LK8Ø OPERATOR'S GUIDE

PRELIMINARY

Compiler Systems, Inc.
P.O. Box 145
37 N. Auburn Avenue
Sierra Madre, CA 91Ø24

213-355-4211

First Printing: September, 1981

TABLE OF CONTENTS

1. INTRODUCTION TO LK8Ø

LK8Ø^{tm} is a linkage editor that combines relocatable object (REL) modules into an executable file and optional overlay files. LK8Ø is designed to be used with the CB8Ø^{tm} compiler. It can also link modules created by a relocatable assembler such as RMAC^{tm}. When LK8Ø is used with a language such as CB8Ø, it produces a composite program by combining the language's default library with the REL modules.

LK8Ø is designed to link any program that occupies less than 64K bytes of memory unless the length of symbols exhausts the space reserved for the symbol table.

This manual describes version 1 of LK8Ø which operates with Digital Research's CP/M-8Ø or MP/M-8Ø^{tm} operating systems. LK8Ø is provided with CB8Ø. The End User Licensing Agreement for CB8Ø covers the use of LK8Ø. Refer to the CB8Ø Licensing Guide for information concerning support for LK8Ø and the distribution policy for composite programs. If you do not have a Licensing Guide please contact Compiler Systems at (213) 355-4211.

The Licensing Guide explains how to use CK8Ø to verify that your copy of LK8Ø is correct and has not been altered during disk copying or due to hardware or software failure. It also contains copies of all applicable licensing agreements.

2. OPERATION OF LK8Ø

This chapter describes the operation of LK8Ø. The general form of a LK8Ø command line is shown below.

LK8Ø [<fn>=]<fn.ft>{,<fn.ft>} { ([<fn>=]<fn.ft>{,<fn.ft>}) }

The brackets ([]) denote optional portions of the command. The braces ({}) indicate that the enclosed section may be repeated zero or more times. The symbol "fn" indicates a file name without a type extension. The symbol "fn.tn" represents a file name with an optional type extension and optional command toggles.

Each option of the LK8Ø command line will be explained in detail in the remainder of this chapter.

2.1. Linking Modules

LK8Ø is executed by typing LK8Ø followed by the name of the module to link.

LK8Ø TEST

This command will link the REL module "TEST.REL" producing an executable file "TEST.COM". In addition a symbol location (SYM) file "TEST.SYM" will be produced. The SYM file may be used with Digital Research's symbolic debugging program SIDtm.

When linking CB8Ø programs, LK8Ø will automatically search the default disk for the CB8Ø runtime library "CB8Ø.IRL". Any library modules required by the program being linked will be combined into the executable module produced by LK8Ø. The combination of one or more REL files with a language library forms a composite program.

LK8Ø prints information on the display about the module being linked. The next example shows the results of linking a simple program. The CB8Ø program below, "TEST.BAS", can be compiled to produce a REL file "TEST.REL":

```
PRINT "THIS IS A TEST PROGRAM"
PRINT "IT IS USED TO DEMO LK8Ø"
STOP
```

The module "TEST.REL" is then linked using the following command.

```
LK8Ø TEST
```

The information that will be printed on the display by LK8Ø is shown below:

```
A>LK8Ø
-------------------------------------------------
LK8Ø Version 1.ØØ Serial No. 12345678 Copyright (c)
1981 Compiler Systems, Inc.  All rights reserved
-------------------------------------------------

code size:       1173 (Ø1ØØ-1273)
common size:     ØØØØ
data size:       Ø168 (128Ø-13E7)
symbol table space remaining:   ØA4C
```

The amount of memory allocated to code, common data, and local data is shown next. In this example there is no common data. All the values are hexadecimal numbers.

The amount of symbol table space remaining provides an indication of the number of additional symbols that could be added to the modules being linked without running out of symbol table space.

Normally LK8Ø produces a COM file with the same name as the REL file. For example linking the example above would result in an executable file "TEST.COM" being placed on the default drive.

```
LK8Ø PAYROLL=B:PAY
```

The command above links the module "PAY.REL" from drive B but creates an executable file "PAYROLL.COM" on the default drive.

```
LK8Ø B:PAYROLL = B:PAY
```

This command produces the same executable file as the previous example but the file "PAYROLL.COM" is placed on the B drive instead of the default drive.

The names of the modules being linked may have type extensions.

```
LK8Ø A.REL,B.C
```

Regardless of the type extension, LK8Ø assumes that the file is a REL file unless the type extension is IRL. This means that, in the example above, the module "B.C" will be read assuming it is a relocatable object module. This does not mean that the type extension "C" is ignored. It only means that what is in the file is treated as a REL file.

2.2. Linking Multiple REL Files

Multiple REL modules may be combined into one executable file by listing a group of REL modules separated by commas.

```
LK8Ø A, B, C, D
```

This command will link the four REL modules, "A.REL", "B.REL", "C.REL", and "D.REL", to an executable file "A.COM". The first name in the list is used as the name of the COM file.

As many as 4Ø REL files may be linked at one time by LK8Ø. However, the total length of the command line is limited to 128 characters. Thus it may be necessary to rename some REL files to short names when a large number of files are being linked.

The modules are linked in the order they appear in the LK8Ø command. If no drive reference is specified, the files are read from the default drive. However REL files may be linked from any drive.

```
LK8Ø AP,B:APMENUE,A:APSCN
```

When multiple modules are linked together the executable file name may be specified in the command line.

```
LK8Ø MYPROG = TEST,RTN
```

In this example the modules "TEST.REL" and "RTN.REL" are linked together forming an executable module "MYPROG.COM".

2.3. Producing Overlays

LK8Ø will produce overlay files that can be loaded and executed by a CB8Ø CHAIN statement. The CB8Ø Language Manual contains detailed information on using the CHAIN statement.

LK8Ø produces overlay (OVL) files which preserve variables in COMMON including any dynamically created data such as arrays or strings. To place a REL module in an overlay the name of the REL file is placed in parentheses.

```
LK8Ø A(B)
```

When this command is executed, LK8Ø will produce two files:

A.COM

B.OVL

The COM file is the root. The file "B.OVL" is an overlay file which may be loaded only by a CHAIN statement contained in the root "A.COM".

Chaining to an overlay is different than the conventional concept of loading overlays. When the root chains to an overlay the root itself is replaced by the overlay. Likewise when the overlay chains to another overlay or back to the root, the new overlay replaces the currently executing overlay.

CB8Ø insures that all library routines are contained in the root. Chaining preserves the libraries used by the overlay files. This reduces the size of overlays and decreases the time required to load an overlay file.

An overlay file may return to the root that loaded it by chaining back to the COM file, or the overlay may load another OVL as long as the second overlay was also linked with the same root.

```
LK8Ø A(B)(C)
```

This example produces a root "A.COM" and two overlays "B.COM" and "C.COM".

When LK8Ø produces an overlay it places all the library routines which are required by the root and an overlay into the root. This minimizes the size of overlays and ensures that the same library is not contained in multiple overlays.

Up to 4Ø overlays may be created by LK8Ø. However the total number of REL modules linked may not exceed 4Ø. A particular overlay may contain multiple REL files.

```
LK8Ø A(B)(C,D,E)(F)(G)
```

The name of each overlay as well as the name of the root may be specified in the command.

```
LK8Ø A=ROOT(B=OV1)(C=OV2)
```

The command above will produce a root "A.COM" and two overlays: "B.OVL" and "C.OVL".

2.4. LK8Ø Toggles

Information may be passed to LK8Ø by placing toggles between square brackets.

```
LK8Ø TEST[Q]
```

This command passes the Q toggle to LK8Ø.

2.4.1. Q Toggle

The Q toggle causes LK8Ø to place symbols beginning with a question mark into the SYM file. A symbol beginning with a question mark is normally used by language developers for library names. Using the Q toggle will add about 1ØØ symbols to the SYM file.

If the Q toggle is not specified the SYM file will normally only contain the symbols defined in the programs being linked.

2.5. LK8Ø Error Messages

When LK8Ø detects an error it prints on the desplay a descriptive phrase to describe the error. If the error is a fatal error control is then returned to CP/M.

The following error messages may occur:

Unresolved external: <symbol name>

The symbol name is defined as an external symbol but is never defined as a public symbol.

Out of Directory Space

LK8Ø ran out of directory space while writing the root or overlay file.

Disk Full

LK8Ø ran out of disk space while writing the root or overlay file.

Multiple Definition: <symbol name>

The symbol name is defined twice.

Too many overlays

More than 4Ø overlays were specified in the command line.

Too many modules

More than 4Ø modules were specified in the command line.

Symbol table overflow

There is not sufficent memory for the symbol table.

Cannot open source file

A source file, specified in the command line, cannot be opened.

3. LINKING WITH ASSEMBLY LANGUAGE

LK8Ø can link modules produced by the RMAC relocating Macro Assembler with REL files created by CB8Ø. The same commands explained in the previous chapter apply. The programmer should be aware that using assembly language routines makes a program machine dependent.

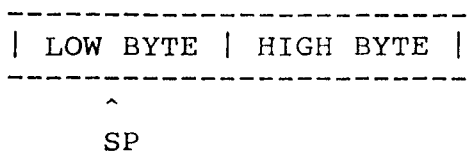
An assembly language module that is linked with CB8Ø must not contain any initialized data. This restriction is necessary because of the runtime environment required by CB8Ø. Any data that must have initial values may be placed in the code segment.

3.1. Passing Parameters

CB8Ø passes all parameters on the 8Ø8Ø hardware stack. The top of, or last entry on, the stack will contain the return address. Parameters are stored below the return address. When a routine is called, the first, or left most, parameter is placed on the stack first. Each remaining parameter from left to right is then placed on the stack.

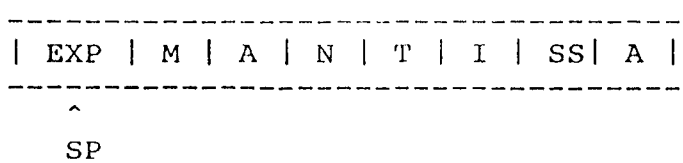
3.1.1. Integer Parameters

Integer numbers are passed on the hardware stack as sixteen bit signed integers. The integers are stored with the low-order byte in the lower memory address.



3.1.2. Real Parameters

Real numbers are passed on the hardware stack as eight byte floating point decimal numbers.



The seven bytes of the mantissa each contain two binary coded decimal digits. The left four bits of each byte in the mantissa contain the most significant digit in that byte. The mantissa is normalized so that the most significant digit is always non-zero.

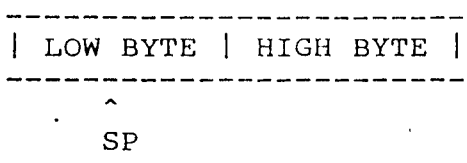
The left most bit of the exponent is the sign of the mantissa. If the bit is a one the mantissa is negative, and if it is a zero the mantissa is positive.

The remaining seven bits of the exponent represent the power of ten multiplier to be applied to the mantissa. The actual multiplier used is determined by subtracting 64 from the seven low-order bits of the exponent byte.

A number with a value of zero is represented by setting the exponent byte to Ø. The mantissa is ignored. All eight bits of the exponent must be zero for the value to be zero.

3.1.3. String Parameters

Strings are passed by placing a pointer to the actual string on the hardware stack. The pointer is an unsigned sixteen bit integer.



If the value of the pointer is zero, the string is a null string. Otherwise the pointer is the address of the string. The first two bytes of the string contain an allocation bit and a fifteen bit string length. The left most bit of the first byte is the allocation bit.

If the allocation bit is a one, the string must be returned to the CB8Ø pool of available storage prior to returning from the assembly language routine, and after all references to characters within the string have occurred. String space is returned to CB8Ø using the ?RELS library routine. The ?RELS routine is explained in chapter four.

If the 8Ø8Ø registers H and L contain the pointer to the string passed as the string parameter, the following assembly

language statements will release a string with its allocation bit set:

```
MOV  A,H        ;IF PTR = Ø THEN
ORA  L          ;   NO RELEASE
RZ
MOV  A,M        ;GET HIGH BYTE OF LNG
ORA  A          ;IS ALLOC BIT = 1?
RP              ;IF NOT NO RELEASE
CALL ?RELS      ;RELEASE THE STRING
RET
```

If the allocation bit is a zero, the characters in the string should not be changed since the calling program will still have access to the string. If the allocation bit is Ø the string cannot be released.

3.2. Returning Values to CB8Ø

An assembly language routine can return integer, real, or string values to CB8Ø. Prior to returning to CB8Ø, all parameters passed on the stack must have been removed and the stack pointer adjusted accordingly.

3.2.1. Returning an Integer

An integer number is returned in registers H and L.

3.2.2. Returning a Real Number

Real numbers are returned by placing a pointer in registers H and L to an eight byte data area containing the real number to be returned.

The number being returned must be stored in the format described above. The H and L registers contain the address of the exponent byte.

3.2.3. Returning a String

Strings are returned by placing a pointer to the string in registers H and L. The string must have been allocated by the CB8Ø's dynamic storage management routines. Dynamic storage library routines are explained in chapter four.

The allocation bit of the string being returned should be set to one to ensure that the space will be reclaimed when it is no longer required.

4. CB8Ø LIBRARY ROUTINES

This chapter describes CB8Ø runtime library routines that can be called from assembly language programs.

4.1. Dynamic Storage Allocation Routines

The CB8Ø runtime library provides four routines that allow a programmer to allocate and release memory and to determine the amount of space that is available for allocation.

The ?GETS routine allocates space. The number of bytes of memory required is placed in registers H and L. The maximum amount of space that may be allocated is 32,762 bytes.

?GETS returns a pointer in registers H and L to a contiguous block of memory. There is no restriction on what may be placed in the allocated memory but the adjacent space at either end of the area may not be modified.

If sufficient space is not available, an "OM" error will occur.

The ?RELS routine releases previously allocated memory. The address of the space being released is placed in registers H and L. ?RELS does not return a value.

The ?MFRE routine returns the size of the largest contiguous space that can currently be allocated using the ?GETS routine. The value returned is an unsigned integer and it is placed in registers H and L.

The ?IFRE routine returns the total amount of dynamic space that is currently not allocated. The value returned is an unsigned integer and it is placed in registers H and L.

4.2. Arithmetic Routines

The CB8Ø runtime library provides routines for signed integer multiplication and division. The ?IMUL routine will multiply the signed integer in registers D and E by the signed integer in registers H and L. The result is placed in registers H and L.

The ?IDIV routine will divide the signed integer in registers D and E by the signed integer in H and L. The result is placed in registers H and L.