Advanced
Micro
Computers
A subsidiary of
Advanced Micro Devices

LINK8000
AmZ8000 Linker

User's Manual

00680148

$5.00

	REVISION RECORD
REVISION	**DESCRIPTION**
01	Preliminary Issue
(1/18/80)	
02	Manual Revised to correct documentation
(3/21/80)	
A	Manual Released
(4/25/80)	
B	Manual updated and reprinted to support AmZ8001.
(12/31/80)	
Publication No. 00680148	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning
this manual to:

ADVANCED MICRO COMPUTERS
Publications Department
3340 Scott Boulevard
Santa Clara, CA 95051

PREFACE

The AMC LINK8000 product is primarily intended for linking relocatable code assembled by MACRO8000. LINK8000 is considered a supporting product for MACRO8000 users. The LINK8000 directives, statement structure, and general design are similar to MACRO8000. For instance, the distinction between PROGRAM and MODULE is identical in MACRO8000 and LINK8000.

MACRO8000 and LINK8000 together support development of programs for both the AmZ8001 and AmZ8002 processors.

The notations used in this manual are:

UPPERCASE — In syntax indicates a name that is specified as shown.

lowercase — In syntax indicates that a name or value must be supplied by the user.

... — In syntax indicates that an item can be repeated.

⋮ — In examples indicates that some part of the program is not shown.

Important related information can be found in the:

Manual	Number
AMC MACRO8000 AmZ8000 Assembler User's Manual	00680119

NOTE

The information in this publication is intended to be accurate in all respects. However, Advanced Micro Computers disclaims responsibility for any errors and any consequences resulting from errors. This product is intended for use as described in this manual.

TABLE OF CONTENTS

Chapter

TABLE OF CONTENTS (Cont.)

APPENDIXES

FIGURES

TABLES

CHAPTER 1
OVERVIEW OF LINK8000

LINK8000 takes several modules of relocatable AmZ8000 code and combines them into a single module of either absolute code or relocatable code. Absolute code can be targeted for either the AmZ8001 or the AmZ8002. The relocatable output module can be either a library or a module to be used in a later linking operation. The relocatable module can also be used as input for a user-defined loader.

LINK8000 requires 64K bytes of memory, either on an AmSYS 8/8 or System 29. LINK8000 itself requires more than 30K, AMDOS uses about 8K, and the rest is symbol table space and working storage.

1-1. THE AmZ8000 PROGRAMMING ENVIRONMENT

The AmZ8000 programming environment is determined by the answer to one fundamental question:

> Does the source program or program module use any segmented addresses?

A segmented address is represented by a pair of numbers, a 7-bit segment number and a 16-bit offset; it is stored in a 32-bit register pair. A non-segmented address, on the other hand, is represented by a single, 16-bit number; it is stored in a word register. Segmented addresses can be used only by the AmZ8001, while non-segmented addresses can be used by either the AmZ8001 or the AmZ8002. (A bit in the FCW of the AmZ8001 controls the type of addresses it uses. See the AmZ8001/2 Instruction Set Manual for more details.)

The user assembles a program with MACRO8000 before calling the linker. When the assembler is invoked, the S option controls the programming environment. If the S option is not chosen, the output code will use exclusively non-segmented addresses. If the S option is chosen, the output code may use segmented addresses as well as non-segmented addresses, and the code must be run on an AmZ8001. If the S option is not chosen, the code will usually be run on an AmZ8002 (although with a user-supplied loader it is possible to run the code on an AmZ8001). Hereafter in this manual and in the MACRO8000 User's Manual, we will use such phrases as "targeted for the AmZ8001" or "AmZ8001 code" to mean that the assembler S option has been chosen, and conversely, we will use "targeted for the AmZ8002" or "AmZ8002 code" to mean that the S option has not been chosen. Chapter 5 of the MACRO8000 User's Manual contains a discussion of segmented an non-segmented addresses and how users can specify which kind is generated by the assembler.

If the S option is chosen, relocatable code is produced, but either relocatable or absolute code may be produced if the S option is not chosen. Relocatable code must be further processed by LINK8000, which takes one or more relocatable files and combines them into a single absolute file or into another relocatable file. Figure 1-1 illustrates all the possible paths from source file to absolute file.

An AmZ8000 program typically consists of one or more modules. The module is the smallest programming unit that can be assembled separately. Each module is assembled using the assembler option 0, which generates relocatable code, and each module exists as a single, relocatable file.

Programmers may subdivide modules into program segments; for example, a module might be partitioned into a code segment and a data segment. Segments cannot be assembled separately; they are simply used to partition modules. However, once several modules have been assembled, each containing several segments, LINK8000 can be used to rearrange and combine the segments in an arbitrary manner. A segment is thus the smallest programming unit that can be manipulated by the linker.

1-2. LINKING OPERATION

When the user calls the linker, the user provides linker directives that tell the linker what to do. The linker directives tell the linker what input to use, what to do with the input, what addresses to assign, and what output to produce. The linker begins processing by accepting linker directives. The user can:

Save the linker directives in a directive file (default file type .DIR) that is read by the linker.

Enter linker directives interactively at the console.

The linker determines the general type of linking operation from the first directive. The user can specify a:

PROGRAM directive to link relocatable code into absolute code for downloading to an AmZ8001 or AmZ8002 processor.

MODULE directive to combine relocatable code into a single relocatable module to be used in later linking operations.

LIBRARY directive to create a user library of relocatable code that can be used in later linking operations.

ROMLIB directive to create a ROM library that contains only a directory of globals associated with absolute code in hex or binary (AMC Bin) file form (i.e., pre-defined entry points).

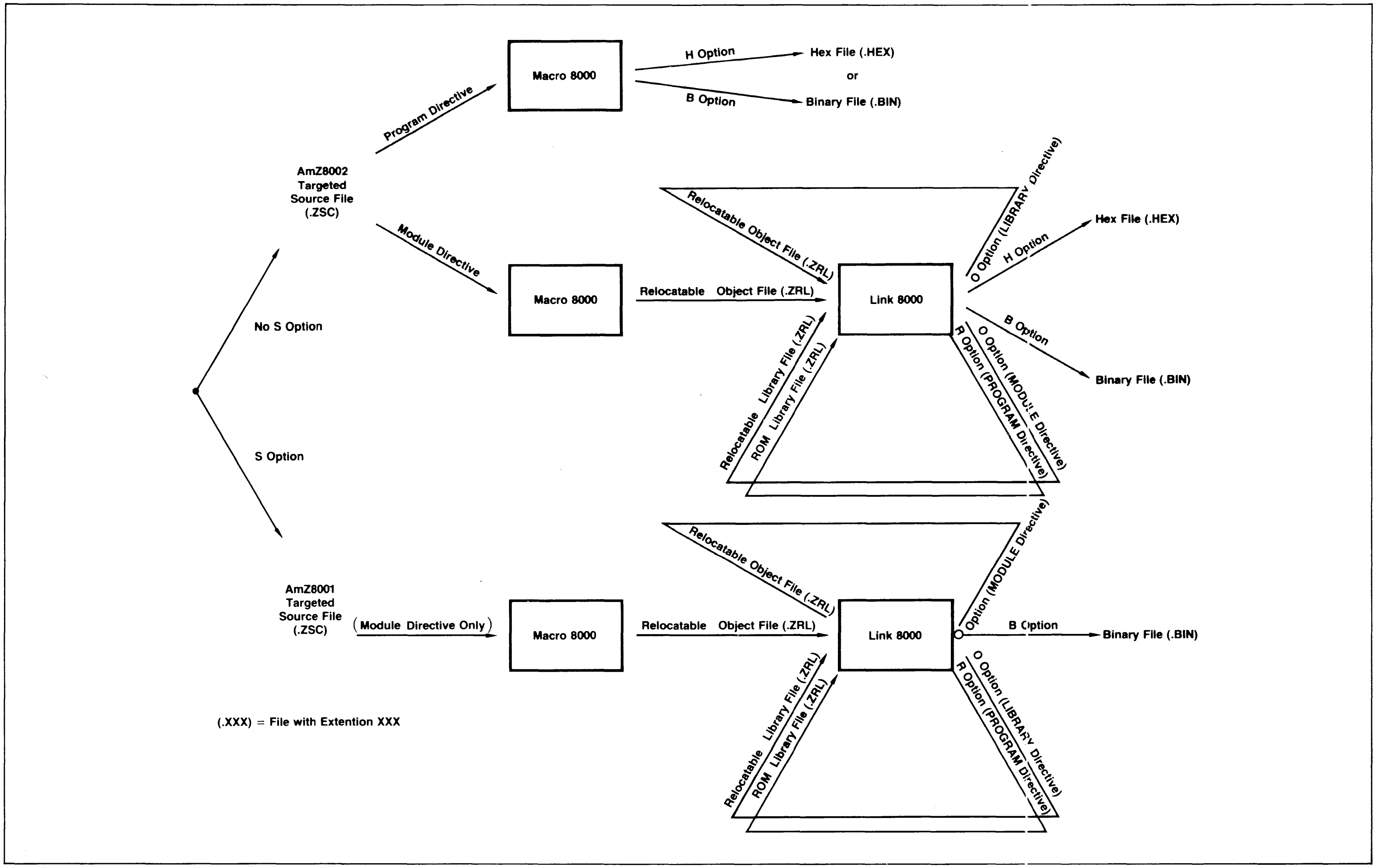
Macro 8000
H Option
Hex File (.HEX)
or
B Option
Binary File (.BIN)
Program Directive
AmZ8002
Targeted
Source File
(.ZSC)
Module Directive
Macro 8000
Relocatable Object File (.ZRL)
Relocatable Object File (.ZRL)
Link 8000
O Option (LIBRARY Directive)
H Option
Hex File (.HEX)
B Option
Binary File (.BIN)
Relocatable Library File (.ZRL)
ROM Library File (.ZRL)
R Option (PROGRAM Directive)
O Option (MODULE Directive)
No S Option
S Option
AmZ8001
Targeted
Source File
(.ZSC)
(Module Directive Only)
Macro 8000
Relocatable Object File (.ZRL)
Relocatable Object File (.ZRL)
Link 8000
O Option (MODULE Directive)
B Option
Binary File (.BIN)
Relocatable Library File (.ZRL)
ROM Library File (.ZRL)
O Option (LIBRARY Directive)
R Option (PROGRAM Directive)
(.XXX) = File with Extention XXX

Figure 1-1. Source to Absolute File Paths

The linker accepts input files for the linking operation. The relocatable files can be modules containing program segments. The relocatable files can also be combined relocatable modules or libraries created in previous linking runs. The FILE directive accesses relocatable files as input to the linker.

The linker also accepts directives that indicate the libraries to be searched in the linking operation. The libraries have the default file type .ZRL and are either regular libraries or ROM libraries. The SEARCH directive accesses libraries for satisfying externals.

The linker resolves symbol references among the program segments contained in the relocatable modules. This step is the main function of the linker. A single global symbol, such as an entry point, might be referenced in one or more other program segments as an external. In all cases, the linker matches the externals to the global. Certain externals might not be satisfied among the segments in the modules but might be satisfied by routines in a library. The linker first attempts to satisfy external references among the segments in the modules and then checks any specified libraries. The user can also directly assign absolute addresses to unsatisfied externals during the linking run.

> NOTE
>
> The externals and globals processed by the linker are the symbols declared as EXTERNAL and as GLOBAL in MACRO8000 modules. Since the modules have already been assembled, the linker does not have any record of the other identifiers used in the MACRO8000 program.

For some programs or modules targeted for an AmZ8001, the linker has another important function: to assign program segments to hardware segments. In order to explain this function we will have to explain the notion of a segmented address.

The AmZ8001 processor generates two-component segmented addresses. The first component, a seven-bit segment number, is generated on lines SN_0-SN_6 (see the AmZ8000 Family Data Book). The second component, a 16-bit offset, is generated on lines AD_0-AD_{15}. The AmZ8002 processor, which lacks lines SN_0-SN_6, generates one-component non-segmented addresses that are 16-bits long.

The address space of the AmZ8002 is thus a single, 64K linear space. The AmZ8001 address space, on the other hand, consists of 2^7=128 separate 64K linear address spaces, which we will call hardware segments (to distinguish them from program segments). The linker directive SETLSEG can be used to assign program segments to different AmZ8001 hardware segments. In the AmZ8002, all program segments are put into the same 64K address space. Read chapter 3 for more information.

In this manual, whenever there could be confusion between hardware segments and program segments, they will be explicitly distinguished. Usually, the meaning is clear from the context.

1-3. INVOKING THE LINKER

The command to call the LINK8000 linker specifies the location of the linker directives and the other options for the linking run. The call is:

```
LNKZ dirfile options overrides
```

The call is entered with a carriage return (new line key).

For interactive linking, the linker directives are supplied one by one at the console. For a linking test, where defaults are used on all options, the call is:

```
LNKZ
```

For an interactive linking run where other options on the product call are needed, the specification * indicates that linker directives will be entered from the console. For example:

```
LNKZ * B
```

calls for linking directives from the console, specifies the option B, requests defaults for the other options, and ignores overrides.

For a file containing linker directives, the specification of dirfile is:

Field	Meaning	Default
dev:	Optionally specifies a device name, such as A: or B:	Currently selected drive
filename	Specifies the name of the directives file	--
.ext	Optionally specifies the file type of the directive file	.DIR

For example:

```
LNKZ DIR4
```

calls for linking according to the directives in file DIR4.DIR, requests defaults for the options, and ignores overrides.

NOTE

Since LINK8000 has free format statements, a number of linker directives (separated by semicolons) can be entered on a single line. This technique is recommended for interactive input to the linker, since it saves time and keystrokes.

The dirfile specification (or *) can be followed by at least one space and then the selected options. The options listed in Table 1-1 are similar to the MACRO8000 options. Options can be specified in any order and are separated by commas or spaces.

For example:

```
    LNKZ DIR5 L,B=XRPROC
```

calls for linking according to the directives on file DIR5.DIR, requests a listing named DIR5.PRN, produces a file named XPROC.BIN that is suitable for downloading, and ignores overrides.

The format of the hex file produced by the linker is described in Appendix B. The binary file formats for both the AmZ8001 and the AmZ8002 are described in Appendix C.

The product call line can optionally include overrides for one or more symbolic constant values in the linker directives. The overrides follow all of the other options and are separated by commas. The overrides work in the same way as for MACRO8000 (see the MACRO8000 manual).

NOTE

> When LNKZ is invoked with a directive file, the display of directives encountered is normally suppressed unless one of the L options is used. The L option causes the directives to be displayed on the named output device.

TABLE 1-1. LINK8000 OPTIONS LIST

Name	Default	Form	Meaning
Listing	L=CON:	L	Send listing to dev:name.PRN on currently selected drive, with same name as dirfile
		L=file	Send listing to the file dev:name.ext as specified
		L=CON:	Send listing to console device (if printer is enabled with CONTROL P, listing also prints)
		L=LST:	Send listing to printer device

TABLE 1-1. LINK8000 OPTIONS LIST (Cont.)

Name	Default	Form	Meaning
Object (for MODULE, LIBRARY, ROMLIB run only)	No object file	O	Create object file dev:name.ZRL on same drive as dirfile, with same name as dirfile
		O=file	Create object file dev:name.ext as specified. The file type should not be $$$
Hex (for AmZ8002 PROGRAM only)	No hex file for PROM burning	H	Create hex file dev:name.HEX on same drive as dirfile, with same name as dirfile
		H=file	Create hex file dev:name.ext as specified
Binary (for PROGRAM run only)	No binary file for downloading	B	Create binary file dev:name.BIN on same drive as dirfile, with same name as dirfile
		B=file	Create binary file dev:name.ext as specified
ROMLIB (for PROGRAM run only)	No ROMLIB as linker output from RETAIN or OMIT	R	Create ROMLIB dev:ROMLIB.ZRL on same drive as dirfile
		R=file	Create ROMLIB file dev:name.ext as specified. This file, which is called a ROM library index or ROMLIB, contains global symbol definition. A ROMLIB might contain entry points for shared code (particularly in ROM), such as for a shared set of floating point routines which are always resident and at a fixed address. A ROMLIB can also be used to supply addresses of global symbols for symbolic debugging.

NOTE

When interactive input is specified (*) with options L, O, H, or B (without explicit filename), a default filename LINK is supplied in lieu of dirfile.

CHAPTER 2
GENERAL PURPOSE LINKER DIRECTIVES

Since LINK8000 and MACRO8000 general purpose directives are similar in many ways, this chapter makes frequent reference to features of MACRO8000. Users should consult the MACRO8000 User's Manual referred to in the preface for more information.

The user should briefly check the information covered in this chapter and then study the functional linker directives described in Chapter 3. The sample PROGRAM run for the AmZ8002 (Chapter 4) and for the AmZ8001 (Chapter 5), the MODULE run (Chapter 6), the LIBRARY run (Chapter 7), and the ROMLIB run (Chapter 8) should also be examined.

2-1. STATEMENT FORM

The statements in LINK8000 are linker directives, but the rules are the same as for MACRO8000. For example:

```
PROGRAM START;
FILE MOD1, MOD2;
ABSOLUTE #4000
END.
```

is equivalent to:

```
PROGRAM START; FILE MOD1, MOD2; ABSOLUTE #4000 END.
```

2-2. SINGLE STATEMENT

The single statement consists of a statement beginner followed by zero or more operands, as in MACRO8000. For example:

```
SEARCH LIB1, LIB2;       % statement beginner is SEARCH
                         % operands are LIB1 and LIB2
```

NOTE

> For LINK8000 interactive input of directives, the semicolon at the end of a line can and should be omitted, since a carriage return (new line key) indicates the end of a statement. The semicolon must still be used between statements on the same line.

For example:

```
    ==> FILE MOD1, MOD2
    ==> FILE MOD3; SEARCH LIB1, LIB2
```

2-3. COMPOUND STATEMENT

A compound statement consists of BEGIN, single statements, and END, as in MACRO8000. For example:

```
    BEGIN
    FILE MOD1, MOD2;
    SEARCH LIB1, LIB2
    END;
```

2-4. COMMENTS

Comments can be embedded anywhere in the source text (except within literal strings) by enclosure between (* and *), as in MACRO8000. For example:

```
    SEARCH LIB1(*I/O ROUTINES*), LIB2;
```

A percent sign comment is terminated by end of line, as in MACRO8000. For example:

```
    SEARCH LIB1, LIB2;       % LIB1 is I/O ROUTINES
```

2-5. DELIMITERS

Within statements, the standard delimiters are blanks, commas, and parentheses, as in MACRO8000. Blanks can be used freely in statements. Commas are used to separate operands. Parentheses are primarily used for lists.

NOTE

> LINK8000 additionally has brackets [and] that are used in forming sets.

The keywords BEGIN and END are special delimiters used in compound statements. The keywords THEN and ELSE are special delimiters used in IF directives. The keywords IN and DO are special delimiters used in FOR directives.

2-6. IDENTIFIERS

The identifiers that can be used in LINK8000 directives are:

Linker directives (predefined statement beginners)
Macro names (user-defined statement beginners)
File names (operands)
Module names (operands)
Segment names (operands)
Symbolic constants (operands)
Object variables (operands)

The identifiers are similar to MACRO8000 identifiers and can be as long as 80 characters. The characters A through Z, 0 through 9, underline, and @ can be used in an identifier, but an identifier cannot start with a digit or an underline. For example, valid symbols are:

```
DEX
FILE3
@B14INC
TEST_FOR_VALUE
```

2-7. STATEMENT BEGINNERS

The statement beginners are the identifiers that indicate the purpose of the statement. A statement beginner can be a linker directive or the name of a macro defined by the user.

2-8. DIRECTIVE NAMES

A directive is a special instruction to the linker. For instance, directives are used to specify input files and library names. For example:

```
FILE MOD1, MOD2;          % FILE directive is the statement beginner
```

The directives are statement beginners, but some directives are considered reserved words and some can be redefined. The directives CONST, VAR, MACRO, IF, and FOR are considered reserved words. The names of all the other directives can be redefined as macro names by the user.

2-9. MACRO NAMES

A macro is defined by the user with the MACRO directive (described later in this chapter). The macro name is an identifier. A macro must be defined before being referenced; that is, the macro definition must precede any references to the macro. For example:

```
MACRO HL7 PARM1;           % HL7 is defined as a macro
      BEGIN
      .
      .
      .
      END;
.
.
.
HL7 5;                     % HL7 macro name is the statement beginner
```

2-10. OPERANDS

In general, the operands in a statement always follow the statement beginner. For directives, the operands are values required for the directives. For macro references, the operands are the macro parameters.

2-11. FILE NAMES

The file names used as operands can be specified in the same way as file names in AMDOS commands. A complete file name has the general form:

```
dev:name.type
```

where dev is the drive designator such as A: or B:. The default is the same drive as for the directives file (for interactive input, the current drive)

where name is the file name consisting of 1 to 8 characters. Just as for AMDOS file names, the name can be * or can include ? wild card characters. The * indicates any name of any length; the ? indicates any character in that position

and where type is the file type (extension) consisting of 1 to 3 characters. Just as for AMDOS file names, the type can be * or can include ? "wild card" characters. The * indicates any type; the ? indicates any character in that position. The default is .ZRL for the file type

A complete file name, or any part of the full form, can be specified as an identifier or as a string enclosed in apostrophes. Any part that includes special characters (characters which cannot be used in an identifier) must be specified as a string. Therefore, any file name involving pattern matching with * or ? must be specified as a string.

The drive and extension can be specified or allowed to default. The : and . in the full form are effectively delimiters and can be used between the device, name, and extension. The following specifications are equivalent to using the full FILE A:PROG.ZRL; form:

```
    FILE PROG;              % using one identifier and defaults

    FILE 'PROG';            % using one string and defaults

    FILE A:PROG.ZRL;        % using three identifiers

    FILE A : PROG . ZRL;    % using three identifiers

    FILE 'A:PROG.ZRL';      % using one string

    FILE 'A:' & 'PROG.ZRL'; % using a concatenated string

    CONST DRIVE = A,
          NAME = PROG,
          TYPE = ZRL;

    FILE DRIVE:NAME.TYPE;   % using three symbolic constants
```

For more compact file name specifications, the user can take advantage of the AMDOS-type file specification. For instance:

```
    FILE '*';
```

indicates that all files with file type .ZRL are to be used as relocatable input. The files named X.ZRL and Y.ZRL would both be used. As another example, the linker directive:

```
    FILE  'RF?.ZRL';
```

indicates that all files with RF as the first two letters, any character (or no character) for the next position, and file type .ZRL are to be used. If present on the diskette, the files RFA.ZRL and RF8.ZRL would both be used. See the FILE directive in chapter 3.

2-12. NAMES OF MODULES AND SOFTWARE SEGMENT

The module names used as operands in linker directives are the module names assigned during MACRO8000 assemblies. The linker supports the use of a single module name or a list of module names. The software segment names used in the directives are just the segment names assigned during MACRO8000 assemblies.

The module names and segment names can be identifiers, strings, or (in certain cases, as noted below) as pattern strings containing the ? wild card character. Just as for file names, module names and segment names must be strings if the names include characters that cannot be used in identifiers. For example:

```
    COMBINE  'CRT_IO'.DATA;
```

specifies module name 'CRT_IO' and segment name DATA.

2-13. CONSTANTS

Numeric constants can be decimal, binary, octal, hexadecimal, variable base, or in K, just as for MACRO8000. For example:

```
    5
    11B
    642Q
    #6F
    4#123
    2K
```

are all valid numeric constants.

2-14. NUMERIC EXPRESSIONS

A numeric expression can be evaluated at link time to produce a 32-bit signed value, just as for MACRO8000. For example:

```
    5
    4K / 8
    5 * 4 + 1
    5 * (4 + 1)
```

are all valid numeric expressions.

2-15. LOGICAL EXPRESSIONS

A logical expression can be written in the IF directive for evaluation at link time. The logical expressions are the same as for MACRO8000. For example:

```
    NULL X
    URT OR SWITCH
    L123 LT 4
```

are all valid logical expressions, as long as they result in a true or false value.

2-16. STRINGS

Strings can be used for file names, module names, or segment names in a number of LINK8000 directives. Strings are specified just as in MACRO8000. In all cases, a string or string expression can be used. For example:

```
'ABCDEF'
'00A0'
'IT''S'
'B:' & 'ABCD' & '.ZRL'
```

are all valid strings.

2-17. LISTS

A list is a composite object that represents a grouping of items, just as for MACRO8000. The primary uses of a list are in the FOR statement, and in the COMBINE directive for a list of module names.

NOTE

In LINK8000, the operator & is extended and can be used for lists as well as strings.

The operator that can be used for building lists is:

Operator	Meaning	Example
&	Concatenate	A & (B, C, D)

The operators that can be used for manipulating lists are:

Operator	Meaning	Example
ATOM	ATOM Y is TRUE if Y is neither a list nor a set (that is, atomic) and FALSE otherwise	ATOM LIST
FIRST	Take and use the first item in a list; undefined if its argument is not a list	FIRST (X, Y, Z)
REST	Create a new list of all items in the list except the first element; undefined if the argument is not a list	REST (X, Y, Z)

For example, FIRST (X, Y, Z) has the value X, and REST (X, Y, Z) has the value (Y, Z). (When REST is applied to a single-element list, the result is NIL. For example, REST (A)=NIL.) For processing an item that might be a file name or a list of file names, the user might write:

```
VAR X: OBJECT;
IF ATOM PP
     THEN FILE PP        % uses the single file
     ELSE FOR X IN PP
          DO FILE X;     % uses each file in the list
```

Sublists are possible, where an item is itself a list. For example:

```
(X1, X2, (Y1, Y2))
```

2-18. SETS

Like a list, a set is a composite object containing items. A set is used to specify a group to be processed by the linker. Certain linker directives accept a specification of a set of files, modules, segments, or globals. A set with one item has the form:

```
[item]
```

A set with two or more items has the form:

```
[item,item...]
```

Each item in the set can be any valid operand, except another set or a list. For example, a set of module names might be:

```
[CRT_IN, CRT_OUT, CRT_STAT, PRINTER_OUT, PRINTER_STAT]
```

The operators that can be applied to sets are the Pascal operators:

Operator	Meaning	Example
+	Set union (combination of all elements)	S1 + S2
-	Set disunion (removal of selected elements)	S1 - S2
*	Set intersection (result containing only elements found in both sets)	S1 * S2

The empty set has the special value NIL.

NOTE

Sets, as well as lists, can be used in a FOR loop.

2-19. SYMBOLIC CONSTANTS

A symbolic constant is an identifier that represents a constant value during the linking process. Symbolic constants are declared by the CONST directive (described later in this chapter). The CONST declaration works just as in MACRO8000, and the same rules apply. For example:

```
    CONST DRIVE = B;
    •
    •
    •
    FILE DRIVE:XR.ZRL;          %value B:XR.ZRL
```

2-20. OBJECT VARIABLES

An object variable is an identifier that represents a variable value during the linking process. Object variables are declared with the VAR directive (described later in this chapter). The VAR declaration works just as in MACRO8000, and the same rules apply. For example:

```
    VAR LIB: OBJECT;            % LIB is initially undefined
    IF SWITCH
         THEN LIB ::= LITH      % LIB is defined as LITH
         ELSE LIB ::= LITH3;    % LIB is defined as LITH3
    SEARCH LIB;                 % uses either LITH or LITH3
```

NOTE

> For interactive input to the linker, the special value @INPUT is defined for object variables.

The special value @INPUT tells the linker to accept a value supplied from the console during a linking run. For example:

```
    VAR ANSWER: OBJECT;
    PRIN 'ENTER STACK SIZE: ';
    ANSWER ::= @INPUT;
    ABSOLUTE ($ + ANSWER);
```

2-21. LOCATION COUNTER

The special symbol $ represents the current value of the location counter. The location counter changes with the assignment of base addresses to the user segments. For example:

```
    CONST STACK_SIZE = #100;
    •
    •
    •
    ASSIGN STACK := ($ + STACK_SIZE)^;
                                % assigns a value #100 greater than the
                                % location counter to identifier STACK
                                % STACK is assumed to be an unsatisfied
                                % external.
    PRINT 'LOCATION COUNTER IS NOW: ', $;
```

2-22. GENERAL PURPOSE DIRECTIVES

A number of linker directives are the same as for MACRO8000 or similar to MACRO8000. These directives can be used as needed for the linking process and placed anywhere within the linker directives.

2-23. CONST DIRECTIVE

The CONST directive declares a symbolic constant (described earlier). A symbolic constant is an identifier that represents a constant value. The CONST directive has the same form as in MACRO8000. For example:

```
CONST ROM_OFFSET = #5000,
      CRTSET     = [CRT_IN, CRT_OUT, CRT_STAT],
      PRNSET     = [PRINTER_OUT, PRINTER_STAT],
      IOSET      = CRTSET + PRNSET;
```

2-24. VAR DIRECTIVE

The VAR directive declares an object variable (described earlier). The VAR directive has the same form as in MACRO8000. For example:

```
VAR NAME: OBJECT;          % declares NAME as object variable
NAME ::= XY;               % defines value of NAME
FILE NAME;                 % identical to FILE    XY;
```

2-25. IF DIRECTIVE

The IF directive can be used for conditional linking. In conditional linking, linking operations are performed or not performed depending on a particular condition. The IF directive has the same form as in MACRO8000.

NOTE

> For LINK8000, the IF statement is always effective at the time when the linking directives are being processed. In this respect, the LINK8000 IF is like the MACRO8000 assembly time IF and not like the MACRO8000 run time IF.

The test for conditional linking is a logical expression that can be:

TRUE or FALSE

An expression with the NULL operator and an object variable

An arithmetic comparison

A string comparison

A logical operation with NOT

A logical comparison with AND or OR

For example:

```
    IF SWITCH AND NOT NULL X
         THEN BEGIN
              SEARCH HRTLIB;
              SEARCH FOLLIB
              END
         ELSE SEARCH USERLIB4;
```

2-26. FOR DIRECTIVE

The FOR directive is used for repetitive linking, where linking directives are used repeatedly in a specific way. The FOR directive has the same form as for MACRO8000. For example:

```
    CONST SEGLIST = (DATA1, DATA2, DATA3);
    VAR X: OBJECT;
    •
    •
    •
    FOR X IN SEGLIST DO
         BEGIN
         SEGMENT X;
         COMBINE .X
         END;
```

NOTE

> EXIT can be used to terminate a FOR loop. EXIT must be used in the immediate context of the FOR loop. When EXIT is encountered, repetitive linking terminates and the linker moves on to the next statement after the FOR statement.

2-27. PAGE DIRECTIVE

The PAGE directive sets the size of each listing page in the .PRN file and has the same form as in MACRO8000. For example:

```
    PAGE 48;
```

2-28. EJECT DIRECTIVE

The EJECT directive causes a page eject in the .PRN file and has the same form as in MACRO8000. For example:

```
    EJECT;
```

2-29. MACRO DIRECTIVE

The MACRO directive declares a macro and takes the same form as in MACRO8000. In LINK8000, macros can be used for the expansion of linker directives. Each linker directive macro has the same general form as in MACRO8000, and the macro parameters work in the same way. For example:

```
    MACRO LOGICALSEGMENT SEGNAME, MODULELIST;
          BEGIN
          SEGMENT SEGNAME;
          COMBINE MODULELIST.DATA, MODULELIST.CODE
          END;
    •
    •
    •
    LOGICALSEGMENT SEG4, (MAB,MAD,MAF,MAH);
```

NOTE

EXIT can be used to terminate a macro. EXIT must be used in the immediate context of the macro. When EXIT is encountered, macro expansion terminates and the linker moves on to the next statement after the macro call.

2-30. PRINT DIRECTIVE

The PRINT directive is used to display one or more objects or operands at the console. The form is:

```
    PRINT objectsequence;
```

where objectsequence is a sequence of one or more operands or objects separated by commas.

Any strings to be displayed at the console are enclosed in apostrophes in the PRINT directive but displayed at the console without apostrophes. For example:

```
    ==> X ::= 16
    ==> PRINT X
    #00000010
    ==> Y ::= 'SIZE = '
    ==> PRINT Y
    SIZE =
    ==> PRINT Y,X
    SIZE = #0000010
    ==>
```

Note that PRINT displays the objects and then terminates the line with a carriage return/line feed sequence.

2-31. PRIN AND TERPRI DIRECTIVES

PRIN has the same form as PRINT:

```
    PRIN objectsequence;
```

but the display is not terminated with carriage return/line feed. The TERPRI directive has the form:

```
    TERPRI;
```

The TERPRI directive terminates the line and is normally used after one or more PRIN directives:

```
    ==> VAR Z: OBJECT
    ==> FOR Z IN (THIS, IS, A, LIST) DO PRIN Z,' ';TERPRI
    THIS IS A LIST
    ==> FOR Z IN (THIS, IS, A, LIST) DO PRINT Z
    THIS
    IS
    A
    LIST
    ==>
```

2-32. INCLUDE DIRECTIVE

The INCLUDE directive specifies a file containing additional linker directives and arguments to be used in the linking process. The contents of the file are substituted for the directive in the linker input. This directive can be nested to three levels. The INCLUDE directive has the same form as in MACRO8000. For example:

```
    INCLUDE 'IO_SET';
```

CHAPTER 3
FUNCTIONAL LINKER DIRECTIVES

The user supplies linker directives to describe the linking operation. The general purpose directives (described in chapter 2) can be interspersed with the functional linker directives described in this chapter. The linking operation itself involves directives to:

- Specify the basic type of link to produce a program, module, library or ROM library. A PROGRAM, MODULE, LIBRARY, or ROMLIB must be the first directive.

- Specify the relocatable input to be used for the linking operation. The relocatable input consists of files containing relocatable code, as well as any libraries needed for the linking operation. The linker must have access to the modules and segments before it can manipulate them.

- Specify and control the actual linking process. The user can set absolute addresses, define new relocatable software segments, combine software segments in an arbitrary way, assign software segments to AmZ8001 hardware segments, and assign absolute entry points to specific unsatisfied externals.

- Specify any additional output from the linker. For an absolute program, the user can generate an additional file (ROM library) containing entry points to absolute code in PROM. At any time during the linking run, the user can generate a variety of linker maps.

In the linker directives, the specifications of link type, relocatable input, linking control, and additional linker output should essentially be in the order shown. The one exception is that maps can be requested at any time.

Note that an AmZ8002 user does not necessarily need LINK8000 to produce absolute code suitable for PROM burning or downloading. A single monolithic program can simply be assembled through MACRO8000 to produce a hex file for PROM burning (H option on the MACZ call) or an AMC binary file for downloading (B option on the MACZ product call). In this case, PROGRAM is used as the first directive in the assembly program and not run through the linker.

A user targeting for the AmZ8001 must always use LINK8000 to produce absolute code suitable for PROM burning or down-loading. Only binary files can be produced; hex is not available for the AmZ8001.

If the user has a program structured into a number of interrelated parts, MACRO8000 and LINK8000 are used together to prepare the program.

In this case, MODULE is used as the first directive in each part of the program, and the assembler produces files containing relocatable code (O option on the MACZ product call).

3-1. TYPE OF LINK

The choice of the basic type of link is required as the first linker directive. In all cases, the user chooses one directive specifying the type of link and supplies it as the first directive. Therefore, the linker directives have the framework:

Program creation	Module creation	Library creation	ROM library creation
PROGRAM	MODULE	LIBRARY	ROMLIB
through	through	through	through
END.	END.	END.	END.

The special terminator END. is used, just as in MACRO8000, to mark the end of the linking run.

3-2. PROGRAM

The PROGRAM directive specifies the creation of an absolute program suitable for downloading. PROGRAM also specifies the main entry point, which must be a global label defined in one of the relocatable input modules. The PROGRAM directive has the form:

```
PROGRAM lab;
```

where PR is abbreviation of PROGRAM

and where lab is a label that specifies the main entry point of the program. The label can also be specified as a string or string expression

Appearance of a PROGRAM directive indicates that the user is linking relocatable code in order to produce absolute code.

The relocatable files produced by MACRO8000 and/or LINK8000 are used as input to LINK8000. From those input files targeted for the AmZ8002, the linker can produce either a hex (option H) or a binary (option B) output file. This output file can be used either for PROM burning or down-loading. Input files targeted for the AmZ8001 can produce only a binary output file. See figure 3-1. For a description of the hex and binary file formats, refer to Appendices B and C.

```
PROGRAM mode    _|
directives       |                                     _
(required)       |                                    |‾ optional
                 |                                    |  binary file
relocatable      | ------> LINK8000 -----------> |
input files      |                                    |  optional
(required)       |                                    |  hex file
                 |                                    |  (AmZ8002 only)
                 |                                    |
optional         |                                    |  optional
libraries to     |                                    |  ROM library
be searched     _|                                    |_ creation
```

Figure 3-1. PROGRAM Creation Run

The user has separate assemblies of modular program parts. The program parts reference each other in user-defined ways through the use of global and external declarations in separate modules. The result of the linking operation is simply to combine the separate modules and produce a coherent program.

See Chapter 4 and 5 for examples of program creation runs for the AmZ8002 and AmZ8001 processors.

3-3. MODULE

The MODULE directive specifies the creation of a combined module that is still relocatable. The MODULE directive has the form:

MODULE modname;

where MOD is the abbreviation of MODULE

and where modname is an identifier, a string, or a string expression to be used as the name of the combined module

The combined relocatable can be an intermediate step in the creation of an absolute program. In this case, the module can be used for an incremental linking operation. Incremental linking involves the condensation of a set of input modules into a single relocatable module that can be later combined with other single or combined relocatable modules. Effectively, incremental linking represents a succession of intermediate steps in the creation of a coherent program. In a final PROGRAM creation run, the program would be set up for downloading to an AmZ8002 processor. See figure 3-2.

```
MODULE mode      _
directives        |
(required)        |
                  |                                _
                  |                              |  combined
relocatable       | ------> LINK8000 ----------> |  relocatable
input files       |                              |_ module file
(required)        |
                  |
optional          |
libraries to      |
be searched      _|
```

Figure 3-2. MODULE Creation Run

The MODULE directive can be used in any case where it is desired to defer address resolution until a future time, for example, the case where a user operating system is handling the loading.

See Chapter 6 for an example of a module creation run.

3-4. LIBRARY

The LIBRARY directive specifies the creation of a library of relocatable routines that can be accessed in subsequent linking operations. The LIBRARY directive has the form:

LIBRARY libname;

where LIB is the abbreviation of LIBRARY

and where libname is an identifier, a string, or a string expression to be used as the library name

A library can be constructed from an arbitrary collection of modules or from subsets of other library files. The library as created contains:

A directory of globals and externals associated with the routines in the library

The library routines in relocatable form

The user can choose to create relocatable libraries and access them during creation of a particular program. In this case, the library can be used at different times for the creation of any number of particular programs. See figure 3-3.

```
LIBRARY mode      _
directives         |
(required)         |
                   |
relocatable        |  ------> LINK8000 -----------> |‾ relocatable
input files        |                                |_ library file
(required)        _|
```

Figure 3-3. LIBRARY Creation Run

See Chapter 7 for an example of library creation run.

3-5. ROMLIB

The ROMLIB directive specifies the creation of a ROM library index from subsets of other ROM libraries or from a set of explicitly-defined entry points assigned with the ASSIGN directive. The ROMLIB directive has the form:

```
ROMLIB rlibname;
```

where RLIB is the abbreviation of ROMLIB

and where rlibname is an identifier, a string, or a string expression to be used as the ROM library name

The ROMLIB directive is not normally used for the initial ROM library creation (although it can be done using the ASSIGN directive). Note that a ROM library index can be created with the R option as additional linker output during a PROGRAM run. See figure 3-4. The library index produced can be used to access a ROM resident library or used as input for symbolic debugging.

```
ROMLIB mode       _
directives         |
(required)         |
                   |
relocatable        |  ------> LINK8000 -----------> |‾ new
ROM library        |                                |  ROM library
files (from        |                                |_ file
PROGRAM runs)     _|
```

Figure 3-4. ROMLIB Creation Run

See Chapter 8 for an example of ROM library creation run.

3-6. HEADER

The HEADER directive supplies one or more header lines at the beginning of the relocatable output file (file type .ZRL by default) produced as the main linker output of a MODULE, LIBRARY, or ROMLIB creation run. The HEADER directive has the form:

```
HEADER strings;
```

where HDR is the abbreviation of HEADER

where strings is a sequence of strings separated by commas

and where each string is a string or string expression that produces one identification line in the .ZRL file

The HEADER directive is highly recommended for supplying any additional information the user wishes at the beginning of the .ZRL file. When a .ZRL file is listed at the console with an AMDOS TYPE or DISPL command, the header lines are displayed as part of the first block of information in the .ZRL file. The other information in the relocatable file is not displayed, unless the user dumps the entire file with the AMDOS DUMP command.

For example:

```
HEADER 'I/O routines';
```

3-7. RELOCATABLE INPUT

A number of linker directives specify the relocatable input to be used for the linking operation. The directives that can be used are:

Program creation	Module creation	Library creation	ROM library creation
FILE	FILE	FILE	FILE
SEARCH	SEARCH		
ATTACH	ATTACH		
DETACH	DETACH		

These directives are described in the following paragraphs.

3-8. FILE

One or more FILE directives specify the .ZRL files that are to be used as relocatable input to the linking operation. The .ZRL files can be assembled MACRO8000 modules, combined modules, libraries, or ROM libraries. The FILE directive has the form:

```
    FILE filesequence;
```

where FL is the abbreviation of FILE

where filsequence is a list of one of more file specifications separated by commas.

and where each file specification in the filesequence has one of the forms:

name — A relocatable module file. The name can be in the form dev:name.ext. The default drive is the current drive, and the default file type is .ZRL

pattern — A file name pattern of the AMDOS type, where the name can contain * as a general specification for any file name or file type, and where ? can be used as a wild card for any individual character in the file name or file type.

lib — A relocatable library or ROM library file. The name can be in the form dev:name.ext. The default drive is the current drive, and the default file type is .ZRL.

lib * set — A relocatable library file name followed by * and set of module names. The entry points to be used are restricted to the ones that are both in the library and in the set of modules specified

lib - set — A relocatable library file name followed by - and a set of module names. The entry points to be used are the ones in the library, except that entry points in the specified set of modules are omitted

The set of module names can be any set or parenthesized set expression (see SETS, chapter 2). In this context, a module name in a non-empty set can be an identifier, string, or a pattern string containing the ? wild card character.

A standard example of the FILE directive for relocatable input is:

```
    FILE X,Y;           % specifies relocatable files X.ZRL and Y.ZRL
```

Some examples using patterns for the file names are:

```
    FILE '*.ZRL';           % might specify files A.ZRL, B.ZRL, and F.ZRL
                            % that exist on the current drive

    CONST ALL = '*';
    FILE ALL;               % same as FILE '*.ZRL'
```

When the FILE directive is used for a library, all the routines in the library are used as relocatable input to the linker. For a library search, see the SEARCH directive. An example of the FILE directive for a library might be:

```
    FILE DEF;               % specifies library file DEF.ZRL
```

An additional feature of the FILE directive for a library is the ability to restrict entry points to be used or to omit selected entry points. The special forms of the lib specification are:

```
    FILE DEF * [DEF3];        % specifies library DEF3, restricting the
                              % entry points to the ones also contained in
                              % the module DEF3.ZRL

    FILE DEF - ['DEF?'];      % specifies library DEF.ZRL but omits the
                              % entry points in modules designated by the
                                pattern 'DEF?'.
```

3-9. SEARCH

The SEARCH directive accesses the library specified by providing access to the entry points in the library routines. The SEARCH directive initiates the process of satisfying any outstanding externals from the modules in the library. The SEARCH directive has the form:

```
    SEARCH libsequence;
```

where SR is the abbreviation of SEARCH

and where libsequence is a sequence of one or more relocatable library and/or ROM library files separated by commas. The library name has the form dev:name:ext. The defaults are current drive and file type .ZRL.

The SEARCH directive for a library simply uses the library directory to satisfy externals. Note that the SEARCH directive is equivalent to an ATTACH immediately followed by a DETACH directive. Therefore, the library is accessed and then released.

For a library, a SEARCH is different from a FILE. Using the FILE directive for a library moves in all of the library routines as relocatable input; using the SEARCH directive moves in only those routines required to satisfy outstanding externals.

For example:

```
    SEARCH XREF.ZRL;
```

When two or more files are specified in on e SEARCH directive (e.g., SEARCH LIB1,LIB2), any new externals introduced by a module in one library file can be satisfied by entry points in modules in the other library. In other words, all libraries in a sequence specified in a single SEARCH directive are ATTACHed before the corresponding DETACH.

3-10. ATTACH

The ATTACH directive accesses the library specified and leaves the library attached until a subsequent DETACH directive is encountered. The ATTACH directive has the form:

```
    ATTACH libsequence;
```

where AT is the abbreviation of of ATTACH

and where libsequence is a sequence of one or more relocatable libraries and/or ROM libraries separated by commas. The library name has the form dev:name.ext. The defaults are the current drive and file type .ZRL

For example:

```
    ATTACH XREF.ZRL;
```

3-11. DETACH

The DETACH directive detaches the specified library. The DETACH directive is only used for a library accessed with ATTACH. The DETACH directive has the form:

```
    DETACH libsequence;
```

where DT is the abbreviation of DETACH

and where libsequence is a sequence of one or more relocatable libraries and/or ROM libraries separated by commas. The library name has the form dev:name.ext. The defaults are the current drive and file type .ZRL

For example:

```
    DETACH XREF.ZRL;
```

DETACH removes (from the linker symbol table) all global and external labels and module names associated with the specified files. The reclaimed symbol table space may then be used in subsequent SEARCHs and ATTACHs.

3-12. LINKING CONTROL

After the type of link has been chosen and the relocatable input has been specified, the user can specify the linking operation itself in detail with a set of linking control directives. The number of control directives that can be used depends on the type of link chosen and whether the target processor is an AmZ8001 or AmZ8002.

The directives that can be used are:

PROGRAM link (AmZ8001)	PROGRAM link (AmZ8002)	MODULE link	LIBRARY link	ROMLIB link
OFFSET	ABSOLUTE	SEGMENT	RETAIN	RETAIN
SEGMENT	ASSIGN	COMBINE	OMIT	OMIT
SETLSEG	COMBINE	RETAIN	MAP	MAP
ASSIGN	XSPACE	OMIT		
COMBINE	RETAIN	MAP		
XSPACE	OMIT			
RETAIN	MAP			
OMIT				
MAP				

These directives are described in the following paragraphs.

3-13. ABSOLUTE

The ABSOLUTE directive can be used to specify an absolute destination address. This directive may be used only during a PROGRAM link that is targeted for an AmZ8002 processor. (For AmZ8001 links, the equivalent directive is OFFSET.) This directive has the following form:

```
ABSOLUTE exp;

     where ABS is the abbreviation of ABSOLUTE

     and where exp is a numeric expression (usually hexadecimal)
     specifying a memory address
```

In program creation runs, the user needs to assign a starting location for the linked code. The user can also use ABSOLUTE directives to specify a number of destination addresses. For example, the user might assign absolute addresses to all globals in the program.

ABSOLUTE directives may also be intermixed with COMBINE directives to define the placement of all program code and data by assigning a destination address to each segment in the relocatable input.

For example:

```
    ABSOLUTE #4200;
    COMBINE .CODE;          % all code segments
    ABSOLUTE #5000;
    COMBINE .DATA;          % all data segments
```

The ABSOLUTE/COMBINE combination of directives in a program run corresponds to the SEGMENT/COMBINE directives in a module creation run.

3-14. OFFSET

This directive is the AmZ8001 equivalent to the ABSOLUTE directive. OFFSET is used to specify an 16-bit address offset in the current hardware segment. (See section 3-16 SETLSEG for a discussion of the current hardware segment.)

In one special case OFFSET can be used to specify a hardware segment number as well as the offset. If OFFSET is used to specify an entry point of an attached ROMLIB, then the hardware segment number of the entry point address is assigned to the current output segment via an implicit invocation of SETLSEG. (The current hardware segment number counter is unaffected.)

This directive may be used only during a PROGRAM link that is targeted for an AmZ8001 processor. It has the following form:

```
    OFFSET exp;
```

where OFF is the abbreviation for OFFSET

and where exp is an expression specifying a 16-bit offset into the current hardware segment or an entry point in an attached ROMLIB.

For example, to specify an offset value of 1000 hex in the current hardware segment, the following linker directive would be used:

```
    OFFSET #1000;
```

3-15. SEGMENT

The SEGMENT directive can be used in PROGRAM links for the AmZ8001 or in MODULE links for both processors. In both links, the directive defines the name of an output software segment for incremental linking. The SEGMENT directive has the following form:

```
SEGMENT segname;

SEGMENT [attr], segname;

SEGMENT @PRIOR;
```

where SEG is the abbreviation of SEGMENT

where segname is the segment name

where attr is an optional segment attribute: @COM common common (MODULE links only)

and where @PRIOR indicates a reset to the segment previously defined

The segment name can either be an identifier or a string. The common attribute (@COM) will force the assignment of segments (with the same name) in different modules to a common memory space. Common segments in different modules should have the same size. A common segment is analogous to a Fortran common block.

The set of input segments from the input modules have no necessary relationship with the set of output segments defined by the SEGMENT directive. For example, a set of input segments with the common attribute may be combined into a single output segment:

```
SEGMENT COMDATA;
COMBINE .COMBLOCK;
```

The special indicator @PRIOR is used to reset to the previously defined (that is, prior) segment. In this case, the segment offset is set to the value last assigned. For example:

```
SEGMENT @PRIOR;
```

If no prior segment exists, SEGMENT @PRIOR generates an informative error and the current segment is used.

3-16. SETLSEG

This directive allows for explicit assignment of hardware segment numbers to the software segment names defined in the SEGMENT directive. SETLSEG can be used only in PROGRAM links for the AmZ8001. If a SETLSEG directive is not given, the linker will assign consecutive hardware segment numbers by default. The linker maintains an internal hardware segment number counter in order to implement this directive. The directive has the form:

```
    SETLSEG;

    or

    SETLSEG assignment_sequence;
```

where assignment_sequence is a series of expressions separated by commas, each with the format

```
segment_name := exp
or
segment_name
```

where segment_name is a software segment name and where exp is an arithmetic expressions for the hardware segment number with value 0-127.

The SETLSEG directive with no arguments simply assigns segment numbers to the segment names in alphabetical order, beginning with the last assigned number (initially zero).

The assignment expression segment_name := exp sets the internal segment number counter to the value exp, then assigns that value to the segment named segment_name. The assignment expression segment_name assigns the current value of the segment number counter to segment_name. After each assignment expressions, the linker increments the segment number counter. Default segment number assignments begin with the most recent segment counter value.

For example:

```
    SETLSEG IOTASK  := 1,DATABASETASK,SPACEMGRTASK;
```

assigns the software segment name IOTASK to hardware segment number 1, DATABASETASK to 2, and SPACEMGRTSK to 3. Subsequent SETLSEG directives will assign hardware segment numbers from 4 up, unless reset higher by an explicit assignments.

Hardware segment number assignments must be strictly ascending.

3-17. COMBINE

The COMBINE directive specifies the combination of relocatable input segments (either from a MACRO8000 assembly or a previous linker run) to absolute addresses in the linker output. For program creation, COMBINE directs the assignment of modules and/or segments to absolute addresses (ABSOLUTE directive). For module creation, COMBINE directs the assignment of segments to a named output segment (SEGMENT directive). The COMBINE directive has the form:

```
    COMBINE;

    COMBINE BY MODULE;
```

COMBINE BY SEGMENT;

COMBINE segsequence;

where CMB is the abbreviation of COMBINE

and where segsequence is an optional sequence of one or more segment sepcifications separated by commas. Each segment specification can take one of the forms:

mod	specifies a module name or list of module names enclosed in parentheses. In this form, each module name must be an identifier or string.
.seg	specifies . followed by the segment name. In this form, the segment name must be an identifier or string.
mod.seg	specifies a module name, or list of module names enclosed in parentheses, followed by . and the segment name. In this form, each module name can be an identifier, a string, or a pattern string containing the ? wild card character.

When COMBINE is used with no operands, COMBINE means COMBINE BY MODULE and specifies the grouping by module of the sequence of input segments in the module.

The basic scheme for COMBINE with no operands or COMBINE BY MODULE is:

```
Relocatable input                COMBINE or COMBINE BY MODULE

Module X Segment CODE --------> X.CODE
Module X Segment DATA --------> X.DATA
Module Y Segment CODE --------> Y.CODE
Module Y Segment DATA --------> Y.DATA
Module Z Segment CODE --------> Z.CODE
Module Z Segment DATA --------> Z.DATA
```

The basic scheme for COMBINE BY SEGMENT is:

```
Relocatable input                COMBINE BY SEGMENT

Module X Segment CODE --------> X.CODE
Module X Segment DATA _ _ _ _-> Y.CODE
Module Y Segment CODE ->< _ _ > Z.CODE
Module Y Segment DATA _ _ >< -> X.DATA
Module Z Segment CODE ->< _ _-> Y.DATA
Module Z Segment DATA --------> Z.DATA
```

When the COMBINE directive specifies a segsequence, the user is combining segments in an arbitrary way. For example:

```
    COMBINE .DATA;      % combines all segments with the name DATA

    COMBINE SYMBOL_TABLE.DATA, HASH_TABLE.DATA;
                        % combines segments with the names sepcified

    COMBINE (SYMBOL_TABLE, HASH_TABLE).DATA;
                        % same as the previous example

    CONST PARSER = (SCAN, NEXT_CHAR, FIND, ENTER, PURGE);
    .
    .
    .
    SEGMENT 'PARSER';   % defines the relocatable output segment
    COMBINE PARSER.DATA, PARSER.CODE;  % uses the symbolic constant
                        % PARSER to specify the list of module names
```

If COMBINE has been used to combine some but not all segments, COMBINE with no operands can be used to combine all remaining unassigned segments. The user should request one or more link maps after every COMBINE in order to manage this process effectively. If any input segments remain unassigned at the end of the directive sequence (at END.), an implicit COMBINE will be invoked.

3-18. XSPACE

The status lines of the AmZ8000 can be used to extend the address space. For example, the status lines distinguish a data access from an instruction access, thus allowing instructions and data to be stored in different address spaces, if the memory hardware can decode the status signals.

LINK8000 supports this extended address space concept through the XSPACE directive. An XSPACE directive can appear only once in such links. All program segments COMBINEd after the XSPACE directive are assigned to extended space. For example:

```
    COMBINE   .CODE;
    XSPACE;
    COMBINE   .DATA;
```

assigns all .CODE segments to regular address space and all .DATA segments to extended space.

XSPACE can be used with both AmZ8002 and AmZ8001 code. The only output file formed available is AMC binary format (option B). The directive causes the location counter to be reset to zero. For the AmZ8001, any previously unassigned output program segments are assigned hardware segment numbers (via an implicit call to the SETLSEG directive).

The output file has several distinctive record types if XSPACE is used in the link. These additional types hold the extended space values; they are documented in Appendix C. Users must write software to read the binary file and select the extended space record types. Users must also write software to load the extended space information into the correct memory locations, a procedure that will depend on how the user has decoded the status lines. Contact your AMD Field Application Engineer for additional information and support.

3-19. ASSIGN

The ASSIGN directive causes the assignment of absolute addresses to identifiers. The ASSIGN directive has the form:

```
ASSIGN  assignmentsequence;
```

and where assignmentsequence is a list of assignments separated by commas. Each assignment takes the form:

lab := abs — specifies a label that is set to the absolute address constant specified. The label cannot be a symbolic constant or object variable. The absolute address constant is either a single arithmetic value (AmZ8002) followed by ^, or a pair of values followed by ^, representing a segment number and an offset (AmZ8001).

Constant is either a single arithmetic value (AmZ8002) or a pair of values, representing a segment number and an offset (AmZ8001).

For example (AmZ8002):

```
CONST STACK_SIZE = #500;
•
•
•
ASSIGN CRT_STAT := #FF20^,
       CRT_IN :=   #FF46^,
       CRT_OUT :=  #FF80^,
       STACK := ($ + STACK_SIZE)^;
```

(AmZ8001):

```
ASSIGN LAST_ROM_LOC := (3,#2000)^;
```

For program creation, ASSIGN can be used to assign addresses for any unsatisfied externals. For a ROM library creation, ASSIGN causes the creation of global entry points with absolute addresses.

In the program mode, the ASSIGN directive may also be used to assign (previously entered) globals to unsatisfied externals. For example, if DISK_READ_TEMP and DISK_READ_DIAG are entry points of a (previously entered) module, then

```
    IF DIAG THEN
         ASSIGN DISK_READ := DISK_READ_DIAG

    ELSE
         ASSIGN DISK_READ := DISK_READ_DIAG
```

will equate DISK_READ to one of two entry points depending on the value of DIAG.

3-20. RETAIN AND OMIT

The RETAIN directive is used to retain a record of all or some globals in the program. The RETAIN directive has the form:

```
    RETAIN  globalsequence;
```

where RET is the abbreviation of RETAIN

and where globalsequence is an optional sequence of globals separated by commas

The OMIT directive has the form:

```
    OMIT  globalsequence;
```

where globalsequence is a sequence of globals separated by commas

A global may be specified as an identifier, a string, or a pattern string containing the ? wild card character.

The RETAIN directive retains the selected globals and omits the rest. The OMIT directive is used to omit selected globals but retain the rest. RETAIN and OMIT are opposite in meaning, and the user chooses the more convenient directive to use.

For a program run with the R option, RETAIN or OMIT causes creation of a ROM library with a selected subset of (assigned) global entry points. The ROM library is a directory of entry points to the absolute code produced in the program run. Unlike a library, the ROM library contains only the directory and not the library routines themselves. The library entry points themselves are the main output of the link and are in absolute form.

For a module run, RETAIN or OMIT affects the main linker output and selects a subset of entry points in the relocatable output. The entry points that remain, or are not omitted, can be used in a subsequent linking operation. For incremental linking, the entry points that were excluded are effectively hidden and are not available in any subsequent link. This technique can be used to protect the integrity of selected parts of the program. It can also be used to reduce the size of the symbol table required for linking very large, modular programs.

For a library run, RETAIN or OMIT affects the main output by restricting the set of globals that can be used to satisfy externals.

For a ROM library run, RETAIN or OMIT affects only the directory and specifies a subset of the previously defined entry points for the ROM libraries.

3-21. MAP

This directive produces a sorted list of globals, showing assigned addresses, segment names, any unresolved externals, and other significant information. For the AmZ8001, assigned segment numbers are displayed as a pair of hex digits (if the segment number has been assigned at the time the MAP directive was executed). The directive has the following form:

```
MAP;

MAP BY option;
```

the map options are:

LABEL (same as MAP;) shows symbols, addresses, module names, and segment names, in the same order in which the relocatable files were accessed. For unassigned symbols, the address is an offset marked with *

MODULE shows module names, mod.seg, sizes, and assigned addresses, as well as unit sizes, in a list sorted by module name

SEGMENT shows segments names, mod.seg, sizes, and assigned addresses, as well as unit sizes, in a list sorted by segment name

ADDRESS shows symbols and addresses in a list sorted by address. Because of space constraints, this map destroys some of the information in the MAP BY LABEL display. Therefore, this map should be the last map requested. (For a large number of entry points, there may be a noticeable delay.)

LIBRARY shows library entry points

LIB_MOD shows library modules (LIBRARY run only)

OUT_SEG shows output segments (MODULE run only)

EXTERNAL shows symbols currently unassigned

CHAPTER 4
A SAMPLE PROGRAM RUN (AmZ8002)

This chapter contains a sample PROGRAM run for the AmZ8002 processor. Separate parts are assembled as modules and then linked together to produce absolute output. The absolute output is a binary file which is then downloaded to the AmZ8002 evaluation board and executed.

```
A>MACZ BUBEXEC 0 ←—— Produce relocatable .ZRL file


BUBEXEC                                   MACRO8000 AMZ8000 ASSEMBLER    1.0.1


0000
0000                                      MODULE    'BUBEXEC';
0000
0000                                      GLOBAL   START;
0000
0000                                      EXTERNAL  PROMPT;
0000                                      EXTERNAL  READ;
0000                                      EXTERNAL  SORT;
0000                                      EXTERNAL  WRITE;
0000                                      EXTERNAL  EXIT;
0000                      main entry Point
0000                                      SEGMENT  'CODE';  ←— Program Segment
0000                             → START:
0000     5F00*0000                        CALL      PROMPT;
0004     5F00*0000                        CALL      READ;
0008                                      (* R3 CHAR COUNT FROM READ *)
0008     0B03 0001                        IF R3 LE 1 THEN
000C     EA02 5E08*0000                             JP        START;
0012     5F00*0000                        CALL      SORT;
0016     5F00*0000                        CALL      WRITE;
001A     5F00*0000                        CALL      EXIT;
001E
001E                                      END.
```

```
A>MACZ BUBPRPT 0


BUBPRPT                                   MACRO8000 AMZ8000 ASSEMBLER    1.0.1


0000
0000                                      MODULE   'BUBPRPT';
0000
0000                                      GLOBAL   PROMPT;
0000
0000                                      EXTERNAL BUFLEN, BUFFER;
0000                                      EXTERNAL WRITE;
0000
0000                                      SEGMENT 'CODE';
0000                              PROMPT:
0000    2104*0000                         LD       R4, ↑BUFFER;
0004    2105*0020                         LD       R5, ↑MESSAGE + 1;
0008    6106*001E                         LD       R6, MESLEN;
000C                                      (* MOVE MESSAGE INTO BUFFER *)
000C    BA51 0640                         LDIRB    R4↑, R5↑, R6;
0010                                      (* SET BUFFER LENGTH *)
0010    6106*001E                         LD       R6, MESLEN;
0014    6F06*0000                         LD       BUFLEN, R6;
0018    5F00*0000                         CALL     WRITE;
001C    9E08                              RET;
001E                              MESLEN:
001E    00                                BYTE:    0;
001F                              MESSAGE:
001F                                      STRING:  'ENTER CHARACTERS TO BE ',
001F    2B45 4E54 4552                             'SORTED, THEN RETURN ';
0025    2043 4841 5241
002B    4354 4552 5320
0031    544F 2042 4520
0037    534F 5254 4544
003D    2C20 5448 454E
0043    2052 4554 5552
0049    4E20
004B
004B                                      END.

NEITHER WARNING NOR ERROR MESSAGES
```

entry point →

← message length accessed as a word value ØØ2B

```
A>MACZ BUBSORT O

BUBSORT                                   MACRO8000 AMZ8000 ASSEMBLER    1.0.1


0000
0000                                      MODULE  'BUBSORT';
0000
0000                                      GLOBAL  SORT;
0000
0000                                      EXTERNAL BUFLEN, BUFFER;
0000
0000                                      CONST   LAST     = R2, ← symbolic
0000                                              THIS     = R3,   constants
0000                                              NEXT     = R4,   used to
0000                                              LAST←CH  = RH5,  designate
0000                                              THIS←CH  = RL5,  working registers
0000                                              COUNT    = R6,
0000                                              SWAPS    = R7;
0000                   entry point
0000                                      SEGMENT 'CODE';
0000                         →  SORT:
0000                                      (* INITIALIZE FOR SORT *)
0000    2107 0000                         LD      SWAPS, 0;
0004                                      (* INITIALIZE POINTERS *)
0004    2103*0000                         LD      THIS, ↑BUFFER;
0008    A134                              LD      NEXT, THIS;
000A    A940                              INC     NEXT, 1;
000C                                      (* ADJUST WORKING COUNT *)
000C    6106*0000                         LD      COUNT, BUFLEN;
0010    AB60                              DEC     COUNT, 1;
0012                                      (* CY LATER USED FOR OV *)
0012    8D83                              RESFLG  CY;
0014                            LOOK:
0014    BA46 063B                         CPSIRB  THIS↑, NEXT↑, COUNT, LGT;
0018                                      IF OV (* AT END OF STRING *) THEN
0018    EC01 8D81                                 SETFLG  CY;
001C                                      IF ZR (* SWAP NEEDED *) THEN
001C    EE07                                      BEGIN
001E                                              (* GET POINTER TO LAST *)
001E    A132                                      LD      LAST, THIS;
0020    AB20                                      DEC     LAST, 1;
0022                                              (* SWAP LAST AND THIS *)
0022    2025                                      LDB     LAST←CH, LAST↑;
0024    203D                                      LDB     THIS←CH, THIS↑;
0026    2E2D                                      LDB     LAST↑, THIS←CH;


0028    2E35                                      LDB     THIS↑, LAST←CH;
002A                                              (* INCREMENT SWAP COUNTER *)
002A                                              INC     SWAPS, 1
002A    A970                                      END;
```

```
BUBSORT                                     MACRO8000 AMZ8000 ASSEMBLER    1.0.1


0020                                        IF NC (* NOT AT END OF LINE *) THEN
002C    E701 E8F2                                   JR       LOOK;
0030    8577                                IF SWAPS NE 0 (* CHANGES MADE *) THEN
0032    E601 E8E5                                   JR       SORT;
0036    9E08                                RET;
0038
0038                                        END.

NEITHER WARNING NOR ERROR MESSAGES


A>MACZ MTRREAD O


MTRREAD                                     MACRO8000 AMZ8000 ASSEMBLER    1.0.1


0000
0000                                        MODULE   'MTRREAD';
0000
0000                                        GLOBAL   READ;
0000
0000                                        EXTERNAL BUFLEN, BUFFER;
0000
0000                                        SEGMENT 'CODE';
0000                  entry point →  READ:
0000                                        (* SET UP CALL BLOCK *)
0000    4D05*0000 0100                      LD         CALL←BLOCK, #0100;
0006    4D05*0002 0000                      LD         CALL←BLOCK(2), 0;
000C    2102*0000                           LD         R2, ↑BUFFER;
0010    6F02*0004                           LD         CALL←BLOCK(4), R2;
0014    4D05*0006 0050                      LD         CALL←BLOCK(6), 80;
001A                                        (* SET UP POINTER *)
001A    2101*0000                           LD         R1, ↑CALL←BLOCK;
001E    7F00                                SC         0;
0020                                        (* GET, ADJUST, SAVE COUNT *)
0020    6103*0006                           LD         R3, CALL←BLOCK(6);
0024    AB30                                DEC        R3, 1;
0026    6F03*0000                           LD         BUFLEN, R3;
002A    9E08                                RET;
0000
0000                                        SEGMENT [@COM], 'CALLBLK';
0008                                CALL←BLOCK:
0008                                        WORD       (4);
0008
0008                                        END.

NEITHER WARNING NOR ERROR MESSAGES
```

↑ common program segment known among monitor routines

```
A>MACZ MTRWRIT O


MTRWRIT                            MACRO8000 AMZ8000 ASSEMBLER   1.0.1


0000
0000                               MODULE   'MTRWRIT';
0000
0000                               GLOBAL   WRITE;
0000
0000                               EXTERNAL BUFLEN, BUFFER;
0000
0000                entry point    SEGMENT 'CODE';
0000                     --> WRITE:
0000                               (* SET UP CALL BLOCK *)
0000   4D05*0000 0200              LD       CALL←BLOCK, #0200;
0006   4D05*0002 0000              LD       CALL←BLOCK(2), 0;
000C   2102*0000                   LD       R2, ↑BUFFER;
0010   6F02*0004                   LD       CALL←BLOCK(4), R2;
0014                               (* ADD CR/LF AT END OF BUFFER *)
0014   6103*0000                   LD       R3, BUFLEN;
0018   C80D                        LDB      RL0, #0D;
001A   7228 0300                   LDB      R2↑(R3), RL0;
001E   A930                        INC      R3, 1;
0020   C80A                        LDB      RL0, #0A;
0022   7228 0300                   LDB      R2↑(R3), RL0;
0026                               (* STORE ACTUAL COUNT *)
0026   A930                        INC      R3, 1;
0028   6F03*0006                   LD       CALL←BLOCK(6), R3;
002C                               (* SET UP POINTER *)
002C   2101*0000                   LD       R1, ↑CALL←BLOCK;
0030   7F00                        SC       0;
0032   9E08                        RET;
0000
0000                               SEGMENT [@COM], 'CALLBLK';
0008                         CALL←BLOCK:             ↑
0008                               WORD     (4);     common segment
0008
0008                               END.

NEITHER WARNING NOR ERROR MESSAGES
```

```
A>MACZ MTREXIT O


MTREXIT                         MACRO8000 AMZ8000 ASSEMBLER    1.0.1


0000
0000                            MODULE   'MTREXIT';
0000
0000                            GLOBAL   EXIT;
0000
0000                            SEGMENT  'CODE';
0000                    EXIT:
0000    7FA1                    SC       161;
0002
0002                            END.

NEITHER WARNING NOR ERROR MESSAGES
```

entry point →

```
A>MACZ MTRDATA O


MTRDATA                         MACRO8000 AMZ8000 ASSEMBLER    1.0.1


0000
0000                            MODULE   'MTRDATA';
0000
0000                            GLOBAL   BUFLEN, BUFFER;
0000
0000                            SEGMENT  'DATA';
0002                    BUFLEN:
0002                            BYTE     (2);
0052                    BUFFER:
0052                            BYTE     (80);
0000
0000                            SEGMENT  [@COM], 'CALLBLK';
0008                    CALL←BLOCK:
0008                            WORD     (4);
0008
0008                            END.

NEITHER WARNING NOR ERROR MESSAGES
```

↑ common segment

```
A>LNKZ * B=LNKPR
LINK8000 * PRE-RELEASE TEST VERSION (1/22/80)
===> PROGRAM START
===> FILE 'BUB*'

 ENTER MODULE: BUBEXEC
 ENTER MODULE: BUBPRPT
 ENTER MODULE: BUBSORT

===> FILE 'MTR*'

 ENTER MODULE: MTRDATA
 ENTER MODULE: MTREXIT
 ENTER MODULE: MTRREAD
 ENTER MODULE: MTRWRIT

===> ABSOLUTE #5000
===> COMBINE BUBEXEC
===> MAP
BUFFER          0002*          MTRDATA     .DATA
BUFLEN          0000*          MTRDATA     .DATA



EXIT            0000*          MTREXIT     .CODE
PROMPT          0000*          BUBPRPT     .CODE
READ            0000*          MTRREAD     .CODE
SORT            0000*          BUBSORT     .CODE
START           5000           BUBEXEC     .CODE
WRITE           0000*          MTRWRIT     .CODE
===> COMBINE
===> MAP
BUFFER          50AC           MTRDATA     .DATA
BUFLEN          50AA           MTRDATA     .DATA
EXIT            50FC           MTREXIT     .CODE
PROMPT          501E           BUBPRPT     .CODE
READ            50FE           MTRREAD     .CODE
SORT            506A           BUBSORT     .CODE
START           5000           BUBEXEC     .CODE
WRITE           512A           MTRWRIT     .CODE
===> MAP BY MODULE
BUBEXEC
     BUBEXEC.CODE          001E     5000
     UNIT SIZE = 001E
BUBPRPT
     BUBPRPT.CODE          004C     501E
     UNIT SIZE = 004C
BUBSORT
     BUBSORT.CODE          0038     506A
     UNIT SIZE = 0038
```

```
LNKZ * B=LNKPB:LNKPR
LINK8000:            VERSION 2.0,  10/13/80

===> PROGRAM START
===> FILE 'B:BUB????'
 ENTER MODULE: BUBEXEC
 ENTER MODULE: BUBPRPT
 ENTER MODULE: BUBSORT
===> FILE 'B:MTR????'
 ENTER MODULE: MTRDATA
 ENTER MODULE: MTREXIT
 ENTER MODULE: MTRREAD
 ENTER MODULE: MTRWRIT
```

← get relocatable input from .ZRL files

```
===> ABSOLUTE #5000
===> COMBINE .CODE
===> COMBINE .CALLBLK
===> MAP
```

← Put all .CODE segments and the .CALLBLK segment in normal space

```
  ENTRY POINT   ADDRESS     MODULE       .SEGMENT

  BUFFER         0002*      MTRDATA      .DATA
  BUFLEN         0000*      MTRDATA      .DATA
  EXIT           50A2       MTREXIT      .CODE
  PROMPT         501E       BUBPRPT      .CODE
  READ           50A4       MTRREAD      .CODE
  SORT           506A       BUBSORT      .CODE
  START          5000       BUBEXEC      .CODE
  WRITE          50D0       MTRWRIT      .CODE

              *  INDICATES OFFSET
===> XSPACE
===> COMBINE .DATA
===> MAP
```

← Put all .DATA segments in extended space

```
  ENTRY POINT   ADDRESS     MODULE       .SEGMENT

  BUFFER         0002X      MTRDATA      .DATA
  BUFLEN         0000X      MTRDATA      .DATA
  EXIT           50A2       MTREXIT      .CODE
  PROMPT         501E       BUBPRPT      .CODE
  READ           50A4       MTRREAD      .CODE
  SORT           506A       BUBSORT      .CODE
  START          5000       BUBEXEC      .CODE
  WRITE          50D0       MTRWRIT      .CODE
```

} ← note "X" in address

```
===> MAP BY MODULE
```

```
  MODULE                   SIZE    ADDRESS

  BUBEXEC
       BUBEXEC.CODE        001E    5000
     UNIT SIZE = 001E
  BUBPRPT
       BUBPRPT.CODE        004C    501E
     UNIT SIZE = 004C
  BUBSORT
       BUBSORT.CODE        0038    506A
     UNIT SIZE = 0038
  MTRDATA
       MTRDATA.CALLBLK     0008    5104
       MTRDATA.DATA        0052    0000X
     UNIT SIZE = 005A
  MTREXIT
       MTREXIT.CODE        0002    50A2
     UNIT SIZE = 0002
  MTRREAD
       MTRREAD.CALLBLK     0008    5104
       MTRREAD.CODE        002C    50A4
     UNIT SIZE = 002C
  MTRWRIT
       MTRWRIT.CALLBLK     0008    5104
       MTRWRIT.CODE        0034    50D0
     UNIT SIZE = 0034

  TOTAL PROGRAM SIZE = 015E

==> END.
 LOAD MODULE: BUBEXEC
 LOAD MODULE: BUBPRPT
 LOAD MODULE: BUBSORT
 LOAD MODULE: MTRDATA
 LOAD MODULE: MTREXIT
 LOAD MODULE: MTRREAD
 LOAD MODULE: MTRWRIT
****(EXECUTIVE)         NORMAL TERMINATION
```

CHAPTER 5
A SAMPLE PROGRAM RUN (AmZ8001)

This chapter contains a sample AmZ8001 PROGRAM run. A Modules containing several software segments are linked to produce an output file. Each named software segment is assigned to a separate hardware segment.

```
MACRO8000:           Version 2.0   9/19/80
MACZ  B:BUBEXEC1 S O L=B:BUBEXEC1.PRN
BUBEXEC1


0000
0000                                  MODULE  'BUBEXEC1';
0000
0000                                  (* AMZ8001 VERSION *)
0000
0000                                  GLOBAL  START;
0000
0000                                  EXTERNAL PROMPT;
0000                                  EXTERNAL READ;
0000                                  EXTERNAL SORT;
0000                                  EXTERNAL WRITE;
0000                                  EXTERNAL EXIT;
0000
0000                                  SEGMENT 'CODE';
0000                          START:
0000   5F00S0002 0000                 CALL     PROMPT;
0006   5F00S0003 0000                 CALL     READ;
000C                                  (* R3 CHAR COUNT FROM READ *)
000C   0B03 0001                      IF R3 LE 1 THEN
0010   EA03 5E08S0007                         JP       START;
0014   0000
0018   5F00S0004 0000                 CALL     SORT;
001E   5F00S0005 0000                 CALL     WRITE;
0024   5F00S0006 0000                 CALL     EXIT;
002A
002A                                  END.
```

```
MACRO8000:           Version 2.0    9/19/80
MACZ  B:BUBPRPT1 S O L=B:BUBPRPT.PRN
BUBPRPT1


0000
0000                                MODULE    'BUBPRPT1';
0000
0000                                (* AMZ8001 VERSION *)
0000
0000                                GLOBAL   PROMPT;
0000
0000                                EXTERNAL BUFLEN, BUFFER;
0000                                EXTERNAL WRITE;
0000
0000                                SEGMENT 'CODE';
0000                        PROMPT:
0000   1404S0003 0000               LDL      RR4, ^BUFFER;
0006   1406S0005 002C               LDL      RR6, ^MESSAGE + 1;
000C   6108S0005 002A               LD       R8, MESLEN;
0012                                (* MOVE MESSAGE INTO BUFFER *)
0012   BA61 0840                    LDIRB    RR4^, RR6^, R8;
0016                                (* SET BUFFER LENGTH *)
0016   6108S0005 002A               LD       R8, MESLEN;
001C   6F08S0002 0000               LD       BUFLEN, R8;
0022   5F00S0004 0000               CALL     WRITE;
0028   9E08                         RET;
002A                        MESLEN:
002A   00                           BYTE:    0;
002B                        MESSAGE:
002B                                STRING:  'ENTER CHARACTERS TO BE ',
002B   2B45 4E54 4552                        'SORTED, THEN RETURN ';
0031   2043 4841 5241
0037   4354 4552 5320
003D   544F 2042 4520
0043   534F 5254 4544
0049   2C20 5448 454E
004F   2052 4554 5552
0055   4E20
0057
0057                                END.
```

note use of register pairs. Compare Ch. 4

```
MACRO8000:          Version 2.0   9/19/80
MACZ  B:BUBSORT1 S O L=B:BUBSORT1.PRN
BUBSORT1


0000
0000                                   MODULE  ´BUBSORT1´;
0000
0000                                   (* AMZ8001 VERSION *)
0000
0000                                   GLOBAL  SORT;
0000
0000                                   EXTERNAL BUFLEN, BUFFER;
0000
0000                                   CONST   LAST     = RR2,
0000                                           THIS     = RR4,
0000                                           NEXT     = RR6,
0000                                           LAST_CH  = RH1,
0000                                           THIS_CH  = RL1,
0000                                           COUNT    = R8,
0000                                           SWAPS    = R9;
0000
0000                                   SEGMENT ´CODE´;
0000                           SORT:
0000                                   (* INITIALIZE FOR SORT *)
0000    2109 0000                      LD      SWAPS, 0;
0004                                   (* INITIALIZE POINTERS *)
0004    1404S0003 0000                 LDL     THIS, ^BUFFER;
000A    9446                           LDL     NEXT, THIS;
000C    A970                           INC     R7, 1;
000E                                   (* ADJUST WORKING COUNT *)
000E    6108S0002 0000                 LD      COUNT, BUFLEN;
0014    AB80                           DEC     COUNT, 1;
0016                                   (* CY LATER USED FOR OV *)
0016    8D83                           RESFLG  CY;
0018                           LOOK:
0018    BA66 084B                      CPSIRB  THIS^, NEXT^, COUNT, LGT;
001C                                   IF OV (* AT END OF STRING *) THEN
001C    EC01 8D81                              SETFLG  CY;
0020                                   IF ZR (* SWAP NEEDED *) THEN
0020    EE07                                   BEGIN
0022                                           (* GET POINTER TO LAST *)
0022    9442                                   LDL     LAST, THIS;
0024    AB30                                   DEC     R3, 1;
0026                                           (* SWAP LAST AND THIS *)
0026    2021                                   LDB     LAST_CH, LAST^;
0028    2049                                   LDB     THIS_CH, THIS^;
002A    2E29                                   LDB     LAST^, THIS_CH;
002C    2E41                                   LDB     THIS^, LAST_CH;
002E                                           (* INCREMENT SWAP COUNTER *)
```

```
MACRO8000:          Version 2.0   9/19/80
MACZ  B:BUBSORT1 S O L=B:BUBSORT1.PRN
BUBSORT1

002E                                         INC     SWAPS, 1
002E   A990                                  END;
0030                                 IF NC (* NOT AT END OF LINE *) THEN
0030   E701 E8F2                             JR      LOOK;
0034   8599                          IF SWAPS NE 0 (* CHANGES MADE *) THEN
0036   E601 E8E3                             JR      SORT;
003A   9E08                          RET;
003C
003C                                 END.
```

```
MACRO8000:          Version 2.0    9/19/80
MACZ  B:MTRDATA1 S O L=B:MTRDATA1.PRN
MTRDATA1

0000
0000                             MODULE   'MTRDATA1';
0000
0000                             (* AMZ8001 VERSION *)
0000
0000                             GLOBAL  BUFLEN, BUFFER;
0000
0000                             SEGMENT 'DATA';
0000                    BUFLEN:
0000                             BYTE      (2);
0002                    BUFFER:
0002                             BYTE      (80);
0052
0052                             SEGMENT [@COM], 'CALLBLK';
0000                    CALL_BLOCK:
0000                             WORD     (5);
000A
000A                             END.
```

↑

note that block is one word longer, because it now Contains a segmented address.

```
MACRO8000:          Version 2.0   9/19/80
MACZ  B:MTREXIT1 S O L=B:MTREXIT1.PRN
MTREXIT1


0000
0000                              MODULE   'MTREXIT1';
0000
0000                              (* AMZ8001 VERSION *)
0000
0000                              GLOBAL   EXIT;
0000
0000                              SEGMENT  'CODE';
0000                    EXIT:
0000   7FA1                       SC       161;
0002
0002                              END.
```

```
MACRO8000:          Version 2.0   9/19/80
MACZ  B:MTRREAD1 S O L=B:MTRREAD1.PRN
MTRREAD1


0000
0000                                 MODULE   ´MTRREAD1´;
0000
0000                                 (* AMZ8001 VERSION *)
0000
0000                                 GLOBAL   READ;
0000
0000                                 EXTERNAL BUFLEN, BUFFER;
0000
0000                                 SEGMENT ´CODE´;
0000                       READ:
0000                                 (* SET UP CALL BLOCK *)
0000    4D05S0005 0000               LD       CALL_BLOCK, #0100;
0006    0100
0008    4D05S0005 0002               LD       CALL_BLOCK(2), 0;
000E    0000
0010    1402S0003 0000               LDL      RR2, ^BUFFER;
0016    5D02S0005 0004               LDL      CALL_BLOCK(4), RR2;
001C    4D05S0005 0008               LD       CALL_BLOCK(8), 80;
0022    0050
0024                                 (* SET UP POINTER *)
0024    1402S0005 0000               LDL      RR2, ^CALL_BLOCK;
002A    7F00                         SC       0;
002C                                 (* GET, ADJUST, SAVE COUNT *)
002C    6103S0005 0008               LD       R3, CALL_BLOCK(8);
0032    AB30                         DEC      R3, 1;
0034    6F03S0002 0000               LD       BUFLEN, R3;
003A    9E08                         RET;
003C
003C                                 SEGMENT [@COM], ´CALLBLK´;
0000                       CALL_BLOCK:
0000                                 WORD     (5);
000A
000A                                 END.
```

```
MACRO8000:          Version 2.0   9/19/80
MACZ  B:MTRWRIT1 S O L=B:MTRWRIT1.PRN
MTRWRIT1

0000
0000                            MODULE   'MTRWRIT1';
0000
0000                            (* AMZ8001 VERSION *)
0000
0000                            GLOBAL   WRITE;
0000
0000                            EXTERNAL BUFLEN, BUFFER;
0000
0000                            SEGMENT  'CODE';
0000                    WRITE:
0000                            (* SET UP CALL BLOCK *)
0000   4D05S0005 0000           LD       CALL_BLOCK, #0200;
0006   0200
0008   4D05S0005 0002           LD       CALL_BLOCK(2), 0;
000E   0000
0010   1402S0003 0000           LDL      RR2, ^BUFFER;
0016   5D02S0005 0004           LDL      CALL_BLOCK(4), RR2;
001C                            (* ADD CR/LF AT END OF BUFFER *)
001C   6103S0002 0000           LD       R3, BUFLEN;
0022   C80D                     LDB      RL0, #0D;
0024   7228 0300                LDB      RR2^(R3), RL0;
0028   A930                     INC      R3, 1;
002A   C80A                     LDB      RL0, #0A;
002C   7228 0300                LDB      RR2^(R3), RL0;
0030                            (* STORE ACTUAL COUNT *)
0030   A930                     INC      R3, 1;
0032   6F03S0005 0008           LD       CALL_BLOCK(8), R3;
0038                            (* SET UP POINTER *)
0038   1402S0005 0000           LDL      RR2, ^CALL_BLOCK;
003E   7F00                     SC       0;
0040   9E08                     RET;
0042
0042                            SEGMENT  [@COM], 'CALLBLK';
0000                    CALL_BLOCK:
0000                            WORD     (5);
000A
000A                            END.
```

```
LNKZ * B=B:LNKPR1
LINK8000:          VERSION 2.0,  10/13/80

===> PROGRAM START
===> FILE 'B:BUB????1'
 ENTER MODULE: BUBEXEC1
 ENTER MODULE: BUBPRPT1
 ENTER MODULE: BUBSORT1
===> FILE 'B:MTR????1'
 ENTER MODULE: MTRDATA1
 ENTER MODULE: MTREXIT1
 ENTER MODULE: MTRREAD1
 ENTER MODULE: MTRWRIT1
===> SEGMENT COMMON←BLOCJK
===> COMBINE .CALLBLK
===> MAP BY SEGMENT

  SEGMENT                        SIZE    ADDRESS      OUTPUT SEGMENT

  CALLBLK
       MTRDATA1.CALLBLK          000A      0000       COMMON←BLOCK
       MTRREAD1.CALLBLK          000A      0000       COMMON←BLOCK
       MTRWRIT1.CALLBLK          000A      0000       COMMON←BLOCK
    UNIT SIZE = 000A
  CODE
       BUBEXEC1.CODE             002A
       BUBPRPT1.CODE             0058
       BUBSORT1.CODE             003C
       MTREXIT1.CODE             0002
       MTRREAD1.CODE             003C
       MTRWRIT1.CODE             0042
    UNIT SIZE = 013E
  DATA
       MTRDATA1.DATA             0052
    UNIT SIZE = 0052

  TOTAL PROGRAM SIZE = 019A

===> SEGMENT CODE←AREA
===> OFFSET %1000
===> COMBINE .CODE
===> SETLSEG COMMON←BLOCK := 2
```

get the relocatable input from .ZRL files

Put .CALLBLK in an output segment in normal space called Common ← Block. Use of segment directive implies Am Z8001 code

note base addresses are the same because .CALLBLK is a common segment

these segments have not yet been assigned to output segments

a new output segment

put all .CODE segments in another segment in normal space called CODE ← AREA

Assign .CALLBLK to #2 hardware segment

```
===> MAP BY SEGMENT

   SEGMENT                       SIZE     ADDRESS        OUTPUT SEGMENT

   CALLBLK
        MTRDATA1.CALLBLK         000A     02 0000        COMMON←BLOCK
        MTRREAD1.CALLBLK         000A     02 0000        COMMON←BLOCK
        MTRWRIT1.CALLBLK         000A     02 0000        COMMON←BLOCK
      UNIT SIZE = 000A
```

↑ segment number (only .CALLBACK has been assigned a segment number)

```
   CODE
        BUBEXEC1.CODE            002A        1000        CODE←AREA
        BUBPRPT1.CODE            0058        102A        CODE←AREA
        BUBSORT1.CODE            003C        1082        CODE←AREA
        MTREXIT1.CODE            0002        10BE        CODE←AREA
        MTRREAD1.CODE            003C        10C0        CODE←AREA
        MTRWRIT1.CODE            0042        10FC        CODE←AREA
      UNIT SIZE = 013E
   DATA
        MTRDATA1.DATA            0052
      UNIT SIZE = 0052

   TOTAL PROGRAM SIZE = 019A
```

} unassigned input segments

```
===> XSPACE
```

← put any new input segments in extended space assign all remaining normal space output segments to hardware segment numbers.

```
===> MAP BY SEGMENT

   SEGMENT                       SIZE     ADDRESS        OUTPUT SEGMENT

   CALLBLK
        MTRDATA1.CALLBLK         000A     02 0000        COMMON←BLOCK
        MTRREAD1.CALLBLK         000A     02 0000        COMMON←BLOCK
        MTRWRIT1.CALLBLK         000A     02 0000        COMMON←BLOCK
      UNIT SIZE = 000A
   CODE
        BUBEXEC1.CODE            002A     03 1000        CODE←AREA
        BUBPRPT1.CODE            0058     03 102A        CODE←AREA
        BUBSORT1.CODE            003C     03 1082        CODE←AREA
        MTREXIT1.CODE            0002     03 10BE        CODE←AREA
        MTRREAD1.CODE            003C     03 10C0        CODE←AREA
        MTRWRIT1.CODE            0042     03 10FC        CODE←AREA
      UNIT SIZE = 013E
   DATA
```

↑ note CODE ← AREA assigned to next hardware segment

```
        MTRDATA1.DATA          0052
      UNIT SIZE = 0052

   TOTAL PROGRAM SIZE = 019A

==> SEGMENT DATA←AREA
==> COMBINE .DATA
==> SETLSEG DATA←AREA := 3
==> MAP BY SEGMENT
```

Put all .DATA segments in an output segment called DATA ← AREA which is in extended space.

← Put DATA ← AREA in extended space hardware segment 3 parallel with CODE ← AREA segment 3

```
   SEGMENT                     SIZE     ADDRESS         OUTPUT SEGMENT

   CALLBLK
        MTRDATA1.CALLBLK       000A     02 0000         COMMON←BLOCK
        MTRREAD1.CALLBLK       000A     02 0000         COMMON←BLOCK
        MTRWRIT1.CALLBLK       000A     02 0000         COMMON←BLOCK
      UNIT SIZE = 000A
   CODE
        BUBEXEC1.CODE          002A     03 1000         CODE←AREA
        BUBPRPT1.CODE          0058     03 102A         CODE←AREA
        BUBSORT1.CODE          003C     03 1082         CODE←AREA
        MTREXIT1.CODE          0002     03 10BE         CODE←AREA
        MTRREAD1.CODE          003C     03 10C0         CODE←AREA
        MTRWRIT1.CODE          0042     03 10FC         CODE←AREA
      UNIT SIZE = 013E
   DATA
        MTRDATA1.DATA          0052     03 0000X        DATA←AREA
      UNIT SIZE = 0052

   TOTAL PROGRAM SIZE = 019A
```

↑ note x for extended space

```
==> MAP

   ENTRY POINT   ADDRESS      MODULE      .SEGMENT     OUTPUT SEGMENT

   BUFFER        03 0002X     MTRDATA1    .DATA        DATA←AREA
   BUFLEN        03 0000X     MTRDATA1    .DATA        DATA←AREA
   EXIT          03 10BE      MTREXIT1    .CODE        CODE←AREA
   PROMPT        03 102A      BUBPRPT1    .CODE        CODE←AREA
   READ          03 10C0      MTRREAD1    .CODE        CODE←AREA
   SORT          03 1082      BUBSORT1    .CODE        CODE←AREA
   START         03 1000      BUBEXEC1    .CODE        CODE←AREA
```

```
  WRITE         03 10FC    MTRWRIT1   .CODE      CODE←AREA

===> END.
 LOAD MODULE: BUBEXEC1
 LOAD MODULE: BUBPRPT1
 LOAD MODULE: BUBSORT1
 LOAD MODULE: MTRDATA1
 LOAD MODULE: MTREXIT1
 LOAD MODULE: MTRREAD1
 LOAD MODULE: MTRWRIT1
****(EXECUTIVE)         NORMAL TERMINATION
```

CHAPTER 6
MODULE CREATION

This chapter contains a sample MODULE run. Module creation is used for incremental linking, where the linking operation is done is done in two or more steps. Typically, selected program parts are linked together in a MODULE run that produces a combined relocatable module. The combined module is then used in a later linking operation as part of the relocatable input. The last step is a PROGRAM run that produces absolute code suitable for downloading.

produce relocatable .ZRL combined module

```
A>LNKZ * O=LNKMOD
LINK8000:            VERSION 2.0,  10/13/80
===> MODULE 'MONITOR'
===> HEADER 'COMBINED RELOCATABLE'
===> FILE 'MTR*'
```

get relocatable input from .ZRL files

```
 ENTER MODULE: MTRDATA
 ENTER MODULE: MTREXIT
 ENTER MODULE: MTRREAD
 ENTER MODULE: MTRWRIT

===> MAP
BUFFER        0002*       MTRDATA     .DATA
BUFLEN        0000*       MTRDATA     .DATA
EXIT          0000*       MTREXIT     .CODE
READ          0000*       MTRREAD     .CODE
WRITE         0000*       MTRWRIT     .CODE
===> END.
 LOAD MODULE: MTRDATA
 LOAD MODULE: MTREXIT
 LOAD MODULE: MTRREAD
 LOAD MODULE: MTRWRIT
```

type combined module

```
A>TYPE LNKMOD.ZRL
MODULE: MONITOR
        COMBINED RELOCATABLE
```

produce binary file .BIN for download

```
A>LNKZ    * B=LNKMOD
LINK8000:            VERSION 2.0,  10/13/80
===> PROGRAM START
===> FILE 'BUB*'
```

get relocatable input

```
 ENTER MODULE: BUBEXEC
 ENTER MODULE: BUBPRPT
 ENTER MODULE: BUBSORT
```

```
===> FILE LNKMOD
```

get combined module as input

```
 ENTER MODULE: MONITOR

===> MAP
BUFFER        000A*       MONITOR     .MONITOR
BUFLEN        0008*       MONITOR     .MONITOR
EXIT          005A*       MONITOR     .MONITOR
PROMPT        0000*       BUBPRPT     .CODE
READ          005C*       MONITOR     .MONITOR
SORT          0000*       BUBSORT     .CODE
START         0000*       BUBEXEC     .CODE
WRITE         0088*       MONITOR     .MONITOR
===> ABSOLUTE #5200
===> COMBINE BUBEXEC
```

assign executive routine

```
===> MAP
BUFFER        000A*       MONITOR     .MONITOR
BUFLEN        0008*       MONITOR     .MONITOR
EXIT          005A*       MONITOR     .MONITOR
PROMPT        0000*       BUBPRPT     .CODE
READ          005C*       MONITOR     .MONITOR
SORT          0000*       BUBSORT     .CODE
START         5200        BUBEXEC     .CODE
WRITE         0088*       MONITOR     .MONITOR
===> COMBINE
```

assign other routines

```
===> MAP
BUFFER        52AC        MONITOR     .MONITOR
BUFLEN        52AA        MONITOR     .MONITOR
EXIT          52FC        MONITOR     .MONITOR
PROMPT        521E        BUBPRPT     .CODE
READ          52FE        MONITOR     .MONITOR
SORT          526A        BUBSORT     .CODE
START         5200        BUBEXEC     .CODE
WRITE         532A        MONITOR     .MONITOR
===> MAP BY MODULE
BUBEXEC
     BUBEXEC.CODE        001E     5200
```

main entry point

```
     UNIT SIZE = 001E
BUBPRPT
     BUBPRPT.CODE        004C     521E
     UNIT SIZE = 004C
BUBSORT
     BUBSORT.CODE        0038     526A
     UNIT SIZE = 0038
MONITOR
     MONITOR.MONITOR     00BC     52A2
     UNIT SIZE = 00BC
  TOTAL PROGRAM SIZE = 015E

===> END.
 LOAD MODULE: BUBEXEC
 LOAD MODULE: BUBPRPT
 LOAD MODULE: BUBSORT
 LOAD MODULE: MONITOR
```

file LNKMOD.BIN
can now be downloaded

CHAPTER 7
LIBRARY CREATION

This chapter contains a sample LIBRARY run. A number of program parts are collected into a library. The relocatable library is then used to satisfy externals in a PROGRAM run that produces absolute code suitable for downloading.

```
A>LNKZ * O=LNKLIB
```

produce relocatable .ZRL library file

```
LINK8000:              VERSION 2.0,  10/13/80
===> LIBRARY 'MONITOR'
===> HEADER 'STANDARD LIBRARY'
===> FILE 'MTR*'

===> MAP
```

get relocatable input from .ZRL files

```
BUFFER        MTRDATA
BUFLEN        MTRDATA
EXIT          MTREXIT
READ          MTRREAD
WRITE         MTRWRIT
===> END.

A>TYPE LNKLIB.ZRL
```

type library file

```
LIBRARY: MONITOR
         STANDARD LIBRARY

A>LNKZ * B=LNKLIB
```

produce binary file. BIN for download

```
LINK8000:              VERSION 2.0,  10/13/80
===> PROGRAM START
===> FILE 'BUB*'

 ENTER MODULE: BUBEXEC
```

get relocatable input

```
 ENTER MODULE: BUBPRPT
 ENTER MODULE: BUBSORT

===> MAP
BUFFER        EXTERNAL
BUFLEN        EXTERNAL
EXIT          EXTERNAL
PROMPT        0000*       BUBPRPT      .CODE
READ          EXTERNAL
SORT          0000*       BUBSORT      .CODE
START         0000*       BUBEXEC      .CODE
WRITE         EXTERNAL
```

library references are unsatisfied

```
===> SEARCH LNKLIB

 ENTER LIBRARY: MONITOR
 ENTER MODULE: MTRDATA
 ENTER MODULE: MTREXIT
 ENTER MODULE: MTRREAD
 ENTER MODULE: MTRWRIT
```

externals now satisfied

```
===> MAP
BUFFER       0002*       MTRDATA     .DATA
BUFLEN       0000*       MTRDATA     .DATA
EXIT         0000*       MTREXIT     .CODE
PROMPT       0000*       BUBPRPT     .CODE
READ         0000*       MTRREAD     .CODE
SORT         0000*       BUBSORT     .CODE
START        0000*       BUBEXEC     .CODE
WRITE        0000*       MTRWRIT     .CODE
===> ABSOLUTE #5100
===> COMBINE BUBEXEC
```

assign executive routine

```
===> MAP
BUFFER       0002*       MTRDATA     .DATA
BUFLEN       0000*       MTRDATA     .DATA
EXIT         0000*       MTREXIT     .CODE
PROMPT       0000*       BUBPRPT     .CODE
READ         0000*       MTRREAD     .CODE
SORT         0000*       BUBSORT     .CODE
START        5100        BUBEXEC     .CODE
WRITE        0000*       MTRWRIT     .CODE
===> COMBINE
```

assign other routines

```
===> MAP
BUFFER       51AC        MTRDATA     .DATA
BUFLEN       51AA        MTRDATA     .DATA
EXIT         51FC        MTREXIT     .CODE
PROMPT       511E        BUBPRPT     .CODE
READ         51FE        MTRREAD     .CODE
SORT         516A        BUBSORT     .CODE
START        5100        BUBEXEC     .CODE
WRITE        522A        MTRWRIT     .CODE
===> MAP BY MODULE
BUBEXEC
     BUBEXEC.CODE        001E     5100
     UNIT SIZE = 001E
```

main entry point

```
BUBPRPT
     BUBPRPT.CODE        004C     511E
     UNIT SIZE = 004C
BUBSORT
     BUBSORT.CODE        0038     516A


     UNIT SIZE = 0038
MTRDATA
     MTRDATA.CALLBLK     0008     51A2
     MTRDATA.DATA        0052     51AA
     UNIT SIZE = 005A
```

```
MTREXIT
     MTREXIT.CODE         0002     51FC
     UNIT SIZE = 0002
MTRREAD
     MTRREAD.CALLBLK      0008     51A2
     MTRREAD.CODE         002C     51FE
     UNIT SIZE = 002C
MTRWRIT
     MTRWRIT.CALLBLK      0008     51A2
     MTRWRIT.CODE         0034     522A
     UNIT SIZE = 0034
  TOTAL PROGRAM SIZE = 015E
==> END.
 LOAD MODULE: BUBEXEC
 LOAD MODULE: BUBPRPT
 LOAD MODULE: BUBSORT
 LOAD MODULE: MTRDATA
 LOAD MODULE: MTREXIT
 LOAD MODULE: MTRREAD
 LOAD MODULE: MTRWRIT
```

file LNKLIB.BIN can
now be downloaded

CHAPTER 8
ROMLIB CREATION

This chapter contains a sample ROMLIB run. A ROM library can be created as additional output from a PROGRAM run. The R or R=file option must be used on the LNKZ product call, and the RETAIN or OMIT directive may be used to specify a subset of entry points for the ROM library.

The absolute code referenced by the ROM library is a hex file that is burned into PROMs. The ROM library itself remains as a file that can be combined with other ROM libraries and used in a later linking operation.

```
                  ←——— produce hex file for PROM burning
A>LNKZ * H=LNKROM,R=LNKROM ←——— produce ROM library
LINK8000:           VERSION 2.0,  10/13/80
===> PROGRAM MTRCALLS
===> FILE 'MTR*'

 ENTER MODULE: MTRDATA←——— get relocatable input
 ENTER MODULE: MTREXIT
 ENTER MODULE: MTRREAD
 ENTER MODULE: MTRWRIT

===> MAP
BUFFER        0002*       MTRDATA     .DATA
BUFLEN        0000*       MTRDATA     .DATA
EXIT          0000*       MTREXIT     .CODE
MTRCALLS      EXTERNAL
READ          0000*       MTRREAD     .CODE
WRITE         0000*       MTRWRIT     .CODE
===> ABSOLUTE #1000
===> COMBINE .CODE ←——— assign all code segments


===> MAP
BUFFER        0002*       MTRDATA     .DATA
BUFLEN        0000*       MTRDATA     .DATA
EXIT          1000        MTREXIT     .CODE
MTRCALLS      EXTERNAL
READ          1002        MTRREAD     .CODE
WRITE         102E        MTRWRIT     .CODE
===> ABSOLUTE #4F00
===> COMBINE MTRDATA←——— assign addresses for data
===> MAP
BUFFER        4F0A        MTRDATA     .DATA
BUFLEN        4F08        MTRDATA     .DATA
EXIT          1000        MTREXIT     .CODE
MTRCALLS      EXTERNAL
READ          1002        MTRREAD     .CODE
WRITE         102E        MTRWRIT     .CODE
```

```
===> MAP BY MODULE
MTRDATA
     MTRDATA.CALLBLK      0008     4F00
     MTRDATA.DATA         0052     4F08
     UNIT SIZE = 005A
MTREXIT
     MTREXIT.CODE         0002     1000
     UNIT SIZE = 0002
MTRREAD
     MTRREAD.CALLBLK      0008     4F00
     MTRREAD.CODE         002C     1002
     UNIT SIZE = 002C
MTRWRIT
     MTRWRIT.CALLBLK      0008     4F00
     MTRWRIT.CODE         0034     102E
     UNIT SIZE = 0034
  TOTAL PROGRAM SIZE = 00BC
===> END.
1 UNDEFINED EXTERNALS
 LOAD MODULE: MTRDATA
 LOAD MODULE: MTREXIT
 LOAD MODULE: MTRREAD
 LOAD MODULE: MTRWRIT

A>TYPE LNKROM.HEX
```

type hex file containing routines intended for PROM

```
:021000007FA1CE
:101002004D054F0001004D054F02000021024F0A1D
:101012006F024F044D054F06005021014F007F0023
:0C10220061034F06AB306F034F089E08BF


:10102E004D054F0002004D054F02000021024F0AF0
:10103E006F024F0461034F08C80D72280300A930D8
:10104E00C80A72280300A9306F034F0621014F0012
:04105E007F009E0869
:0000000000

A>TYPE LNKROM.ZRL
```

type ROM library containing addresses of routines

```
ROMLIB: MTRCALLS

A>LNKZ * B=LNKRAM
```

Produce binary file .BIN for other routines and data

```
LINK8000:              VERSION 2.0,  10/13/80
===> PROGRAM START
===> FILE 'BUB*'

 ENTER MODULE: BUBEXEC
```

get relocatable input

```
 ENTER MODULE: BUBPRPT
 ENTER MODULE: BUBSORT
```

```
===> FILE MTRDATA                          get relocatable input

 ENTER MODULE: MTRDATA

===> MAP
BUFFER       0002*      MTRDATA     .DATA
BUFLEN       0000*      MTRDATA     .DATA
EXIT         EXTERNAL
PROMPT       0000*      BUBPRPT     .CODE
READ         EXTERNAL
SORT         0000*      BUBSORT     .CODE
START        0000*      BUBEXEC     .CODE
WRITE        EXTERNAL
===> SEARCH LNKROM                         search ROM library

 ENTER ROMLIB: MTRCALLS

===> MAP
BUFFER       0002*      MTRDATA     .DATA
BUFLEN       0000*      MTRDATA     .DATA
EXIT         1000
PROMPT       0000*      BUBPRPT     .CODE
READ         1002
SORT         0000*      BUBSORT     .CODE
START        0000*      BUBEXEC     .CODE
WRITE        102E
===> ABSOLUTE #4F00

                                           assign data segment
===> COMBINE MTRDATA
===> MAP
BUFFER       4F0A       MTRDATA     .DATA
BUFLEN       4F08       MTRDATA     .DATA
EXIT         1000
PROMPT       0000*      BUBPRPT     .CODE
READ         1002
SORT         0000*      BUBSORT     .CODE
START        0000*      BUBEXEC     .CODE
WRITE        102E
===> ABSOLUTE #5000                        assign code segments
===> COMBINE .CODE
===> MAP
BUFFER       4F0A       MTRDATA     .DATA
BUFLEN       4F08       MTRDATA     .DATA
EXIT         1000
PROMPT       501E       BUBPRPT     .CODE
READ         1002
SORT         506A       BUBSORT     .CODE
START        5000       BUBEXEC     .CODE
WRITE        102E
```

```
===> MAP BY MODULE
BUBEXEC
     BUBEXEC.CODE        001E     5000 ← main entry point
     UNIT SIZE = 001E
BUBPRPT
     BUBPRPT.CODE        004C     501E
     UNIT SIZE = 004C
BUBSORT
     BUBSORT.CODE        0038     506A
     UNIT SIZE = 0038
MTRDATA
     MTRDATA.CALLBLK     0008     4F00
     MTRDATA.DATA        0052     4F08
     UNIT SIZE = 005A
  TOTAL PROGRAM SIZE = 00FC
===> END.
 LOAD MODULE: BUBEXEC
 LOAD MODULE: BUBPRPT
 LOAD MODULE: BUBSORT
 LOAD MODULE: MTRDATA
```

dump binary file that can be downloaded

```
A>DUMP LNKRAM.BIN

0000 01 50 00 02 50 00 03 10 5F 00 50 1E 5F 00 10 02  .P..P...←.P.←...


0010 0B 03 00 01 EA 02 5E 08 02 50 10 03 0E 50 00 5F  ......↑..P...P.←
0020 00 50 6A 5F 00 10 2E 5F 00 10 00 02 50 1E 03 10  .PJ←...←....P...
0030 21 04 4F 0A 21 05 50 3E 61 06 50 3C BA 51 06 40  !.O.!.P>A.P<.Q.@
0040 02 50 2E 03 10 61 06 50 3C 6F 06 4F 08 5F 00 10  .P...A.P<O.O.←..
0050 2E 9E 08 00 2B 02 50 3E 03 10 45 4E 54 45 52 20  ....+.P>..ENTER
0060 43 48 41 52 41 43 54 45 52 53 02 50 4E 03 10 20  CHARACTERS.PN..
0070 54 4F 20 42 45 20 53 4F 52 54 45 44 2C 20 54 02  TO BE SORTED, T.
0080 50 5E 03 0B 48 45 4E 20 52 45 54 55 52 4E 20 02  P↑..HEN RETURN .
0090 50 6A 03 10 21 07 00 00 21 03 4F 0A A1 34 A9 40  PJ..!...!.O..4.@
00A0 61 06 4F 08 02 50 7A 03 10 AB 60 8D 83 BA 46 06  A.O..PZ...@...F.
00B0 3B EC 01 8D 81 EE 07 A1 32 02 50 8A 03 10 AB 20  ;........2.P....
00C0 20 25 20 3D 2E 2D 2E 35 A9 70 E7 01 E8 F2 02 50   % =.-.5.P.....P
00D0 9A 03 08 85 77 E6 01 E8 E5 9E 08 00 00 1A 1A 1A  ....W...........
00E0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A  ................
00F0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A  ................
```

APPENDIX A
ASCII CHARACTER SET

The ASCII character set is shown in the following table:

TABLE A-1. ASCII.

Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
00	0	NUL	20	32	SP	40	64	@	60	96	`
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	EOT	24	36	$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	´	47	71	G	67	103	g
08	8	BS	28	40	(	48	72	H	68	104	h
09	9	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	DEL

APPENDIX B
HEX FILE FORMAT

The linker can produce hex files suitable for putting code into PROMs. Hex file creation is requested with the LINK8000 H option, described in chapter 1. The format of a hex file is the INTEL hex file format:

```
Colon, 1 character
|
| Data length, 2 characters (00 for final record)
| |
| |  Record address, 4 characters (in final record, specifies entry
| |  | point)
| |  |
| |  |    Relocation map
| |  |    |
| |  |    |  Data, 2 through 32 characters representing 1 through 16
| |  |    |  | byte values (empty for final record)
| |  |    |  |
| |  |    |  |                                Checksum, 2 characters
| |  |    |  |                                |
| |  |    |  |                                |  CR/LF (carriage
| |  |    |  |                                |  | return/line feed)
| |  |    |  |                                |  |
V V  V    V  V                                V  V
_ __ ____ __ ________________________________ __ __
|:|__|____|00|________________________________|__|__|
```

APPENDIX C
BINARY FILE FORMAT

The linker can produce binary files suitable for program downloading or PROM burning. Binary file creation is requested with the LINK8000 B option described in Chapter 1. The AMC binary file contains the same type of information as that found in hex files, but the data is in the more efficient hexadecimal representation.

Each binary file contains a main entry point group, a destination address group followed by one or more data groups, any additional destination address groups, each followed by one or more data groups, and finally a terminator group. The format of each group is described below for both processors.

C-1. AmZ8002 BINARY FILE FORMAT

```
 01 signal for main entry point, 1 byte
 |
 |  Main entry point (transfer) address, 2 bytes
 |  |
 V  V

|01|____|
```

```
 02 signal for destination address, 1 byte
 |
 |  Destination address for following data, 2 bytes
 |  |
 V  V

|02|____|
```

```
 03 signal for data, 1 byte
 |
 |  Data length, 1 byte
 |  |
 |  |  Data, 1 through 255 bytes
 |  |  |
 V  V  V

|03|__|__________________________________________________________|
```

```
 00 signal for terminator, 2 bytes
 |
 V

|0000|
```

For AmZ8002 extended space groups (see section 3-18), the signals are as follows:

09	signal for main entry point	(1 byte)
0A	signal for destination address	(1 byte)
0B	signal for data	(1 byte)
0000	signal for terminator	(2 bytes)

C-2. AmZ8001 BINARY FILE FORMAT

The AmZ8001 binary file format is very similar to the AmZ8002 format, except for the address representation and the actual signal numbers. The addresses in AmZ8001 binary files have the same format as AmZ8001 32-bit address operands:

```
              ___ _________ _________ ______________________
             |___|_________|_________|______________________|
bits          31            23        15                     0
contents      1   segment   00000000          offset
```

```
 05 signal for main entry point, 1 byte
 |
 |  Main entry point (transfer) address, 4 bytes
 |  |
 V  V
 __  ____
|05||____|
```

```
 06 signal for destination address, 1 byte
 |
 |  Destination address for following data, 4 bytes
 |  |
 V  V
 __  ____
|06||____|
```

```
 07 signal for data, 1 byte
 |
 |  Data length, 1 byte
 |  |
 |  |  Data, 1 through 255 bytes
 |  |  |
 V  V  V
 __  __  ___________________________________________________
|07||__||___________________________________________________|
```

```
 00 signal for terminator, 2 bytes
 |
 V
 ____
|0000|
```

For AmZ8001 extended space groups (see section 3-18), the signals are as follows:

0D	signal for main entry point	(1 byte)
0E	signal for destination address	(1 byte)
0F	signal for data	(1 byte)
0000	signal for terminator	(2 bytes)

APPENDIX D
ERROR MESAGES

The LINK8000 error messages are error description rather than error numbers. The following messages exist.

RELATIVE ADDRESS OUT OF RANGE:
ODD ADDRESS BOUNDARY DETECTED:
UNDEFINED LABEL:
UNDEFINED MACRO:
INVALID NUMBER OF OPERANDS:
INVALID FILE NAME:
STRING TOO LONG:
MISSING OR INVALID IMMEDIATE (CONSTANT) OPERAND:
SYSTEM ERROR
IMMEDIATE OPERAND TOO LARGE:
PRODUCT CALL OVERRIDE:
DIGIT EXCEEDS RADIX
RADIX EQ 0
MISSING (
RADIX TOO LARGE
TOO MANY INVALID CHARS
INVALID NUMBER FORMAT
MISSING OR INVALID OPERAND:
MISSING)
MISSING]
INVALID LABEL IDENTIFIER:
UNRECOGNIZED STATEMENT FORM:
INVALUD LOCATION COUNTER RESET:
MISSING END
MISSING : OR (
MISSING OR INVALID STRING
MISSING OR INVALID CONST OBJECT
MISSING =
INVALID IN MACRO BODY OR CONDITIONAL LINK
MISSING OR INVALID INTEGER
INVALID MACRO STATEMENT
MISSING OR INVALID IDENTIFIER
REDEFINITION OF IDENTIFIER:
DIVISION BY 0
UNRECOGNIZED STATEMENT FORM:
MISSING DELIMITER:
INVALID DEFINITION
UNDEFINED EXPRESSION:
MISSING CONDITION CODE:
MISSING END. (OR EXTRA END)
INVALID STATEMENT BEGINNER:
MISSING STATEMENT TERMINATOR:
MISSING OR INVALID LINKING MODE (PROGRAM,MODULE, ETC.)
INVALID CHARACTER:
MISSING OR INVALID SEGMENT ATTRIBUTE SET:

```
MISSING OR INVALID SEGMENT NAME:
SEGMENT STACK UNDERFLOW
SEGMENT STACK OVERFLOW
STATEMENT INAPPROPRIATE TO LINKING MODE
INVALID ASSIGNMENT:
SHORT ADDRESS OFFSET TOO LARGE
FATAL ERROR - LINK TERMINATED
FILE STACK ERROR
ERROR IN EXTENDING FILE -
FILE SPACE OVERFLOW  -
DIRECTORY OVERFLOW
FILE CLOSE ERROR -
ATTEMPT TO READ UNWRITTEN DATA -
ATTEMPT TO READ BEYOND EOF -
UNABLE TO OPEN INPUT FILE -
OBJECT SPACE OVERFLOW
ILLEGAL SYSTEM FUNCTION (message should never appear)
DEREF SYSTEM ERROR: (message should never appear)
ATTEMPT TO COMPUTE ADDRESS DIFFERENCE
ACROSS SEGMENT BOUNDARY AT
ATTEMPT TO COMPUTE RELATIVE ADDRESS
ACROSS SEGMENT BOUNDARY AT
PHASE III ERROR
RELATIVE ADDRESS OUT OF RANGE:
ODD WORD BOUNDARY DETECTED:
UNDEFINED LABEL:
UNDEFINED EXTERNALS:
NORMAL TERMINATION
INVALID OPTION(S)
EMPTY INPUT FILE
INVALID OPTION(S)
ROMLIB NOT PERMITTED:
RELOCATABLE MODULE NOT ALLOWED IN ROMLIB EDIT:
DUPLICATE ENTRY POINT:
MISSING COMMON ATTRIBUTE:
WARNING: CURRENT ADDRESS + SEGMENT SIZE > 64K
RETAIN (OMIT) PATTERN STRING:
WARNING - "COMMON" SEGMENT SIZE ERROR:
PATTERN SPECIFIED COMBINE:
MAP VECTOR OVERFLOW
INVALID MODULE/SEGMENT SPECIFICATION (COMBINE)
INVALID ARGUMENT (RETAIN OR OMIT)
UNIDENTIFIED ENTRY
BINARY FILE RQRD -- Z8001 PROGRAM MODE
DUPLICATE ENTRY POINT:
MIXING SEGMENTED AND UNSEGMENTED MODULES PROHIBITED:
DUPLICATE ROMLIB LABEL:
DUPLICATE LIBRARY LABEL:
DUPLICATE MODULE NAME:
DUPLICATE LIBRARY MODULE:
UNABLE TO OPEN FILE:
INVALID MAP DIRECTIVE:
INVALID FILE DIRECTIVE (LIBRARY MODE)
COMBINE DIRECTIVE NOT ALLOWED
```

COMMENT SHEET

Address comments to:

Advanced Micro Computers
Publications Department
3340 Scott Boulevard
Santa Clara, CA 95051

TITLE: LINK8000 User's Manual
PUBLICATION NO: 00680148B

COMMENTS: (Describe errors, suggested additions or deletions, and include page numbers, etc.)

From: Name: ______________________ Position: ______________

Company: ______________________

Address: ______________________

Advanced
Micro
Computers

A subsidiary of
Advanced Micro Devices
3340 Scott Boulevard
Santa Clara,
California 95051
(408) 988-7777
TELEX: 171 142

Printed in U.S.A. 1/81 AMC-553