LINK EDITOR UTILITY REFERENCE MANUAL

93463-90



ComputerAutomation®

NAKED MINI® Division

HERTFORD HOUSE
DENHAM WAY, MAPLE CROSS
RICKMANSWORTH, HERTFORDSHIRE WD3 2XD

Telephone Rickmansworth 71211 Telex 92265

Engineering Reference Number
90-93463-00

LINK EDITOR UTILITY
REFERENCE MANUAL

93463-90

MAY 1981



ComputerAutomation®

NAKED MINI® Division

Engineering Reference Number
90-93463-00E0

HERTFORD HOUSE
DENHAM WAY, MAPLE CROSS
RICKMANSWORTH, HERTFORDSHIRE WD3 2XD

Telephone Rickmansworth 71211 Telex 922654

Printed in U.S.A.

REVISION HISTORY

Revision	Date	Comments
A0	July 1980	Original issue.
B0-D0		Skipped for configuration purposes.
E0	May 1981	Adds command line option TOPLOAD. The information in this manual corresponds to the OS4 software package released at revision level F000.

Computer Automation, Inc., reserves the right to make changes in specifications at any time and without notice. The information furnished by Computer Automation, Inc., in this publication is believed to be accurate and reliable. However, no responsibility is assumed by Computer Automation, Inc., for its use; nor for any infringements of patents or other rights of third parties resulting from its use. No license is granted under any patents or patent rights of Computer Automation, Inc.

Computer Automation, Inc., welcomes comments and suggestions for improvements to our manuals. Should difficulty be encountered with the information in the manual or if you have a suggestion for manual improvement, please send the comments or suggestions to Computer Automation, Inc., Technical Publications, Department 11-0770, 18651 Von Karman, Irvine, California, 92713. Please provide as much information as possible to help evaluate the comment or suggestion.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

PREFACE

The LINK Editor Utility Manual is a reference document and operating guide for linking programs, using overlays, and constructing ROM-resident programs for the NAKED MINI® 4 Family computers. The manual is designed to assist Computer Automation, Inc., personnel and customers in using the utility as part of the OS4 system.

Section one serves as an introduction to the LINK Editor and its associated terminology. Because the utility processes object text and type-codes (a subset of object text), the section ends with an analysis of type-code record layouts and instructions on reading type-code dumps. Sections 2 through 4 are operation-oriented. Various ways of using the utility are desribed and each section is highlighted with command line examples and simple program walk-throughs.

The multi-tasking capability realized by the LINK Editor Utility, is supported in section 5 with a multi-tasking program listing. The listing is enhanced with comments on program construction and notations on how the utility handles a program at link time. The back-matter (appendixes to this manual) contain type-code definitions, and error messages. The messages can be informative only (no operator intervention required) or demand operator response for command line errors.

Because this manual contains format information for object programs, the NAKED MINI® 4 Object Program Format Manual (Marketing No. 21127-90) is replaced by the release of this manual.

The following Computer Automation, Inc., documents provide related information:

	Order No.
• NAKED MINI® FORTRAN IV Language Manual	93560-80
• NAKED MINI® 4 Assembler User's Manual	93500-80
• Input/Output Subsystem IOS4 User's Manual	93405-90
• Real-Time Executive RTX4 User's Manual	93410-90
• OS4 User's Manual	93460-90

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

CONTENTS

SECTION 1 LINK EDITOR UTILITY 1-1

1.1 REMOVABLE FILES 1-1

1.2 TERMINOLOGY 1-1

1.3 PROGRAM DEVELOPMENT 1-2

1.4 TASKS AND THE LINK EDITOR 1-3

1.5 MEMORY AND THE LINK EDITOR 1-4

1.6 MODULES AND THE LINK EDITOR 1-4

1.7 DUMMY MODULES 1-5

1.8 TEMPLATES 1-5

1.9 SEGMENTS 1-6
1.9.1 Common Segments 1-6
1.9.2 Non-Common Segments 1-6
1.9.3 Shareable Segments 1-6
1.9.4 Non-Shareable Segments 1-7

1.10 OBJECT PROGRAMS 1-7
1.10.1 TC Record Layouts 1-8
1.10.2 User Type-Codes 1-9
1.10.3 TC Dump 1-10

SECTION 2 LINK EDITOR OPERATION 2-1

2.1 LINK COMMAND LINE 2-1
2.1.1 Using the LINK Command Line 2-2
2.1.2 Command Line Options 2-3
2.1.3 Command Formats 2-7
2.1.4 Command Line Arrangement for an RTX4 LINK . 2-9

2.2 OPERATOR COMMUNICATION 2-10

2.3 SELECTION PROCESS 2-11
2.3.1 Module Selection 2-12
2.3.2 Loading Modules 2-15
2.3.3 Scratch Files 2-21

SECTION 3 ROMABILITY OF PROGRAMS 3-1

3.1 ROM ENCHANCEMENTS 3-1
3.1.1 Overview: A System in ROM 3-1

3.2 MEMORY ASSIGNMENT 3-2
3.2.1 ROM Location Command Option 3-3

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

3.3 ROM SEGMENTS 3-3
3.3.1 Putting Segments in ROM 3-3
3.3.2 ROMMABLE Directive 3-4

SECTION 4 OVERLAYS 4-1

4.1 LINK COMMAND WITH OVERLAY OPTION 4-2

4.2 CODING AN OVERLAY 4-2

4.3 LINKING AN OVERLAY 4-3

4.4 LOADING AN OVERLAY 4-3

4.5 OVMERGE UTILITY 4-3
4.5.1 Overlay Loading Services 4-3

4.6 CLOSING AN OVERLAY FILE 4-4

4.7 ERROR CHECKING 4-4

SECTION 5 PROGRAM EXAMPLE 5-1

5.1 MULTI-TASKING PROGRAM 5-2

5.2 THE MEMORY MAP 5-10

APPENDIXES

A TYPE CODE FUNCTIONS A-1

B LINK EDITOR UTILITY MESSAGES B-1

B.1 LINK EDITOR WARNING MESSAGES B-1

B.2 LINK EDITOR UTILITY ERROR MESSAGES B-4

B.3 LINK EDITOR RUNTIME ERRORS B-7

FIGURES

Figure		Page
1-1	The Development Process	1-3
1-2	Modules to the LINK	1-5
1-3	Shareable Segments	1-7
1-4	Layouts for TC Records	1-9
1-5	TCDUMP-Generated Listing	1-10
2-1	Command Line Example	2-1
2-2	Command Line Arrangement	2-10
2-3	LINK Output to CI/SL Device	2-10

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Figure		Page
2-4	FILE USAGE SUMMARY	2-11
2-5	Module Selection	2-13
2-6	Secondary NAMs	2-14
2-7	Unconditionally Loading the First File	2-16
2-8	LIBRARY LOADING the Second File	2-17
2-9	LIBRARY LOADING the Third File	2-18
2-10	Final Result of Library Load Example	2-19
2-11	Memory Map for Previous Example	2-20
2-12	LINK Output to CRT Screen	2-21
3-1	Memory for a ROMable System	3-2
4-1	Overlays in Memory	4-1

TABLES

Table		
1-1	Type Code Definitions	1-8

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

SECTION 1

LINK EDITOR UTILITY

Linking program modules and assigning memory locations to program data are two functions of the Link Editor Utility. The Link Editor's procedure of handling these and other functions is discussed in this manual. The manual provides general information on the LINK Editor process, a look at the implementation of overlays and a discussion on the Romability of programs.

The object modules produced by a compiler or assembler cannot be executed. These modules must be processed by a service module that resolves the addresses and symbolic references and places the object modules in a form acceptable to a loader. The Computer Automation, Inc., LINK Editor Utility (LINK) is designed to perform this resolution process. The LINK Editor Utility coordinates communication between modules so that several system capabilities can be combined. The LINK also is used to produce Computer Automation, Inc., software executable in a multi-tasking environment.

1.1 REMOVABLE FILES

The LINK Editor Utility consists of several files which consume a large amount of disk storage space. Because floppy disks cannot handle the volume of hard disks, it can be necessary to delete some of the LINK files when executing "floppy only" links. The following files can be deleted from the LINK Editor Utility should disk space become a problem.

- LNKDEBUG.POB
- LNKXREF.POB

These two files, DEBUG and CROSS REFERENCE, can be removed from the disk using the DELETE command. (Refer to the OS4 System User's Manual, Section 2 on OS4/Programmer Communication.) A copy of the original LINK diskette should be made before deleting any files. This precaution provides the user with back-up support.

1.2 TERMINOLOGY

This manual employs several terms that are not commonly used or have a specific meaning to the LINK Editor Utility. For these reasons, the following terms and acronyms are defined briefly.

LINK	Acronym for the LINK Editor Utility.
TASK	The characteristics of a task (shareable data, nonshareable data, resource allocations, etc.) which define a task to the system.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

MODULE	A portion of an assembled or compiled program.
DUMMY MODULE	Instructions designed to serve as an error routine.
OBJECT PROGRAM	The result of a language translation.
OBJECT TEXT	Typecode records representing the data, instruction, and control information of a program.
TYPE-CODE	A one-byte control field which informs the system of the type-code record layout format used for a particular type-code record.
SEGMENTS	A named area in a program reserved for all data having the same attribute. Attributes accepted are shareable, common and rommable.
TEMPLATE	A description of the non-shareable region of a program which is expanded at runtime.
NAM	A named entry point to a module.
EXTERNAL	A symbol used in one module and defined in another module.

1.3 PROGRAM DEVELOPMENT

An operating system provides several facilities (service programs and language translators, etc.) for the development and maintenance of software programs. These facilities are called and executed in an ordered sequence using a standard set of procedures. Because the LINK Editor Utility is a service program called and executed during these procedures, a condensed description of the development process, as shown in Figure 1-1, is given.

A source program, consisting of a set of source language statements, is input to the system and stored on a file created by the programmer. These language statements are arranged into modules which logically interconnect with each other using language directives. The modules also can be constructed to execute independently.

The source program is presented to a language translator (such as a compiler or assembler) through operating system commands. The translator produces an object module written in object text which consists of a series of type-code records. The TC records, as they are called, define the data, instructions, and control information of a program. The language translator usually outputs a source listing to the LO device unless the operator chooses an option which suppresses the listing.

The LINK Editor Utility processes the TC records further and produces more type-codes. The LINK processing results in a binary file which is later processed by a loader. The LINK resolves external references in modules, creates templates, and assigns memory locations. These locations can be either absolute or relative to load point depending on program design. The LINK also produces a LINK map which specifies memory locations, lists symbols for the various tasks, and provides other information as indicated in section 5 of this manual. The LINK map can be suppressed if the appropriate option is specified on the LINK command line.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

©1980 COMPUTER AUTOMATION, INC.

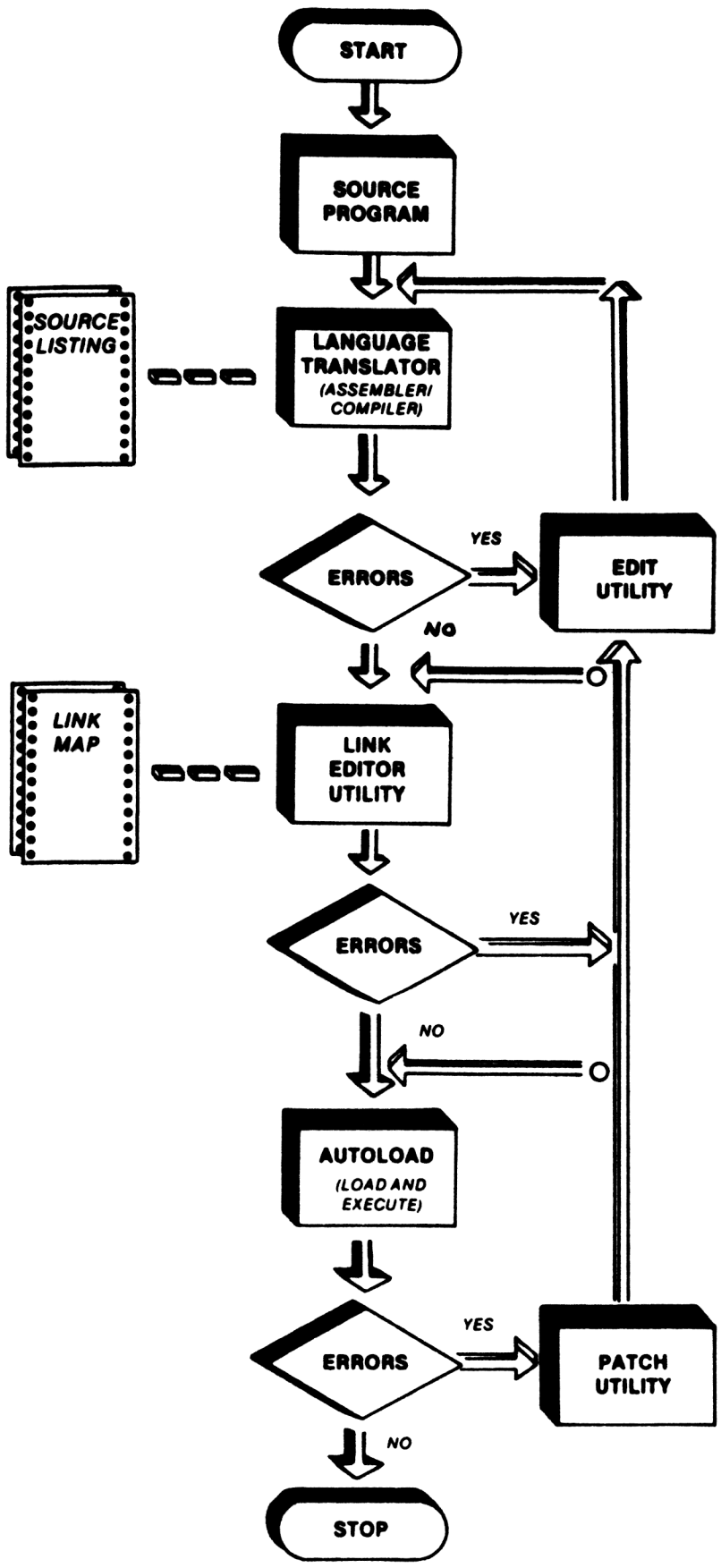


Figure 1-1. The Development Process

After a successful LINK, the binary file, produced by the utility, can be autoloaded by the Autoload Program (or loader) and tested. Source edit programs, Debugging programs, and other utilities are usually available to assist the programmer in developing a system. When a syntax or logical error is detected, the programmer uses the source edit program for error correction.

1.4 TASKS AND THE LINK EDITOR

Unlike RTX4, the LINK considers a task in terms of shareable data, non-shareable data, resource allocations, etc. Thus, as the result of the link process, the LINK defines a task to the system so that the task is executable at runtime.

A task is defined to the LINK Editor Utility by the TASK directive (assembly language) and the TASK statement (FORTRAN). In assembly language, the TASK directive is generated automatically by the TDB:A macro if initialized-Y is requested. (Additional information on initialized-Y is provided in the Real-Time Executive RTX4 User's Manual, Section 3: TASKS and the TASK statement).

In a FORTRAN program, the TASK statement is used to generate a task descriptor block in a main program and declares that program as an RTX4 task. A TASK statement appearing in a subprogram or function declares that the subprogram or function is to be used only by the listed tasks. Subprograms or functions without a TASK statement can be used by any task. (Additional information regarding the FORTRAN TASK statement is provided in the NAKED MINI® 4 FORTRAN IV Language Manual, Section 9: TASKS and the TASK statement.)

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

1.5 MEMORY AND THE LINK EDITOR

The LINK Editor Utility recognizes the object code generated by two assembler directives to allocate memory locations.

- ABS directive which allows the user to allocate portions of a program to specific areas of memory.

- REL directive which allows the user to leave portions of a program relative to memory availability.

Code requiring absolute locations in memory is stored at the addresses specified by the input program using the Absolute (ABS) directive. The user is responsible for these addresses and the amount of memory used. Neither the assembler nor the LINK follows the ABS address path to calculate memory requirements.

Often, it is convenient to postpone the decision of program location in memory by using the REL directive. When this happens, the assembler, compiler or LINK flags all items which depend on memory locations as being relocatable and outputs the data as if the memory location started at the specified offset.

When the actual memory locations are determined, the program (LINK, memory loader, etc.) adds the actual memory start address of the program to each item flagged as relocatable to set-up memory. The ability to relocate code in memory is almost mandatory in multi-tasking when calling and recalling shareable and common code.

LINK handles the linking of absolute code and named relocatable segments in the following manner.

1. The shareable segments are allocated memory when encountered, beginning at the effective load offset determined by the user at link time.

2. The template of the last task, encountered by LINK, is allocated memory separate from but contiguous with the shareable segments. The template for the next-to-last task segment encountered is allocated memory contiguous with that task segment, and so on.

3. The non-sharable code for the initialization task is allocated memory contiguous with the other task segments.

1.6 MODULES AND THE LINK EDITOR

The LINK Editor assumes a module represents a portion of a separate assemblable or compilable program stored in a file. Several modules can logically interconnect with each other through the use of language directives or statements or execute independently.

Modules can range in size from no instructions to a large complex entity that requires all of the available space in memory. Modules can be either absolute or relocatable. All the entry points of a module are introduced to the LINK by the NAM directive (assembly language) and SUBROUTINE statement (FORTRAN). Each module terminates with an END statement. One file can contain several modules (Figure 1-2).

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

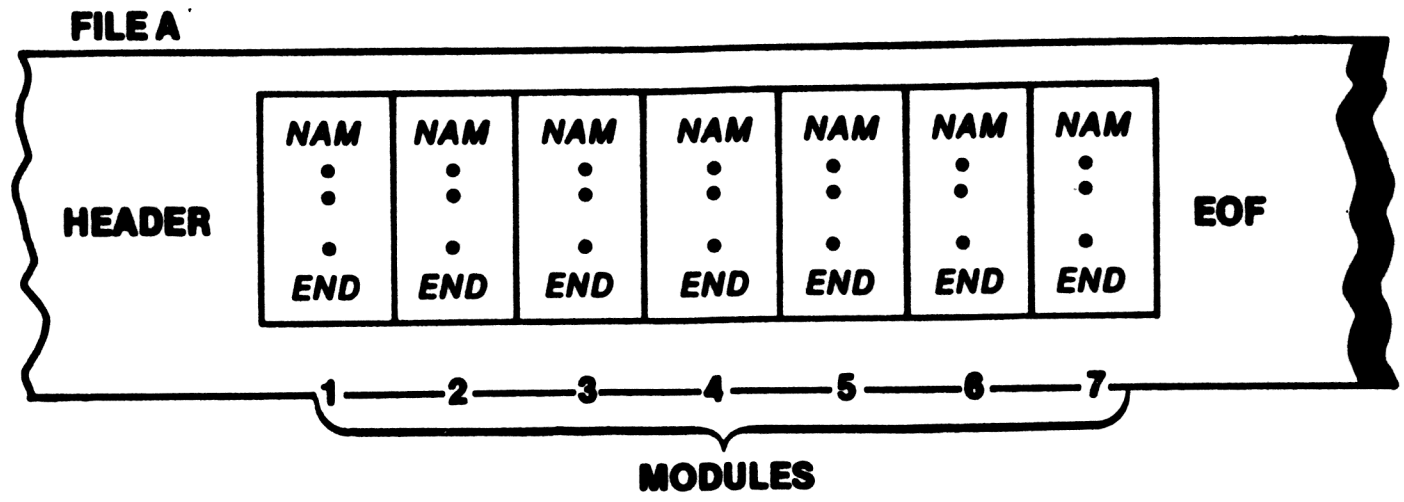


Figure 1-2. Modules to the LINK

1.7 DUMMY MODULES

Occasionally, a reference to a module not selected for system configuration can occur in modular programming with Computer Automation, Inc., software. To avoid operational problems, these references are channeled to dummy modules: instructions designed to serve as error routines. For example, a Computer Automation, Inc., system can be initialized with pointers to all RTX4 services. When a user chooses not to select a service for his configuration, the pointer for that service is resolved by a dummy module so system operation is not disrupted.

A user-defined module called SRF:ER is an example of a dummy module. This module can be used to resolve references of secondary externals when the references are not satisfied during the LINK process (see Section 2).

1.8 TEMPLATES

Instead of writing several copies of the same data region to memory, the LINK constructs one description of the data region called a template and stores that in memory. A template consists of a series of typecodes which describe all the characteristics of the data region.

In RTX4, a programmer can create a task descriptor block pointing to the template even when the data region is unknown. The task descriptor block macro generates a TASK directive. The result of that directive informs the LINK to allocate space needed for the data region. Then, at execution time for that task, RTX4 expands the template, initializes the program, points the Y register to the expanded template, and starts the task.

The LINK generates one template per task. Each template is allocated the next highest available block of memory.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

1.9 SEGMENTS

Quite often, a programmer can describe the same area of memory for several different programs (i.e., FORTRAN COMMON) or describe several different areas from one program (i.e., separation of procedure and data). Both conditions can be programmed to occur by specifying segments when using the LINK. The LINK allows the programmer to specify up to 254 named areas or segments in each program. A segment refers to an area reserved for all data having the same attribute. The LINK recognizes three attributes:

- shareable
- common
- ROMMABLE

The LINK has the responsibility of collecting the data for each segment from the modules loaded during the linking process. The LINK either concatenates or overlaps the data depending on the attributes indicated. (See Section 3 for information regarding the ROMMABLE directive.)

1.9.1 Common Segments

Common segments overlap and are used as working area for a program or task. Usually multiple tasks have access to a common segment. The LINK ensures that location zero (0) of each declaration (definition) references the same location of a segment. The total segment size is the largest block of common defined in any module loaded.

Regardless of whether a segment is common or not, a module can reference the first location of a segment by declaring the segment in the module to have a length of zero. Link causes all labels in a zero length segment to point to the beginning of the segment.

1.9.2 Non-Common Segments

An area of memory dedicated to one task or one program constitutes a non-common segment. Non-common segments are concatenated in memory when the linking process is completed. The total size of the total non-common segment is the sum of the sizes of all non-common segments (of the same name) defined in all the modules loaded.

1.9.3 Shareable Segments

The shareable attribute in a segment informs the LINK that only one copy of the segment is needed in memory. The code (procedure and data) is not modified and no timing restrictions control the execution of the segment or the number of users referencing the data (Figure 1-5).

In some cases, a segment can be classified as shareable although it does modify itself if the programmer is cautious of the segment's accessibility. Particular care should be taken in serializing so that two tasks are not executing the code interspersed. This situation occurs if two or more tasks share the same data region when accessing a particular segment.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

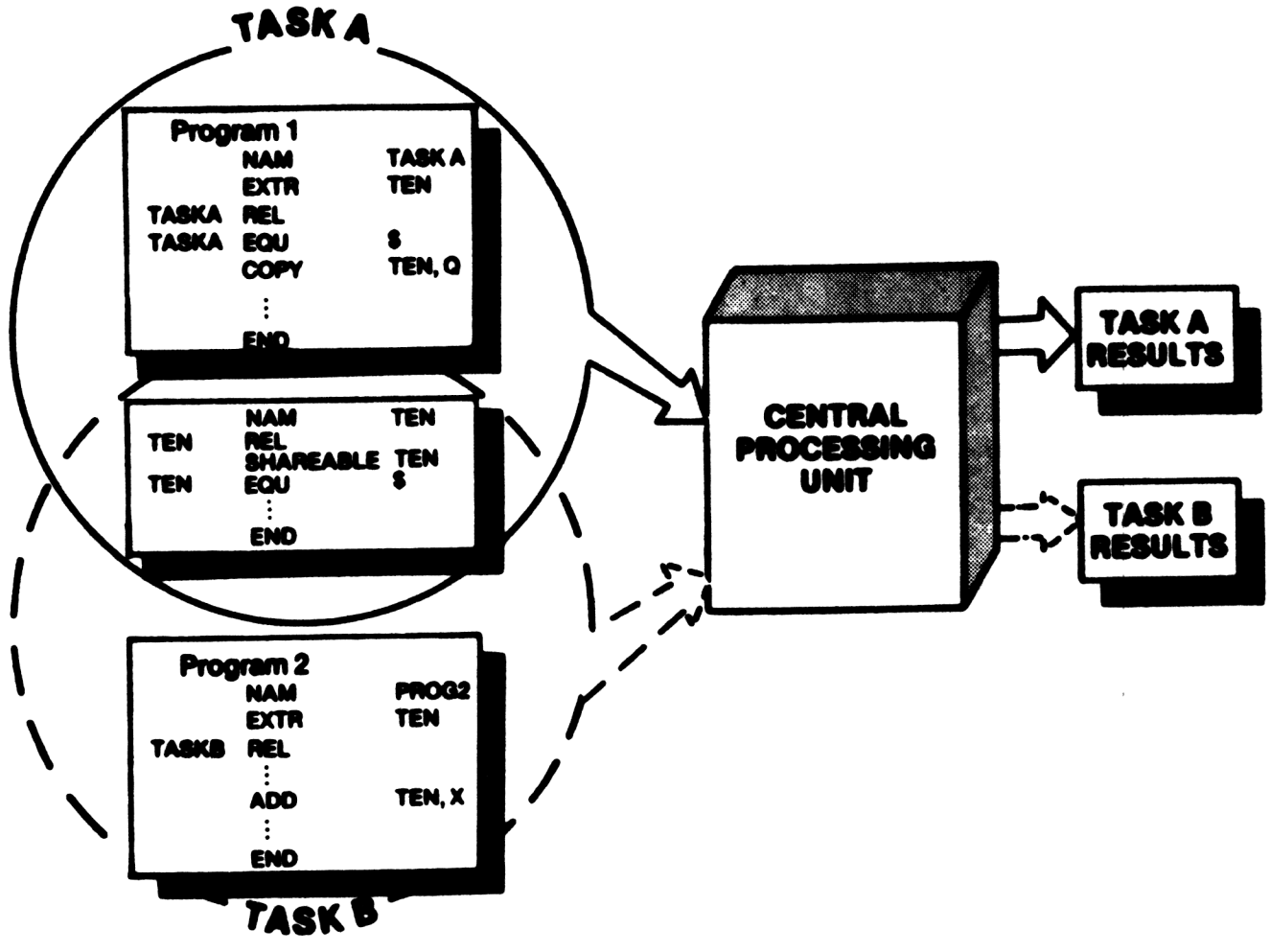


Figure 1-3. Shareable Segments

1.9.4 Non-Shareable Segments

Non-shareable data refers to the code (procedure and data) that is modified whenever executed or reserved for private use. For this reason, all tasks that need to reference a non-shareable segment must reference their own copy of the segment. Furthermore, each activation of the same task must access its own copy of the non-shareable segment. Producing copies of non-shareable segments is accomplished by the LINK and RTX4. The LINK consolidates the data into a template, then RTX4 expands the template at runtime.

1.10 OBJECT PROGRAMS

After a source program is digested by the language translator, the output produced is an object program or object text. The LINK is the primary consumer of object text so knowing the semantics of this language is advantageous when communicating with the LINK. Object text consists of a series of Type-Code (TC) records. Each record starts with a one-byte control field called a type-code. The type-code indicates to the LINK what TC layout format to expect before reading the record. Most type-codes signal the beginning of a TC record of some fixed length and layout; a few type-codes require a count field to indicate the exact length of the associated TC record. No special control character is needed to indicate the end of a TC record because a new type-code indicates a new record. Table 1-1 lists all valid type-codes generated by the assembler and/or recognized by the LINK.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

1.10.1 TC Record Layouts

All NAKED MINI® 4 TC records are in one of 13 standard layouts. These layouts are built from a few basic elements, each with a fixed length and function. These elements are identified in the layout presentations as follows:

TC	One byte type-code.
BYTES	One byte indicating the number of bytes in the layout.
COUNT 16	Two bytes (16 bits) which contain an unsigned binary number having a value between :0000 and :FFFF, or 0 and 65535 decimal.
COUNT 8	One byte (8 bits) containing a positive binary number, with a value between :00 and :FF, or 0 and 255 decimal.
FLAGS	One byte of the bits used to identify the attributes of a given segment.
INDEX	One byte representing an index (NAM, EXTERNAL, CHAIN, TASK NAME (TN) or SEGMENT) to a symbol list.
MASK	Two bytes defining the field in the skeleton affected by the operation.
SHIFT	One byte indicating the number of places to shift.

The type-code name, function, record layout, and its order of occurrence in a program unit can be found in Table 1-1. Refer to Appendix A for type-code functions.

Table 1-1. Type Code Definitions

TYPE CODE	RECORD LAYOUT	ORDER IN PROG. UNIT	NAME	FUNCTION
:02[1]	C	7	TCENDA	End program, absolute
:03[1]	C	7	TCENDR	End program, relocatable
:04[1]	C	6	TCORGA	Origin, absolute
:05[1]	C	6	TCORGR	Origin, relocatable
:06[1]	D	6	TCTAB	Data, absolute
:07[1]	D	6	TCTRLW	Data, relocatable
:08[1]	E	6	TCRESF	Reserve and fill here
:09[1]	C	6	TCBRLW	Byte address constant
:18[1]	C	1	TCBGN3	Begin program
:40[2]	A	6	TCNULL	Null
:4A	F	2	TCENTP	List primary definitions
:4B	F	4	TCEXTP	List primary externals
:4C	F	4	TCENTS	List secondary definitions
:4D[2]	F	4	TCEXTS	List secondary externals
:4E	F	4	TCCHNP	List chains
:4F[2,3]	F	4	TCSGMT	List segment names
:64	G	6	TCINTR	Intermediate access, relocatable
:66	A	6	TCZOP	Stack definition
:67	B	6	TCDENP	Primary definition
:68	B	6	TCDENS	Secondary definition
:69	B	5 & 6	TCLOC	Location counter
:6B	B	6	TCNCHP	Chain element
:6C	B	6	TCHCHP	Chain head
:6D[2]	J	6	TCNOTE	Note record
:6F[2,3]	H	4	TCSGML	List segment attributes
:74	B	6	TCTEXP	Text external (primary)
:75	B	6	TCTEXS	Text external (secondary)

[1]Processed by AutoLoad.
[2]Ignored by AutoLoad.
[3]The TCSGMT for a given segment must precede the corresponding TCSGML for that segment.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

The layouts for TC records (Figure 1-3) are as follows.

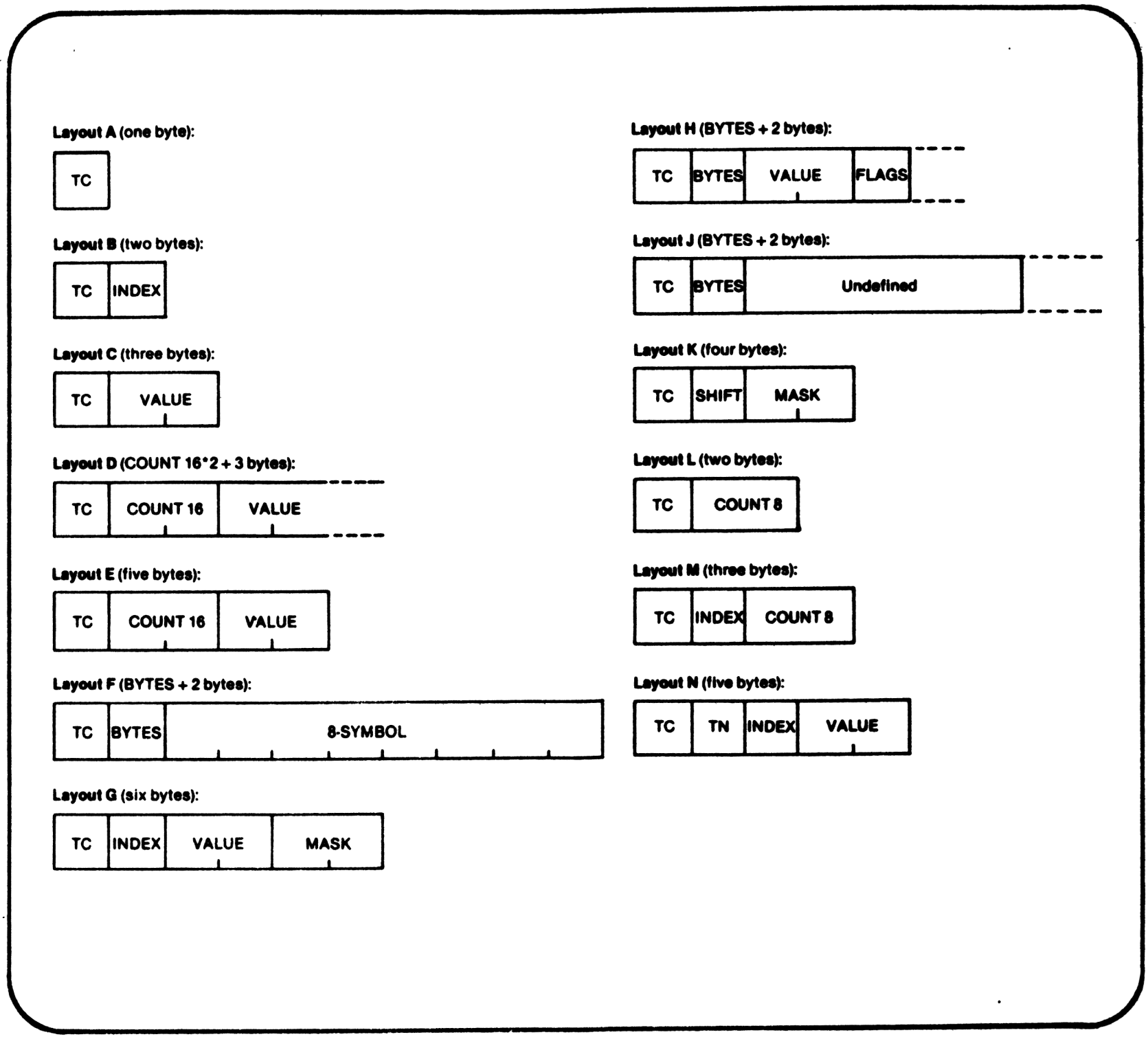


Figure 1-4. Layouts for TC Records

1.10.2 User Type-Codes

The type-codes ranging from :F0 to :FF hexadecimal (decimal equivalent 240 to 255) are reserved for users who want to construct their own language processors and loaders. These type-codes are not used in Computer Automation, Inc., software.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

1.10.3 TC Dump

Figure 1-4 illustrates the printed output produced by the TCDUMP utility. The first two digits of each record comprise the type-code. The format and meaning of subsequent digits depends on the layout associated with that type-code, as discussed. The second and third columns list the name and function of the type-code in the corresponding line of the listing.

One of the object format records listed in Figure 1-4 is boxed. This line is discussed following the illustration to show how TCDUMP output can be read.

TCDUMP-GENERATED LISTING

```
18 0000
4A 10 A D D R S       C I N C O
4F 18 0 S E G M E N T B L A N K       S E G 4
4F 10 S E G 2         S E G 1
6F 0F 00 00 02 00 00 00 00 64 01 00 64 00 00 64 00
4B 18 A D D R 4       A D D R 3       A D D R 2
4B 18 A D D R 1       C U A T R O     T R E S
4B 10 D O S           U N O
4E 18 T H R E E       O N E           T W O
69 01
05 0000
69 03
05 0000
67 01
06 0001 8D00
67 00
6B 02
08 0059 6666
6C 01
6C 02
74 07
74 06
74 05
74 04
06 0003 6666 6666 6666
69 00
04 132C
68 02
08 0063 4444
69 04
05 0000
06 0001 7777
68 00
74 03
74 02
74 01
74 00
64 03 00 01 FF FF
08 005D 7777
69 00
04 3000
06 0001 0000
08 0063 CCCC
69 02
05 0000
08 0064 9999
02 FFFF
```

TYPE CODE	FUNCTION
TCBGN3	Begin program
TCENTP	List primary definitions
TCSGMT	List segment names
TCSGMT	List segment names
TCSGML	List segment attributes
TCEXTS	List primary externals
TCEXTS	List primary externals
TCEXTS	List primary externals
TCCHNP	List chains
TCLOC	Location counter
TCORGR	Origin, relocatable
TCLOC	Location counter
TCORGR	Origin, relocatable
TCDENP	Primary definition
TCTAB	Data, absolute
TCDENP	Primary definition
TCNCHP	Chain element
TCRESF	Reserve and fill here
TCHCHP	Chain head
TCHCHP	Chain head
TCTEXP	Text external (primary)
TCTEXP	Text external (primary)
TCTEXP	Text external (primary)
TCTEXP	Text external (primary)
TCTAB	Data, absolute
TCLOC	Location counter
TCORGA	Origin, absolute
TCNCHP	Chain element
TCRESF	Reserve and fill here
TCLOC	Location counter
TCORGR	Origin, relocatable
TCTAB	Data, absolute
TCNCHP	Chain element
TCTEXT	Text external (primary)
TCTEXT	Text external (primary)
TCTEXT	Text external (primary)
TCTEXT	Text external (primary)
TCINTR	Intermediate access, relocatable
TCRESF	Reserve and fill here
TCLOC	Location counter
TCORGA	Origin, absolute
TCTAB	Data, absolute
TCRESF	Reserve and fill here
TCLOC	Location counter
TCORGR	Origin, relocatable
TCRESF	Reserve and fill here
TCENDA	End program, absolute

Figure 1-5. TCDUMP-Generated Listing

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

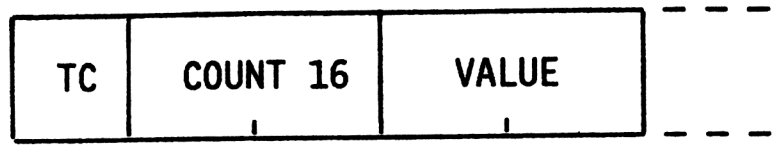
© 1980 COMPUTER AUTOMATION, INC.

Line 25, of the listing, reads:

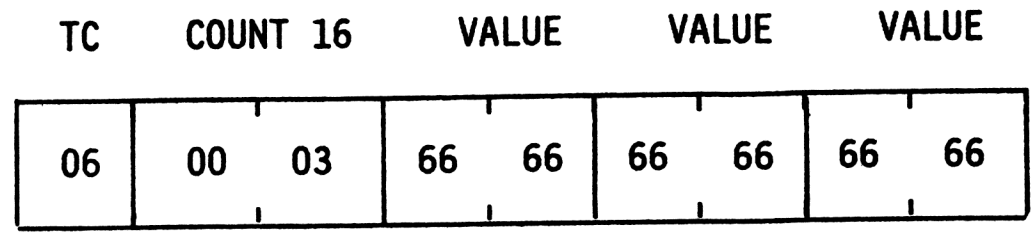
06 0003 6666 6666 6666

The type code :06 (TCTAB--Data Absolute) indicates the following layout:

Layout D (COUNT 16*2+3 bytes):



The COUNT field indicates how many VALUE fields follow. The VALUE fields are taken as a block of absolute data. Thus line 25 can be read as:



The type-code causes the data to be deposited at the current value of the location counter and the location counter to be incremented.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

SECTION 2

LINK EDITOR OPERATION

The LINK Editor Utility converts object code into a form which can be loaded by OS4, AutoLoad, or DEBUG4. Type-code record types are accepted as input, but only those types compatible with these loading methods are produced as output. Both input and output are in standard NAKED MINI® Family object format.

By default, LINK functions according to the following procedures: LINK generates a linked module in relocatable format with an offset value of 0. The first input medium is linked unconditionally meaning all modules are linked into the output module. (If any multiply-defined symbols are detected during an unconditional link, both modules are linked into the output module but only the first instance of the symbol is referenced.) Subsequent input media are linked in library mode meaning only those modules which resolve a prior unresolved primary external reference are linked into the output module. The first-pass output of the link-edit process is written to files named LNKSHARABLE.TMP and LNKNONSHRBL.TMP on the XX unit. During the latter phases, LINK generates memory maps to the LO unit.

2.1 LINK COMMAND LINE

A LINK command line (Figure 2-1) provides the communication between the LINK and the user. The command is used to link one of more object modules into one binary file. An arbitrary name is given to the binary file which results as the loadable module.

COMMAND — FILES LINKED

/LINK PARTNO=FILEA+FILEB+FILEC+DEBUG(UN)

BINARY FILE NAME — LINK COMMAND OPTION

PARTNO.BIN is the name of the binary file which is the result of the link. Every module of FILEA is unconditionally loaded by default. The file called DEBUG also is unconditionally loaded because of the UN option on the command line. Modules of FILEB and FILEC are loaded as needed.

Figure 2-1. Command Line Example

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

2.1.1 Using the LINK Command Line

The LINK command instructs OS4 to load and execute the LINK utility program. The command makes three specifications about the operation to be performed.

- It identifies the input media. For instance, the file(s) and/or device(s) containing the object code and library files to be linked.
- It specifies any LINK options to be invoked.
- It defines the destination medium. For instance, the file or device to which the linked module is to be written.

The programmer has a great deal of flexibility in supplying input/output information. The various ways of issuing the LINK command are described individually on the following pages. The following rules apply to all forms of the command, except where stated otherwise.

- When the LINK command line, input on a CRT, ends with a plus sign (+), a comma (,), or an equals sign (=), or when the line ends on parenthesized data (i.e., data preceded by a left parenthesis but not followed by a right one), LINK generates the prompt:

  LNK..

  to the CI unit to indicate that the specified LINK operations are completed and additional input can be accepted. The programmer can enter a continuation of the LINK command from the CI unit. When the programmer enters a line which does not end with one of symbols specifed above, LINK assumes the line as the final command line command. There is no limit on the number of continuation lines that can be entered in this manner.
- The file name extension is optional in a file identifier specified as an input medium. If the extension is omitted from a file identifier specified as an input medium, LINK searches first for the .LIB version and then for the .OBJ version. An error occurs if neither version is found.
- The file name extension also is optional in a file identifier specified as the destination medium. If the extension is omitted, LINK writes the linked program to the .BIN version of the file. However, if Format 5 is used (see Section 2.1.3) the SO assignment command must specify the file name and extension.
- If the first input medium is a file, the file name also is optional in a file identifier specified as the destination medium. Thus, if the destination medium is a file-type device, LINK writes the linked program to a file on that device. This file has the same file name as the first input file and the .BIN extension. This rule does not apply if Format 5 is used.
- If both the first input medium and the destination medium are non-file-type devices, the linked program is written to the file LINK:OUTPUT.BIN on the destination device.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

- If the destination medium is a file, the command overwrites any current contents of that file. The LINK command has no effect on input files or devices.

A programmer can abort the link-edit process anytime the LINK is awaiting input. The operator enters ABORT! from the CI unit. Control returns to the Executive.

2.1.2 Command Line Options

The programmer can override default link-edit procedures or request non-default procedures by specifying LINK options, separated by commas and enclosed in parentheses, on the LINK command line. The options can be input in any order.

LINK options are processed in the order input. All link options can be abbreviated to the number of characters required to uniquely identify them. The underlined characters in the option descriptions, below, indicate the abbreviations accepted by the LINK. Where option names differ from those in earlier versions of LINK, the older abbreviations, listed in parentheses, are still supported.

In some of the option descriptions, the term "task set" is used in referencing a set of tasks. A task set can take one of two forms.

```
Form 1:  task1[+task2+...+taskn]

Form 2:  EVERYBODY  -  task1[-...-taskn]
                       (task1[+...+taskn])
```

In either form "task" is a valid task name. "EVERYBODY" represents the entire set of tasks known to LINK.

NOBINARY (NB)

This option suppresses the generation of a linked module.

BINARY

This option re-enables the generation of a linked module (default).

MEMORY=constant (MM)

This option sets the effective high memory limit to the value of the specified constant. The default is a high memory limit of :FFFF.

OVERLAYS({segment=olay(label[.label]...)[+olay(label[.label]...)+...]}[,...])

This option names the segment(s) containing the named overlay(s) ("olay"). Each specified overlay name is an output file name with the extension .OLY. Overlay modules containing primary entry points are selected by specified NAM. All symbols in modules subservient to a primary module are considered internal to the overlay, meaning they are accessible only within the overlay.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

All NAM labels within a module, identified in this option, are considered overlay entry points. A module of an overlay can access any entry points in any overlay subsequently identified in the option. A module cannot directly access an entry point in an overlay specified earlier.

ABSOLUTE .segment[...] =constant

This option writes the named segment in absolute format with an offset having the value of the specified constant and arranges the remaining segments in memory according to the command line.

RELOCATABLE .segment[...] [=constant] (RL)

This option writes the named segment in relocatable format with an offset having the value of the specified constant and arranges the segments in memory according to the command line. (The command line cannot mix shareable and non-shareable segments.)

EXECUTION= symbol[(task)][+constant]
constant[segment]

This option sets a new transfer address, overriding any previously specified transfer address, including any found in the object code. A symbol specified in the EXECUTION option is an external symbol whose value has been defined previously in the linking process. In the case of alpha characters representing hexadecimal values, the hex value is assumed over the symbol representation.

LIBRARY (LL)

This option continues the link operation with conditional linking of the current input medium (default).

NOLIBRARY (UN)(UNCONDITIONAL)

This option continues the link operation with unconditional linking of the current input medium. Library mode is resumed automatically on end-of-file.

NOLISTING (NL)(NOMAP)

This option suppresses listing of memory map information.

LISTING (MAP)

This option re-enables listing of memory map information (default).

PAGESIZE=[length][*width]

This option defines the page size of the LO device, where length is the number of lines per page and width is the width of the page in characters. The default pagesize values are those supplied at system generation time.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

CONCORDANCE (CROSSREFERENCE)(XREFERENCE)(ALPHABETIZED)

This option generates an alphabetized listing of all symbols in the linked program. For each symbol the following are included: the location at which it is defined, the name of the segment in which it is defined, and the names of the tasks that reference it.

NOCONCORDANCE

This option suppresses the concordance listing (default).

EXTWARNINGS

This option prints an error message when missing externals are detected (default).

NOEXTWARNINGS

This option prevents printing of error message when missing externals are detected.

SEXTWARNINGS

This option prints an error message when missing secondary externals are detected.

NOSEXTWARNINGS

This option prevents the printing of an error message when missing secondary externals are detected (default).

LOAD.symbol[(task set)] +symbol[(task set)][+...]
(OD)(DEFINE)

This option causes the loading of programs that define the specified symbols for any of the specified tasks (or for all tasks if no task set is specified).

DEFINE.symbol[(task set)] +symbol[(task set)]= constant[(segment)] symbol[(task)]
(OD)(LOAD)

This option causes the specified symbols for any of the specified tasks (or for all tasks if no task set is specified) to assume the value of the constant or external symbol. In the case of alpha characters representing hexadecimal values, the hex value is assumed over the symbol representation.

SAVESCRATCH

This option saves the scratch file used by LINK.

NOSAVESCRATCH

This option does not save the scratch file used by LINK (default).

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

SYSTEMFILE

This option informs LINK that the file cannot load any non-shareable data and that NAMs are allowed to be backward-referenced.

NOSYSTEMFILE

This option informs LINK that the file can load any non-shareable data and that NAMs are not allowed to be backward-referenced (default).

HOST

This option currently has no meaning to LINK.

NOHOST (NH)

This option currently has no meaning to LINK.

The following options affect the output of commentary information specified by the TCNOTE type code (:6D). Such commentary information is divided into three classes for processing by LINK, depending on the value of the byte immediately following the count in the Type Code Record. These classes are as follows:

Class 1	byte value = 0
Class 2	byte value = :80-:FF
Class 3	byte value = :01-:7F

NOTES

This option copies Class 3 commentary information onto link map (default).

NONOTES

This option does not copy Class 3 commentary information onto link map.

COPYNOTES

This option copies Class 3 commentary information into binary file.

NOCOPYNOTES

This option does not copy Class 3 commentary information into binary file (default).

OBJNOTES

This option copies Class 2 commentary information onto link map in hexadecimal.

NOOBJNOTES

This option does not copy Class 2 commentary information onto link map (default).

COPYOBJNOTES

This option copies Class 2 commentary information into binary file (default).

REVISION EU

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

NOCOPYOBJNOTES

This option does not copy Class 2 commentary information into binary file.

XLATID

This option copies Class 1 commentary information onto link map.

NOXLATID

This option does not copy Class 1 commentary information onto link map (default).

COPYXLATID

This option copies Class 1 commentary information into binary file.

NOCOPYXLATID

This option does not copy Class 1 commentary information into binary file (default).

TOPLOAD

This option generates segment flags (TCSGML), recognized by the loader (I:LOAD), specifying that this module be loaded at the highest available user memory locations. If not specified, the module will be loaded at the lower limit of user memory.

2.1.3 Command Formats

The command formats provide various alternatives to presenting files to the LINK. Five different formats are available.

Format 1: LINK medium[(options)]

This form of the LINK command links object code read from the specified medium and writes the resulting linked module to that same medium. Any of the LINK options described on the preceding pages can be specified.

In this form of the LINK command, the medium is normally a file, because it serves both as a source of object code and as the destination medium and most peripheral devices are uni-directional.

Example: LINK TABLE(AB=100)

This command links code read from the file UF.TABLE.LIB if the file exists or from the file UF.TABLE.OBJ and writes the resulting linked module to the file UF.TABLE.BIN in absolute format with an offset of 100.

Revised E0

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

©1980 COMPUTER AUTOMATION, INC.

Format 2: LINK medium1=[medium2][(options)]

This form of the LINK command links object code read from medium2 (or from the SI unit) according to any specified options and writes the resulting linked module to medium1. The output medium in formats 2 and 3 must be a disk file because nonsequential writes are used.

Example: LINK DF1.DATA.BIN=DF.IND.OBJ(NOLIS)

This command links code read from the file DF.IND.OBJ and writes the resulting object module to the file DF1.DATA.BIN. The output of memory map information is suppressed.

Format 3: LINK medium1=[medium2][(options)]{+[medium3][(options)]}...

This form of the LINK command links object code read from medium2 (or the SI unit) according to any specified options with object code read from medium3 (or the SA unit) according to any specified options, and so on through the end of the command. The resulting linked module is written to medium1. This format is used to link library files to user programs. The following file names can appear in a LINK command to link Computer Automation Inc.-supplied system library files to a user program:

File	Contents	Where Provided
OS4.LIB	OS4 library	OS4 system diskette (F46001)
RTX.LIB	RTX4 library	RTX4 product diskette (F41001) or RTX4 system software paper tape 93410-51
IOS.LIB	IOS4 library	RTX4 product diskette (F41001) or RTX4 system software paper tape 93430-51
SFM.LIB	SFM library	RTX4 product diskette (F41001) or RTX4 system software paper tape 93440-50

Example: LINK DF00=DF1.MNTN.OBJ+DF1.MLHL.OBJ+(NOLIB)+IOS.LIB+SFM.LIB+RTX.LIB+

This command links object code from the file DF1.MNTN.OBJ with code read from the file DF1.MLHL.OBJ, then with object code read from the SA unit in NOLIBRARY mode, and then with IOS, SFM, and RTX library files. Finally, LINK generates its input prompt to the CI unit and waits for additional input from the user. When the programmer completes the command, the linked module is written to the file DF00.MNTN.BIN.

Format 4: LINK parameter[,parameter]...

where each "parameter" defines a link-edit operation to be performed in the form:

[medium1=][medium2][(options)] +[medium3][(options)] ...

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

This format represents the complete syntax of the LINK command and indicates how the link-edit operations described thus far can be combined in a single LINK command. Medium1 defaults to the SO unit, medium2 to the SI unit, and each additional input medium (medium3, etc.) to the SA unit.

Any number of link-edit operations can appear in a LINK command.

Example: LINK TABLE(AB=100),DF1.DATA.BIN=DF.IND.OBJ(NOLIS)

This command performs both link-edit operations illustrated under Format 1 and Format 2.

Format 5: LINK [(options)]

A LINK command with no parameters other than options links object code read from the SI unit and optionally the SA unit and writes the resulting linked module to the SO unit. The SO unit assignment command must specify both the file name and the extension. Options can be specified as just described. If code from the SA unit is to be linked, the NOLIBRARY option must be specified for the SI unit and the LIBRARY option for the SA unit.

Example: Assume that the programmer has issued the following commands to assign logical units:

```
ASSIGN SA=DF.PROG.LIB
ASSIGN SI=DF.PROG.OBJ
ASSIGN SO=DF.PROG.BIN
ASSIGN LO=LP
```

(The ASSIGN command is described in Topic 4.2.1.) A subsequent LINK command in the form:

LINK (NOLIB)+(LIB)

links object code read from the file DF.PROG.OBJ, unconditionally, and links code read from DF.PROG.LIB in library mode. The resulting linked module is written to the file DF.PROG.BIN., and a memory map is generated to the LO device.

2.1.4 Command Line Arrangement for an RTX4 LINK

Because the utility automatically loads the entire first file, the order in which the files appear on the command line should be considered. For an RTX4 link, the first file should contain the INIT:A macro, the ECB:A macro and the Unit Assignment Table or UAT. These elements are necessary for initialization of RTX4 and other services. The code is shareable (does not modify itself) and can be used by all modules. The INIT:A macro also contains the label of the first TASK directive for the first user task.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Figure 2-2 illustrates the order in which data files and system libraries should be introduced to the LINK at link time. Although it is necessary to specify the Standard File Management library (SFM.LIB) on the command line for an RTX4 link, BUF:N is specified only if the SFM library is used.

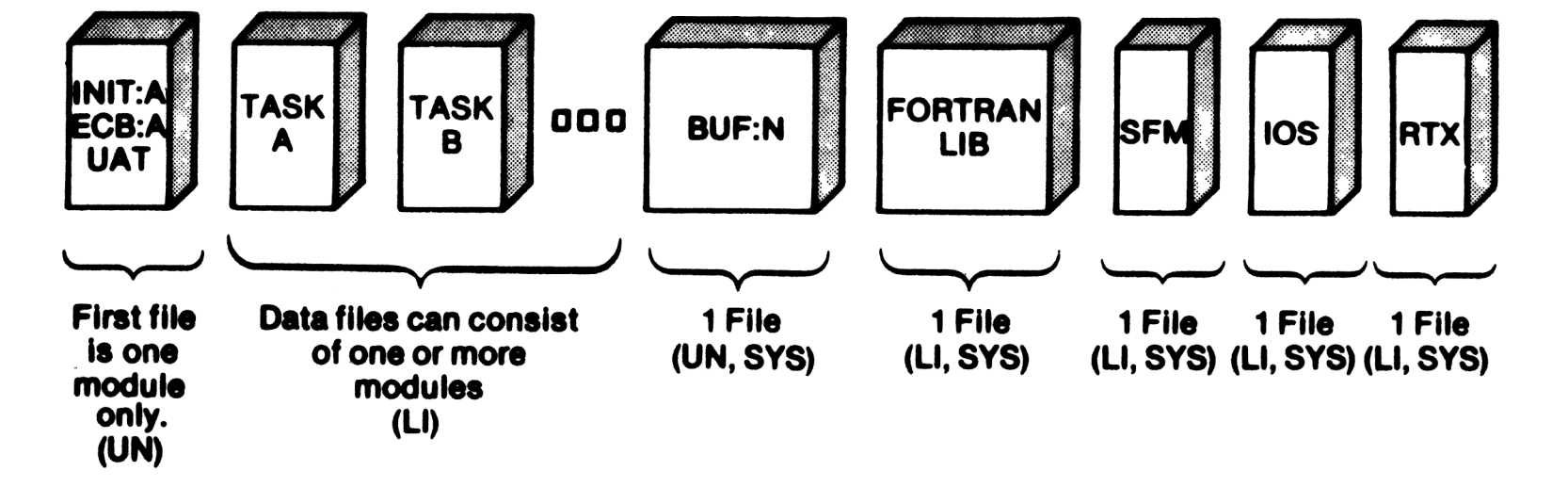


Figure 2-2. Command Line Arrangement

2.2 OPERATOR COMMUNICATION

The first function of the LINK is to process the command line and edit the input for errors. When the input is accepted by the utility as a valid command, the current revision level operating the LINK is presented to the user on the CI/SL device. (CRT screen, Figure 2-3).

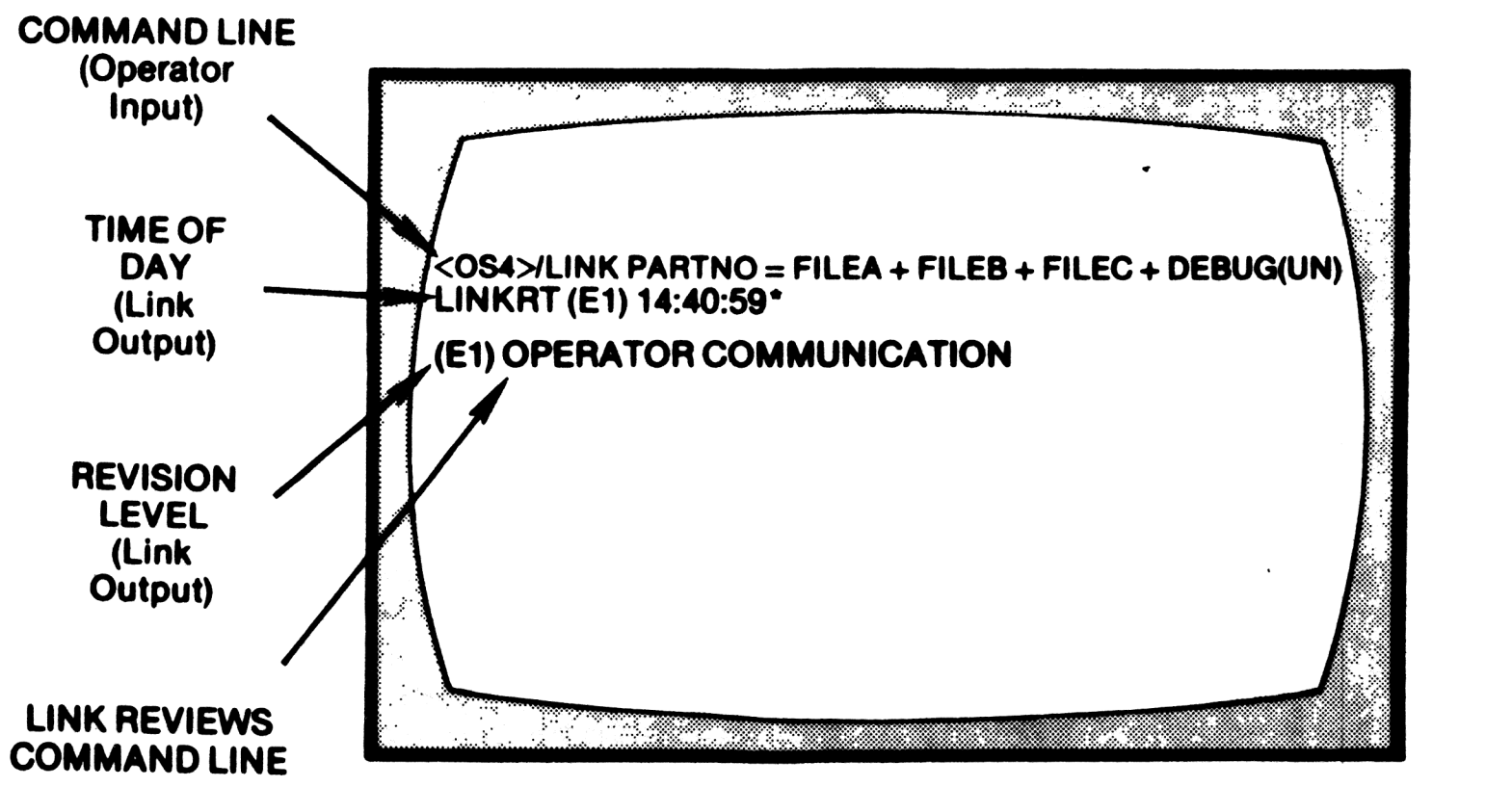


Figure 2-3. LINK Output to CI/SL Device

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

©1980 COMPUTER AUTOMATION, INC.

During "operator communication" the LINK analyzes the command line to set up the input/output files specified by the user. Options are registered together with file extensions. If file extensions are not specified, the utility uses the defaults as explained earlier in this section. A FILE USAGE SUMMARY is constructed at this time and later prints out on the LO device during the selection process as each file is processed.

2.3 SELECTION PROCESS

The LINK Editor Utility conducts a selection process for each file on the command line. As each file is processed, the file name is output to the LO device along with any object notes the file contains (Figure 2-4). An object note can be generated by either an OBJECT or OBJNOTE directive in assembly language. These directives provide means for defining comments to be included with object code, and for generating object code which the Assembler cannot otherwise produce.

The output also shows any options selected by the user together with their related files. Although file extensions are not necessary on the command line, the listing shows the extensions so the programmer knows which file was referenced by the LINK.

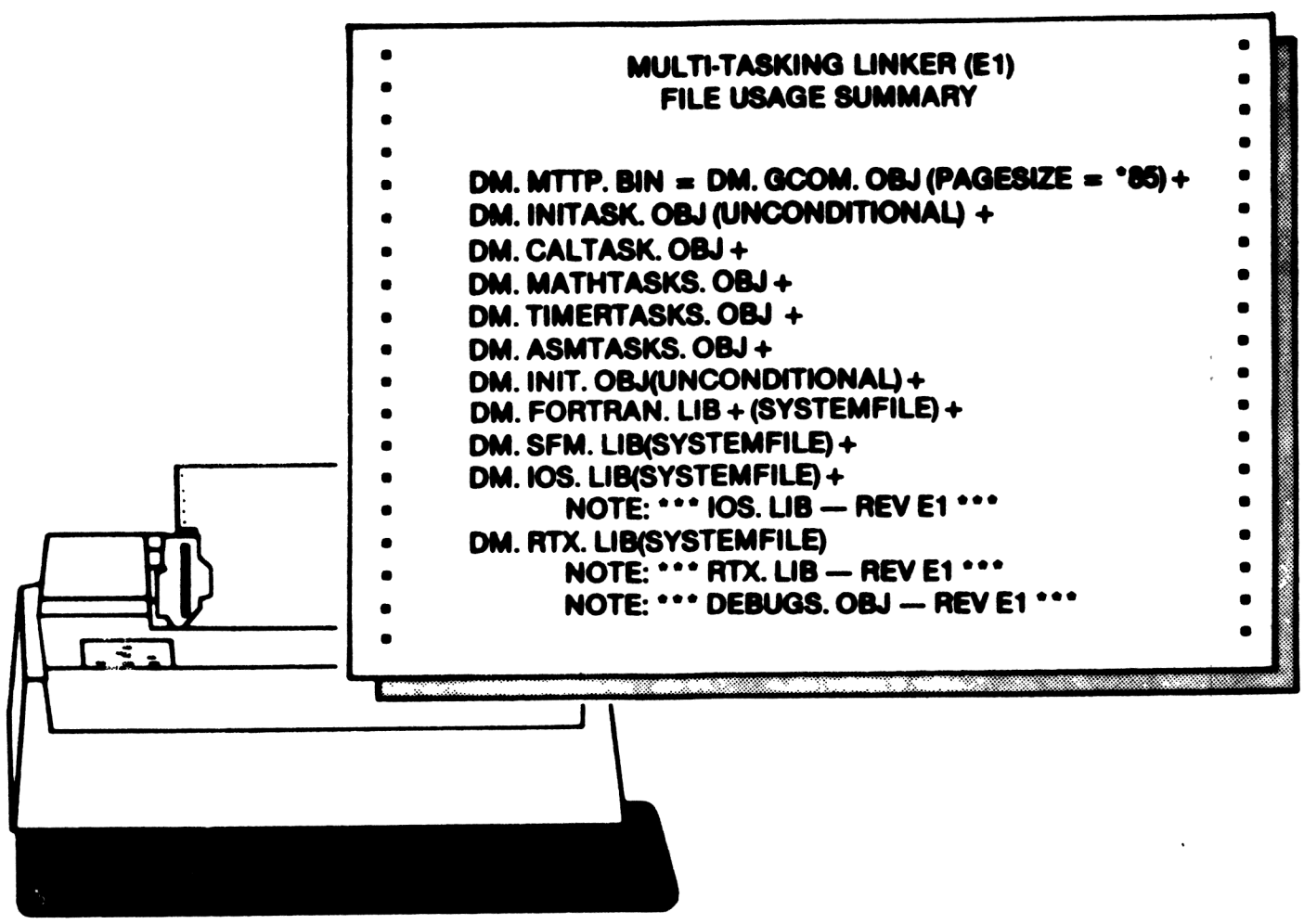


Figure 2-4. FILE USAGE SUMMARY

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

©1980 COMPUTER AUTOMATION, INC.

2.3.1 Module Selection

The LINK responds to assembler or FORTRAN directives by creating an internal library to manage module communication during module selection. The service program does one independent pass through each file to establish the library. For this reason, symbols in the code assembly must be presented to the LINK in a specific order. These symbols are referred to as Named Entry points (NAM) and Externals (EXTR.)

An External is a symbol used in one module to reference code in another. The assembler directive EXTR or the FORTRAN statement CALL is used to declare a symbol definition external to a particular module.

Symbols, within modules, representing entry points to the module are referred to as Named Entry directives. These symbols use the assembler directive NAM and the FORTRAN declarative SUBROUTINE. (Although the FORTRAN directives are not mentioned elsewhere, the LINK reacts likewise to FORTRAN as discussed in this section.)

2.3.1.1 Primary Externals and Primary NAMs

When a programmer references code beyond a particular module boundary using the EXTR directive, the reference is called a primary external. A primary external creates a primary demand entry in the LINK's symbol table. The entry causes the LINK, as the LINK processes other modules, to load the first module containing the definition of the external reference.

A primary demand is satisfied when the LINK processes the object text of another module and finds a primary NAM with the same symbol name as that of the EXTR. The LINK then links the two modules to the same task and drops the primary demand from the symbol table.

To create a primary demand, the EXTeRnal must be detected. In assembly language, however, the mere appearance of a symbol with an EXTR directive is not sufficient to create a primary demand (unless using LOAD directive, refer to Section 2.3.2.1). If the symbol is not referenced somewhere in the module assembly, the translator does not produce object text for the statement and the LINK is not aware of the reference. To make an EXTR statement effective, the symbol must appear in the operand field of a source statement.

Figure 2-5, on the following page, illustrates NAM and EXTeRnal resolution. The structure of a task is based on selection. The modules selected for Task A can be selected for Task B and any module can begin a new task.

Programs must present the externals and NAMs to the LINK in a specific order: an external creating a primary demand before a NAM is detected to resolve the demand. The ordering makes library loading possible (in a single pass over the library) so that only the modules satisfying an outstanding primary demand, at the time they are encountered, are selected.

An exception to library loading occurs in NAM and EXTR resolution when using system modules. There are two ways to declare a system module:

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

©1980 COMPUTER AUTOMATION, INC.

- The Systemfile (SY) option specified on the command line declares the modules in a file accessible to all tasks in the link. The SY option allows backwards referencing of NAMs between tasks.

- A module containing the SYSTEM directive in the assembly code. The SYSTEM directive informs the LINK that all code within the module is shareable (does not modify itself) and can belong to all tasks in the system.

If the SYSTEM directive is coded in a module assembly, the programmer can reverse the directive at LINK time by using the Nosystemfile (NOS) option on the command line.

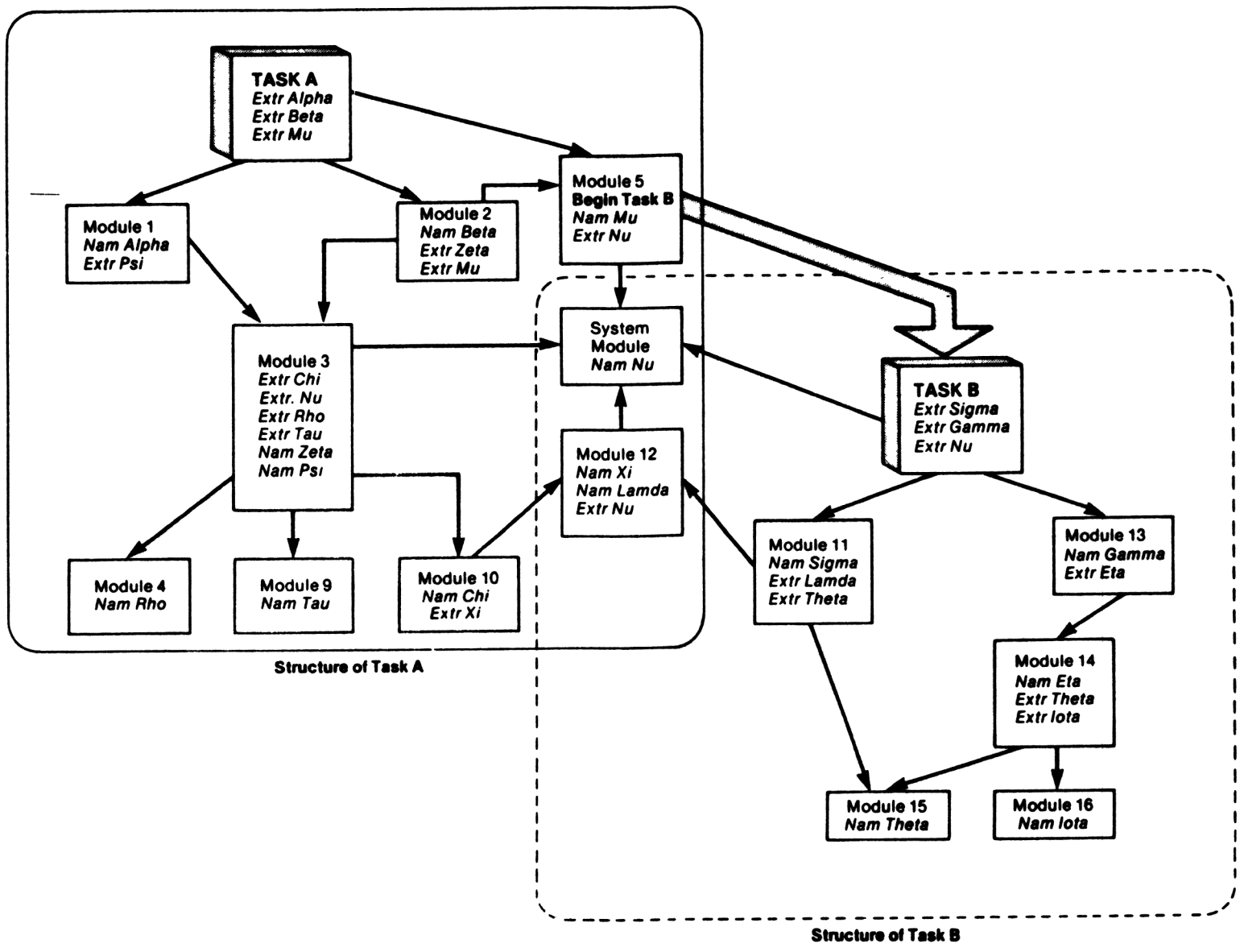


Figure 2-.5 Module Selection

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

2.3.1.2 Secondary NAMs and Secondary Externals

Most often, a confusing part of the linking process is understanding the role of secondary NAMs and secondary EXTRs. For this reason, two facts about these directives should be emphasized:

First: Secondary NAMs and secondary EXTRs do not resolve each other simply because they are both secondary in nature.

Second: These symbols do not have any control over module selection.

A secondary NAM identifies a symbol to be an entry point in a module but the symbol does not satisfy a primary demand to force module loading. The LINK recognizes a secondary NAM only when its module is loaded in response to a demand LOAD, an unconditional load or some other NAM and External resolution. Once the module is selected, the secondary NAM resolves any related primary demand or secondary external (if the secondary external module happens to be loaded). The assembler directive SNAM declares a symbol to be a secondary NAM.

For example, in Figure 2-6, the demand LOAD directive is used to force load either module 1, SMEVAL or module 2, LGEVAL. The LINK has a demand for EVAL but does not recognize the SNAM until one of the modules is loaded. If secondary NAMs were capable of causing a module to be loaded to satisfy a primary demand, both modules would load resulting in a multiply-defined symbol error.

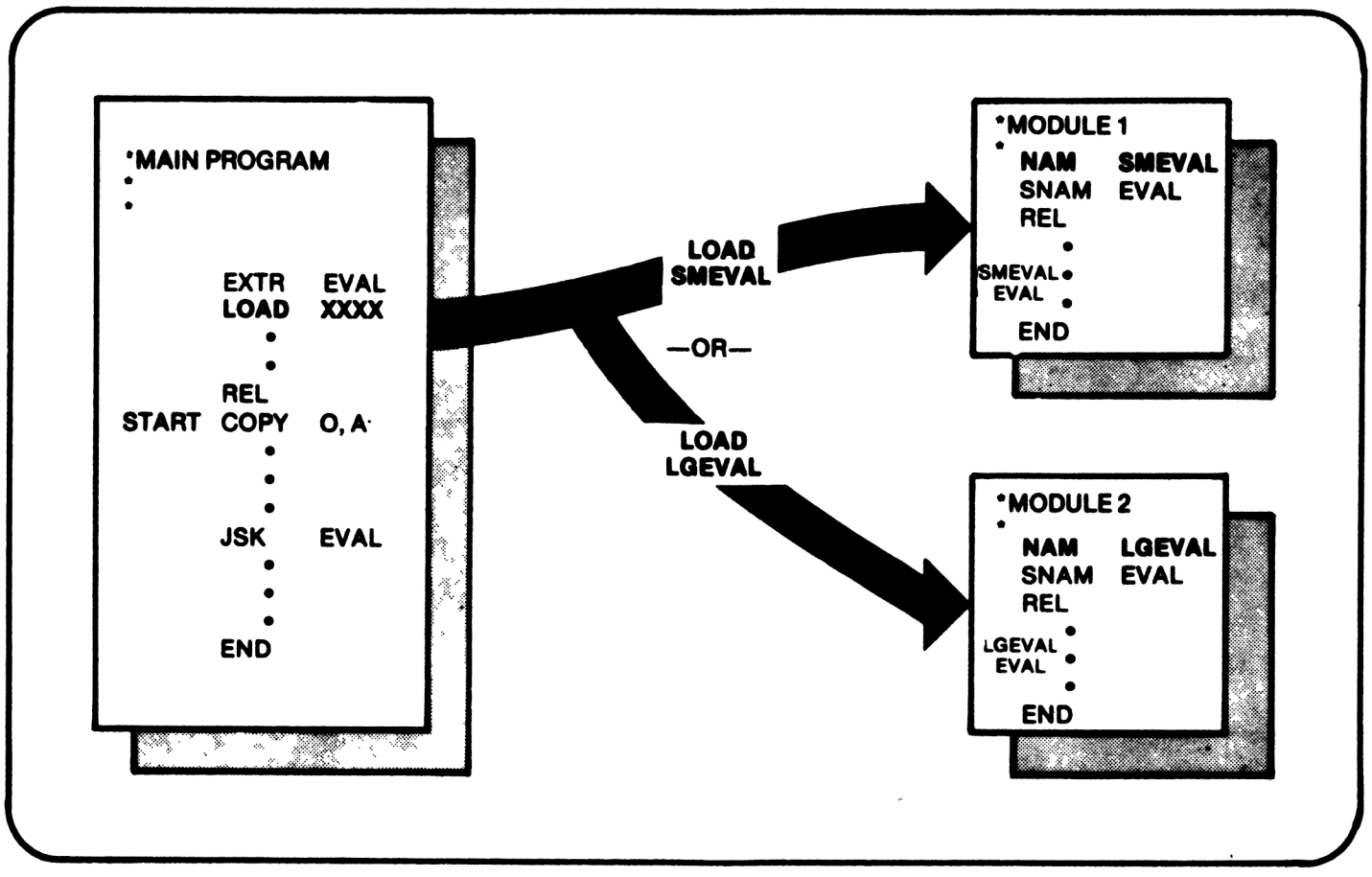


Figure 2-6. Secondary NAMs

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Much like primary externals, secondary externals reference code outside a module's boundary. Although the directive does not force module selection, it does inform the LINK to link its module with the module containing the resolving NAM (if the NAM module is loaded). The SEXT directive is used in assembly language to indicate secondary external symbols.

Since the possibility of ending the link with unresolved secondary externals exist, the programmer should include in the code assembly an error routine called SRF:ER. Although SRF:ER is a user-supplied subroutine, the LINK creates an independent reference to the routine when any unresolved secondary reference is found.

If a primary definition for SRF:ER exists when LINK terminates, all unresolved secondary references are linked to it. If SRF:ER is needed but is not defined, any unresolved secondary references remain unresolved and access location :0. The need for SRF:ER can not be recognized by LINK until the last object program is processed. Therefore, if there are secondary references in the object programs which need the SRF:ER routine, this routine should be at the end of the last object program to be linked.

An example of the SRF:ER Error Routine is as follows:

```
            NAM       SRF:ER
SRF:ER      EQU       $
            WORD      0
            RSK
            END
```

2.3.2 Loading Modules

Module selection during LINK operation depends on the method of loading used:

- Demand LOAD (assembler directive or LOAD,DD,DEFINE)
- Library Loading (LI or LL)
- Unconditional Loading (UN or NOLIB)

Both library loading and unconditional loading can be specified as an option on the command line (using the LI or UN abbreviated option names). Demand LOAD is an assembler directive used in a module's source or can be specified on the LINK command line using LOAD, DD or DEFINE. (Refer command line options.)

By default, the LINK unconditionally loads the first file on the LINK command line and library loads any remaining files.

2.3.2.1 Demand LOAD Directive

This directive creates a primary demand at LINK time for a symbol. Unlike the EXTR directive symbol, the Demand LOAD symbol does not need to be referenced within the module to create a primary demand. The directive is mainly used to load test programs, where an internal reference to a test routine is unlikely to exist, or to force load modules for some other reason. No storage is consumed by the Demand LOAD statement.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

2.3.2.2 Library Loading

Resolving NAM and External references to load modules is called library loading. Primary externals create primary demands and primary NAMs (or secondary NAMs, if loaded) resolve the demands. The best approach to understanding library loading is to step through a library load process. In the command line:

```
/LINK PARTNO=FILEA+FILEB+FILEC+DEBUG(UN)
```

the LINK processes four files (LIB or OBJ file extension assumed) to create the binary file PARTNO. (BIN file extension assumed.) Since FILEA is the first file in the LINK operation, it is unconditionally loaded by default. The LINK constructs a symbol table and resolves any primary demands satisfied within the file. Figure 2-7 shows FILEA and the results of the first load: all modules loaded, 4 NAMs posted, and 2 unresolved externals.

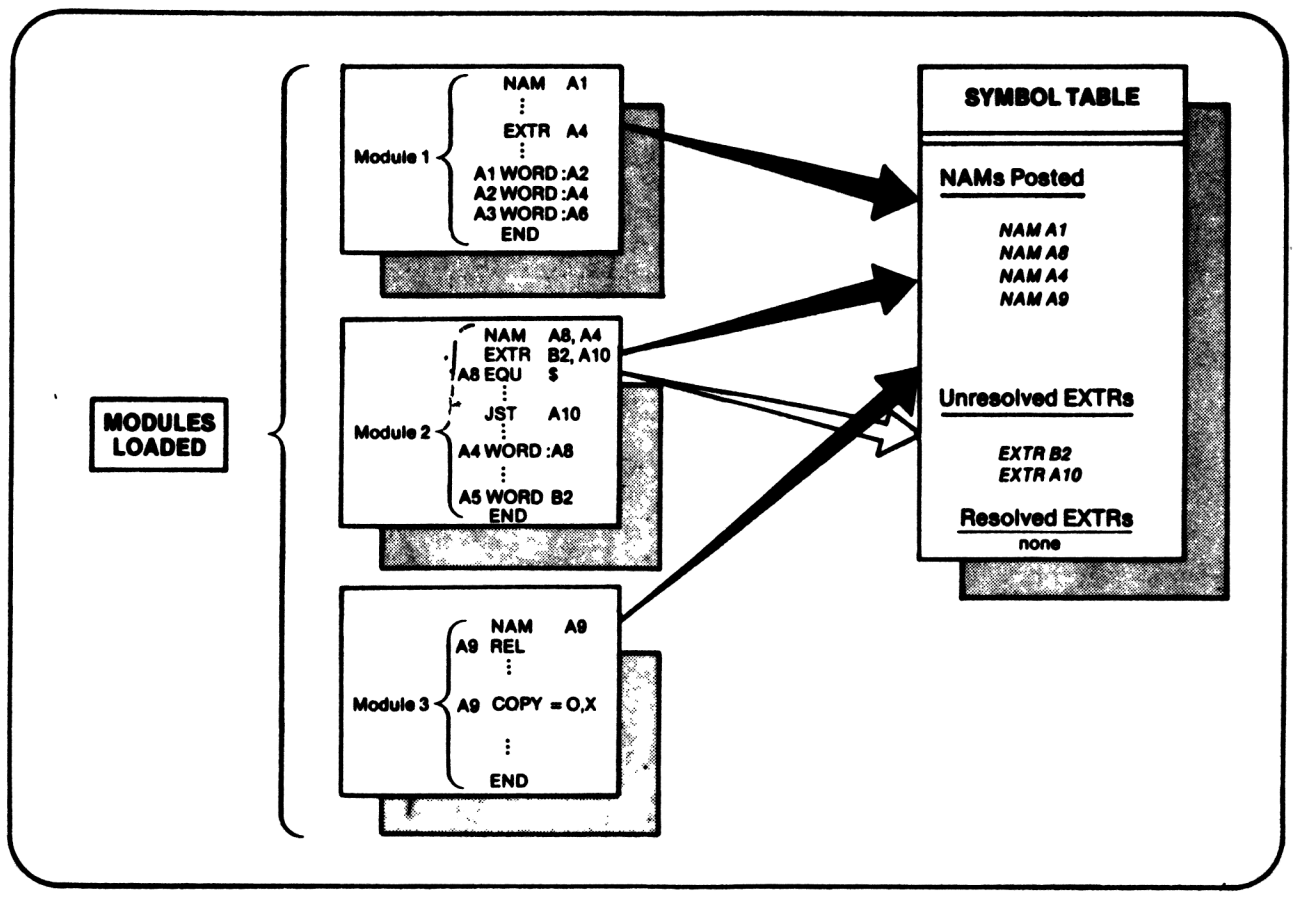


Figure 2-7. Unconditionally Loading the First File

No externals were resolved in the first three interrogations. External A4 in module one was not referenced in the assembly so the assembler did not generate object text for the statement (EXTR A4 is an address and word :A4 is a literal).

In module 3, the REL (area) name and the NAM (symbol) name are the same. Identical REL names and symbol names are accepted by the LINK and do not conflict during the LINK operation.

The LINK interrogates FILEB on a more selective basis. FILEB consists of three modules, two of which contain secondary NAMs (Figure 2-8).

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

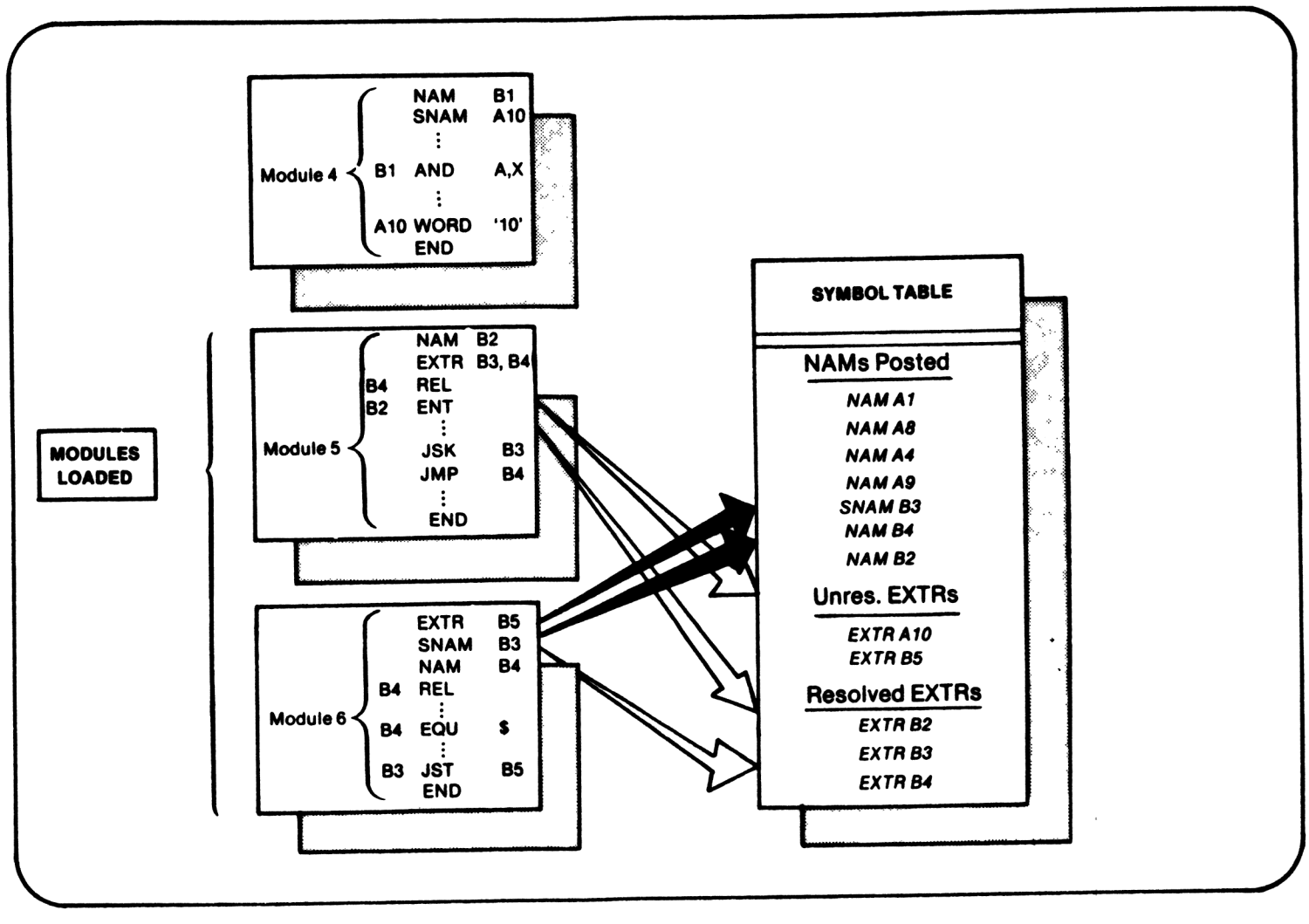


Figure 2-8. LIBRARY LOADING the Second File

The results of the LINK's interrogation of FILEB shows module 4 not loaded. The symbol table does now show a primary demand for NAM B1. Since the module is not loaded, SNAM A10 is not recognized.

The symbol table shows a primary demand for NAM B2 causing module 5 to be loaded. EXTRs B3 and B4 are posted as unresolved symbols. When the LINK processes module 6, NAM B4 satisfies a primary demand causing the module to be loaded; thus, both of the externals in module 5 are resolved. Once the module is loaded, the LINK recognizes the SNAM B3 and resolves the primary demand for that symbol. To conclude the processing of module 6, the LINK posts a primary demand for B5. When the LINK finishes processing FILEB, the symbol table shows 2 unresolved symbols, 3 resolved symbols, and 7 NAMs posted (Figure 2-8). Notice that both modules 5 and 6 contain the same REL name B4. The LINK responds to identical REL names in different modules by concatenating the segments in memory.

FILEC is the last file on this command line to be library loaded. NAM B5 in module 7 resolves the external reference for B5 and module 7 is loaded. Because the module is loaded, the secondary external A8 is resolved by NAM A8 which was previously loaded. The remaining secondary external B7 is posted as an unresolved secondary external (SEXT) in the symbol table. The unresolved SEXT entry causes the LINK to post a primary demand for the error routine SRF:ER.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

If the SEXT is resolved before this LINK process is completed, the demand for SRF:ER is dropped. Otherwise, this entry force loads the error routine, if one exists, at the end of the LINK. To complete the processing of module 7, EXTR A4 is resolved by NAM A4 which was loaded earlier in this link.

A primary demand for NAM B6, in module 8, does not exist so module 8 is not loaded. Notice that module 8 contains a secondary NAM for B7 but the secondary external in the symbol table does not force module selection so the SEXT B7 remains unresolved.

The processing of FILEC ends with 2 unresolved primary symbols, 1 unresolved secondary symbol, 8 NAMs posted and 6 resolved symbols (FIGURE 2-9).

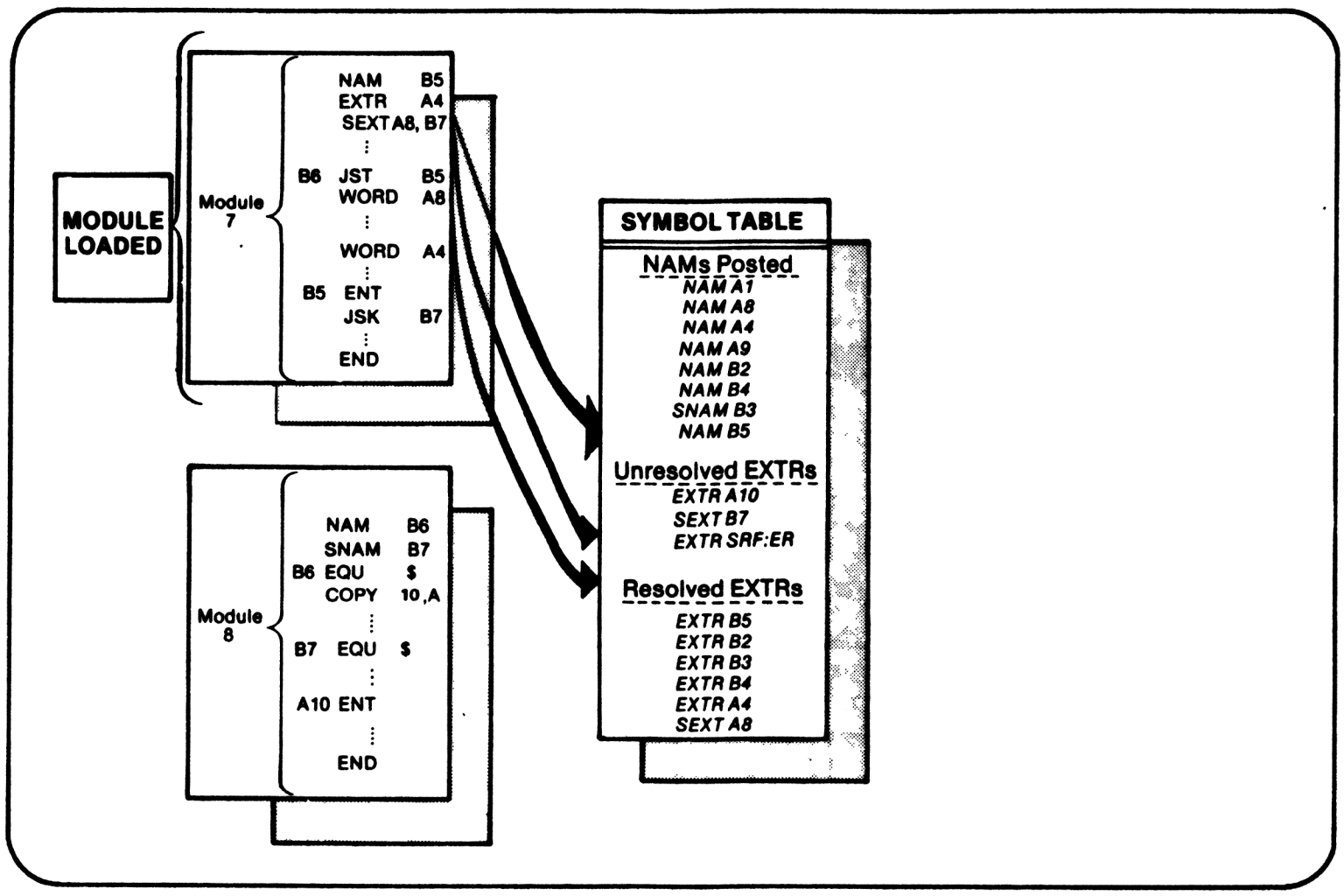


Figure 2-9. LIBRARY LOADING the Third File

File DEBUG is unconditionally loaded in response to the UN option on the command line. Loading the DEBUG module posts the NAM DEBUG4. Since no other symbols are resolved with the loading of FILE DEBUG and no other files appear on the command line, the module selection process is completed. The process ends with 2 unresolved primary externals as shown in Figure 2-10. The SRF:ER error routine was not found by the LINK to resolve the SEXT B7. The primary demands for A10 and SRF:ER remains unresolved.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

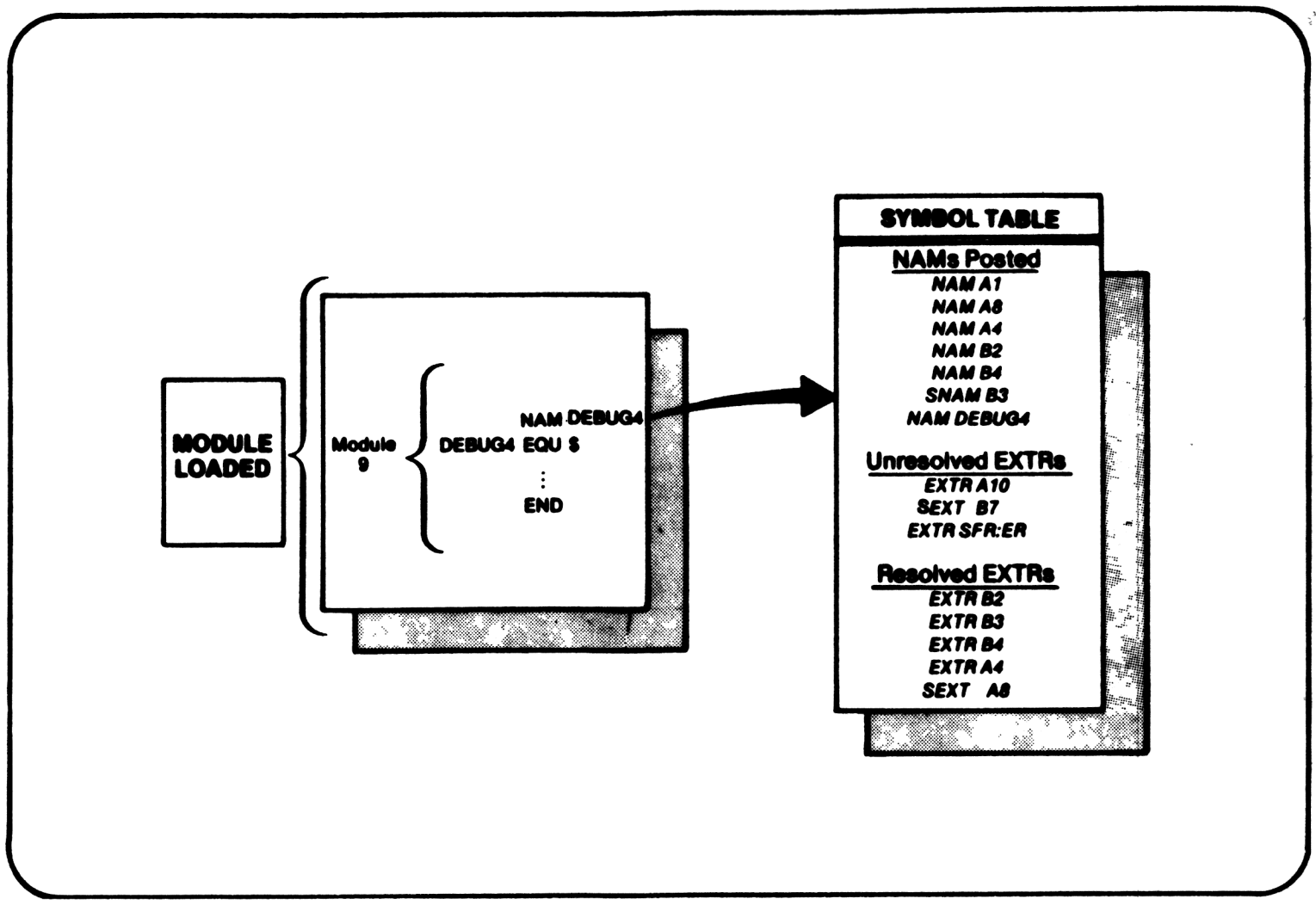


Figure 2-10. Final Result of Library Load Example

Unresolved primary symbols are listed on the last page of the LINK listing under the header "MISSING" (Figure 2-11). Correcting unresolved symbols (such as primary sumbol A10 from the previous example) can be handled in two ways.

1. The programmer can return to the source code and provide a value for the symbol. This, however, requires another run through the language translator before relinking.

2. The programmer can exercise the LINK option which allows the value of a symbol to be specified on the command line. This option is called DEFINE. When the programmer inputs a new command line for relinking, the DEFINE option is used to equate the symbol to the necessary value. Because the LINK allows the user to assign a value to any symbol from the CI unit during the LINK process, the user can also specify options to:

   a. prevent the linking of a program in a conditional link section by defining the external definition within the program before it is reached in the input, or

   b. cause the linking of a program in a conditional link section by creating an external reference which can be resolved by an external definition within the program.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE 0002                MULTI-TASKING LINKER (E7,E4,E5,E3)
1980/06/04 12:57.32                   INITIALIZE

 A10 UNDEFINED, REFERENCED IN A8 AT 0003+0003+0000(BLANK)
```

```
PAGE 0003          MULTI-TASKING LINKER (E7,E4,E5,E3,--,E2,E0)
1980/06/04 12:57:52                ALL SYMBOLS


NO TRANSFER ADDRESS SPECIFIED

 MEMORY  RELOCATABLE    0000-R:LOW      0014-R:HIGH

 SEGMENT BLANK     (0000 - 000C) RELOCATABLE - RAM
  0000-A1          0000-DEBUG4     0003-A8          0004-A4          000A-B5

 SEGMENT A9        (000D - 000D) RELOCATABLE - RAM
  000D-A9

 SEGMENT B4        (000E - 0014) RELOCATABLE - RAM
  000E-B2          0013-B3         0013-B4


MISSING: A10, SRF:ER

1 WARNING
```

Figure 2-11. Memory Map for Previous Example

2.3.2.3 Unconditional Loading

Unconditional loading forces all modules in a file to load and unconditionally link to all tasks in the LINK operation. Because unconditional loading does not entail NAM and EXTR resolution, modules in a file can be presented to the LINK in any order. Although unconditional loading eliminates a certain amount of programmer planning, it rarely provides a very efficient means of linking files.

- The program can be inefficient in memory management because all modules are loaded, needed or not. When a task is loaded at runtime, actual memory consumed can exceed the actual memory needed to perform the task.

- A program using several secondary externals (loaded unconditionally) can result in multiple-defined symbol errors since these directives are used to provide alternative routines to programs by referencing the same symbol name.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

- Unconditional load forces all NAM's contained in a file to belong to all tasks known to the LINK at the time of the Load. Several problems (too numerous to mention) occur if task directives are involved.

2.3.3 Scratch Files

Following the module loading process of the LINK, two scratch files are created. From this point on, the LINK does not consider the data to be in modules. Instead the LINK deals with segments:

Scratch File 1	containing all the shareable segments of each module.
Scratch File 2	containing all the nonshareable segments of each module.

During the selection processes, the LINK is able to keep track of these segments to generate the two scratch files at the end of selection.

The scratch files are consolidated into templates and can be saved to be generated later on the line printer by using the LINK option SAVESCRATCH. The files are named LNKSHARABLE.TMP and LNKNONSHRBL.TMP for shareable and nonshareables files respectively.

To complete the linkage processes, the LINK performs three passes through the data to assign memory locations and write the output files (Figure 2-12).

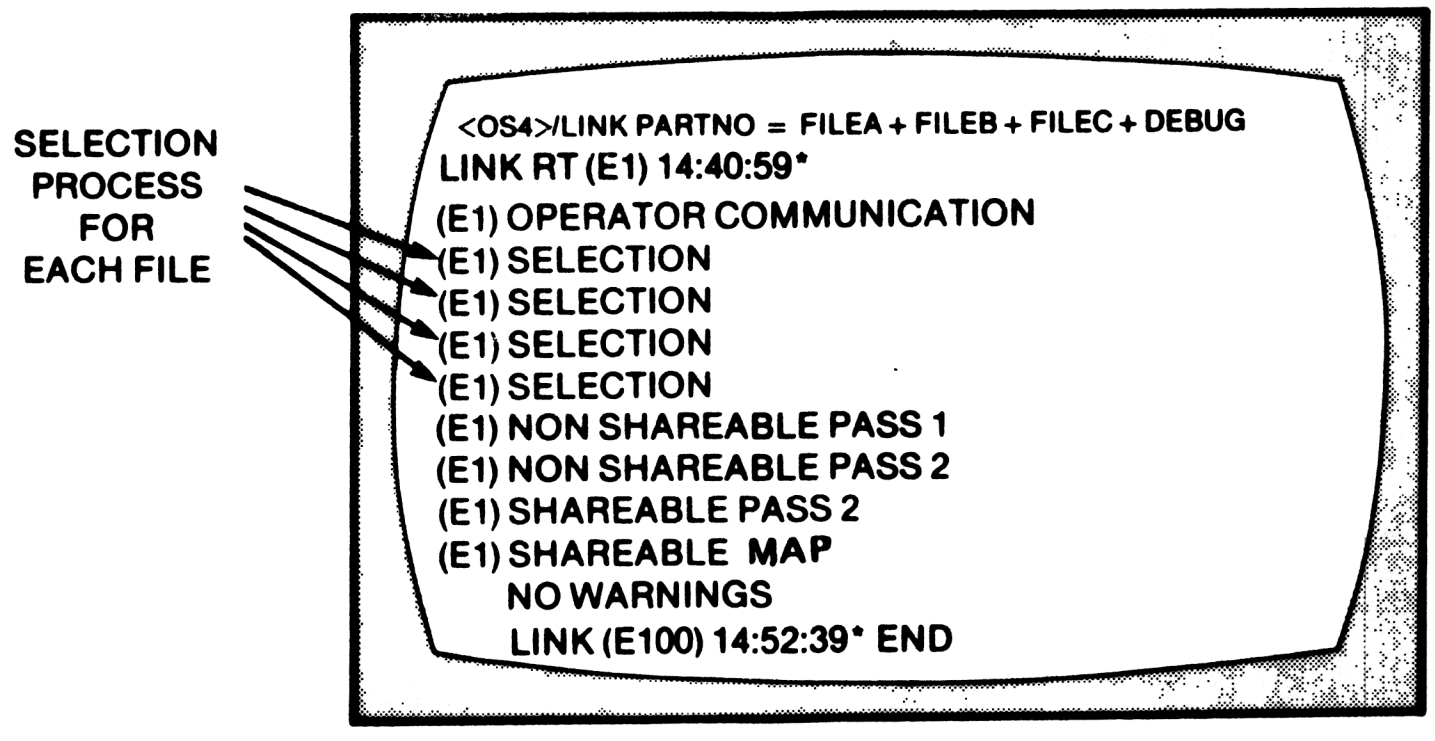


Figure 2-12. LINK Output to CRT Screen

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

The LINK performs SHAREABLE PASS 1 during the selection process to assign memory locations to all shareable symbols. Likewise, NONSHAREABLE PASS 1 assigns memory locations to all non-shareable symbols. The LINK also informs the user that Overlay locations are assigned in two passes (shown on screen when used). The program however,for the NONSHAREABLE PASS 1 is actually performing the assignments for overlays so the LINK does not really make 2 passes for overlays specifically. NONSHAREABLE PASS 2 writes all the code to the output files.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

SECTION 3

ROMABILITY OF PROGRAMS

Read-Only Memory (ROM) is a storage area which exists to enable the user to store complete programs in permanent memory. Using the LINK Editor Utility, any program can be linked romable. The following information describes the organization of data and required assembler directives to enable romability in a system.

Two other utility programs are necessary for user preparation and initiation of a romable program: ROMGEN formats the user's ROM program according to DATA I/O ROM Programmer V specifications. ROMGEN is discussed in the ROM Utilities Guide. ROMINIT is a loader which is allocated starting at the first location in ROM during linking. The loader copies the initialization data required to start the user's program from ROM to RAM. ROMINIT is discussed within this section.

3.1 ROM ENHANCEMENTS

For Romability, the LINK Editor Utility functions as follows:

- ROMMABLE, SHAREABLE segments are allocated space in the ROM.
- Templates, describing NON-SHAREABLE and NON-ROMMABLE segments for each task, are allocated to ROM.
- The LINK creates a template of SHAREABLE, NON-ROMMABLE data for all tasks, and NON-SHAREABLE, NON-ROMABLE data for the Initialization task and allocates memory in ROM for the template.
- The user can specify where ROM is located in memory.

3.1.1 Overview: A System in ROM

Memory, in romable systems, should be considered as having two parts: a READ-ONLY MEMORY area (ROM) and a Read/Write Memory area (RAM). The ROM area is described by the user by a command. The remainder of memory can accommodate RAM or remain unused. Multiple ROM areas are allowed. The program can be composed of mainly romable segments or the entire program can be packed in templates and stored in ROM. The latter, however, is not desirable since the excessive use of memory is unnecessary.

Figure 3-1 shows a general overview of building a ROM/RAM system in theory. The NAM/External resolution process occurs before the segments are packed in a template. After template expansion at runtime, the non-ROMable segments appear in the RAM area as shown in the diagram. In short, linking, templating, and referencing occurs as if ROM did not exist.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

©1980 COMPUTER AUTOMATION, INC.

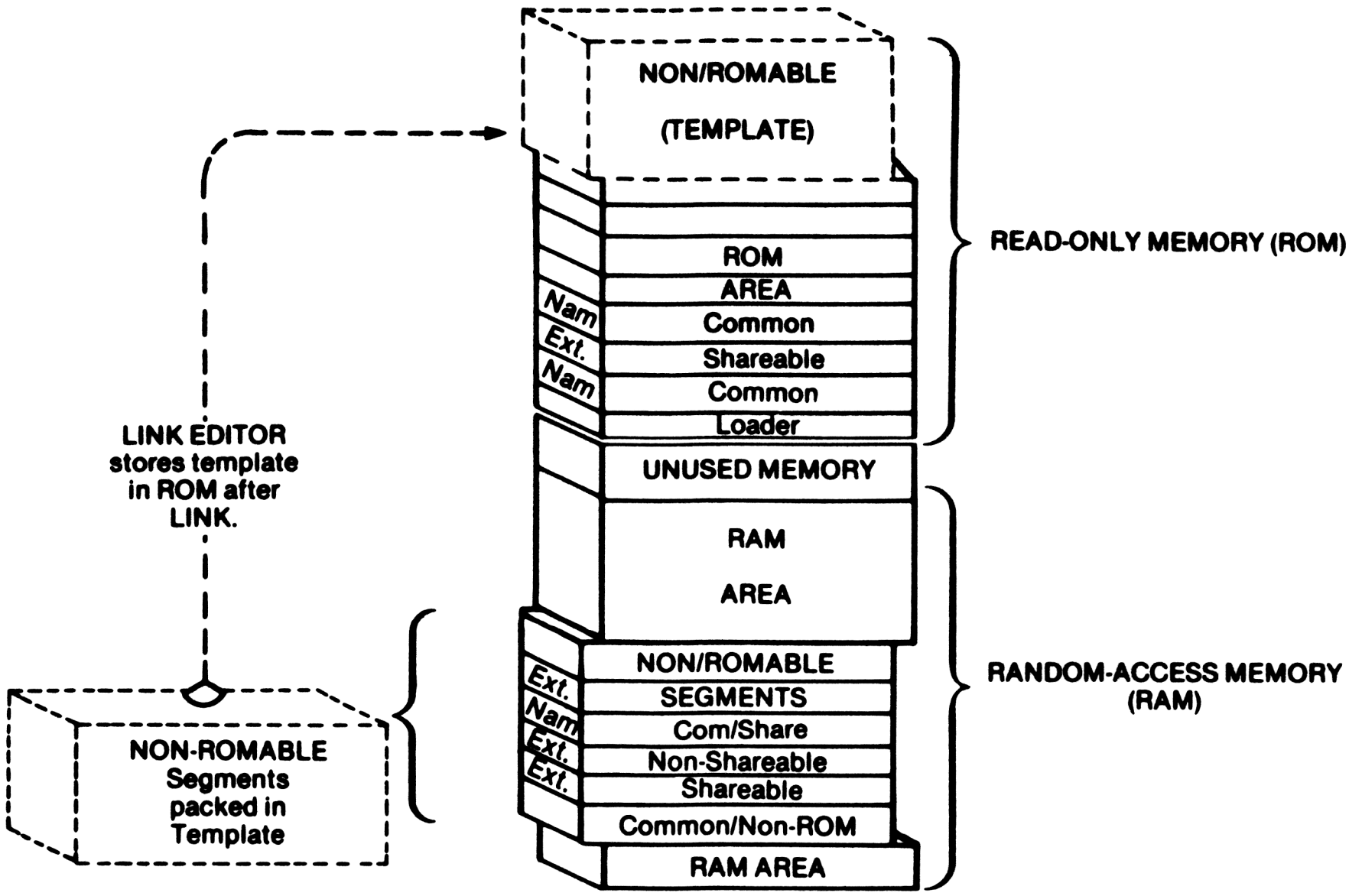


Figure 3-1. Memory for a ROMable System

The ROM part of a system is essentially subdivided into two more areas: a romable area for all shareable/romable code and a non-romable template for non-romable code. ROMINIT, the loader resident at the beginning of ROM, expands all the appropriate non-romable code for initialization. All romable code is stored following ROMINIT.

3.2 MEMORY ASSIGNMENT

In previous releases, the LINK recognized the romable attribute in segments but did not allow the programmer to give ROM a memory location. LINK allows a programmer to specify the location and amount of memory dedicated to ROM on the command line.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

©1980 COMPUTER AUTOMATION, INC.

3.2.1 ROM Location Command Option

```
                Start location          Start location
                1st ROM area            2nd ROM area

ROM = A D D R 1 - A D D R 2 [+A D D R 3 - A D D R 4]

                Ending location         Ending location
                1st ROM area            2nd ROM area
```

A programmer is not limited to one segment of ROM per system. When exercising the command option, the programmer codes a beginning and ending address for two areas; separating the two by a plus sign (+). The dash (-) between each location represents contiguous memory. ROM areas can be assigned anywhere in memory but high locations are commonly used except for interrupt locations.

The following example provides for two ROM areas in memory. The command instructs the LINK to link 3 files using the ROM locations indicated. Memory locations must be represented in hexadecimal values. The system must include ROMINIT.

```
EXAMPLE:  LINK.FILE1(ROM=7000-9000+A000-AFFF)+FILE2+FILE3+ROMINIT(UN,
          AB.ROM::=7000.)
```

ROMINIT must be unconditionally loaded since a task does not specifically reference it. Also, the loader is assigned the first memory location of the ROM area.

The example above shows ROM beginning at location :7000 and ROMINIT loaded at location :7000.

3.3 ROM SEGMENTS

A RAM template is created for all non-romable segments; the LINK assigns ROM locations directly to the romable segment. If the ROMMABLE directive is not present in a segment of code, the LINK assumes the segment to be non-romable.

3.3.1 Putting Segments in ROM

Any program can be linked romable. But the total memory required for execution (ROM+RAM) varies depending on the attributes of the segments linked. When all segments are romable and shareable, the size of the RAM template is zero. ROM, in this case, is not copied to RAM at initialization time. Less memory is required to operate the system.

If all segments are non-romable, the entire program is contained in the RAM template, the system is non-romable and all ROM is copied to RAM at initialization time. Normally, most code segments of a program are romable and some data segments (probably tables) reside in the RAM template.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

3.3.2 Format ROMMABLE Directive

```
ROMMABLE        segment[,segment]
```

The ROMMABLE directive is an assembler directive. It declares the specified named location counter to have the ROMable attribute which is passed onto the LINK. The directive can appear anywhere within an assembly. If a named location counter does not exist, it is created with an initial value of zero.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

SECTION 4

OVERLAYS

Programmers can section or divide programs, too large for memory, into logical sections or overlays. These sections remain on mass media and are called only when a task requires the function they perform. Overlays are especially useful in multi-tasking since memory resources may be limited by two or more tasks executing concurrently.

When called by a task, these sections are loaded into a fixed or predetermined area of memory called an overlay region. The LINK allocates memory for an overlay region based on the largest segment assigned to execute in that area. More than one overlay region can exist in a system with several overlays assigned to each region (Figure 4-1).

Disregarding total memory size, programmers are not restricted by overlay region size limit. However, the total number of overlays and tasks combined must be less than or equal to 126.

A program, using overlays, must be well organized. Several overlays will share the same memory region and no restrictions protect one overlay from overwriting another. Also, too many overlays can introduce a few time penalties or wait states since they are retrieved from disk or some other means of storage. Finally, a major mistake in overlaying is a failure to keep an overlay free of all working storage which is needed across overlay loads.

Ideally, the routines to be considered as overlays should be those which are used infrequently or only once such as initialization, end-of-run, or abort procedures.

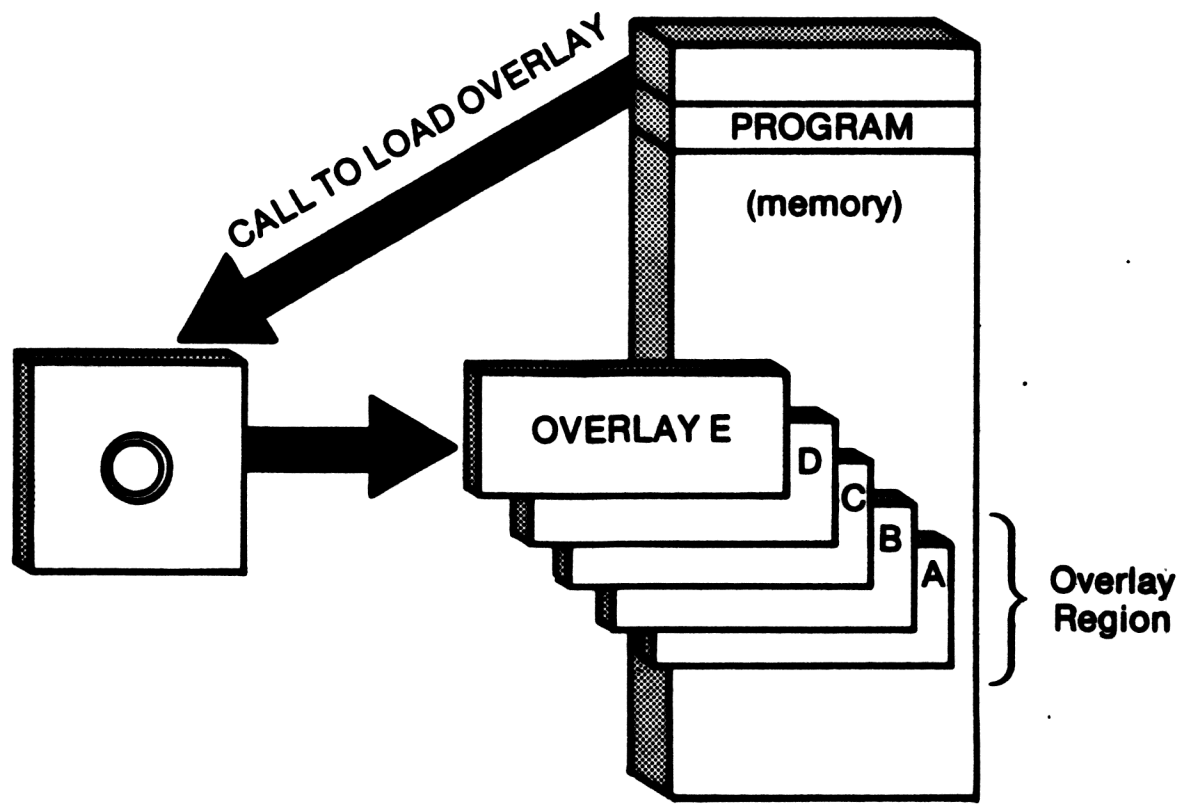


Figure 4-1. Overlays in Memory

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

4.1 LINK COMMAND WITH OVERLAY OPTION

The following LINK command is used to provide NAM and External resolution between tasks and overlays. The forced file extension for overlays is .OLY. The first file is followed by the overlay option:

Command format:

/LINK file1(OVERLAYS(SEG_1=OV_1(nam,nam1,...)+OV2(nam3),SEG2=OV3(nam4,nam5,nam6)))
+file2+file3(other options).

where: OVERLAYS = option

SEG1
SEG2 = overlay regions in memory

OV1
OV2 = overlay names (disk files)
OV3

nam
nam1
nam2
nam3 = overlay NAMs (specified in program modules)
nam4
nam5
nam6

Any symbol in an overlay, specified as a NAM, is considered an overlay entry point and accessible by all tasks and overlays in the system.

The names of the overlay regions must be unique to other segment names in the system. A programmer can indirectly refer to the overlay region by putting a label in a zero length declaration of the overlay region:

CODE		COMMENT
SEG1	REL	
GORPH	EQU $	GORPH points to the first word of the overlay region.
ZORDA	REL	

4.2 CODING AN OVERLAY

Since the LINK acknowledges an overlay option on the command line to distinguish between tasks and overlays, an overlay is coded the same as a task. Figure 4-2 shows an example of an overlay listing.

```
 OVERLAY SEGMENT FORTRAN  (0200 - 025B) ABSOLUTE - RAM
  0204-OVRLY1

 ROOT SEGMENT FOVRG2    (025C - 0299) ABSOLUTE - RAM, OVERLAY REGION
 ROOT SEGMENT F:LI3R    (0202 - 16C2) ABSOLUTE - ROM, SHAREABLE

MISSING: I:OVL
```

Figure 4-2. Example Overlay Listing

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

4.3 LINKING AN OVERLAY

In Computer Automation, Inc., software overlays resemble tasks. The difference between a task and overlay is not apparent to the LINK until the linking of NAMs and Externals is complete. Thus it is possible for NAMs in overlays to reference other data regions and other data regions to reference NAMs in overlays. This allows intercommunication between overlays and other segments to exist.

4.4 LOADING AN OVERLAY

The following information provides a brief description of the optional utility and services used to load an overlay. For additional information reference the respective manuals as indicated.

4.5 OVMERGE UTILITY

The OVMERGE utility program provides a mechanism for combining binary files output by the LINK program into absolute core image overlays via the I:OVL IOS4 service. In the normal course of events, the programmer uses the OVERLAY option of the LINK to produce overlay modules and combines these files with OVMERGE. However, OVMERGE can process any file which has been processed by the LINK, treating relocatable modules as absolute (with relocation zero). The OS4 System User's Manual, revision E0 provides information concerning the use of the OVMERGE utility.

4.5.1 Overlay Loading Services

IOS4 provides two services for loading program overlays from peripheral devices into memory: the I:OVL and I:LOAD services.

I:LOAD

- The I:LOAD service for overlay loading operates as described in Section 4.2.1 of the Input/Output Subsystem (IOS4) Reference Manual. The service loads either absolute or relocatable overlays. On completion of an I:LOAD service, the specified logical unit was opened, used for input, and closed. The process of loading using the I:LOAD causes IOS4 to interpret object code as it loads.

I:OVL

- The I:OVL service differs from the I:LOAD in several ways. I:OVL only loads absolute core images previously generated from object files via the OVMERGE OS4 utility. I:OVL does not process type codes and is, therefore, considerably faster than I:LOAD.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

On completion of the first call to I:OVL, the logical unit used for input remains open and must not be used for any other purpose than further I:OVL service requests. If a logical unit used by I:OVL is required for another I/O operation, the user may request that I:OVL close the logical unit. In the case of I:LOAD, overlays are contained in different files and additional files must be opened and closed in loading. For I:OVL, a single file contains many overlays and it is unlikely that a user would need to close the logical unit before the program terminates.

The I:OVL service contains a 'directory' of overlays each being accessed by overlay number. The service is an RTX4 call to IOS4 and requires an overlay number and parameter block address as its arguments. The parameter block itself can be generated by the OVL: macro, which must be labeled with the argument from the I:OVL request. Additional information on the I:OVL service and the parameter block generated by the OVL: macro are provided in the Input/ Output Subsystem IOS4 Reference Manual.

4.6 CLOSING AN OVERLAY FILE

An overlay file can be closed by an I:OVL service requesting overlay 0 (zero) to load. After an overlay file is closed, the logical unit and directory block can be used for other purposes.

4.7 ERROR CHECKING

No allowances are made for error checking when dealing with overlays. If a programmer inadvertently references a symbol in an overlay, not residing in memory at the time of the call, an error is not detected. It is the responsibility of the programmer to insure that an overlay resides in memory prior to executing a task requiring an overlay function.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

SECTION 5

PROGRAM EXAMPLE

Part 1 of this program example uses the multi-tasking approach to programming. Multi-tasking offers a certain simplicity to program design. To clarify terminology, a task in a multi-tasking program task is a unit of work that consists of a collection of machine instructions and performs a logical function. An activity is a specific instance of performing that task.

In multi-tasking, several activities for a given task can allow the various stages of transaction processing to be easily interleaved by creating a separate activity for each transaction. In the same respect, each task can be designed to perform a specific function with transactions flowing through each task via queues.

To maintain control over these and other asynchronous processes, it is usually necessary to make use of various RTX4 services, as described in the Real-Time Exectutive RTX4 User's Manual. These services include:

- activity creation and destruction
- clock and timer services
- mailboxes for inter-activity communication
- resource sharing and interlocks
- system initialization

Additionally, a programmer can specify certain items at compile-time to communicate with the LINK. These items concern:

- task identification (TASK directive)
- code and data module sharing or private use (REL, COMMON, SHARABLE directives)
- ROMability of segments (ROMMABLE directive)

The LINK uses this information to build its 'image' of memory. When the LINK finishes, a MEMORY MAP is generated to the LO unit which is illustrated as Part 2 of this program example. (Additional information on Multi-tasking directives can be found in the NAKED MINI® 4 Assembler User's Manual.)

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

5.1 MULTI-TASKING PROGRAM

This program is an example application involving FORTRAN tasks and assembly language tasks which use dynamic initialized data regions. The example consists of four parts:

1. An Assembly language module which defines the RTX4 initialization block (INIT:A), unit assignment table (UAT:AA), environment control block (ECB:A), and the global common definition for the labeled common GLOBAL.

2. Initialization task (ITASK)--This FORTRAN task is initiated by the INIT:A block. ITASK begins all tasks required for the example to run, then ends itself.

3. FORTRAN print task (FTASK)--This task computes the print line and outputs to the LO unit.

4. Assembly language print task (ATASK)--This task performs the same logical function as FTASK.

The FORTRAN Initialization task (ITASK) is the first to run. This task begins FTASK four times and ATASK four times. Each execution instance of the task is passed an identifier (ID) and a tick count (ITICKS). ITASK then ends itself (STOP). FTASK (written in FORTRAN) and ATASK (written in assembly language) are identical in function. Each one waits the specified number of clock ticks, gets the current time of day, and prints a summary message.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

ASSEMBLY LANGUAGE LISTING

```
PAGE  0001  MACRO (D1) INIT:A, UAT:A AND ECB:A DEFINITIONS
1979/12/14 10:44:00.25 TABLES.ASM

      03F4        0003              NAM       UAT              UNIT ASSIGNMENT TABLE
      0404        0004              NAM       ECB              ENVIRONMENT CONTROL BLOCK
      0000        0005              NAM       GLOBAL           GLOBAL COMMON REGION
                  0006              EXTR      ITASK            INITIAL FORTRAN TASK NAME

                  0008              LOAD      XDBUG4

      000003E8    0010  PRIORITY    EQU       1000             INITIAL PRIORITY
      000003E8    0011  FREEPOOL    EQU       1000             FREEPOOL SIZE
0000  0000        0012              INIT:A    0,0,0,0,ECB,ITASK,PRIORITY,FREEPOOL
0001  0000        0012+
0002  0000        0012+
0003  0000        0012+
0004  0404        0012+
0005  0005        0012+
0006  03E8        0012+
0007  F00E        0012+
0008  03E8        0012+
0009  000B        0012+
000A  0000        0012+
000B  8080        0012+
                  0013  * UNIT ASSIGNMENT TABLE
03F3  F09E        0014              UAT:AA    UAT
03F4  0006        0014+
03F5  0000        0014+
03F8  5456        0015              UAT:EE    'TV',D:TV00      UNIT 1
03F9  0001        0015+
03FA  3032        0016              UAT:EE    '02',D:TV00      UNIT 2
03FB  0001        0016+
03FC  3033        0017              UAT:EE    '03',D:TV00      UNIT 3
03FD  0001        0017+
03FE  3034        0018              UAT:EE    '04',D:TV00      UNIT 4
03FF  0001        0018+
0400  3035        0019              UAT:EE    '05',D:TV00      UNIT 5
0401  0001        0019+
0402  4C50        0020              UAT:EE    'LP',D:LP00      UNIT 6
0403  0000        0020+
      0000  0001  0021              UAT:ZZ


                  0023  * ENVIRONMENT CONTROL BLOCK
0404              0024              ECB:A     ECB,UAT
0405  0000        0024+
0406  0404        0024+
0407  0001        0024+
0408  0000        0024+
040A  0000        0024+
040B  0000        0024+
040C  0000        0024+
040D  0000        0024+
040E  0000        0024+
```

This macro creates the freepool and sets priorities.

For an RTX link, the first file is one module consisting of the INIT:A macro, the ECB:A macro and the UAT (unit assignment table).

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

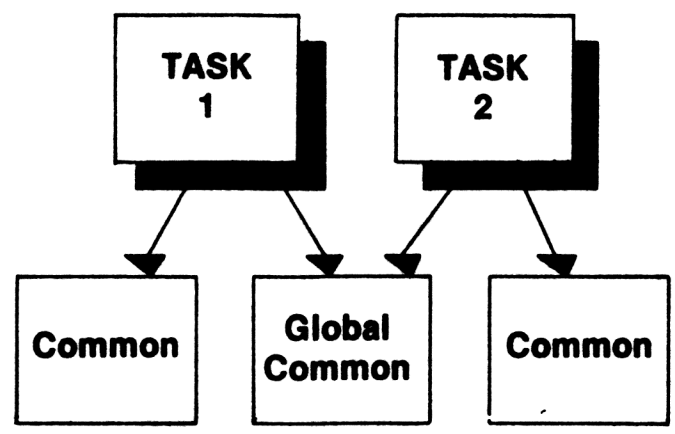
© 1980 COMPUTER AUTOMATION, INC.

```
PAGE  0002  MACRO (D1) INIT:A, UAT:A AND ECB:A DEFINITIONS
1979/12/14 10:44:02.25 TABLES.ASM

040F F06E          0024+
0410 0010          0024+
0411 0030          0024+
0412 0000          0024+
0414 0000          0024+
0424 03F4          0024+
0425 0000          0024+
0426 0000          0024+
0427 0000          0024+
0428 0000          0024+
0429 7FFF          0024+
042A               0024+
042B               0024+
042C               0024+
042D 0000          0024+
042E 0000          0024+
042F 0000          0024+
0430 0000          0024+
0431 0000          0024+
0432 0000          0024+
0433 0000          0024+
0434 0000          0024+
                   0025   * GLOBAL COMMON DEFINITION
     0000 0003     0026   GLOBAL   REL       0
                   0027              SHARABLE GLOBAL
                   0028              COMMON   GLOBAL
     0000          0029              NAM      GLOBAL
0000               0030   GLOBAL     RES      1
                   0031              END

0000  ERRORS     (0000)
0000  WARNINGS   (0000)
```

Declares segment 'Global' of type common. The segment is defined as shareable.



Each task has its own common area but the Global area can be shared by all.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE    1 1979/12/14 10:44:14.00   C  ITASK -- FORTRAN INITIALIZATION TASK
FORTRAN(D2)      FTASK      .FTN

0001 C  ITASK -- FORTRAN INITIALIZATION TASK
0002 C  THIS TASK BEGINS ATASK AND FTASK 4 TIMES.
0003 C  EACH TIME THE TASK ID IS INCREMENTED AS
0004 C  IS IS PASSED TO THE TASK.   THE DELAY
0005 C  INTERVAL IS COMPUTED FROM THE ID AND IS PASSED
0006 C  TO THE TASK THROUGH ITICKS.
0007 C
0008          TASK ITASK(A,Q,X,1,:100)
0009          EXTERNAL ATASK
0010          EXTERNAL FTASK
0011          DO 10 ID=1,4
0012          ITICKS=ID*120
0013          CALL G:BGIN(FTASK,400,ID,ITICKS,0,IEX)
0014          ITICKS=ITICKS+60
0015   10     CALL G:BGIN(ATASK,200,ID,ITICKS,0,IEX)
0016          STOP
0017          END
```


® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE    4 1979/12/14 10:44:26.50    C  FTASK -- FORTRAN PRINT TASK
FORTRAN(D2)       FTASK       .FTN

0018 C  FTASK -- FORTRAN PRINT TASK
0019 C  THIS TASK RECEIVES AN IDENTIFICATION (ID)
0020 C  AND THE NUMBER OF TICKS TO WAIT (TICKS) WHEN
0021 C  IT IS STARTED.  FTASK STARTS A TICK CLOCK, WAITS
0022 C  FOR IT TO TIME OUT, GETS THE CURRENT ASCII TIME
0023 C  OF DAY AND PRINTS THE RESULTS.
0024 C
0025          TASK FTASK(ID,TICKS,X,4,:100)
0026          INTEGER SDBS(4,4),TIME(7)
0027          COMMON /GLOBAL/ SDBS
0028          CALL G:SDB(SDBS(3,ID),0,IEX)
0029   10     CALL G:ITIC(SDBS(3,ID),SDBS(3,ID),TICKS,IEX)
0030          CALL G:WAIT(SDBS(3,ID),IEX)
0031          CALL G:GATD(TIME,IEX)
0032          WRITE(6,100) ID,TIME
0033          GOTO 10
0034  100     FORMAT(' FORTRAN TASK ',I2,' AT ',7A2)
0035          END
```


● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE  0001  MACRO (D1) ATASK -- ASSEMBLY LANGUAGE PRINT TASK
1979/12/14 10:45:38.00 ATASK.ASM

                    0003  *  ATASK -- ASSEMBLY LANGUAGE PRINT TASK.
                    0004  *  THIS TASK RECEIVES AN IDENTIFICATION (A REGISTER)
                    0005  *  AND THE NUMBER OF TICKS TO WAIT (Q REGISTER) WHEN
                    0006  *  IT IS STARTED.  ATASK STARTS A TICK CLOCK, WAITS
                    0007  *  FOR IT TO TIME OUT, GETS THE CURRENT ASCII TIME
                    0008  *  OF DAY AND PRINTS THE RESULTS.
                    0009  *
      00000100      0010  STACK     EQU       :100            STACK SIZE
      00000004      0011  USAGE     EQU       4               NUMBER OF CONCURRENT EXEC

0000                0013            TDB:A     ATASK,START,*,0,STACK,0,USAGE
0001 8000           0013+
0002 0008           0013+
0003 0004           0013+
0004                0013+
0005 0000           0013+
0006 0100           0013+
0007 0000           0013+
0008 0004           0013+
0009 0000           0013+
000B F01E           0013+
                    0014  * SHARABLE PROGRAM SEGMENT
      0000 0003     0015  ATASK     REL
                    0016            SHARABLE ATASK


                    0018  * ATASK ENTRY POINT
      0000          0019            NAM       START
      00000000      0020  START     EQU       $
0000 2000           0021            COPY      A,X             A=TASK ID (1 TO 4)
0001 AA8F 0011      0022            ADD       =SDBTABLE-1,X   INDEX INTO SDB TABLE
0002 A080 0000      0023            COPY      0(X),X          X=ADDRESS OF SDB
0003 A458 0018      0024            COPY      X,SEMA4-YREG(Y) STORE IN TICK:A BLOCK
0004 8A8D 0012      0025            ADD       =' 0',A         CONVERT TASK ID TO ASCII
0005 844C 000C      0026            COPY      A,ID-YREG(Y)    STORE IN MESSAGE BUFFER
0006 C459 0019      0027            COPY      Q,TICKS-YREG(Y) SET TIME IN TICK:A BLOC
      00000007      0028  LOOP      EQU       $
0007 1A0B           0029            R:ITIC    TIMER-YREG(Y)   START THE TIMER
0008 1016 0016      0029+
0009 1A02           0030            R:WAIT    0(X)            WAIT FOR TIMER TO EXPIRE
000A 2000 0000      0030+
000B 1A1B           0031            R:GATD    TIME-YREG(Y)    GET CURRENT DATE/TIME
000C 100F 000F      0031+
000D 1A07           0032            I:IO      OUTPUT-YREG(Y)  OUTPUT MESSAGE
000E 101A 001A      0032+
000F 0E0D           0033            HLT                       HALT IF ABNORMAL RETURN
0010 9E76 0007      0034            JMP       LOOP            GO DO IT AGAIN
      0008          0035            LPOOL
0011 0018
0012 2030
0013
```

Macro equivalent to TASK statement in Fortran.

Sharable declarative defines program segment as sharable.

This points Y to a template expansion. Since the location of a template expansion is not known at compile or Link time, a template expansion can only be referenced relative to the beginning.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE  0003  MACRO (D1) ATASK -- ASSEMBLY LANGUAGE PRINT TASK
1979/12/14 10:45:41.25 SEMAPHORE DEFINITION BLOCKS

                    0037  * TABLE OF POINTERS TO SEMAPHORE DESCRIPTOR
                    0038  * BLOCKS (SDB).
     00000019       0039  SDBTABLE EQU      $
0019 000E           0040           WORD     SDB1
001A 0012           0041           WORD     SDB2
001B 0016           0042           WORD     SDB3
001C 001A           0043           WORD     SDB4
                    0044  * SEMAPHORE DESCRIPTOR BLOCK DEFINITIONS
000C                0045           SDB:A    SDB1,0
000D 0000           0045+
000E 0000           0045+
000F F03E           0045+
0010                0046           SDB:A    SDB2,0
0011 0000           0046+
0012 0000           0046+
0013 F03E           0046+
0014                0047           SDB:A    SDB3,0
0015 0000           0047+
0016 0000           0047+
0017 F03E           0047+
0018                0048           SDB:A    SDB4,0
0019 0000           0048+
001A 0000           0048+
001B F03E           0048+
```

The semaphore descriptor blocks (SDB's) are pulled into a different REL area. Therefore, SDB's are referenced by pointers. All tables are built in TABLE:REL area.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE  0004  MACRO (D1) ATASK -- ASSEMBLY LANGUAGE PRINT TASK
1979/12/14 10:45:43.25 NON-SHARABLE DATA SEGMENT

                    0050  * NON-SHARABLE DATA SEGMENT
      0000 0004     0051  ATASK:    REL
           0000     0052              INITY     ATASK,YREG     DECLARE INITIAL Y REGISTE
      00000000      0053  YREG       EQU       $

                    0055  * OUTPUT MESSAGE BUFFER
0000 2041           0056  MESSAGE    BYTE      ' ASSEMBLY LANGUAGE TASK '
0001 5353
0002 454D
0003 424C
0004 5920
0005 4C41
0006 4E47
0007 5541
0008 4745
0009 2054
000A 4153
000B 4B20
000C 0000           0057  ID         WORD      0              TASK IDENTIFICATION (1-4)
000D 2041           0058              BYTE      ' AT '
000E 5420
000F 0000           0059  TIME       RES       7,0            ASCII DATE AND TIME BUFFE
     0000002C       0060  LENGTH     EQU       $-MESSAGE*2    MESSAGE LENGTH IN BYTES


                    0062  * R:ITIC ARGUMENT LIST
0016 0000           0063              TICK:A    TIMER,0,0,0
0017 0000           0063+
0018 0000           0063+
0019 0000           0063+
     00000018       0064  SEMA4      EQU       TIMER+2        SEMAPHORE TO SIGNAL
     00000019       0065  TICKS      EQU       TIMER+3        NUMBER OF TICKS TO DELAY


                    0067  * I/O CONTROL BLOCK
001A 4C50           0068              IOB:A     OUTPUT,'LP',WR:,FA:,LENGTH,MESSAGE,0
001B 0000           0068+
001C 0000           0068+
001D 001B           0068+
001E 002C           0068+
001F 0000           0068+
0020 0000           0068+
0021 0000           0068+
                    0069              END

0000  ERRORS     (0000)
0000  WARNINGS   (0000)
```

Lack of sharable directive indicates data is non-sharable.

Label where Y will point.

Non-shareable data is compressed into a template and Y points to the template.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

5.2 THE MEMORY MAP

Unless explicitly instructed otherwise, LINK generates a memory map to the LO unit during the link-edit process. The memory map is a summary of various link-edit process parameters and memory usage. The memory map of a multi-tasking program is actually composed of several individual maps: a map of all the symbols in each task, a map of all symbols in the initialization program, and a map of all symbols in the entire program. A memory map of a non-multitasking program lists all symbols in the program.

The first page of the sample map, entitled "File Usage Summary," breaks the LINK command line into its component parts. The first line lists the output file names, each followed by the names of any options specified for it. All unit and option names are given in their complete, unabbreviated forms.

Individual maps of the tasks follow. These maps indicate the following items for each task:

- the location of the task's template and its memory limits
- the names and locations of the segments that comprise the template
- the absolute symbols defined for the entire task
- the relocatable symbols in the task (separated by the names of the segments in which each is defined)
- the limits of each segment
- the locations of the heads of chains used in the task (if any) (followed by the names of the segments in which they appear)
- a list of all unresolved primary externals found in the task (if any) under the heading "MISSING"

The "INITIALIZE" map follows the individual maps. This map uses the same format as that of the individual maps for each task. However, the map only contains the symbols used in the initialization task.

The remaining pages, entitled "ALL SYMBOLS," lists all symbols in the program, using the same format as that of the individual maps for each task.

If the appropriate option is specified, a concordance listing (not provided in this example) can be generated comprised of all program symbols. The symbols are alphabetized and the listing shows:

- the location of each symbol's definition
- the name of the segment in which each symbol is defined
- the names of the tasks which reference the symbol

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Throughout the link-edit process, LINK keeps a log of error conditions it detects, interspersing error messages with the listing lines. It also lists some NOTE records encountered in the object files. These NOTEs are used in CA software (RTX4, etc.) to identify revision levels of the software modules containing them.

The LINK command line used to link the previous program example which produced the following LINK Map is as follows:

```
/LINK EXAMPLE=TABLES(AB=100,PA=54*80)+
      FTASK+ATASK+SF.FORTRAN(SYS)+
      SF.SFM(SYS)+SF.IOS(SYS)+
      SF.RTX(SYS)
```


● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

LINK MEMORY MAP

```
PAGE 0001                                          MULTI-TASKING LINKER (D9)
1979/12/14 10:45:54                                     FILE USAGE SUMMARY

DM1.EXAMPLE.BIN=DM1.TABLES.OBJ(ABSOLUTE=100,PAGESIZE=54*80)+
DM1.FTASK.OBJ+
DM1.ATASK.OBJ+
DM.FORTRAN.LIB(SYSTEMFILE)+
     NOTE: ***  FORTRAN/RTX INTERFACE - REV D1  ***
 NON-SHAREABLE DATA NOT ALLOWED IN -SYSTEM- MODULES IN F:XRDA AT
05A9+0E50+0000(F:LIBR)
     NOTE: ***  FORTRAN.LIB  REV D2 ***
 NON-SHAREABLE DATA NOT ALLOWED IN -SYSTEM- MODULES IN F:XFDA AT
011B+0000+0000(F:LIBR::)
 NON-SHAREABLE DATA NOT ALLOWED IN -SYSTEM- MODULES IN F:XFUT AT
0095+0000+0000(F:LIBR::)
DM.SFM.LIB(SYSTEMFILE)+
DM.IOS.LIB(SYSTEMFILE)+
     NOTE: ***  IOS.LIB - REV D1  ***
DM.RTX.LIB(SYSTEMFILE)
     NOTE: ***  RTX.LIB - REV D1  ***
     NOTE: ***  DEBUGX.OBJ - REV D0  ***
```

Must have SFM.LIB for RTX Link.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE 0002                MULTI-TASKING LINKER (D9, D7, D7, D7, D7)
1979/12/14 10:50:28                  TASK. . . ATASK

TEMPLATE ATASK     (3C33 - 3C60) ABSOLUTE - RAM
[LIMITS] RELOCATABLE    0000-N:LOW      0021-N:HIGH

 SEGMENT ATASK:    (0000 - 0021) RELOCATABLE - RAM
 SEGMENT TABLE:    (0100 - 0568) ABSOLUTE - RAM, SHAREABLE
  054D-ATASK      055B-SDB1       055F-SDB2       0563-SDB3       0567-SDB4

 SEGMENT ATASK     (0612 - 062E) ABSOLUTE - RAM, SHAREABLE
  0612-START
 SEGMENT IOS:      (22FB - 2A67) ABSOLUTE - RAM, SHAREABLE
 SEGMENT RTX:      (2A68 - 31B3) ABSOLUTE - RAM, SHAREABLE
 SEGMENT DUMMY:    (3C1B - 3C32) ABSOLUTE - RAM, SHAREABLE
```

Address of Template.

Size of Expanded Template.

SDB's

Execution Address

RTX Segments.

Shareable Program region

TDB

Non-Shareable Program region

In the Link map, there is a separate list of symbols for each task.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE 0003                MULTI-TASKING LINKER (D9, D7, D7, D7, D7)
1979/12/14 10:51:10                  TASK. . . FTASK

TEMPLATE FTASK     (3C61 - 3CA7) ABSOLUTE - RAM
[LIMITS] RELOCATABLE   0000-N:LOW      02E4-N:HIGH
 SEGMENT FTASK:::  (0000 - 001C) RELOCATABLE - RAM
 SEGMENT F:LIBR::  (001D - 02E4) RELOCATABLE - RAM
  001E-F:XRDA     0141-F:XFDS     0143-F:XFDM     014B-F:XFCI     01CD-F:XFUT
  0138-F:XFDA     0142-F:XFDE     0145-F:XFCR

ABSOLUTE SYMBOLS
  0072-R:MPM1     0073-R:MPM2     FFF6-F:XNFU

 SEGMENT TABLE:    (0100 - 0568) ABSOLUTE - RAM, SHAREABLE
  0541-FTASK
 SEGMENT GLOBAL    (0569 - 0578) ABSOLUTE - RAM, GLOBAL COMMON
 SEGMENT FORTRAN   (0579 - 0611) ABSOLUTE - ROM, SHAREABLE
  05C2-FTASK:
 SEGMENT G:LIBR    (062F - 06ED) ABSOLUTE - ROM, SHAREABLE
  0650-G:WAIT     0670-G:TIC      068D-G:SDB      06C1-G:CONTS    06DA-G:ENTER
  0661-G:ITIC     067E-G:GATD     06B4-G:ADDR     06CF-G:CHECK    06EA-G:EXIT

 SEGMENT F:LIBR    (06EE - 1AF1) ABSOLUTE - ROM, SHAREABLE
  06EE-F:RREF     0796-F:R4OL     0BD3-F:RFG      10FA-F:XGNP     15F7-F:XSBW
  06FD-F:RWEF     079A-F:RROL     0C25-F:RFIR     1112-F:XLYA     160F-F:XSBR
  0732-F:RDLW     079E-F:RDOL     0C4E-F:RFF      1122-F:XTRM     1624-F:XBTR
  0733-F:RRIF     07A2-F:RCOL     0C67-F:RFE      1165-F:XGIU     164B-F:XEBT
  073E-F:RWIF     07A4-F:RLOL     0C70-F:RFD      1169-F:XGOU     1671-F:XOPN
  0751-F:RETF     07AD-F:RHUS     1006-F:RFTS     1184-F:XECT     1745-F:XCLS
  0766-F:RIUS     096D-F:RFRW     1010-F:RFAD     147F-F:ERRS     178D-F:XEND
  076A-F:R4US     0985-F:IOER     1023-F:RFDA     1480-F:XSNE     17B3-F:XBCK
  076E-F:RAUS     0987-F:RFFD     103E-F:RFFQ     1486-F:ERRC     1812-F:XREW
  0773-F:RRUS     09B6-F:RFWD     1049-F:RFWB     1487-F:ERCS     1832-F:XSRN
  0777-F:RDUS     09BB-F:RFSW     1063-F:RPAU     153C-F:XSTP     184D-F:XENQ
  077B-F:RLUS     09C5-F:RFSO     106E-F:RSTO     153E-F:XCAF     1863-F:XPSE
  077F-F:RCUS     0ACA-F:RFI      10A3-F:X1PP     1555-F:XGLS     1884-F:XINT
  0792-F:RAOL     0B2B-F:RFL      10C9-F:XMPP     1584-F:XASW     1AE7-F:RSTU
  0792-F:RIOL     0B57-F:RFZ      10ED-F:XVPP     15B2-F:XASR     1AF0-F:XSTF

SEGMENT OS4:       (1AF2 - 1C6F) ABSOLUTE - RAM, SHAREABLE
  1AF2-E:GFN      1C06-ICH:E      1C55-CIBP:      1C57-E:SF       1C5D-UF:E
  1AF4-GFN:E      1C0E-ICH:A      1C56-CIAC:      1C58-E:UF       1C6C-E:SLU
  1BAE-ISM:E      1C41-SBP:       1C57-CIBUF:     1C5B-SF:E       1C6E-E:MON
```

Size of Expanded Template

Non-Shareable Segment

TDB

Execution Address

Program will execute at first executable address; not the first memory location.

Shareable Libraries

SHAREABLE SEGMENT

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE 0004              MULTI-TASKING LINKER (D9, D7, D7, D7, D7)
1979/12/14 10:51:39                  TASK. . . FTASK

 SEGMENT SFM:      (1C70 - 22FA) ABSOLUTE - RAM, SHAREABLE
  1DB7-F:DMOUNT  1F83-F:SFC      208D-F:GDIB     21EE-F:GBS      22B3-F:RDS
  1E37-F:MOUNT   1F8E-F:SLU      2091-F:GLUS     21FD-F:RBW      22B5-F:WRS
  1E8F-F:CKMT    1F90-F:SND      2096-F:GTAU     2205-F:RBF      22E4-F:CBA
  1E98-F:INITS   1FD0-F:SVU      20FE-F:GTFN     2208-F:IRBF     22EA-F:RLST
  1EAC-F:EXITS   2029-F:AGTW     2128-F:INB      222D-F:RDW      22F5-R:BFLN
  1EB2-F:ADE     2029-G:C        2130-F:INFO     2235-F:RBD      22F6-BUF:S1
  1EDF-F:DFNO    204E-F:AULS     2136-F:RDIB     2238-F:IRBD     22F7-BUF:S2
  1EE7-F:FOP     206A-F:BAX      213C-F:SLUS     2248-F:GLST     22F8-BUF:CT
  1EF9-F:GFCB    206E-F:CFEN     2141-F:RTAU     224D-F:GSEC     22F9-B:HEAD
  1F00-F:GNE     207A-F:CRAU     2145-F:RTFN     228B-F:AQB      22FA-B:HEDS
  1F2C-F:RLU     2086-F:GCB      2163-F:SFAU

 SEGMENT IOS:      (22FB - 2A67) ABSOLUTE - RAM, SHAREABLE
 SEGMENT RTX:      (2A68 - 31B3) ABSOLUTE - RAM, SHAREABLE
  2B77-R:BGINI   2E89-R:GETM     2ED0-R:GIVM     2F04-R:MPCNT    2F06-R:MPBLKS
  2B8B-R:BGIN2   2EBE-R:EMPI     2EDA-R:GIVMX    2F05-R:MPMAX

 SEGMENT DUMMY:    (3C1B - 3C32) ABSOLUTE - RAM, SHAREABLE

MISSING: BUF:N
```

Shareable Libraries

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE 0005                MULTI-TASKING LINKER (D9, D7, D7, D7, D7)
1979/12/14 10:52:05                    TASK. . . ITASK

TEMPLATE ITASK     (3CA8 - 3CF3) ABSOLUTE - RAM
[LIMITS] RELOCATABLE    0000-N:LOW      02DA-N:HIGH

 SEGMENT ITASK::  (0000 - 0012) RELOCATABLE - RAM
 SEGMENT F:LIBR:: (0013 - 02DA) RELOCATABLE - RAM
  0014-F:XRDA     0137-F:XFDS     0139-F:XFDM     0141-F:XFCI     01C3-F:XFUT
  012E-F:XFDA     0138-F:XFDE     013B-F:XFCR

ABSOLUTE SYMBOLS
  0072-R:MPM1     0073-R:MPM2     FFF6-F:XNFU

 SEGMENT TABLE:    (0100 - 0568) ABSOLUTE - RAM, SHAREABLE
  0535-ITASK      0541-FTASK      054D-ATASK

 SEGMENT FORTRAN  (0579 - 0611) ABSOLUTE - ROM, SHAREABLE
  057D-ITASK:

 SEGMENT G:LIBR   (062F - 06ED) ABSOLUTE - ROM, SHAREABLE
  062F-G:BGIN     06C1-G:CONTS    06CF-G:CHECK    06DA-G:ENTER    06EA-G:EXIT
  06B4-G:ADDR

 SEGMENT F:LIBR   (06EE - 1AF1) ABSOLUTE - ROM, SHAREABLE
  1063-F:RPAU     153C-F:XSTP     15F7-F:XSBW     1745-F:XCLS     184D-F:XENQ
  106E-F:RSTO     153E-F:XCAF     160F-F:XSBR     178D-F:XEND     1863-F:XPSE
  147F-F:ERRS     1555-F:XGLS     1624-F:XBTR     17B3-F:XBCK     1884-F:XINT
  1480-F:XSNE     1584-F:XASW     164B-F:XEBT     1812-F:XREW     1AE7-F:RSTU
  1486-F:ERRC     15B2-F:XASR     1671-F:XOPN     1832-F:XSRN     1AF0-F:XSTF
  1487-F:ERCS

 SEGMENT OS4:     (1AF2 - 1C6F) ABSOLUTE - RAM, SHAREABLE
  1AF2-E:GFN      1C06-ICH:E      1C55-CIBP:      1C57-E:SF       1C5D-UF:E
  1AF4-GFN:E      1C0E-ICH:A      1C56-CIAC:      1C58-E:UF       1C6C-E:SLU
  1BAE-ISM:E      1C41-SBP:       1C57-CIBUF:     1C5B-SF:E       1C6E-E:MON

 SEGMENT SFM:     (1C70 - 22FA) ABSOLUTE - RAM, SHAREABLE
  1DB7-F:DMOUNT   1F83-F:SFC      208D-F:GDIB     21EE-F:GBS      22B3-F:RDS
  1E37-F:MOUNT    1F8E-F:SLU      2091-F:GLUS     21FD-F:RBW      22B5-F:WRS
  1E8F-F:CKMT     1F90-F:SND      2096-F:GTAU     2205-F:RBF      22E4-F:CBA
  1E98-F:INITS    1FD0-F:SVU      20FE-F:GTFN     2208-F:IRBF     22EA-F:RLST
  1EAC-F:EXITS    2029-F:AGTW     2128-F:INB      222D-F:RDW      22F5-R:BFLN
  1EB2-F:ADE      2029-G:C        2130-F:INFO     2235-F:RBD      22F6-BUF:S1
  1EDF-F:DFNO     204E-F:AULS     2136-F:RDIB     2238-F:IRBD     22F7-BUF:S2
  1EE7-F:FOP      206A-F:BAX      213C-F:SLUS     2248-F:GLST     22F8-BUF:CT
  1EF9-F:GFCB     206E-F:CFEN     2141-F:RTAU     224D-F:GSEC     22F9-B:HEAD
  1F00-F:GNE      207A-F:CRAU     2145-F:RTFN     228B-F:AQB      22FA-B:HEDS
  1F2C-F:RLU      2086-F:GCB      2163-F:SFAU

 SEGMENT IOS:     (22FB - 2A67) ABSOLUTE - RAM, SHAREABLE
```

Also indicates how much memory needed for template expansion.

Task descriptor block

Start address of ITASK.

Fortran Library

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE 0006              MULTI-TASKING LINKER (D9, D7, D7, D7, D7)
1979/12/14 10:52:36                  TASK... ITASK

 SEGMENT RTX:      (2A68 - 31B3) ABSOLUTE - RAM, SHAREABLE
  2B77-R:BGINI   2E89-R:GETM     2ED0-R:GIVM     2F04-R:MPCNT    2F06-R:MPBLKS
  2B8B-R:BGIN2   2EBE-R:EMPI     2EDA-R:GIVMX    2F05-R:MPMAX

 SEGMENT DUMMY:    (3C1B - 3C32) ABSOLUTE - RAM, SHAREABLE

MISSING: BUF:N
```


® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE 0007              MULTI-TASKING LINKER (D9,D7,D7,D7,D7)
1979/12/14 10:52:56                  INITIALIZE

[LIMITS] ABSOLUTE   0100-N:LOW     FFFF-N:HIGH

ABSOLUTE SYMBOLS
  001E-R:HLF      006C-R:PFLG     006F-R:SNSW     0072-R:MPM1     007E-R:FATL
  0066-R:CNTK     006D-R:PFK      0070-R:CDRG     0073-R:MPM2     007F-R:FAT1
  0068-R:CNSM     006E-R:SREG     0071-R:BTC1     0078-R:RTCI     0080-RTX:
  006A-R:WLKS

 SEGMENT TABLE:    (0100 - 0568) ABSOLUTE - RAM, SHAREABLE
  0100-R:INIT     04F4-UAT        0504-ECB        0535-ITASK

 SEGMENT GLOBAL    (0569 - 0578) ABSOLUTE - RAM, GLOBAL COMMON
  0569-GLOBAL

 SEGMENT SFM:      (1C70 - 22FA) ABSOLUTE - RAM, SHAREABLE
  1C70-F:CREA     1D56-F:CONN     1DB7-F:DMNT     1E1E-F:MONT     2029-G:C
  1CDA-F:DELE     1D97-F:CFNO

 SEGMENT IOS:      (22FB - 2A67) ABSOLUTE - RAM, SHAREABLE
  22FB-I:IO       2475-I:DOGY     24BD-D:TV00     267A-PR:RD      29E4-I:SIV
  237C-I:SLU      2477-I:LWDT     24D9-C:TY0      267A-TY:RD      2A02-I:RPF
  23A3-I:DOIO     2487-I:KWDT     24F1-TYEOL:     27D6-LP:FUN     2A07-I:COMP
  23BF-I:RET      248E-I:WDAC     24F3-TYTOF:     27D6-PP:FUN     2A0B-I:STAT
  23C9-I:STUP     249A-I:INIT     24F6-TYELI:     27D6-TY:FUN     2A19-I:RST
  23CC-I:STU0     24B4-I:ERTB     24F8-TYBUF:     28AA-FBR:       2A1F-I:EMEM
  23EF-I:STUR     24B9-I:ERS1     24FB-D:LP00     2933-FBW:       2A30-I:EOB
  2403-I:WAIT     24BA-I:ERS2     2520-C:LP0      29B8-I:TRLL     2A34-I:ECTT
  240B-I:FINI     24BB-I:ERS3     2539-C:HEAD     29BF-I:TRTB     2A43-I:VCNT
  2427-I:NEOB     24BC-I:ABRT     253A-LP:WRT     29D9-I:DIO      2A48-I:OPEN
  2433-I:POWR     24BC-I:DOER     253A-PP:WRT     29DA-I:SDIO     2A53-I:OPCL
  2440-I:WDT      24BC-I:DOSV     253A-TY:WRT

SEGMENT RTX:      (2A68 - 31B3) ABSOLUTE - RAM, SHAREABLE
  2A68-R:PATCH    2DD1-R:LTIC     2ED0-R:GIVM     2FEC-R:PWRF     3090-R:SETH
  2A7C-R:TMPL     2DE1-R:KTIC     2EDA-R:GIVMX    300D-R:GTS      30AB-R:STR0EX
  2B0C-R:TSIZE    2DE8-R:TKAC     2F04-R:MPCNT    301D-R:GVST     30AD-R:DOOREX
  2B18-R:BGIN     2DF4-R:ISIG     2F05-R:MPMAX    302D-R:GVSH     30C0-R:RTXEX
  2B77-R:BGINI    2DF4-R:SIG      2F06-R:MPBLKS   303A-R:FPCNT    30C0-R:XPTE
  2B8B-R:BGIN2    2DFD-R:SSIG     2F0B-R:RE12     303B-R:FPMAX    30C4-R:TABL
  2BF7-R:END      2E14-R:IWAI     2F0E-R:DAQX     303F-R:AFB      30C9-R:SYSX
  2C38-R:HALT     2E14-R:WAIT     2F12-R:DA       3046-R:RFB      30E4-R:DCHK
  2C41-R:PRID     2E1C-R:SWAI     2F16-R:DISP     304A-R:AETH     30FD-R:NOPF
  2C4D-R:RIWAL    2E43-R:GPRI     2F4C-R:SACT     3051-R:CNTH     3153-R:IDOR
  2C7C-R:WLAC     2E47-R:GPR      2F52-R:IACB     3058-R:USTH     3181-R:EX
  2CCF-R:WCP      2E4B-R:SPRI     2F7C-R:LSTK     305F-R:UMTH     3182-R:ODOR
  2CD0-R:GATD     2E5F-R:CINT     2FBC-R:KSTK     3066-R:UITH     3186-R:DBUG
  2D52-R:ITIC     2E79-R:CNSL     2FBC-R:STRT     3068-R:UIRTN    3190-R:SERB
  2DA4-R:TICI     2E89-R:GETM     2FCD-R:INI1     307B-R:USTREX   31B3-R:SERL
  2DAA-R:TICP     2EBE-R:EMPI
```

INITIALIZATION TASK

The symbols, above, are in modules loaded. The modules, however, were loaded prior to the LINK encountering a task directive, thus, the modules were not loaded as a result of a TASK.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE 0008                MULTI-TASKING LINKER (D9,D7,D7,D7,D7)
1979/12/14 10:53:29                    INITIALIZE

 SEGMENT DEBUG:    (31B4 - 3C1A) ABSOLUTE - RAM, SHAREABLE
  31B4-DEBUG4     31B4-XDBUG4     3797-DBG4:CNT   37A9-DBG4:LP

 SEGMENT DUMMY:    (3C1B - 3C32) ABSOLUTE - RAM, SHAREABLE
  3C1B-R:RINT     3C23-R:RECV     3C27-R:STOD     3C2B-R:CTIC     3C2F-I:LOAD
  3C1D-R:UINI     3C24-R:PAUS     3C28-R:GTOD     3C2C-R:MTIC     3C30-I:LD410S
  3C1F-R:EMLTR    3C25-R:ABUF     3C29-R:AWAL     3C2D-R:CWAL     3C31-R:DUMMY
  3C22-R:SEND     3C26-R:RBUF     3C2A-R:IWAL     3C2E-R:SATD
```


● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE 0009              MULTI-TASKING LINKER (D9, D7, D7, D7, D7, D7)
1979/12/14 10:54:17                  ALL SYMBOLS

 BUF:N UNDEFINED, REFERENCED IN BUF:S1 AT 0003+0685+1C70(SFM:)

THE TRANSFER ADDRESS IS 31B4(DEBUG:)

ABSOLUTE SYMBOLS
  001E-R:HLF      006C-R:PFLG     006F-R:SNSW     0072-R:MPM1     007E-R:FATL
  0066-R:CNTK     006D-R:PFK      0070-R:CDRG     0073-R:MPM2     007F-R:FAT1
  0068-R:CNSM     006E-R:SREG     0071-R:BTC1     0078-R:RTCI     0080-RTX:
  006A-R:WLKS

 MEMORY  ABSOLUTE    0100-R:LOW      3CF3-R:HIGH

 SEGMENT TABLE:    (0100 - 0568) ABSOLUTE - RAM, SHAREABLE
  0100-R:INIT     04F4-UAT        0504-ECB        0535-ITASK

 SEGMENT GLOBAL    (0569 - 0578) ABSOLUTE - RAM, GLOBAL COMMON
  0569-GLOBAL

 SEGMENT FORTRAN   (0579 - 0611) ABSOLUTE - ROM, SHAREABLE
 SEGMENT ATASK     (0612 - 062E) ABSOLUTE - RAM, SHAREABLE
 SEGMENT G:LIBR    (062F - 06ED) ABSOLUTE - ROM, SHAREABLE
 SEGMENT F:LIBR    (06EE - 1AF1) ABSOLUTE - ROM, SHAREABLE
 SEGMENT OS4:      (1AF2 - 1C6F) ABSOLUTE - RAM, SHAREABLE
 SEGMENT SFM:      (1C70 - 22FA) ABSOLUTE - RAM, SHAREABLE
  1C70-F:CREA     1D56-F:CONN     1DB7-F:DMNT     1E1E-F:MONT     2029-G:C
  1CDA-F:DELE     1D97-F:CFNO

 SEGMENT IOS:      (22FB - 2A67) ABSOLUTE - RAM, SHAREABLE
  22FB-I:IO       2475-I:DOGY     24BD-D:TV00     267A-PR:RD      29E4-I:SIV
  237C-I:SLU      2477-I:LWDT     24D9-C:TY0      267A-TY:RD      2A02-I:RPF
  23A3-I:DOIO     2487-I:KWDT     24F1-TYEOL:     27D6-LP:FUN     2A07-I:COMP
  23BF-I:RET      248E-I:WDAC     24F3-TYTOF:     27D6-PP:FUN     2A0B-I:STAT
  23C9-I:STUP     249A-I:INIT     24F6-TYELI:     27D6-TY:FUN     2A19-I:RST
  23CC-I:STU0     24B4-I:ERTB     24F8-TYBUF:     28AA-FBR:       2A1F-I:EMEM
  23EF-I:STUR     24B9-I:ERS1     24FB-D:LP00     2933-FBW:       2A30-I:EOB
  2403-I:WAIT     24BA-I:ERS2     2520-C:LP0      29B8-I:TRLL     2A34-I:ECTT
  240B-I:FINI     24BB-I:ERS3     2539-C:HEAD     29BF-I:TRTB     2A43-I:VCNT
  2427-I:NEOB     24BC-I:ABRT     253A-LP:WRT     29D9-I:DIO      2A48-I:OPEN
  2433-I:POWR     24BC-I:DOER     253A-PP:WRT     29DA-I:SDIO     2A53-I:OPCL
  2440-I:WDT      24BC-I:DOSV     253A-TY:WRT
```

Offset into module

Start of REL area

Start of module relative to SFM; thus address =1C70+ 685 + 3

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

```
PAGE 0010          MULTI-TASKING LINKER (D9, D7, D7, D7, D7, D7, D8)
1979/12/14 10:55:27                ALL SYMBOLS

 SEGMENT RTX:      (2A68 - 31B3) ABSOLUTE - RAM, SHAREABLE
  2A68-R:PATCH   2DD1-R:LTIC    2ED0-R:GIVM    2FEC-R:PWRF    3090-R:SETH
  2A7C-R:TMPL    2DE1-R:KTIC    2EDA-R:GIVMX   300D-R:GTS     30AB-R:STR0EX
  2B0C-R:TSIZE   2DE8-R:TKAC    2F04-R:MPCNT   301D-R:GVST    30AD-R:DOOREX
  2B18-R:BGIN    2DF4-R:ISIG    2F05-R:MPMAX   302D-R:GVSH    30C0-R:RTXEX
  2B77-R:BGINI   2DF4-R:SIG     2F06-R:MPBLKS  303A-R:FPCNT   30C0-R:XPTE
  2B8B-R:BGIN2   2DFD-R:SSIG    2F0B-R:RE12    303B-R:FPMAX   30C4-R:TABL
  2BF7-R:END     2E14-R:IWAI    2F0E-R:DAQX    303F-R:AFB     30C9-R:SYSX
  2C38-R:HALT    2E14-R:WAIT    2F12-R:DA      3046-R:RFB     30E4-R:DCHK
  2C41-R:PRID    2E1C-R:SWAI    2F16-R:DISP    304A-R:AETH    30FD-R:NOPF
  2C4D-R:RIWAL   2E43-R:GPRI    2F4C-R:SACT    3051-R:CNTH    3153-R:IDOR
  2C7C-R:WLAC    2E47-R:GPR     2F52-R:IACB    3058-R:USTH    3181-R:EX
  2CCF-R:WCP     2E4B-R:SPRI    2F7C-R:LSTK    305F-R:UMTH    3182-R:ODOR
  2CD0-R:GATD    2E5F-R:CINT    2FBC-R:KSTK    3066-R:UITH    3186-R:DBUG
  2D52-R:ITIC    2E79-R:CNSL    2FBC-R:STRT    3068-R:UIRTN   3190-R:SERB
  2DA4-R:TICI    2E89-R:GETM    2FCD-R:INI1    307B-R:USTREX  31B3-R:SERL
  2DAA-R:TICP    2EBE-R:EMPI

 SEGMENT DEBUG:   (31B4 - 3C1A) ABSOLUTE - RAM, SHAREABLE
  31B4-DEBUG4    31B4-XDBUG4    3797-DBG4:CNT  37A9-DBG4:LP

 SEGMENT DUMMY:   (3C1B - 3C32) ABSOLUTE - RAM, SHAREABLE
  3C1B-R:RINT    3C23-R:RECV    3C27-R:STOD    3C2B-R:CTIC    3C2F-I:LOAD
  3C1D-R:UINI    3C24-R:PAUS    3C28-R:GTOD    3C2C-R:MTIC    3C30-I:LD410S
  3C1F-R:EMLTR   3C25-R:ABUF    3C29-R:AWAL    3C2D-R:CWAL    3C31-R:DUMMY
  3C22-R:SEND    3C26-R:RBUF    3C2A-R:IWAL    3C2E-R:SATD

TEMPLATE ATASK    (3C33 - 3C60) ABSOLUTE - RAM, SHAREABLE
TEMPLATE FTASK    (3C61 - 3CA7) ABSOLUTE - RAM, SHAREABLE
TEMPLATE ITASK    (3CA8 - 3CF3) ABSOLUTE - RAM, SHAREABLE
C H A I N S
LP0:      24FB(IOS:)
CIB:      2520(IOS:)
TY:       24BD(IOS:)
BUF:      0000(0SEGMENT)
BUF::     0000(0SEGMENT)
R:ECBL    0504(TABLE:)
R:TDBL    054D(TABLE:)
R:SL      0565(TABLE:)
R:MBXL    0000(0SEGMENT)

MISSING: SRF:ER, BUF:N

4 WARNINGS
```

Dummy Modules

Normal completion. These two references always appear here.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

APPENDIX A

TYPE-CODE FUNCTIONS

The following information defines the type-codes according to their individual functions:

Type-Code :02 Name TCENDA Layout C

Meaning: End absolute program unit

The VALUE field is either a starting address of the source program or a minus one (:FFFF) to indicate the omission of the transfer address. Type-code :02 indicates the end of an absolute program unit.

Type-Code :03 Name TCENDR Layout C

Meaning: End relocatable program unit

The VALUE field is either a starting address of the source program or a minus one (:FFFF) to indicate the omission of the transfer address. Type-code :03 indicates the end of a relocatable program unit. The transfer address is assumed relocatable in the segment defined by the last :69 type-code.

Type-Code :04 Name TCORGA Layout C

Meaning: Absolute origin

The location counter is set to the VALUE field. The absolute value is used.

Type-Code :05 Name TCORGR Layout C

Meaning: Relocatable origin

The location counter is set to the VALUE field, relocated by the base address of the segment specified in the last :69 type-code.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Type-Code :06 Name TCTAB Layout D

Meaning: Absolute data

The COUNT field indicates how many VALUE fields follow. The VALUE fields are taken as a block of absolute data. If type-code :06 is placed in a legal stack expression in LINK Postfix Notation (i.e., a :66-:67 or :68 sequence or a :6A-:62 sequence), the data generated is placed onto the Postfix stack. In any other context this type-code causes the data to be deposited at the current value of the location counter and the location counter to be incremented.

Type-Code :07 Name TCTRLW Layout D

Meaning: Relocatable data

The COUNT field indicates how many VALUE fields follow. The VALUE fields are taken as a block of relocatable data. If this type-code is placed in a legal stack expression in LINK Postfix Notation (i.e., a :66-:67 or :68 sequence or a :6A-:62 sequence), the data generated is placed onto the Postfix stack. In any other context type code :07 causes the data to be deposited at the current value of the location counter and the location counter to be incremented. The data is relocated by the base address of the segment specified in the last :69 type code.

Type-Code :08 Name TCRESF Layout E

Meaning: Reserve and fill here

Type-code :08 causes a block of absolute data (with value VALUE and length count) to be deposited at the current position of the location counter. The location counter is incremented by COUNT.

Type-Code :09 Name TCBRLW Layout C

Meaning: Byte address constant

The VALUE field is taken to be a byte address constant and VALUE+R+R is deposited at the current position of the location counter, where R is the base of relocation of the segment defined by the last :69 type-code. The location counter is incremented by one.

Type-Code :18 Name TCBGN3 Layout C

Meaning: Begin program

The VALUE field currently contains the system flag in bit 15. This type-code must be the first type-code of an NM4 object program unit.

REVISION 00

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Type-Code :40 Name TCNULL Layout A

Meaning: Null operation

Type-code :40 is usually used for filler and is ignored.

Type-Code :47 Name TCINTY Layout N

Meaning: Initial Y-register

The TN (Task Name) index in this type-code refers to the Task Name list generated by the :49 type code, TCTASK, List Task Name. The SEGMENT Index refers to the Segment list generated by the :4F type code, TCSGMT, List Segment Names. During R:BEGIN, the Y-register will be set to the VALUE + the offset of Segment in the expanded template.

Type-Code :48 Name TCTMPL Layout B

Meaning: Template Address

The Index in the type-code refers to the Task Name list generated by the :49 type-code, TCTASK List Task Names. A word is generated which is the address of the template for the Indexed Task.

Type-Code :49 Name TCTASK Layout F

Meaning: List Task Names

A list is generated as for type-code :4A TCENTP, List (primary) Definitions. This list however is of Task Names and is referenced by type-codes :48 TCMPL, Template Address and :47 TCINTY, Initial Y-register. NAM's in a program with type-code :49 will be defined only for those tasks listed with one exception: NAM's with the same spelling as a Task Name will be defined for all tasks.

Type-Code :4A Name TCENTP Layout F

Meaning: List primary definitions

The BYTES field indicates the number of bytes of data which follow. It must be a multiple of 8. The 8-SYMBOL fields which follow are 8-byte symbols, filled with blanks if necessary. A maximum of 32 symbols may be defined in one type-code. To define more than 32 symbols, two or more type-codes must be used. Multiple :4A type-codes must be contiguous. A block of these type-codes generates an ordered list of primary definition symbols which are to be referenced later by its INDEX. The first symbol is taken to be INDEX 0, the second INDEX 1, etc. The symbols are defined by a :67 type-code.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Type-Code :4B Name TCEXTP Layout F

Meaning: List primary externals

A list is generated as for type-code :4A, except that the list is of primary external symbol references. External symbols are referenced by :74 type-codes.

Type-Code :4C Name TCENTS Layout F

Meaning: List secondary definitions

A list is generated as for type-code :4A, except that the list is of secondary symbol definitions defined by a :68 type-code.

Type-Code :4D Name TCEXTS Layout F

Meaning: List secondary externals

A list is generated as for type-code :4A except that the list is of secondary external symbols referenced by :75 type-codes.

Type-Code :4E Name TCCHNP Layout F

Meaning: List Chains

A list is generated as for type-code :4A except that the list is of chain symbols referenced by :6B and :6C type-codes.

Type-Code :4F Name TCSGMT Layout F

Meaning: List segment names

A list is generated as for type-code :4A except that the list consists of segment names referenced by type-codes :47, :64 and :69.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Stack Type-Codes Names as below Layout A

Type-Code	Name	Meaning
:56	TCMNEG	Stack negate (monadic) operator
:57	TCMSFT	Stack shift operator
:58	TCMAND	Stack logical AND operator
:59	TCMIOR	Stack logical inclusive OR operator
:5A	TCMXOR	Stack logical exclusive OR operator
:5B	TCMNOT	Stack logical NOT (monadic) operator
:5C	TCMEQ	Stack relational equate operator
:5D	TCMNE	Stack relational not equal operator
:5E	TCMGT	Stack relational greater than operator
:5F	TCMLT	Stack relational less than operator
:60	TCMGE	Stack relational greater than or equal operator
:61	TCMLE	Stack relational less than or equal operator
:70	TCMADD	Stack Add operator
:71	TCMSUB	Stack Subtract operator
:72	TCMMPY	Stack Multiply operator
:73	TCMDVD	Stack Divide operator

The stack operators operate on the Postfix stack under the LINK. Postfix Notation allows 1) expressions to be written without parentheses and 2) operators to appear after operands. Type-codes :56 and :5B are monadic operators which cause the top element of the stack to be removed and the result of the operation to be deposited back on the stack. All other type-codes are diadic operators which cause two elements to be taken off the stack. The operation yields a single result which is deposited back on the stack.

Type-Code :62 Name TCKOFD Layout K

Meaning: Field from stack

Type-code :62 provides a termination for a sequence of stack operations begun with a begin stack :6A type code. When a :62 type code is encountered, the top element of the stack is removed, shifted as indicated by the shift count (+ is left, - is right, 0 is no shift), and ANDed with the MASK. The result is placed in the current position of the location counter. The location counter is incremented by one. Type-codes :06, :07, :64, :74, and :75 have a different effect when placed between a begin stack :6A type-code and a stack terminater :62 type-code. The value of the operand concerned is pushed onto the Postfix stack under the LINK rather deposited at the position of the location counter.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Type-Code :64 Name TCINTR Layout G

Meaning: Intermediate access, relocatable

Type-code :64 is used to produce as data a reference to a location in another segment without referring to an external reference. The data stored is determined as R+VALUE, where R is the base of relocation of the segment associated with index INDEX in this program unit. If this type-code is placed in a legal stack expression in Link Postfix Notation (i.e., a :66-:67 or :68 sequence or a :6A-:62 sequence), the data generated is placed onto the Postfix stack. In any other context this type-code causes the data to be deposited at the current value of the location counter and the location counter to be incremented.

Type-Code :67 Name TCDENP Layout B

Meaning: Primary Symbol Definition

Type-code :67 causes the primary definition indexed by INDEX in the symbol list generated by type-code :4A to be defined. The definition is derived from the top element of the stack if it is used as a termination of a Postfix stack expression. In any other use the defined value is taken to be that of the current value of the location counter.

Type-Code :68 Name TCDENS Layout B

Meaning: Secondary Symbol Definition

Type-code :68 is like type code :67 above except that a secondary definition is defined and the INDEX field refers to a symbol list generated by a :4C type-code.

Type-Code :69 Name TCLOC Layout B

Meaning: Location Counter Change

Type-code :69 is used to change the location counter from referencing one segment to another. The INDEX is an index into the list generated by the :4F type-code. This type-code should be followed immediately by either origin type-codes :04, :05 or by end program type-codes :02, :03. If followed by :04, :05 type-codes, the value of the location counter in the indexed segment is defined. If followed by :02, :03 type codes, the start address of the program is defined to be in the indexed segment.

Type-Code :6B Name TCNCHP Layout B

Meaning: Chain Element

Type-code :6B provides an element in a chain linked list. At link or load time, each new Chain Element Record causes a word to be written which is the address of the previous Chain Element TC Record element. The first Chain Element Record is written as a zero. The INDEX refers to the Chain Element list as generated by the :4E type-code.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Type-Code :6C Name TCHCHP Layout B

Meaning: Chain Head

Type-code :6C provides a head for a chain linked list. At link or load time, a word is written which is the address of the last element in the linked list. The INDEX refers to the Chain Element list as generated by the :4E type-code.

Type-Code :6D Name TCNOTE Layout J

Meaning: Note Record

Type-code :6D specifies BYTES of commentary information and does not affect the operation of the LINK. TCNOTES may optionally be printed and/or copied into the output file(s).

Although no inherent meaning is associated with TCNOTE type-codes, they are divided into three classes for processing by the LINK. A particular TCNOTE type-code is assigned to a class based on the byte immediately following the count. The classes are listed below.

CLASS	DEFINITION
FIRST BYTE 0	This class is used by language translators to insert their names and revision into the object text. The LINK deletes (unless the COPYXLATID LINK option is specified) and does not print this class (unless the XLATID LINK option is specified).
FIRST BYTE :80..:FF	This class has no meaning to the LINK. The LINK copies (unless the NOCOPYOBJ NOTES LINK option is specified) and does not print this class (unless the OBJNOTES LINK option is specified). If this class is printed (OBJNOTES), the hexadecimal representation of the data is printed.
FIRST BYTE :01...:7F	This class is assumed to be commentary and is deleted (unless the COPYNOTES LINK option is specified) and copied and printed (unless the NONOTES LINK option is specified) by the LINK.

Type-Code :6F Name TCSGML Layout H

Meaning: List Segment Attributes

Type-code :6F is used in conjunction with the :4F type-code to specify the length of, and the flags associated with, each segment of the list generated by type-code :4F. If the number of segments generated by the :4F type-codes is N, then BYTES must be $3_{*}N$. The data for each segment is a two-byte VALUE and a one-byte FLAGS. The VALUE is the length (in words) of that segment and the FLAGS are as defined below. Type-code :6F is mandatory and must follow the :4F type-code group, with which it is associated. The defined flags are:

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

Bit	Meaning if set
0	ROMMABLE
1	COMMON
2	SHARABLE

The FLAGS byte can assume the following meanings:

Bits: 2 1 0

0 0 0	Non Sharable Data
0 0 1	Non Sharable Data in ROM
0 1 0	Common Data
0 1 1	Common Data in ROM
1 0 0	Sharable Data
1 0 1	Sharable Data in ROM
1 1 0	Global Common
1 1 1	Global Common in ROM

Type-Code :74 Name TCTEXP Layout B

Meaning: Text External (primary)

The INDEX in type-code :74 refers to the symbol list generated by the :4B type-code. A word is generated which is an address of a primary external symbol. If this type-code is placed in a legal stack expression (i.e., a :66-:67 or :68 sequence, or a :6A-:62 sequence) then the word generated is placed on the stack. In any other context type code :74 causes a word to be deposited at the current value of the location counter and the location counter to be incremented.

Type-Code :75 Name TCTEXS Layout B

Meaning: Text External (secondary)

Type-code :75 has the same effect as type-code :74 except that the symbol referenced is a secondary external symbol and the INDEX refers to the list generated by type-code :4D.

Type-Code :77 Name TCNTSK Layout L

Meaning: Task Execution Instances (Reserved)

Type-Code :78 Name TCDADR Layout M

Meaning: Task Y-Pool Generation (Reserved)

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

APPENDIX B

LINK EDITOR UTILITY MESSAGES

The LINK Editor utility generates both warning messages and error messages.

B.1 LINK EDITOR WARNING MESSAGES

ATTEMPTING TO ORG OUTSIDE OF SEGMENT

Probable cause:	User attempted to reference an address beyond the defined limits of the segment.
Suggested response:	Provide correct address for the segment.

ASSIGNED GROUP SEGMENTS MUST BE SHAREABLE

Probable cause:	User attempted to assign an address to a segment that was not shareable.
Suggested response:	Segment must be reassigned.

COMMON CONFLICT

Probable cause:	A segment has been defined as common in one area and non-common in another. This caused conflicting segment definition.
Suggested response:	Segment should be redefined.

CONFLICTING ORDERING DEMANDS

Probable cause:	User requested contradictory arrangement of segments. (For example, indicated segment B follow segment A but at the same time indicated segment A follow segment B.)
Suggested response:	Segment should be redefined.

EXCEEDED MEMORY LIMIT

Probable cause:	Memory allocation exceeds user-defined memory limit. The LINK Editor Utility will continue to operate as if there is 64K in target system.
Suggested response:	Memory requirement should be reduced.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

EXPECTED DELIMITER, got XXXX

Probable cause:	Error in command line.
Suggested response:	Correct and relink.

EXPECTED RIGHT PAREN.

Probable cause:	Syntax error.
Suggested response:	Correct and relink.

GROUP SHAREABLE CONFLICT

Probable cause:	User requested that shareable and non-shareable segments be placed next to one another and caused a segment error.
Suggested response:	Redefine segments.

I DON'T UNDERSTAND--XXXX*

Probable cause:	Input entered by user cannot be interpreted by the LINK.
Suggested response:	Correct command line and re-enter.

INITIAL Y MUST POINT TO NON-SHAREABLE SEGMENT

Probable cause:	When multitasking, the initial value of Y can be specified by the INITY macro in the assembler. However, for a non-shareable segment, initial Y value must be defined.
Suggested response:	Define initial Y.

INTERMEDIATE ACCESS--ABS Ø OUTPUT USED

Probable cause:	User attempted to access data defined in a template.
Suggested response:	Since the LINK Editor does not know where the template will be expanded at runtime, absolute Ø was used.

MULTIPLY DEFINED

Probable cause:	User tried to assign two (or more) different memory addresses to the same symbol.
Suggested response:	Redefine symbol address.

---

*XXXX is the actual command input by the user.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

NON-SHAREABLE DATA NOT ALLOWED IN SYSTEM MODULES

Probable cause:	User can specify a module as a system module if the module does not contain non-shareable data to be loaded. The LINK Editor then allows backwards referencing of the module.
Suggested response:	Remove non-shareable data from module.

ROMMABLE CONFLICT

Probable cause:	A segment has been defined as ROMable in one area and non-ROMable in another, result ing in a conflicting segment definition.
Suggested response:	Redefine the segment.

SHAREABLE CONFLICT

Probable cause:	A segment was defined as shareable in one area and non-shareable in another resulting in a conflicting segment definition.
Suggested response:	Segment should be redefined.

TASK NAMS MUST BE SHAREABLE

Probable cause:	The NAM indicated corresponds to a name in the task directive; thus the module defines the main program of a task and must be shareable.
Suggested response:	Redefine TASK NAMs to be shareable.

TEMPLATE FOR UNDECLARED TASK

Probable cause:	LINK Editor Utility detects a template defined for an undeclared task.
Suggested response:	Review program.

TEMPLATE SIZE 64K--TRUNCATED

Probable cause:	Template requires more memory than available.
Suggested response:	Reduce template size.

TOO LARGE

Probable cause:	The input number on the command line exceeds character limit.
Suggested response:	Default used. (Refer to command line option.)

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

TOO MANY TASKS

Probable cause:	Program contains more tasks than can be accommodated by the LINK. Maximum is 126 tasks or the total of tasks and overlays cannot exceed 126.
Suggested response:	Reduce task requirement.

TOO SMALL

Probable cause:	The input number on the command line is too small.
Suggested response:	Default used. (Refer to command line option.)

UNDEFINED, REFERENCED

Probable cause:	A task has referenced a symbol that is undefined.
Suggested response:	Define symbol.

WHAT DO YOU MEAN?...

Probable cause:	Program cannot interpret input.
Suggested response:	(Depends on input.)

64K MEMORY EXCEEDED--USING REL 0

Probable cause:	Program exceeds the 64K memory limit at runtime.
Suggested response:	Reduce memory requirement.

B.2 LINK EDITOR UTILITY ERROR MESSAGES

ABORT

Probable cause:	An irrecoverable error, mistake, or malfunction has occurred. An ABORT can be caused by the operating program or by the operator.

AT OPERATOR'S REQUEST--ABORT

Probable cause:	The operator has entered ABORT! or KILL! on the command line as a file name or option causing the function in process to ABORT.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

BAD BEGIN TYPE CODE

Probable cause: Bad input.

Suggested response: The correct type-code is :18 (type-code name is TCBGN3).

BAD FILE NAME

Probable cause: Utility unable to interpret or find on disk the file name provided.

Suggested response: Recheck file command line.

BAD OUTPUT FILE NAME

Probable cause: Utility unable to interpret or find on disk the output file name as entered.

Suggested response: Recheck file command line.

DICTIONARY TOO LARGE

Probable cause: Program module exceeds 256 symbols.

Suggested response: Re-evaluate program!

EQUAL SIGN INVALID WITH NOMAXWARNINGS

Probable cause: Syntax error in the command line. This error indicates that the statement "NOMAXWARNING" followed by an equal sign is illogical and incorrect. (NOMAXWARNINGS indicates that the LINK Editor will not abort because of warnings.)

Suggested Response: Correct command line and relink.

EXPECTED TCLOC

Probable cause: Bad input. Program requires this type code for the internal location counter.

Suggested response: The correct type code is :69.

INVALID DELIMITER

Probable cause: Syntax error in command line.

Suggested response: Correct and relink.

● REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

INVALID DELIMITER--EXPECTED "="

Probable cause: Syntax error in the command line.

Suggested response: Correct and relink.

INVALID SYMBOL

Probable cause: The LINK detected an invalid symbol name in the command.

Suggested response: Correct and relink.

NO BEGIN TYPE-CODE

Probable cause: Program requires TCBGN3.

Suggested response: The correct type-code is :18.

NOT DECIMAL VALUE

Probable cause: User specified a non-decimal value as decimal in the command line.

Suggested response: Correct and recompile.

NOT HEX VALUE

Probable cause: User specified a non-hexadecimal value as hexadecimal in the command line.

Suggested response: Correct and recompile.

ONLY COUNTS OF 1 IN STACKS
ONLY HANDLE 1 LEVEL STACKS
ONLY HANDLE 1 STACK LEVEL

Probable cause: These errors refer to an internal stack controlled by the LINK. A language should not be emitting type codes to force this stack deeper than 1 level.

SEGML COUNT EXCEEDS DICTIONARY LENGTH

Probable cause: Bad input. User provided data for segments not indicated.

Suggested response: Re-evaluate the program.

SEGML COUNT NOT DIVISIBLE BY 3

Probable cause: Bad input. Should be 3 bytes per segment.

Suggested response: Review segment definitions.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

TOO MANY SEGMLS

Probable cause: Language processor problem. User provided more definitions of segments than segments.

Suggested response: Try recompiling or reassembling.

UNIT RECORD OUTPUT NOT SUPPORTED

Probable cause: User attempted to output to a unit record device.

Suggested response: User should output to a file on a device.

USER MUST SUPPLY NUMBER WITH MAXWARNINGS

Probable cause: MAXWARNINGS option requires that the user specify the maximum number of warnings for the LINK Editor to provide before aborting the program.

Suggested response: Provide MAXWARNINGS limit.

B.3 LINK EDITOR RUNTIME ERRORS

LINKER RUNTIME I/O INTERFACE ERROR!

Probable cause: This error occurs if LINK encounters a problem with one of the following:

- Initial program load
- MEM Setup
- EOF: LUN
- GET: LUN
- PUT: OP
- READ: OP
- REWIND: LUN
- RUN: PROG LOAD
- WRITE: OP
- POSITION: LUN

Suggested response: The user should replace the copy of the LINK Editor Utility and try to relink the program. If the problem persists diagnostics should be run to test for a hardware error. If a hardware error is not found and the problem continues, contact the local Computer Automation, Inc., representative.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.

LINKER RUNTIME INTERPRETER ERROR!

Probable cause:

This error occurs if the LINK encounters a problem with one of the following:

- OVERFLOW
- POINTER
- RANGE
- VARIANT
- HEAP
- STACK

Suggested response:

The user should replace the copy of the LINK Editor Utility and try to relink the program. If the problem continues, the user should contact the local Computer Automation, Inc., representative.

® REGISTERED TRADEMARK OF COMPUTER AUTOMATION, INC.

© 1980 COMPUTER AUTOMATION, INC.