memoria del usuario antes de escribir un programa en el módulo de memoria. Esta rutina:

- determina el número de programas del módulo de memoria
- determina el número de bytes que quedan en el módulo de memoria
- determina el número de bytes en el programa RAM
- imprime un mensaje que indica si hay suficiente espacio disponible en el módulo de memoria para el programa RAM
- verifica la suma de comprobación del módulo de memoria si se encuentra el programa
- imprime un mensaje de advertencia si falla la suma de comprobación

Importante: CALL 81 no puede detectar un módulo de memoria defectuoso.

No se necesitan ni PUSH ni POP.

Sintaxis:

CALL 81

Ejemplo:

```
>CALL 81

Number of BASIC programs in (E)EPROM.........  3
Available bytes to end of user (E)EPROM.......  7944
Available bytes to beginning of assembly pgm..  3848
Length of BASIC program in RAM...............  76

Program will fit in (E)EPROM.

READY
>
```

CALL 82 - Verificación del mapa del módulo de memoria del usuario

Objetivo:

Use CALL 82 para verificar el módulo de memoria del usuario y mostrar un mapa del lugar donde están almacenados todos los programas BASIC. El programador puede determinar, usando esta llamada, en qué parte del módulo de memoria hay espacio vacío y cuánto espacio está disponible. No se necesitan ni PUSH ni POP.

Sintaxis:

CALL 82

Ejemplo:

```
>CALL 82

 8010H --  805CH --> ROM 1
 805DH --  80A9H --> ROM 2
 80AAH --  80F6H --> ROM 3
 80F7H --  FFFFH --> UNUSED
>
```

CALL 101 - Carga del código del módulo de memoria del usuario al terminal principal

Objetivo:

Use CALL 101 para cargar el código del módulo de memoria del usuario al terminal principal. Esta llamada requiere dos PUSH y ningún POP. El primer PUSH es la dirección inicial. El segundo PUSH es la dirección final. Esta llamada convierte datos dentro del rango de direcciones al formato hex Intel™, luego imprime la información al puerto de programación. Si las direcciones no son correctas se imprimirá un mensaje de error.

Sintaxis:

PUSH [dirección inicial]
PUSH [dirección final]
CALL 101

Ejemplo:

```
>PUSH 8000 : PUSH 804FH : CALL 101

:108000003107021327CC3313276607005FFF473081
:108010005509000A8B41E034290D1000149C3130C1
:108020002C32302C33302C34300D0A001EA049EA9B
:1080300030A6330D0900289B41E049290D06003286
:1080400097490D0A003CA04AEA4FA6330D090046A5
:00000001F
>
```

CALL 103 - Impresión del búfer de salidas y señalador del PRT1

Objetivo:

Use CALL 103 para imprimir todo el búfer de salidas con dirección, señalador frontal y número de caracteres en el búfer en la pantalla del puerto de programación. No se necesitan ni PUSH ni POP.

Use esta información como ayuda en la localización y corrección de fallos. No afecta el contenido del búfer.

Sintaxis:

CALL 103

Ejemplo:

```
>CALL 103
                                        PRT1 Output Queue

6D00H 3AH 31H 30H 38H 30H 34H 30H 30H 30H 39H 37H 34H 39H 30H 44H
30H
6D10H 41H 30H 30H 33H 43H 41H 30H 34H 41H 45H 41H 34H 46H 41H 36H
33H
6D20H 33H 30H 33H 30H 48H 20H 33H 33H 48H 20H 33H 30H 48H 20H 34H
38H
6D30H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 33H 48H 20H 34H
38H
6D40H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 30H 48H 20H 34H
38H
6D50H 48H 20H 32H 30H 48H 20H 33H 34H 48H 20H 33H 38H 48H 0DH 0AH
20H
6D60H 36H 44H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 34H
38H
6D70H 48H 20H 32H 30H 48H 20H 33H 34H 48H 20H 33H 38H 48H 0DH 0AH
20H
6D80H 36H 44H 37H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H
32H
6D90H 48H 20H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H
33H
6DA0H 48H 20H 33H 34H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H
33H
6DB0H 48H 20H 33H 38H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H
34H
6DC0H 48H 20H 33H 38H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H
32H
6DD0H 48H 20H 33H 30H 48H 20H 34H 38H 48H 20H 32H 30H 48H 20H 33H
33H
6DE0H 48H 20H 33H 34H 48H 0DH 0AH 20H 36H 44H 43H 30H 48H 20H 34H
38H
6DF0H 48H 20H 32H 30H 48H 20H 33H 33H 48H 20H 33H 38H 48H 20H 34H
34H

Output queue front pointer is:  6D29H
```

CALL 104 - Impresión del búfer de entradas y señalador del PRT1

Objetivo:

Use CALL 104 para imprimir todo el búfer de entradas con dirección, señalador frontal y número de caracteres en el búfer en la pantalla del puerto de programación. No se necesitan ni PUSH ni POP.

Use esta información como ayuda en la localización y corrección de fallos. No afecta el contenido del búfer.

Sintaxis:

CALL 104

Ejemplo:

```
>CALL 104
                                        PRT1 Input Queue

6C00H 33H 0DH 43H 41H 4CH 4CH 20H 31H 30H 34H 7FH 7FH 7FH 7FH 7FH
7FH
6C10H 7FH 7FH 52H 45H 4DH 20H 45H 58H 41H 4DH 50H 4CH 45H 53H 7FH
7FH
6C20H 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 7FH 0DH 0DH 0DH 0DH 0DH
0DH
6C30H 0DH 0DH 0DH 45H 58H 41H 4DH 7FH 7FH 7FH 7FH 52H 45H 4DH 20H
45H
6C40H 58H 41H 4DH 50H 4CH 45H 53H 20H 4FH 4EH 20H 50H 41H 47H 45H
20H
6C50H 36H 2DH 37H 0DH 43H 41H 4CH 4CH 20H 31H 30H 34H 0DH 52H 4DH
41H
6C60H 4EH 54H 20H 7FH 7FH 7FH 54H 20H 44H 41H 54H 41H 0DH 52H 45H
4DH
6C70H 20H 54H 48H 45H 52H 45H 20H 49H 53H 20H 4EH 4FH 20H 52H 45H
41H
6C80H 4CH 20H 52H 45H 53H 50H 4FH 4EH 53H 45H 20H 57H 48H 49H 43H
48H
6C90H 20H 57H 49H 4CH 4CH 20H 53H 48H 4FH 57H 20H 55H 50H 20H 49H
4EH
6CA0H 20H 41H 4EH 20H 45H 58H 41H 4DH 50H 4CH 45H 0DH 0DH 0DH 0DH
0DH
6CB0H 52H 45H 4DH 20H 45H 58H 41H 4DH 50H 4CH 45H 53H 20H 4FH 4EH
20H
6CC0H 50H 41H 47H 45H 20H 36H 2DH 36H 0DH 50H 55H 53H 48H 20H 38H
30H
6CD0H 30H 30H 48H 3AH 50H 7FH 7FH 20H 70H 55H 53H 7FH 7FH 7FH 3AH
20H
6CE0H 50H 55H 53H 48H 20H 38H 30H 7FH 30H 34H 46H 48H 20H 3AH 43H
41H
6CF0H 4CH 4CH 20H 31H 30H 31H 0DH 0DH 0DH 43H 41H 4CH 4CH 20H 31H
30H

Input queue front pointer is:  6C5DH
```

CALL 109 - Impresión de la pila de argumentos

Objetivo:

Use CALL 109 para imprimir los primeros 9 valores en la pila de argumentos a la consola. No se necesitan ni PUSH ni POP. Use esta información como ayuda en la localización y corrección de fallos. No afecta el contenido de la pila de argumentos ni el señalador de la pila.

Sintaxis:

CALL 109

Ejemplo:

```
>CALL 109

 1C9H 00H 00H 00H 00H 00H 00H
 1CFH 00H 00H 00H 00H 00H 00H
 1D5H 00H 00H 00H 00H 00H 00H
 1DBH 41H 67H 50H 00H 00H 7EH
 1E1H 83H 75H 00H 00H 00H 7CH
 1E7H 13H 04H 00H 00H 00H 7CH
 1EDH 32H 84H 70H 00H 00H 85H
 1F3H 32H 76H 80H 00H 00H 85H
 1F9H 00H 00H 00H 00H 00H 00H

Argument stack pointer is:  01FEH
```

CALL 110 - Impresión del búfer de salidas y señalador del PRT2

Objetivo:

Use CALL 110 para imprimir todo el búfer de salidas con direcciones, señalador frontal y número de caracteres en el búfer al dispositivo de la consola. No se necesitan ni PUSH ni POP.

Use esta información como ayuda en la localización y corrección de fallos. No afecta el contenido del búfer.

Sintaxis:

CALL 110

Ejemplo:

```
>CALL 110
                                        PRT2 Output Queue

6F00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6F10H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6F20H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6F30H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6F40H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6F50H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6F60H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6F70H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6F80H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6F90H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6FA0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6FB0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6FC0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6FD0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6FE0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6FF0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H

Output queue front pointer is:  6F00H
```

CALL 111 - Impresión del búfer de entradas y señalador del PRT2

Objetivo:

Use CALL 111 para imprimir todo el búfer de entradas con direcciones, señalador frontal y número de caracteres en el búfer al dispositivo de la consola. No se necesitan ni PUSH ni POP.

Use esta información como ayuda en la localización y corrección de fallos. No afecta el contenido del búfer.

Sintaxis:

CALL 111

Ejemplo:

```
>CALL 111
                                        PRT2 Input Queue

6E00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6E10H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6E20H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6E30H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6E40H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6E50H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6E60H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6E70H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6E80H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6E90H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6EA0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6EB0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6EC0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6ED0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6EE0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H
6EF0H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H 00H
00H

Input queue front pointer is:  6E00H
```

Capítulo 6

Funciones de asignación

Este capítulo describe e ilustra comandos que asignan almacenamiento, restablecen almacenamiento de datos y valores a variables dentro del programa BASIC o desde la línea de comando. La Tabla 6.A lista los mnemónicos correspondientes.

Tabla 6.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Borrar variables, interrupciones y cadenas.	CLEAR	6-1
Borrar interrupciones.	CLEARI	6-2
Borrar todas las pilas.	CLEARS	6-3
Leer datos mediante la instrucción Read.	DATA	6-4
Asignar memoria para variables de matrices.	DIM	6-5
Asignar una variable o un valor de cadena (LET es opcional).	LET	6-6
RESTAURAR el señalador de lectura	RESTORE	6-7

CLEAR

Objetivo:

Use la instrucción CLEAR para poner todas las variables a 0 y restablecer todas las interrupciones y pilas llamadas con BASIC. Esto significa que después que se ejecute la instrucción CLEAR, debe ejecutarse una instrucción ONTIME antes que el módulo confirme las interrupciones del temporizador interno. La captura de ERRORES con la instrucción ONERR tampoco se realiza hasta que se ejecute una instrucción ONERR [ln num].

La instrucción CLEAR no afecta el reloj autónomo que se habilita con la instrucción CLOCK1. La única instrucción del módulo que puede inhabilitar el reloj autónomo es CLOCK0.

La instrucción CLEAR tampoco restablece la memoria que se ha asignado para cadenas, por lo tanto no es necesario introducir la instrucción STRING [expr], [expr] para reasignar memoria para cadenas después de ejecutar la instrucción CLEAR. En general, CLEAR se usa para borrar todas las variables.

Syntax:

CLEAR

Ejemplo:

```
>CLEAR

>LIST
1  REM EXAMPLE PROGRAM
10 DIM A(4)
20 DATA 10,20,30,40
30 FOR I=0 TO 3
40 READ A(I)
50 NEXT I
60 FOR J=O TO 3
70 PRINT A(J)
80 NEXT J

READY
>PRINT A(1),I,J
 0  0  0

>RUN

 10
 20
 30
 40

READY
>PRINT A(1),I,J
 20  4  4

>CLEAR

>PRINT A(1),I,J
 0  0  0
```

CLEARI

Objetivo:

Use la instrucción CLEARI para borrar todas las interrupciones invocadas mediante BASIC. La interrupción ONTIME queda inhabilitada después que se ejecuta la instrucción CLEARI.

CLEARI no afecta el reloj autónomo habilitado por la instrucción CLOCK1. La única instrucción del módulo que puede inhabilitar el reloj autónomo es CLOCK0.

Esta instrucción puede usarse para INHABILITAR instrucciones ONTIME selectivamente durante secciones específicas de su programa BASIC. Se debe ejecutar la instrucción ONTIME otra vez para que se vuelvan a habilitar las interrupciones específicas.

Importante: Cuando la instrucción CLEARI está LISTada, aparecerá como CLEARI.

Sintaxis:

CLEARI

Ejemplo:

```
READY
>CLEARI
```

CLEARS

Objetivo:

Use la instrucción CLEARS para restablecer todas las pilas. El control, argumento y las pilas internas se restablecen a sus valores de inicialización. Se puede usar este comando para restablecer las pilas si se produce un error en una subrutina.

Importante: Cuando la instrucción CLEARS está LISTada, aparece como CLEARS.

Sintaxis:

CLEARS

Ejemplo:

```
READY
>CLEARS
```

DATA

Objetivo:

Use la instrucción DATA para especificar las expresiones que usted puede recuperar con una instrucción READ. Si se usan múltiples expresiones por línea, *debe* separarlas con una coma.

Cada vez que se encuentra una instrucción READ, la siguiente expresión consecutiva en la instrucción DATA es evaluada y asignada a la variable en la instrucción READ. Se pueden colocar instrucciones DATA en cualquier lugar dentro de un programa. Estas no se ejecutan y no causan ningún error. Las instruccones DATA se consideran concatenadas y aparecen como una sola instrucción grande de DATA. Si en algún momento todos los datos se leen y se ejecuta otra instrucción READ, el programa se termina y se imprime el mensaje **`ERROR: NO DATA – IN LINE XX`** al dispositivo de la consola. El módulo regresa al modo de comando.

Sintaxis:

DATA

Ejemplo:

```
>LIST
1  REM EXAMPLE PROGRAM
10 DIM A(4)
20 DATA 10,ASC(A),ASC(C),35.627
30 FOR I=0 TO 3
40 READ A(I)
50 NEXT I
60 FOR J=O TO 3
70 PRINT A(J)
80 NEXT J

READY
>RUN

 10
 65
 67
 35.627
```

Importante: No se puede usar el operador CHR en una instrucción DATA.

DIM

Objetivo:

Use la instrucción DIM para reservar almacenamiento para matrices. En principio se supone que el área de almacenamiento es cero. Las matrices en el módulo BASIC pueden tener sólo una dimensión y el tamaño del conjunto dimensionado no debe exceder de 254 elementos.

Una vez que una variable ha sido dimensionada en un programa, no puede ser dimensionada otra vez. Todo intento de dimensionar de nuevo una matriz producirá un error de dimensión de matriz que obligará al módulo a entrar al modo de comando.

Si se usa una variable matricial que no fue dimensionada por una instrucción DIM, BASIC asigna un valor predeterminado de 10 al tamaño de la matriz. Todas las matrices se ponen a cero cuando se ejecuta el comando RUN, el comando NEW o la instrucción CLEAR.

El número de bytes asignados a una matriz es seis veces la dimensión de la matriz más uno (6 * (dimensión de la matriz + 1)). Por ejemplo, la matriz A (100) necesita 606 bytes de almacenamiento. El tamaño de la memoria generalmente limita el tamaño de una matriz dimensionada.

Sintaxis:

DIM

Ejemplos:

Una sola instrucción DIM puede dimensionar más de una variable.

```
>1  REM EXAMPLE PROGRAM
>10 DIM A(25), C(15), A1(20)
```

Error al intentar redimensionar la matriz.

```
>1  REM EXAMPLE PROGRAM
>10 A(5) = 10 : REM BASIC ASSIGNS DEFAULT OF 10 TO ARRAY A
>20 DIM A(5) : REM ARRAY RE-DIMENSION ERROR
>

READY
>RUN

ERROR: ARRAY SIZE - IN LINE  20

20     DIM A(5) :  REM ARRAY RE-DIMENSION ERROR
---------------X
READY
>
```

LET

Objetivo:

Use la instrucción LET para asignar una variable al valor de una expresión.

Sintaxis:

LET [var] = [expr]

Ejemplos:

```
>1  REM EXAMPLE PROGRAM
>10 LET A = 10*SIN(C)/100
```

```
>1  REM EXAMPLE PROGRAM
>10 LET A = A +1
```

Nótese que el signo = tal como se usa en la instrucción LET no es un operador de igualdad. Es un operador de reemplazo. La instrucción debe leerse A es reemplazada por A más uno. La palabra LET siempre es opcional (ejemplo: **`LET A = 2`** es lo mismo que **`A = 2`**).

Cuando se omite LET, la instrucción LET se llama IMPLIED LET. Usamos la palabra LET para referirnos tanto a la instrucción LET como a la instrucción IMPLIED LET.

La instrucción LET se usa también para asignar variables de cadena:

```
LET $(1)="THIS IS A STRING"
```

o bien

```
LET $(2)=$(1)
```

Antes de asignar cadenas se debe ejecutar la instrucción STRING [expr], [expr], de lo contrario se producirá un error de asignación de memoria que obligará al módulo a entrar al modo de comando.

RESTORE

Objetivo:

Use la instrucción RESTORE para restablecer el señalador de lectura interno al principio de los datos para que pueda ser leído otra vez,

Sintaxis:

RESTORE

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 3
>20 READ A,C
>30 PRINT A,C
>40 NEXT I
>50 RESTORE
>60 READ A,C
>70 PRINT A,C
>80 DATA 10,20,10/2,20/2,SIN(PI),COS(PI)


READY
>RUN

 10        20
 5         10
 0         -1
 10        20
```

Capítulo 7

Funciones de control

Este capítulo describe e ilustra comandos ejecutados dentro del programa BASIC o desde la línea de comando para controlar el reloj interno o el flujo del programa BASIC. La Tabla 7.A lista los mnemónicos correspondientes.

Tabla 7.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Inhabilitar el reloj en tiempo real.	CLOCK0	7-2
Habilitar el reloj en tiempo real.	CLOCK1	7-1
Establecer un lazo condicional “Do”	DO-UNTIL	7-5
Establecer un lazo condicional “Do”	DO-WHILE	7-3
Terminar una ejecución del programa.	END	7-5
Establecer un lazo “For-next”	FOR-TO-(STEP)-NEXT	7-6
Ir al número de línea del programa	GOTO	7-8
Una prueba condicional	IF–THEN–ELSE	7-9
Probar una condición de lazo “for-next”	NEXT	7-10
Un GOTO condicional	ON-GOTO	7-11

CLOCK1

Objetivo:

Use la instrucción CLOCK1 para habilitar el reloj autónomo residente en el módulo BASIC. El operador de función especial TIME se incrementa una vez cada 5 milisegundos después que la instrucción CLOCK1 se ejecuta. La instrucción CLOCK1 usa un TEMPORIZADOR interno para generar una interrupción una vez cada 5 milisegundos. Debido a esto, el operador de función especial TIME tiene una resolución de 5 milisegundos. El operador de función especial TIME cuenta desde 0 a 65535.995 segundos. Después de llegar al conteo de 65535.995 segundos, TIME regresa a cero. Las interrupciones asociadas con la instrucción CLOCK1 hacen que los programas del módulo funciones aproximadamente a 99.6% de la velocidad normal. Esto significa que el tratamiento de interrupciones para el reloj autónomo usa aproximadamente 0.4% del tiempo total de la CPU.

Importante: Esto no incluye tiempo adicional para el manejo de interrupciones ON-TIME del usuario.

Sintaxis:

CLOCK1

Ejemplo:

```
>NEW

>1   REM EXAMPLE PROGRAM
>10  TIME = 0
>15  DBY(71) = 0
>20  CLOCK1
>30  ONTIME 2,100
>40  DO
>50  WHILE TIME < 10
>60  END
>100 PRINT “TIMER INTERRUPT AT – ”,TIME,“ SECONDS”
>110 ONTIME TIME+2,100
>120 RETI

READY
>RUN
TIMER INTERRUPT AT –  2.01  SECONDS
TIMER INTERRUPT AT –  4.015  SECONDS
TIMER INTERRUPT AT –  6.01  SECONDS
TIMER INTERRUPT AT –  8.01  SECONDS
TIMER INTERRUPT AT –  10.01  SECONDS
```

CLOCK0

Objetivo:

Use la instrucción CLOCK0 (cero) para inhabilitar o apagar el reloj autónomo residente en el módulo BASIC. Después que se ejecuta CLOCK0, el operador de función especial TIME no sigue incrementando. CLOCK0 es la única instrucción del módulo que puede inhabilitar el reloj autónomo. CLEAR y CLEARI *no* inhabilitan el reloj autónomo, sólo su interrupción ONTIME asociada.

Importante: CLOCK1 y CLOCK0 son independientes del reloj/calendario.

Sintaxis:

CLOCK0

Ejemplo:

```
READY
>CLOCK0
```

DO-WHILE

Objetivo:

Use la instrucción DO-WHILE para configurar un control de lazo dentro de un programa del módulo. La operación de esta instrucción es similar a la de DO-UNTIL [rel expr]. Todas las instrucciones entre DO y WHILE [rel expr] se ejecutan siempre que la expresión de relación que sigue a la instrucción WHILE sea verdadera. Las instrucciones DO-WHILE se pueden anidar .

La pila de control (pila C) almacena toda información asociada con control de lazo (ejemplo: DO-WHILE, DO-UNTIL, FOR-NEXT y las subrutinas BASIC). La pila de control tiene 157 bytes de longitud. Los lazos DO-WHILE y DO-UNTIL y los comandos GOSUB usan 3 bytes de la pila de control. Los lazos FOR-NEXT usan 17 bytes.

Importante: Un excesivo anidamiento excederá los límites de la pila de control, generará un error y hará que el módulo entre al modo de comando.

Sintaxis:

DO-WHILE [rel expr]

Ejemplos:

DO-WHILE simple

```
>NEW

>1  REM EXAMPLE PROGRAM
>10 DO
>20 A = A + 1
>30 PRINT A
>40 WHILE A < 4
>50 PRINT “DONE”
>60 END

READY
>RUN

 1
 2
 3
 4
DONE

READY
>
```

DO-WHILE anidado

```
>NEW

>1  REM EXAMPLE PROGRAM
>10 A=0 : C=0
>20 DO
>30 A=A+1
>40 DO
>45 C=C+1
>50 PRINT A,C,A*C
>60 WHILE C<>3
>70 C=0
>80 WHILE A<4
>90 END

READY
>RUN
 1  1  1
 1  2  2
 1  3  3
 2  1  2
 2  2  4
 2  3  6
 3  1  3
 3  2  6
 3  3  9

READY
>
```

DO-UNTIL

Objetivo:

Use la instrucción DO-UNTIL para establecer control de lazo dentro de un programa del módulo. Todas las instrucciones entre DO y UNTIL[rel expr] se ejecutan hasta que la expresión de relación que sigue a la instrucción UNITL sea VERDADERA. Los lazos DO-UNTIL se pueden anidar.

La pila de control (pila C) almacena toda información asociada con control de lazo (ejemplo: DO-WHILE, DO-UNTIL, FOR-NEXT y las subrutinas BASIC). La pila de control tiene 157 bytes de largo. Los lazos DO-WHILE y DO-UNTIL y los comandos GOSUB usan 3 bytes de la pila de control. Los lazos FOR-NEXT usan 17 bytes.

Importante: Un excesivo anidamiento excederá los límites de la pila de control, generará un error y hará que el módulo entre al modo de comando.

Sintaxis:

DO-UNTIL [rel expr]

Ejemplos:

DO-UNTIL simple

```
>1  REM EXAMPLE PROGRAM
>10 A=0
>20 DO
>30 A=A+1
>40 PRINT A
>50 UNTIL A=4
>60 PRINT “DONE”
>70 END
>RUN
```

DO-UNTIL anidado

```
>1  REM EXAMPLE PROGRAM
>10 DO
>20 A=A+1
>30 DO
>40 C=C+1
>50 PRINT A,C,A*C
>60 UNTIL C=3
>70 C=0
>80 UNTIL A=3
>90 END
RUN
```

END

Objetivo:

Use la instrucción END para terminar la ejecución del programa. CONT no opera si la instrucción END se usa para terminar la ejecución. **`ERROR : CAN’T CONTINUE`** se imprime en la consola. Siempre incluya una instrucción END para terminar correctamente un programa.

Sintaxis:

END

Ejemplo:

Terminación con la instrucción END

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 4
>20 PRINT I,
>30 NEXT I
>40 END

READY
>RUN

 1  2  3  4
READY
>
```

FOR-TO-(STEP)-NEXT

Objetivo:

Use la instrucción FOR – TO – (STEP) – NEXT para instalar y controlar lazos del programa.

La pila de control (pila C) almacena toda información asociada con control de lazo (ejemplo: DO-WHILE, DO-UNTIL, FOR-NEXT y las subrutinas BASIC). La pila de control tiene 157 bytes de longitud. Los lazos DO-WHILE y DO-UNTIL y los comandos GOSUB usan 3 bytes de la pila de control. Los lazos FOR-NEXT usan 17 bytes.

Importante: Un excesivo anidamiento excederá los límites de la pila de control, generará un error y hará que el módulo entre al modo de comando.

Sintaxis:

FOR [expr] TO [expr] STEP [expr]
.
.
.
NEXT [expr]

Ejemplos:

```
>1  REM EXAMPLE PROGRAM
>5  E=0 : C=10 : D=2
>10 FOR A=E TO C STEP D
>20 PRINT A
>30 NEXT A
>40 END
>RUN


0
2
```

```
4
6
8
10

READY

>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 4
>20 PRINT I,
>30 NEXT I
>40 END

READY
>RUN

 1  2  3  4
```

En el primer ejemplo, como E=0, C=10, D=2, y la instrucción PRINT en la línea 20 se ejecuta 6 veces, los valores de A que se imprimen son 0, 2, 4, 6, 8 y 10. A representa el nombre del índice o contador de lazos. El valor de E es el valor inicial del índice. El valor de C es el valor límite del índice y el valor de D es el incremento del índice.

Si la instrucción STEP y el valor D se omiten, el incremento cambia a 1 de manera predeterminada, por lo tanto STEP es una instrucción opcional. La instrucción NEXT regresa el lazo a su comienzo y añade el valor de D al valor actual del índice. Luego el valor actual del índice es comparado al valor de C, el valor límite del índice.

Si el índice es menor o igual al límite, el control se transfiere a la instrucción que sigue a la instrucción FOR. En el módulo BASIC se permite dar un paso atrás (FOR I = 100 TO 1 STEP–1). La instrucción NEXT siempre está seguida por la variable apropiada. Puede anidar lazos FOR–NEXT hasta 9 veces.

```
>1  REM EXAMPLE PROGRAM
>10 FOR I=1 TO 4
>20 PRINT I,
>30 NEXT I
>40 END
>RUN
>1 2 3 4


READY
```

```
>1  REM EXAMPLE PROGRAM
>10 FOR I=0 TO 8 STEP 2
>20 PRINT I
>30 NEXT I
>40 END
>RUN
0
2
4
6
8

READY
```

GOTO

Objetivo:

Use la instrucción GOTO para que BASIC transfiera el control al número de línea ([ln num]) especificado.

Sintaxis:

GOTO [ln num]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>50 GOTO 100
```

Si existe la línea 100, esta instrucción hace que la ejecución del programa continúe en la línea 100. Si el número de línea 100 no existe, se imprime el mensaje **`ERROR:  INVALID LINE NUMBER`** en el dispositivo de la consola y el módulo BASIC entra al modo de comando.

A diferencia del comando RUN, la instrucción GOTO, si se ejecuta en el modo de comando, no borra el espacio de almacenamiento de variables ni las interrupciones. Sin embargo, si la instrucción GOTO se ejecuta en el modo de comando después que se ha editado una línea, el módulo borra el espacio de almacenamiento de variables y todas las interrupciones BASIC.

IF-THEN-ELSE

Objetivo:

Use la instrucción IF-THEN-ELSE para establecer una prueba condicional.

Sintaxis:

IF [rel expr] THEN instrucción válida ELSE instrucción válida

Ejemplos:

```
>1  REM EXAMPLE PROGRAM
>10 IF A =100 THEN A=0 ELSE A=A+1
```

Después de la ejecución de la línea 10, si A es igual a 100, entonces a A se le asigna un valor de 0. Si A no es igual a 100, a A se le asigna un valor de A+1. Si desea transferir el control a números de línea diferentes usando la instrucción IF, puede omitir la instrucción GOTO. Los siguientes ejemplos dan los mismos resultados:

```
>20 IF INT(A)<10 THEN GOTO 100 ELSE GOTO 200
          o
>20 IF INT(A)<10 THEN 100 ELSE 200
```

Puede reemplazar la instrucción THEN con cualquier instrucción válida del módulo BASIC, tal como se muestra a continuación:

```
>30 IF A<>10 THEN PRINT A ELSE 10
>30 IF A<>10 PRINT A ELSE 10
```

Puede ejecutar instrucciones múltiples después de THEN o ELSE si usa un signo de dos puntos para separarlas.

Ejemplo:

```
>30 IF A<>10 THEN PRINT A : GOTO 150 ELSE 10
>30 IF A<>10 PRINT A : GOTO 150 ELSE 10
```

En estos ejemplos, si A no es igual a 10, ambos PRINT A y GOTO 150 se ejecutan. Si A es igual a 10, el control pasa a 10.

Puede omitir la instrucción ELSE. Si omite la instrucción ELSE, el control pasa a la siguiente instrucción.

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>20 IF A=10 THEN 40
>30 PRINT A
```

En este ejemplo, si A es igual a 10, el control pasa a la línea número 40. Si A no es igual a 10, la líne número 30 se ejecuta.

NEXT

Objetivo:

Use la instrucción NEXT para volver el FOR-TO-(STEP)-NEXT al principio del lazo y añadir el valor del incremento de índice al índice. El valor actual de índice se compara con el límite del índice para determinar si se debe realizar otro lazo.

Sintaxis:

NEXT

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>5  E=0 : C=10 : D=2
>10 FOR A=E TO C STEP D
>20 PRINT A
>30 NEXT A
>40 END
>RUN

0
2
4
6
8
10

READY
>

>NEW

>1  REM EXAMPLE PROGRAM
>10 FOR I = 0 TO 8 STEP 2
>20 PRINT I
>30 NEXT I
>40 END

>RUN

 0
 2
 4
 6
 8
```

ON-GOTO

Objetivo:

Use la instrucción ON-GOTO para transferir el control de la(s) línea(s) especificada(s) por la instrucción GOTO cuando el valor de la expresión que sigue a la instrucción ON se encuentra en el programa BASIC.

Sintaxis:

ON [expr] GOTO [ln num]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 ON Q GOTO 100,200,300
```

El control se transfiere a la línea 100 si Q es igual a 0 y luego a la línea 200 si Q es igual a 1. Si Q es igual a 2, el control se transfiere a la línea número 300, y así sucesivamente. Todos los comentarios que se aplican a GOTO se aplican a la instrucción ON. Si Q es menor que cero, se genera un mensaje **ERROR: BAD ARGUMENT** y el módulo BASIC entra al modo de comando. Si Q es mayor que el número de línea de la lista que sigue a la instrucción GOTO, se genera un error **ERROR: BAD SYNTAX**. La instrucción ON-GOTO proporciona opciones de bifurcación condicional dentro del programa del módulo BASIC.

Capítulo 8

Funciones de soporte de interrupción y control de ejecución

Este capítulo describe e ilustra los comandos que controlan el flujo de datos y la transferencia de programas entre la ROM y la RAM dentro del programa BASIC o desde la línea de comando. La Tabla 8.A lista los mnemónicos correspondientes.

Tabla 8.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Habilitar la capacidad de interrupción cuando se recibe un paquete DF1.	CALL 16	8-2
Inhabilitar la capacidad de interrupción del paquete DF1.	CALL 17	8-2
Habilitar la capacidad de interrupción del procesador SLC.	CALL 20	8-3
Inhabilitar la capacidad de interrupción del procesador SLC.	CALL 21	8-4
Generar una interrupción al procesador SLC.	CALL 26	8-5
Iniciar transacciones definidas por CALL 27, 28, 122 y 123.	CALL 38	8-6
La transferencia de programa de ROM a RAM.	CALL 70	8-8
La transferencia de programa de ROM/RAM a ROM.	CALL 71	8-9
Retornar a RAM/ROM.	CALL 72	8-10
Ejecutar una subrutina.	GOSUB	8-11
Ir al número de línea cuando se detecta un error.	ONERR	8-12
Un GOSUB condicional.	ON-GOSUB	8-13
Generar una interrupción cuando TIME es igual o mayor que el número de argumento–línea ONTIME.	ONTIME	8-14
Aplicar POP a la pila de argumentos a variables.	POP	8-17
Aplicar PUSH a expresiones en la pila de argumentos.	PUSH	8-15
Retornar desde una interrupción.	RETI	8-18
RETORNAR desde una subrutina	RETURN	8-18
Interrumpir la ejecución del programa.	STOP	8-20

CALL 16 - Habilitación de interrupción de paquete DF1

Objetivo:

Use CALL 16 para habilitar la capacidad de interrupción del paquete DF1. Un argumento recibe PUSH y ningún argumento recibe POP. El argumento de entrada es el número de línea BASIC del comienzo de la rutina de interrupción al cual debe saltar el programa cuando se recibe un paquete DF1 válido en el búfer del puerto PRT2. Usted debe procesar el paquete dentro de la rutina de interrupción. Una RET1 ejecutada dentro de la rutina le hará regresar al punto anterior a donde se produjo la interrupción en el programa. Este comando no tiene ningún efecto si el protocolo DF1 no está habilitado (CALL 108). Además, el puente JW4 debe estar en una posición que habilite DF1 para el puerto PRT2.

Una vez que este CALL está habilitado, el puerto PRT2 es revisado por el procesador al final de cada línea del código BASIC para determinar si hay un mensaje DF1 recibido.

Si el paquete DF1 llega debido a CALL 122 o CALL 123 cuando CALL 16 está habilitado, usted recibirá la interrupción del paquete DF1, pero el paquete DF1 habrá sido retirado del búffer de entradas.

Las interrupciones se inhabilitan cuando el módulo BASIC está en el modo de comando. CALL 16 inhabilitado es el valor predeterminado del módulo cuando entra al modo de marcha. CALL 16 debe volverse a ejecutar cada vez que se entra al modo de marcha.

Sintaxis:

PUSH [número de línea BASIC]
CALL 16

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE DF1 PACKET INTERRUPT
>20  PUSH 800: REM LINE NUMBER OF START OF DF1 INTERRUPT
     ROUTINE
>30  CALL 16
>800 (BEGINNING OF INTERRUPT ROUTINE)
 :   (PROCESS THE PACKET)
>850 RETI
```

CALL 17 - Inhabilitación de interrupción de paquete DF1

Objetivo:

Use CALL 17 para inhabilitar la capacidad de interrupción del paquete DF1 habilitada con CALL 16. Esta rutina no tiene argumentos de entrada ni de salida.

Sintaxis:

CALL 17

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM DISABLE DF1 PACKET INTERRUPT ENABLED WITH CALL 16
>20 CALL 17
```

CALL 20 - Habilitación de interrupción del procesador

Objetivo:

Use CALL 20 para permitir que el procesador interrumpa al módulo BASIC. Un argumento recibe PUSH y ningún argumento recibe POP. PUSH es el número de línea BASIC del comienzo de la rutina de interrupción al cual debe saltar el programa, cuando la palabra 0, bit 15 en la tabla de imagen de salida CPU conmuta de un valor bajo a un valor alto. El módulo BASIC detecta esta transición automáticamente y salta a una rutina de interrupción. Una RETI ejecutada dentro de la rutina de interrupción le hará regresar al punto anterior a donde se produjo la interrupción en el programa del módulo BASIC.

El módulo BASIC monitoriza la palabra 0, bit 15 del archivo de salida CPU al final de cada línea BASIC y genera la interrupción si el bit se hace alto. La CPU debe mantener bajo el bit de petición de interrupción durante por lo menos 10 milisegundos antes de solicitar servicio de interrupción. El bit debe mantenerse alto durante por lo menos un escán para que el bit pueda ser detectado por el módulo BASIC.

Si se detecta otra interrupción antes que la anterior haya recibido servicio, la nueva interrupción es marcada como pendiente. Sólo una interrupción está pendiente a la vez.

Las interrupciones se inhabilitan cuando el módulo BASIC está en el modo de comando. CALL 20 inhabilitado es el valor predeterminado del módulo cuando entra al modo de marcha. CALL 20 debe volverse a ejecutar cada vez que se entra al modo de marcha.

Sintaxis:

PUSH [número de línea BASIC]
CALL 20

Ejemplo:

```
>1    REM EXAMPLE PROGRAM
>10   REM ENABLE PROCESSOR INTERRUPTS
>20   PUSH 1000 : REM LINE NUMBER OF START OF PROCESSOR
      INTERRUPT ROUTINE
>30   CALL 20
>1000 (BEGINNING OF THE PROCESSOR INTERRUPT ROUTINE)
 :
>1050 RETI
```

CALL 21 - Inhabilitación de interrupción del procesador

Objetivo:

Use CALL 21 para inhabilitar la capacidad de interrupción del procesador habilitada con CALL 20. Esta rutina no tiene argumentos de entrada ni de salida.

Sintaxis:

CALL 21

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM DISABLE PROCESSOR INTERRUPTS ENABLED WITH CALL 20
>20 CALL 21
```

CALL 26 - Interrupción del módulo BASIC

Objetivo:

Use CALL 26 para generar una interrupción de los procesadores SLC 5/02™ y de mayor capacidad. Ningún argumento recibe PUSH y un argumento recibe POP. POP muestra el estado del procesador SLC. Cuando se ejecuta este CALL, el módulo BASIC emite una interrupción de evento de E/S para interrumpir el ciclo de operación normal del procesador con el fin de escanear una subrutina especificada. Esta interrupción hace que el procesador SLC ejecute el archivo de subrutina de interrupción configurado en la configuración de ranuras del módulo BASIC (archivo con numeración ISR). El módulo BASIC permanece en esta rutina de CALL hasta que se recibe una confirmación de interrupción desde el procesador. CALL 26 debe ejecutarse en el programa BASIC cada vez que el procesador SLC va a ser interrumpido.

Este CALL no tiene ningún efecto si el procesador SLC no está en el modo de marcha.

Después que el módulo BASIC emite el CALL, puede tomar hasta 5 milisegundos para que se produzca la ejecución de la interrupción.

POP muestra el estado del procesado SLC:

- 0 = El procesador SLC confirma que la interrupción puede no haber ejecutado la rutina de interrupción todavía
- 1 = El procesador SLC canceló la interrupción
- 2 = El procesador SLC no está en el modo de marcha
- 3 = El procesador SLC 500 compacto y el procesador SLC 5/01™ no pueden aceptar interrupciones

Sintaxis:

CALL 26
POP[estado del procesador SLC]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM ENABLE BASIC MODULE INTERRUPT TO THE SLC PROCESSOR
    TO EXECUTE THE ISR FILE
>20 CALL 26
>30 POP S : REM SLC PROCESSOR STATUS
```

CALL 38 - Reinicio de ONERR expandido

Objetivo:

Use CALL 38 para expandir el tipo de errores atrapados y manejados por la función ONERR. Un argumento recibe PUSH y ningún argumento recibe POP. El reinicio de ONERR permite que el módulo salte a una rutina de manejo de error cuando se encuentra un overflow aritmético, una división entre cero y un mal argumento. Todos los otros errores hacen que el módulo BASIC entre al modo de comando y detenga la ejecución del programa. Cuando CALL 38 está habilitado, todos los errores, excepto los fallos de hardware específico (temporizador de control (watchdog), fallo de RAM, etc.) hacen que el programa entre a la rutina de fallo definida en la función ONERR, en vez de regresar al modo de comando. Esta rutina puede usarse para restablecer el error y continuar la operación normal del programa. Si se produce un error que causa un reinicio, las pilas se borran y las variables y puertos no se reinicializan. Este CALL no se hace efectivo hasta que se ejecute el comando ONERR dentro del programa.

PUSH determina si esta función ONERR expandida se habilita o se inhabilita, tal como se muestra a continuación:

- 0 = Inhabilita el reinicio ONERR expandido
- 1 (o cualquier otro número) = Habilita el reinicio expandido

Este CALL se restablece cuando el módulo BASIC regresa al modo de comando. CALL 38 debe volverse a ejecutar cada vez que se entra al modo de marcha.

Sintaxis:

PUSH [0 ó 1]
CALL 38

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE EXPANDED ONERR FUNCTION
>20  ONERR 160
>30  PUSH 1
>40  CALL 38
>50  CALL 53: REM GET DATA FROM OUTPUT IMAGE
>60  PUSH 201: REM ADDRESS OF SECOND WORD IN BUFFER
>70  CALL 14: REM GET DATA FROM INPUT BUFFER
>80  POP X: REM VALUE FROM INPUT BUFFER
>90  A=(X*2.499733)–8191.625
>100 PUSH A: REM RESULT OF ABOVE CALCULATION
>110 PUSH 201: REM WORD NUMBER OF BASIC OUTPUT BUFFER
>120 CALL 24: REM BASIC FLOATING POINT TO 16–BIT SIGNED
     INTEGER
>130 CALL 54: REM OUTPUT BUFFER TO SLC INPUT FILE
>140 POP Y: REM SLC PROCESSOR STATUS
>150 IF (Y<>0) THEN PRINT “PROCESSOR NOT IN RUN MODE”
>160 GOTO 50
>170 PRINT “ERROR CODE WAS”,XBY(257) : REM BEGINNING OF
     ONERR ROUTINE
>180 GOTO 50
```

El error en el ejemplo anterior es un POP ausente para CALL 53. El POP ausente causará un error de pila A y normalmente pondría al procesador en el modo de comando. Si se produce esta situación, imprima el código de error y continúe ejecutando el programa.

CALL 70 - Transferencia de programa de ROM a RAM

Objetivo:

Use CALL 70 para mover la ejecución del programa desde un programa ROM en funcionamiento hasta el principio del programa RAM. Ningún argumento recibe ni PUSH ni POP.

Importante: La primera línea del programa RAM no se ejecuta. Recomendamos que esto se haga un comentario.

Sintaxis:

CALL 70

Ejemplo:

```
READY
>LIST
1  REM EXAMPLE PROGRAM
10 REM SAMPLE ROM PROGRAM FOR CALL 70
20 PRINT “NOW EXECUTING ROM 5”
30 CALL 70 :  REM GO EXECUTE RAM
40 END

READY
>RUN

NOW EXECUTING ROM 5
NOW EXECUTING RAM

READY
>LIST
1  REM EXAMPLE PROGRAM
10 REM SAMPLE RAM PROGRAM FOR CALL 70
20 PRINT “NOW EXECUTING RAM”
30 END

READY
```

CALL 71 - Transferencia de programa de ROM/RAM a ROM

Objetivo:

Use CALL 71 para pasar de un programa ROM o RAM en funcionamiento al comienzo de cualquier programa ROM disponible. Un argumento recibe PUSH (el programa ROM). Ninguno recibe POP. Aparece un error de programa y usted entra al modo de comando si el número ROM no existe.

Importante: La primera línea del programa ROM no se ejecuta. Recomendamos que esto se haga un comentario.

Sintaxis:

PUSH [número de programa ROM]
CALL 71

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM THIS ROUTINE WILL CALL AND EXECUTE A ROM ROUTINE
>20 INPUT “ENTER ROM ROUTINE TO EXECUTE”,N
>30 PUSH N
>40 CALL 71
>50 END

>RUN

ENTER ROM ROUTINE TO EXECUTE 4
```

Ahora usted está ejecutando ROM 4, si existe. Si la rutina ROM solicitada no existe, el resultado es:

```
PROGRAM NOT FOUND.
READY
>
```

CALL 72 - Retorno a RAM/ROM

Objetivo:

Use CALL 72 para regresar a la rutina de ROM o RAM que invocó a esta rutina ROM o RAM. La ejecución empieza en la línea siguiente a la línea que invocó (CALL) a la rutina. Ningún argumento recibe ni PUSH ni POP. Esta rutina funciona a un nivel de profundidad. El control del programa vuelve a la línea que sigue al CALL en el programa anterior.

Importante: Es necesario que haya una línea siguiente en la rutina ROM o RAM, de lo contrario pueden producirse eventos impredecibles que podrían destruir el contenido de la RAM. Por esta razón, siempre asegúrese de que haya por lo menos una instrucción END después de CALL 70 ó 71.

Sintaxis:

CALL 72

Ejemplo:

Programa en ROM 1:

```
>1  REM EXAMPLE PROGRAM
>10 REM SAMPLE PROG FOR CALL 72
>20 PRINT “NOW EXECUTING ROM 1”
>30 PUSH 3
>40 CALL 71 : REM EXECUTE ROM 3 THEN RETURN
>50 PRINT “EXECUTING ROM 1 AGAIN”
>60 END
```

Programa en ROM 3:

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “NOW EXECUTING ROM 3”
>20 CALL 72
>30 END
```

Con ROM 1 seleccionado:

```
>RUN

NOW EXECUTING ROM 1
NOW EXECUTING ROM 3
EXECUTING ROM 1 AGAIN

READY
>
```

GOSUB

Objetivo:

Use la instrucción GOSUB para hacer que el módulo BASIC transfiera el control del programa al número de línea [ln num] después de la instrucción GOSUB. Además, la instrucción GOSUB guarda la ubicación de la instrucción que sigue a GOSUB en la pila de control, de manera que usted pueda ejecutar una instrucción RETURN para devolver el control a la instrucción siguiente a la instrucción GOSUB más recientemente ejecutada. Se puede anidar la instrucción GOSUB hasta 9 veces.

La pila de control (pila C) almacena toda la información asociada con control de lazo (ejemplo: DO-WHILE, DO-UNTIL, FOR-NEXT y subrutinas BASIC). La pila de control tiene 157 bytes de longitud. Los lazos DO-WHILE y DO-UNTIL y los comandos GOSUB usan 3 bytes de la pila de control. Los lazos FOR-NEXT usan 17 bytes.

Importante: El anidamiento excesivo excederá los límites de la pila de control, generará un error y hará que el módulo entre al modo de comando.

Sintaxis:

GOSUB [ln num]

Ejemplos:

Subrutina simple

```
READY
>1   REM EXAMPLE PROGRAM
>10  FOR I = 1 TO 5
>20  GOSUB 100
>30  NEXT I
>40  END
>100 PRINT I
>110 RETURN

READY
>RUN

 1
 2
 3
 4
 5

READY
>NEW
```

Subrutina anidada

```
>1   REM EXAMPLE PROGRAM
>10  FOR I = 1 TO 3
>20  GOSUB 100
>30  NEXT I
>40  END
>100 REM USER SUBROUTINE HERE
>105 PRINT I,
>110 GOSUB 200
>120 RETURN
>200 REM START OF NESTED SUBROUTINE
>210 PRINT I*I
 220 RETURN

READY
>RUN

 1  1
 2  4
 3  9

READY
>
```

ONERR

Objetivo:

Use la instrucción ONERR para tratar errores aritméticos, si ocurren, durante la ejecución del programa. La instrucción ONERR sólo captura errores de overflow aritmético, underflow aritmético, división entre cero y mal argumento. Los demás errores no son capturados y obligan al módulo BASIC a entrar en el modo de comando. Si se produce un error aritmético después que se ha ejecutado la instrucción ONERR, el intérprete del módulo pasa el control al número de línea [ln num] que sigue a la instrucción ONERR. La condición de error se manejará según la aplicación. El comando ONERR no captura datos erróneos introducidos durante una instrucción de entrada. Esto da lugar al mensaje **`TRY AGAIN`** o al mensaje **`EXTRA IGNORED`**. Para obtener información sobre funcionalidad ONERR expandida, consulte CALL 38 en la página 8-6.

Después de ejecutar la instrucción ONERR se puede determinar qué tipo de error ha ocurrido examinando la ubicación de memoria externa 257 (101H).

Los códigos de error son:

- **ERROR CODE = 10–DIVIDE BY ZERO**
- **ERROR CODE = 20–ARITH. OVERFLOW or ARITH.UNDERFLOW**
- **ERROR CODE = 40–BAD ARGUMENT**

Se puede examinar esta ubicación usando una instrucción XBY(257).

Sintaxis:

ONERR [ln num]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  ONERR 500
>20  FOR I = 5 TO  0 STEP –1
>30  PRINT 1/I
>40  NEXT I
>50  END
>500 PRINT “ERROR CODE WAS ”,XBY(257)
>510 END

READY
>RUN

 .2
 .25
 .33333333
 .5
 1
ERROR CODE WAS  10

READY
>
```

Una instrucción GOTO puede reemplazar a la instrucción END en este ejemplo para proporcionar un método de recuperación de error programado por el usuario.

ON-GOSUB

Objetivo:

Use la instrucción ON-GOSUB para transferir el control a la(s) línea(s) especificada(s) por la instrucción GOSUB cuando el valor de la expresión que sigue a la instrucción ON se encuentra en el programa BASIC.

Sintaxis:

ON [expr] GOSUB [ln num], [ln num],...[ln num]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 ON Q GOSUB 100,200,300
```

Si Q es igual a 0, el control se transfiere al número de línea 100. Si Q es igual a 1, el control se transfiere al número de línea 200. Si Q es igual a 2, el control se transfiere al número de línea 300, y así sucesivamente. Todos los comentarios que se aplican a GOSUB se aplican a la instrucción ON. Si Q es menor que cero, se genera el mensaje **ERROR: BAD ARGUMENT**. Si Q es mayor que la lista de números de línea que sigue a la instrucción GOSUB, se genera el mensaje **ERROR: BAD SINTAX**. La instrucción ON-GOSUB proporciona opciones de bifurcación condicional dentro del programa del módulo BASIC.

ONTIME

Objetivo:

Use la instrucción ONTIME [expr], [ln num] para compensar la incompatibilidad entre los temporizadores/contadores en el microprocesador y el módulo BASIC. Su módulo BASIC puede procesar una línea en milisegundos, mientras que los temporizadores/contadores en el microprocesador operan en microsegundos. La instrucción ONTIME genera una interrupción cada vez que el operador de función especial TIME es igual o mayor que la expresión que sigue a la instrucción ONTIME.

Sólo la parte entera de TIME se compara con la parte entera de la expresión que da segundos. Esta comparación se realiza al final (CR o bien :) de cada línea de BASIC. La interrupción fuerza un GOSUB en el número de línea [ln num] que sigue a la expresión [expr] en la instrucción ONTIME.

La instrucción ONTIME no interrumpe un comando de entrada ni una rutina de CALL. Dado que la instrucción ONTIME usa el operador de función especial TIME, se debe ejecutar la instrucción CLOCK1 para que ONTIME funcione. Si no se ejecuta CLOCK1, el operador de función especial, TIME, no se incrementa.

Sintaxis:

ONTIME [expr], [ln num]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  TIME = 0
>15  DBY(71) = 0
>20  CLOCK1
>30  ONTIME 2,100
>40  DO
>50  WHILE TIME < 10
>60  CLOCK0
>70  END
>100 PRINT “TIMER INTERRUPT AT – ”,TIME, “ SECONDS”
>110 ONTIME TIME+2,100
>120 RETI

READY
>RUN

TIMER INTERRUPT AT –  2.01  SECONDS
TIMER INTERRUPT AT –  4.005  SECONDS
TIMER INTERRUPT AT –  6.015  SECONDS
TIMER INTERRUPT AT –  8.01  SECONDS
TIMER INTERRUPT AT –  10.01  SECONDS
```

En el ejemplo anterior, el tiempo impreso tiene un retardo de 0.01 segundos con respecto al tiempo que supuestamente tendría que imprimirse. Esto se debe al terminal usado en el ejemplo operando a 19200 baudios, lo cual causa un retardo de 0.01 segundos en la impresión.

Para ejecutar la interrupción ONTIME a una fracción de segundo, use DBY(71) = X donde X = 0 a 200. Cada conteo representa un intervalo de tiempo de 5 milisegundos.

PUSH

Objetivo:

Use la instrucción PUSH para colocar expresiones aritméticas o expresiones en la pila de argumentos del módulo BASIC. Esta instrucción evalúa la expresión aritmética, o expresiones, que siguen a la instrucción PUSH y luego las coloca en secuencia en la pila de argumentos.

Las instrucciones PUSH y POP proporcionan un modo sencillo de pasar parámetros a las rutinas CALL. Además, las instrucciones PUSH y POP se usan para pasar parámetros a subrutinas BASIC y para intercambiar (SWAP) variables. El último valor empujado (PUSH) a la pila de argumentos es el primer valor sacado (POPped) de la pila de argumentos.

Usted puede empujar (push) más de una expresión a la pila de argumentos usando una sola instrucción PUSH con múltiples expresiones ([expr], [expr],.[expr]). Cada expresión debe estar seguida de una coma. El último valor empujado (PUSH) a la pila de argumentos es la última expresión [expr] encontrada en la instrucción Push.

Importante: La pila de argumentos puede contener hasta 33 números de punto (coma) flotante antes de tener un overflow.

Sintaxis:

PUSH [expr], [expr],.[expr]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 A = 10
>20 C = 20
>30 PRINT “A = ”,A,“ AND C = ” C
>40 PUSH A,C
>50 POP A,C
>60 PRINT “A = ”,A,“  AND C = ”,C
>70 END

READY
>RUN

A =  10  AND C =  20
A =  20  AND C =  10

READY
>

>NEW

>1  REM EXAMPLE PROGRAM
>10 PUSH 0
>20 CALL 14
>30 POP W
>40 PRINT W
>50 END

READY
>RUN

 0

READY
>
```

POP

Objetivo:

Use la instrucción POP para sacar valores de la pila de argumentos del módulo BASIC. El valor en la parte superior de la pila de argumentos es asignado a la variable que sigue a la instrucción POP y la pila de argumentos recibe un POP (ejemplo: se incrementa en un valor de 6). Se puede colocar valores en la pila usando la instrucción PUSH.

Importante: Si se ejecuta una instrucción POP y no hay ningún número en la pila de argumentos, se produce un error de pila A (A–Stack error) y el módulo BASIC entra al modo de comando.

Se puede sacar (pop) más de una variable de la pila de argumentos usando una sola instrucción POP con múltiples variables ([var], [var],[var]). Cada expresión debe estar seguida de una coma.

Sintaxis:

POP [var], [var],.....[var]

Ejemplo:

Vea la instrucción PUSH.

Las instrucciones PUSH y POP se pueden usar para minimizar problemas de la variable GLOBAL. Estos problemas son causados por el programa principal y todas las subrutinas del programa principal que usan los mismos nombres de variables (ejemplo: GLOBAL VARIABLES). Si las mismas variables que se usan en una subrutina no pueden usarse en el programa principal, se pueden reasignar un número de variables (ejemplo: A=Q) antes de ejecutar una instrucción GOSUB.

Si se reservan algunos nombres de variables exclusivamente para subrutinas (S1, S2) y se pasan variables a la pila tal como se muestra en el ejemplo anterior, se puede evitar cualquier problema de variable GLOBAL en el módulo BASIC.

Las instrucciones PUSH y POP aceptan variables dimensionadas A(4) y S1(12) así como variables escalares. Esto es útil cuando se se usan rutinas CALL en las que se debe aplicar PUSH y POP a grandes cantidades de datos.

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>40 FOR I=1 TO 64
>50 PUSH I
>60 CALL 10
>70 POP A(I)
>80 NEXT I
```

RETI

Objetivo:

Use la instrucción RETI para salir de una interrupción (ONTIME, CALL 16, o CALL 20) procesada en un programa del módulo BASIC. La instrucción RETI funciona igual que la instrucción RETURN, excepto que también borra un indicador de interrupción de software, de manera que las interrupciones pueden ser confirmadas de nuevo. Si no se ejecuta la instrucción RETI en el procedimiento de interrupción, todas las interrupciones futuras serán ignoradas.

Sintaxis:

RETI

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  TIME=0 : CLOCK1 : ONTIME 2, 100 : DO
>20  WHILE TIME<10 : END
>100 PRINT “TIMER INTERRUPT AT -”, TIME,“ SECONDS”
>110 ONTIME TIME+2, 100 : RETI
>RUN

TIMER INTERRUPT AT - 2.045 SECONDS
TIMER INTERRUPT AT - 4.045 SECONDS
TIMER INTERRUPT AT - 6.045 SECONDS
TIMER INTERRUPT AT - 8.045 SECONDS
TIMER INTERRUPT AT - 10.045 SECONDS

READY
```

RETURN

Objetivo:

Use la instrucción RETURN para devolver el control a la instrucción que sigue a la instrucción GOSUB STATEMENT ejecutada más recientemente. Use un “return” para cada GOSUB para evitar un overflow de la pila de control. Esto significa que una subrutina llamada a través de la instrucción GOSUB puede invocar a otra subrutina con otra instrucción GOSUB.

Sintaxis:

RETURN

Ejemplos:

Subrutina simple

```
>1   REM EXAMPLE PROGRAM
>10  FOR I = 1 TO 5
>20  GOSUB 100
>30  NEXT I
>40  END
>100 PRINT I
>110 RETURN

READY
>RUN

 1
 2
 3
 4
 5

READY
>
```

Subrutina anidada

```
>1   REM EXAMPLE PROGRAM
>10  FOR I = 1 TO 5
>20  GOSUB 100
>30  NEXT I
>40  END
>100 PRINT I,
>110 GOSUB 200
>120 RETURN
>200 PRINT I*I,
>210 GOSUB 300
>220 RETURN
>300 PRINT I*I*I
>310 RETURN

READY
>RUN

 1  1  1
 2  4  8
 3  9  27
 4  16  64
 5  25  125

READY
>
```

STOP

Objetivo:

Use la instrucción STOP para detener la ejecución de un programa en puntos específicos del programa. Después de haber detenido (STOP) un programa, se pueden ver o modificar las variables. Se puede continuar la ejecución del programa con el comando CONT. El objetivo de la instrucción STOP es permitir una fácil depuración del programa.

Sintaxis:

STOP

Ejemplo:

```
1
STOP – IN LINE 40

>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 100
>20 PRINT I
>30 STOP
>40 NEXT I

READY
>RUN

 1
STOP – IN LINE  40
READY
>CONT

 2
STOP – IN LINE  40
READY
>CONT

 3
STOP – IN LINE  40
READY
>CONT

 4
STOP – IN LINE  40
READY
>
```

Nótese que el número de línea impreso después de la ejecución de la instrucción STOP es el número de línea que sigue a la instrucción STOP, no el número de línea que contiene la instrucción STOP.

Capítulo 9

Funciones matemáticas y de conversión del backplane

Este capítulo describe e ilustra comandos que convierten números enteros a números de punto (coma) flotante BASIC y viceversa. Este capítulo también describe e ilustra comandos que transfieren datos desde el búfer de salidas del módulo BASIC hasta la imagen de entrada del procesador o desde la imagen de salida al búfer de entrada del módulo BASIC. Estos comandos pueden usarse dentro del programa BASIC o desde la línea de comando. La Tabla 9.A lista los mnemónicos correspondientes.

Tabla 9.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Convertir un entero con signo de 16 bits a punto (coma) flotante BASIC.	CALL 14	9-1
Convertir un entero sin signo de 16 bits a punto (coma) flotante BASIC.	CALL 15	9-2
Convertir punto (coma) flotante BASIC a entero con signo de 16 bits.	CALL 24	9-3
Convertir punto (coma) flotante BASIC a binario de 16 bits	CALL 25	9-3

CALL 14 - Entero con signo de 16 bits a punto (coma) flotante BASIC

Objetivo:

Use CALL 14 para convertir un número entero con signo de 16 bits del controlador SLC 500 a un número de punto (coma) flotante BASIC. El argumento de entrada es el número de dirección (0 a 207) de la palabra en el búfer de entrada del módulo BASIC a ser convertido. El argumento de salida es el valor convertido.

Sintaxis:

PUSH [número de palabra del búfer de entradas del módulo BASIC]
CALL 14
POP [valor convertido]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>20 PUSH 0 : REM CONVERT 1ST WORD OF BASIC INPUT BUFFER
>30 CALL 14 : REM DO 16–BIT SIGNED TO F.P. CONVERSION
>40 POP W : REM GET CONVERTED VALUE
>50 PRINT W
>RUN

 0

READY
>
```

CALL 15 - Entero sin signo de 16 bits a punto (coma) flotante BASIC

Objetivo:

Use CALL 15 para convertir un número entero sin signo de 16 bits del controlador SLC 500 a un número de punto (coma) flotante BASIC. El argumento de entrada es el número de dirección (0 a 207) de la palabra en el búfer de entrada del módulo BASIC a ser convertido. El argumento de salida es el valor convertido.

Sintaxis:

PUSH [número de palabra del búfer de entradas del módulo BASIC]
CALL 15
POP [valor convertido]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>50 PUSH 9 : REM CONVERT 10TH WORD OF BASIC INPUT BUFFER
>60 CALL 15 : REM DO 16–BIT UNSIGNED INTEGER TO
    F.P. CONVERSION
>70 POP W : REM GET CONVERTED VALUE
>80 PRINT W
>RUN

 0

READY
>
```

CALL 24 - Punto (coma) flotante BASIC a entero con signo de 16 bits

Objetivo:

Use CALL 24 para convertir un número en punto (coma) flotante del módulo BASIC a un entero de 16 bits con signo y colocar el resultado en el búfer de salidas del módulo BASIC. El primer valor empujado (PUSH) es la variable de datos. El segundo valor empujado es el número de dirección (0 a 207) de la palabra en el búfer de salida del módulo BASIC.

Importante: Si se intenta escribir a la palabra 200 del búfer de salidas BASIC, aparecerá un mensaje de error y el módulo regresará al modo de comando. Los bits de la palabra 200 están definidos.

La parte fraccionaria del valor en punto (coma) flotante del módulo BASIC se trunca. Si el valor en punto (coma) flotante del módulo BASIC es menor que -32768, el valor colocado en el búfer de salidas del módulo BASIC es -32768. Si el valor en punto (coma) flotante del módulo BASIC es mayor que +32767, el valor colocado en el búfer de salidas del módulo BASIC es +32767. El programador es responsable de verificar el rango de los números antes de efectur la conversión.

Sintaxis:

PUSH [valor que se va a convertir]
PUSH [número de palabra del búfer de salidas del módulo BASIC]
CALL 24

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 W = 17
>40 PUSH W : REM THE VALUE TO BE CONVERTED
>50 PUSH 0 : REM 1ST WORD OF BASIC OUTPUT BUFFER
>60 CALL 24 : REM DO THE F.P. TO 16–BIT SIGNED CONVERSION

READY
>
```

CALL 25 - Punto (coma) flotante BASIC a binario de 16 bits

Objetivo:

Use CALL 25 para convertir un valor en punto (coma) flotante del módulo BASIC entre 0 y 65535 a su representación binaria de 16 bits. El valor resultante se almacena en el búfer de salidas del módulo BASIC. El primer valor empujado (PUSH) son los datos que se van a convertir. El segundo valor empujado es el número de dirección (0 a 207) de la palabra en el búfer de salidas del módulo BASIC.

Importante: Si se intenta escribir a la palabra 200 del búfer de salidas BASIC, aparecerá un mensaje de error y el módulo regresará al modo de comando. Los bits de la palabra 200 están definidos.

La parte fraccionaria del valor en punto (coma) flotante del módulo BASIC se trunca. Si el valor en punto (coma) flotante del módulo BASIC es menor que 0, el valor colocado en el búfer de salidas del módulo BASIC es 0. Si el valor es mayor que +65535, el valor colocado en el búfer de salidas del módulo BASIC es +65535. El programador es responsable de verificar el rango de los números antes de efectur la conversión.

Sintaxis:

PUSH [valor a ser convertido]
PUSH [número de palabra del búfer de salidas del módulo BASIC]
CALL 25

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>40 PUSH 9E+1 : REM THE VALUE TO BE CONVERTED
>50 PUSH 0 : REM 1ST WORD OF BASIC OUTPUT BUFFER
>60 CALL 25 : REM DO F.P. TO 16–BIT BINARY CONVERSION
>
READY
>
```

Capítulo 10

Funciones del reloj/calendario

Este capítulo describe e ilustra los comandos que establecen y muestran el reloj en tiempo real/calendario dentro del programa BASIC o desde la línea de comando. La Table 10.A lista los mnemónicos correspondientes.

Table 10.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Ajustar la hora del reloj/calendario (hora, minuto, segundo).	CALL 40	10-2
Ajustar la fecha del reloj/calendario (día, mes, año).	CALL 41	10-3
Ajustar el reloj/calendario – día de la semana.	CALL 42	10-4
Invocar la cadena de fecha/hora.	CALL 43	10-4
Invocar la fecha numérica (día, mes año).	CALL 44	10-5
Invocar la cadena de la hora.	CALL 45	10-5
Invocar la hora numérica.	CALL 46	10-6
Invocar la cadena de día de la semana.	CALL 47	10-7
Invocar el día de la semana numérico.	CALL 48	10-7
Invocar la cadena de la fecha.	CALL 52	10-8

CALL 40 - Ajuste de hora del reloj/calendario

Objetivo:

Use CALL 40 para ajustar las siguientes funciones del reloj/calendario:

- H = horas (0 A 23; sólo hay un reloj de 24 horas disponible)
- M = minutos (0 a 59)
- S = segundos (0 a 59)

Sintaxis:

PUSH [horas]
PUSH [minutos]
PUSH [segundos]
CALL 40

Ejemplo:

Programe el reloj para la 1:35 P.M. (programado como 13:35; sólo hay un reloj de 24 horas disponible)

```
>1  REM EXAMPLE PROGRAM
>10 H = 13 : M = 35 : S = 00
>20 REM HOURS = 13, MINUTES = 35, SECONDS = 00
>30 PUSH H,M,S
>40 CALL 40
READY
>RUN
READY
>
```

CALL 41 - Ajuste de fecha del reloj/calendario

Objetivo:

Use CALL 41 para establecer las siguientes funciones del reloj calendario:

- D = fecha
- M = mes
- Y = año

Tres valors se PUSH y ninguno se POP.

Sintaxis:

PUSH [fecha]
PUSH [mes]
PUSH [año]
CALL 41

Ejemplo:

Programe el reloj/calendario para el 16 de junio de 1991.

```
>1   REM EXAMPLE PROGRAM
>5   STRING 100,20
>10  H=13 : M=35 : S=00
>20  REM HOURS = 13, MINUTES = 35, SECONDS = 00
>30  PUSH H,M,S
>40  CALL 40
>60  D=16 : MO=6 : Y=91
>70  PUSH D,MO,Y
>80  CALL 41
>90  PUSH 3
>100 CALL 42
>120 PUSH 0
>130 CALL 43
>140 PRINT $(0)

READY
>RUN

16-JUN-91 13:35:00
```

CALL 42 - Ajuste del día de la semana

Objetivo:

Use CALL 42 para ajustar el día de la semana. El domingo es el día 1. El sábado es el día 7.

Sintaxis:

PUSH [día de la semana]
CALL 42

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 3: CALL 42:REM DAY IS TUESDAY.
```

CALL 43 - Invocar la cadena de fecha/hora

Objetivo:

Use CALL 43 para llamar la fecha y hora actuales como una cadena. PUSH el número de la cadena para recibir la fecha/hora (dd-mmm-yy HH:MM:SS). Debe asignar un mínimo de 18 caracteres para la cadena. Esto requiere establecer que la longitud máxima para todas las cadenas sea por lo menos 18 caracteres.

Sintaxis:

PUSH [número de cadena]
CALL 43

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 1: CALL 43: REM PUT DATE/TIME IN STRING 1
>30 PRINT $(1)
>40 END

READY
>RUN

16-JUN-91 13:35:00

READY
>
```

CALL 44 - Invocar la fecha numérica

Objetivo:

Use CALL 44 para llamar la fecha actual a la pila de argumentos como tres números. No hay argumento de entrada en esta rutina y se llaman tres variables. La fecha POP (aparece) en el orden de día, mes y año.

Sintaxis:

CALL 44
POP [día]
POP [mes]
POP [año]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM DATE RETRIEVE – NUMERIC EXAMPLE
>20 CALL 44 : REM INVOKE THE UTILITY ROUTINE
>30 POP D,M,Y : REM GET THE DATA FROM THE ARGUMENT STACK
>40 PRINT “CURRENT DATE IS ”,Y,M,D
>50 END

READY
>RUN

CURRENT DATE IS  91  6  19

READY

>
```

CALL 45 - Invocar la cadena de la hora

Objetivo:

Use CALL 45 para llamar la hora actual en una cadena (HH:MM:SS). PUSH el número de la cadena para recibir la hora. Debe asignar un mínimo de 8 caracteres a la cadena.

Sintaxis:

PUSH [número de cadena]
CALL 45

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 1 : CALL 45 : REM PUT TIME IN STRING 1
>30 PRINT $(1)
>40 END

READY
>RUN

06:40:49

READY
>
```

CALL 46 - Invocar la hora numérica

Objetivo:

Use CALL 46 para llamar la hora del día en forma numérica. Llame la hora del día en forma numérica ejecutando CALL 46 y POP las tres variables de la pila de argumentos. No hay argumentos de entrada. La hora POP (aparece) en horas, minutos y segundos.

Sintaxis:

CALL 46
POP [hora]
POP [minuto]
POP [segundo]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM TIME IN VARIABLES EXAMPLE : REM GET THE WALL CLOCK
    TIME
>20 CALL 46
>30 POP H,M,S
>40 PRINT  "CURRENT TIME IS ",H,M,S
>50 END

READY
>RUN

CURRENT TIME IS  6  43  7

READY
>
```

CALL 47 - Invocar la cadena del día de la semana

Objetivo:

Use CALL 47 para llamar el día actual de la semana en una cadena de 3 caracteres. PUSH el número de la cadena para recibir el día de la semana. Debe asignar un mínimo de 3 caracteres por cadena. Las cadenas llamadas son SUN, MON, TUE, WED, THU, FRI, and SAT.

Sintaxis:

PUSH [número de cadena]
CALL 47

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 0 :CALL 47
>30 PRINT "TODAY IS ",$(0)

READY
>RUN

TODAY IS FRI

READY
>
```

CALL 48 - Invocar el día de la semana numérico

Objetivo:

Use CALL 48 para llamar el día actual de la semana a la pila de argumentos como un número (ejemplo: **`Sunday=1, Saturday=7`**). Este número puede POP (aparecer) como una variable.

Sintaxis:

CALL 48
POP [día de la semana]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM DAY OF WEEK RETRIEVE – NUMERIC EXAMPLE
>20 CALL 48 : REM INVOKE UTILITY TO GET D.O.W.
>30 POP D
>40 PRINT D
>50 END

READY
>RUN

 5

READY
>
```

CALL 52 - Invocar la cadena de la fecha

Objetivo:

Use CALL 52 para invocar la fecha actual en una cadena (dd-mmm-yy). PUSH el número de la cadena para recibir la fecha. Debe asignar un mínimo de 9 caracteres por cadena.

Sintaxis:

PUSH [número de cadena]
CALL 52

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 PUSH 1 : CALL 52 : REM PUT DATE IN STRING 1
>30 PRINT $(1)
>40 END

READY
>RUN

16–JUN–91

READY
>
```

Capítulo 11

Funciones de estado

Este capítulo describe e ilustra comandos que monitorizan el estado del módulo BASIC. Este capítulo también describe e ilustra comandos que permiten la configuración del controlador DF1 dentro del programa BASIC o desde la línea de comando. La Tabla 11.A lista los mnemónicos correspondientes.

Tabla 11.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Obtener el número de caracteres en los búfers PRT2.	CALL 36	11-2
Verificar el búfer de la imagen de salida de la CPU.	CALL 51	11-3
Verificar el búfer de la imagen de entrada de la CPU.	CALL 55	11-4
Verificar el estado del archivo M0.	CALL 58	11-5
Verificar el estado del archivo M1.	CALL 59	11-6
Verificar el estado de la CPU del controlador SLC 500.	CALL 75	11-7
Verificar el estado de la batería.	CALL 80	11-8
Verificar el estado de escritura remota del archivo del interface DH–485.	CALL 86	11-8
Verificar el estado de lectura remota del archivo del interface DH–485.	CALL 87	11-9
Obtener el número de caracteres en los búfers PRT1.	CALL 95	11-10
Habilitar la señal DTR del puerto PRT2.	CALL 97	11-11
Inhabilitar la señal DTR del puerto PRT2	CALL 98	11-11
Habilitar las comunicaciones del controlador DF1	CALL 108	11-12
Inhabilitar las comunicaciones del controlador DF1	CALL 113	11-19
Borrar los búfers de entrada y salida del módulo BASIC	CALL 120	11-19
Obtener el número de ID del programa del procesador SLC	CALL 121	11-20

CALL 36 - Obtención del número de caracteres en los búfers PRT2

Objetivo:

Use CALL 36 para obtener el número de caracteres en el búfer seleccionado del puerto PRT2.

Se debe hacer PUSH al búfer que se va a examinar:

- PUSH 1 para el búfer de entradas
- PUSH 0 para el búfer de salidas

Se requiere un POP para obtener el número de caracteres.

Sintaxis:

PUSH [selección de búfer]
CALL 36
POP [número de caracteres]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : REM EXAMINES THE OUTPUT BUFFER
>20 CALL 36
>30 POP X : REM GET THE NUMBER OF CHARACTERS
>40 PRINT “NUMBER OF CHARACTERS IN OUTPUT BUFFER IS”,X
>50 END
READY
>RUN
NUMBER OF CHARACTERS IN OUTPUT BUFFER IS 0
READY
>
```

CALL 51 - Verificación del búfer de imagen de salida de la CPU

Objetivo:

Use CALL 51 para determinar si el búfer de imagen de salida del controlador SLC 500 ubicado en el módulo BASIC ha sido actualizado desde la última vez que se verificó. (En este caso, actualizar significa que los datos fueron escritos a estos búfers desde la CPU, aunque los datos sean el mismo valor). Esta rutina no tiene argumentos de entrada pero tiene un argumento de salida.

El argumento de salida es igual a:

- 0 si el procesador lógico no ha escrito al búfer de imagen de salida desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último
- 1 si el procesador lógico ha escrito al búfer de imagen de salida desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último
- 2 si el procesador lógico no tiene esta capacidad (como en el caso del procesador SLC 5/01)

Sintaxis:

CALL 51
POP [estado del búfer de imagen de salida]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>120 CALL 51 : REM WAIT ON SLC
>130 POP S
>140 IF (S = 2) THEN PRINT “THE SLC DOES NOT SUPPORT THIS
     FUNCTION”
>150 IF (S = 2) THEN STOP
>160 IF (S = 0) THEN GOTO 120
>170 PRINT “OUTPUT IMAGE HAS BEEN UPDATED”

READY
>RUN

THE SLC DOES NOT SUPPORT THIS FUNCTION
STOP – IN LINE  160
READY
```

CALL 55 - Verificación del búfer de imagen de entrada de la CPU

Objetivo:

Use CALL 55 para determinar si el búfer de imagen de entrada del controlador SLC 500 ubicado en el módulo BASIC ha sido leído por el procesador lógico desde la última vez que se verificó. Esta rutina no tiene argumentos de entrada pero tiene un argumento de salida.

El argumento de salida es igual a:

- 0 si el procesador lógico no ha leído desde el búfer de imagen de entrada desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último
- 1 si el procesador lógico ha leído desde el búfer de imagen de entrada desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último
- 2 si el procesador lógico no tiene esta capacidad (comoEen el caso del procesador SLC 5/01)

Sintaxis:

CALL 55
POP [estado del búfer de imagen de entrada]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>120 CALL 55 :  REM WAIT ON SLC
>130 POP S
>140 IF (S=2) THEN  PRINT “THE SLC DOES NOT SUPPORT THIS
     FUNCTION”
>150 IF (S=2) THEN  STOP
>160 IF (S=0) THEN  GOTO 120
>170 PRINT “INPUT IMAGE HAS BEEN READ”

READY
>RUN

INPUT IMAGE HAS BEEN READ

READY
>
```

CALL 58 - Verificación del archivo M0

Objetivo:

Use CALL 58 para determinar si el archivo M0 del módulo ubicado en el módulo BASIC ha sido actualizado desde la última vez que se verificó. (En este caso, actualizar significa que los datos fueron escritos a estos búfers desde la CPU, aunque los datos sean el mismo valor). Esta rutina no tiene argumentos de entrada pero tiene un argumento de salida.

El argumento de salida es igual a:

- 0 si el procesador lógico no ha escrito al archivo M0 del módulo desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último
- 1 si el procesador lógico ha escrito al archivo M0 del módulo desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC
- 2 si el procesador lógico no tiene esta capacidad (como en el caso del procesador SLC 5/01)

Sintaxis:

CALL 58
POP [estado de escritura del archivo M0 del módulo]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>120 CALL 58 : REM START WAITING ON M0 UPDATE
>130 POP S
>140 IF (S = 2) THEN PRINT “PROCESSOR DOES NOT SUPPORT
     THIS FUNCTION”
>150 IF (S = 2) THEN STOP
>160 IF (S = 0) THEN GOTO 120
>170 PUSH 64
>180 CALL 56
>190 POP A
>200 PRINT “CALL 56 OUTPUT IS ”,A


READY
>RUN

CALL 56 OUTPUT IS  0
```

CALL 59 - Verificación del archivo M1

Objetivo:

Use CALL 59 para determinar si el archivo M1 del módulo ubicado en el módulo BASIC ha sido leído por el procesador lógico desde la última vez que se verificó. Esta rutina no tiene argumentos de entrada pero tiene un argumento de salida.

El argumento de salida es igual a:

- 0 si el procesador lógico no ha leído desde el archivo M1 del módulo desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último
- 1 si el procesador lógico ha leído desde el archivo M1 del módulo desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último
- 2 si el procesador lógico no tiene esta capacidad (como en el caso del procesador SLC 5/01)

Sintaxis:

CALL 59
POP [estado de lectura del archivo M1 del módulo]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>100 PUSH 64
>110 CALL 56 :  REM COPY BASIC OUTPUT BUFFER TO M1
>120 POP A
>130 IF (A=2) THEN  PRINT “SLC DOES NOT SUPPORT THIS
     FUNCTION”
>140 IF (A=2) THEN  STOP
>150 IF (A<>0)THE GOTO 110
>160 CALL 59 :  REM START WAITING NOW
>170 POP S
>180 IF (S=2) THEN  PRINT “SLC DOES NOT SUPPORT THIS
     FUNCTION”
>190 IF (S=2)THE STOP
>200 IF (S=0) THEN  GOTO 170
>210 PRINT “CALL 59 OUTPUT IS ”,S

READY
>RUN

CALL 59 OUTPUT IS  1
```

CALL 75 - Verificación del estado de la CPU del controlador SLC 500

Objetivo:

Use CALL 75 para verificar el modo (marcha/programación/prueba) del procesador SLC. No se requiere ningún PUSH, pero se requiere un POP.

En el modo de operación del SLC 5/01, los valores POP son:

- 0 = procesador SLC en el modo de marcha (Run)
- 1 = procesador SLC fuera del modo de marcha (Run)

En el modo de operación del SLC 5/02, los valores POP son:

- 0 = procesador SLC en el modo de marcha (Run)
- 1 = procesador SLC en el modo de programación (Programa)
- 2 = procesador SLC en el modo de prueba (Test)

Sintaxis:

CALL 75
POP [modo del procesador]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>100 CALL 75
>110 POP S
>120 IF (S=0) THEN PRINT “SLC IS IN RUN MODE”
>130 IF (S=1) THEN PRINT “SLC IS NOT IN RUN MODE”
>140 IS (S=2) THEN PRINT “SLC IS IN TEST MODE”

READY
>RUN

SLC IS IN RUN MODE

READY
>
```

CALL 80 - Verificación del estado de la batería

Objetivo:

Use CALL 80 para verificar el estado de la batería del módulo BASIC. Si aparece (POP) un cero después del CALL 80, la batería está en buen estado. Si aparece (POP) un 1, existe una condición de batería baja.

Sintaxis:

CALL 80
POP [estado de la batería]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 80
>20 POP C
>30 IF (C<>0) THEN PRINT ”BATTERY LOW!”
>40 END

READY
>RUN

BATTERY LOW!
```

CALL 86 - Verificación del estado de escritura remota del archivo de interface DH–485

Objetivo:

Use CALL 86 para determinar si el archivo de interface común DH–485 ubicado en el módulo BASIC ha sido actualizado desde la última vez que se verificó. Esta rutina no tiene argumentos de entrada pero tiene un argumento de salida.

El argumento de salida es igual a:

- 0 si un dispositivo en la red de comunicaciones en serie DH-485 no ha escrito al archivo de interface común en serie DH-485 desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último
- 1 si un dispositivo en la red de comunicaciones en serie DH-485 ha escrito al archivo de interface común DH-485 desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último

Sintaxis:

CALL 86
POP [Estado de escritura remota del archivo de interface DH-485]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>100 CALL 86 : REM CHECK FILE STATUS
>110 POP X : REM GET THE STATUS
>120 IF(X<>1) THEN GOTO 100 : REM WAIT ON THE DATA

READY
>
```

CALL 87 - Verificación del estado de lectura remota del archivo de interface DH–485

Objetivo:

Use CALL 87 para determinar si el archivo de interface común DH–485 ubicado en el módulo BASIC ha sido leído por un dispositivo en la red de comunicaciones en serie DH-485 desde la última vez que se verificó. Esta rutina no tiene argumentos de entrada pero tiene un argumento de salida.

El argumento de salida es igual a:

- 0 si un dispositivo no ha leído desde el archivo de interface común DH-485 desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último
- 1 si un dispositivo en la red de comunicaciones en serie DH-485 ha leído el archivo de interface común DH-485 desde la última vez que se ejecutó este CALL o desde que se encendió el módulo BASIC, lo que ocurrió al último

Sintaxis:

CALL 87
POP [estado de lectura remota del archivo de interface DH-485]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>100 CALL 87 : REM CHECK FILE STATUS
>110 POP X : REM GET THE STATUS
>120 IF (X<>1) GOTO 100: REM WAIT ON DATA TO BE READ

READY
>
```

CALL 95 - Obtención del número de caracteres en los búfers PRT1

Objetivo:

Use CALL 95 para obtener el número de caracteres en el búfer seleccionado del puerto PRT1.

Se debe hacer PUSH al búfer que se va a examinar:

- PUSH 1 para el búfer de entradas
- PUSH 0 para el búfer de salidas

Se requiere un POP para obtener el número de caracteres.

Sintaxis:

PUSH [selección de búfer]
CALL 95
POP [número de caracteres]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : REM EXAMINES THE OUTPUT BUFFER
>20 CALL 95
>30 POP X : REM GET THE NUMBER OF CHARACTERS
>40 PRINT “NUMBER OF CHARACTERS IN PRT1 OUTPUT BUFFER IS
    ”,X
>50 END

READY
>RUN

NUMBER OF CHARACTERS IN PRT1 OUTPUT BUFFER IS  0

READY
>
```

CALL 97 - Habilitación de la señal DTR del puerto PRT2

Objetivo:

Use CALL 97 para habilitar la señal del terminal de datos listos (DTR) del puerto PRT2. La señal DTR del puerto PTR2 se habilita de manera predeterminada al momento del encendido. Este CALL vuelve a habilitar la señal DTR si ésta ha sido inhabilitada por el CALL 98.

Sintaxis:

CALL 97

Ejemplo:

```
>10 REM EXAMPLE PROGRAM
>20 CALL 97 : REM ENABLE DTR SIGNAL


READY
>
```

CALL 98 - Inhabilitación de la señal DTR del puerto PRT2

Objetivo:

Use CALL 98 para inhabilitar la señal del terminal de datos listos (DTR) del puerto PRT2.

Sintaxis:

CALL 98

Ejemplo:

```
>10 REM EXAMPLE PROGRAM
>20 CALL 98 : REM DISABLE DTR SIGNAL


READY
>
```

CALL 108 - Habilitación de comunicaciones del controlador DF1

Objetivo:

Use CALL 108 para habilitar las comunicaciones del controlador DF1 a través del puerto PRT2.

Importante: DF1 sólo puede habilitarse si el puente JW4 en el módulo BASIC está en la posición correcta. Para obtener más información, consulte el Manual de diseño e integración del módulo BASIC SLC 500 (número de publicación 1746-6.1ES).

Esta rutina tiene seis argumentos de entrada y ningún argumento de salida. El primer argumento de entrada especifica la selección del código de operación que indica el modo de operación del controlador DF1. El código de operación especifica los siguientes parámetros DF1:

- operación full-duplex o half-duplex esclavo
- selección de detección de paquete duplicado (DPD)
- selección de verificación de errores BCC o CRC
- habilitar las respuestas integradas (ER) o auto-detectar las respuestas integradas (ADER). Si se selecciona ADER, las respuestas habilitadas no se transmitirán hasta que se reciba una respuesta integrada. Se supone que si el dispositivo con el que se está comunicando puede enviar un ER, también los puede recibir (se aplica solamente a operación full-duplex).
- selección de handshaking de modem:
  - opciones half-duplex:
    Sin HandShaking (NHS)
    Modem Half-Duplex sin portadora continua (HDMwoCC)
    Modem Half-Duplex con portadora continua (HDMwCC)
  - opciones full-duplex:
    Sin HandShaking (NHS)
    Modem Full-Duplex (FDM)

Los valores válidos para el código de operación son 0 a 11 para el modo half-duplex y 16 a 31 para el modo full-duplex. La Tabla 11.B lista los valores válidos para los códigos de operación half-duplex y su modo de operación correspondiente. La Tabla 11.C lista los valores legales para los códigos de operación full-duplex y su modo de operación correspondiente.

También se acepta un rango especial de códigos de operación (32 - 43). Estos códigos son idénticos a los códigos 0 – 11, excepto que se suprimen los paquetes de fin de transmisión (EOT). Esta operación es una variación del protocolo DF1 estándar y sólo debe usarse cuando se minimizan las transmisiones desde un módulo esclavo. Cuando se usa una de estas selecciones, el controlador DF1 no responderá a las peticiones (ENQ) de un DF1 maestro a menos que haya un paquete de datos transmitido.

Importante: Otros parámetros de puerto, tales como velocidad en baudios, número de bits de parada y la paridad se seleccionan usando el comando MODE antes que se habilite DF1. La selección de handshaking de modem que se hace aquí anula el parámetro de handshaking del comando MODE hasta que se inhabilite DF1.

Tabla 11.B
Códigos de operación DF1 Half-Duplex

Código de operación	Modo correspondiente de operación	Código de operación especial (igual que 0 - 11, excepto que se suprime EOT)
0	NHS, inhabilita DPD, verificación de errores BCC	32
1	NHS, habilita DPD, verificación de errores BCC	33
2	NHS, inhabilita DPD, verificación de errores CRC	34
3	NHS, habilita DPD, verificación de errores CRC	35
4	HDMwoCC, inhabilita DPD, verificación de errores BCC	36
5	HDMwoCC, habilita DPD, verificación de errores BCC	37
6	HDMwoCC, inhabilita DPD, verificación de errores CRC	38
7	HDMwoCC, habilita DPD, verificación de errores CRC	39
8	HDMwCC, inhabilita DPD, verificación de errores BCC	40
9	HDMwCC, habilita DPD, verificación de errores BCC	41
10	HDMwCC, inhabilita DPD, verificación de errores CRC	42
11	HDMwCC, habilita DPD, verificación de errores CRC	43

Tabla 11.C
Códigos de operación DF1 Full-Duplex

Código de operación	Modo correspondiente de operación
16	NHS, ER, inhabilita DPD, verificación de errores BCC
17	NHS, ER, habilita DPD, verificación de errores BCC
18	NHS, ER, inhabilita DPD, CRC Error Checking
19	NHS, ER, habilita DPD, verificación de errores CRC
20	NHS, ADER, inhabilita DPD, verificación de errores BCC
21	NHS, ADER, habilita DPD, verificación de errores BCC
22	NHS, ADER, inhabilita DPD, verificación de errores CRC
23	NHS, ADER, habilita DPD, verificación de errores CRC
24	FDM, ER, inhabilita DPD, verificación de errores BCC
25	FDM, ER, habilita DPD, verificación de errores BCC
26	FDM, ER, inhabilita DPD, verificación de errores CRC
27	FDM, ER, habilita DPD, verificación de errores CRC
28	FDM, ADER, inhabilita DPD, verificación de errores BCC
29	FDM, ADER, habilita DPD, verificación de errores BCC
30	FDM, ADER, inhabilita DPD, verificación de errores CRC
31	FDM, ADER, habilita DPD, verificación de errores CRC

Control de modem sin handshaking Half–Duplex

El control de modem sin handshaking Half-duplex, seleccionado por los códigos de operación 0 a 3, tiene las siguientes características:

- La línea de salida RTS se activará durante la transmisión, pero no se realizará ningún retardo a la conexión o a la desconexión de RTS.
- La línea de salida DTR no es manipulada por el controlador DF1. Se recomienda que usted active DTR en su programa BASIC mientras se realizan las comunicaciones DF1.
- Las líneas de entrada CTS y DSR no son monitorizadas, ni tienen ningún efecto sobre las transmisiones o recepciones.
- Un monitor de transmisión garantiza que las interrupciones del transmisor se generen de manera oportuna. Si se produce un límite de tiempo, el estado_DF1 se establecerá en el valor de código 5 si se estaba transmitiendo un paquete de datos. Además, RTS se cancela inmediatamente cuando se produce este tiempo límite.

El control de módem sin portadora continua Half-duplex, seleccionado por los códigos de operación 4 a 7, tiene las siguientes características:

Importante: Para una correcta operación, la línea de detección de portadora de datos (DCD) del módem debe estar conectada a la entrada DSR del puerto PRT2.

- La línea de salida RTS se activa sólo durante las transmisiones. La transmisión del paquete empieza después del retardo especificado por el parámetro de retardo a la conexión RTS, suponiendo que entonces la entrada CTS está activada. Cuando se completa la transmisión y el período de retardo especificado por el parámetro de retardo a la desconexión RTS termina, RTS se desactiva.

- Una transmisión no empezará hasta que la entrada CTS esté activa. Una transmisión garantiza que las interrupciones del transmisor se generen de manera oportuna. Si se excede el tiempo límite, el estado_DF1 se establecerá en el valor de código 5 si se estaba transmitiendo un paquete de datos. Además, RTS se cancela inmediatamente cuando se produce este tiempo límite.

- Si todavía no está activa, la línea DTR se levantará cuando se habilite el controlador DF1. Aún después que se inhabilite el controlador DF1, éste *permanecerá* activo; el usuario puede desactivarlo a través de un CALL BASIC.

- Los caracteres que se reciban sólo se aceptarán si la línea DCD está activa. Una recepción de paquete se cancelará si DCD cambia a inactivo durante la recepción de byte a byte del paquete.

  No hay monitorización constante de DCD incluso entre paquetes como sucede con la selección de portador constante. Por lo tanto, la línea DTR nunca se desactiva.

Control de modem con portadora continua Half–Duplex

El control de modem con portadora continua Half-duplex, seleccionado por los códigos de operación 8 a 11, tiene las siguientes características:

Importante: Para una correcta operación, la línea de detección de portadora de datos (DCD) del módem debe estar conectada a la entrada DSR del puerto PRT2.

- La línea de salida RTS se activa sólo durante las transmisiones. La transmisión del paquete empieza después del retardo especificado por el parámetro de retardo a la conexión RTS, suponiendo que entonces la entrada CTS está activada. Cuando se completa la transmisión y el período de retardo especificado por el parámetro de retardo a la desconexión RTS termina, RTS se desactiva.

- Una transmisión no empezará hasta que la entrada CTS esté activa. Una transmisión garantiza que las interrupciones del transmisor se generen de manera oportuna. Si se excede el tiempo límite, el estado_DF1 se establecerá en el valor de código 5 si se estaba transmitiendo un paquete de datos. Además, RTS se deja inmediatamente cuando se produce este tiempo límite.

- Si todavía no está activa, la línea DTR se levantará cuando se habilite el controlador DF1. Se dejará sólo cuando se pierda DCD según se describe en el siguiente párrafo. Aún después que se inhabilite o que permanezca activo el controlador DF1, el usuario puede desactivarlo a través de un CALL BASIC.

- Para la recepción del paquete, la señal DCD se monitorizará (a través de la línea de entrada DSR). Si DCD todavía no está activo cuando se habilita el controlador DF1, entonces se detectará inmediatamente cuando éste cambie a activo. En este punto, DCD es controlado cada 5 ms para cerciorarse que permanece activo. Si DCD cambia a inactivo, el controlador espera 10 segundos para que cambie a activo otra vez. Si DCD no cambia a activo otra vez en este período de tiempo, entonces la línea de salida DTR se cancelará por un tiempo de 5 a 10 ms.

  Además, los caracteres que se reciben son aceptados si la línea DCD está activa. La recepción del paquete se cancelará si DCD cambia a inactivo durante la recepción byte a byte de un paquete.

Full–Duplex sin Handshaking

Full-duplex sin handshaking, seleccionado por los códigos de operación 16 a 23, tiene las siguientes características:

- La línea de salida RTS se activará cuando se habilite el controlador DF1 y permanecerá en ese estado hasta que se inhabilite el controlador DF1.

- La línea de salida DTR no es manipulada por el controlador DF1. Se recomienda que el usuario active DTR en el programa BASIC mientras se realizan las comunicaciones DF1.

- Las líneas de entrada CTS y DSR *no* se monitorizan ni tienen ningún efecto en las transmisioens.

- Un monitor de transmisión garantiza que las interrupciones del transmisor se generen de manera oportuna. Si se excede el tiempo límite, el estado_DF1 se enviará al valor de código 5 si el paquete en proceso de transmisión era un paquete de datos. RTS *no* se desactivará cuando se exceda este tiempo límite.

Modem Full–Duplex (FDM)

El modem Full-Duplex (FDM), seleccionado por los códigos de operación 24 a 31 tiene las siguientes características:

- La línea de salida RTS se activará cuando se habilite el controlador DF1 y permanecerá en ese estado hasta que se inhabilite el controlador DF1.
- Una transmisión no empezará hasta que la entrada CTS esté activa. Un monitor de transmisión garantiza que las interrupciones del transmisor se generen de manera oportuna. Si se excede un tiempo límite, el estado_DF1 se enviará al valor de código 5 si el paquete en proceso de ser transmitido era un paquete de datos. RTS *no* se desactiva cuando se excede este tiempo límite.
- Si DTR todavía no está activo cuando se habilita el controlador DF1, se activará inmediatamente. Cambiará a activo sólo si DCD se pierde tal como se describe en el siguente párrafo. Aún después que se inhabilita el controlador DF1, DTR permanece activo; el usuario puede desactivarlo a través de un CALL BASIC.
- Para recepciones de paquetes, la señal DCD se monitorizará a través de la línea de entrada DSR. Si DCD todavía no está activo cuando se habilita el controlador DF1, entonces se detectará inmediatamente cuando éste cambie a activo. En este punto, DCD es controlado cada 5 ms para cerciorarse que permanece activo. Si DCD cambia a inactivo, el controlador espera 10 segundos para que cambie a activo otra vez. Si DCD no cambia a activo otra vez en este período de tiempo, entonces la línea de salida DTR se desactivará por un tiempo de 5 a 10 ms.

  Además, los caracteres que se reciben sólo son aceptados si la línea DCD está activa. La recepción del paquete se cancelará si DCD cambia a inactivo durante la recepción byte a byte de ese paquete.

El segundo argumento de entrada especifica el tiempo límite de Poll cuando está en el modo half–duplex, o el tiempo límite de ACK (confirmación) cuando está en el modo full–duplex. El tiempo límite de Poll especifica, en incrementos de 5 ms, cuánto tiempo esperar antes que se realice la encuesta (poll) por el DF1 maestro, antes de que se ignore una petición de transmisión. PUSH 0 indica que no hay tiempo límite de encuesta. El tiempo límite de ACK (confirmación) especifica, en incrementos de 5 ms, cuánto tiempo esperar por un ACK/NAK antes de transmitir un ENQuiry. El rango válido para el tiempo límite ACK es 2 a 65535.

El tercer argumento de entrada especifica el número de reintentos de mensaje cuando está en el modo half–duplex, o el número de reintentos de ENQuiry que hay que hacer cuando está en el modo full–duplex. Los reintentos de mensaje especifican el número de intentos de transmisión de mensaje hechos antes de abandonar y señalar la transmisión como fallida. PUSH 0 indica que se hará sólo la tentativa inicial y si no hay confirmación por parte del maestro, se señalará como fallida. Los reintentos ENQuiry especifican el número de ENQ a transmitir antes de que la transmisión de un paquete se señale como fallida. El rango válido para ambos es 0 a 254.

El cuarto argumento de entrada especifica el tiempo de retardo a la conexión RTS en el modo half–duplex o el número de reintentos recibidos NAK a realizar en el modo full–duplex. El retardo a la conexión RTS especifica, en incrementos de 5 ms, el retardo entre el momento en que se activa una petición de emitir (RTS) y el inicio de la transmisión. Se usa sólo si HDMwCC o HDMwoCC se selecciona a través del primer argumento de entrada. El rango válido para el retardo a la conexión RTS es 0 a 65535. Los reintentos recibidos NAK especifican el número de reintentos que hay que transmitir debido a las respuestas NAK recibidas. El rango válido para los reintentos NAK recibidos es 0 a 254.

El quinto argumento de entrada especifica el tiempo de retardo a la desconexión RTS. El retardo a la desconexión RTS especifica, en incrementos de 5 ms, el retardo entre el momento cuando se completa una transmisión y cuando se desactiva una petición de emitir (RTS). El rango válido para el retardo a la desconexión RTS es 0 a 65499. Este argumento se usa sólo si HDMwCC o HDMwoCC se selecciona a través del primer argumento de entrada. Este argumento de entrada sólo se usa para el modo half–duplex. Cuando se selecciona el modo full–duplex, se debe PUSH un valor NULL.

El sexto argumento de entrada especifica la dirección del módulo BASIC a la cual el controlador DF1 responde cuando recibe pedidos desde un dispositivo DF1 remoto. Los valores válidos son 0 a 254. Este argumento de entrada se usa para los modos half–duplex y full–duplex.

Sintaxis:

PUSH [código de operación]
PUSH [tiempo límite de Poll o tiempo límite de ACK]
PUSH [reintentos de mensaje o reintentos de ENQuiry]
PUSH [retardo a la conexión RTS o reintentos NAK recibidos]
PUSH [retardo a la desconexión RTS o valor NULL]
PUSH [dirección DF1 del módulo BASIC]
CALL 108

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 5 : REM HDMWOCC, ENABLE DPD, BCC ERROR CHECKING
>20 PUSH 200 : REM WAIT 1 SECOND TO BE POLLED BY MASTER
>30 PUSH 2 : REM PERFORM 2 RETRIES
>40 PUSH 4 : REM 20 MS RTS ON DELAY
>50 PUSH 4 : REM 20 MS RTS OFF DELAY
>60 PUSH 10 : REM BASIC MODULE ADDRESS OF 10
>70 CALL 108
>80 END
```

CALL 113 - Inhabilitación de comunicaciones del controlador DF1

Objetivo:

Use CALL 113 para inhabilitar las comunicaciones del controlador DF1. Esta rutina no tiene argumentos de entrada ni de salida. Este CALL termina la comunicación DF1 inmediatamente, aunque se esté llevando a cabo la transmisión en serie de un paquete de datos. Se debe escribir el programa de usuario de manera que se complete cualquier transmisión antes de realizar el CALL 113. Este CALL borra la transmisión de PRT2 y los búfers de recepción.

Sintaxis:

CALL 113

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 113
>20 END
```

CALL 120 - Borrado de los búfers de entrada y salida del módulo BASIC

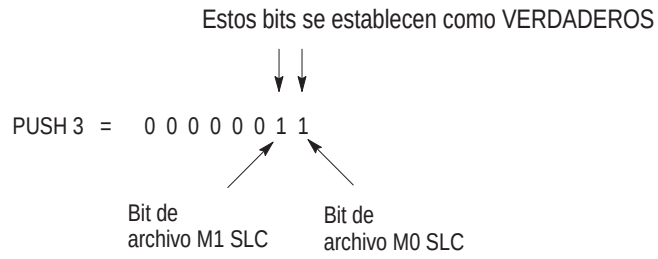
Objetivo:

Use CALL 120 para borrar los búfers de entrada y salida del módulo BASIC. Esta rutina tiene un argumento de entrada y ningún argumento de salida. El argumento de entrada es una palabra de 8 bits que corresponde a los búfers de entrada y salida del módulo BASIC, tal como se muestra en la siguiente tabla:

Tabla 11.D
Información para borrar el búfer de entrada y de salida

Bit	Equivalente decimal	Areas de búfers de entrada y salida del módulo BASIC
0	1	Archivo M0 SLC
1	2	Archivo M1 SLC
2	4	Tabla de imagen de salida SLC
3	8	Tabla de imagen de entrada SLC
4	16	Archivo de entrada de interface común
5	32	Archivo de salida de interface común
6		No se usa
7		No se usa

Se debe PUSH el equivalente decimal de las áreas de los búfers de entrada y salida que se desea borrrar. Por ejemplo, para borrar los archivos M0 SLC y M1 SLC se PUSH el valor 3. El módulo BASIC establece los bits 0 y 1 como verdaderos y borra las áreas del archivo M0 y M1 SLC de los búfers de entrada y salida.



Sintaxis:

PUSH [equivalente decimal]
CALL 120

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 3 : REM CLEAR SLC MO AND M1 FILES
>20 CALL 120
>30 END
```

CALL 121 - Obtención del número de ID del programa del procesador SLC

Objetivo:

Use CALL 121 para obtener el número de ID del programa activo del procesador SLC. Esta rutina no tiene argumentos de entrada y tiene un argumento de salida. El argumento de salida es un valor entero en 0 y 65536 que corresponde al número de ID del programa activo del procesador SLC.

Sintaxis:

CALL 121
POP [número de ID del programa]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 121
>20 POP X : REM GET SLC PROCESSOR PROGRAM ID
>30 END
```

Capítulo 12

Funciones de salida

Este capítulo describe e ilustra comandos que permiten la transferencia de datos desde el módulo BASIC a los puertos externos PRT1, PRT2 y DH485 dentro del programa BASIC o desde la línea de comando. La Tabla 12.A lista los mnemónicos correspondientes.

Tabla 12.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Transferir datos desde el sistema de E/S SLC o los archivos M al PRT1 o PRT2.	CALL 23	12-2
Transferir datos desde el procesador SLC a un archivo de datos DH–485 remoto.	CALL 28	12-9
Manejar todos los errores que no son manejados por la instrucción ONERR.	CALL 29	12-16
Ver la configuración actual del puerto PRT2.	CALL 31	12-17
Borrar los búfers de entradas y salidas del puerto PRT2.	CALL 37	12-18
Transferir el búfer de salidas del módulo BASIC al búfer de imagen de entrada de la CPU.	CALL 54	12-18
Transferir el búfer de salidas del módulo BASIC al archivo M1 de la CPU.	CALL 57	12-19
Transferir el búfer de salidas del módulo BASIC al archivo de interface DH–485.	CALL 85	12-20
Escribir el búfer de salidas del módulo BASIC al archivo de datos DH–485 remoto.	CALL 91	12-21
Escribir el búfer de salidas del módulo BASIC al archivo de interface DH–485 remoto.	CALL 93	12-25
Imprimir la configuración actual del puerto PRT1.	CALL 94	12-27
Borrar los búfers de entradas y salidas del puerto PRT1.	CALL 96	12-28
Control de indicadores LED del usuario.	CALL 112	12-28
Transmitir el paquete DF1.	CALL 114	12-29
Verificar el estado de DF1 XMIT.	CALL 115	12-30
Escribir a un archivo de datos PLC.	CALL 123	12-31

Si necesita	Use este mnemónico	Página
Imprimir un valor hexadecimal con supresión de cero al dispositivo de la consola.	PH0.	12-42
Imprimir un valor hexadecimal con supresión de cero al PRT2.	PH0.#	12-42
Imprimir un valor hexadecimal con supresión de cero al PRT1.	PH0.@	12-42
Imprimir un valor hexadecimal sin supresión de cero al dispositivo de la consola.	PH1.	12-42
Imprimir un valor hexadecimal sin supresión de cero al PRT2.	PH1.#	12-42
Imprimir un valor hexadecimal sin supresión de cero al PRT1.	PH1.@	12-42
Imprimir variables, cadenas o literales al dispositivo de la consola; P. es la abreviatura de Print.	PRINT	12-40
Imprimir al puerto PRT2.	PRINT#	12-40
Imprimir al puerto PRT1.	PRINT@	12-40
Imprimir un retorno de carro.	PRINT CR	12-41
Imprimir espacios.	PRINT SPC( )	12-41
Imprimir tabulaciones.	PRINT TAB( )	12-41
Imprimir valores numéricos en notación científica.	PRINT USING(Fx)	12-41
Imprimir valores numéricos en notación decimal.	PRINT USING(#.#)	12-42
Restaurar el modo de impresión predeterminado.	PRINT USING(0)	12-42
Almacenar una variable.	ST@	12-43

CALL 23 - Transferencia de datos desde archivos de la CPU al puerto 1 ó 2

Objetivo:

Use CALL 23 para transferir datos desde el archivo de datos de salida CPU o el archivo M0 CPU directamente al puerto en serie del módulo BASIC y/o a una cadena dentro del módulo BASIC. Los datos se transfieren byte inferior primero, luego byte superior, o byte superior primero luego byte inferior al puerto del módulo BASIC. Los datos también pueden almacenarse en una cadena para acceso del programa BASIC.

La selección de intercambio de bytes (byte inferior primero, luego byte superior, o byte superior primero, luego byte inferior) del último CALL 23 o CALL 22 ejecutado determina el método de empaquetado de datos para todos los puertos habilitados por CALL 23.

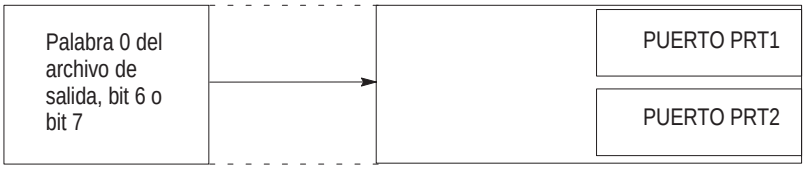
El byte inferior de la primera palabra del archivo fuente contiene el conteo de caracteres (conteo de bytes) de los datos que se están transfiriendo. Si el conteo de bytes es mayor que el archivo seleccionado, sólo se transfiere el número máximo de bytes dentro del archivo. El byte superior de la primera palabra no se usa.

Ejecute CALL 23 para configurar los parámetros de transferencia de datos. Después que se ejecuta el CALL, el módulo BASIC obtiene datos desde el archivo fuente y los transfiere al puerto PRT1, al puerto PRT2, o a una cadena interna. Los bits del archivo de imagen de entrada y salida (palabra 0, bits 6 y 7) para la ranura que contiene el módulo BASIC se usan para iniciar y notificar que se ha completado la transferencia. La operación se describe a continuación:

1. El programa de lógica de escalera del procesador SLC construye el búfer de datos. Luego el SLC determina el conteo de bytes del archivo que se va a transferir y lo coloca en el byte inferior de la primera palabra disponible que se va a transferir. Esta palabra, más los datos, forman el archivo de datos que se va a transferir.

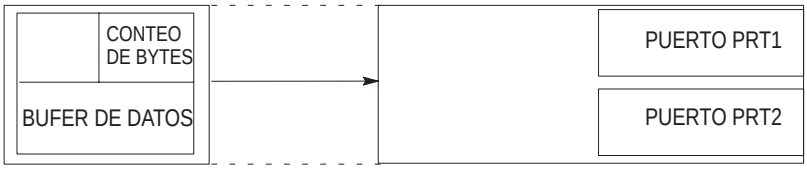
Procesador SLC

2. El programa de lógica de escalera del procesador SLC debe establecer la palabra 0 del archivo de salida, bit 6 o bit 7 para informar al módulo BASIC que hay datos válidos disponibles. El bit 6 indica que los datos están disponibles para el puerto 1 y el bit 7 indica que los datos están disponibles para el puerto 2.



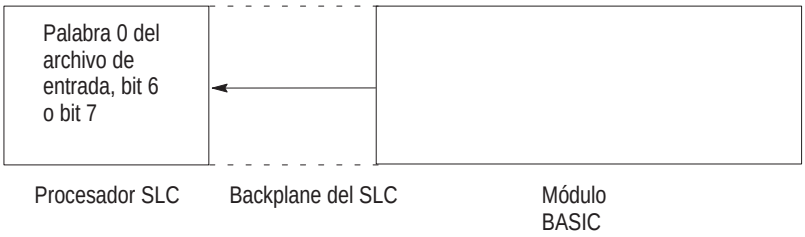
Procesador SLC Backplane del SLC Módulo BASIC

3. El módulo BASIC transfiere automáticamente los datos al puerto en serie de destino (PRT1 o PRT2) desde el procesador.

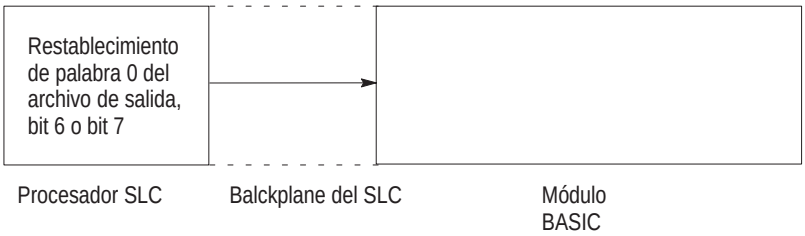


Procesador SLC Backplane del SLC Módulo BASIC

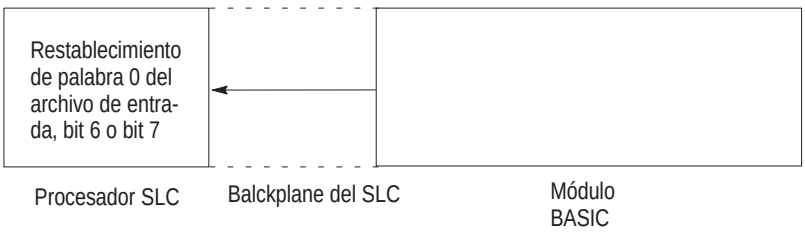
4. El módulo BASIC establece la palabra 0 del archivo de entrada, bit 6 o bit 7 para informar al procesador SLC que la transferencia se realizó correctamente.



5. El programa de lógica de escalera del procesador SLC restablece la palabra 0 del archivo de salida, bit 6 o bit 7.



6. El módulo BASIC restablece la palabra 0 del archivo de entrada, bit 6 o bit 7 en el mismo fin de ciclo de exploración en el que la palabra 0 del archivo de salida, bit 6 o bit 7 fue restablecida.



Las transferencias continúan de esta manera hasta que se vuelve a ejecutar el CALL para el puerto con parámetros de entrada diferentes. Si ocurre esto, el CALL 23 previo para el puerto se inhabilita automáticamente y se hace efectivo el nuevo CALL 23. No se ejecutan múltiples CALL 23 para el mismo puerto en paralelo. Sin embargo, el puerto 1 y el puerto 2 pueden activarse simultáneamente emitiendo CALL 23 separados para estos puertos.

Este CALL tiene cinco argumentos de entrada y un argumento de salida.

El primer argumento de entrada selecciona el destino de los datos. Puede ser el número de puerto (1 ó 2) y/o la cadena interna:

- 0 = Inhabilita CALL 23 para todos los puertos activos y cadenas habilitadas por CALL 23 anteriores.
- 1 = Cadena interna solamente
- 2 = Puerto en serie 1
- 3 = Cadena interna y puerto en serie 1
- 4 = Puerto en serie 2
- 5 = Cadena interna y puerto en serie 2

Si se selecciona una cadena interna (1, 3 ó 5) el primer carácter de la cadena contiene el conteo de bytes. El segundo carácter (número de transacción) se incrementa para informar al módulo BASIC que hay datos nuevos en la cadena. El valor de este carácter vuelve de 255 a 0. Los datos del búfer fuente empiezan con el tercer carácter de la cadena.

El segundo argumento de entrada es la selección de la fuente de datos. Puede ser el archivo de datos de salida CPU o el archivo M0 CPU:

- 0 = Archivo de imagen de salida CPU
- 1 = Archivo M0 CPU

El tercer argumento de entrada es el offset de la palabra dentro del archivo de imagen de salida CPU o archivo M0. Este offset apunta a la primera palabra, la cual contiene el conteo de bytes de los datos válidos en el archivo. La segunda palabra del archivo CPU contiene los primeros dos caracteres de los datos que se van a transferir. Si se selecciona el archivo de imagen de salida CPU, este offset no debe ser la palabra 0 puesto que la palabra 0 está reservada para los bits de handshaking usados en este CALL. Por lo tanto, la primera palabra disponible para el conteo de bytes cuando se selecciona la imagen de salida es la segunda palabra (palabra 1).

El quinto argumento de entrada es el número de cadena interna. Si el segundo argumento de entrada no selecciona uso de cadena interna, el valor de este argumento de entrada es ignorado (pero debe ser PUSH). Si los datos exceden la longitud de la cadena, los datos restantes se truncan.

El quinto argumento de entrada es la selección de intercambio de bytes. Tiene los siguientes valores:

- 0 = Los bytes de datos transferidos desde la CPU *no* son intercambiados cuando pasan a la cadena o puerto del módulo BASIC. El orden de la transferencia de datos es byte inferior primero, luego byte superior por palabra. El byte inferior de la primera palabra en el búfer fuente contiene el conteo de bytes.
- 1 = Los bytes de datos transferidos desde la CPU son intercambiados cuando pasan a la cadena o puerto del módulo BASIC. El orden de la transferencia de datos es byte superior primero, luego byte inferior por palabra. El intercambio no afecta a la primera palabra. El byte inferior de la primera palabra en el búfer fuente todavía contiene el conteo de bytes.

El último CALL 23 ejecutado determina la opción de intercambio de bytes para todos los comandos de CALL 23 y CALL 22 activos previamente ejecutados.

El argumento de salida es el estado del CALL. Tiene los siguientes valores:

- 0 = Con éxito
- 1 = Inhabilitado
- 2 = Parámetro de entrada incorrecto
- 3 = PRT2 está seleccionado pero ya está habilitado para el protocolo DF1. El CALL no se ejecuta.
- 4 = La cadena es muy pequeña
- 5 = La cadena no está dimensionada

La transferencia de datos no empieza hasta que el procesador SLC establece la palabra 0 del archivo de imagen de salida, bit 6 o bit 7 (dependiendo del puerto de destino). El módulo BASIC establece la palabra 0 del archivo de entrada, bit 6 o bit 7 para indicar una transferencia efectuada a un puerto.

Si la cadena interna se selecciona como destino, la palabra 0 del archivo de imagen de salida, bit 6, se usa para iniciar la transferencia. El módulo BASIC establece la palabra 0 del archivo de entrada, bit 6, para indicar una transferencia efectuada a la cadena.

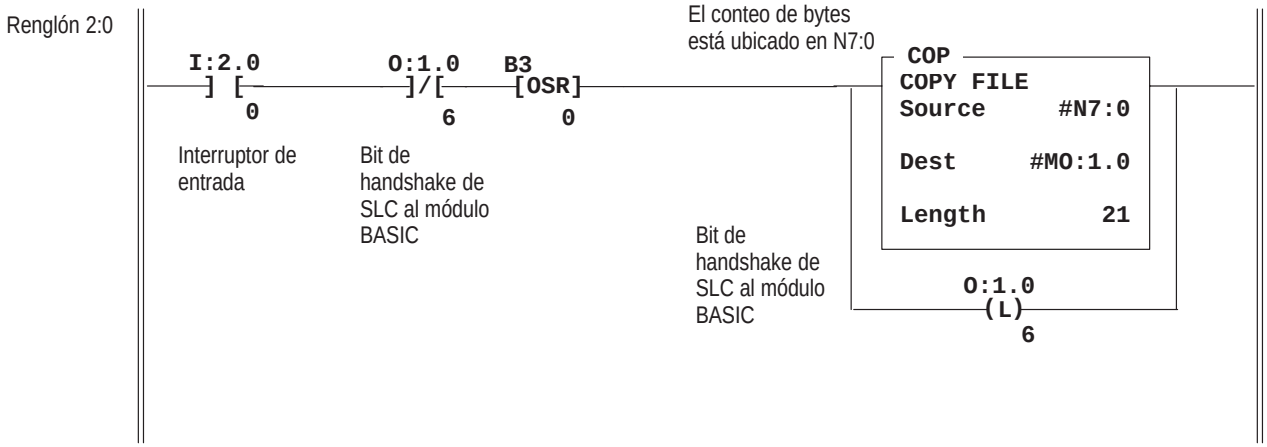
Sintaxis:

PUSH [número de puerto de destino y/o cadena interna]
PUSH [selección de archivo fuente]
PUSH [offset de palabra dentro del archivo fuente]
PUSH [número de cadena]
PUSH [selección de intercambio de bytes]
CALL 23
POP [estado del CALL 23]

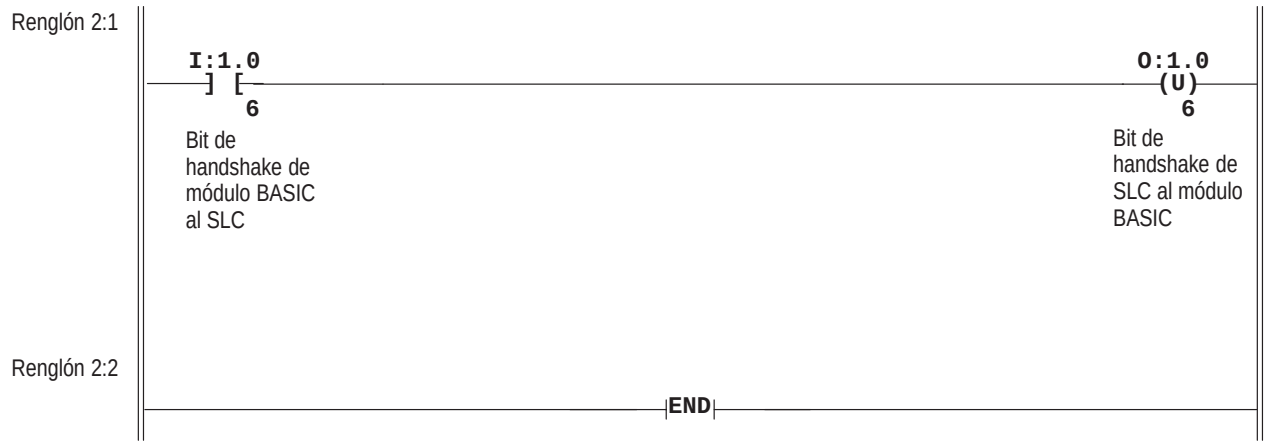
Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM ENABLE CALL 23 INTERRUPTS
>20 PUSH 2 : REM SEND DATA TO PRT1
>30 PUSH 1 : REM GET DATA FROM M0 FILE
>40 PUSH 0 : REM WORD OFFSET INTO M0 FILE
>50 PUSH 0 : REM STRING NUMBER/NOT USED HERE
>60 PUSH 1 : REM ENABLE BYTE SWAPPING
>70 CALL 23
>80 POP S : REM STATUS OF CALL SETUP
>90 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 23 SETUP”
```

A continuación proporcionamos un ejemplo de programa de lógica de escalera para el CALL 23. El módulo BASIC está ubicado en la ranura 1 del chasis SLC. En el renglón 2:0 copie los datos al archivo M0 y establezca el bit de handshake (O:1.0/6) para informar al módulo BASIC que hay datos nuevos disponibles para el puerto PRT1. No envíe más datos hasta que los datos previos hayan sido confirmados por el módulo BASIC. El archivo de datos para el módulo BASIC empieza con el conteo de bytes (40 en este ejemplo) en N7:0. Los datos siguen en N7:1–N7:20. La palabra de conteo de bytes de datos no está incluida en el conteo de bytes de datos. La longitud de la palabra de la instrucción copiar es 21. El conteo máximo de bytes es 40 bytes (20 palabras) en este ejemplo.



En el renglón 2:1, desenclave el bit de handshake del SLC (O:1.0/6) para el puerto PRT1 cuando el bit de handshake del módulo BASIC está establecido indicando que el módulo ha recibido los datos.



CALL 28 - Escritura al archivo de datos DH–485 SLC remoto

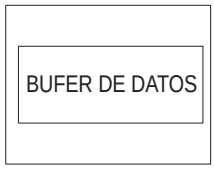
Objetivo:

Use CALL 28 para escribir hasta 40 palabras de datos desde el archivo de imagen de salida CPU, archivo M0 CPU y/o una cadena dentro del módulo BASIC al archivo DH–485 remoto en la dirección de nodo designada.

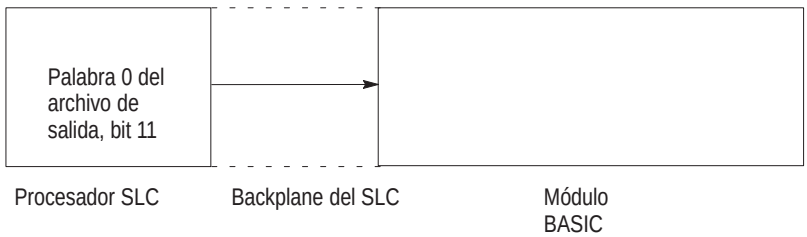
Si se selecciona una cadena interna, el primer carácter (número de transacción) se incrementa al completarse la escritura correctamente para informar al módulo BASIC que se enviaron los datos. El valor del número de transacción regresa de 255 a 0.

Ejecute CALL 28 una vez para configurar los parámetros de transferencia de datos. Los bits de imagen de entrada y salida (palabra 0, bit 11) para la ranura que contiene el módulo BASIC, se usan para iniciar y notificar que se ha completado la transferencia. La operación se describe a continuación:

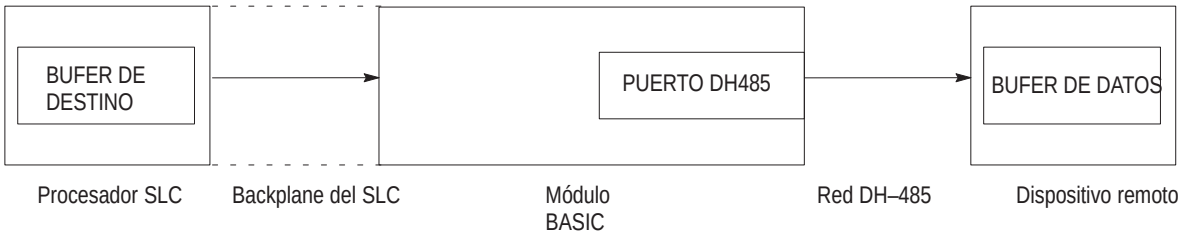
1. El procesador SLC local construye el búfer de datos.



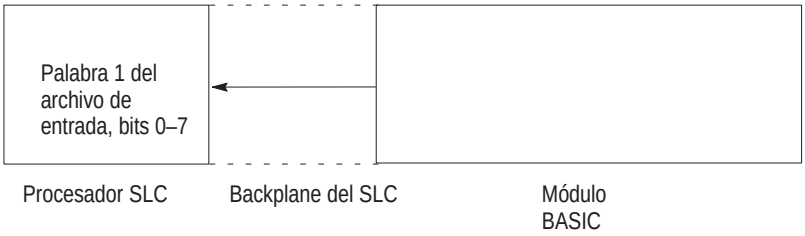
2. El procesador SLC establece la palabra 0 del archivo de salida, bit 11, para informar al módulo BASIC que los datos pueden ser transferidos.



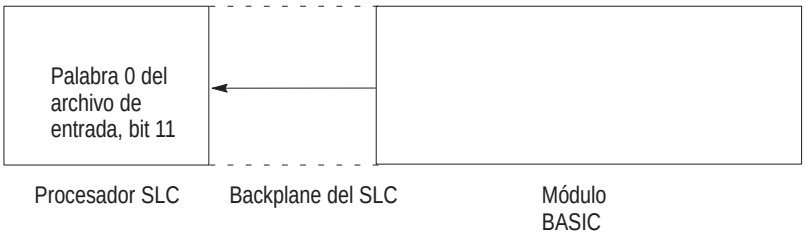
3. El módulo BASIC transfiere los datos al búfer de datos remoto.



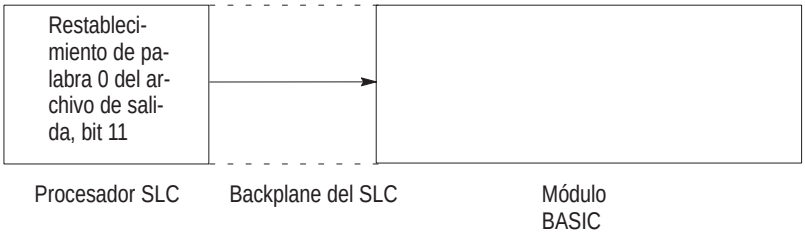
4. El módulo BASIC coloca el estado de la transferencia en la palabra 1 del archivo de entrada, bits 0–7.



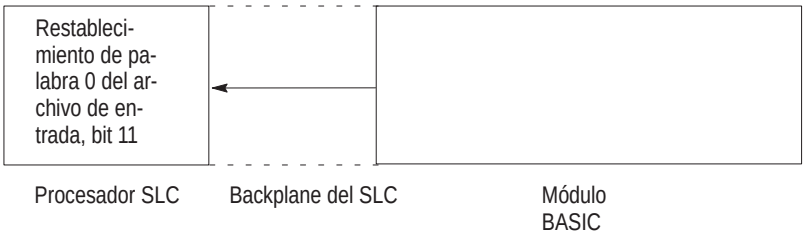
5. El módulo BASIC establece la palabra 0 del archivo de entrada, bit 11 para informar al procesador SLC local que datos válidos han sido transferidos y el estado de la transferencia está disponible.



6. El SLC recupera el estado y restablece la palabra 0 del archivo de salida, bit 11. El búfer de datos puede ser reutilizado por el SLC.



7. El módulo BASIC restablece la palabra 0 del archivo de entrada, bit 11, en el mismo ciclo de fin de escán en el que la palabra 0 del archivo de salida, bit 11 fue restablecida.



Una vez que este CALL esté activo, permanecerá activo y enviará datos al nodo remoto cada vez que el procesador SLC establezca la palabra 0 del archivo de salida, bit 11.

Este CALL tiene diez argumentos de entrada y un argumento de salida.

El primer argumento de entrada es el tipo de comando de escritura WRITE SLC emitido:

- 0 = Inhabilita el CALL 28 ejecutado previamente
- 1 = Escritura de archivo de interface común
- 2 = Mensaje de escritura Write SLC

El segundo argumento de entrada es la dirección de nodo del dispositivo remoto SLC en la red DH–485 (0 a 31). Si el número no está dentro de este rango, el estado es igual a 2 y el mensaje de escritura no se realiza.

El tercer argumento de entrada es el número de archivo en el dispositivo remoto SLC (0 a 255). Si el número no está dentro de este rango, el estado es igual a 2 y el mensaje de escritura no se realiza. El parámetro es ignorado si se selecciona el archivo de interface común en el primer parámetro. CIF siempre es el archivo 9.

El cuarto argumento de entrada es el tipo de archivo a ser escrito al dispositivo remoto. Este número es ignorado si se selecciona el CIF en el primer parámetro (supone archivo de enteros). Si el tipo de archivo no es uno de los listados a continuación, el estado es igual a 2 y el mensaje de escritura no se realiza. Introduzca el código de tipo de archivo tal como se muestra:

Tabla 12.B
Tipo de archivo

Tipo de archivo	Código de tipo de archivo	Palabras/elemento
Archivo de enteros	ASC(N)	1 palabra/elemento
Archivo de contador	ASC(C)	3 palabras/elemento
Archivo de temporizador	ASC(T)	3 palabras/elemento
Archivo de bit	ASC(B)	1 palabra/elemento
Archivo de control	ASC(R)	3 palabras/elemeno

El quinto argumento de entrada es el offset de palabra inicial dentro del archivo en el dispositivo remoto SLC (0 a 32766). Si el número no está dentro de este rango, el estado es igual a 2 y la transferencia no se realiza.

El sexto argumento de entrada es el número de palabras que se va a transferir. Si el número no está dentro del rango mostrado, el estado es igual a 2 y la transferencia no se realiza.

Tabla 12.C
Rango de longitud válida

Código de tipo de archivo	Rango de longitud válida
ASC(N)	1 a 40
ASC(C)	1 a 13
ASC(T)	1 a 13
ASC(B)	1 a 40
ASC(R)	1 a 13
Archivo de interface común	1 a 40

El séptimo argumento de entrada es el valor de tiempo límite del mensaje. Este valor (1 a 255) corresponde al número de centésimas de milisegundos permitido para recibir la respuesta de escritura (0.1 a 25.5 segundos). Si la respuesta de escritura no se recibe dentro de este tiempo, el mensaje se cancela con el estado igual a 55 en la palabra 1 del archivo de entrada, bits 0–7. Si el valor de tiempo límite no está dentro de este rango (1 a 255), el estado POP es igual a 2 y la transferencia no se realiza.

El octavo argumento de entrada es la selección del archivo de imagen de salida CPU fuente, archivo M0 CPU, o la cadena interna.

- 0 = Archivo de imagen de salida CPU
- 1 = Archivo M0 CPU
- 2 = Cadena interna

Si selecciona la cadena interna (2), el CALL 29 puede ejecutarse para iniciar cada transferencia de datos sin requerir interacción del procesador SLC. La palabra 0 del archivo de salida, bit 11, también inciará una transacción de cadena.

El noveno argumento de entrada es el offset de palabra dentro del archivo CPU fuente. El offset de palabra apunta a la primera ubicación de datos. Si usted selecciona el archivo de imagen de salida CPU, el offset no debe ser 0 puesto que esta palabra está reservada para los bits de handshaking de transferencia de datos. El offset para la cadena interna siempre es 1. El carácter de cadena (número de transacción) en la ubicación 0 se incrementa con cada transferencia de datos.

El décimo argumento de entrada es el número de cadena. Si el octavo argumento de entrada no selecciona uso de cadena interna, el valor de este argumento de entrada es ignorado pero debe ser PUSH.

El argumento de salida es la validación de los parámetros del CALL. Tiene los siguientes valores:

- 0 = Con éxito
- 1 = Inhabilitado
- 2 = Parámetro de entrada incorrecto
- 3 = Puerto DH485 no habilitado (DF1 habilitado)
- 4 = Cadena demasiado pequeña
- 5 = La cadena no está dimensionada
- 6 = El paquete de datos es muy largo

Para inhabilitar este CALL, un cero debe ser PUSH en el primer parámetro de entrada. Todos los otros parámetros son ignorados, pero deben ser PUSH.

Cada vez que se intenta escribir a un nodo remto, el módulo BASIC coloca el estado de la escritura en la palabra 1 del archivo de entrada SLC, bits 0–7. Los valores de estado tienen la misma definición que los valores a los cuales se les aplicó POP en CALL 93.

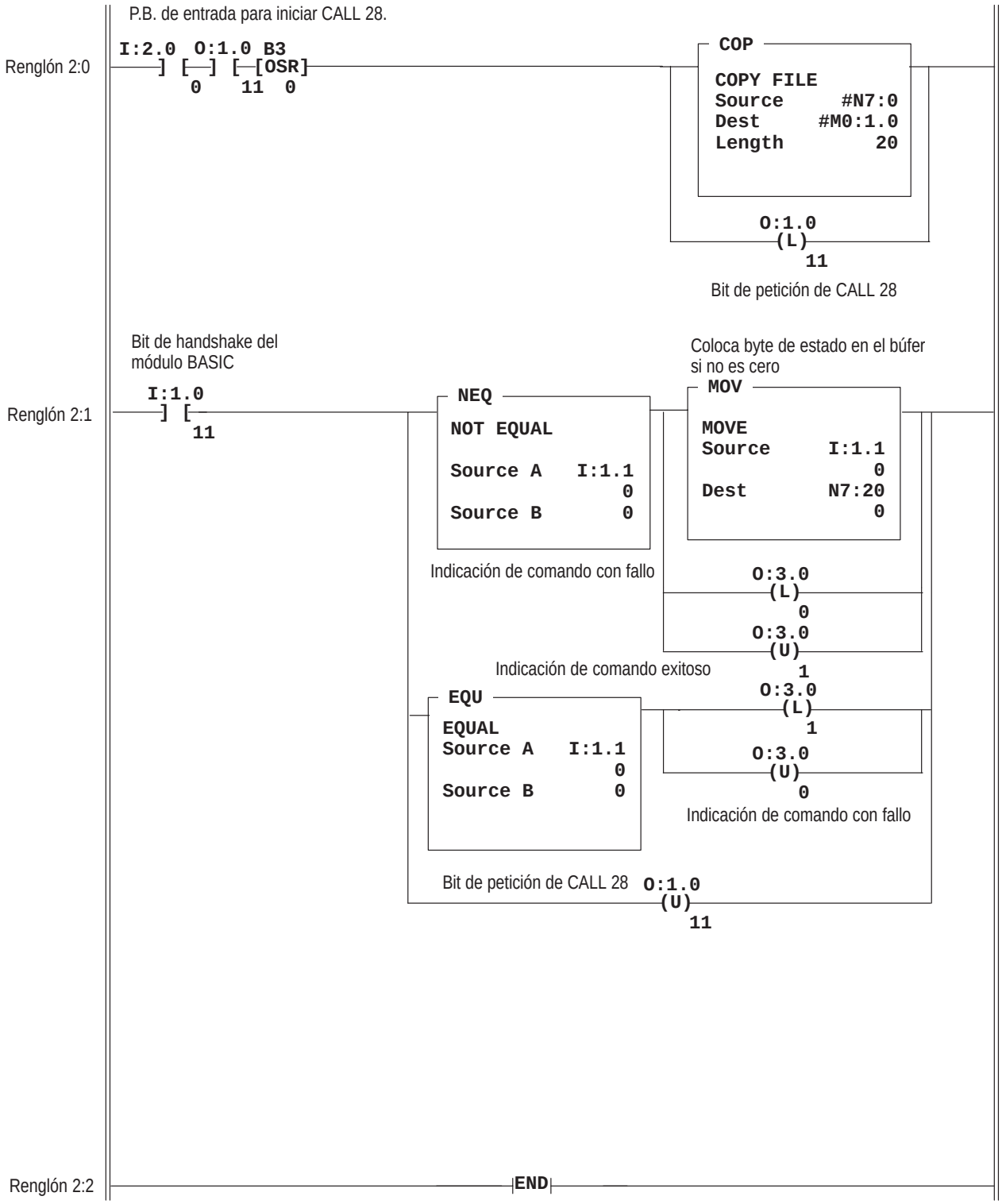
Sintaxis:

PUSH [tipo de comando de ESCRITURA]
PUSH [dirección de nodo remoto]
PUSH [número de archivo remoto]
PUSH [tipo de archivo remoto]
PUSH [offset de palabra inicial remota]
PUSH [número de palabras que va a ser transferido}
PUSH [valor de tiempo límite de mensaje]
PUSH [selección de archivo fuente]
PUSH [offset de palabra dentro de archivo fuente]
PUSH [número de cadena]
CALL 28
POP [estado de CALL 28]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE INTERRUPTS FOR READ COMMANDS TO A
     REMOTE DH–485 NODE
>20  PUSH 2 : REM SLC TYPED WRITE
>30  PUSH 1 : REM REMOTE SLC NODE NUMBER
>40  PUSH 7 : REM REMOTE SLC FILE NUMBER
>50  PUSH ASC(N) : REM REMOTE SLC FILE TYPE
>60  PUSH 0 : REM ELEMENT OFFSET INTO DESTINATION FILE
>70  PUSH 20 : REM NUMBER OF ELEMENTS TO BE TRANSFERRED
>80  PUSH 10 : REM MESSAGE TIME–OUT VALUE(X100MS)
>90  PUSH 1 : REM LOCAL SLC FILE TYPE (M0)
>100 PUSH 0 : REM WORD OFFSET INTO LOCAL SLC FILE
>110 PUSH 0 : REM INTERNAL STRING NUMBER – NOT USED FOR THIS
     EXAMPLE
>120 CALL 28
>130 POP S
>140 IF (S=1) THEN PRINT “CALL 28 DISABLED”
>150 IF (S=2) THEN PRINT “CALL 28 BAD INPUT PARAMETER”
>160 IF (S=3) THEN PRINT “PORT DH485 NOT ENABLED”
```

Copie los datos que se van a enviar al nodo DH–485 remoto en el archivo local (el archivo M0 asociado con el número de ranura del módulo BASIC). Enclave el bit de petición del CALL 28 (palabra 0, bit 11).



CALL 29 - Lectura/escritura a un PLC/SLC desde cadena interna del módulo BASIC

Objetivo:

Use CALL 29, junto con CALL 122 ó 123, para comunicarse entre PLC remotos y la cadena interna del módulo BASIC sin interacción del procesador SLC local. También puede usar el CALL 29, junto con CALL 27 ó 28, para comunicarse entre SLC remotos y la cadena interna del módulo BASIC sin interacción del procesador SLC local. Los CALL 27, 28, 122, ó 123 deben ejecutarse dentro del programa BASIC antes que el CALL 29.

El CALL 29 está activo cuando la cadena interna es la única selección en CALL 27, 28, 122, ó 123. En esta situación, no es práctico usar los archivos de imagen de entrada y salida SLC para empezar la transferencia y pasar el estado. El procesador SLC no necesita interactuar. Si se selecciona un archivo SLC en su lugar, el procesador SLC local controla la transferencia con los bits de imagen de entrada y salida. En este caso, cuando se intenta el CALL 29, se devuelve un estado de 255.

Un argumento se PUSH y un argumento se POP. El argumento de entrada es el CALL que va a ser activado (CALL 27, 28, 122 ó 123).

Cuando se ejecuta CALL 29, se intenta realizar la transferencia. Si el CALL seleccionado (27, 28, 122 ó 123) no se ejecuta antes que el CALL 29, se devuelve un 1 en el estado POP. Cuando se ejecuta correctamente CALL 29, el valor en el primer carácter de la cadena (número de transacción) se incrementa para indicar que se realizó la transferencia. El rango de este carácter es 0–255.

Después que se ejecuta el CALL 29, se POP una palabra. Esta palabra es el estado de la transacción:

- 0 = Se completó correctamente
- 1 = El CALL seleccionado (27, 28, 122 ó 123) no está activo
- 255 = Búfer SLC es seleccionado para CALL 27, 28, 122 ó 123 y CALL 29 es ignorado
- Todos los otros códigos son idénticos a CALL 90/92

Sintaxis:

PUSH [27, 28, 122 ó 123 para el CALL que usted desea activado]
CALL 29
POP [estado de transacción]

Ejemplo:

CALL 122 debe estar habilitado con la cadena interna solamente antes de ejecutar CALL 29 en este ejemplo. Después de la ejecución de CALL 29, se intenta transferir un elemento del archivo de enteros 10, empezando en el elemento 0 del PLC–5® en el nodo 3, a la cadena interna $(1) del módulo BASIC.

```
>1   REM EXAMPLE PROGRAM
>10  REM EXECUTE DF1 PLC REMOTE READ FROM INTERNAL
>20  REM STRING WITH NO SLC INTERVENTION
>21  REM SET UP CALL 122
>25  PUSH 5, 3, 10, ASC(N), 0, 10, 10, 1, 1, 1: CALL 122:
     POP STATUS
>30  PUSH 122
>40  CALL 29
>50  POP S
>60  IF (S=1) THEN PRINT “CALL 122 NOT ACTIVE”
>70  IF (S=255) THEN PRINT “SLC FILE CHOSEN FOR CALL 122”
>80  IF (S=0) THEN PRINT “SUCCESSFUL TRANSFER”
>90  IF (S<>0) THEN PRINT “UNSUCCESSFUL TRANSFER”
>100 END
```

CALL 29 no requiere handshaking de bit SLC al final del comando, a diferencia de CALL 27, 28, 122 y 123 cuando se usa un archivo SLC como fuente o destino.

CALL 31 - Mostrar la configuración actual del puerto PRT2

Objetivo:

Use CALL 31 para mostrar la configuración actual del puerto PRT2 en la pantalla del terminal del puerto de programación. Ningún argumento se PUSH o POP.

Sintaxis:

CALL 31

Ejemplo:

```
>CALL 31

1200 Baud
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff

READY
>
```

CALL 37 - Borrado de los búfers de entradas/salidas del PRT2

Objetivo:

Use CALL 37 para borrar los búfers de entradas y/o salidas del puerto periférico. Use los siguientes PUSH para borrar el búfer corespondiente:

- PUSH 0 para borrar el búfer de salidas
- PUSH 1 para borrar el búfer de entradas
- PUSH 2 para borrar ambos búfers

Sintaxis:

PUSH [selección de búfer]
CALL 37

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : REM CLEARS THE OUTPUT BUFFER
>20 CALL 37
>30 END

READY
>RUN

READY
>
```

CALL 54 - Transferencia del búfer de salidas BASIC a la imagen de entrada de la CPU

Objetivo:

Use CALL 54 para transferir las palabras 200 a 207 del búfer de salidas del módulo BASIC a las palabras 0 a 7 de la tabla de imagen de entrada de la CPU. Esta rutina no tiene ningún argumento de entrada pero tiene un argumento de salida. El argumento de salida es el estado del procesador lógico.

La palabra 200 en el búfer de salidas BASIC está reservada y no puede modificarse. Esta palabra proporciona información de estado del módulo al procesador SLC 500. El bit 15 es el bit de modo del módulo. Puede tener uno de los siguientes valores:

- 0 = El procesador SLC está en el modo de marcha (Run)
- 1 = El procesador SLC no está en el modo de marcha

El bit 14 de la palabra 200 es el bit de suma de comprobación de EEPROM. Puede tener uno de los siguientes valores:

- 0 = La suma de comprobación de EEPROM es correcta
- 1 = La suma de comprobación de EEPROM es incorrecta

El bit 13 de la palabra 200 es el bit de estado de la batería. Puede tener uno de los siguientes valores:

- 0 = La batería está en buen estado
- 1 = La batería está baja

La integridad de la palabra se garantiza durante esta transferencia. La integridad del archivo no se garantiza. Los bits de handshaking pueden usarse en su programa de aplicación para proporcionar integridad al archivo.

Sintaxis:

CALL 54
POP [modo del procesador]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>30 CALL 54 : REM XFER BASIC OUTPUT BUFFER TO CPU INPUT
    IMAGE TABLE
>40 POP X : REM LOGIC PROCESSOR STATUS IS IN X
>50 IF X<>0 THEN PRINT “PROCESSOR NOT IN RUN MODE”

READY
>RUN

READY
>
```

CALL 57 - Transferencia del búfer de salidas BASIC al archivo M1 de la CPU

Objetivo:

Use CALL 57 para transferir hasta 64 palabras empezando en la palabra 100 desde el búfer de salidas del módulo BASIC al archivo M1 de la CPU empezando en la palabra 0. Esta rutina tiene un argumento de entrada y un argumento de salida. El argumento de entrada es el número de palabras que se va a transferir (1 a 64). Si el número no está dentro del rango de 1 a 64, no se realiza la transferencia y el argumento de salida se establece en 10.

Si el argumento de entrada es un número válido, entonces el argumento de salida es el estado del procesador lógico. Puede tener uno de los siguientes valores:

- 0 = Transferencia efectuada, el procesador SLC está en el modo de marcha (Run)
- 1 = Transferencia efectuada, el procesador SLC está en el modo de programación
- 2 = Transferencia efectuada, el procesador SLC está en el modo de prueba (Test)
- 10 = Se especificó longitud ilegal
- 11 = El procesador SLC no tiene esta capacidad

La integridad de la palabra se garantiza durante esta transferencia. La integridad del archivo no se garantiza. Los bits de handshaking pueden usarse en su programa de aplicación para proporcionar integridad al archivo.

Sintaxis:

PUSH [número de palabras a ser transferido]
CALL 57
POP [modo del procesador]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>50 PUSH 64 : REM TRANSFER LENGTH IS 64 WORDS
>60 CALL 57 : REM TRANSFER BASIC OUTPUT BUFFER TO M1
>70 POP X : REM LOGIC PROCESSOR STATUS IS IN X
>80 PRINT X

READY
>RUN 0

READY
>
```

CALL 85 - Transferencia del búfer de salidas BASIC al archivo de interface común DH-485

Objetivo:

Use CALL 85 para transferir hasta 40 palabras al archivo de interface común DH-485.

Esta rutina tiene dos argumentos de entrada y un argumento de salida. El primer argumento de entrada es el offset de la palabra inicial del archivo de interface común DH-485. Si el valor de offset de la palabra no está dentro del rango de 0 a 127, el argumento de salida es igual a 1. El segundo argumento de entrada es el número de palabras que se va a transferir desde el búfer de salida del módulo BASIC al archivo de interface común DH–485. Si el número del segundo argumento de entrada no está dentro del rango de 1 a 40, el argumento de salida es igual a 2.

El argumento de salida especifica el estado de la transferencia. Puede tener uno de los siguientes valores:

- 0 = Transferencia efectuada correctamente

- 1 = Offset inicial no válido
- 2 = Longitud no válido

La integridad de la palabra se garantiza durante esta transferencia. La integridad del archivo no se garantiza.

Sintaxis:

PUSH [offset de palabra inicial en archivo de interface DH-485]
PUSH [número de palabras que se va a transferir]
CALL 85
POP [estado de la transferencia]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>40 PUSH 31 : REM OFFSET ADDRESS = 31
>50 PUSH 3 : REM WORD LENGTH = 3
>60 CALL 85 : REM TRANSFER DATA TO DH-485 COMMON INTERFACE
    FILE
>70 POP R
>80 IF R<>0 PRINT “TRANSFER ERROR CODE = ”,R : REM PRINT
    ERROR

READY
>RUN

READY
>
```

CALL 91 - Escritura del búfer de salidas BASIC al archivo de datos DH-485 remoto

Objetivo:

Use CALL 91 para escribir hasta 40 palabras empezando en la palabra 0 del búfer de salidas del módulo BASIC al archivo de datos DH-485 remoto en la dirección del nodo designado, número de archivo, tipo de archivo y offset de elemento. Esta rutina tiene seis argumentos de entrada y un argumento de salida.

El primer argumento de entrada es la dirección de nodo del dispositivo remoto (0 a 31). Si el número no está dentro del rango de 0 a 31, entonces el argumento de salida es igual a 10, y el mensaje de escritura no se realiza.

El segundo argumento de entrada es el número de archivo en el dispositivo remoto (0 a 255). Si el número no está dentro del rango de 0 a 255, el argumento de salida es igual a 11 y el mensaje de escritura no se realiza.

El tercer argumento de entrada es el tipo de archivo escrito al dispositivo remoto. Los códigos de tipo válidos son ASC(N), ASC(S), ASC(C), ASC(T), ASC(B) y ASC(R). Si el tipo de archivo no es uno de estos tipos válidos, el argumento de salida es igual a 241 y el mensaje de escritura no se realiza.

Tabla 12.D
Tipo de archivo escrito al dispositivo remoto

Tipo de archivo	Código de tipo de archivo	Palabras/elemento
Archivo de enteros	ASC(N)	1 palabra/elemento
Archvio de estado	ASC(S)	1 palabra/elemento
Archivo de contador	ASC(C)	3 palabras/elemento
Archivo de temporizador	ASC(T)	3 palabras/elemento
Archvio de bit	ASC(B)	1 palabra/elemento
Archivo de control	ASC(R)	3 palabras/elemento

El cuarto argumento es el offset de elemento inicial dentro del archivo en el dispositivo remoto (0 a 32767). Si el número no está dentro del rango (0 a 32767), entonces el argumento de salida es igual a 12 y la transferencia no se realiza.

Importante: El offset será el doble de lo esperado. Por ejemplo, si un offset de 3 fue PUSH, los datos serán escritos al archivo de datos DH–485 remoto empezando en el elemento 6.

El quinto argumento es el número de elementos que se va a transferir. Si el número no está dentro del rango especificado a continuación, el argumento de salida es igual a 13 y la transferencia no se realiza.

Tabla 12.E
Rango de longitud válida

Tipo de archivo	Rango de longitud válida
ASC(N)	1 to 40
ASC(S)	1 to 40
ASC(C)	1 to 13
ASC(T)	1 to 13
ASC(B)	1 to 40
ASC(R)	1 to 13

El sexto argumento es el valor de tiempo límite del mensaje. Este valor es el número de centenas de milisegundos permitidos para recibir la respuesta de escritura (1 a 50 = 0.1 a 5.0 segundos). Si no se recibe la respuesta de escritura dentro de este tiempo, el mensaje se cancela con el argumento de salida igual a 55. Si el número no está dentro del rango de 1 a 50, el argumento de salida es igual a 14 y la transferencia no se realiza.

Los datos de escritura del búfer de salidas del módulo BASIC se escriben al dispositivo remoto empezando en la palabra 0 hasta el número de palabras especificado por la longitud de elemento del mensaje.

El argumento de salida especifica el estado de la instrucción del mensaje. Cuando el CALL responde, el argumento de salida tiene la siguiente definición.

Tabla 12.F
Argumento de salida

Salida decimal	Salida hexadecimal	Descripción
0	00	Realización efectuada correctamente
2	02	El nodo de destino no puede aceptar ahora el mensaje.
3	03	El nodo de destino no puede responder porque el mensaje es muy largo.
4	04	El nodo de destino no puede responder porque no entiende los parámetros de comando.
5	05	El módulo BASIC está fuera de línea (no está en la red).
6	06	El nodo de destino no puede responder porque la función solicitada no está disponible.
7	07	El nodo de destino no responde.
10	0A	El módulo BASIC detecta dirección ilegal de nodo de destino.
11	0B	El módulo BASIC detecta número de archivo ilegal.
12	0C	El módulo BASIC detecta offset de elemento ilegal de archivo de destino.
13	0D	El módulo BASIC detecta longitud ilegal de archivo de destino.
14	0E	El módulo BASIC detecta valor de tiempo límite ilegal.
16	10	El nodo de destino no puede responder debido a parámetros de comando incorrectos o comando no aceptado.
55	37	Mensaje sobrepasó el tiempo permitido (valor de tiempo límite excedido)
80	50	Se agotó la memoria en el modo de destino.
96	60	El nodo de destino no puede responder porque el archivo está protegido.

Salida decimal	Salida hexadecimal	Descripción
231	E7	El nodo de destino no puede responder porque la longitud solicitada es muy larga.
235	EB	El nodo de destino no puede responder porque el nodo de destino niega el acceso.
236	EC	El nodo de destino no puede responder porque la función solicitada actualmente no está disponible.
241	F1	El módulo BASIC detecta tipo de archivo de destino ilegal.
250	FA	El nodo de destino no puede responder porque otro nodo es el propietario del archivo (tiene acceso único al archivo).
251	FB	El nodo de destino no puede responder porque otro nodo es el propietario del programa (tiene acceso único a todos los archivos).

Este CALL se implementa como escritura lógica tipo protegido con dos campos de dirección.

Sintaxis:

PUSH [dirección de nodo de dispositivo remoto]
PUSH [número de archivo de dispositivo remoto]
PUSH [tipo de archivo de dispositivo remoto]
PUSH [offset de elemento inicial (x2) de archivo de dispositivo remoto]
PUSH [número de elementos que se van a transferir]
PUSH [valor de tiempo límite de mensaje]
CALL 91
POP [estado de instrucción de mensaje]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>20 PUSH 7 : REM REMOTE FILE 7
>30 PUSH ASC(N) : REM FILE TYPE = INTEGER
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10: REM WORD LENGTH = 10
>60 PUSH 5 : REM THE TIME-OUT VALUE = 0.5 SECOND
>70 CALL 91 : REM WRITE DATA FROM OUTPUT BUFFER
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF R<>0 PRINT "READ ERROR CODE =",R

READY
>RUN

READ ERROR CODE =  5

READY
>
```

CALL 93 - Escritura del búfer de salidas al archivo de interface común DH-485 remoto

Objetivo:

Use CALL 93 para escribir hasta 40 palabras empezando en la palabra 0 del búfer de salidas del módulo BASIC al archivo de interface común DH-485 remoto en la dirección de nodo designada, empezando en el offset de la palabra dsignada. Esta rutina tiene cuatro argumentos de entrada y un argumento de salida.

El primer argumento de entrada es la dirección de nodo del dispositivo remoto (0 a 31). Si el número no está dentro del rango de 0 a 31, el argumento de salida es igual a 10 y el mensaje de escritura no se realiza.

El segundo argumento es el offset de la palabra inicial dentro del archivo en el dispositivo remoto (0 a 32767). Si el número no está dentro del rango (0 a 32767), el argumento de salida es igual a 12 y la transferencia no se realiza.

Importante: El offset será el doble de lo esperado. Por ejemplo, si un offset de 3 fue PUSH, los datos serán escritos al archivo de datos DH–485 remoto empezando en el elemento 6.

El tercer argumento es el número de palabras que se va a transferir. Si el número no está dentro del rango especificado a continuación, el argumento de salida es igual a 13 y la transferencia no se realiza.

El cuarto argumento es el valor de tiempo límite del mensaje. Este valor es el número de centenas de milisegundos permitidos para recibir la respuesta de escritura (1 a 50 = 0.1 a 5.0 segundos). Si no se recibe la respuesta de escritura dentro de este tiempo, el mensaje se cancela con el argumento de salida igual a 55. Si el número no está dentro del rango de 1 a 50, el argumento de salida es igual a 14 y la transferencia no se realiza.

Los datos del búfer de salidas del módulo BASIC, empezando en la palabra 0, son escritos al archivo de interface común remoto empezando en la palabra especificada, hasta el número de palabras especificado por la longitud de palabra del mensaje.

El argumento de salida especifica el estado de la instrucción del mensaje. Cuando el CALL responde, el argumento de salida tiene la siguiente definición.

Tabla 12.G
Argumento de salida

Salida decimal	Salida hexadecimal	Descripción
0	00	Realización efectuada correctamente.
2	02	El nodo de destino no puede aceptar ahora el mensaje.

Salida decimal	Salida hexadecimal	Descripción
3	03	El nodo de destino no puede responder porque el mensaje es muy largo.
4	04	El nodo de destino no puede responder porque no entiende los parámetros de comando.
5	05	El módulo BASIC está fuera de línea (no está en la red).
6	06	El nodo de destino no puede responder porque la función solicitada no está disponible.
7	07	El nodo de destino no responde.
10	0A	El módulo BASIC detecta dirección ilegal de nodo de destino.
11	0B	El módulo BASIC detecta número de archivo ilegal.
12	0C	El módulo BASIC detecta offset de elemento ilegal de archivo de destino.
13	0D	El módulo BASIC detecta longitud ilegal de archivo de destino.
14	0E	El módulo BASIC detecta valor de tiempo límite ilegal.
16	10	El nodo de destino no puede responder debido a parámetros de comando incorrectos o comando no aceptado.
55	37	Mensaje sobrepasó el tiempo permitido (valor de tiempo límite excedido)
80	50	El nodo de destino está sin memoria.
96	60	El nodo de destino no puede responder porque el archivo está protegido.
231	E7	El nodo de destino no puede responder porque la longitud solicitada es muy larga.
235	EB	El nodo de destino no puede responder porque el nodo de destino niega el acceso.
236	EC	El nodo de destino no puede responder porque la función solicitada actualmente no está disponible.
241	F1	El módulo BASIC detecta tipo de archivo de destino ilegal.
250	FA	El nodo de destino no puede responder porque otro nodo es el propietario del archivo (tiene acceso único al archivo).
251	FB	El nodo de destino no puede responder porque otro nodo es el propietario del programa (tiene acceso único a todos los archivos).

Sintaxis:

PUSH [dirección de nodo de dispositivo remoto]
PUSH [offset de elemento inicial (x2) de archivo de dispositivo remoto]
PUSH [número de palabras que se van a transferir]
PUSH [valor de tiempo límite de mensaje]
CALL 93
POP [estado de instrucción de mensaje]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>30 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10 : REM WORDLENGTH = 10
>60 PUSH 5 : REM THE TIME-OUT VALUE = 0.5 SECONDS
>70 CALL 93 : REM WRITE DATA FROM BASIC OUTPUT BUFFER
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF R<>0 THEN PRINT  READ ERROR CODE = ",R


READY
>RUN

READ ERROR CODE =  5

READY
>
```

CALL 94 - Mostrar la configuración actual del puerto PRT1

Objetivo:

Use CALL 94 para mostrar la configuración actual del puerto PRT1 en la pantalla del terminal del puerto del programa. Ningún argumento se PUSH ni POP.

Sintaxis:

CALL 94

Ejemplo:

```
>CALL 94

19200 Baud
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff
```

CALL 96 - Borrado de los búfers de entradas/salidas del PRT1

Objetivo:

Use CALL 96 para borrar los búfers de entradas y salidas del puerto PRT1. Use los siguientes PUSH para borrar el búfer correspondiente:

- PUSH 0 para borrar el búfer de salidas
- PUSH 1 para borrar el búfer de entradas
- PUSH 2 para borrar ambos búfers

Importante: Si el puerto PRT1 está configurado para el protocolo DH-485, este CALL no tiene efecto.

Sintaxis:

PUSH [selección de búfer]
CALL 96

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 0 : CALL 96 : REM CLEAR PRT1 OUTPUT BUFFER

READY
>
```

CALL 112 - Control de LED del usuario

Objetivo:

Use CALL 112 para activar o desactivar los LED del usuario (LED1 y LED2). Se requieren dos argumentos de entrada y ningún argumento de salida. El primer argumento de entrada activa o desactiva el LED1. El segundo argumento de entrada activa o desactiva el LED2. Un argumento de entrada de uno activa el LED. Un argumento de entrada de cero desactiva el LED. Cualquier otro valor no tiene efecto sobre ese LED particular.

Sintaxis:

PUSH [estado del LED1]
PUSH [estado del LED2]
CALL 112

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>100 PUSH 1 : REM TURN ON LED1
>110 PUSH 0 : REM TURN OFF LED2
>120 CALL 112 : REM SET THE LEDS

READY
>RUN

READY
>
```

CALL 114 - Transmisión del paquete DF1

Objetivo:

Use CALL 114 para transmitir el paquete de datos DF1. Esta rutina no tiene argumentos de entrada ni de salida. Cuando se ejecuta el CALL 114, los datos DF1 se muestran para que el controlador DF1 transmita un solo paquete de mensaje. Si se selecciona operación half–duplex esclava, el paquete de mensaje se transmite la siguiente vez que se recibe un ENQuiry del DF1 maestro. Si se selecciona operación full–duplex, el paquete de mensaje se transmite inmediatamente.

Use una o más instrucciones PRINT#, PH0.#, o PH1.# para elaborar los datos deseados en el búfer de transmisión del puerto PRT2. Después de elaborar los datos en el búfer de transmisión, use CALL 114 para iniciar la transmisión de los datos dentro de un paquete de mensaje DF1.

Se debe tener cuidado al elaborar los paquetes de datos DF1. Si se intenta transmitir cinco o menos bytes de datos (el mínimo es seis bytes), el mensaje **`ERROR: DF1 DATA PACKET TO TRANSMIT IS TOO SMALL`** se envía al puerto de programación y el módulo BASIC entra al modo de comando.

Si se intenta colocar más de 256 bytes de datos en el búfer de transmisión, el mensaje **`ERROR: BUFFER OVERFLOW`** se envía al puerto de programación y el módulo BASIC entra al modo de comando.

El programa del usuario debe esperar que se complete una transmisión antes de elaborar otro paquete de datos. Use CALL 115 para verificar el estado de la transmisión DF1 y determinar cuando se ha completado una transmisión.

Sintaxis:

CALL 114

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 114
>20 END
```

CALL 115 - Verificación del estado del DF1 XMIT

Objetivo:

Use CALL 115 para verificar el estado de la transmisión DF1. Esta rutina no tiene ningún argumento de entrada y tiene un argumento de salida. El argumento de salida devuelve un valor que representa el estado de la transmisión DF1. Los valores de estado de transmisión DF1 posibles se muestran a continuación:

- 0 = No transmite el resultado
- 1 = Transmite el resultado pendiente
- 2 = Transmisión efectuada correctamente
- 3 = Transmisión fallida
- 4 = Consulta fuera de tiempo, ninguna transmisión
- 5 = Si se ha seleccionado handshaking de módem, puede haber ocurrido una pérdida de la señal CTS durante la transmisión o un fallo irremediable de la transmisión. Si no se ha seleccionado handshaking, ha ocurrido un fallo irremediable de transmisión.
- 6 = Si se ha seleccionado handshaking de modem con portadora constante para los modos half–duplex o full–duplex, este error indica fallo de transmisión debido a desconexión del modem (pérdida de la señal DCD durante más de 10 segundos).
- 7 = El controlador DF1 no está habilitado.

Importante: El valor de estado de transmisión 4 nunca debe retornarse si se selecciona el modo full–duplex.

Sintaxis:

CALL 113
POP [estado de transmisión DF1]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 113
>20 POP X
>30 END
```

CALL 123 - Escritura a archivo de datos PLC DF1 remoto

Objetivo:

Use CALL 123 para escribir hasta 64 palabras de datos desde el archivo de imagen de salida CPU, el archivo M0 CPU y/o una cadena dentro del módulo BASIC a un nodo DF1 remoto (PLC–2®, –3®, ó –5).

La siguiente tabla lista notas específicas cuando se usa CALL 123 con el PLC–3 y PLC–5.

Tabla 12.H
Notas de aplicación de PLC

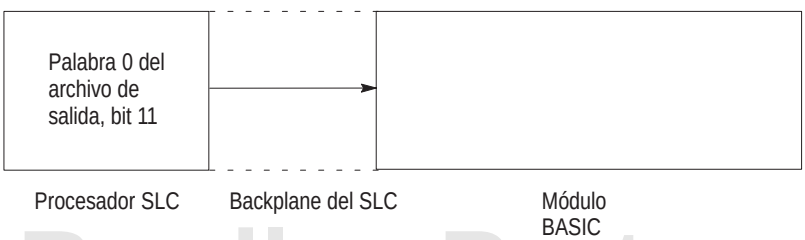
PLC	Notas
–3	Para temporizadores y contadores, el número de archivo PUSH (tercer parámetro) es el número de estructura, limitado a un máximo de 255 palabras.
	No se puede tener acceso a archivos de entrada y salida con este CALL. El seleccionar estos tipos de archivo causará que un 2 (parámetro de entrada incorrecto) sea POP.
–5	Para datos de temporizador, un elemento es tres palabras de 16 bits, almacenadas en el archivo fuente en el siguiente orden: control, valor preseleccionado y acumulador.

Si se selecciona una cadena interna, el primer carácter (número de transacción) se incrementa con una transacción de escritura efectuada para informar al módulo BASIC que se escribieron datos de cadena al PLC. El valor del número de transacción regresa de 255 a 0.

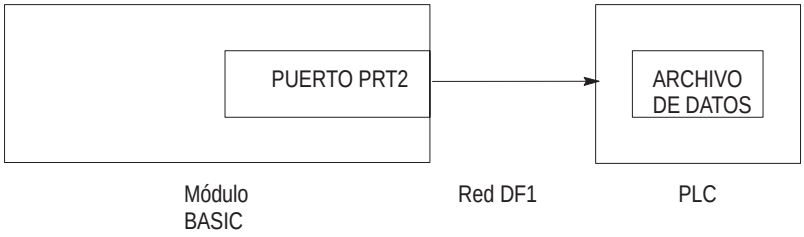
Los parámetros del puerto DF1 se configuran con el CALL 108. El puerto DF1 puede operar con el protocolo full–duplex o half–duplex esclavo.

Ejecute CALL 123 una vez para configurar los parámetros de transferencia de datos. Los bits de imagen de entrada y salida (palabra 0, bit 11) para la ranura que contiene el módulo BASIC, se usan para iniciar y notificar que se completó la transferencia. La operación se describe a continuación:

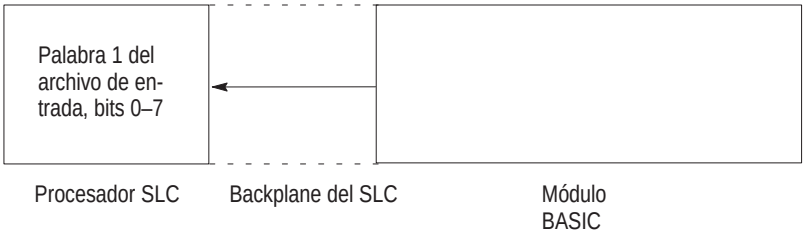
1. El procesador SLC construye el búfer de datos y establece la palabra 0 del archivo de salida, bit 11, para informar al módulo BASIC que hay datos válidos disponibles.



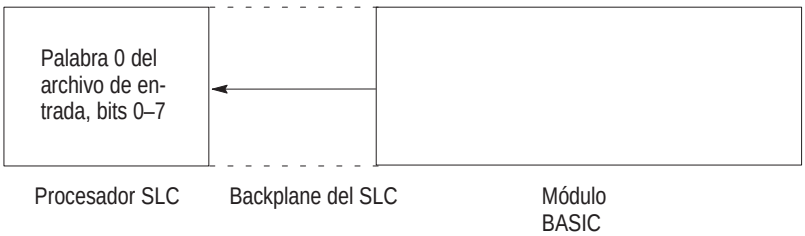
2. El módulo BASIC transfiere los datos al archivo de datos PLC de destino.



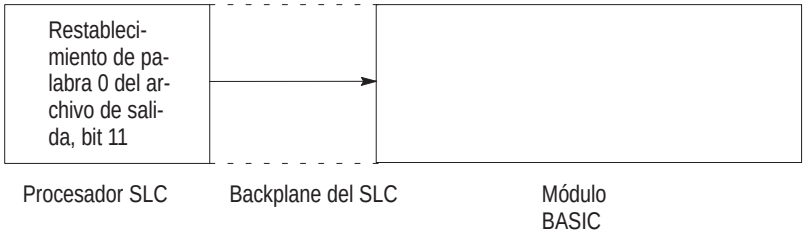
3. El módulo BASIC coloca el estado de la transacción en la palabra 1 del archivo de entrada, bits 0–7.



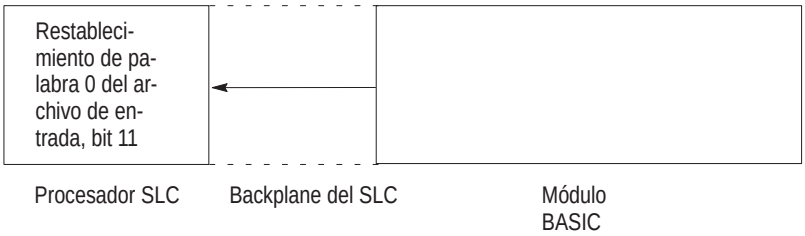
4. El módulo BASIC establece la palabra 0 del archivo de entrada, bit 11, para informar el procesador SLC que los datos fueron transmitidos, y que el estado de la transferencia es válido.



5. El SLC recupera el estado y restablece la palabra 0 del archivo de salida, bit 11. El búfer de datos puede ser reutilizado por el procesador SLC.



6. El módulo BASIC restablece la palabra 0 del archivo de entrada, bit 11, en el mismo fin de ciclo de escán en el que se restableció la palabra 0 del archivo de salida, bit 11.



Este CALL está activo hasta que se vuelve a ejecutar con parámetros de entrada diferentes.

Este CALL tiene diez argumentos de entrada y un argumento de salida.

El primer argumento de entrada es el tipo de comando PLC WRITE emitido:

- 0 = Inhabilita el CALL ejecutado previamente
- 2 = Archivo de interface común – Comando WRITE no protegido PLC–2
- 3 = Archivo PLC–3 – Comando WRITE de rango de palabra
- 5 = Archivo PLC–5 – Comando de escritura WRITE

El segundo argumento de entrada es la dirección de nodo decimal del dispositivo PLC remoto (0 a 254). Si el número no está dentro de este rango, el estado es igual a 2 y el mensaje de escritura no se realiza.

El tercer argumento de entrada es el número de archivo a ser escrito en el dispositivo remoto PLC (0 a 255). Si el número no está dentro de este rango, el estado es igual a 2 y el mensaje de escritura no se realiza. El parámetro es PUSH, pero ignorado, si el archivo de interface común se selecciona en el primer parámetro.

El cuarto argumento de entrada es el tipo de archivo de destino en el dispositivo remoto. Este número se ignora si el archivo de interface común se selecciona en el primer parámetro (supone archivo de enteros). Si el código de tipo de archivo no es uno de los mostrados a continuación, el estado es igual a 2 y el mensaje de escritura no se realiza

Tabla 12.I
Tipo de archivo escrito al dispositivo remoto

Tipo de archivo	Código de tipo de archivo	Palabras/elemento (1 palabra = 16 bits)
Archivo de enteros	ASC(N)	1 palabra/elemento
Archivo de estado	ASC(S)	1 palabra/elemento
Archivo de contador	ASC(C)	3 palabras/elemento
Archivo de temporizador	ASC(T)	3 palabras/elemento
Archivo de bit	ASC(B)	1 palabra/elemento
Archivo de control	ASC(R)	3 palabras/elemento
Archivo de entrada	ASC(I)	1 palabra/elemento
Archivo de salida	ASC(O)	1 palabra/elemento

El quinto argumento es el offset de la palabra inicial dentro del archivo en el dispositivo remoto PLC–2 (0 a 32766). para archivos de enteros, binarios o de estado PLC–3, el valor es 0–9999 (decimal). Para archivos de E/S PLC–3, el valor es 0–4095 (decimal). Para archivos de temporizador o contador PLC–3, el valor es 0. Si el número no está dentro de este rango, el estado es igual a 2 y la transferencia no se realiza.

El sexto argumento de entrada es el número de elementos que se va a transferir. Si el número no está dentro del rango que se muestra a continuación, el estado es igual a 2 y la transferencia no se realiza.

Tabla 12.J
Rango de longitud de elemento válido

Código de tipo de archivo	Rango de longitud de elemento válido
ASC(N)	1 a 64
ASC(S)	1 a 64
ASC(C)	1 a 21
ASC(T)	1 a 21
ASC(B)	1 a 64
ASC(R)	1 a 21
ASC(I)	1 a 64

Código de tipo de archivo	Rango de longitud de elemento válido
ASC(O)	1 a 64
Archivo de interface común	1 a 64

El séptimo argumento es el valor de tiempo límite del mensaje. Este valor (1 a 255) corresponde al número de centenas de milisegundos permitidos para recibir la respuesta de escritura (0.1 a 25.5 segundos). Si la respuesta de escritura no se recibe dentro de este tiempo, el mensaje se cancela con el estado igual a 55 en la palabra 1 del archivo de entrada, bits 0–7. Si el valor de tiempo límite no está dentro del rango (1 a 255), el estado POP es igual a 2 y la transferencia no se realiza.

El octavo argumento de entrada es la selección del archivo de imagen de salida CPU fuente, el archivo M0 CPU o la cadena interna:

- 0 = Archivo de imagen de salida CPU
- 1 = Archivo M0 CPU
- 2 = Cadena interna

Si selecciona la cadena interna (2), el CALL 29 puede ejecutarse para iniciar cada transferencia de datos sin requerir interacción del procesador SLC. La palabra 0 del archivo de salida, bit 11, también iniciará una transacción de cadena.

El noveno argumento de entrada es el offset de palabra dentro del archivo CPU. Este offset apunta a la primera palabra de los datos. Si selecciona el archivo de imagen de salida CPU (0), el offset no puede ser 0 porque esta palabra está reservada para los bits de handshaking de transferencia de datos. El offset para la cadena interna siempre es uno. El primer carácter de la cadena (número de transacción en la ubicación 0) se incrementa con cada transferencia efectuada para informar al módulo BASIC que los datos fueron transferidos. El valor de la transacción regresa de 255 a 0.

El décimo argumento de entrada es el número de cadena. Si el octavo argumento de entrada no selecciona el uso de cadena interna, el valor de este argumento de entrada se ignora pero aún debe ser PUSH.

El argumetno de salida es la validación del CALL. Tiene los siguientes valores:

- 0 = Efectuado correctamente
- 1 = Inhabilitado
- 2 = Parámetro de entrada incorrecto
- 3 = DF1 no habilitado
- 4 = Cadena demasiado pequeña
- 5 = La cadena no está dimensionada

Para inhabilitar este CALL, un cero debe ser PUSH en el primer parámetro de entrada. Todos los otros parámetros se ignoran pero deben ser PUSH.

Cada vez que se intenta escribir a un paquete remoto, el módulo BASIC coloca el estado de la escritura en la palabra 0 de entrada, bits 0–7. Los códigos de estado posibles se muestran a continuación. Este estado es válido cuando el módulo BASIC establece la palabra 0 del archivo de entrada, bit 11.

Tabla 12.K
Códigos de estado de transacción

Código	Indica
0	Transferencia OK.
1	Transmsión fallida.
2	Tiempo límite de consulta.
3	Con handshaking seleccionado – ocurrió una pérdida de señal CTS durante la transmisión o un fallo irremediable de transmisión.
	Sin handshaking seleccionado – ocurrió un fallo irremediable de transmisión..
4	Fallo de transmisión debido a desconexión de modem (pérdida de señal DCD durante más de10 segundos) si se seleccionó handshaking de modem con portadora constante.
5	El controlador DF1 no está habilitado.
6	Tiempo límite de mensaje.
81	Comando o formato ilegal.
82	El dispositivo principal tiene un problema y no se comunica.
83	El dispositivo principal de estación remota no está presente, está desconectado o desactivado.
84	El dispositivo principal no pudo completar la función debido a fallo de hardware.
85	Problema de direccionamiento o renglones de protección de memoria.
86	Función no permitida debido a selección de protección de comando.

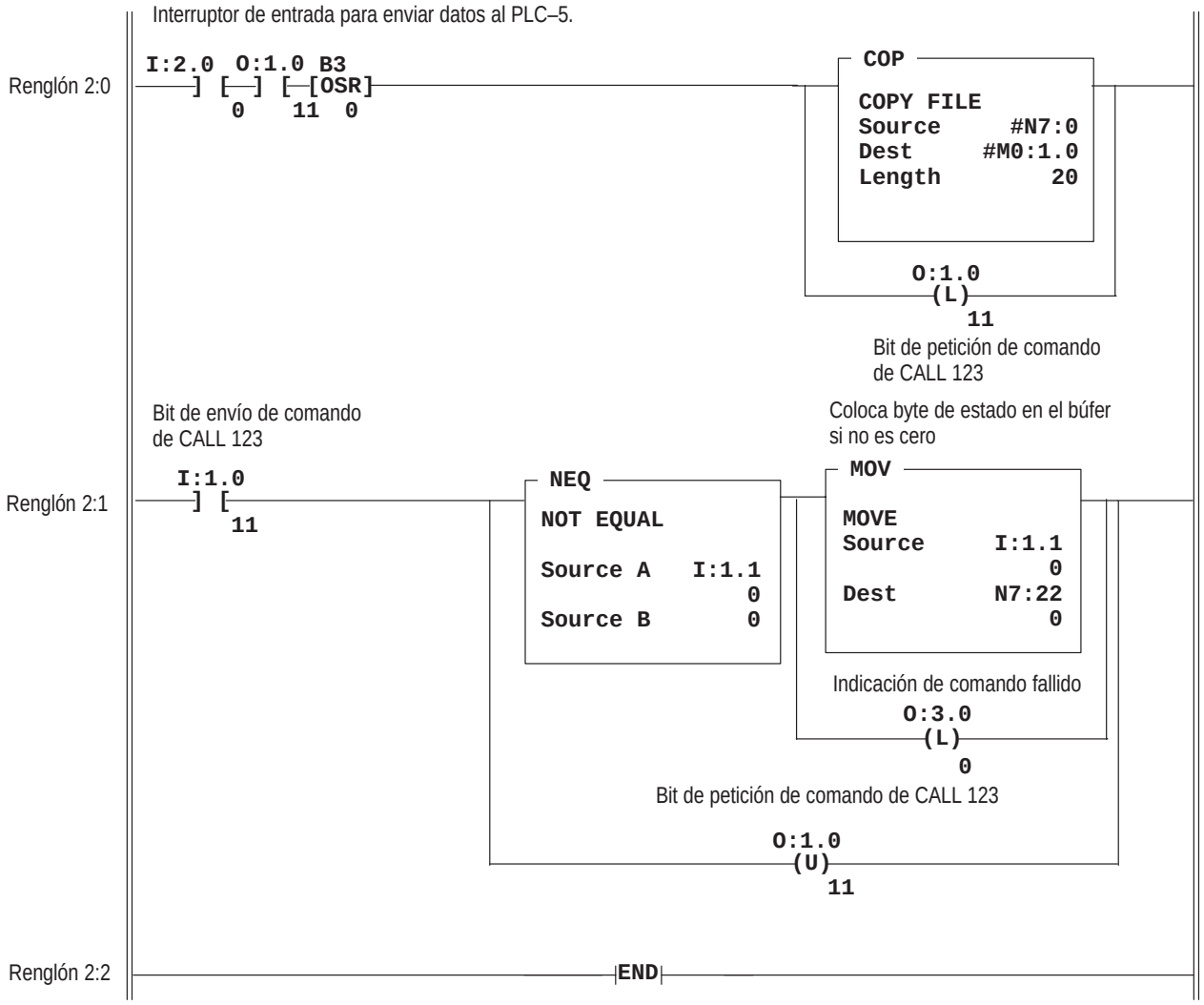
Código	Indica
87	El procesador está en el modo de Programación.
88	Archivo de modo de compatibilidad ausente o problema de zona de comunicación.
89	La estación remota no puede colocar comando en el búfer.
8B	Problema de estación remota debido a descarga.
8C	La estación local no puede ejecutar comando debido a IPB activos..
C1	Formato de dirección ilegal – el campo tiene un valor ilegal.
C2	Formato de dirección ilegal – no hay suficientes campos especificados.
C3	Formato de dirección ilegal – hay demasiados campos especificados.
C4	Formato de dirección ilegal – no se encontró símbolo.
C5	Formato de dirección ilegal – el símbolo es 0 ó mayor que el número máximo de caracteres aceptados por este dispositivo.
C6	Dirección ilegal – la dirección no existe o no apunta a algo utilizable en este comando.
C7	Tamaño ilegal – archivo de tamaño incorrecto; la dirección está más allá del final del archivo.
C8	No puede completar petición.
C9	Datos o archivo muy grande.
CA	La petición es muy larga; el tamaño de la transacción más la dirección de palabra es muy grande.
CB	Acceso negado, violación de privilegio.
CC	Recurso no disponible; la condición no puede generarse.
CD	El recurso ya está disponible; la condición ya existe.
CE	El comando no puede ejecutarse.
CF	Overflow; overflow de histograma.
D0	No hay acceso.
D1	Información de tipo ilegal de datos.
D2	Parámetro no válido datos no válidos en búsqueda o bloque de comando.
D3	Existe referencia de dirección a área borrada.
D4	Fallo de ejecución de comando por razón desconocida; overflow de histograma PLC–3.
D5	Error de conversión de datos.
D6	El escáner no se puede comunicar con un adaptador de chasis 1771.
D7	El adaptador no se puede comunicar con el módulo.
D8	La respuesta del módulo 1771 no fue válida.
D9	Etiqueta duplicada.
DA	Archivo abierto – otra estación es propietaria del mismo.
DB	Otra estación es propietaria del programa.

Sintaxis:

PUSH [tipo de comando WRITE PLC]
PUSH [dirección de nodo PLC remoto]
PUSH [número de archivo de PLC remoto]
PUSH [tipo de archivo en PLC remoto]
PUSH [offset de palabra inicial en PLC remoto]
PUSH [número de elementos que se van a transferir]
PUSH [valor de tiempo límite de mensaje]
PUSH [selección de archivo fuente]
PUSH [offset de palabra dentro de archivo fuente]
PUSH [número de cadena]
CALL 123
POP [estado de CALL 123]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE DF1 PLC REMOTE WRITE COMMAND
>20  PUSH 5 : REM PLC–5 FILE
>30  PUSH 0 : REM PLC–5 NODE ADDRESS
>40  PUSH 7 : REM PLC–5 FILE NUMBER
>50  PUSH ASC(N) : REM PLC–5 FILE TYPE
>60  PUSH 0 : REM STARTING WORD OFFSET FOR PLC–5
>70  PUSH 20 : REM NUMBER OF WORDS TO TRANSFER
>80  PUSH 10 : REM COMMAND TIME–OUT VALUE (X100MS)
>90  PUSH 1 : REM USE M0 FILE
>100 PUSH 0 : REM OFFSET WITHIN M0 FILE
>110 PUSH 0 : REM STRING NUMBER – NOT APPLICABLE FOR THIS
     EXAMPLE
>120 CALL 123
>130 POP S : REM STATUS OF THE CALL
>140 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 123 SETUP”
```



PRINT

Objetivo:

Use la instrucción PRINT para hacer que el módulo BASIC produzca un valor en el dispositivo de la consola. Puede imprimir el valor de expresiones, cadenas, valores literales, variables o cadenas de texto. Puede combinar las diversas formas en la lista de impresión separándolas con comas. Si la lista se termina con una coma, se suprime el retorno de carro/avance de línea. P. es la abreviatura de PRINT.

Los valores se imprimen uno junto a otro con dos espacios en blanco. Una instrucción PRINT sin argumentos envía una secuencia de retorno de carro/avance de línea al dispositivo de la consola.

Importante: El intérprete BASIC termina la impresión de una cadena si encuentra un carácter NULL (0), o CR (13). Si desea imprimir cadenas con estos valores, imprima los caracteres individualmente dentro de un lazo. Para suprimir CR LF en la instrucción PRINT, use una coma a la derecha. Ejemplo: imprima A,

Sintaxis:

PRINT

Ejemplo:

```
>PRINT 10*10,3*3
 100  9

>PRINT ”1746-BAS”
1746-BAS

>PRINT 5,1E3
 5  1000
```

Importante: Debe asegurarse que haya espacio de búfer disponible cada vez que esté imprimiendo datos del puerto en serie usando handshaking de hardware o handshaking de software (Xon/Xoff). El incumplimiento de esta indicación hace que el programa BASIC detenga la ejecución mientras espera espacio de búfer. Cuando hay espacio disponible en el búfer, el módulo BASIC continúa la ejecución desde el punto en donde la interrumpió. El búfer de salidas de cada puerto puede contener 256 caracteres. Para obtener más información vea las descripciones de los CALL 36, 37, 95 y 96.

los símbolos @ y # pueden usarse para dirigir la salida de la impresión a los puertos PRT1 y PRT2 respectivamente.

Use la expresión PRINT CR para producir un retorno de carro sin avance de línea.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “A”, CR,
>20 PRINT “B”

READY
>RUN

B

READY
>
```

A se imprimió y luego B la sobreescribió.

Use la expresión PRINT SPC( ) para producir un número especificado de espacios.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “A”, SPC(10), “B”

READY
>RUN

A          B
```

Use la expresión PRINT TAB( ) ara producir un número especificado de caracteres de tabulación.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT “A”, TAB(1), “B”

READY
>RUN

A    B
```

Use la expresión PRINT USING(Fx) para producir todos los valores numéricos en notación científica. La x representa el número total de dígitos mostrados de la mantisa. Un dígito se muestra antes del punto decimal. El valor de x es un mínimo de tres y un máximo de ocho. El valor mostrado se ajusta según estos límites.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT USING(F4), 123.45678

READY
>RUN

1.234 E+2
```

Use la expresión PRINT USING(#.#) para producir todos los valores numéricos en notación decimal según el formato especificado por la instrucción.

```
>1  REM EXAMPLE PROGRAM
>10 PRINT USING(###.##),
>20 PRINT 4.67890, 123.456
>30 PRINT .0123, .234
>40 PRINT 123.456, 2.1

READY
>RUN

4.67      123.45
0.01      0.23
123.45    2.10

READY
>
```

Use la expresión PRINT USING(0) para restaurar el modo de impresión predeterminado si el modo fue alterado por la expresión PRINT USING(Fx) o por la expresión PRINT USING(#.#).

PH0., PH1.

Objetivo:

Use las instrucciones PH0. y PH1. para que el módulo BASIC envíe un valor hexadecimal al dispositivo de la consola. Estas instrucciones funcionan igual que la instrucción PRINT excepto que los valores se imprimen en formato hexadecimal. La instrucción PH0. suprime dos ceros a la izquierda si el número impreso es menor que 255 (0FFH). La instrucción PH1. siempre imprime cuatro dígitos hexadecimales.

El carácter H siempre se imprime después del número cuando se usa PH0. o PH1. para enviar una salida. Los valores impresos siempre son enteros truncados. Si el número impreso no está dentro del rango de enteros válidos (ejemplo: está entre 0 y 65535 [0FFFFH] inclusive), el módulo BASIC pasa de manera predeterminada al modo normal de impresión. Si esto sucede, no se imprime ningún H después del valor. Puesto que los enteros se introducen en formato decimal o en hexadecimal, las instrucciones PRINT, PH0. y PH1. pueden usarse para realizar conversiones decimal a hexadecimal y hexadecimal a decimal. Todos los comentarios que se aplican a la instrucción PRINT se aplican a las instrucicones PH0. y PH1.

Importante: Debe asegurarse que haya espacio de búfer disponible cada vez que esté imprimiendo datos del puerto en serie usando handshaking de hardware o handshaking de software (Xon/Xoff). El incumplimiento de esta indicación hace que el programa BASIC detenga la ejecución mientras espera espacio de búfer. Cuando hay espacio disponible en el búfer, el módulo BASIC continúa la ejecución desde el punto en donde la interrumpió. El búfer de salidas de cada puerto puede contener 256 caracteres. para obtener más información vea las descripciones de los CALL 36, 37, 95 y 96.

Sintaxis:

PH0., PH1.

Ejemplo:

```
>PH0.2*2
 04H

>PH1.2*2
 0004H

>PH0. 100
 64H

>PH0. 1000
 3E8H

>PH1. 1000
 03E8H

>PH1. 3E8
 3.0 E+8

>PH0. PI
 03H

>
```

ST@

Objetivo:

Use la instrucción ST@ para almacenar números en punto (coma) flotante del módulo BASIC a una dirección específica. La expresión [expr] después de la instrucción ST@ especifica la dirección donde se almacena el número en la RAM. La instrucción ST@ toma el valor en la parte superior de la pila de argumentos y lo almacena en la RAM en la ubicación de la dirección especificada por [expr].

Esta instrucción puede usarse con CALL 77 para almacenar variables en un área protegida de la memoria. Esta área protegida no se pone a cero en el encendido ni cuando se emite el comando RUN.

Sintaxis:

ST@ [expr]

Ejemplo:

```
>P. MTOP
 24515

P. MTOP–10*6
 24455

>PUSH 24455 : CALL 77

>P. MTOP
 24455
>1   REM EXAMPLE PROGRAM
>5   DIM A(10),B(10)
>10  REM *** ARRAY SAVE ***
>20  FOR I = 0 TO 9
>30  A(I) = I+20
>40  PUSH A(I) : REM PUT NUMBER ON STACK
>50  ST@ 5FFFH-I*6
>60  NEXT I
>70  REM *** GET ARRAY ***
>80  FOR I = 0 TO 9
>90  LD@ 5FFFH-I*6
>100 POP B(I) : REM GET NUMBER FROM STACK
>110 PRINT B(I)
>120 NEXT I

READY
>RUN

 20
 21
 22
 23
 24
 25
 26
 27
 28
 29

READY
>PUSH 5FFFH : CALL 77

>P. MTOP
 24575
```

Capítulo 13

Funciones de entrada

Este capítulo describe e ilustra comandos que permiten que el módulo BASIC lea datos de entrada desde sus puertos externos dentro del programa BASIC o desde la línea de comando. La Tabla 13.A lista los mnemónicos correspondientes.

Tabla 13.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Transferir datos desde el PRT1 o PRT2 al sistema de E/S SLC o archivos M	CALL 22	13-2
Transferir datos desde un archivo de datos DH–485 remoto al procesador SLC	CALL 27	13-9
Manejar todos los errores que no son manejados por la instrucción ONERR.	CALL 29	13-17
Obtener un carácter de entrada numérico del puerto PRT2.	CALL 35	13-19
Transferir el búfer de imagen de salida de la CPU al búfer de entradas del módulo BASIC.	CALL 53	13-21
Transferir el archivo M0 CPU al búfer de entradas del módulo BASIC.	CALL 56	13-22
Transferir el archivo de interface DH–485 al búfer de entradas del módulo BASIC.	CALL 84	13-23
Leer el archivo de datos DH–485 remoto al búfer de entradas del módulo BASIC.	CALL 90	13-24
Leer el archivo de interface DH–485 remoto al búfer de entradas del módulo BASIC.	CALL 92	13-27
Obtener la longitud del paquete DF1.	CALL 117	13-29
Permitir escrituras no solicitadas desde un nodo SLC o PLC remoto.	CALL 118	13-30
Leer un arhcivo de datos PLC.	CALL 122	13-36
Leer el dispositivo de entrada de la consola.	GET	13-46
Leer el dispositivo de entrada de la consola conectado al PRT2.	GET#	13-46
Leer el dispositivo de entrada de la consola conectado al PRT1.	GET@	13-46
Leer una línea de caracteres desde el búfer del puerto de programación.	INPL	13-47

Si necesita	Use este mnemónico	Página
Leer una línea de caracteres desde el búfer del puerto PRT2.	INPL#	13-47
Leer una línea de caracteres desde el búfer del puerto PRT1.	INPL@	13-47
Leer una cadena de caracteres desde el búfer del puerto de programación.	INPS	13-47
Leer una cadena de caracteres desde el búfer del puerto PRT2.	INPS#	13-47
Leer una cadena de caracteres desde el búfer del puerto PRT1.	INPS@	13-47
Introducir una cadena o variable.	INPUT	13-48
Introducir una cadena o variable desde el puerto PRT2.	INPUT#	13-48
Introducir una cadena o variable desde el puerto PRT1.	INPUT@	13-48
Cargar una variable.	LD@	13-50
LEER datos en la instrucción de datos.	READ	13-52

CALL 22 - Transferencia de datos desde el puerto 1 ó 2 a los archivos CPU

Objetivo:

Use CALL 22 para transferir datos desde los puertos en serie del módulo BASIC directamente al archivo de datos de entrada CPU, al archivo M1 CPU y/o a una cadena interna dentro del módulo BASIC. Durnte la transferencia de datos, los datos se transfieren automáticamente en bloques de 8 bits desde el búfer de entrada del puerto seleccionado al búfer del procesador SLC seleccionado y/o a una cadena interna BASIC para almacenamiento. La transferencia se realiza cuando se detecta el número especificado de caracteres en el búfer de entrada del puerto o cuando el delimitador definido por el usuario es recibido en el puerto. Los datos se almacenan en byte inferior primero luego byte superior, o byte superior primero luego byte inferior dentro de la palabra de 16 bits de destino. Los datos se transfieren con límites de palabra. Si se va a transferir un número impar de bytes, el byte no usado contiene un cero.

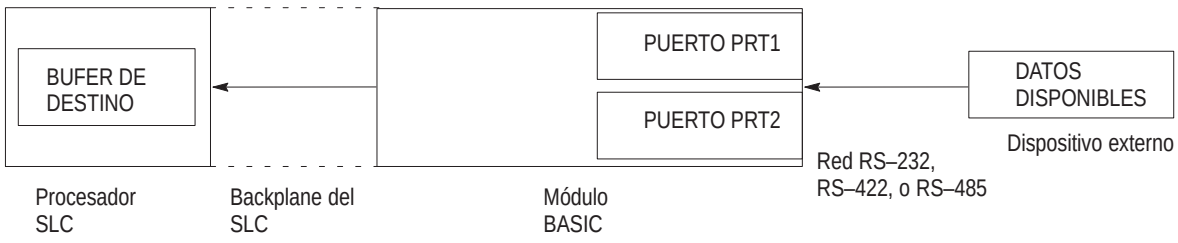
La selección de intercambio de bytes (byte inferior primero, luego byte superior, o byte superior primero, luego byte inferior) del último CALL 22 o CALL 23 ejecutado, determina el método de empaquetado de datos para todos los puertos habilitados por el CALL 22.

El byte inferior de la primera palabra del archivo de destino contiene el conteo de caracteres (conteo de bytes) de los datos que se están transfiriendo. Si se encuentra un delimitador, el conteo de bytes se expande para incluir la primera ocurrencia del delimitador. La segunda palabra del archivo de destino contiene los dos primeros caracteres de datos.

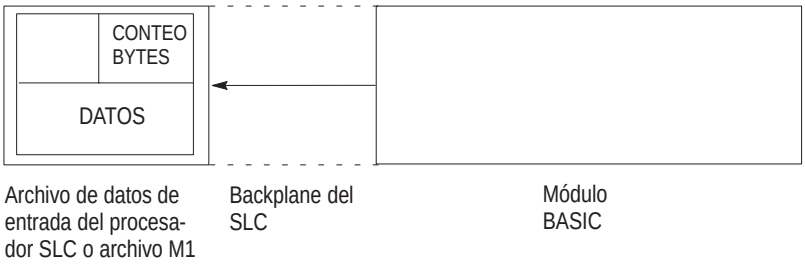
Si se selecciona una cadena interna, el primer carácter de la cadena contiene el conteo de bytes. El segundo carácter de la cadena interna es un número de transacción, el cual se incrementa para informar al módulo BASIC que hay datos nuevos en la cadena. El valor de este carácter vuelve de 255 a 0. El tercer carácter de la cadena contiene el primer carácter de los datos.

Ejecute el CALL 22 para configurar los parámetros de la transferencia de datos. Después que se ejecuta el CALL, el módulo BASIC obtiene los datos del puerto y los transfiere al archivo de destino. Los bits del archivo de imagen de entrada y salida (palabra 0, bits 8 y 9) para la ranura que contiene el módulo BASIC, se usan para iniciar y notificar que se completó la transferencia. La operación se describe a continuación:

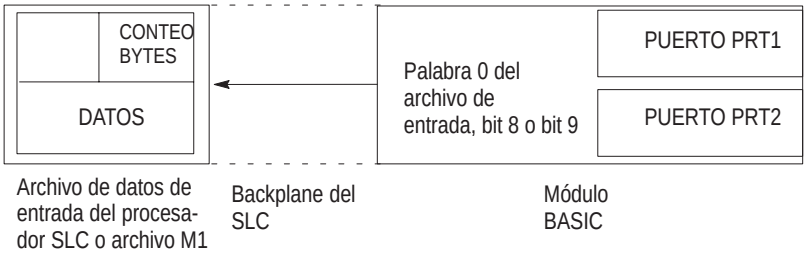
1. Cuando hay datos disponibles del puerto, el módulo BASIC automáticamente transfiere los datos al búfer de destino. Este mismo puerto se verifica para deterinar si hay datos al final de cada línea del código BASIC.



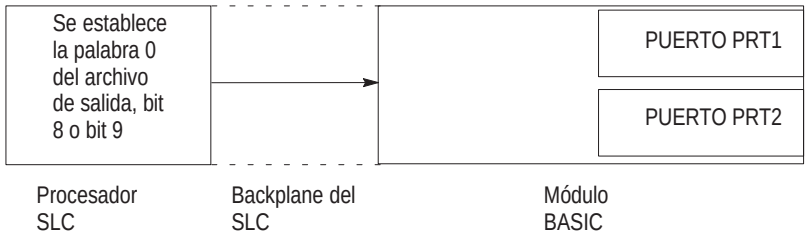
2. El módulo BASIC coloca el conteo de bytes de los datos válidos en el byte inferior de la primera palabra disponible del búfer de destino. El byte superior de la primera palabra disponible no se usa.



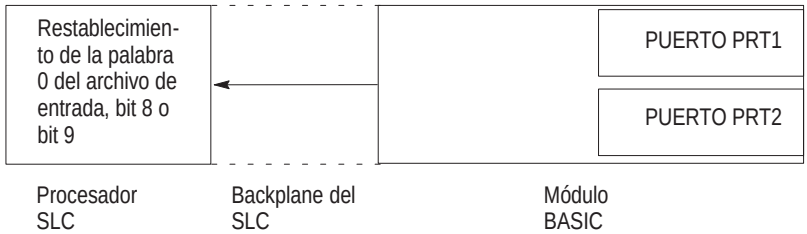
3. El módulo BASIC establece la palabra 0 del archivo de entrada, bit 8 o bit 9 para informar al procesador SLC que hay datos válidos disponibles. El bit 8 indica que hay datos disponibles del puerto 1 y el bit 9 indica que hay datos disponibles del puerto 2.



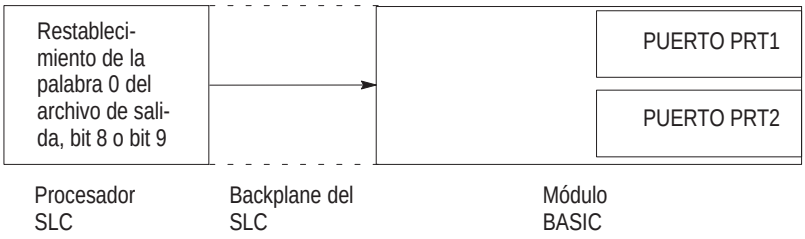
4. El programa de lógica de escalera del procesador SLC obtiene los datos del archivo. El programa de lógica de escalera establece la palabra 0 del archivo de salida, bit 8 o bit 9 para informar al módulo BASIC que se recibieron los datos.



5. el módulo BASIC restablece la palabra 0 del arhcivo de entrada, bit 8 o bit 9 en el mismo ciclo de fin de escán en el que fue establecida la palabra 0 del archivo de salida, bit 8 o bit 9.



6. El programa lógico de escalera del SLC restablece la palabra 0 del archivo de salida, bit 8 o bit 9. El módulo BASIC puede empezar la carga del búfer de destino con el siguiente paquete a medida que llegan los datos desde el puerto.



Las transferencias de datos continúan hasta que se vuelve a ejecutar el CALL para el puerto con parámetros de entrada diferentes. Si ocurre esto, el CALL 22 previo para el puerto se inhabilita automáticamente y el CALL 22 nuevo se hace efectivo. Múltiples CALL 22 para el mismo puerto no se ejecutan en paralelo. Sin embargo, el puerto 1 y el puerto 2 pueden activarse simultáneamente emitiendo CALL 22 separados.

Este CALL tiene siete argumentos de entrada y un argumento de salida.

El primer argumento de entrada es el número de puerto fuente (1 ó 2) del módulo BASIC. Un cero inhabilita todos los comandos de CALL 22 activos previamente.

- 0 = Inhabilita CALL 22 para todos los puertos activos habilitados por CALL 22 anteriores.
- 1 = El puerto 1 es la fuente.
- 2 = El puerto 2 es la fuente.

El segundo argumento de entrada es el número máximo de caracteres de 8 bits que serán copiados desde el puerto en serie BASIC al archivo de destino. El número máximo de caracteres es seleccionado por el cuarto argumento de entrada:

- Archivo de imagen de entrada CPU: máximo de caracteres = 10 (5 palabras)
- Archivo M1 CPU: máximo de caracteres = 126 (63 palabras)
- Cadena interna: máximo de caracteres = tamaño de cadena–3 (El primer carácter es el valor de conteo de bytes; el segundo carácter es el número de transacción incrementado; y el último carácter es el carácter de terminación. El número máximo de caracteres para una cadena interna es 254.)

Si se obtiene menos del máximo cuando se recibe un carácter delimitador, el paquete, incluyendo el delimitador, se envía al archivo CPU. El procesador SLC determina la cantidad de datos válidos transferidos al archivo de destino desde el conteo de bytes colocado en el byte inferior de la primera palabra del archivo.

Si los datos recibidos exceden la longitud de cadena del tamaño del archivo CPU, los datos restantes se truncan.

El tercer argumento de entrada es el valor decimal del delimitador de caracteres ASCII. Se puede seleccionar cualquier carácter ASCII. Si no se desea un delimitador, introduzca un valor NULL (0 decimal). Los datos serán transferidos al búfer de destino cuando se reciba el delimitador desde el puerto seleccionado, independientemente el número de caracteres recibidos.

El cuarto argumento de entrada es la selección del archivo de imagen de entrada CPU de destino con o sin la cadena interna, el archivo M1 CPU con o sin la cadena interna, o la cadena interna sola:

- 0 = Archivo de imagen de entrada CPU
- 1 = Archivo M1 CPU
- 2 = Archivo de imagen de entrada CPU y cadena interna
- 3 = Archivo M1 CPU y cadena interna
- 4 = Cadena interna solamente

Cuando se transfieren datos a la cadena interna del módulo BASIC, verifique su número de transacción para actualizaciones de cadenas porque no hay indicación de que se han colocado datos en la cadena interna. Su programa BASIC debe revisar el número de transacción para verificar que los datos fueron actualizados.

El quinto argumento de entrada es el offset de palabra dentro del archivo CPU de destino. Si se selecciona el archivo de datos de entrada CPU, este offset no debe ser 0 ni 1, puesto que las palabras 0 y 1 están reservadas. Cero está reservado para bits de handshaking de transferencia de datos y la palabra 1 está reservada para el estado de la transacción. Un 0 ó un 1 hará que el estado de CALL 2 sea POP. Este offset apunta a la ubicación del búfer del conteo de bytes. Si se selecciona el archivo M1, el offset puede ser cero. Si se selecciona la cadena interna, la ubicación de datos siempre empieza con el tercer carácter de la cadena. (El primer carácter contiene el conteo de bytes y el segundo carácter contiene el número de transacción). Por lo tanto, el valor de offset no tiene efecto en la ubicación de datos de la cadena, sólo en la ubicación de datos en el archivo de imagen de entrada y en el archivo M1.

El sexto argumento de entrada es el número de cadena. Si el cuarto argumento de entrada no selecciona uso de cadena interna, el valor de este argumento de entrada se ignora pero tiene que ser PUSH.

El séptimo argumento de entrada es la selección de intercambio de bytes. Tiene los siguientes valores:

- 0 = Los bytes de datos transferidos desde el puerto BASIC *no* se intercambian cuando pasan al destino. El orden del paquete de datos es byte inferior primero, luego byte superior por palabra. El byte inferior de la primera palabra en el archivo de destino contiene el conteo de bytes.
- 1 = Los bytes de datos transferidos desde el puerto BASIC se intercambian cuando pasan al destino. El orden del paquete de datos es byte superior primero, luego byte inferior por palabra. El intercambio no afecta a la primera palabra. El byte inferior de la primera palabra sigue conteniendo el conteo de bytes.

El último CALL 22 o CALL 23 ejecutado determina la opción de intercambio de bytes para todos los comandos de CALL 22 activos previamente ejecutados.

El argumento de salida es el estado del CALL. Tiene los siguientes valores:

- 0 = Configuración efectuada correctamente
- 1 = Inhabilitado
- 2 = Parámetro de entrada incorrecto
- 3 = Se ha seleccionado PRT2 pero ya está habilitado para protocolo DF1. La transferencia no se ejecuta.
- 4 = La cadena es demasiado pequeña
- 5 = La cadena no está dimensionada

Si se están recibiendo datos en el puerto en serie más rápidamente que la velocidad de recuperación de datos del procesador SLC, el búfer de entrada del puerto se llena. Si usa handshaking entre el puerto y el dispositivo externo, no se perderán datos.

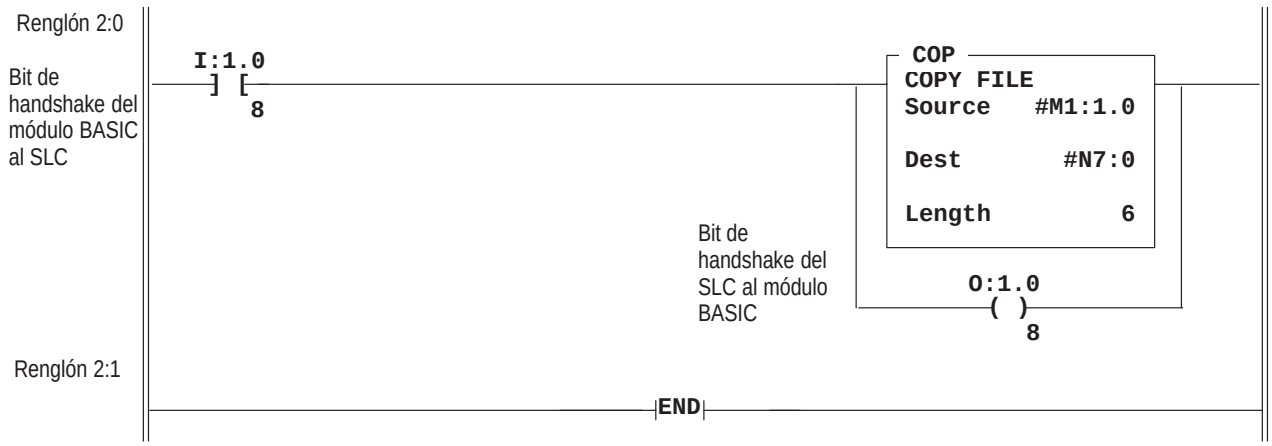
Sintaxis:

PUSH [número de puerto fuente]
PUSH [número máximo de caracteres que se van a transferir]
PUSH [valor decimal del delimitador de caracteres]
PUSH [selección de archivo de destino y/o cadena]
PUSH [offset de palabra dentro del archivo de destino]
PUSH [número de cadena]
PUSH [selección de intercambio de bytes]
CALL 22
POP [estado del CALL 22]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE CALL 22 INTERRUPTS
>20  PUSH 1 : REM PRT1 ACTIVE FOR CALL 22
>30  PUSH 10 : REM RECEIVING 10 BYTES OF DATA MAXIMUM
>40  PUSH 13 : REM <CR> USED AS TERMINATION CHARACTER
     (13 DECIMAL)
>50  PUSH 1 : REM SEND DATA TO M1 FILE
>60  PUSH 0 : REM OFFSET INTO M1 FILE
>70  PUSH 0 : REM STRING NUMBER – NOT USED
>80  PUSH 1 : REM BYTE SWAPPING ENABLED
>90  CALL 22
>100 POP S : REM STATUS OF CALL 22 SETUP
>110 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 22 SETUP”
>120 END
```

El siguiente es un ejemplo de programa de lógica de escalera para el CALL 22. El módulo BASIC está ubicado en la ranura 1 del rack SLC. En el renglón 2:0, copie los datos desde el archivo M1 cuando el bit de handshake (I:1.0/8) sea establecido por el módulo BASIC. El SLC establece el bit de handshake (O:1.0/8) una vez que los datos han sido copiados del archivo M1. Esto indica al módulo BASIC que desactive (I:1.0/8). La primera palabra del archivo M1 contiene el conteo de bytes, y esta palabra no se incluye en el conteo de bytes de datos. En este ejemplo se espera un máximo de 10 bytes de datos.



CALL 27 - Lectura del archivo de datos DH–485 SLC remoto

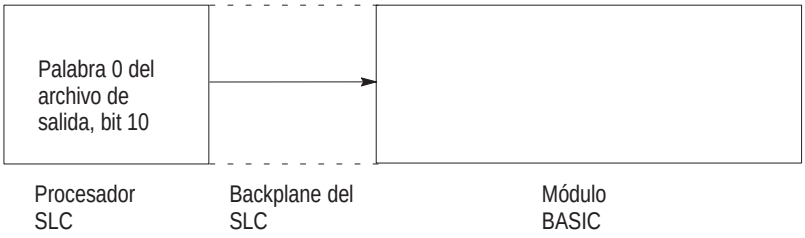
Objetivo:

Use CALL 27 para leer hasta 64 palabras de datos desde un archivo de datos de nodo DH–485 remoto al archivo de imagen de entrada CPU local, archivo M1 CPU y/o una cadena dentro del módulo BASIC.

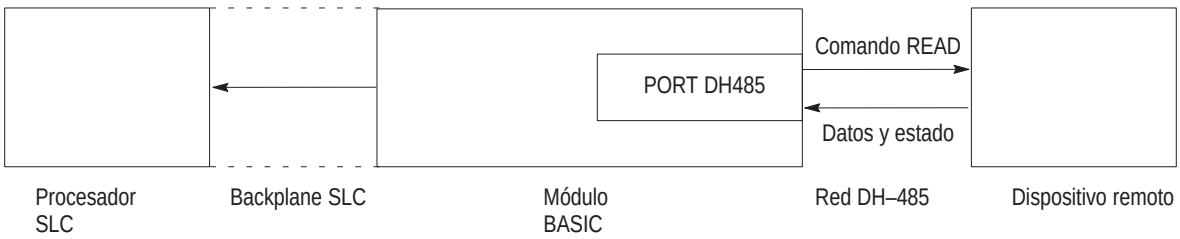
Si se selecciona una cadena interna, el primer carácter se incrementa para informar al módulo BASIC que hay datos nuevos en la cadena. El valor de este carácter vuelve de 255 a 0.

Ejecute CALL 27 una vez para configurar los parámetros de la transferencia de datos. Los bits de imagen de entrada y salida (palabra 0, bit 10) para la ranura que contiene el módulo BASIC, se usan para iniciar y notificar que se ha completado la transferencia. El módulo BASIC envía el comando READ configurado en CALL 27 al dispositivo DH–485 remoto designado en la red. La operación se describe a continuación:

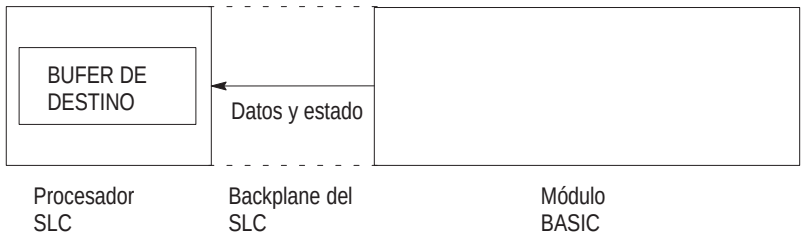
1. El procesador SLC local establece la palabra 0 del archivo de salida, bit 10, para informar al módulo BASIC que el comando READ configurado en CALL 27 debe ejecutarse.



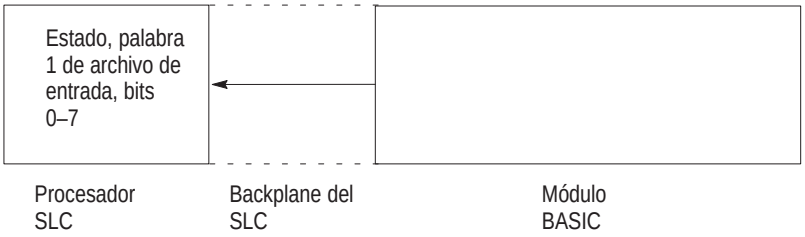
2. El módulo BASIC emite automáticamente el comando READ apropiado al dispositivo remoto en la red DH–485. Los datos y el estado se envían otra vez al procesador SLC local.



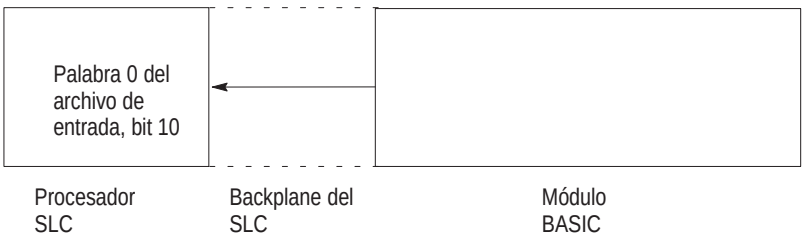
3. Cuando hay datos disponibles, el módulo BASIC transfiere los datos al búfer de destino SLC local.



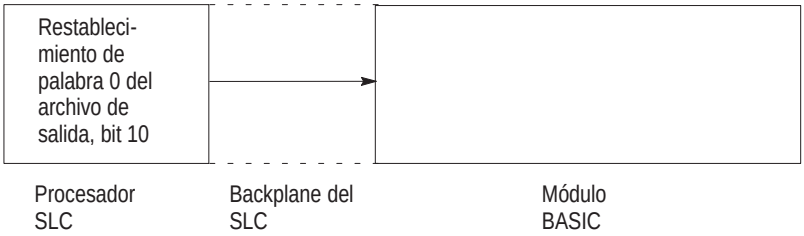
4. El módulo BASIC coloca el estado en la palabra 1 del archivo de entrada, bits 0–7.



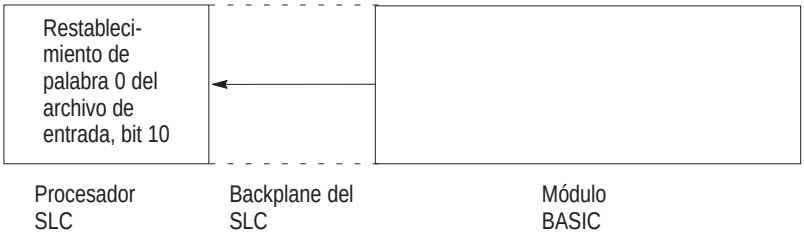
5. El módulo BASIC establece la palabra 0 del archivo de entrada, bit 10, para informar al procesador SLC que hay datos válidos disponibles.



6. El SLC local recupera los datos y el estado desde el búfer y luego restablece la palabra 0 del archivo de salida, bit 10, para informar al módulo BASIC que se recibieron los datos.



7. El módulo BASIC restablece la palabra 0 del archivo de entrada, bit 10 en el mismo ciclo de fin de escán en el que se restableció la palabra 0 del archivo de salida, bit 10.



El procesador SLC no debe establecer, y luego restablecer, la palabra 0 del archivo de salida, bit 10, en el mismo ciclo de escán de lógica de escalera. Si esto ocurre, el módulo BASIC puede perder el bit que se está estableciendo.

Este CALL estará activo hasta que se vuelva a ejecutar con parámetros de entrada diferentes.

Este CALL tiene diez argumentos de entrada y un argumento de salida.

El primer argumento de entrada es el tipo de comando SLC READ emitido:

- 0 = Inhabilita el CALL 27 ejecutado previamente
- 1 = Lectura de archivo de interface común
- 2 = Mensaje de lectura SLC

El segundo argumento de entrada es la dirección de nodo del dispositivo remoto SLC (0 a 31). Si el número no está dentro de este rango, el estado es igual a 2 y el mensaje de lectura no se realiza.

El tercer argumento de entrada es el número de archivo en el dispositivo remoto SLC (0 a 255). Si el número no está dentro de este rango, el estado es igual a 2 y el mensaje de lectura no se realiza. El parámetro se ignora si se selecciona el archivo de interface común (CIF) en el primer parámetro. El CIF siempre es el archivo 9.

El cuarto argumento de entrada es el tipo de archivo que se va a leer desde el dispositivo remoto. Este número se ignora si se selecciona el CIF en el primer parámetro (supone archivo de enteros). Si el tipo de archivo no es uno de los listados a continuación, el estado es igual a 2 y el mensaje de lectura no se realiza. Introduzca el código de tipo de archivo tal como se muestra a continuación cuando PUSH el cuarto parámetro de entrada.

Tabla 13.B
Tipo de archivo que se va a leer desde el dispositivo remoto

Tipo de archivo	Código de tipo de archivo	Palabras/elemento
Archivo de enteros	ASC(N)	1 palabra/elemento
Archivo de contador	ASC(C)	3 palabras/elemento
Archivo de temporizador	ASC(T)	3 palabras/elemento
Archivo de bits	ASC(B)	1 palabra/elemento
Archivo de control	ASC(R)	3 palabras/elemento

El quinto argumento de entrada es el offset de la palabra inicial dentro del archivo en el dispositivo remoto (0 a 32766). Si el número no está dentro de este rango, el estado es igual a 2 y la transferencia no se realiza. (El procesador SLC 500 sólo acepta 0 a 255 palabras por archivo).

El sexto argumento de entrada es el número de palabras que se van a transferir. Si el número no está dentro del rango mostrado, el estado es igual a 2 y la transferencia no se realiza. Los procesadores SLC 5/01 y SLC 5/02 aceptan transferencias de hasta 41 palabras como máximo.

Tabla 13.C
Rango de longitud válido

Código de tipo de archivo	Rango de longitud válido
ASC(N)	1 a 64
ASC(C)	1 a 21
ASC(T)	1 a 21
ASC(B)	1 a 64
ASC(R)	1 a 21
Archivo de interface común	1 a 64

El séptimo argumento de entrada es el valor de tiempo límite del mensaje. Este valor (1 a 255) corresponde al número de centenas de milisegundos permitidos para recibir la respuesta de lectura (0.1 a 25.5 segundos). Si la respuesta de lectura no se recibe dentro de este período de tiempo, el mensaje se cancela con el estado igual a 55 en la palabra 1 del archivo de entrada, bits 0–7. Si el valor de tiempo límite no está dentro del rango (1 a 255), el estado POP es igual a 2 y la transferencia no se realiza.

El octavo argumento de entrada es la selección del archivo de destino y/o cadena:

- 0 = Archivo de imagen de entrada CPU
- 1 = Archivo M1 CPU
- 2 = Cadena interna
- 3 = Archivo de imagen de entrada CPU y cadena interna
- 4 = Archivo M1 CPU y cadena interna

Si selecciona la cadena interna (2), el CALL 29 puede ejecutarse para iniciar cada transferencia de datos sin requerir interacción del procesador SLC. La palabra 0 del archivo de salida, bit 10, también iniciará una transacción de cadena.

El noveno argumento de entrada es el offset de palabra dentro del archivo CPU de destino. Este offset apunta a la primera palabra transferida. El offset para la cadena interna siempre es 1 (el número de transacción en la ubicación 0) seguido por los datos. El número de transacción se incrementa con cada transferencia de datos y vuelve de 255 a 0.

Si se selecciona el archivo de imagen de entrada CPU, este offset no debe ser 0 ni 1. Cero está reservado para bits de handshaking de transferencia de datos y la palabra 1 está reservada para el estado de la transacción. Un 0 ó un 1 causan que un error sea POP (2 = parámetro de entrada incorrecto) y el CALL no se ejecuta.

El décimo argumento de entrada es el número de cadena. Si el octavo argumento de entrada no selecciona uso de cadena interna, el valor de este argumento de entrada se ignora, pero debe ser PUSH.

El argumento de salida es el estado del CALL. Tiene los siguientes valores:

- 0 = Efectuado correctamente
- 1 = Inhabilitado
- 2 = Parámetro de entrada incorrecto
- 3 = Puerto DH485 no habilitado (DF1 habilitado)
- 4 = La cadena es demasiado pequeña
- 5 = La cadena no está dimensionada

Para inhabilitar este CALL un 0 debe ser PUSH en el primer parámetro de entrada. Todos los otros parámetros se ignoran pero deben ser PUSH.

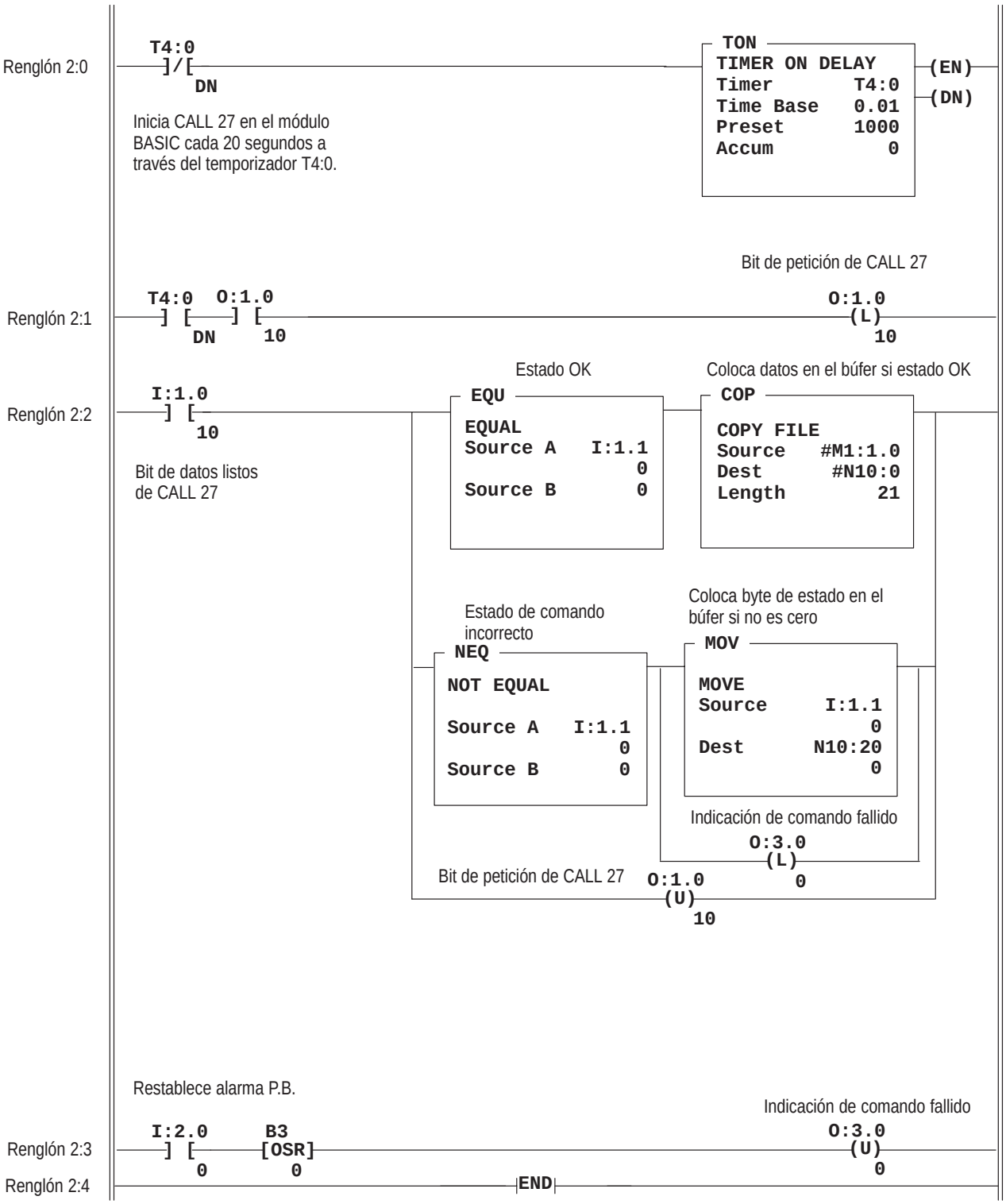
Cada vez que se intente leer un paquete remoto, el estado de la lectura se coloca en la palabra 1 de entrada, bits 0–7. Estos valores tienen la misma definición que los valores POP en CALL 92. El estado se hace válido cuando el módulo BASIC establece la palabra 0 del archivo de entrada, bit 10.

Sintaxis:

PUSH [tipo de comando READ]
PUSH [dirección de nodo remoto]
PUSH [número de archivo remoto]
PUSH [tipo de archivo remoto]
PUSH [offset de palabra inicial de archivo remoto]
PUSH [número de palabras que se va a transferir]
PUSH [valor de tiempo límite de mensaje]
PUSH [selección de archivo de destino]
PUSH [offset de palabra dentro de archivo de destino]
PUSH [número de cadena]
CALL 27
POP [estado de CALL 27]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE REMOTE DH-485 READ COMMAND INTERRUPT
>20  PUSH 2 : REM SLC TYPED READ COMMAND
>30  PUSH 2 : REM NODE ADDRESS OF REMOTE SLC
>40  PUSH 7 : REM FILE NUMBER OF REMOTE SLC
>50  PUSH ASC(N) : REM FILE TYPE OF REMOTE SLC
>60  PUSH 100 : REM REMOTE ELEMENT OFFSET INTO REMOTE SLC
     FILE
>70  PUSH 20 : REM NUMBER OF ELEMENTS TO BE TRANSFERRED
>80  PUSH 5 : REM MESSAGE TIMEOUT (X100MS)
>90  PUSH 1 : REM DESTINATION FILE TO PUT DATA (M1 FILE)
>100 PUSH 0 : REM WORD OFFSET INTO DESTINATION FILE
>110 PUSH 0 : REM STRING NUMBER – NOT AVAILABLE FOR THIS
     EXAMPLE
>120 CALL 27
>130 POP S
>140 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 27 SETUP”
```


Renglón 2:0
T4:0
]/[
DN
TON
TIMER ON DELAY
Timer T4:0
Time Base 0.01
Preset 1000
Accum 0
(EN)
(DN)
Inicia CALL 27 en el módulo BASIC cada 20 segundos a través del temporizador T4:0.
Bit de petición de CALL 27
Renglón 2:1
T4:0 O:1.0
] [ ] [
DN 10
O:1.0
(L)
10
Estado OK
Coloca datos en el búfer si estado OK
Renglón 2:2
I:1.0
] [
10
EQU
EQUAL
Source A I:1.1
0
Source B 0
COP
COPY FILE
Source #M1:1.0
Dest #N10:0
Length 21
Bit de datos listos de CALL 27
Estado de comando incorrecto
Coloca byte de estado en el búfer si no es cero
NEQ
NOT EQUAL
Source A I:1.1
0
Source B 0
MOV
MOVE
Source I:1.1
0
Dest N10:20
0
Indicación de comando fallido
O:3.0
(L)
0
Bit de petición de CALL 27
O:1.0
(U)
10
Restablece alarma P.B.
Indicación de comando fallido
Renglón 2:3
I:2.0
] [
0
B3
[OSR]
0
O:3.0
(U)
0
Renglón 2:4
END

CALL 29 - Lectura/escritura a un PLC/SLC desde la cadena interna del módulo BASIC

Objetivo:

Use CALL 29, junto con CALLs 122 ó 123, para comunicarse entre PLC remotos y la cadena interna del módulo BASIC sin interacción del procesador SLC local. También puede usar CALL 29, junto con CALL 27 ó 28, para establecer comunicación entre los SLC remotos y la cadena interna del módulo BASIC sin interacción del procesador SLC local. Los CALL 27, 28, 122 ó 123 deben ejecutarse dentro del programa BASIC antes que el CALL 29.

El CALL 29 está activo cuando la cadena interna es la única opción en CALL 27, 28, 122 ó 123. En esta situación, no es práctico usar los archivos de imagen de entrada y salida SLC para empezar la transferencia y pasar el estado. El procesador SLC no necesita estar involucrado. Si en su lugar se selecciona un archivo SLC, el procesador SLC local controla la transferencia con los bits de imagen de entrada y salida. En este caso, cuando se intenta ejecutar CALL 29, retorna un estado de 255.

Un argumento se PUSH y un argumento se POP El argumento de entrada es el CALL que se va a activar (CALL 27, 28, 122 ó 123).

Cuando se ejecuta CALL 29, se intenta ejecutar la transferencia. Si no se ejecuta el CALL seleccionado (27, 28, 122 ó 123) antes del CALL 29, un 1 retorna en el estado que fue POP. Cuando se efectúa correctamente el CALL 29, el valor en el primer carácter de la cadena (número de transacción) se incrementa para indicar que se realizó la transferencia. El rango de este carácter es 0–255.

Después que se ejecuta CALL 29, una palabra es POP. Esta palabra es el estado de la transacción:

- 0 = Se efectuó correctamente
- 1 = El CALL seleccionado (27, 28, 122, ó 123) no está activo
- 255 = El búfer SLC está seleccionado para CALL 27, 28, 122 ó 123 y el CALL 29 se ignora
- Todos los otros códigos son idénticos a CALL 90/92

Sintaxis:

PUSH [27, 28, 122 ó 123 para el CALL que usted desea activar]
CALL 29
POP [estado de la transacción]

Ejemplo:

El CALL 122 debe estar habilitado con cadena interna sólo antes de ejecutar el CALL 29 en este ejemplo. Con la ejecución del CALL 29 se intenta transferir un elemento del archivo de enteros 10, empezando en el elemento 0 del PLC–5 en el nodo 3, a la cadena interna $(1) del módulo BASIC.

```
>1   REM EXAMPLE PROGRAM
>10  REM EXECUTE DF1 PLC REMOTE READ FROM INTERNAL
>20  REM STRING WITH NO SLC INTERVENTION
>21  REM SET UP CALL 122
>25  PUSH 5, 3, 10, ASC(N), 0, 10, 10, 1, 1, 1: CALL 122:
POP
     STATUS
>30  PUSH 122
>40  CALL 29
>50  POP S
>60  IF (S=1) THEN PRINT “CALL 122 NOT ACTIVE”
>70  IF (S=255) THEN PRINT “SLC FILE CHOSEN FOR CALL 122”
>80  IF (S=0) THEN PRINT “SUCCESSFUL TRANSFER”
>90  IF (S<>0) THEN PRINT “UNSUCCESSFUL TRANSFER”
>100 END
```

El CALL 29 reemplaza la función del bit de handhsaking en CALL 27, 28, 122 y 123 cuando se usa un archivo SLC o cadena del módulo BASIC.

CALL 35 - Obtención de carácter de entrada numérico desde el PRT2

Objetivo:

Use CALL 35 para obtener el carácter actual en el búfer de entradas de 256 caracteres del puerto PRT2. Retorna la representación decimal de los caracteres recibidos como su argumento de salida. El puerto PRT2 recibe los datos transmitidos por su dispositivo y los almacena en este búfer. Si no hay ningún carácter, el argumento de salida es 0 (nulo). Si hay un carácter, el argumento de salida es el valor ASCII de ese carácter. No hay argumento de entrada para esta rutina.

Sintaxis:

CALL 35
POP [valor ASCII del carácter]

Ejemplos:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 35
>20 POP X
>30 IF X=0 THEN GOTO 10
>40 PRINT CHR(X)

READY
>RUN

A
READY
>
```

El ejemplo anterior supone que el búfer de entrada PRT2 contiene el carácter A ASCII.

Importante: Un 0 (nulo) es un carácter válido en algunos protocolos de comunicación. Use el CALL 36 para determinar el número de caracteres en el búfer.

Importante: Purgue el búfer antes de almacenar datos para asegurar la validez de los datos.

```
>1   REM EXAMPLE PROGRAM
>10  REM PERIPHERAL PORT INPUT USING CALL 35
>20  STRING 200,20
>30  DIM D(254)
>40  CALL 35 : POP X
>50  IF X <>2 GOTO 40
>55  REM WAIT FOR DEVICE TO SEND START OF TEXT
>60  REM
>70  DO
>80  I=I+1
>90  CALL 35 : POP D(I): REM STORE DATA IN ARRAY
>100 UNTIL D(I)=3 : REM WAIT FOR DEVICE TO SEND END OF
     TEXT
>120 REM
>130 REM FORMAT AND PRINT DATA TYPES
>140 PRINT “RAW DATA=”
>150 FOR J=1 TO I : PRINT D(J),: NEXT J
>155 REM PRINT RAW DECIMAL DATA
>160 PRINT: PRINT: PRINT
>170 PRINT “ASCII DATA=”
>180 FOR J=1 TO I : PRINT CHR(D(J)),:NEXT J
>185 REM PRINT ASCII DATA
>190 PRINT: PRINT: PRINT
>200 PRINT “$(1)=”
>210 FOR J=1 TO I: ASC($(1),J)=D(J): NEXT J
>215 REM STORE DATA IN STRING
>220 PRINT $(1)
>230 PRINT: PRINT: PRINT
>240 I=0
>250 REM
>260 GOTO 40

READY
>RUN


RAW DATA=
 65 66 67 68 69 70 71 49 50 51 52 53 54 55 56 57 3


ASCII DATA=
 ABCDEFG123456789


$(1)=
 ABCDEFG123456789
```

CALL 53 - Transferencia de la imagen de salida CPU al búfer de entradas BASIC

Objetivo:

Use CALL 53 para transferir las palabras 0 a 7 de la tabla de imagen de salida CPU a las palabras 200 a 207 del búfer de entradas del módulo BASIC. Esta rutina no tiene argumentos de entrada y tiene un argumento de salida. El argumento de salida es el estado del procesador lógico. Puede tener uno de los siguientes valores:

- 0 = El procesador lógico está en el modo de marcha (Run)
- 1 = El procesador lógico no está en el modo de marcha

La integridad de la palabra se garantiza durante esta transferencia. La integridad del archivo no se garantiza. Para proporcionar integridad del archivo, pueden usarse los bits de handshaking en su programa de aplicación.

Todos los datos transferidos al módulo BASIC desde el procesador SLC 500 pueden ser dirigidos a través del búfer de entradas del módulo BASIC. La Tabla 13.D lista la definición de las direcciones en el búfer de entrada del módulo BASIC.

Tabla 13.D
Direcciones del búfer de entrada del módulo BASIC

Dirección	Definición
0 a 39	Datos transferidos desde el archivo de interface común DH-485.
40 a 99	Reservado
100 a 163	Datos transferidos desde el archivo M0 de la CPU del módulo SLC 500.
164 a 199	Reservado
200 a 207	Datos transferidos desde la tabla de imagen de salida de la CPU del SLC 500.

Sintaxis:

CALL 53
POP [estado del procesador]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>30 CALL 53 : REM XFER CPU OUTPUT IMAGE TO BASIC INPUT
    BUFFER
>40 POP X : REM LOGIC PROCESSOR STATUS
>50 IF (X<>0) THEN PRINT “PROCESSOR NOT IN RUN MODE”

READY
>RUN
```

CALL 56 - Transferencia del archivo M0 CPU al búfer de entradas BASIC

Objetivo:

Use CALL 56 para transferir hasta 64 palabras empezando en la palabra 0 del archivo M0 CPU al búfer de entradas del módulo BASIC empezando en la palabra 100. Esta rutina tiene un argumento de entrada y un argumento de salida. El argumento de entrada es el número de palabras que se van a transferir (0 a 64). Si el número no está dentro del rango de (0 a 64), no se produce la transferencia y el argumento de salida se establece en 10. Si el argumento de entrada es un número válido, el argumento de salida es el estado del procesador lógico. Puede tener uno de los siguientes valores:

- 0 = Transferencia efectuada correctamente, procesador lógico en el modo de marcha (Run)
- 1 = Transferencia efectuada correctamente, procesador lógico en el modo de programación (Program)
- 2 = Transferencia efectuada correctamente, procesador lógico en el modo de prueba (Test)
- 10 = Longitud ilegal especificada
- 11 = El procesador lógico no acepta esta capacidad

La integridad de la palabra se garantiza durante esta transferencia. La integridad del archivo no se garantiza. Para proporcionar integridad del archivo, pueden usarse los bits de handshaking en su programa de aplicación.

Sintaxis:

PUSH [número de palabras que se van a transferir]
CALL 56
POP [estado del procesador]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>30 PUSH 64 :  REM TRANSFER 64 WORDS
>40 CALL 56 :  REM TRANSFER M0 TO BASIC INPUT BUFFER
>50 POP X :  REM LOGIC PROCESSOR STATUS IS IN X
>60 IF (X=10) PRINT “ILLEGAL INPUT ARGUMENT”
>70 IF (X<>0).AND.(X<>10) THEN  PRINT “PROCESSOR NOT IN
    RUN MODE”

READY
>RUN
```

CALL 84 - Transferencia del archivo de interface DH-485 al búfer de entradas BASIC

Objetivo:

Use CALL 84 para transferir hasta 40 palabras empezando en el offset designado del archivo de interface común DH-485 al búfer de entradas del módulo BASIC empezando en el mismo offset designado de la palabra 0. Esta rutina tiene dos argumentos de entrada y un argumento de salida. El primer argumento de entrada es el offset inicial en el archivo de interface común DH-485 y el búfer de entradas del módulo BASIC (0 a 39). Si el número no está dentro del rango de 0 a 39, el argumento de salida es igual a 1 y la transferencia no se realiza. El segundo argumento de entrada es la longitud en palabras que se van a transferir (1 a 40). Si el número de palabras no está dentro del rango de 1 a 40, el argumento de salida es igual a 2 y la transferencia no se realiza.

- 0 = Transferencia efectuada correctamente
- 1 = Offset inicial ilegal
- 2 = Longitud ilegal

La integridad de la palabra se garantiza durante esta transferencia. La integridad del archivo no se garantiza. Para proporcionar integridad del archivo, pueden usarse los bits de handshaking en su programa de aplicación.

Sintaxis:

PUSH [offset de palabra inicial en el archivo de interface DH-485]
PUSH [número de palabras que se van a transferir]
CALL 84
POP [estado de la transferencia]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>40 PUSH 0 : REM OFFSET ADDRESS = 0
>50 PUSH 32 : REM WORD OFFSET = 32
>60 CALL 84 : REM TRANSFER THE DATA TO THE BASIC INPUT
    BUFFER
>70 POP R : REM GET THE OUTPUT ARGUMENT
>80 IF (R<>0) THEN PRINT “TRANSFER ERROR CODE = ”,R : REM
    PRINT ERROR

READY
>RUN

READY
>
```

CALL 90 - Lectura de archivo de datos DH-485 remoto al búfer de entradas BASIC

Objetivo:

Use CALL 90 para leer hasta 40 palabras desde la dirección de nodo designada, número de archivo, tipo de archivo y offset de elemento de un archivo de datos DH-485 remoto al búfer de entradas del módulo BASIC en la palabra 0. Esta rutina tiene seis argumentos de entrada y un argumento de salida.

El primer argumento de entrada es la dirección de nodo del dispositivo remoto (0 a 31). Si el número no está dentro del rango de 0 a 31, el argumento de salida es igual a 10 y el mensaje de lectura no se realiza.

El segundo argumento de entrada es el número de archivo del dispositivo remoto (0 a 255). Si el número no está dentro del rango de 0 a 255, el argumento de salida es igual a 11 y el mensaje de lectura no se realiza.

El tercer argumento de entrada es la lectura de tipo de archivo desde el dispositivo remoto. Los códigos de tipo de archivo válidos son ASC(N), ASC(S), ASC(C), ASC(T), ASC(B) y ASC(R). Si el tipo de archivo no es uno de estos tipos válidos, el argumento de salida es igual a 241, y el mensaje de lectura no se realiza.

Tabla 13.E
Tipo de archivo que se va a leer desde el dispositivo remoto

Tipo de archivo	Código de tipo de archivo	Palabras/elemento
Integer File	ASC(N)	1 palabra/elemento
Status File	ASC(S)	1 palabra/elemento
Counter File	ASC(C)	3 palabras/elemento
Timer File	ASC(T)	3 palabras/elemento
Bit File	ASC(B)	1 palabra/elemento
Control File	ASC(R)	3 palabras/elemento

El cuarto elemento de entrada es el offset de elemento inicial dentro del archivo en el dispositivo remoto (0 a 32767). Si el número no está dentro del rango de 0 a 32767, el argumento de salida es igual a 12 y la transferencia no se realiza.

Importante: El offset será el doble de lo esperado. Por ejemplo, si un offset de 3 fue PUSH, los datos se escribirán al archivo de datos DH–485 remoto empezando en el elemento 6.

El quinto argumento de entrada es el número de elementos que se va a transferir. Si el número no está dentro del rango de longitud válida especificado en la Tabla 13.F, el argumento de salida es igual a 13 y la transferencia no se realiza.

Tabla 13.F
Rango de longitud válida

Código de tipo de archivo	Rango de longitud válida
ASC(N)	1 a 40
ASC(S)	1 a 40
ASC(C)	1 a 13
ASC(T)	1 a 13
ASC(B)	1 a 40
ASC(R)	1 a 13

El sexto argumento de entrada es el valor de tiempo límite del mensaje. Este valor es el número de centenas de milisegundos permitidos para recibir la respuesta de lectura (1 a 50 = 0.1 a 5.0 segundos). Si la respuesta de lectura no se recibe dentro de este período de tiempo, el mensaje se cancela con el argumento de salida igual a 55. Si el número no está dentro del rango de 1 a 50, el argumento de salida es igual a 14 y la transferencia no se realiza.

Los datos de lectura del dispositivo remoto se leen a los búfers de entradas del módulo BASIC empezando en la palabra 0 hasta el número de palabras especificado por la longitud del elemento del mensaje.

El argumento de salida especifica el estado de la instrucción de mensaje. Al regresar del CALL, el argumento de salida tiene la siguiente definición.

Tabla 13.G
Argumento de salida

Salida decimal	Salida hexadecimal	Descripción
0	00	Efectuada correctamente.
2	02	El nodo receptor ahora no puede aceptar el mensaje.
3	03	El nodo receptor no puede responder porque el mensaje es demasiado grande.
4	04	El nodo receptor no puede responder porque no entiende los parámetros de comando.
5	05	El módulo BASIC está fuera de línea (no está en la red).
6	06	El nodo receptor no puede responder porque la función solicitada no está disponible.
7	07	El nodo receptor no responde.
10	0A	El módulo BASIC detecta dirección de nodo receptor ilegal.
11	0B	El módulo BASIC detecta número de archivo ilegal.

Salida decimal	Salida hexadecimal	Descripción
12	0C	El módulo BASIC detecta offset de elemento de archivo receptor ilegal.
13	0D	El módulo BASIC detecta longitud de archivo receptor ilegal.
14	0E	El módulo BASIC detecta valor de tiempo límite ilegal.
16	10	El nodo receptor no puede responder debido a parámetros de comando incorrectos o comando no aceptado.
55	37	Tiempo límite de mensaje (valor de tiempo límite excedido).
80	50	El nodo receptor no tiene memoria.
96	60	El nodo receptor no puede responder porque el archivo está protegido.
231	E7	El nodo receptor no puede responder porque la longitud solicitada es demasiado grande.
235	EB	El nodo receptor no puede responder porque el nodo receptor niega el acceso.
236	EC	El nodo receptor no puede responder porque la función solicitada actualmente no está disponible.
241	F1	El módulo BASIC detecta tipo de archivo receptor ilegal.
250	FA	El nodo receptor no puede responder porque otro nodo es el propietario del archivo (tiene acceso único al archivo).
251	FB	El nodo receptor no puede responder porque otro nodo es el propietario del programa (tiene acceso único a todos los archivos).

Sintaxis:

PUSH [dirección de nodo de dispositivo remoto]
PUSH [número de archivo de dispositivo remoto
PUSH [tipo de archivo de dispositivo remoto]
PUSH [offset de elemento inicial (x2) de dispositivo remoto]
PUSH [número de elementos que se van a transferir]
PUSH [valor de tiempo límite de mensaje]
CALL 90
POP [estado de instrucción de mensaje]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>20 PUSH 5 : REM REMOTE FILE   5
>30 PUSH ASC(C) : REM FILE TYPE = COUNTER
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10 : REM ELEMENT LENGTH = 10 = 30 WORDS
>60 PUSH 5 : REM TIMEOUT = 0.5 SECONDS
>70 CALL 90
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF (R<>0) THEN PRINT “READ ERROR CODE =”,R

READY
>RUN

READ ERROR CODE = 5
```

CALL 92 - Lectura de archivo de interface común DH-485 remoto al búfer de entradas BASIC

Objetivo:

Use CALL 92 para leer hasta 40 palabras desde el archivo de interface común DH-485 remoto de la dirección de nodo designada, empezando en el offset de la palabra designada, al búfer de entradas del módulo BASIC empezando en la palabra 0. Esta rutina tiene cuatro argumentos de entrada y un argumento de salida.

El primer argumento de entrada es la dirección de nodo del dispositivo remoto (0 a 31). Si el número no está dentro del rango de 0 a 31, el argumento de salida es igual a 10 y el mensaje de lectura no se realiza.

El segundo elemento de entrada es el offset de elemento inicial dentro del archivo en el dispositivo remoto (0 a 32767). Si el número no está dentro del rango de 0 a 32767, el argumento de salida es igual a 12 y la transferencia no se realiza.

Importante: El offset será el doble de lo esperado. Por ejemplo, si un offset de 3 fue PUSH, los datos serán escritos al archivo de datos DH–485 remoto empezando en el elemento 6.

El tercer argumento es el número de palabras que se va a transferir. Si el número no está dentro del rango (1 a 40), el argumento de salida es igual a 13 y la transferencia no se realiza.

El cuarto argumento de entrada es el valor de tiempo límite del mensaje. Este valor es el número de centenas de milisegundos permitidos para recibir la respuesta de lectura (1 a 50 = 0.1 a 5.0 segundos). Si la respuesta de lectura no se recibe dentro de este período de tiempo, el mensaje se cancela con el argumento de salida igual a 55. Si el número no está dentro del rango de 1 a 50, el argumento de salida es igual a 14 y la transferencia no se realiza.

Los datos de lectura del dispositivo remoto se leen al búfer de entradas del módulo BASIC empezando en la palabra 0 hasta el número de palabras especificado por la longitud de la palabra del mensaje.

El argumento de salida especifica el estado de la instrucción de mensaje. Al regresar del CALL, el argumento de salida tiene la siguiente definición.

Tabla 13.H
Argumento de salida

Salida decimal	Salida hexadecimal	Descripción
0	00	Efectuada correctamente.
2	02	El nodo receptor ahora no puede aceptar el mensaje.
3	03	El nodo receptor no puede responder porque el mensaje es demasiado grande.
4	04	El nodo receptor no puede responder porque no entiende los parámetros de comando.
5	05	El módulo BASIC está fuera de línea (no está en la red).
6	06	El nodo receptor no puede responder porque la función solicitada no está disponible.
7	07	El nodo receptor no responde.
10	0A	El módulo BASIC detecta dirección de nodo receptor ilegal.
11	0B	El módulo BASIC detecta número de archivo ilegal.
12	0C	El módulo BASIC detecta offset de elemento de archivo receptor ilegal.
13	0D	El módulo BASIC detecta longitud de archivo receptor ilegal.
14	0E	El módulo BASIC detecta valor de tiempo límite ilegal.
16	10	El nodo receptor no puede responder debido a parámetros de comando incorrectos o comando no aceptado.
55	37	Tiempo límite de mensaje (valor de tiempo límite excedido).
80	50	El nodo receptor no tiene memoria.
96	60	El nodo receptor no puede responder porque el archivo está protegido.
231	E7	El nodo receptor no puede responder porque la longitud solicitada es demasiado grande.
235	EB	El nodo receptor no puede responder porque el nodo receptor niega el acceso.
236	EC	El nodo receptor no puede responder porque la función solicitada actualmente no está disponible.
241	F1	El módulo BASIC detecta tipo de archivo receptor ilegal.

Salida decimal	Salida hexadecimal	Descripción
250	FA	El nodo receptor no puede responder porque otro nodo es el propietario del archivo (tiene acceso único al archivo).
251	FB	El nodo receptor no puede responder porque otro nodo es el propietario del programa (tiene acceso único a todos los archivos).

Sintaxis:

PUSH [dirección de nodo de dispositivo remoto]
PUSH [offset de elemento inicial (x2) de archivo de dispositivo remoto]
PUSH [número de palabras que se van a transferir]
PUSH [valor de tiempo límite de mensaje]
CALL 92
POP [estado de instrucción de mensaje]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>30 PUSH 1 : REM REMOTE NODE ADDRESS = 1
>40 PUSH 0 : REM OFFSET = 0
>50 PUSH 10 : REM WORD LENGTH = 10
>60 PUSH 5   REM TIME–OUT VALUE = 0.5 SECONDS
>70 CALL 92
>80 POP R : REM GET THE OUTPUT ARGUMENT
>90 IF (R<>0) THEN PRINT “READ ERROR CODE IS”,R : REM
    PRINT ERROR

READY
>RUN

READ ERROR CODE IS 5
```

CALL 117 - Obtención de la longitud del paquete DF1

Objetivo:

Use CALL 117 para obtener la longitud del paquete de datos DF1. Esta rutina no tiene argumentos de entrada y tiene un argumento de salida. El argumento de salida señala la longitud del paquete DF1 más antiguo en la cola del búfer de recepción DF1.

Importante: Si el búfer de recepción se encuentra vacío, 0000 regresa a la pila de argumentos.

Cuando el CALL 117 se lee en un programa, el módulo BASIC hace una verificación para ver si la comunicación DF1 se ha habilitado a través del CALL 108. Si la comunicación DF1 no se ha habilitado, se imprime un mensaje de error en el dispositivo de la consola y el módulo BASIC entra al modo de comando.

Después que se ha llamado la longitud del paquete DF1, ésta debe usarse junto con la instrucción GET para recuperar los datos en el paquete DF1 recibido.

Sintaxis:

CALL 117
POP [longitud del paquete DF1]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 117
>20 POP X
>30 END
```

CALL 118 - Escrituras no solicitadas PLC/SLC

Objetivo:

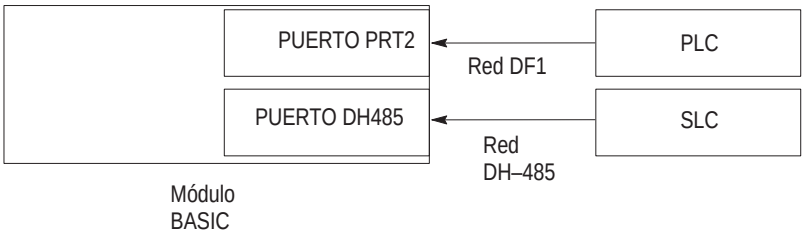
Use CALL 118 para permitir que el módulo BASIC reciba paquetes de datos enviados por instrucciones de mensajes PLC–2, PLC–3 o PLC–5 en la red DF1. Este CALL también configura el módulo BASIC para que reciba paquetes de datos desde un nodo SLC en la red DH–485. Los puertos DF1 (PRT2) y DH485 no pueden estar activos simultáneamente. El puente JW4 en el módulo BASIC se usa para seleccionar su configuración de puerto.

Cualquier instrucción de mensaje de escritura enviada al módulo BASIC desde estos PLC/SLC hace que los datos sean colocados en una cadena interna dentro del archivo M1 CPU, en el archivo de imagen de entrada CPU y/o en la cadena del módulo BASIC, empezando en el offset de la palabra designada.

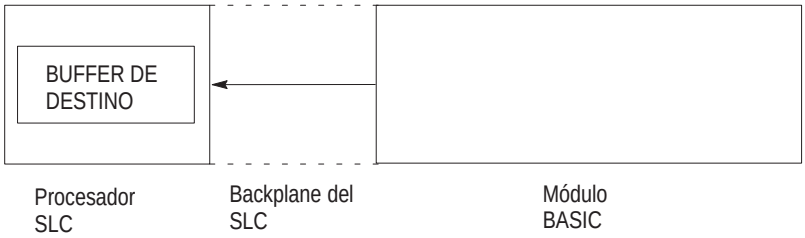
El byte inferior de la primera palabra del archivo de destino contiene el conteo de caracteres (conteo de bytes) de los datos transferidos. El byte superior de esta palabra no se usa. Si se selecciona una cadena interna, el primer carácter contiene el conteo de bytes. El segundo carácter (número de transacción) de la cadena interna se incrementa cuando se efectúa la recepción de un paquete, para informar al módulo BASIC que hay datos nuevos en la cadena. El valor de este número de transacción vuelve de 255 a 0.

Ejecute el CALL 118 una vez. Después que se ejecuta el CALL, el módulo BASIC verifica el puerto al final de cada línea de código BASIC. El módulo obtiene datos nuevos desde el PLC o SLC y los transfiere como se describe a continuación:

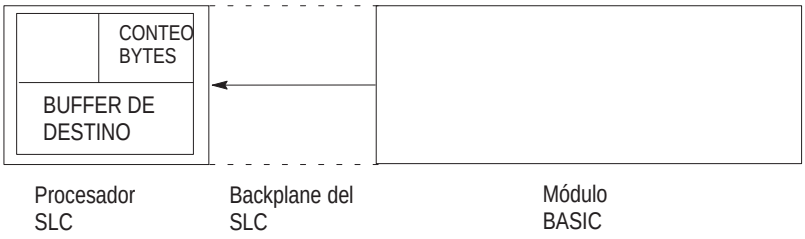
1. El módulo BASIC recibe paquetes iniciados desde el PLC/SLC configurado en este CALL a través de los puertos PRT2 o DH485. Los puertos PRT2 y DH485 no pueden estar activos simultáneamente. Para hacer esta selección se usa el puente JW4.



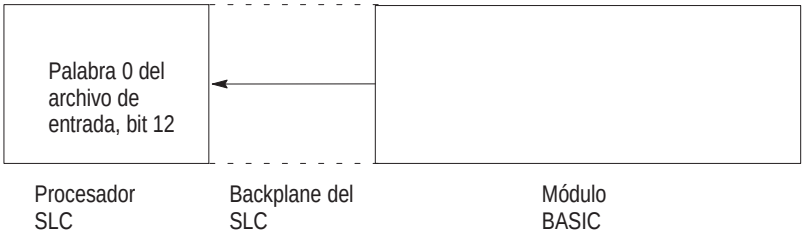
2. El módulo BASIC transfiere los datos al búfer de destino SLC local.



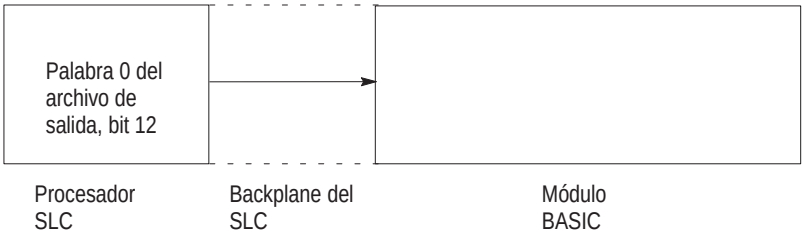
3. El módulo BASIC coloca el conteo de bytes en el byte inferior de la primera palabra disponible del búfer de destino.



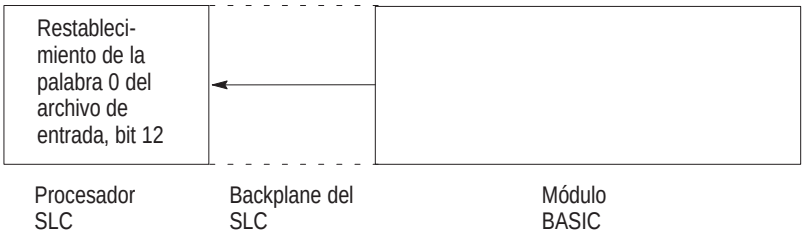
4. El módulo BASIC establece la palabra 0 del archivo de entrada, bit 12 para iformar al procesador SLC que hay datos válidos disponibles.



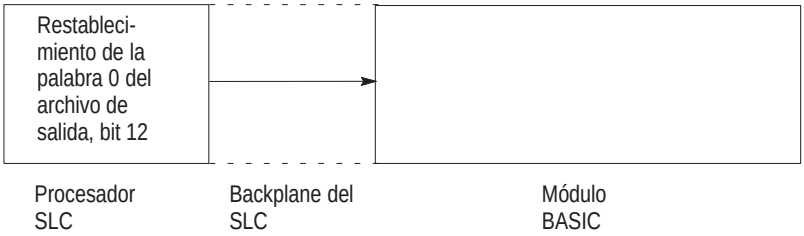
5. El procesador SLC recupera los datos del búfer y establece la palabra 0 del archivo de salida, bit 12, para informar al módulo BASIC que los datos fueron recibidos.



6. El módulo BASIC restablece la palabra 0 del archivo de entrada, bit 12, en el mismo ciclo de fin de escán en el cual se restableció la palabra 0 del archivo de salida, bit 12.



7. El SLC restablece la palabra 0 del archivo de salida, bit 12. El módulo BASIC puede empezar a cargar el búfer de destino con el siguiente paquete cuando éste se recibe.



El procesador SLC no debe establecer, y luego restablecer, la palabra 0 del archivo de salida, bit 12, en el mismo ciclo de escán de lógica de escalera. Si esto ocurre, el módulo BASIC puede perder el bit que se está estableciendo.

Este CALL está activo hasta que se vuelve a ejecutar con parámetros de entrada diferentes. Si esto ocurre, el CALL 118 previo se inhabilita automáticamente y el CALL 118 nuevo se hace efectivo. Múltiples CALL 118 no se ejecutan en paralelo.

Este CALL tiene cinco argumentos de entrada y un argumento de salida.

El primer argumento de entrada habilita o inhabilita el CALL:

- 0 = Inhabilita el CALL 118 ejecutado previamente
- 1 = Habilita el CALL. Se aceptan los siguientes comandos:

  PLC (Escrituras no protegidas)
  PLC (Escrituras con rango de palabra)
  PLC (Mensajes de escritura)
  SLC 5/02 (Escrituras no protegidas)
  SLC 5/02 (Mensajes de escritura)

Si los datos recibidos exceden la longitud de la cadena o el tamaño del archivo CPU, los datos restantes se truncan.

El segundo argumento de entrada es la selección del archivo de imagen de entrada CPU de destino con o sin la cadena interna, el archivo M1 CPU con o sin la cadena interna, o la cadena interna sola:

- 0 = Archivo de imagen de entrada CPU
- 1 = Archivo M1 CPU
- 2 = Cadena interna
- 3 = Archivo de imagen de entrada CPU y cadena interna
- 4 = Archivo M1 CPU y cadena interna

Si se selecciona la cadena interna (2), los bits de handshaking de datos de imagen de entrada/salida (palabra 0, bit 12) no son usados por el módulo BASIC para indicar que se recibieron datos. En el programa BASIC, usted debe monitorizar el segundo carácter de la cadena (número de transacción), el cual se incrementa con cada transferencia de datos. Luego usted debe retirar los datos de la cadena antes que se reciba el siguiente paquete, de lo contrario se perderán los datos.

El tercer argumento de entrada es el offset de palabra dentro del archivo CPU de destino. Este offset apunta a la primera palabra que contiene el conteo de bytes de los datos válidos que son transferidos. El offset de la cadena interna siempre es 2. El conteo de bytes se coloca en la ubicación de carácter 0, y la ubicación 1 es un número de transacción que se incrementa cada vez que se completa un paquete de datos.

Si se usa el puerto DH485 para transferencia de datos, no se debe usar un offset de más de 40 palabras hexadecimales (64 decimales). Los paquetes de escrituras no solicitadas de más de 64 causan una escritura al búfer del puerto de programación DH485, lo cual causa una operación incorrecta. El tamaño del paquete de datos puede ser hasta el máximo para el archivo de entrada seleccionado.

Si se selecciona el archivo de imagen de entrada CPU como destino, este offset no debe ser 0 ni 1. Cero es una palabra reservada para los bits de handshaking. La palabra 1 está reservaa para el número de transacción PLC usado con CALL 27, 28, 122 y 123. Un 0 ó 1 causa que un error sea POP (2 = parámetro de entrada incorrrecto) y el CALL no se ejecuta.

El cuarto argumento de entrada es el número de cadena. Si el segundo argumento de entrada no selecciona uso de cadena interna, el valor de este argumento de entrada se ignora, pero debe ser PUSH.

El quinto argumento de entrada es la longitud máxima de palabra permitida para el paquete de datos. Los paquetes de mayor tamaño recibidos por el módulo BASIC son rechazados. El introducir 0 hace que el módulo BASIC acepte paquetes de cualquier tamaño y todos los paquetes se reciben hasta la longitud máxima del archivo de destino. El exceso de datos se trunca.

El argumento de salida es el estado del CALL. Tiene los siguientes valores:

- 0 = Efectuado correctamente
- 1 = Inhabilitado
- 2 = Parámetro de entrada incorrecto
- 3 = Puerto DH485/DF1 seleccionado no habilitado
- 4 = La cadena es demasiado pequeña
- 5 = La cadena no está dimensionada

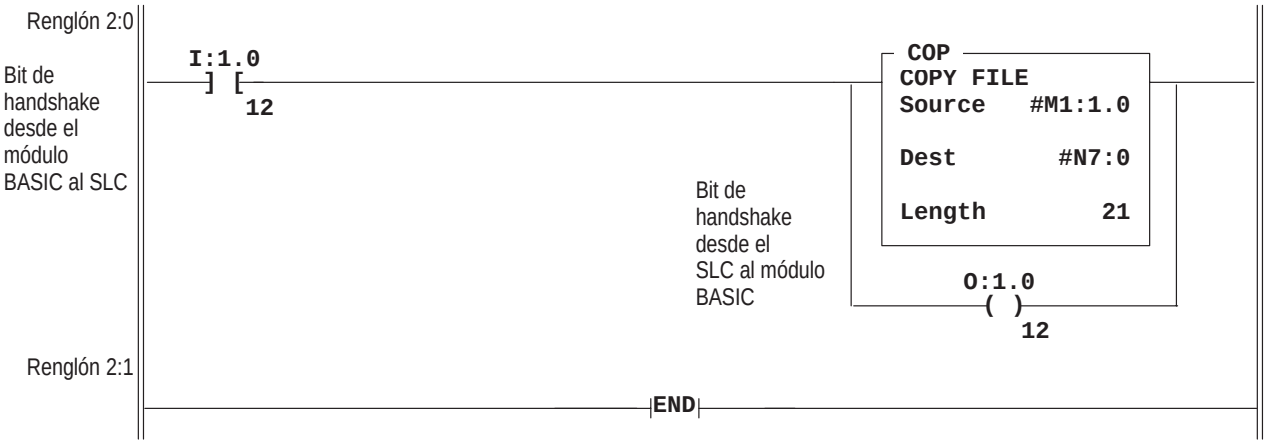
Sintaxis:

PUSH [habilitación/inhabilitación de CALL]
PUSH [selección de archivo y/o cadena de destino]
PUSH [offset de palabra en archivo de destino]
PUSH [número de cadena]
PUSH [longitud máxima de palabra]
CALL 118
POP [estado de CALL 118]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM ENABLE PLC/SLC UNSOLICITED WRITE INTERRUPT
>20 PUSH 1 : REM ENABLE THE CALL
>30 PUSH 1 : REM DESTINATION SLC M1 FILE
>40 PUSH 0 : REM WORD OFFSET INTO M1 FILE
>50 PUSH 0 : REM STRING NUMBER – NOT USED
>60 PUSH 20 : REM MAX ALLOWED WORD LENGTH OF DATA PACKET
>70 CALL 118
>80 POP S
>90 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 118 SETUP”
```

A continuación presentamos un ejemplo de programa de lógica de escalera para el CALL 118. El módulo BASIC está ubicado en la ranura 1 del rack SLC. El renglón 2:0 copia los datos desde el archivo M1 cuando el bit de handshake (I:1.0/12) es establecido por el módulo BASIC. El renglón 2:0 establece el bit de handshake (O:1.0/12) una vez que los datos se han copiado del archivo M1. Esto indica al módulo BASIC que desactive I:1.0/12. La primera palabra es el conteo de bytes. Se espera un máximo de 20 palabras de datos.



CALL 122 - Lectura de archivo de datos PLC DF1 remoto

Objetivo:

Use CALL 122 para leer hasta 64 palabras de datos desde un nodo DF1 remoto (PLC–2, –3 ó –5) al archivo de imagen de entrada CPU, al archivo M1 CPU y/o a una cadena dentro del módulo BASIC.

La siguiente tabla lista notas específicas cuando se usa el CALL 122 con el PLC–3 y PLC–5.

Tabla 13.I
Notas de aplicación PLC

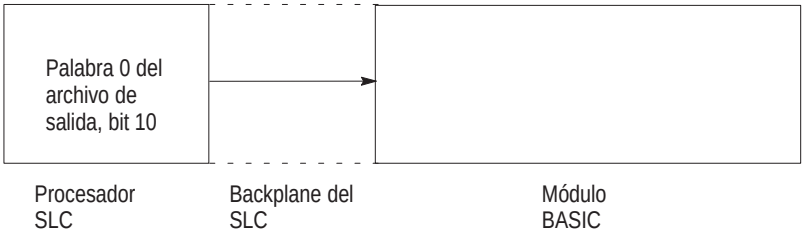
PLC	Notas
–3	Para temporizadores y contadores, el número de archivo PUSH (tercer parámetro) es el número de estructura, limitado a un máximo de 255 palabras.
–5	Para datos de temporizador, un elemento es tres palabras de 16 bits, almacenadas en el archivo de destino en el siguiente orden: control, valor preseleccionado y acumulador.

Si se selecciona una cadena interna, el primer carácter (número de transacción) se incrementa cuando se efectúa una transacción de lectura para informar al módulo BASIC que hay datos nuevos en la cadena. El valor del número de transacción vuelve de 255 a 0.

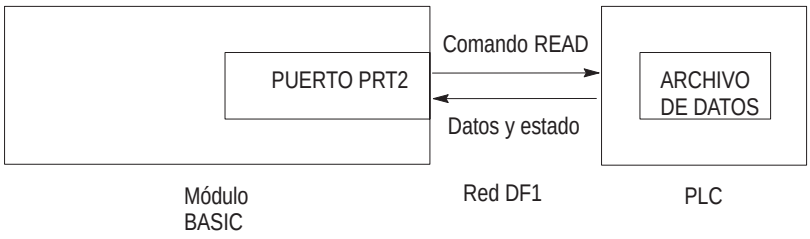
Los parámetros del puerto DF1 deben configurarse con el CALL 108. El puerto DF1 puede operar con el protocolo full–duplex o half–duplex esclavo.

Ejecute el CALL 122 una vez para establecer los parámetros de transferencia de datos. Los bits de imagen de entrada y salida (palabra 0, bit 10) para el slot que contiene el módulo BASIC se usan para iniciar y notificar que se completó la transferencia. La operación se describe a continuación:

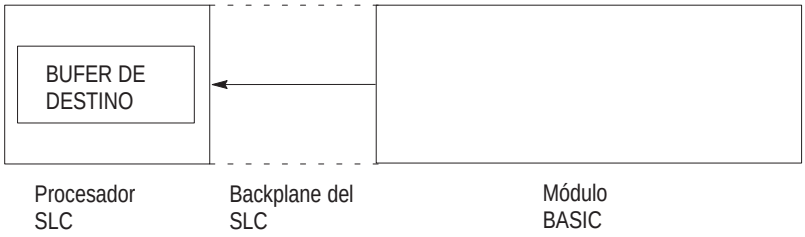
1. El procesador SLC establece la palabra 0 del archivo de salida, bit 10, para informar al módulo BASIC que el comando READ configurado en el CALL 122 debe ejecutarse.



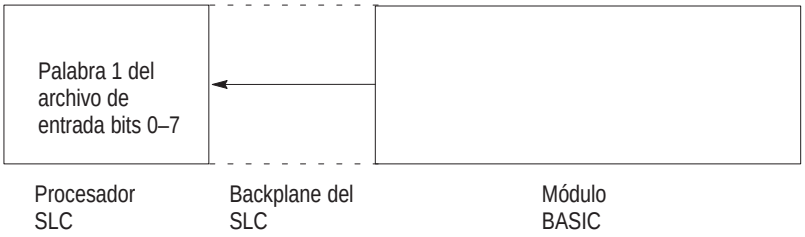
2. El módulo BASIC emite el comando READ apropiado al PLC. Los datos y el estado se reciben desde el PLC.



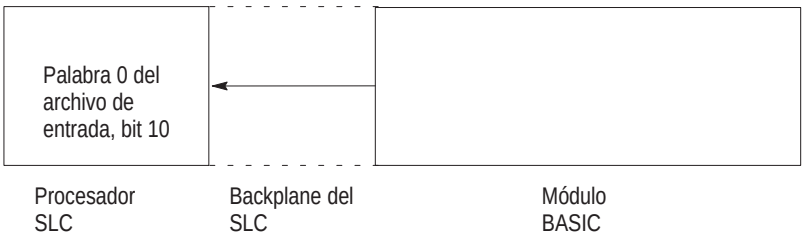
3. Cuando hay datos disponibles, el módulo BASIC transfiere los datos al búfer de destino.



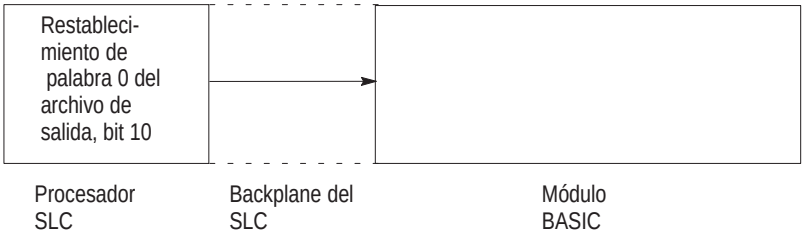
4. El módulo BASIC coloca el estado de la transacción en la palabra 1 de entrada, bits 0–7.



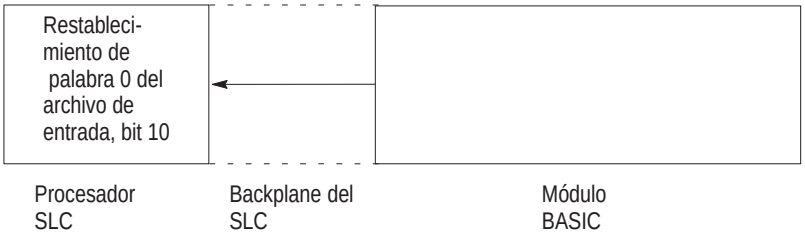
5. El módulo BASIC establece la palabra 0 del archivo de entrada, bit 10, para informar al procesador SLC que hay datos válidos e información de estado disponible.



6. El SLC recupera los datos e información de estado del búfer y restalece la palabra 0 del archivo de salida, bit 10, para informar al módulo BASIC que los datos fueron recibidos.



7. El módulo BASIC restablece la palabra 0 del archivo de entrada, bit 10, en el mismo ciclo de fin de escán en el que se restableció la palabra 0 del archivo de salida, bit 10.



El procesador SLC no debe establecer, y luego restablecer, la palabra 0 del archivo de salida, bit 10, en el mismo ciclo de lógica de escalera. Si esto ocurre, el módulo BASIC puede perder el bit que se está estableciendo.

Este CALL está activo hasta que se vuelve a ejecutar con parámetros de entrada diferentes.

Este CALL tiene diez argumentos de entrada y un argumento de salida.

El primer argumento de entrada es el tipo de comando READ PLC emitido:

- 0 = Inhabilita el CALL ejecutado previamente
- 2 = Archivo de interface común – Comando READ PLC–2 no protegido
- 3 = Archivo PLC–3 – comando READ de rango de palabra
- 5 = Archivo PLC–5 – comando de mensaje READ

El segundo argumento de entrada es la dirección de nodo del dispositivo PLC remoto (0 a 255). Si el número no está dentro de este rango, el estado es igual a 2 y el mensaje de lectura no se realiza.

El tercer argumento de entrada es el número de archivo que se va a leer en el dispositivo remoto PLC (0 a 255). Si el número no está dentro de este rango, el estado es igual a 2 y el mensaje de lectura no se realiza. El parámetro se ignora si el archivo de interface común se selecciona en el primer parámetro, pero debe ser PUSH.

El cuarto argumento de entrada es el tipo de archivo que se va a leer desde el dispositivo PLC remoto. Introduzca el código de tipo de archivo tal como se muestra a continuación. Este argumento se ignora si el archivo de interface común se selecciona en el primer parámetro, pero debe ser PUSH (supone tipo entero). Si el tipo de archivo no es uno de éstos, el estado es igual a 2 y el mensaje de lectura no se realiza.

Tabla 13.J
Tipo de archivo que se va a leer desde el dispositivo remoto

Tipo de archivo	Código de tipo de archivo	Palabras/elemento (1 palabra = 16 bits)
Archivo de enteros	ASC(N)	1 palabra/elemento
Archivo de estado	ASC(S)	1 palabra/elemento
Archivo de contador	ASC(C)	3 palabras/elemento
Archivo de temporizador	ASC(T)	3 palabras/elemento
Archivo de bits	ASC(B)	1 palabra/elemento
Archivo de control	ASC(R)	3 palabras/elemento
Archivo de entrada	ASC(I)	1 palabra/elemento
Archivo de salida	ASC(O)	1 palabra/elemento

El quinto argumento de entrada es el offset de la palabra inicial dentro del archivo en el dispositivo remoto PLC–2 (0 a 32766). En el caso de archivos de enteros, binarios o de estado PLC–3, el valor es 0–9999. En el caso de archivos de E/S de temporizador o contador PLC–3, el valor debe ser 0. Si el número no está dentro de este rango, el estado es igual a 2 y la transferencia no se realiza.

El sexto argumento de entrada es el número de elementos que se va a transferir. Si el número no está dentro del rango mostrado, el estado es igual a 2 y la transferencia no se realiza.

Tabla 13.K
Rango válido de longitud de elementos

Código de tipo de archivo	Rango válido de longitud de elemento
ASC(N)	1 a 64
ASC(S)	1 a 64
ASC(C)	1 a 21
ASC(T)	1 a 21
ASC(B)	1 a 64
ASC(R)	1 a 21
ASC(I)	1 a 21
ASC(O)	1 a 21
Archivo de interface común	1 a 21

El séptimo argumento de entrada es el valor de tiempo límite del mensaje. Este valor (1 a 255) corresponde al número de centenas de milisegundos permitidos para recibir la respuesta de lectura (0.1 a 25.5 segundos). Si la respuesta de lectura no se recibe dentro de este período de tiempo, el mensaje se cancela con el estado igual a 55 en la palabra 1 del archivo de entrada, bits 0–7. Si el valor de tiempo límite no está dentro del rango (1 a 255), el estado que fue POP es igual a 2 y la transferencia no se realiza.

El octavo argumento de entrada es la selección del archivo de imagen de entrada CPU de destino con o sin la cadena interna, el archivo M1 CPU con o sin la cadena interna, o la cadena interna sola.

- 0 = Archivo de imagen de entrada CPU
- 1 = Archivo M1 CPU
- 2 = Cadena interna
- 3 = Archivo de imagen de entrada CPU y cadena interna
- 4 = Archivo M1 CPU y cadena interna

Si selecciona la cadena interna (2), el CALL 29 puede ejecutarse para iniciar cada transferencia de datos sin requerir interacción del procesador SLC. La palabra 0 del archivo de salida, bit 10, también iniciará una transacción de cadena.

El noveno argumento de entrada es el offset de palabra dentro del archivo CPU de destino. Este offset apunta a la primera palabra transferida. El offset para la cadena interna siempre es 1. El primer carácter (número de transacción en la ubicación 0) se incrementa con cada transferencia efectuada, para informar al módulo BASIC que hay datos nuevos en la cadena. El valor del número de la transacción vuelve de 255 a 0.

Si se selecciona el archivo de imagen de entrada CPU, este offset no debe ser 0 ni 1, porque éstas son palabras reservadas. Cero está reservado para bits de handshaking de transferencia de datos y la palabra 1 está reservada para el estado de la transacción. Un 0 ó un 1 causan que un error sea POP (2 = parámetro de entrada incorrecto) y el CALL no se ejecuta.

Si los datos recibidos exceden la longitud de cadena o el tamaño del archivo CPU, los datos restantes se truncan.

El décimo argumento de entrada es el número de cadena. Si el octavo argumento de entrada no selecciona uso de cadena interna, el valor de este argumento de entrada se ignora, pero debe ser PUSH.

El argumento de salida es el estado del CALL. Tiene los siguientes valores:

- 0 = Efectuado correctamente
- 1 = Inhabilitado
- 2 = Parámetro de entrada incorrecto
- 3 = DF1 no habilitado
- 4 = La cadena es demasiado pequeña
- 5 = La cadena no está dimensionada

Para inhabilitar este CALL un 0 debe ser PUSH en el primer parámetro de entrada. Todos los otros parámetros se ignoran pero deben ser PUSH.

Cada vez que se intente leer un nodo remoto, el estado de la lectura se coloca en la palabra 1 de entrada, bits 0–7. Los códigos de estado posibles se muestran en la Tabla 13.L.

El estado es válido cuando el módulo BASIC establece la palabra 0 del archivo de entrada, bit 10.

Tabla 13.L
Códigos de estado de transacción

Código	Indica
0	Transferencia OK.
1	Transmsión fallida.
2	Tiempo límite de consulta.
3	Con handshaking seleccionado – ocurrió una pérdida de señal CTS durante la transmisión o un fallo irremediable de transmisión.
	Sin handshaking seleccionado – ocurrió un fallo irremediable de transmisión..
4	Fallo de transmisión debido a desconexión de modem (pérdida de señal DCD durante más de10 segundos) si se seleccionó handshaking de modem con portadora constante.
5	El controlador DF1 no está habilitado.
6	Tiempo límite de mensaje.
81	Comando o formato ilegal.
82	El dispositivo principal tiene un problema y no se comunica.
83	El dispositivo principal de estación remota no está presente, está desconectado o desactivado.
84	El dispositivo principal no pudo completar la función debido a fallo de hardware.
85	Problema de direccionamiento o renglones de protección de memoria.
86	Función no permitida debido a selección de protección de comando.
87	El procesador está en el modo de Programación.
88	Archivo de modo de compatibilidad ausente o problema de zona de comunicación.
89	La estación remota no puede colocar comando en el búfer.
8B	Problema de estación remota debido a descarga.
8C	La estación local no puede ejecutar comando debido a IPB activos..
C1	Formato de dirección ilegal – el campo tiene un valor ilegal.
C2	Formato de dirección ilegal – no hay suficientes campos especificados.
C3	Formato de dirección ilegal – hay demasiados campos especificados.
C4	Formato de dirección ilegal – no se encontró símbolo.
C5	Formato de dirección ilegal – el símbolo es 0 o mayor que el número máximo de caracteres aceptados por este dispositivo.
C6	Dirección ilegal – la dirección no existe o no apunta a algo utilizable en este comando.
C7	Tamaño ilegal – archivo de tamaño incorrecto; la dirección está más allá del final del archivo.
C8	No puede completar petición.
C9	Datos o archivo muy grande.
CA	La petición es muy larga; el tamaño de la transacción más la dirección de palabra es muy grande.

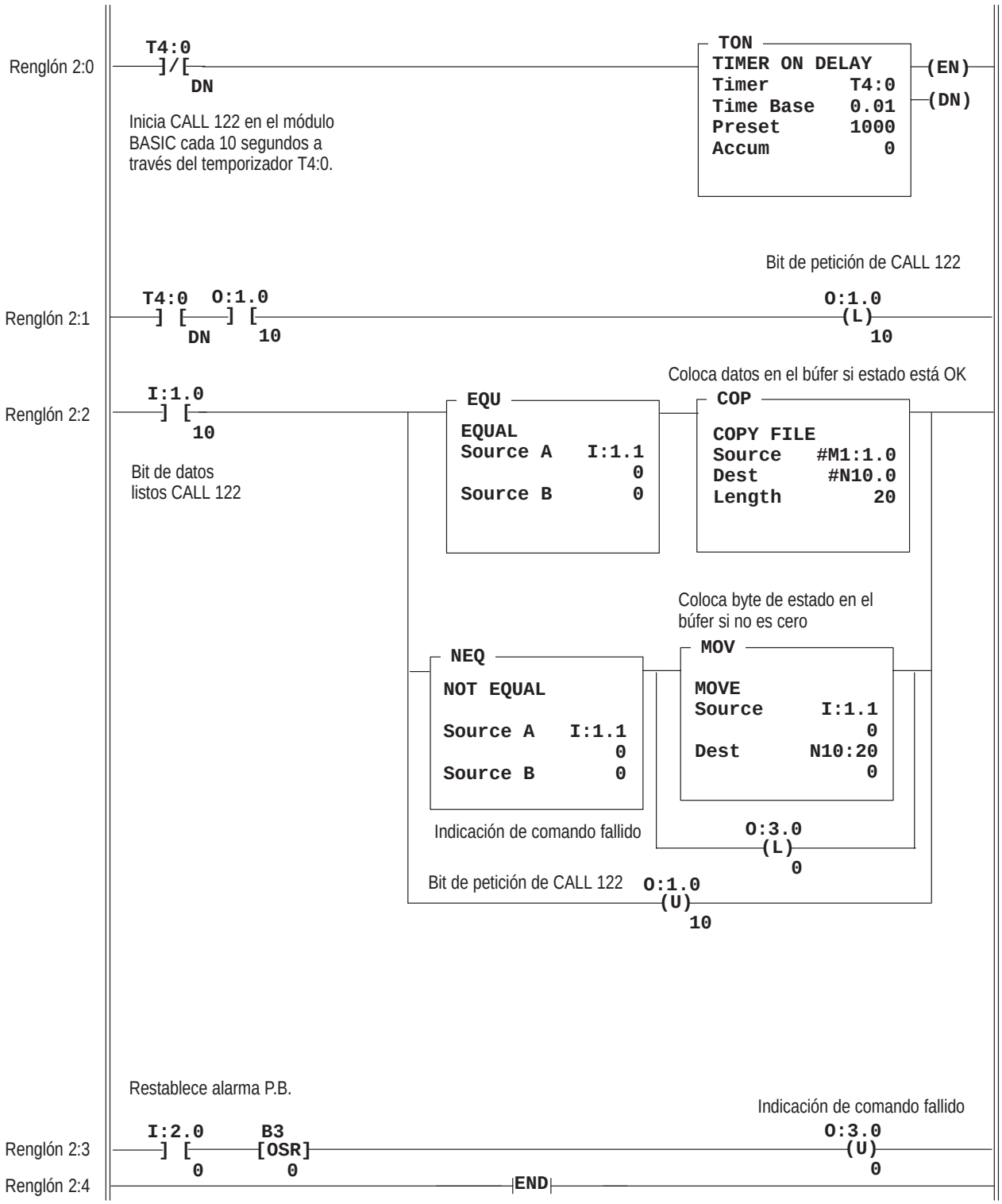
CB	Acceso negado, violación de privilegio.
CC	Recurso no disponible; la condición no puede generarse.
CD	El recurso ya está disponible; la condición ya existe.
CE	El comando no puede ejecutarse.
CF	Overflow; overflow de histograma.
D0	No hay acceso.
D1	Información de tipo de datos ilegal.
D2	Parámetro inválido datos inválidos en búsqueda o bloque de comando.
D3	Existe referencia de dirección a área borrada.
D4	Fallo de ejecución de comando por razón desconocida; overflow de histograma PLC–3.
D5	Error de conversión de datos.
D6	El escáner no se puede comunicar con un adaptador de chasis 1771.
D7	El adaptador no se puede comunicar con el módulo.
D8	La respuesta del módulo 1771 no fue válida.
D9	Etiqueta duplicada.
DA	Archivo abierto – otra estación es propietaria del mismo.
DB	Otra estación es propietaria del programa.

Sintaxis:

PUSH [tipo de comando READ PLC]
PUSH [dirección de nodo PLC remoto]
PUSH [número de archivo de PLC remoto]
PUSH [tipo de archivo en PLC remoto]
PUSH [offset de elemento inicial en PLC remoto]
PUSH [número de elementos que se van a transferir]
PUSH [valor de tiempo límite de mensaje]
PUSH [selección de archivo de destino]
PUSH [offset de palabra dentro de archivo de destino]
PUSH [número de cadena]
CALL 122
POP [estado de CALL 122]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM ENABLE DF1 PLC REMOTE READ COMMAND
>20  PUSH 5 : REM PLC–5 FILE
>30  PUSH 0 : REM NODE ADDRESS OF PLC–5
>40  PUSH 7 : REM FILE NUMBER OF PLC–5
>50  PUSH ASC(N) : REM FILE TYPE OF PLC–5
>60  PUSH 0 : REM STARTING WORD OFFSET OF PLC–5 FILE
>70  PUSH 20 : REM NUMBER OF DATA WORDS TO READ
>80  PUSH 10 : REM COMMAND TIME–OUT VALUE (X100MS)
>90  PUSH 1 : REM DESTINATION IS SLC M1 FILE
>100 PUSH 0 : REM WORD OFFSET WITHIN M1 FILE
>110 PUSH 0 : REM STRING NUMBER – NOT USED FOR THIS
     EXAMPLE
>120 CALL 122
>130 POP S : REM STATUS OF THE CALL
>140 IF (S<>0) THEN PRINT “UNSUCCESSFUL CALL 122 SETUP”
```



GET

Objetivo:

Use el operador GET en el modo de marcha (Run). Devuelve un resultado de cero en el modo de comando. El operador GET lee el dispositivo de entrada de la consola. Si hay un carácter disponible del dispositivo de la consola, el valor del carácter se asigna a GET. Después que se lee GET en el programa, se le asigna el valor de cero hasta que otro carácter sea enviado desde el dispositivo de la consola.

Use el operador GET# para leer el puerto PRT2 y el operador GET@ para leer el puerto PRT1. El siguiente ejemplo imprime la representación decimal de cualquier carácter enviado desde el dispositivo de la consola.

Sintaxis:

GET

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 A = GET
>20 IF (A<>0) THEN PRINT A : REM ZERO MEANS NO ENTRY
>30 GOTO 10
>RUN

 65 [A]
 49 [1]
 24 [^X]
 50 [2]

STOP – IN LINE  30
READY
>
```

El operador GET se lee sólo una vez antes de que se le asigne un valor de cero. Esto garantiza que el primer carácter introducido siempre sea leído, independientemente de dónde se coloque el operador GET en el programa. No hay almacenamiento de caracteres en el búfer en el puerto de programación.

INPL

Objetivo:

Use la instrucción INPL para leer una línea completa (hasta 254 caracteres) desde el búfer del puerto de programación. La línea debe estar almacenada en una variable de cadena. La instrucción INPL lee todos los caracteres del puerto de programación hasta un retorno de carro o hasta llegar al límite de 254 caracteres, lo que ocurra primero. INPL no transmite los caracteres leídos desde el puerto de programación.

Use la instrucción INPL# para leer una línea completa de caracteres desde el búfer del puerto PRT2. Use la instrucción INPL@ para leer una línea completa de caracteres desde el búfer del puerto PRT1. Ambas instrucciones funcionan como la instrucción INPL.

Sintaxis:

INPL variable_de cadena

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 270,254 : REM ONE STRING OF < 254 BYTES
>20 INPL $(0) : REM READ LINE FROM PROGRAM PORT
>30 PRINT# $(0) : REM ECHO STRING TO PORT PRT2
```

INPS

Objetivo:

Use la instrucción INPS para leer una cadena completa de caracteres desde el búfer del puerto de programación. Los caracteres no se transmiten. Para las comunicaciones se prefiere la instrucción INPS en lugar de INPUT o INPL porque todos los caracteres ASCII pueden ser significativos. INPUT es la que menos se prefiere porque se detiene cuando aparece una coma o un retorno de carro. INPL termina cuando aparece un retorno de carro.

Use la instrucción INPS# para leer una cadena completa de caracteres desde el búfer del puerto PRT2. Use la instrucción INPS@ para leer una cadena entera de caracteres desde el búfer del puerto PRT1. Ambas instrucciones funcionan como la instrucción INPS.

Sintaxis:

INPS variable_de cadena, número_de_caracteres

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>100 PRINT, “TYPE P TO PROCEED OR S TO STOP”
>110 REM READ SINGLE CHARACTER FROM PROGRAM PORT
>120 INPS $(0),1
>130 IF ASC($(0),1)= ASC(P) GOTO 500
>140 IF ASC($(0),1)= ASC(S) GOTO 700
>150 GOTO 100
```

INPUT

Objetivo:

Use la instrucción INPUT para introducir datos desde el dispositivo de la consola durante la ejecución del programa. Tiene que asignar datos a una o más variables con una sola instrucción de entrada. Debe separar las variables con una coma.

Use la instrucción INPUT# para entrar datos desde el puerto PRT2. Use la instrucción INPUT@ para entrar datos desde el puerto PRT1. Ambas instrucciones funcionan como la instrucción INPUT.

Sintaxis:

INPUT

Ejemplos:

```
>INPUT A,C
```

>INPUT A,C hace que se imprima un signo de interrogación (?) en el dispositivo de la consola. Esto le indica que introduzca dos números separados por una coma. Si no introduce suficientes datos, el módulo imprime **TRY AGAIN** en el dispositivo de la consola.

```
>1  REM EXAMPLE PROGRAM
>10 INPUT A,C
>20 PRINT A,C
>RUN

?1

TRY AGAIN

?1,2
l      2

READY
```

Puede escribir la instrucción INPUT de manera que un mensaje descriptivo le indique qué debe introducir. El mensaje impreso se coloca entre comillas después de la instrucción INPUT. Si aparece una coma antes de la primera variable en la lista de entrada, no aparecerá el carácter de mensaje de signo de interrogación.

```
>1  REM EXAMPLE PROGRAM
>10 INPUT “ENTER A NUMBER”A
>20 PRINT SQR(A)
>30 END

READY
>RUN

ENTER A NUMBER
?4
 2

READY
>

>NEW

>1  REM EXAMPLE PROGRAM
>10 INPUT “ENTER A NUMBER – ”,A
>20 PRINT SQR(A)
>30 END

>RUN

ENTER A NUMBER – 25
 5

READY
>
```

También puede asignar cadenas con una instrucción INPUT. Las cadenas siempre se terminan con un retorno de carro (cr). Si se solicita más de una entrada de cadena con una sola instrucción INPUT, el módulo le envía un mensaje con un signo de interrogación.

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,20
>20 INPUT "NAME(CR),AGE – ",$(1),A
>30 PRINT "HELLO ",$(1), "YOU ARE ",A," YEARS OLD."
>40 END

READY
>RUN

NAME(CR),AGE – PAM
?29
HELLO PAM YOU ARE 29 YEARS OLD.

READY
>
```

Puede asignar cadenas y variables con una sola instrucción INPUT.

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,10
>20 INPUT "NAME(CR), AGE – ",$(1),A
>30 PRINT "HELLO ",$(1),", YOU ARE ", A," YEARS OLD"
>40 END
>RUN

NAME(CR),AGE – FRED
?15
HELLO FRED, YOU ARE 15 YEARS OLD

READY
>
```

LD@

Importante: Esta instrucción no está asociada con ninguna designación de puerto.

Objetivo:

Use la instrucción LD@ para recuperar números en punto (coma) flotante que fueron almacenados con una instrucción ST@. La expresión [expr] que sigue a la instrucción LD@ especifica la dirección donde se almacena el número después de ejecutar LD@. La instrucción LD@ coloca el número en ARGUMENT STACK (pila de argumentos) en la ubicación de dirección especificada por [expr].

Esta instrucción puede usarse con CALL 77 para recuperar variables desde un área protegida de la memoria. Esta área protegida no se pone a cero en el encendido ni cuando se emite el comando RUN.

Importante: LD@ no se usa con ninguna designación de puerto.

Sintaxis:

LD@ [expr]

Ejemplo:

```
>P. MTOP
 24515

P. MTOP 10*6
 24455

>PUSH 24455 : CALL 77

>1   REM EXAMPLE PROGRAM
>5   DIM A(10),B(10)
>10  REM *** ARRAY SAVE ***
>20  FOR I = 0 TO 9
>30  A(I) = I+20
>40  PUSH A(I) : REM PUT NUMBER ON STACK
>50  ST@ 5FFFH-I*6
>60  NEXT I
>70  REM *** GET ARRAY ***
>80  FOR I = 0 TO 9
>90  LD@ 5FFFH-I*6
>100 POP B(I) : REM GET NUMBER FROM STACK
>110 PRINT B(I)
>120 NEXT I0

READY
>RUN

 20
 21
 22
 23
 24
 25
 26
 27
 28
 29

READY
>PUSH 5FFFH : CALL 77

>P. MTOP
 24575
```

READ

Objetivo:

Use la instrucción READ para recuperar las expresiones que están especificadas en la instrucción DATA y asignar el valor de la expresión a la variable en la instrucción READ. La instrucción READ siempre va seguida de una o más variables. Si más de una variable sigue a la instrucción READ, éstas se separan con una coma.

Sintaxis:

READ

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 FOR I = 1 TO 3
>20 READ A,C
>30 PRINT A,C
>40 NEXT I
>50 RESTORE
>60 READ A,C
>70 PRINT A,C
>80 DATA 10,20,10/2,20/2,SIN(PI),COS(PI)

READY
>RUN

 10  20
 5  10
 0 -1
 10  20

READY
>
```

Cada vez que se encuentra una instrucción READ, la siguiente expresión consecutiva en la instrucción DATA se evalúa y se asigna a la variable en la instrucción READ. Puede colocar instrucciones DATA en cualquier lugar dentro de un programa. Estas no se ejecutan y no causan un error. Las instrucciones DATA se consideran encadenadas y aparecen como una instrucción DATA grande. Si en cualquier momento se leen todos los datos y se ejecuta otra instrucción READ, el programa termina y el mensaje **`ERROR:  NO DATA – IN LINE XX`** se imprime en el dispositivo de la consola.

Capítulo 14

Funciones de configuración

Este capítulo describe e ilustra los comandos usados para establecer los parámetros de puerto dentro del programa BASIC o desde la línea de comando. La Tabla 14.A lista los mnemónicos correspondientes.

Tabla 14.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Establecer los parámetros del puerto PRT2.	CALL 30	14-1
Establecer la velocidad en baudios del puerto de programación.	CALL 78	14-3
Restablecer el señalador del cabezal impresor.	CALL 99	14-4
Restablecer el puerto PRT1 a sus valores predeterminados.	CALL 105	14-4
Restablecer el puerto PRT2 a sus valores predeterminados.	CALL 119	14-5
Establecer los parámetros de los puertos PRT1, PRT2 y DH485.	MODE	14-5

CALL 30 - Establecimiento de los parámetros del puerto PRT2

Objetivo:

Use CALL 30 para establecer los parámetros del puerto PRT2. La Tabla 14.B lista los parámetros del puerto PRT2 y sus selecciones en el orden en que son colocados (PUSH) en la pila antes de ejecutar el CALL.

Tabla 14.B
Parámetros del puerto PRT2

Parámetros del puerto PRT2	Selecciones
Bits por palabra	5, 6, 7, 8
Habilitación de paridad	0 = Ninguna, 1 = Impar, 2 = Par
Números de bits de parada	1 = 1 bit de parada , 2 = 2 bits de parada, 3 = 1.5 bits de parada
Handshaking de software	0 = Ninguna, 1 = XON-XOF
Handshaking de hardware	0 = DCD inhabilitado, 1 = DCD habilitado

Sintaxis:

PUSH [bits por palabra]
PUSH [habilitación de paridad]
PUSH [número de bits de parada]
PUSH [habilitación/inhabilitación de handshaking de software]
PUSH [habilitación/inhabilitación de handshaking de hardware]
CALL 30

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM CALL 30 INPUT PARAMETERS:
>20  REM FIRST PUSH : 5, 6, 7, OR 8 (BITS/CHARACTER)
>30  REM SECOND PUSH : 0, 1, OR 2 (NO PARITY, ODD, OR
     EVEN)
>40  REM THIRD PUSH : 1, 2, OR 3 (1, 2, OR 1.5 STOP BITS)
>50  REM FOURTH PUSH: 0 OR 1 (SOFTWARE HANDSHAKING
     DISABLE, ENABLED)
>60  REM FIFTH PUSH : 0 OR 1 (HARDWARE HANDSHAKING
     DISABLED, ENABLED)
>70  REM PRT2 DEFAULT CONFIGURATION IS:
>80  REM 1200 BAUD, 8 BITS/CHAR, NO PARITY, 1 STOP BIT,
     AND
>90  REM SOFTWARE HANDSHAKING ENABLED
>100 PUSH 8 0,1,1,0 : CALL 30
>110 CALL 31

19200 Baud
Hardware Handshaking OFF
1 Stop Bit(s)
No Parity
8 Bits/Char
Xon/Xoff
>
```

CALL 78 - Establecimiento de velocidad en baudios del puerto de programación

Objetivo:

Use CALL 78 para cambiar la velocidad en baudios del puerto de programación de su valor predeterminado (1200 baudios) a uno de los siguientes: 300, 600, 1200, 2400, 4800, 9600 ó 19200 baudios. La velocidad en baudios predeterminada para el puerto de programación es 1200 baudios si el puerto PRT1 está configurado como puerto de programación, o 19200 baudios si el puerto DH485 está configurado como puerto de programación. Coloque (PUSH) la velocidad en baudios deseada y el CALL 78. El puerto de programación permanece en esta velocidad en baudios a menos que se invoque CALL 73 o se presenten las siguientes condiciones:

- La batería se ha agotado o ha sido retirada.
- El capacitor de la batería de respaldo está descargado.
- La EEPROM ha sido retirada o no está programada.
- Se ha desconectado y vuelto a conectar la alimentación eléctrica.

Si esto sucede, la velocidad en baudios cambia al valor predeterminado de 1200 baudios.

Sintaxis:

PUSH [velocidad en baudios]
CALL 78

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 PUSH 4800
>20 CALL 78
```

CALL 99 - Restablecimiento del señalador del cabezal impresor

Objetivo:

Use CALL 99 para restablecer el contador de caracteres internos del cabezal impresor de su impresora cuando se imprimen formularios anchos. Este CALL evita el CR/LF en el carácter 79. Usted debe controlar los caracteres en cada línea.

Sintaxis:

CALL 99

Ejemplo:

```
>10 REM EXAMPLE PROGRAM
>20 REM THIS PRINTS TIME BEYOND 80TH COLUMN
>30 PRINT TAB(79)
>40 CALL 99
>50 PRINT TAB(41), “TIME -”,
>60 PRINT H,“:”,M,“:”,S
>70 END
```

CALL 105 - Restablecimiento del PRT1 a sus valores predeterminados

Objetivo:

Use CALL 105 para restablecer los parámetros del puerto PRT1 a sus valores predeterminados. La Tabla 14.C lista los parámetros predeterminados para el puerto PRT1.

Tabla 14.C
Valores predeterminados de los parámetros del puerto PRT1

Parámetros del puerto PRT1	Valor predeterminado
Velocidad en baudios	1200 baudios
Número de bits de datos	8 bits
Número de bits de parada	1 bit
Paridad	Sin paridad
Handshaking	Handshaking de software

Sintaxis:

CALL 105

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 105
```

CALL 119 - Restablecimiento del PRT2 a sus valores predeterminados

Objetivo:

Use CALL 119 para restablecer los parámetros del puerto PRT2 a sus valores predeterminados. La Tabla 14.D lista las selecciones de parámetros para el puerto PRT2.

Tabla 14.D
Valores predeterminados de los parámetros del puerto PRT2

Parámetros del puerto PRT2	Valor predeterminado
Veloicidad en baudios	1200 baudios
Número de bits de datos	8 bits
Número de bits de parada	1 bit
Paridad	Sin paridad
Handshaking	Handshaking de software

Sintaxis:

CALL 119

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 CALL 119
```

MODE

Objetivo:

Use el comando MODE para establecer los parámetros de los puertos PRT1, PRT2 y DH485.

Importante: Usted debe asegurarse de que haya espacio de búfer disponible cada vez que imprima datos del puerto en serie usando handshaking de hardware o handshaking de software (Xon/Xoff). El incumplimiento de esta indicación hace que el programa BASIC detenga la ejecución mientras espera espacio de búfer. Cuando hay espacio disponible en el búfer, el módulo BASIC continúa la ejecución desde el punto en donde ésta fue interrumpida. El búfer de salida de cada puerto puede contener 256 caracteres. Para obtener más información, vea las descripciones de CALL 36, 37, 95 y 96.

El módulo BASIC aplica las siguientes reglas cuando el handshaking de hardware está habilitado. El módulo BASIC:

- no transmite hasta que CTS se activa
- examina DSR después de la recepción de un carácter. Si DSR está activo, el carácter se coloca en la cola de entradas. Si DSR está inactivo, se supone que el carácter es ruido y se descarta

La Tabla 14.E lista los parámetros de puerto para los puertos PRT1 o PRT2.

Tabla 14.E
Parámetros de puertos PRT1 y PRT2

Parámetros de puerto	Selecciones	Valores predeterminados
Velocidad en baudios	300, 600, 1200, 2400, 4800, 9600, 19200	1200
arg1 (paridad)	Ninguna (N), Par (E), Impar (O)	N
arg2 (número de bits de datos)	7 ó 8	8
arg3 (número de bits de parada)	1 ó 2	1
arg4 (handshaking)	Sin handshaking (N) Handshaking de software(S) Handshaking de hardware (H) Handshaking de hardware y software (B)	S
arg5 (tipo de almacenamiento)	Almacena la información en la ROM Y RAM de usuario (E). Almacena la información en la RAM con batería de respaldo (R).	R

Importante: Si algún argumento (excepto nombre de puerto y velocidad en baudios) se deja en blanco, el argumento cambia de manera predeterminada al valor especificado previamente para ese argumento.

La Tabla 14.F lista los parámetros de puerto para el puerto DH485.

Tabla 14.F
Parámetros del puerto DH485

Parámetros de puerto	Selecciones	Valores predeterminados
Velocidad en baudios	300, 600, 1200, 2400, 4800, 9600, 19200	19200 baudios
arg1 (dirección de nodo principal)	0 a 31	0
arg2 (dirección de nodo de módulo)	1 a 31	1
arg3 (dirección máxima de nodo)	1 a 31	31
arg4 (no se usa)		
arg5 (tipo de almacenamiento)	Almacena la información en la ROM Y RAM de usuario (E). Almacena la información en la RAM con batería de respaldo (R).	R

Importante: La opción de tipo de almacenamiento E no puede usarse si MODE se usa como instrucción.

Sintaxis:

MODE (nombre de puerto, velocidad en baudios, arg1, arg2, arg3, arg4, arg5)

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 MODE(DH485,19200,0,1,2,,R)
>.
.
>25 MODE(PRT1,1200,N,8,,,)
```

Capítulo 15

Funciones de cadena

Este capítulo describe e ilustra comandos usados para manipular estructuras de datos de cadena dentro del programa BASIC o desde la línea de comando. La Tabla 15.A lista los mnemónicos correspondientes.

Tabla 15.A
Guía de referencia del capítulo

Si necesita	Use este mnemónico	Página
Repetición de cadena.	CALL 60	15-1
Apéndice de cadena (concatenación).	CALL 61	15-2
Conversión de número a cadena.	CALL 62	15-4
Conversión de cadena a número.	CALL 63	15-5
Encontrar una cadena en una cadena.	CALL 64	15-6
Sustituir una cadena en una cadena.	CALL 65	15-7
Insertar una cadena en otra cadena.	CALL 66	15-8
Borrar una cadena de una cadena.	CALL 67	15-9
Determinar la longitud de una cadena.	CALL 68	15-10
Asignar memoria para cadenas.	STRING	15-10

CALL 60 - Repetición de cadena

Objetivo:

Use CALL 60 para repetir un carácter y colocarlo en una cadena. La repetición de cadena puede usarse al designar formatos de salida. Primero se coloca (PUSH) el número de veces que se va a repetir el carácter, luego se coloca (PUSH) el número de la cadena que contiene el carácter repetido. No se aplica POP a ningún argumento. El número de caracteres que se van a repetir no puede ser mayor que la longitud máxima de la cadena.

Sintaxis:

PUSH [número de veces que se va a repetir el carácter]
PUSH [número de cadena base]
CALL 60

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM STRING REPEAT EXAMPLE PROGRAM
>20 STRING 200,48
>30 $(1) = “*”
>40 PUSH 40 : REM THE NUMBER OF TIMES TO REPEAT CHARACTER
>50 PUSH 1 : REM BASE STRING NUMBER
>60 CALL 60
>70 PRINT $(1)
>80 END

READY
>RUN

**************************************

READY
>
```

CALL 61 - Apéndice de cadena

Objetivo:

Use CALL 61 para añadir una cadena al final de otra cadena. Este CALL necesita dos argumentos de cadena. El primero es el número de la cadena que se va a añadir y el segundo es el número de la cadena base. Si la cadena resultante es más larga que la longitud máxima de cadena, los caracteres añadidos se pierden. No hay argumentos de salida. Esto es una asignación de concatenación de cadena: (ejemplo: **`$(1)=$(1)+$(2)`**).

Importante: Si la nueva longitud de cadena excede la longitud asignada por el comando de cadena, se imprime un mensaje de error en el dispositivo de la consola y el módulo BASIC entra al modo de comando.

Sintaxis:

PUSH [número de la cadena que se va a añadir]
PUSH [número de cadena base]
CALL 61

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  STRING 200,20
>20  $(1) = “How are ”
>30  $(2) = “you?”
>40  PRINT “BEFORE ”
>50  PRINT “$(1) = ”,$(1)
>60  PRINT “$(2) = ”,$(2)
>70  PUSH 2 : REM STRING NUMBER TO BE APPENDED
>80  PUSH 1 : REM BASE STRING NUMBER
>90  CALL 61 : REM INVOKE STRING APPEND ROUTINE
>100 PRINT “AFTER:”
>110 PRINT “$(1) = ”,$(1)
>120 PRINT “$(2) = ”,$(2)
>130 END

READY
>RUN

BEFORE:
$(1) = How are
$(2) = you?
AFTER:
$(1) = How are you?
$(2) = you?

READY
>
```

CALL 62 - Conversión de número a cadena

Objetivo:

Use CALL 62 para convertir un número o variable numérica a una cadena. Debe asignar un mínimo de 14 caracteres para la cadena. Si se sabe de antemano que el exponente del valor que se va a convertir es mayor o igual a 100, hay que asignar 15 caracteres. La verificación de errores captura solamente asignaciones de cadena de menos de 14 caracteres. No hay argumentos de salida.

Sintaxis:

PUSH [número que se va a convertir a cadena]
PUSH [número de cadena que va a recibir el valor]
CALL 62

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,14
>20 INPUT “ENTER A NUMBER TO CONVERT TO A STRING ”,N
>30 PUSH N : REM NUMBER TO CONVERT TO STRING
>40 PUSH 1 : REM CONVERT NUMBER TO STRING 1
>50 CALL 62 : REM DO THE CONVERSION
>60 PRINT $(1)
>70 END

READY
>RUN

ENTER A NUMBER TO CONVERT TO A STRING 2E3
2000

READY
>RUN

ENTER A NUMBER TO CONVERT TO A STRING 1.233
1.233

READY
>
```

CALL 63 - Conversión de cadena a número

Objetivo:

Use CALL 63 para convertir el primer número decimal encontrado en la cadena especificada a un número en la pila de argumentos. Los números válidos y caracteres asociados son 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., E, +, –. La coma no es un carácter válido de número y finaliza la conversión.

Si la cadena no contiene un valor legal, aparece un cero. Los valores válidos están entre 1 y 255. Aplique PUSH al número de la cadena que se va a convertir. Se necesitan dos POP. Primero se aplica POP a la validez del valor, luego se aplica POP al valor real. Si una cadena contiene un número seguido de una E y una letra o carácter no numérico, se supone que no se encontró ningún número, puesto que la letra no es un exponente válido. (CALL 63 devuelve un cero en el primer argumento POP, indicando que no había ningún número válido en la cadena).

Sintaxis:

PUSH [número de cadena que se va a convertir]
CALL 63
POP [validez del valor]
POP [valor real]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>20 INPUT “INPUT A STRING TO CONVERT ”,$(1)
>30 PUSH 1 : REM CONVERT STRING 1
>40 CALL 63
>50 POP V,N
>60 IF (V<>0) THEN PRINT $(1),“ ” N:GOTO 80
>70 PRINT “INVALID OR NO VALUE FOUND”
>80 END

READY
>RUN

INPUT A STRING TO CONVERT 123ABC
123ABC  123

READY
>RUN

INPUT A STRING TO CONVERT 1.2E–7
1.2E–7  1.2 E–7

READY
>RUN

INPUT A STRING TO CONVERT 1.3.6
INVALID OR NO VALUE FOUND

READY
>
```

CALL 64 - Encontrar una cadena en una cadena

Objetivo:

Use CALL 64 para encontrar una cadena dentro de una cadena. Este CALL localiza la primera ocurrencia (posición) de esta cadena. Este CALL tiene dos argumentos de entrada. El primero es la cadena que se desea encontrar, el segundo es la cadena en donde que se va a buscar el equivalente. Se necesita un argumento de salida. Si el número no es cero, se ha localizado un equivalente en la posición indicada por el valor del argumento de salida. Esta rutina es similar a INSTR$(findstr$,str$) BASIC (ejemplo: **`L=INSTR$($(1),$(2)`**).

Sintaxis:

PUSH [número de cadena que se desea encontrar]
PUSH [número de cadena base]
CALL 64
POP [posición del equivalente]

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM SAMPLE FIND STRING IN STRING ROUTINE
>20  STRING 100,20
>30  $(1) = “456”
>40  $(2) = “12345678”
>50  PUSH 1 : REM STRING NUMBER OF STRING TO BE FOUND
>60  PUSH 2 : REM BASE STRING NUMBER
>70  CALL 64 : REM GET THE LOCATION OF FIRST CHARACTER
>80  POP L
>90  IF (L=0) THEN PRINT “NOT FOUND”
>100 IF(L>0) THEN PRINT “FOUND AT LOCATION”,L
>110 END

READY
>RUN

FOUND AT LOCATION 4

READY
>
```

CALL 65 - Sustitución de una cadena en una cadena

Objetivo:

Use CALL 65 para reemplazar una cadena dentro de una cadena. El primer argumento es el número de la cadena que reemplaza a la cadena identificada por el número de cadena del segundo argumento. El tercer argumento es el número de cadena base. No hay argumentos de salida.

Importante: Si la nueva longitud de cadena excede la longitud asignada por el comando de cadena, se imprime un mensaje de error en el dispositivo de la consola y el módulo BASIC entra al modo de comando.

Sintaxis:

PUSH [número de cadena nueva]
PUSH [número de cadena antiguo que se va a reemplazar]
PUSH [número de cadena base]
CALL 65

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM SAMPLE OF REPLACE STRING IN STRING
>20  STRING 100,20
>30  $(0) = “RED–LINES”
>40  $(1) = “RED”
>50  $(2) = “BLUE”
>60  PRINT “BEFORE:”
>70  PRINT “$(0) = ”,$(0)
>80  PUSH 2 : REM STRING NUMBER OF THE STRING
     TO BE REPLACED WITH
>90  PUSH 1 : REM STRING NUMBER OF THE STRING TO BE
     REPLACED
>100 PUSH 0 : REM BASE STRING NUMBER
>110 CALL 65 : REM INVOKE REPLACE STRING IN STRING
>120 PRINT “AFTER:”
>130 PRINT “$(0) = ”,$(0)
>140 END

READY
>RUN

BEFORE:
$(0) = RED–LINES
AFTER:
$(0) = BLUE–LINES

READY
>
```

CALL 66 - Inserción de una cadena en otra cadena

Objetivo:

Use CALL 66 para insertar una cadena dentro de otra cadena. El CALL necesita tres argumentos. El primer argumento es la posición en la que se debe comenzar la inserción. El segundo argumento es el número de cadena de los caracteres insertados en la cadena base. El tercer argumento es el número de la cadena base. Esta rutina no tiene argumentos de salida.

Importante: Si la nueva longitud de cadena excede la longitud asignada por el comando de cadena, se imprime un mensaje de error en el dispositivo de la consola y el módulo BASIC entra al modo de comando.

Sintaxis:

PUSH [posición de la inserción]
PUSH [número de cadena de carácter insertado]
PUSH [número de cadena base]
CALL 66

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM SAMPLE ROUTINE TO INSERT A STRING IN A STRING
>20  STRING 100,15
>30  $(0) = “1234590”
>40  $(1) = “67890”
>50  PRINT “BEFORE:”
>60  PRINT “$(0) = ”,$(0)
>70  PUSH 6 : REM POSITION TO START THE INSERT
>80  PUSH 1 : REM STRING NUMBER TO BE INSERTED
>85  PUSH 0 : REM BASE STRING NUMBER
>90  CALL 66 : REM INVOKE INSERT A STRING IN A STRING
>100 PRINT “$(0) = ”, (0)
>110 END

READY
>RUN

BEFORE:
$(0) = 1234590
$(0) = 123456789090

READY
>
```

CALL 67 - Borrado de una cadena en una cadena

Objetivo:

Use CALL 67 para borrar una cadena dentro de otra cadena. El CALL necesita dos argumntos. El primer argumento es el número de cadena base. El segundo es el número de la cadena que se va a borrar de la cadena base. Esta rutina no tiene argumentos de salida.

Importante: Esta rutina borra sólo la primera ocurrencia de la cadena.

Sintaxis:

PUSH [número de cadena base]
PUSH [número de cadena borrada]
CALL 67

Ejemplo:

```
>1   REM EXAMPLE PROGRAM
>10  REM ROUTINE TO DELETE A STRING IN A STRING
>20  STRING 200,14
>30  $(1) = “123456789012”
>40  $(2) = “12”
>50  PRINT “BEFORE:”
>60  PRINT “$(1) = ”,$(1)
>70  PUSH 1 : REM BASE STRING NUMBER
>80  PUSH 2 : REM STRING NUMBER OF THE STRING DELETED
>90  CALL 67 : REM INVOKE STRING DELETE ROUTINE
>100 PRINT “AFTER:”
>110 PRINT “$(1) = ”,$(1)
>120 END

READY
>RUN

BEFORE:
$(1) = 123456789012
AFTER:
$(1) = 3456789012

READY
>
```

CALL 68 - Determinación de la longitud de una cadena

Objetivo:

Use CALL 68 para determinar la longitud de una cadena. Se necesita un argumento de entrada. Este es el número de cadena sobre el que actúa la rutina. Se necesita un argumento de salida. Este es el número de los caracteres sin retorno de carro (CR) en esta cadena. Es semejante al comando BASIC LEN(str$) (ejemplo: **`L=LEN($l)`**). La longitud de la cadena puede determinarse correctamente sólo si la cadena termina con un carácter CR. Si se llena una cadena usando la instrucción ASC, debe añadirse un CR como último carácter para terminar la cadena.

Sintaxis:

PUSH [número de cadena]
CALL 68
POP [número de caracteres]

Ejemplo:

```
>1  REM EXAMPLE PROGRAM
>10 REM SAMPLE OF STRING LENGTH
>20 STRING 1 0,10
>30 $(1) = “1234567”
>40 PUSH 1 : REM BASE STRING
>50 CALL 68 : REM INVOKE STRING LENGTH ROUTINE
>60 POP L : REM GET LENGTH OF BASE STRING
>70 PRINT “THE LENGTH OF ”,$(1),“ IS”,L
>80 END

READY
>RUN

THE LENGTH OF 1234567 IS 7

READY
>
```

STRING

Objetivo:

Use la instrucción STRING para asignar memoria a cadenas. Al principio no hay memoria asignada a cadenas. Si se intenta definir una cadena con una instrucción tal como LET $(1)=HELLO antes de haber asignado memoria para cadenas, se genera el mensaje **`ERROR: MEMORY ALLOCATION`**. La primera expresión ([expr]) en la instrucción STRING es el número total de bytes que usted desea asignar para almacenamiento de cadenas. La segunda expresión ([expr]) da el máximo número de bytes en cada cadena. El segundo valor no debe ser mayor que 254. Estos dos números determinan el número total de variables de cadena definidas.

El módulo BASIC requiere un byte adicional para cada cadena, más un byte adicional en total. El carácter adicional para cada cadena se asigna para el carácter de retorno de carro que termina la cadena. Esto significa que la instrucción STRING 100,10 asigna suficiente memoria para 9 variables de cadena, en un rango de $(0) a $(8) y que los 100 bytes asignados se usan. Nótese que $(0) es una cadena válida en el módulo BASIC.

Importante: Si se usa un carácter ASCII nulo dentro de la cadena, éste actúa como indicador de fin de cadena.

Importante: Una vez que se ha asignado memoria para almacenamiento de cadenas, ni los comandos (ejemplo: **NEW**) ni las instrucciones (ejemplo: **CLEAR**) pueden desasignar esta memoria. El desconectar y volver a conectar la alimentación eléctrica tampoco puede desasignar esta memoria a menos que la batería de respaldo esté inhabilitada. Se puede desasignar memoria ejecutando una instrucción STRING 0,0. STRING 0,0 no asigna memoria a las variables de cadenas.

Importante: El módulo BASIC ejecuta el equivalente de una instrucción CLEAR cada vez que se ejecuta la instrucción STRING [expr],[expr]. Esto es necesario porque las variables de cadena y las variables numéricas ocupan el mismo espacio de memoria externa. Después que se ejecuta la instrucción STRING, todas las variables se borran. Debido a esto, se debe realizar la asignación de memoria para cadenas al principio del programa (durante la primera instrucción si fuera posible). Si se reasigna memoria de cadenas, se destruirán todas las variables definidas.

Sintaxis:

STRING [expr], [expr]

Ejemplos:

```
>1  REM EXAMPLE PROGRAM
>10 STRING 100,30
>20 $(0) = “-----MONTHLY REPORT-----”
>30 PRINT $(0)

READY
>RUN

-----MONTHLY REPORT-----

READY
>
```

Apéndice **A**

Tabla de conversiones ASCII decimal/hexadecimal/octal

Descripción general de la conversión matemática

La siguiente tabla lista las conversiones decimal, hexadecimal, octal y ASCII.

Tabla A.1
Tabla de conversiones

Columna 1				Columna 2				Columna 3				Columna 4			
DEC	HEX	OCT	ASC	DEC	HEX	OCT	ASC	DEC	HEX	OCT	ASC	DEC	HEX	OCT	ASC
00	00	000	NUL	32	20	040	SP	64	40	100	@	96	60	140	`
01	01	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
02	02	002	STX	34	22	042	”	66	42	102	B	98	62	142	b
03	03	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
04	04	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
05	05	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
06	06	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
07	07	007	BEL	39	27	047	’	71	47	107	G	103	67	147	g
08	08	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
09	09	011	HT	41	29	051	)	73	49	111	I	105	69	151	i
10	0A	012	LF	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	0B	013	VT	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	0C	014	FF	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	0D	015	CR	45	2D	055	–	77	4D	115	M	109	6D	155	m
14	0E	016	SO	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	0F	017	SI	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC	59	3B	073	;	91	5B	133	[	123	7B	173	{
28	1C	034	FS	60	3C	074	<	92	5C	134	\	124	7C	174	.
29	1D	035	GS	61	3D	075	=	93	5D	135	]	125	7D	175	}
30	1E	036	RS	62	3E	076	>	94	5E	135	^	126	7E	176	~
31	1F	037	US	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

Apéndice A

Apéndice **B**

Guía de referencia rápida de comandos, instrucciones y CALLS BASIC

Descripción general de lista de mnemónicos

La siguiente tabla lista los diversos mnemónicos encontrados en este manual junto con una descripción y ubicación.

Tabla B.1
Guía de referencia rápida

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
ABS( )		Valor absoluto		3-11
ASC( )		Valor entero de retorno de carácter ASCII.		3-13
ATN( )		Arcotangente de retorno del argumento.		3-10
BRKPNT		Define punto de interrupción del programa.	*	4-2
CALL 14	PUSH [número de palabra de búfer de entrada de módulo BASIC] POP [valor convertido]	Entero con signo de 16 bits a punto (coma) flotante BASIC		9-1
CALL 15	PUSH [número de palabra de búfer de entrada de módulo BASIC] POP [valor convertido]	Entero sin signo de 16 bits a punto (coma) flotante BASIC		9-2
CALL 16	PUSH [número de línea BASIC]	Habilita la capacidad de interrupción cuando se recibe un paquete DF1.		8-2
CALL 17	Ninguno	Inhabilita la capacidad de interrupción del paquete DF1.		8-2
CALL 18	Ninguno	Vuelve a habilitar la función de interrupción **[CTRL–C]**.		4-6
CALL 19	Ninguno	Inhabilita la función de interrupción **[CTRL–C]**.		4-6
CALL 20	PUSH [número de línea BASIC]	Habilita la capacidad de interrupción del procesador SLC.		8-3
CALL 21	Ninguno	Inhabilita la capacidad de interrupción del procesador SLC.		8-4
CALL 22	PUSH [número de puerto fuente] PUSH [número máximo de caracteres que se van a transferir] PUSH [valor decimal de delimitador de caracteres] PUSH [selección de archivo y/o cadena de destino] PUSH [offset de palabra dentro del archivo de destino] PUSH [número de cadena] PUSH [selección de intercambio de bytes] POP [estado de CALL 22]	Transfiere datos desde el PRT1 o PRT2 al sistema de E/S SLC o a los archivos M.		13-2

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
CALL 23	PUSH [número de puerto de destino y/o cadena interna] PUSH [selección de archivo fuente] PUSH [offset de palabra dentro de archivo fuente] PUSH [número de cadena] PUSH [selección de intercambio de bytes] POP [estado de CALL 23]	Transfiere datos desde los archivos M o E/S SLC al PRT1 o PRT2.		12-2
CALL 24	PUSH [valor que se va a convertir] PUSH [número de palabra de búfer de salida de módulo BASIC]	Punto (coma) flotante BASIC a entero con signo de 16 bits.		9-3
CALL 25	PUSH [valor que se va a convertir] PUSH [número de palabra de búfer de salida de módulo BASIC]	Punto (coma) flotante BASIC a binario de 16 bits.		9-3
CALL 26	POP [estado del procesador SLC]	Genera una interrupción al procesador SLC.		8-5
CALL 27	PUSH [tipo de comando READ] PUSH [dirección de nodo remoto] PUSH [número de archivo remoto] PUSH [tipo de archivo remoto] PUSH [offset de palabra inicial de archivo remoto] PUSH [número de palabras que se van a transferir] PUSH [valor de tiempo límite de mensaje] PUSH [selección de archivo de destino] PUSH [offset de palabra dentro de archivo de destino] PUSH [número de cadena] POP [estado del CALL 27]	Transfiere datos desde un archivo de datos DH–485 remoto al procesador SLC.		13-9
CALL 28	PUSH [tipo de comando WRITE] PUSH [dirección de nodo remoto] PUSH [número de archivo remoto] PUSH [tipo de archivo remoto] PUSH [offset de palabra inicial remota PUSH [número de palabras que se van a transferir] PUSH [valor de tiempo límite de mensaje] PUSH [selección de archivo fuente] PUSH [offset de palabra dentro de archivo fuente] PUSH [número de cadena] POP [estado del CALL 28]	Transfiere datos desde el procesador SLC a un archivo de datos DH–485 remoto.		12-9
CALL 29	PUSH [CALL 27, 28, 122 ó 123 for the CALL para el CALL que desea activar] POP [estado de la transacción]	Maneja todos los errores que no son manejados por la instrucción ONERR.		12-16, 13-17
CALL 30	PUSH [bits por palabra] PUSH [habilitación de paridad] PUSH [número de bits de parada] PUSH [habilitación/inhabilitación de handshaking de software] PUSH [habilitación/inhabilitación de handshaking de hardware	Establece los parámetros del puerto PRT2.		14-1
CALL 31	Ninguno	Muestra la configuración actual del puerto PRT2.		12-17
CALL 35	POP [valor ASCII de carácter]	Obtiene caracteres de entrada numéricos desde el puerto PRT2.		13-19

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
CALL 36	PUSH [selección de búfer] POP [número de caracteres]	Obtiene el número de caracteres en los búfers PRT2.		11-2
CALL 37	PUSH [selección de búfer]	Borra los búfers de entradas y salidas del puerto PRT2.		12-18
CALL 38	PUSH [0 ó 1]	Inicia las transacciones definidas por CALL 27, 28, 122 y 123.		8-6
CALL 40	PUSH [horas] PUSH [minutos] PUSH [segundos]	Fija la hora del reloj/calendario (hora, minuto, segundo).		10-2
CALL 41	PUSH [fecha] PUSH [mes] PUSH [año]	Fija la fecha del reloj/calendario (día, mes, año).		10-3
CALL 42	PUSH [día de la semana]	Fija el reloj calendario – día de la semana.		10-4
CALL 43	PUSH [número de cadena]	Recupera la cadena de fecha/hora.		10-4
CALL 44	POP [día] POP [mes] POP [año]	Recupera la fecha numérica (día, mes, año).		10-5
CALL 45	PUSH [número de cadena]	Recupera la cadena de hora.		10-5
CALL 46	POP [hora] POP [minuto] POP [segundo]	Recupera la hora numérica.		10-6
CALL 47	PUSH [número de cadena]	Recupera la cadena de día de la semana.		10-7
CALL 48	POP [día de la semana]	Recupera el día de la semana numérico.		10-7
CALL 51	POP [estado del búfer de imagen de salida]	Verifica el búfer de imagen de salida CPU.		11-3
CALL 52	PUSH [número de cadena]	Recupera la cadena de fecha.		10-8
CALL 53	POP [estado del procesador]	Transfiere el búfer de imagen de salida CPU al búfer de entrada del módulo BASIC.		13-21
CALL 54	POP [modo del procesador]	Transfiere el búfer de salida del módulo BASIC al búfer de imagen de entrada CPU.		12-18
CALL 55	POP [estado del búfer de imagen de entrada]	Verifica el búfer de imagen de entrada CPU.		11-4
CALL 56	PUSH [número de palabras que se van a transferir] POP [estado del procesador]	Transfiere el archivo M0 CPU al búfer de entrada del módulo BASIC.		13-22
CALL 57	PUSH [número de palabras que se van a transferir] POP [modo del procesador]	Transfiere el búfer de salida del módulo BASIC al archivo M1 CPU.		12-19
CALL 58	POP [estado de escritura del archivo M0 del módulo]	Verifica el estado del archivo M0.		11-5
CALL 59	POP [estado de lectura del archivo M1 del módulo]	Verifica el estado del archivo M1.		11-6
CALL 60	PUSH [número de veces que se va a repetir el carácter] PUSH [número de cadena base]	Repite cadena		15-1

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
CALL 61	PUSH [número de cadena que se va a añadir] PUSH [número de cadena base]	Agrega cadena (concatenación)		15-2
CALL 62	PUSH [número que se va a convertir a cadena] PUSH [número de cadena que va a recibir el valor]	Conversión de número a cadena		15-4
CALL 63	PUSH [[número de cadena que se va a convertir] POP [validez del valor] POP [valor real]	Conversión de cadena a número		15-5
CALL 64	PUSH [número de cadena que se va a encontrar] PUSH [número de cadena base] POP [posición del equivalente]	Encuentra una cadena en una cadena.		15-6
CALL 65	PUSH [Número de cadena nueva] PUSH [número de cadena antigua que se va a reemplazar] PUSH [número de cadena base]	Reemplaza una cadena en una cadena.		15-7
CALL 66	PUSH [posición de inserción] PUSH [número de cadena de carácter insertado] PUSH [número de cadena base]	Inserta una cadena en una cadena.		15-8
CALL 67	PUSH [número de cadena base] PUSH [número de cadena borrada]	Borra una cadena de una cadena.		15-9
CALL 68	PUSH [número de cadenar] POP [número de caracteres]	Determina la longitud de una cadena.		15-10
CALL 70	Ninguno	Transfiere programa de ROM a RAM.		8-8
CALL 71	PUSH [número de programa ROM]	Transfiere programa de ROM/RAM a ROM.		8-9
CALL 72	Ninguno	Regresa a RAM/ROM		8-10
CALL 73	Ninguno	Inhabilita RAM con batería de respaldo.		5-2
CALL 74	Ninguno	Habilita RAM con batería de respaldo.		5-2
CALL 75	POP [modo del procesador]	Verifica el estado de CPU del controlador SLC 500.		11-7
CALL 77	PUSH [Dirección MTOP nueva]	Almacenamiento de variables protegido.		5-3
CALL 78	PUSH [velocidad en baudios]	Etablece velocidad en baudios del puerto de programación.		14-3
CALL 80	POP [estado de la batería]	Revisa el estado de la batería.		11-8
CALL 81	Ninguno	Verifica y describe el módulo de memoria de usuario.	*	5-4
CALL 82	Ninguno	Verifica el mapa del módulo de memoria de usuario.	*	5-5
CALL 84	PUSH [offset de palabra inicial en archivo de interface DH–485] PUSH [número de palabras que se van a transferir] POP [estado de la transferencia]	Transfiere el archivo de interface DH–485 al búfer de entrada del módulo BASIC.		13-23
CALL 85	PUSH [offset de palabra inicial en archivo de interface DH–485] PUSH [número de palabras que se van a transferir] POP [estado de la transferencia]	Transfiere del búfer de salida del módulo BASIC al archivo de interface DH–485.		12-20

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
CALL 86	POP [estado de escritura remota de archivo de interface DH–485]	Verifica estado de escritura remota de archivo de interface DH–485.		11-8
CALL 87	POP [estado de lectura remota de archivo de interface DH–485]	Verifica estado de lectura remota de archivo de interface DH–485		11-9
CALL 90	PUSH [dirección de nodo de dispositivo remoto] PUSH [número de archivo de dispositivo remoto] PUSH [tipo de archivo de dispositivo remoto] PUSH [offset de elemento inicial (x2) de archivo de dispositivo remoto] PUSH [número de elementos que se van a transferir] PUSH [valor de tiempo límite de mensaje] POP [estado de instrucción de mensaje]	Lee archivo de datos DH–485 remoto al búfer de entradas BASIC.		13-24
CALL 91	PUSH [dirección de nodo de dispositivo remoto] PUSH [número de archivo de dispositivo remoto] PUSH [tipo de archivo de dispositivo remoto] PUSH [offset de elemento inicial (x2) de archivo de dispositivo remoto] PUSH [número de elementos que se van a transferir] PUSH [valor de tiempo límite de mensaje] POP [estado de instrucción de mensaje]	Escribe búfer de salida del módulo BASIC a archivo de datos DH–485 remoto.		12-21
CALL 92	PUSH [dirección de nodo de dispositivo remoto] PUSH [offset de elemento inicial (x2) de archivo de dispositivo remoto] PUSH [número de palabras que se van a transferir] PUSH [valor de tiempo límite de mensaje] POP [estado de instrucción de mensaje]	Lee archivo de interface DH–485 remoto al búfer de entradas del módulo BASIC.		13-27
CALL 93	PUSH [dirección de nodo de dispositivo remoto] PUSH [offset de elemento inicial (x2) de archivo de dispostiivo remoto] PUSH [número de palabras que se van a transferir] PUSH [valor de tiempo límite de mensaje] POP [estado de instrucción de mensaje]	Escribe búfer de salidas del módulo BASIC al archivo de interface DH–485 remoto.		12-25
CALL 94	Ninguno	Imprime la configuración actual del puerto PRT1.		12-27
CALL 95	PUSH [selección de búfer] POP [número de caracteres]	Obtiene el número de caracteres en los búfers PRT1.		11-10
CALL 96	PUSH [selección de búfer]	Borra los búfers de entradas y salidas del puerto PRT1.		12-28
CALL 97	Ninguno	Habilita la señal DTR del puerto PRT2.		11-11
CALL 98	Ninguno	Inhabilita la señal DTR del puerto PRT2.		11-11
CALL 99	Ninguno	Restablece el señalador de cabezal impresor.		14-4
CALL 101	PUSH [dirección inicial] PUSH [dirección final]	Carga el código del módulo de memoria de usuario al dispositivo princpal.		5-5
CALL 103	Ninguno	Imprime el búfer de salidas del puerto PRT1 y señalador.		5-6
CALL 104	Ninguno	Imprime el búfer de entradas del puerto PRT1 y señalador.		5-7
CALL 105	Ninguno	Restablece el puerto PRT1 a sus valores predeterminados.		14-4

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
CALL 108	PUSH [código de operación] PUSH [tiempo límite de encuesta (poll) o tiempo límite de ACK] PUSH [reintentos de mensaje o reintentos de ENQ] PUSH [retardo conexión RTS o NAK reintentos recibidos] PUSH [retardo desconexión RTS o valor NULL] PUSH [dirección DF1 de módulo BASIC]	Habilita comunicaciones del controlador DF1.		11-12
CALL 109	Ninguno	Imprime la pila de argumentos.		5-8
CALL 110	Ninguno	Imprime el búfer de salidas y señalador del puerto PRT2.		5-9
CALL 111	Ninguno	Imprime el búfer de entradas y señalador del puerto PRT2.		5-10
CALL 112	PUSH [estado de LED1] PUSH [estado de LED2]	Control de LED del usuario.		12-28
CALL 113	Ninguno	Inhabilita comunicaciones del controlador DF1.		11-19
CALL 114	Ninguno	Transmite paquete DF1.		12-29
CALL 115	POP [estado de transmisión DF1]	Verifica estado de1 XMIT DF.		12-30
CALL 117	POP [longitud de paquete DF1]	Obtiene longitud de paquete DF1.		13-29
CALL 118	PUSH [habilitación/inhabilitación de CALL] PUSH [selección de archivo y/o cadena de destino] PUSH [offset de palabra en archivo de destino] PUSH [número de cadena] PUSH [longitud máxima de palabra] POP [estado de CALL 118]	Permite escrituras no solicitadas desde un nodo SLC o PLC remoto.		13-30
CALL 119	Ninguno	Restablece el puerto PRT2 a sus valores predeterminados.		14-5
CALL 120	PUSH [equivalente decimal]	Borra los búfers de entradas y salidas del módulo BASIC.		11-19
CALL 121	POP [número de ID de programa]	Obtiene el número de ID del programa del procesador SLC.		11-20
CALL 122	PUSH [tipo de comando READ PLC] PUSH [dirección de nodo PLC remoto] PUSH [número de archivo de PLC remoto] PUSH [tipo de archivo en PLC remoto] PUSH [offset de elemento inicial en PLC remoto] PUSH [número de elementos que se van a transferir] PUSH [valor de tiempo límite de mensaje] PUSH [selección de archivo de destino] PUSH [offset de palabra dentro de archivo de destino] PUSH [número de cadena] POP [estado del CALL 122]	Lee un archivo de datos PLC.		13-36

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
CALL 123	PUSH [tipo de comando WRITE PLC] PUSH [dirección de nodo PLC remoto] PUSH [número de archivo de PLC remoto] PUSH [tipo de archivo en PLC remoto] PUSH [offset de palabra inicial en PLC remoto] PUSH [número de elementos que se van a transmitir] PUSH [valor de tiempo límite de mensaje] PUSH [selección de archivo fuente] PUSH [offset de palabra dentro de archivo fuente] PUSH [número de cadena] POP [Estado de CALL 123]	Escribe a archivo de datos PLC.		12-31
CBY( )		Recupera datos de dirección de memoria especificada.		3-18
CHR( )		Cuenta el valor convertido a caracteres ASCII.		3-16
CLEAR		Borra variables, interrupciones y cadenas.		6-1
CLEARI		Borra interrupciones.		6-2
CLEARS		Borra todas las pilas.		6-3
CLOCK0		Inhabilita el reloj en tiempo real.		7-2
CLOCK1		Habilita el reloj en tiempo real.		7-1
CONT		Continúa después de un Stop o **[CTRL–C]**.	*	4-4
CONTROL C		Detiene la ejecución y regresa al modo de comando.		4-5
CONTROL Q		Reinicia el listado después de **[CTRL–S]**.		4-8
CONTROL S		Interrumpe un comando de listado.		4-7
COS( )		Devuelve el coseno del argumento.		3-10
DATA		Datos leídos por la instrucción de lectura.		6-4
DBY( )		Recupera o asigna datos desde o hacia la memoria de datos internos del módulo BASIC.		3-19
DIM		Asigna memoria para variables de la matriz.		6-5
DO-UNTIL		Configura un lazo condicional DO.		7-5
DO-WHILE		Configura un lazo condicional DO.		7-3
EDIT		Edita una línea del programa BASIC.	*	4-9
END		Termina la ejecución del programa.		7-5
EOF		Efectúa una prueba para detectar búfer vacío de entradas.		3-17
ERASE		Borra el último programa BASIC almacenado en la ROM por un comando PROG.	*	4-10
EXP( )		“e” (2.7182818) A LA X		3-13

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
FOR-TO-(STEP) -NEXT		Configura un lazo for-next.		7-6
FREE		Efectúa una prueba para determinar el número de bytes libres de memoria RAM.		3-17
GET		Lee el dispositivo de entrada de la consola.		13-46
GET#		Lee el dispositivo de entrada de la consola conectado al PRT2.		13-46
GET@		Lee el dispositivo de entrada de la consola conectado al PRT1.		13-46
GOSUB		Ejecuta la subrutina.		8-11
GOTO		Va al número de línea del programa.		7-8
IDLE		Fuerza al módulo BASIC a entrar al modo “esperar hasta interrupción”.		4-10
IF–THEN–ELSE		Prueba condicional.		7-9
INPL		Lee línea de caracteres del búfer del puerto de programación.		13-47
INPL#		Lee línea de caracteres del búfer del puerto PRT2.		13-47
INPL@		Lee línea de caracteres del búfer del puerto PRT1.		13-47
INPS		Lee cadena de caracteres del búfer del puerto de programación.		13-47
INPS#		Lee cadena de caracteres del búfer del puerto PRT2.		13-47
INPS@		Lee cadena de caracteres del búfer del puerto PRT1.		13-47
INPUT		Entra una cadena o variable.		13-48
INPUT#		Entra una cadena o variable desde el puerto PRT2.		13-48
INPUT@		Entra una cadena o variable desde el puerto PRT1.		13-48
INT( )		Entero		3-11
LD@		Carga variable		13-50
LEN		Lee el número de bytes de memoria en el programa actualmente seleccionado.		3-18
LET		Asigna un valor a una variable o cadena (LET es opcional).		6-6
LIST		Lista el programa al dispositivo de la consola.	*	4-11
LIST#		Lista el programa a la impresora en serie.	*	4-12
LIST@		Lista el programa al dispositivo conectado al puerto PRT1.	*	4-12
LOG( )		Logaritmo natural		3-13
MODE		Establece los parámetros de los puertos PRT1, PRT2 y DH485.		4-13, 14-5

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
MTOP		Lee la última dirección de memoria válida.		3-18
NEW		Borra el programa almacenado en la RAM.	*	4-14
NEXT		Prueba la condición del lazo for-next.		7-10
NOT( )		Complemento a uno.		3-11
NULL		Establece la cuenta NULL después de un retorno de carro–avance de línea.	*	4-14
ONERR		Va al número de línea cuando se detecta un error.		8-12
ON-GOSUB		GOSUB condicional		8-13
ON-GOTO		GOTO condicional		7-11
ONTIME		Genera una interrupción cuando TIME es igual o mayor que el número de línea del argumento ONTIME.		8-14
PH0.		Imprime el valor hexadecimal con supresión de cero al dispositivo de la consola.		12-42
PH0.#		Imprime el valor hexadecimal con supresión de cero al PRT2.		12-42
PH0.@		Imprime el valor hexadecimal con supresión de cero al PRT1.		12-42
PH1.		Imprime el valor hexadecimal sin supresión de cero al dispositivo de la consola.		12-42
PH1.#		Imprime el valor hexadecimal sin supresión de cero al PRT2.		12-42
PH1.@		Imprime el valor hexadecimal sin supresión de cero al PRT1.		12-42
PI		PI-3.1415926		3-12
POP		Pila de argumentos POP a variables.		8-17
PRINT		Imprime variables, cadenas o literales al dispositivo de la consola. P. es la abreviatura de Print.		12-40
PRINT#		Imprime al puerto PRT2.		12-40
PRINT@		Imprime al puerto PRT1.		12-40
PRINT CR		Imprime retorno de carro.		12-41
PRINT SPC( )		Imprime espacios.		12-41
PRINT TAB( )		Imprime tabulaciones.		12-41
PRINT USING(Fx)		Imprime valores numéricos en notación científica.		12-41
PRINT USING(#.#)		Imprime valores numéricos en notación decimal.		12-42
PRINT USING(0)		Restaura el modo de impresión predeterminado.		12-42

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
PROG		Guarda el programa actual en la EPROM.	*	4-15
PROG1		Guarda la información de velocidad en baudios en la EPROM.	*	4-16
PROG2		Guarda la información de velocidad en baudios en la EPROM y ejecuta el programa después del restablecimiento.	*	4-17
PUSH		PUSH (coloca) expresiones en la pila de argumentos.		8-15
RAM		Invoca el modo RAM.	*	4-19
READ		LEER datos en la instrucción de datos.		13-52
REM		Especifica una línea de comentario.		4-19
REN		Vuelve a numerar el programa BASIC.		4-20
RESTORE		RESTAURA el punto de lectura.		6-7
RETI		Regresa de una interrupción.		8-18
RETURN		REGRESA de una subrutina.		8-18
RND		Número aleatorio.		3-12
ROM		Selecciona el modo ROM.	*	4-21
RROM		Selecciona el modo ROM y ejecuta el programa seleccionado.	*	4-22
RUN		Ejecuta un programa.	*	4-23
SGN		Signo		3-12
SIN( )		Devuelve el seno del argumento.		3-9
SNGLSTP		Inicia ejecución de programa de un solo paso.	*	4-24
SQR( )		Raíz cuadrada.		3-12
ST@		Almacena una variable.		12-43
STOP		Interrumpe la ejecución del programa.		8-20
STRING		Asigna memoria para cadenas.		15-10
TAN( )		Devuelve la tangente del argumento.		3-10
TIME		Recupera y/o asigna valor del reloj autónomo.		3-20
VER		Verifica la versión de firmware del módulo BASIC.		4-25
XBY( )		Recupera y/o asigna datos hacia o desde la memoria de datos externa del módulo BASIC.		3-19
XFER		Transfiere un programa de la ROM a la RAM.	*	4-26
+		Suma		3-5
/		División		3-5

Mnemónico	PUSH o POP requeridos	Descripción	Activo sólo en el modo de comando	Página
**		Exponenciación		3-5
*		Multiplicación		3-5
–		Resta		3-6
.AND.		Y lógico		3-8
.OR.		O lógico		3-8
.XOR.		O lógico exclusivo		3-8
@		Comunicaciones directas al puerto PRT1.		3-17
#		Comunicaciones directas al puerto PRT2.		3-17

Símbolos

A

B

C

D

P

R

S

T

V

X

Rockwell Automation

Rockwell Automation ayuda a sus clientes a lograr mejores ganancias de sus inversiones integrando marcas líder de la automatización industrial y creando así una amplia gama de productos de integración fácil. Estos productos disponen del soporte de proveedores de soluciones de sistema además de los recursos de tecnología avanzada de Rockwell.

Con oficinas en las principales ciudades del mundo.

Alemania • Arabia Saudita • Argentina • Australia • Bahrein • Bélgica • Bolivia • Brasil • Bulgaria • Canadá • Chile • Chipre • Colombia • Corea • Costa Rica • Croacia Dinamarca • Ecuador • Egipto • El Salvador • Emiratos Arabes Unidos • Eslovaquia • Eslovenia • España • Estados Unidos • Finlandia • Francia • Ghana • Grecia • Guatemala Holanda • Honduras • Hong Kong • Hungría • India • Indonesia • Irán • Irlanda • Islandia • Israel • Italia • Jamaica • Japón • Jordania • Katar • Kuwait • Las Filipinas • Líbano Macao • Malasia • Malta • México • Morruecos • Nigeria • Noruega • Nueva Zelandia • Omán • Pakistán • Panamá • Perú • Polonia • Portugal • Puerto Rico • Reino Unido República Checa • República de Sudáfrica • República Dominicana • República Popular China • Rumania • Rusia • Singapur • Suecia • Suiza • Taiwan • Tailandia • Trinidad Tunisia • Turquía • Uruguay • Venezuela

Sede central de Rockwell Automation: 1201 South Second Street, Milwaukee, WI 53204 USA, Tel: (1) 414–382–2000, Fax: (10) 414–382–4444
Sede central europea de Rockwell Automation: Avenue Herrmann Debrouxlaan, 46, 1160 Bruselas, Bélgica, Tel: (32) 2 663 06 00, Fax: (32) 2 663 06 40
Sede central de Asia–Pacífico de Rockwell Automation: 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publicación 1746-6.3ES – Noviembre de 1994

PN 40072-010-05(A)