Twenty Dollars

Cromemco®

68000 Linker

Instruction Manual

CROMEMCO, Inc.
280 Bernardo Avenue
Mountain View, CA. 94043

Part No. 023-4055

February 1983

Cromemco®

68000 Linker

Instruction Manual

CROMEMCO, Inc.
280 Bernardo Avenue
Mountain View, CA. 94043

Part No. 023-4055

February 1983

This manual was produced using a Cromemco System Three computer with a Cromemco HDD-22 Hard Disk Storage System running under the Cromemco Cromix® Operating System. The text was edited with the Cromemco Cromix Screen Editor. The edited text was proofread by the Cromemco SpellMaster™ Program and formatted by the Cromemco Word Processing System Formatter II. Camera-ready copy was printed on a Cromemco 3355B printer.

TABLE OF CONTENTS

Chapter 1

LINK68 - 68000 LINKER

Link68 is a two-pass virtual linker that can be used to link the relocatable **.o68** files produced by the Cromemco Macro Assembler into an executable file.

The command format used to call the linker is:

link68 [options] filename ... [-s libname] ...

Options

-q	The -q option inhibits displaying the link map on the terminal.
-n	The -n option prevents creating the link map. Otherwise, the link map is created and written to a file with the same filename as the **.bin** file (defined by the -o option) but with the filename extension **.map.**
-z #	The -z option is used to define the amount of free space at the end of the linked program. The linker rounds the specified (decimal) value (represented by #) up to the nearest 4K boundary and adds the computed value to the total program size. The result is that at least the specified number of bytes is available at the end of the user program.
-o outname	Specifies the created output filename. The actual filename will have the default extension **.bin** if not otherwise specified. If the -o option is not specified, the first input filename will also be used as the output filename.

-s libname — Declares the name of the library to be searched for any missing modules. The search proceeds sequentially through the library so that the library should be ordered correctly in the sense that each module should precede any modules it calls. A library may be formed simply by concatenating the modules (for example, using the Cromix Type utility) or by using the Maklib utility, which sorts, and, if necessary, duplicates modules before concatenating them. Multiple -s options are allowed.

Note that the programs themselves may request the search through a library by means of the *rellib pseudo instruction.

Example:

```
link68 -z 5000 test subl sub2 -s asmlib
```

This command line links the files **test.o68**, **subl.o68**, **sub2.o68**, searches through the library **asmlib.o68** to find the missing modules, and writes the executable program on the file **test.bin** and its map on the file **test.map.** The linked program will have at least 5000 (decimal) bytes allocated at the end of the last module for use by the program.

Chapter 2

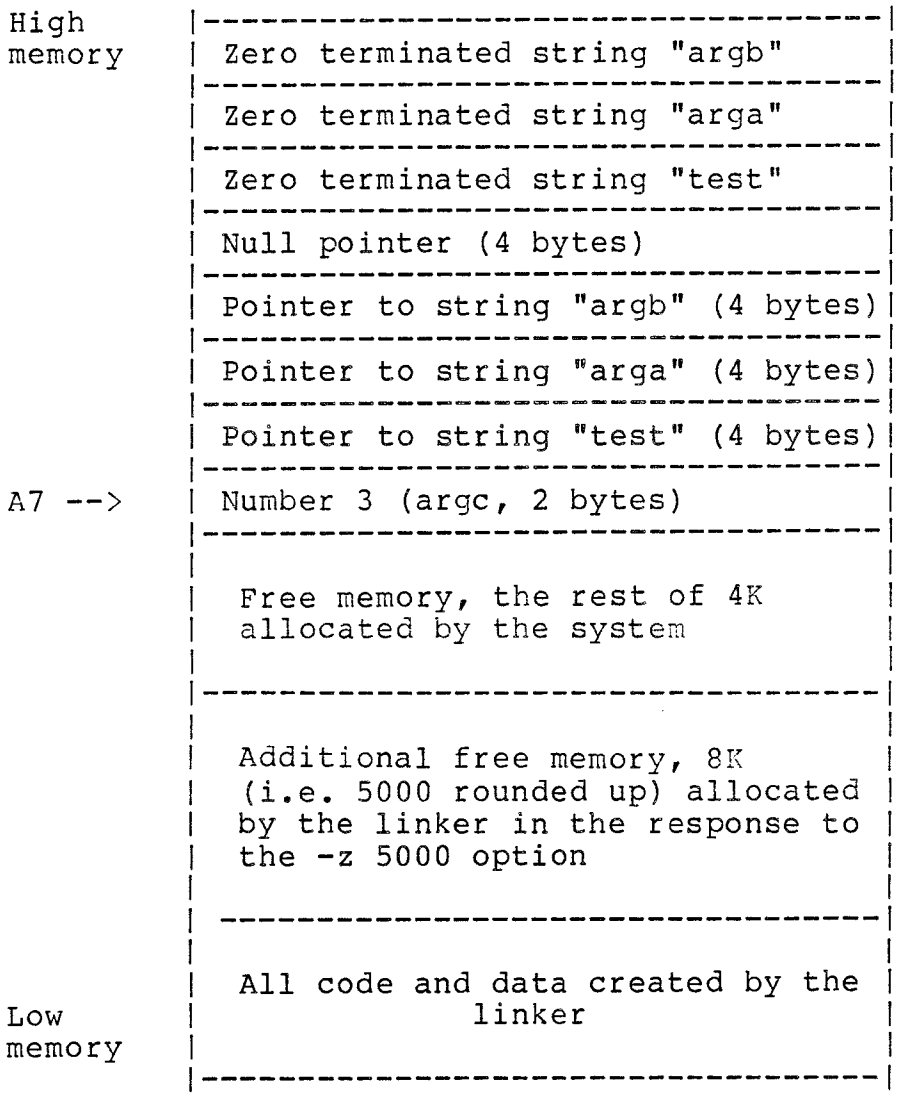
68000 CROMIX PROGRAM MEMORY ALLOCATION

When the Cromix Operating System loads the file **test.bin** for execution, it allocates an additional 4K bytes of memory at the end of the file. This 4K is used to store the command line argument strings, the pointers to them (argv) and the number of them (argc). At the start of execution the stack pointer will point to argc.

Suppose that the program test is executed by the call

```
test arga argb
```

When the program test starts execution, the memory layout is as follows:

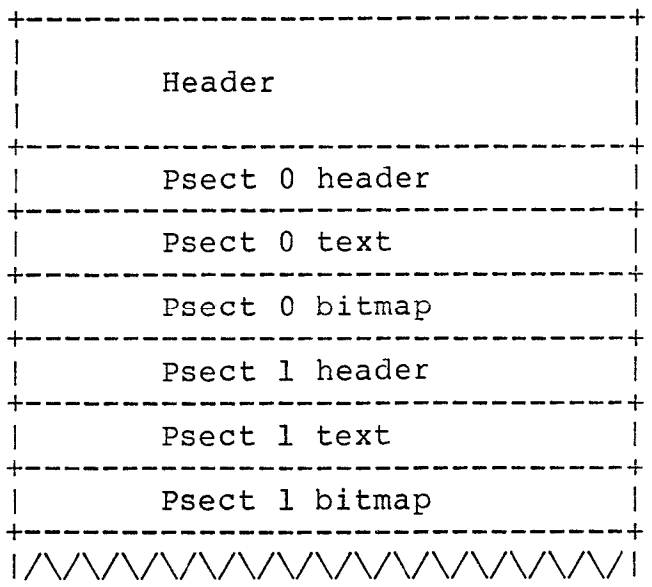


Note that "Low memory" will be at address 20000h or higher as the first 20000h bytes of memory are used by the system.

Chapter 3

FORMAT OF 68000 CROMIX EXECUTABLE FILES

Bin files linked to be executed under the 68000 Cromix Operating System have the following format:



The bit map contains 1 bit for every word in the psect. If the bit is set, then the long word at that position is relocated.

Description of the **.bin** header

```
                struct  0
bn_magic        ds.w    1       ; Magic number (0C7C7h)
bn_pvers        ds.w    1       ; Version number of program
bn_lvers        ds.w    1       ; Version number of linker
bn_lnkid        ds.b    1       ; Linker ID number
bn_npsect       ds.b    1       ; Number of psects
bn_allocate     ds.l    1       ; Allocation size
bn_start        ds.l    1       ; Start of program
                ds.b    16      ; Reserved for later use
bn_length       ds.b    0       ; Length of header
                mend
```

Description of the psect header

```
                struct  0
psect_attr      ds.w    1       ; Attribute
                ds.w    1       ; Reserved
psect_load      ds.l    1       ; Load address for psect
psect_size      ds.l    1       ; Size for psect
psect_bmsize    ds.l    1       ; Bitmap size
psect_len       ds.b    0       ; Length of psect header
                mend
```

Note that bn_allocate in the header may be patched with the Cromix Patch utility to increase or decrease the amount of free space. Bn_allocate should be a multiple of 4K bytes.

Chapter 4

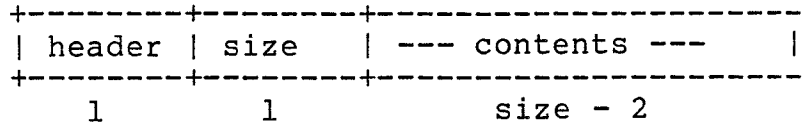
CROMEMCO RELOCATABLE FILE FORMATS

Cromemco supports two different formats of object files. The first, referred to as **.o68** format, is produced by the Assembler; the second is SVS **.obj** format, used by the 68000 FORTRAN-77, Pascal, and C compilers. Link68 uses the **.o68** format and must be used to link the assembly language modules, whereas the Crolinker has to be used to link the compiled high-level language modules.

The following section briefly describes the two relocatable file formats.

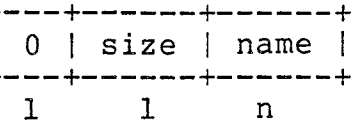
CROMEMCO .o68 FORMAT

The object file consists of a number of records. Each record is a group of bytes with the following structure:



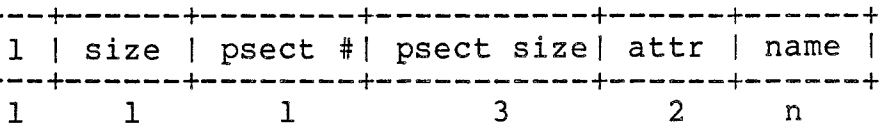
where header is a number in the range 0 - 13 (the **.o68** format currently consists of 14 different record types), size is a record size in bytes (in the range 2 - 255). The contents depends on the header and currently can be any of the following:

Module name



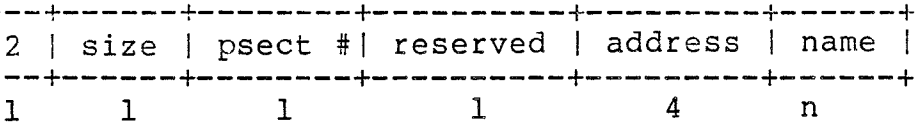
This record passes the module name to the linker. The name is either the string defined via the NAME pseudo instruction code, or the name of the file containing that module (in case the NAME pseudo operation code has not been used).

Psect



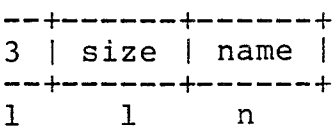
This record passes the psect characteristics to the linker. It consists of psect # (it is actually the ordinal number of the psect in the module), psect size in bytes, psect attributes and name.

Entry



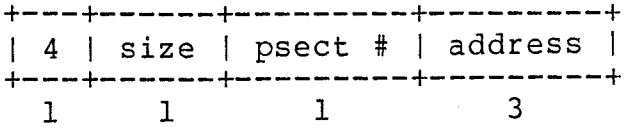
The entry record contains the information for an entry point symbol. The data are the psect # (the ordinal number of the psect in which the entry symbol is defined), the address (relative to the given psect start), followed by entry symbol name (defined via the ENTRY pseudo operation).

External



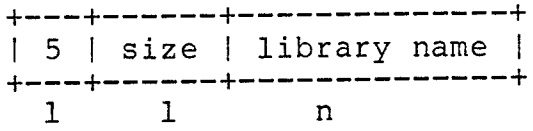
This record declares an external symbol, giving its name (defined by the EXTERN pseudo operation). The linker assigns an integer number (starting with zero) to each external symbol.

End of module



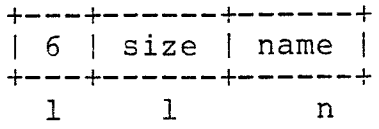
This record declares the end of a module. It also passes the start address to the linker. A psect number = 255 and address = 0 signify the absence of a start address.

Force search of library



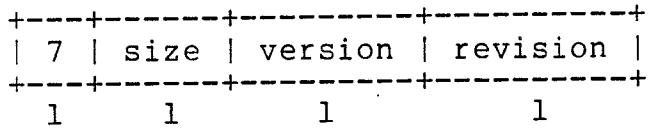
This record informs the linker to search the named library. It is generated by the *RELLIB pseudo operation code.

Force load a file



This record tells the linker to load a specific **.o68** file. It is generated by the *RELOBJ pseudo instruction code.

Version number



The sole purpose of this record is to insert the version and revision numbers into the object file.

Write text record to binary output file

8	size	text size	psect #	unused	address	text
1	1	1	1	1	3	n

This record is a part of the program text (i.e., assembled instructions or initialized data). The record contains the size of the text, the psect number it belongs to, its loading address relative to the start of the psect given, and the text itself.

Fix external symbol

9	size	type	psect #	addr	external id
1	1	2	1	3	2

The information in this record tells the linker that the external number "external id", referenced in psect # at address "addr" must be relocated. The field "type" describes the addressing mode. The structure of this field is described at the end of this section.

Fix symbol in other psect

10	size	type	psect #	addr	psect #	reserved
1	1	2	1	3	1	1

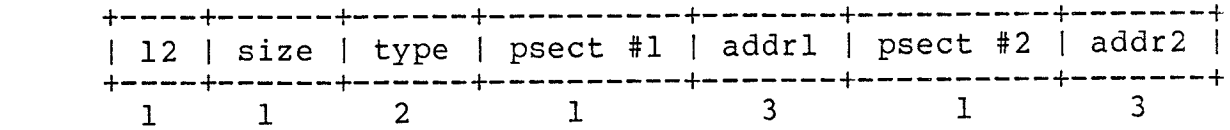
The information in this record tells the linker that the symbol in psect #, referenced at address "addr", which is actually defined in the psect with the second psect # has to be relocated. The field "type" describes the addressing mode.

Fix symbol in same psect

11	size	type	psect #	addr
1	1	2	1	3

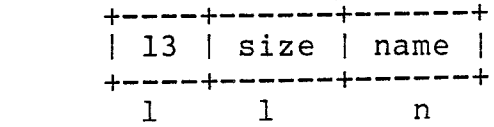
The information in this record tells the linker to relocate the symbol at address "addr" in the psect defined by "psect #". The field "type" contains the information about the addressing mode.

Fix forward reference



The information in this record tells the linker that the actual address of the symbol, referenced in psect "psect #1", at address "addr1", is to be relocated by the value of "addr2". This symbol is defined in psect "psect #2", and, in most cases, "psect #2" = "psect #1". The pseudo operation code IFcc generates this record.

Declare entry symbol



This record declares an new entry symbol by defining its name. The record is generated by the ENTRY pseudo operation code.

Order of Records

Module name (0) must be the first record. Declare entry records (13) must be next. The end module record (4) must be last. Any fixup record must be after the record to be fixed. External name records (3) must be in the file before the external symbols listed with the external name records are referenced.

Addressing Mode Field Description

This 2-byte field contains the information about the addressing mode in fixup records. The following bits are relevant:

Bit	Meaning
4	Absolute
5	Long
6	Word
7	Byte

Bits are numbered in the standard way, 0 - 15, the rightmost bit being least significant.

SVS .OBJ FORMAT

An **.obj** file is a file of bytes, as is a **.o68** file, but with a somewhat different organization. The file consists of records according to the rule:

```
<Module file> ::= <module>* EOF mark

<module>      ::= <module name record>
                  <other record>+
                  <end record>
```

where * means "0 or more occurrences," + means "1 or more occurrences," and <other record> may be any of the following:

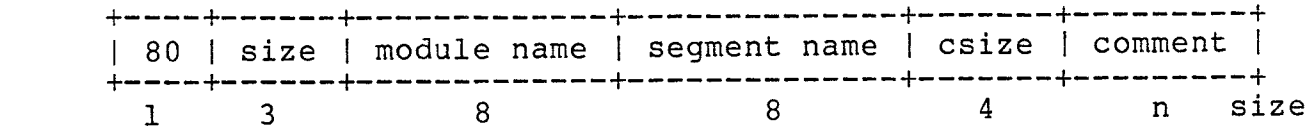
Entry record
External record
Start record
Code record
Relocation record
Common relocation record
Common definition record
Short external record
Data initialization record
FORTRAN data area definition record
FORTRAN data area initialization record
FORTRAN data area reference record
FORTRAN data area relocation record

The organization of a library file (such as **paslib.obj** or **ftnlib.obj**) is as follows:

```
<library file> ::= Library module record+
                   Library entry record+
                   Library starting address record*
                   <module>+ text record*
                   EOF mark
```

with module having the above-defined structure. The descriptions of these records follow:

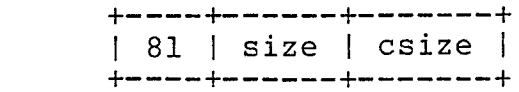
Module name record



where:

80	-	Hexadecimal 80.
size	-	Number of bytes in this record.
module name	-	Blank padded ASCII name of the module.
segment name	-	ASCII name of the segment in which this module will reside.
csize	-	Number of bytes in the code for this module.
comments	-	Arbitrary information. Ignored by the linker.

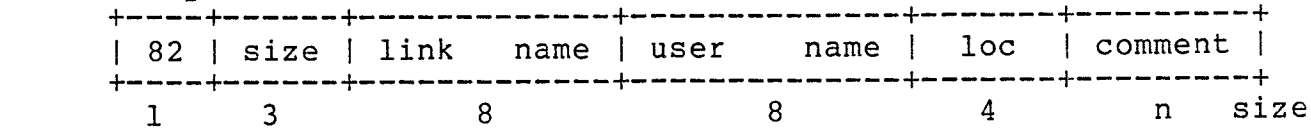
End record



where:

81	-	Hexadecimal 81
size	-	Number of bytes in this record; always 08h.
csize	-	Number of bytes in the code record for this module.

Entry record

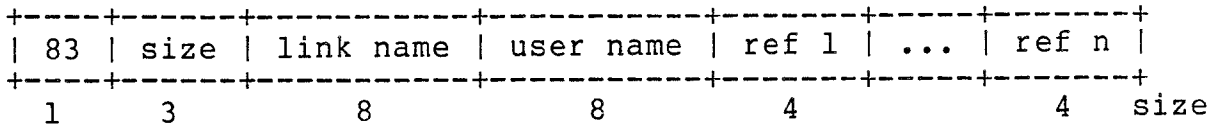


where:

82	-	Hexadecimal 82.
size	-	Number of bytes in this record.
link name	-	Blank padded ASCII linker name of entry point.
user name	-	Blank padded ASCII user name of entry point.

loc	-	Location of entry point relative to this module.
comments	-	Arbitrary information. Ignored by the linker.

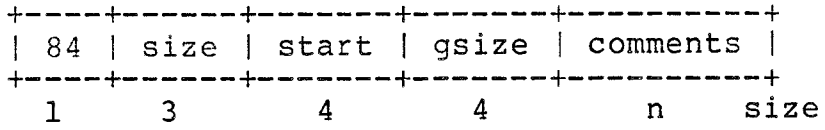
External record



where:

83	-	Hexadecimal 83.
size	-	Number of bytes in this record.
link name	-	Blank padded ASCII linker name of external reference.
user name	-	Blank padded ASCII user name of external reference.
ref 1	-	Location of first reference relative to this module.
...		Other references.
ref n	-	Location of last reference.

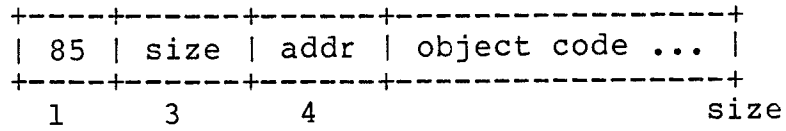
Start record



where:

84	-	Hexadecimal 84.
size	-	Number of bytes in this record.
start	-	Starting address relative to this module.
gsize	-	Number of bytes in the global data area.
comments	-	Arbitrary information. Ignored by the linker.

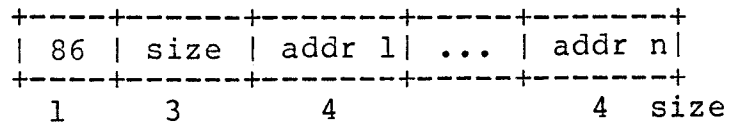
Code record



where:

85	-	Hexadecimal 85.
size	-	Number of bytes in this record.
addr	-	Module relative address of first byte of code.
object code	-	The object code. Always an even number of bytes.

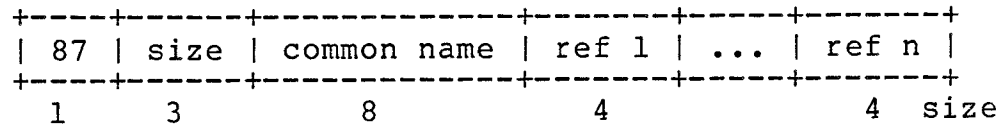
Relocation record: (32 bit)



where:

86	-	Hexadecimal 86.
size	-	Number of bytes in this record.
addr 1	-	Location of first byte to relocate.
...	-	Other references.
addr n	-	Location of last byte to relocate.

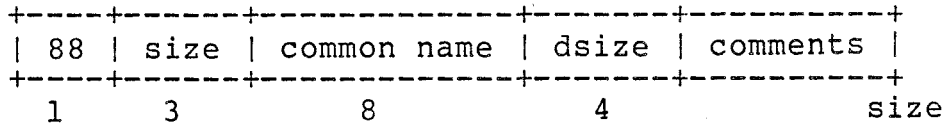
Common record



where:

87	-	Hexadecimal 87.
size	-	Number of bytes in this record.
common name	-	Blank padded ASCII name of common record.
ref 1	-	Location of first reference relative to this module.
...	-	Other references.
ref n	-	Location of last reference.

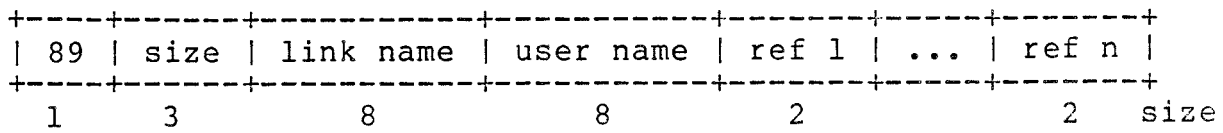
Common definition record



where:

88	-	Hexadecimal 88.
size	-	Number of bytes in this record.
common name	-	Blank padded ASCII name of common area.
dsize	-	Number of bytes in this common data area.
comments	-	Arbitrary information. Ignored by the linker.

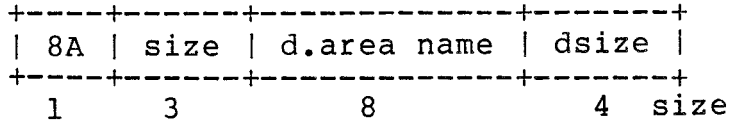
Short external record



where:

89	-	Hexadecimal 89
size	-	Number of bytes in this record.
link name	-	Blank padded ASCII linker name of external reference
user name	-	Blank padded ASCII user name of external reference.
refl	-	Location of first reference relative to this module.
...		Other references.
ref n	-	Location of last reference.

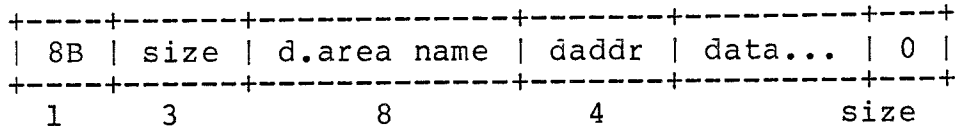
FORTRAN data area definition record



where:

8A	-	Hexadecimal 8A.
size	-	Number of bytes in this record.
d.area name	-	Blank padded ASCII name of FORTRAN fixed data area.
dsize	-	Size of this data area.

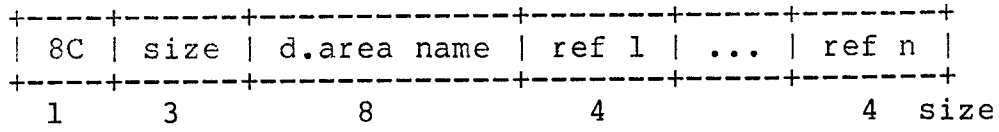
FORTRAN data area initialization record



where

8B	-	Hexadecimal 8B.
size	-	Number of bytes in this record.
d.area name	-	Blank padded ASCII name of FORTRAN fixed data area.
daddr	-	Starting address of this data.
data	-	The initialization data.
0	-	One byte of 00 if size id odd.

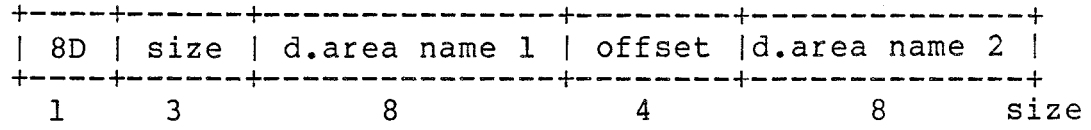
FORTRAN data area reference record



where:

8C	-	Hexadecimal 8C.
size	-	Number of bytes in this record.
d.area name	-	Blank padded ASCII name of FORTRAN fixed data area.
ref 1	-	Address of first reference.
...	-	Other references.
ref n	-	Location of last reference.

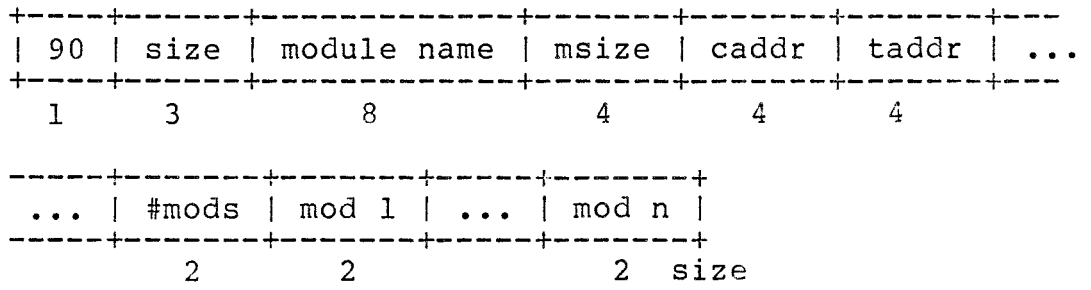
FORTRAN data area relocation record



where:

8D	-	Hexadecimal 8D.
size	-	Number of bytes in this record.
d.area name 1	-	Blank padded ASCII name of FORTRAN data area in which reference appears.
offset	-	Offset in data area 1.
d.area name 2	-	Blank padded ASCII name of FORTRAN data area to be relocated to.

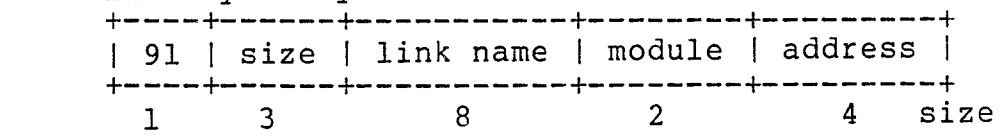
Library module record



where:

90	-	Hexadecimal 90.
size	-	Number of bytes in this record.
module name	-	Name of this module.
msize	-	Size of code for this module in bytes.
caddr	-	Disk address of the module.
taddr	-	Disk address of text record if present. Zero otherwise.
tsize	-	Size of text record.
#mods	-	Number of other modules referenced by this module.
mod 1	-	Number of first module referenced.
...	-	Other references.
mod n	-	Number of last module referenced.

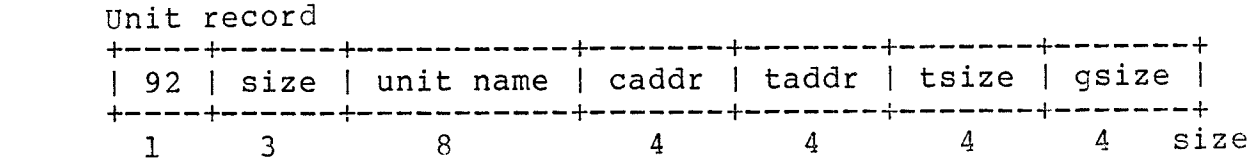
Library entry record



where:

91	-	Hexadecimal 91.
size	-	Number of bytes in this record. Always 12h.
link name	-	Blank padded ASCII link name of entry point.
module	-	Module in which entry point resides.
address	-	Relative address of entry point to that module.

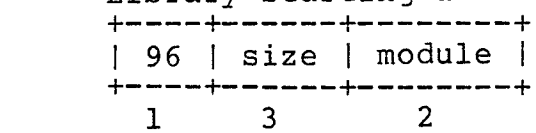
Unit record



where:

92	-	Hexadecimal 92.
size	-	Number of bytes in this record.
unit name	-	Name of this unit.
caddr	-	Disk address of module.
taddr	-	Disk address of text record.
tsize	-	Size of text record.
gsize	-	Number of bytes of globals in this unit.

Library starting address record

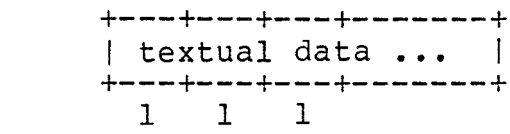


where:

96	-	Hexadecimal 92.
size	-	Number of bytes in this record.
module	-	Module # in the library, which contains the starting address record.

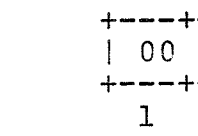
This record appears after all "90" records and before all "91" records.

Text record



The format of a text record is operating system dependent. The current version uses UCSD text file format, excluding the two initial header records. It is always stored on disk record boundaries.

Eof mark



00 00 - Hexadecimal 0000

Chapter 5

UTILITY PROGRAMS

PRTO68 - STRUCTURED DUMP OF .o68 FILE

A special program, **prto68.bin**, is supplied with the 68000 Assembler package. It can be used to display the contents of any **.o68** file. The calling format is:

PRTO68 [-q] <pathname>

where:

-q	is an optional parameter. If specified, **prto68** will not display the contents of code records.
<pathname>	is the pathname of the **.o68** file to be dumped.

PRTOBJ - STRUCTURED DUMP OF .OBJ FILE

Prtobj is a program similar to **prto68** and can be used to display the contents of an SVS **.obj** file. Its calling syntax is also:

PRTOBJ [-q] <pathname>

CONVO68 - CONVERT .o68 TO .OBJ FILE

Cromemco 68000 Pascal, 68000 C, and FORTRAN-77 all generate relocatable files with the **.obj** format, which is different from the **.o68** format generated by the Cromemco 68000 Macro Assembler. Sometimes it may be necessary for a high-level language program to perform an action that can be done only by an assembly language subroutine. The program Convo68, which converts a **.o68** file to **.obj** format can be used in such a case.

The calling syntax is:

convo68 [-v] <pathname> [<pathname>]

where:

-v — is an optional parameter (verbose), which causes Convo68 to display some information about what it is doing.

<pathname> — is a name of the file (**.o68**) to be converted to **.obj** format. The extension need not be given, for Convo68 assumes the **.o68** extension. If the extension is given, Convo68 will not append the **.o68** extension. If only one pathname is given, the resulting output file will have the same name as the input file, with the extension changed to **.obj**. Otherwise, the first pathname specifies the input file (**.o68**) and the second, the output file (**.obj**).

Because of significant differences between the two object file formats, there are some strong restrictions imposed on Convo68:

- Only one code psect per module is allowed. Convo68 recognizes a psect as a code psect if it is an executable psect (see Chapters 4 and 7 of the 68000 Macro Assembler manual).

- Only one data psect per module is allowed. Convo68 recognizes a psect as a data psect if it is not an executable psect (EXE attribute) and not a common psect (COM attribute). (For details, see the 68000 Macro Assembler manual, Chapters 4 and 7.)

- As many as 19 common psects are allowed. Convo68 recognizes a psect as a common psect if it has a COM attribute.

- Convo68 will not convert a file containing a main program. This is not a severe restriction because the main program will always be written in a high-level language.

- Force search of library record is not allowed.

- Force load a file record is not allowed.

- Fix the external symbol record should be generated with PC relative addressing mode because this record is converted into the SVS short external reference record. In other words, all references to code should be PC relative. (This can be accomplished by setting the EXT_PC bit via the OPT pseudo operation code.)

- References to data should use absolute long addressing mode.

- Fix symbol in same psect record is not allowed. That means that the module to be converted cannot contain any absolute (long or short) addresses, but all references must be PC relative. Thus, the following segment of an assembly language program cannot be converted:

```
addr:   move    d1, d3
        dc.l    addr
```

  because the DC.L pseudo operation code will produce the absolute long address.

- Fix symbol forward record not allowed. The 68000 Macro Assembler will generate this record for every IFcc pseudo operation code.

- Version record is ignored.

To illustrate the usage of Convo68, two sample programs are given. In the first example, the main program is written in Pascal, which calls the assembly language subroutine. The subroutine uses the **.gettime** Cromix system call and returns the system time via the parameter list. The second example is a FORTRAN-77 main program which calls a similar assembly language subroutine. The second assembly language routine differs from the first in that it returns the time via the labelled COMMON.

Example 1

```
program time1;

  type tm = packed array [1..3] of char;

  var  time : tm;
       err  : integer;  {Cromix return code}

       function getiml (var t : tm) : integer;
         external;


begin
  err := getiml (time);
  if err = 0 then
    writeln ('Time : ',ord (time[1]):2,':',
                       ord (time[2]):2,':',
                       ord (time[3]):2      )
  else
    writeln ('Error no. ',err)
end.
```

The assembly language module getiml should be:

```
        entry   getiml
;
;       68000 Pascal interface to Cromix
;        gettime system call
;
*include        '/equ/jsysequ.asm'
;
first:  equ     4
;
        struct  first           ; start of parameters
ra:     ds.l    1               ; return address
parl:   ds.l    1               ; first parameter
next:   ds      0               ; end of parameters
fv:     ds      1               ; function value
        mend
;

getiml:
        link    a6,#0             ; Establish stack area
;
        jsys    #_gettime
        bcc.s   noerr             ; If carry set return
          move  d0,fv(a6)         ; return the error #
          bra.s exit
```

```
noerr:                                  ; else
          clr     fv(a6)                ; return zero
          move.l  parl(a6),a0           ; pointer to parameter (time[1])
          move.b  d1,(a0)+              ; put hours
          move.b  d2,(a0)+              ; minutes
          move.b  d3,(a0)               ; and seconds
exit:
;
        move.l  ra(a6),a0               ; get return address
        pop.l   a6                      ; unlink
        add     #next-first,sp          ; adjust stack pointer
        jmp     (a0)                    ; and return to caller
;
        end
```

The commands to produce the executable **.bin** file are (supposing that main program is stored in the file **example1.pas** and the subroutine in the file **getiml.asm**):

```
pascal example1
code example1.i
asm getiml
convo68 getiml
crolinker example1 getiml /usr/lib/paslib
```

Example 2

```
        Program time2
c
        integer       hour, min, sec
        common /time/ hour, min, sec
        integer rerr
c
         getim2 is an integer function, its value
c        is the Cromix error return code
c
        rerr = getim2 (dummy)
c
        if (rerr .eq. 0) then
          write (*, 100) hour, min, sec
        else
          write (*, 110) rerr
        endif
c
 100    format ('Time : ',i2,':',i2,':',i2)
 110    format ('Error no. ',i3)
        end
```

with the getim2 assembly language subroutine:

```
        entry   getim2
;
;       68000 Fortran interface to Cromix
;        gettime system call
;
*include        '/equ/jsysequ.asm'
*include        '/equ/optequ.asm'
;
        OPT      DEFAULT |^FWD_L
;
getim2:
;
        jsys    #_gettime
;
        bcc.s   noerr           ; if carry set (error)
          move.l  d0,4(a6)      ; return the error #
          bra.s   exit          ; and exit
noerr:                          ; else
;
          clr.l   4(a6)         ; return zero (no error)
          move.l  d1,h          ; put hours into common
          move.l  d2,m          ; minutes
          move.l  d3,s          ; seconds
;
exit:
        pop.l   a0              ; retrieve return address
```

```
        jmp     (a0)            ; and return to caller
;
        psect   '/time  /' (REA, WRI, COM)
;
h:      ds.l    1               ; four bytes allocated
m:      ds.l    1               ; because FORTRAN uses
s:      ds.l    1               ; long integers.
;
        end
```

OPT pseudo operation code is used to select the absolute long addresses for the move instructions. As previously noted, references to data (in this case, psect '/time /') must all be absolute long. Because the data psect is placed after the code psect, the Assembler needs help determining whether forward references should be short or long addresses. The problem would be simpler if all data segments were placed before the code segment, in which case, references to data would be backward references. Because the default address mode setting (OPT pseudo opcode) is for references to other psects to be absolute long addresses, the Assembler is able to choose the correct addressing mode without additional help.

To establish the connection with the /time/ labeled common declared in the main (FORTRAN 77) program, the psect name should be defined as a quoted string, as in the preceding example (see Chapters 4 and 7 of the 68000 Macro Assembler manual).

The commands to produce the **.bin** file (using the same assumptions as Example 1) are:

```
fortran example2
code example2.i
asm getim2
convo68 getim2
crolinker example2 getim2 /usr/lib/ftnlib
```

If **fortran.bin** is version 1.0, the last command should be:

```
crolinker example2 getim2 /usr/lib/ftnlib /usr/lib/paslib
```

MAKLIB

Maklib can be used to construct a 68000 **.o68** relocatable library. The program reads all the modules, determines which module is calling which module, and sorts them so that linking any user program finds all the modules necessary. (The program will duplicate some modules if required). If Maklib encounters modules with the same name, it keeps only the first one.

The calling syntax is:

maklib <argl>, ..., <argn>

where arguments may be:

-v	verbose
-m filename	file name of the map generated (extension **.map**)
-o filename	file name of the created library (extension **.rel**)
filename, filename, ...	input files, (extension **.rel**)