General Notice

When using this document, keep the following in mind:

1. This document is confidential. By accepting this document you acknowledge that you are bound by the terms set forth in the non-disclosure and confidentiality agreement signed separately and /in the possession of SEGA. If you have not signed such a non-disclosure agreement, please contact SEGA immediately and return this document to SEGA.

2. This document may include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new versions of the document. SEGA may make improvements and/or changes in the product(s) and/or the program(s) described in this document at any time.

3. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without SEGA'S written permission. Request for copies of this document and for technical information about SEGA products must be made to your authorized SEGA Technical Services representative.

4. No license is granted by implication or otherwise under any patents, copyrights, trademarks, or other intellectual property rights of SEGA Enterprises, Ltd., SEGA of America, Inc., or any third party.

5. Software, circuitry, and other examples described herein are meant merely to indicate the characteristics and performance of SEGA's products. SEGA assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples describe herein.

6. It is possible that this document may contain reference to, or information about, SEGA products (development hardware/software) or services that are not provided in countries other than Japan. Such references/information must not be construed to mean that SEGA intends to provide such SEGA products or services in countries other than Japan. Any reference of a SEGA licensed product/program in this document is not intended to state or simply that you can use only SEGA's licensed products/programs. Any functionally equivalent hardware/software can be used instead.

7. SEGA will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user's equipment, or programs according to this document.

NOTE: A reader's comment/correction form is provided with this document. Please address comments to :

SEGA of America, Inc., Developer Technical Support (att. Evelyn Merritt)
150 Shoreline Drive, Redwood City, CA 94065

SEGA may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

SEGA OF AMERICA, INC.
Consumer Products Division

SH2
Dynamic Load Linkage Editor Function Specification

Doc. #- ST-19-R1-B-050994

READER CORRECTION/COMMENT SHEET

Keep us updated!

If you should come across any incorrect or outdated information while reading through the attached document, or come up with any questions or comments, please let us know so that we can make the required changes in subsequent revisions. Simply fill out all information below and return this form to the Developer Technical Support Manager at the address below. Please make more copies of this form if more space is needed. Thank you.

General Information:

Your Name ______________________ **Phone** __________

Document number ST-19-R1-B-050994 **Date** __________

Document name SH2 Dynamic Load Linkage Editor

Corrections:

Chpt.	pg. #	Correction

Questions/comments: ______________________

Where to send your corrections:

Fax: (415) 802-3963
Attn: Manager,
Developer Technical Support

Mail: SEGA OF AMERICA
Attn: Manager,
Developer Technical Support
275 Shoreline Dr. Ste 500
Redwood City, CA 94065

SH2 Dynamic Load Linkage Editor Function Specification

1st	Revision	May 20, 1993
2nd	Revision	July 20, 1993
3rd	Revision	April 12, 1994

Model Name: HS0700LECU1SM

1.0 Overview

1.1 Introduction

This software was developed by request of SEGA and this specification was determined through arrangements with SEGA.

1.2 Reason for Development

Will be used to develop a tool which will enable dynamic load in the SH2 program (limited to the PC compatible jump command description).

1.3 Basic Direction

(1) Premise Conditions
- Specifications will reflect SH2 user opinions.
- The objects output by this tool can be loaded into the SH2 emulator and debugged at the source level. Emulator specifications will also change for this reason.

(2) Applicable Range
This is a dynamic load tool for the SH2 program developed at the request of SEGA.

USE: This is a tool to create load modules for the SH2 program described in Position Independent Instructions only (data reference is not applicable) for easy loading via the user-created dynamic loader.

Applicable Users: Specialized Customers (SEGA)

(3) Relations with Other Systems
Cross Software: This tool is configured of the linker preprocessor dlt and out put merge/S type conversion dlt2. Therefore, it is also closely related to the H series linker.

Host Systems: Operates on IBM-PC, SPARC, HP9000/700. It also can be connected to the cross software, emulator, and small ever board of each host.

Window: Does not use window interface.

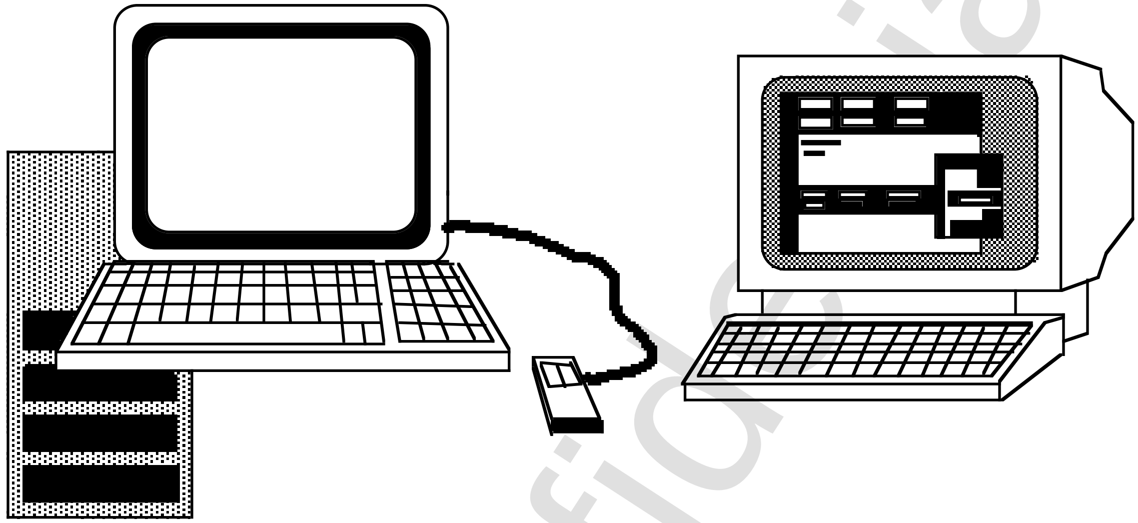
(4) Consideration for Function Expansion
Empty area in the control table has been provided to allow the dynamic load format to use flexible formats depending on how the loader was created.

1.4 Related Documentation

(1) H Series Linkage Editor User's Manual

1.5 System Configuration

(1) Tool Operating Environment (UNIX environment, IBM-PC environment)



Workstation
Sun SPARC station
HP9000/700

IBM-PC/AT

(2) Equipment Configuration

Minimum System Requirements:

IBM-PC: OS must be MS-DOS V5.0 or above
Uses less than 1MB
Uses DOS extender.

SPARC: Sun-OS V4.0.1
Uses less than 1MB

HP9000/700: HP-UX V8.0
Uses less than 1MB

2.0 Dynamic Load Method Overview and How to use DLT

2.1 Overview

2.1.1 Example of a General Dynamic Load Format

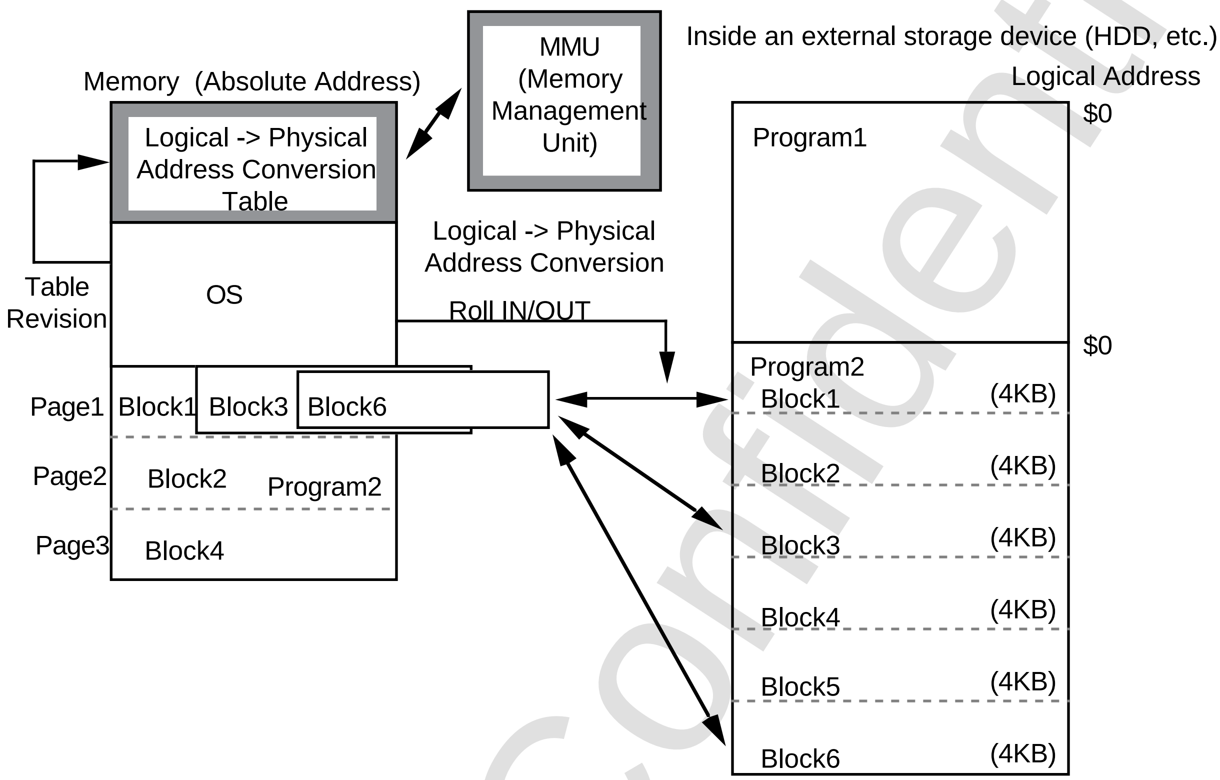


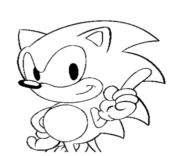
Figure 2-1 Example of a general dynamic load format

Figure 2-1 shows the most commonly used dynamic load format. Dynamic load is the method used when the size of the program being executed is larger than the actual amount of memory; multi-job OS systems such as UNIX, MS-Windows commonly use this method.
The common method of executing dynamic load is listed below.

1. All of the user program is stored at the logical address (in other words, the lead address is $0) on the storage device.

2. The MMU is used to convert the logical address and the physical memory address (absolute address).

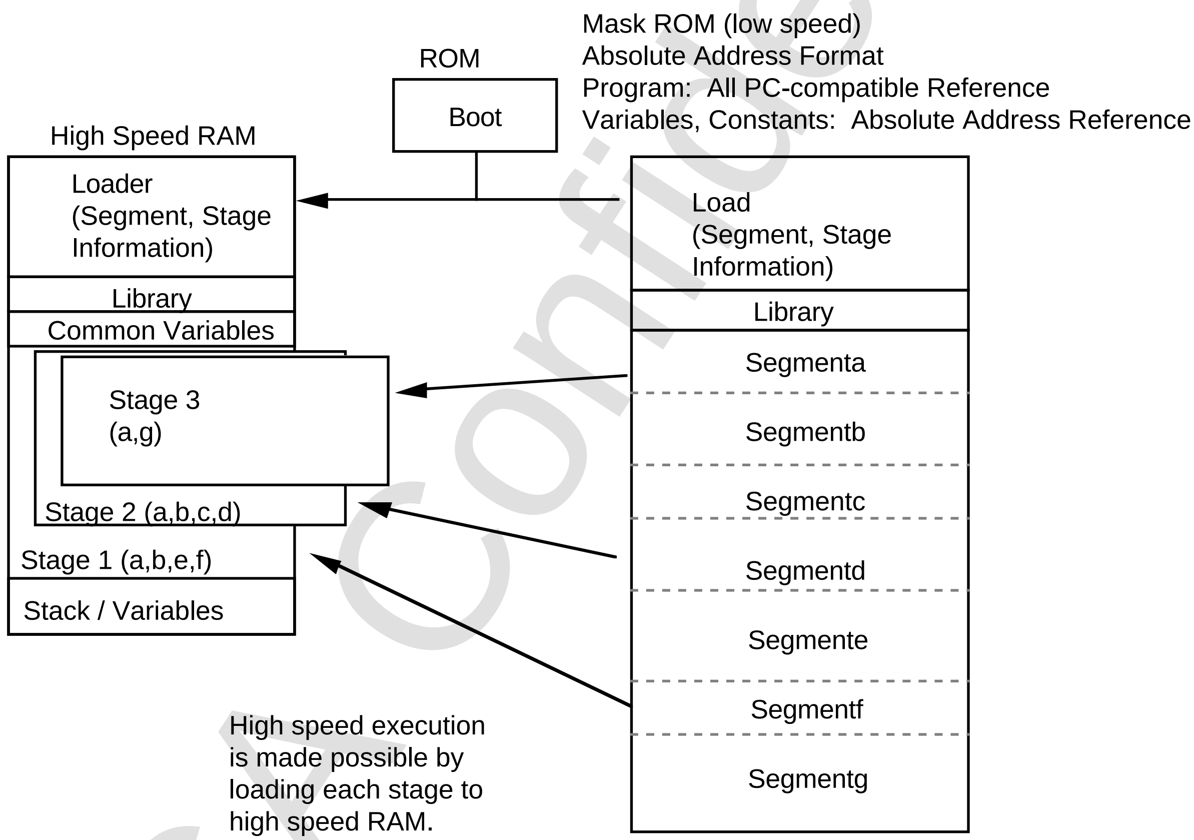
3. The OS revises the table used to control the address conversion information. When the program is executing, only the portions that are required for the program to run are loaded into memory; the unnecessary portion is saved on the storage device. This is



controlled by the OS, and address conversion is controlled by the MMU. (Here load / save units are called pages, and load and save are called ROLL IN/OUT.) To use this method, the MMU hardware and an OS are required.

In the case where SEGA's support is considered, the following problems exist: the external storage device is a ROM cassette; adding the MMU would increase the cost; mutli-job processing is not required; and adding an OS would degrade the performance. For these reasons a simple dynamic load method that does not require an MMU is used.

2.1.2 Dynamic Load Method for SEGA



* Loader:	Program that loads each segment from ROM to RAM. This includes stage and segment information. It is brought to the lead of the RAM by the boot program when the power is first switched on.
* Stage:	A stage indicates the load unit for one time. When one stage is finished this is called to load the next stage.
* Segment:	Link units are called segments; in other words, load modules. Each stage is made up of several segments.
* Library:	Program segments that are referred to commonly by each segment.
* Common Variable:	Variable group used commonly with respect to each segment.

Figure 2-2 SEGA's Dynamic Load Method

SEGA's dynamic load method is as follows:

1. When power is turned on, the boot ROM loads the loader program. The address of each segment, the size and the segments information to be loaded per each stage are included in the loader.
2. The loader first loads the libraries that are common among the segments; then the first stage of the segment is loaded after securing the area of common variables.
3. After loading, jump to the stage entrance .
4. After the end of each stage, the loader is called (user described). With this call, the control passes to the loader and the next stage is loaded.

The dynamic load operates according to the above procedure, but the following items are required to execute the process: a loader that has segment and stage information; information to solve for JMP between each segment; creation of common libraries; common variable information; common constant information, etc. The dynamic load linker (DLT) is what is used to create all of this information.

2.1.3 Procedure for Using the Dynamic Load Linker

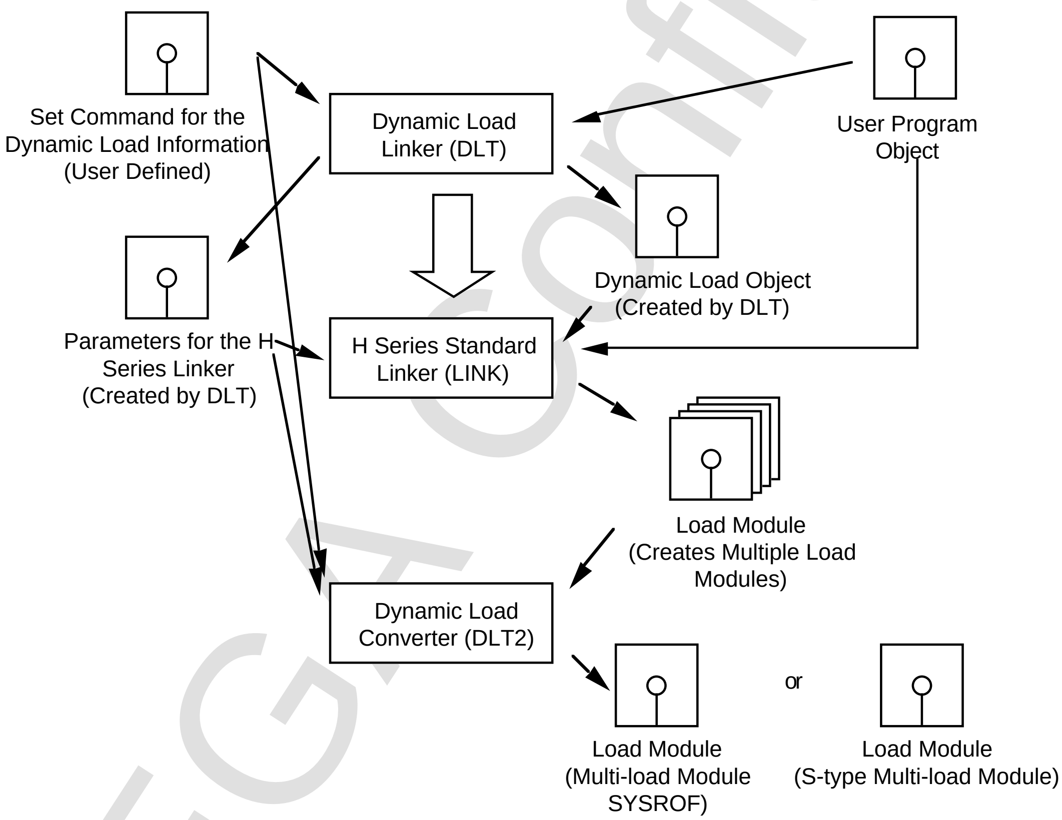


Figure 2-3 Procedure for Using the Dynamic Load Linker

As shown in Figure 2-3, the dynamic load information is created by the DLT, and the overall load module is created by inputting the output of the DLT into a standard H series linker.

Because the H series linker output load module creates one load module for each segment, DLT2 is the tool used to combine the many load modules into one file. Depending on the option selected, the DLT2 can create either SYSROF or S type load modules.

2.2 How to Use the Dynamic Load Linker

2.2.1 Using the DLT

(1) Function

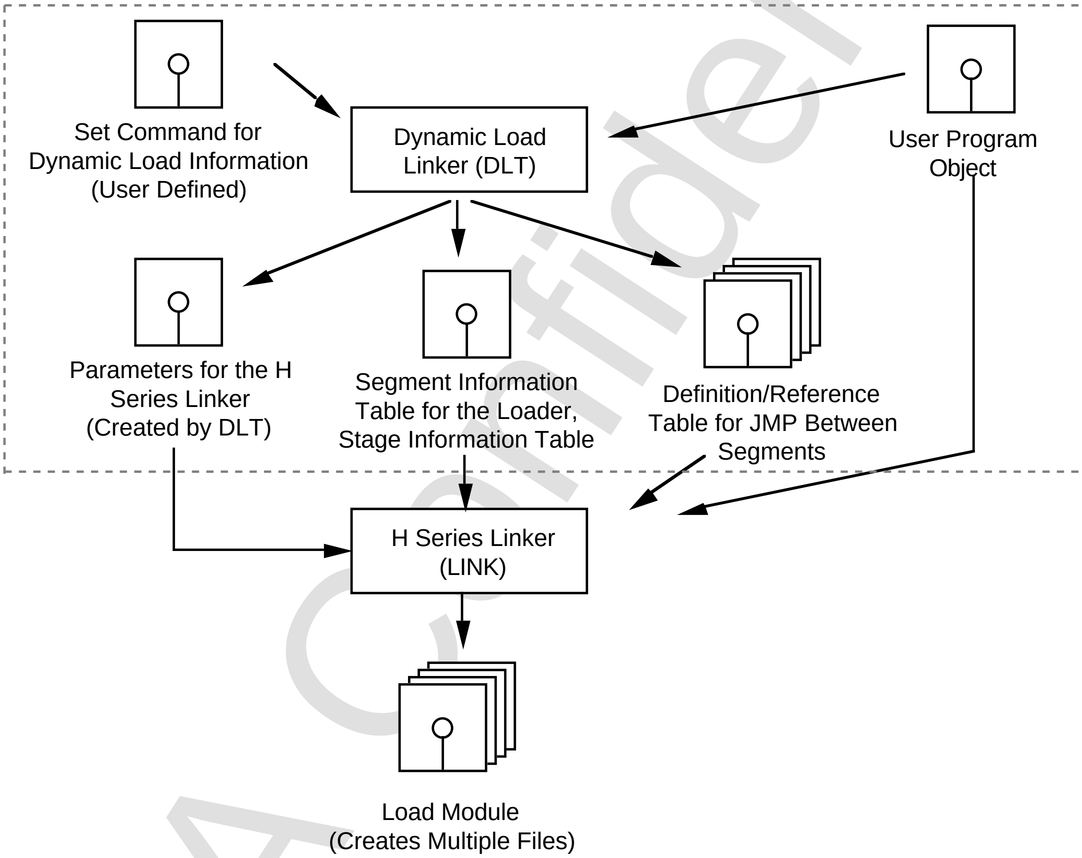


Figure 2-4 Using the DLT

As shown in Figure 2-4, the DLT inputs object files specified by the command file and its commands, and creates specified parameter file of the H-series standard linker as well as creating dynamic load information data.

A description of the dynamic load information can be found in "Dynamic Load Method."

(2) Command Designations

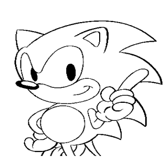
A command file like the one below is created and designated during DLT startup. Details concerning commands can be found in the command volume. Figure 2-6 shows an example of a command designation.

① : Specifies ROM area address and size.
≠ : Specifies RAM area address and size.
③ : Specifies the object file that configures the loader.
④ : Specifies the object file that configures the common variable area.
∞ : Specifies the object file that configures the common constant area.
± : Specifies the object file that configures the library area.
≤ : Specifies the object file that configures the segment.
≥ : Specifies the segment name that configures the stage.
⑨ : Specifies the function name that becomes the entrance to each stage.

```
ROM 1000000,4000

RAM 2000000,2000

loader segalode.obj

glvar glvar.obj
glconst ROM,glconst.obj
libseg clibseg.lib

segment seg00
in sa01.obj,sa02.obj
segment seg01
in ss1.obj,ss2.obj
segment seg02
in s21.obj,s22.obj,s23.obj
segment seg03
in s31.obj,s32.obj
segment seg04
in s41.obj,s42.obj,s43.obj
segment seg05
in sr51.obj,sr52.obj
stage 0
sin seg00,seg01
stage 1
sin seg00,seg02,seg03
stage 2
sin seg00,seg04,seg05

entry 0=st0ent,1=stlent
entry 2=st2ent
end
```

ROM address designation①
RAM address designation②
Loader designation③
Common variable designation④
Common constant designation⑤
Library designation⑥
Segment 0 composition module designation⑦
Segment composition 1 module designation
Segment composition 2 module designation
Segment composition 3 module designation
Segment composition 4 module designation
Segment composition 5 module designation
Stage 0 composition segment designation⑧
Stage 1 composition segment designation
Stage 2 composition segment designation⑨
Stage entrance function name
Specifies command file end

Figure 2-6 Example of DLT Commands



2.2.2 Using DLT2

(1) Function

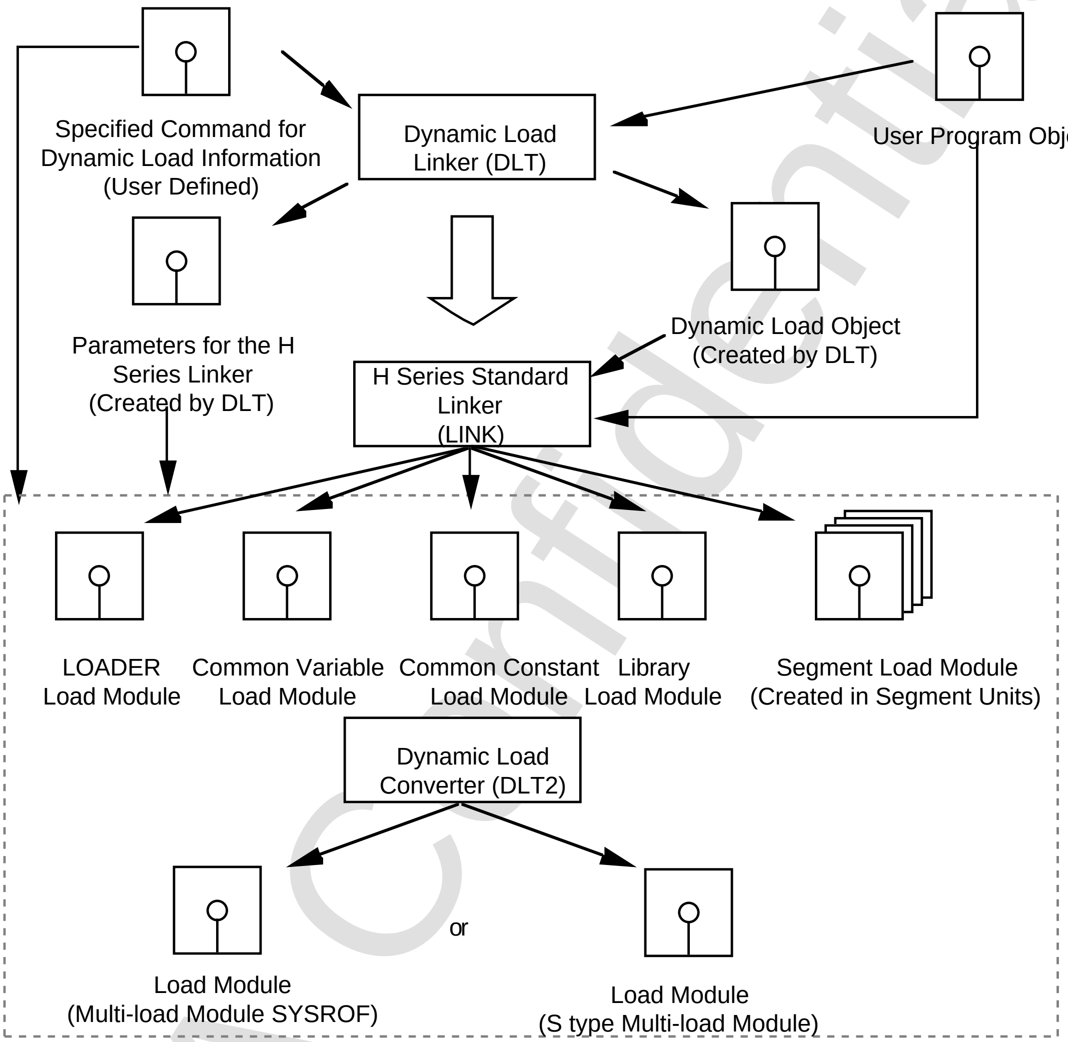


Figure 2-7 Using DLT2

As shown in Figure 2-7, the DLT2 is used to combine the multiple load modules that are output from the H series linker into one file. The output file formats of the load module are SYSROF and S-type format.

(2) Command

By designating the command file specified by the DLT, DLT2 will analyze this command file and automatically select the input file and input it. Startup method and command operand are shown below.

```
DLT2 <Name of the command file specified by DLT>, <Output file
name>, [<Output load module format>]<cr>

<Name of the command file indicated by DLT> :  Automatically selects
the input file from this information.
<Output file name> :  Name of the output file.
<Output load module format> :  If designated by an “S”, an S type
format load module will be output.
```

2.2.3 Limits on User Program Coding

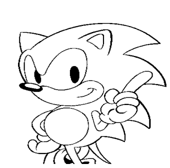
As shown in 2.1.2 of SEGA’s dynamic load method, the following limits must be followed to realize the dynamic load when coding into the user program.

(1) The global variable (common variable) only compiles (assembles) the declaration portion. In other words, it is created and compiled as only a source file that does not include the execution command.
—Specifies this object file to the DLT glvar command.

(2) The global constant (common constant) only compiles (assembles) the declaration portion. In other words, it is compiled as only a source file that does not include the execution command.
—Specifies this object file to the DLT glconst command.

(3) Local variables are all stored in the stack as automatic variable declarations. In the assembly description, area is reserved in either the stack or in the RAM absolute address.
—With static declaration, DLT is error.

(4) When the stage ends, a special function (supplied by SEGA) is called. The next stage is loaded through this function call.



2.2.4 Limits on User Programs (Coding Example)

(1) The global constant (global constant) only compiles (assembles) the declaration portion. In other words, it is created and compiled as only a source file that does not include the execution command. Specifies this object file name to the DLT glvar (glconst) command.

Example 1: C language

Common variable declaration (declaration only file)

```
int ab[10];
struct ss {
 int mas;
 char *cptr;
 } stvar;
char cc;
int *ptr;
struct  ss ssp
;
```

* This file is complied separately (glb.obj)

Common variable reference

```
extern int ab[10];
extern struct ss{
 int mas;
 char *cptr;
 } stvar;
extern char cc;
extern int *ptr;
extern struct ss ssp;

int sub(int a,char cc
)
 {
 a += 4;
```

DLT command

```
glvar
  glb.obj
```

Example 2: ASM language

Common variable declaration (Only the declaration file)

```
ab:   res.W 10
stvar_mas:  res,w
1
stvar_cptr: res.w
1
cc:   res.b 1
ptr:  res.w 1
ssp_mas:  res.w
ssp_cptr:  res.w 1
```

* This file is complied separately (glb.obj)

Common variable reference

```
extern ab
extern stvar_mas
extern stvar_cptr
extern cc
extern ptr
extern ssp_mas
extern ssp_cptr



    add #4, r1
```

DLT command

```
glvar
  glb.obj
```

(2) Local variables are all stored in the stack as automatic variable declarations. In the assembly description, area is reserved in either the stack or in the RAM absolute address.

Example 1: C language

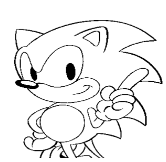
Automatic variable declaration

```
char exa (int a) ;
{
 int cnt;
 char ary[5];
 int *ptri;

 cnt += 1;
   :
```

Example 2: ASM language

Assign local variable to an absolute address

```
cnt: .equ h' 40000
ary: .equ h' 40004
ptri: .equ h' 4000a

 mov.w cnt, r1
 add.w #1,r1
 mov.w r1,cnt
   :
```

(3) Relationship between each area, stage and DLT commands
Commands (shown in Figure 2-5,) specifies the information for each area and stage. ROM indicates a command.

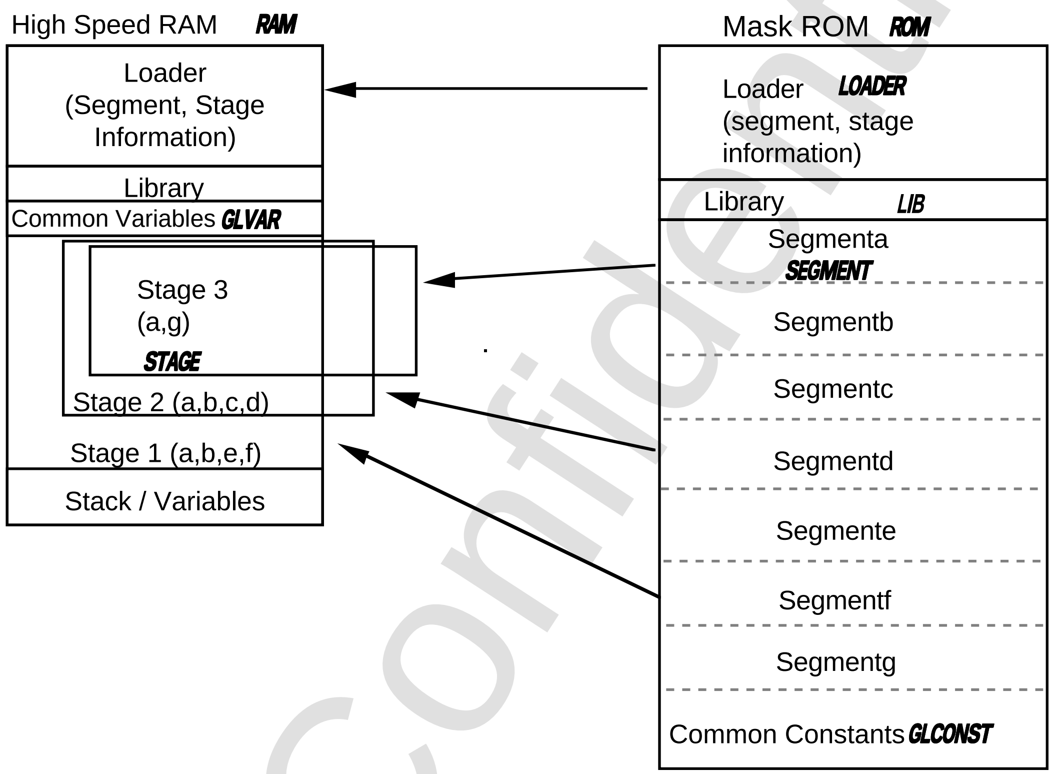


Figure 2-5 Relationship between each area, stage and DLT commands

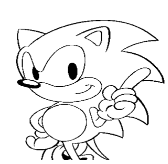
3.0 Dynamic Load Method

3.1 Overview

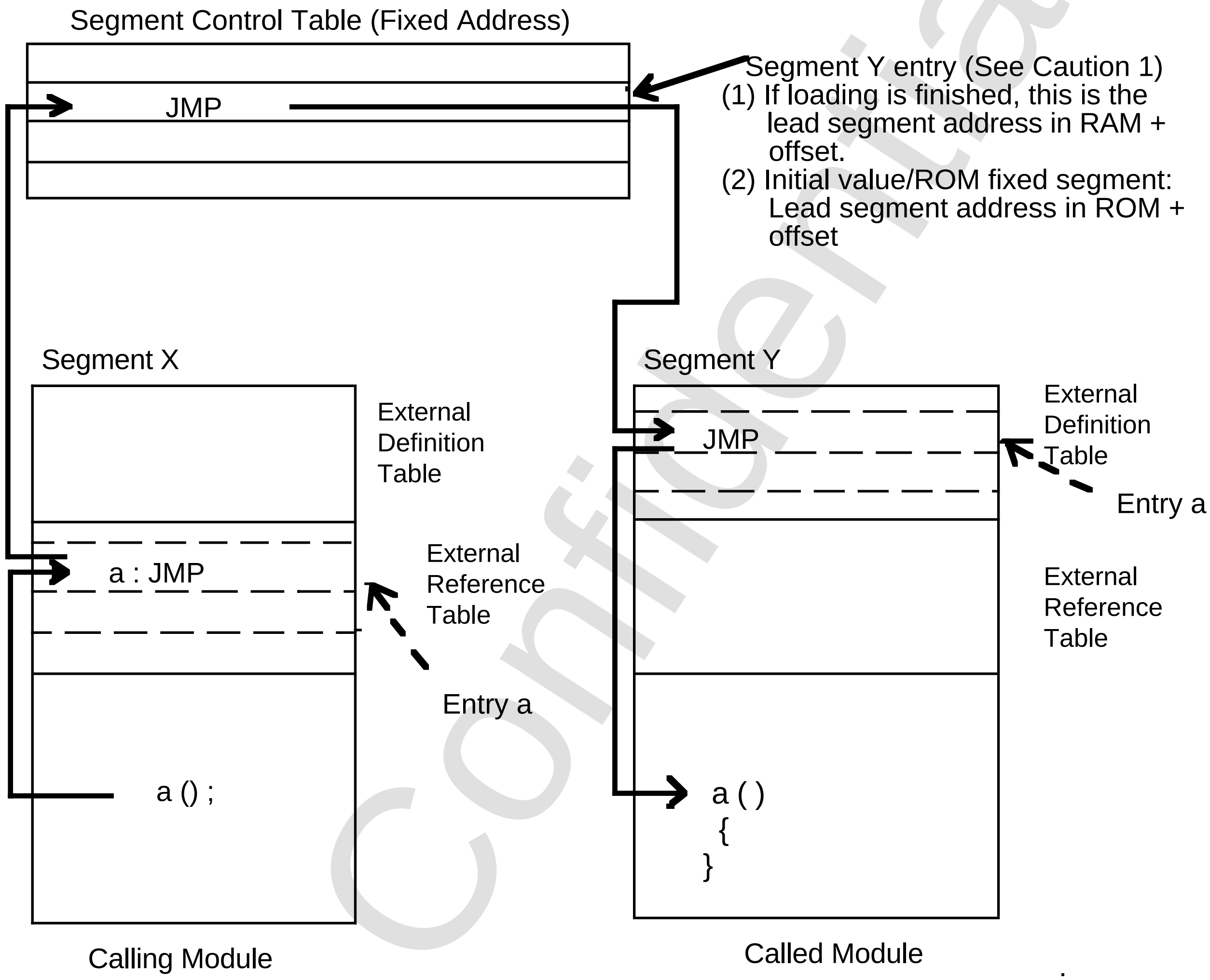
Refer to Figure 2-2 in the SEGA Dynamic Load Method for the basic format. There were execution method problems in the following areas.

(1) Referring to functions between segments (JMP between segments).
(2) Control method for each segment load while loading the stage with the loader (segment load).
(3) C library control method.
(4) Debugging method with the emulator.

The methods used for solving these problems are examined on the following pages.



3.2 Method for JMP Between Segments



Caution: The initial value is (2) ROM address. Each time the stage is initialized (loader), (1) is set.

Figure 3-1 Segment JMP Method

3.2.1 Details of the JMP Method between Segments

3.2.1.1 Between Segment Call Format for Positioning Independent Code

In positioning independent code (PIC), each segment can be loaded into an arbitrary address, but a mechanism for calling segments in the middle is needed. Below, this tool shows the premise to this format.

- Premise Conditions
  Listed below are the premise conditions for this proposal.

1. Call between segments must be described in exactly the same format as in C language.
2. Only segment dynamic load will be able to execute at the lead of each stage. Dynamic load and purge in each stage is not being considered.

- Support Format

1. The tables needed to call up the external reference, external definition symbols and library names when C is executed are created using this tool. The tool creates an external reference table and external definition table for each segment to make an overall system segment control table. By using this type of table, items between segments can be called and executed. Table entries are run in code rather than data to achieve high call process speed.

Example:

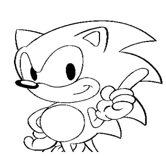
seg_tbl :	Absolute address of the segment control table.
entry_size:	Size of one entry in the segment control table.
seg_addr_arom:	Absolute address of segments in group a (in-ROM address)
seg_addr_a:	Absolute address of segments in group a (Determined at loading)
seg_addr_asize:	Size of the segments in group a.
seg_no_a:	Segment number that belongs to group a.
sym_no_a:	Number of a in segment.
sym_size:	Entry size of the external definition table.

(a) External Reference Table
Table used to call up external segment functions that are referred to from the segment. One entry is made for each external reference. To call up a function external of a segment within a segment, control is transferred to the external of the segment through this table code. This table links each segment. The contents of this entry are shown below.

```
                                        ;     External name reference
a:                                      ;     Label called from inside the segment.
      MOV. L #seg_tbl+entry_size*seg_no_a, R1
                                        ;     Entry address of the segment control table
      MOV. L #sym_no_a*sym_size+4, R0
                                        ;     Symbol number offset calculation
      JMP    R1                         ;     Jump to the segment control table
      MOV. L #seg_addr_arom,R1  ;  (delay slot)
                                        ;     Segment control table lead command
```

(b) External Definition Table
Table used to execute functions that are referenced from the external of the segment. One entry is made for each external definition. With respect to the reference from the external of a segment, the control is transferred to the function entity within the segment through this table. This table links the head of each segment.

```
      MOV    #a–*, R0          ;      Address a acquisition
                                      (Finished execution with segment control table
DELAY SLOT)
      JMP    @ (R0, PC)        ;      Move control to entity a
      N O P
```



(c) Segment Control Table

This table is used to link (a) and (b) above. There is only one per system and it is divided up in fixed addresses. The emulator finds the base address of each segment by referring to this table and executes C level symbol debugging. Contents of this entry are shown below.

```
(Before the segment is loaded)
MOV. L #seg_addr_arom, R1
                                  ;    Loads absolute ROM address of the segment
                      (Executes the external reference table with the DELAY SLOT)
ADD    R0,R1                      ;    Address acquisition of external definition table
JMP           @R1                 ;    Jump to external definition table
MOV    #a–*, R0                   ;    Get the address of a
                                       (External reference table lead command)
.DATA.W  0                        ;    4 byte boundary adjustment
data.l  #seg_addr_arom
data.l  #seg_addr_asize
```

During the segment load, it is converted to the code below and executed again. From here on, control is moved directly to the call up destination. When the segment is purged, the contents are returned to the pre-load state.

```
(After the segment is loaded)
MOV. L #seg_addr_arom, R1
                                  ;    Loads absolute in-ROM address of the segment
                      (Executes the external reference table with the DELAY SLOT)
ADD    R0,R1                      ;    Address acquisition of external definition table
JMP    @R1                        ;    Jump to external definition table
MOV    #a–*, R0                   ;    Get the address of a
                                       (External reference table lead command)
.DATA.W  0                        ;    4 byte boundary adjustment
data.l  #seg_addr_a (Is ROM address when not loaded)
data.l  #seg_addr_asize
```

The route to create and link the above table to call up external functions is shown below.

Segment called up C Object JSR   Calling of the external reference table entry
                                       NOP

External Reference Table   MOV   Loading of the segment control table address
                                       MOV   Load symbol offset from inside the segment
                                       JMP   Jump to the segment control table
                                       MOV Load the segment absolute address

Segment Control Table (When already loaded)   ADD   Calculate external definition table entry address
                                       JMP   Jump to external definition table entry
                                       MOV   Load absolute segment address

Called up segment, external def. tbl.
                                       JMP   Jump to entity
                                       NOP

C Program   a: Entity code

3.3 Loader

The loader is not included in this tool. The user must create it. A hypothetical process format is shown below.

3.3.1 Configuration

The loader is made up of the segment control table, stage information, each stage entry address, the loader program and common variable areas. The segment control table contents are explained above.

(a) Segment control table
Contents of table are shown below.

(b) Stage Number
Currently executing stage number. Set by loader.

(c) Individual stage entry address
Individual stage entrance function address

(d) Loader program

(e) Common variable area
This is the area for common variables used between segments. Only created from C source program global variable declaration (no initial value)

Loader Segment Contents

Loader Segment Contents
Segment Control Table
Stage Information
Individual Stage Entry Address
Loader Program
Common Variable Area

3.3.2 Function of the Loader Program

1. Initial Loader for individual Stage Units
   Searches all segment tables and executes the processing below
   (a)Purges old stage information
   Returns all table information to initial values
   (b)Loads New Stage Segments
   Loads new stage number segments all at one time and rewrites the information in the segment control table. Contents that are rewritten are shown below.

Segment Control Table Contents (1 entry)

Rewrites inside [box]

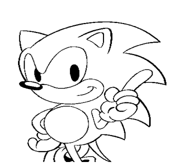
(Before the segment is loaded)

```
MOV.L #seg_addr_arom, R1
                        ;   Loads absolute ROM address of the segment
ADD     R0,R1           ;   Address acquisition of External definition table
JMP     @R1             ;   Jump to external definition table
MOV     #a-*, R0        ;   Get the address of a
                            Jump destination(external reference table) lead command
data.w 0
data.w  #seg_addr_arom
data.w  #seg_addr_asize
```

When loaded into a segment, it is converted to the code below. From here on, control is moved directly to the call up destination. When the segment is purged, contents are returned to the pre-load state.

(After the segment is loaded)

```
MOV.L #seg_addr_a, R1
                        ;   Loads absolute ROM address of the segment
ADD     R0,R1           ;   Address acquisition of external definition table
JMP @R1                 ;   Jump to external definition table
MOV     #a-*, R0        ;   Get the address of a
data.w 0
data.w  #seg_addr_a
data.w  #seg_addr_asize
```

3.4 Handling of the Between Segment Common Library

3.4.1 Overview

By linking in segment units, common routines and C execution libraries are gathered in one location, these segment must be referenced. This method is described below.

3.4.2 Librarian

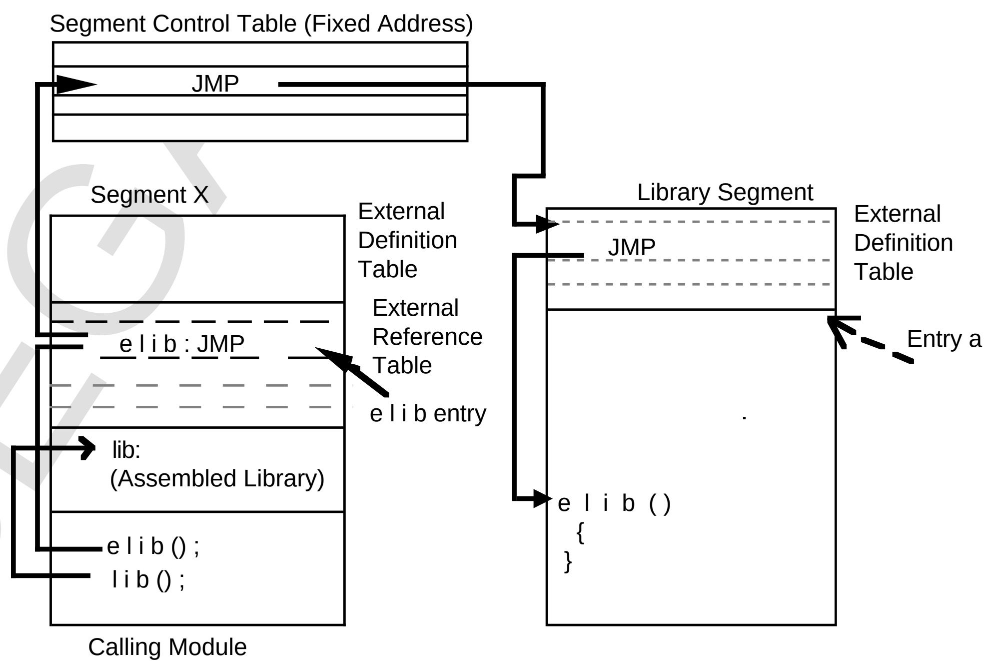
Uses the librarian made by Hitachi (existing) to create libraries. When linking libraries created by librarian and designated as a library file, then only necessary modules are included.

3.4.3 Types of Libraries

If all of the libraries are placed into one segment, in between all segments will be JMP and the speed of calculation processing (particularly with part of the C library during execution) will cause some problems. Because of this, as a precondition common routines that rely on speed
are divided into two types: libraries to JMP between segments and libraries included within the segment. In C execution, libraries are divided into two through the Hitachi.

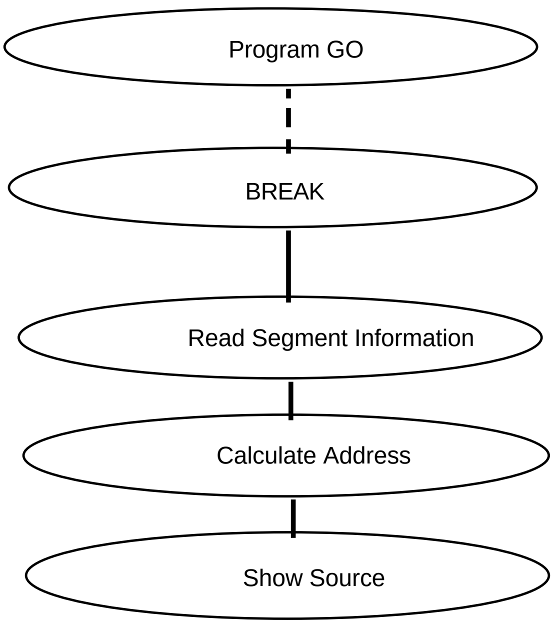
1. Libraries for Segment Inclusion
   Library included inside the segment. C libraries use a lot of integer multiplication and division calculations so small library C execution file can be prepared from standard C execution files (provided by Hitachi). Others can be created by the librarian. The library can be designated when linking the segments and included into all of the segments.

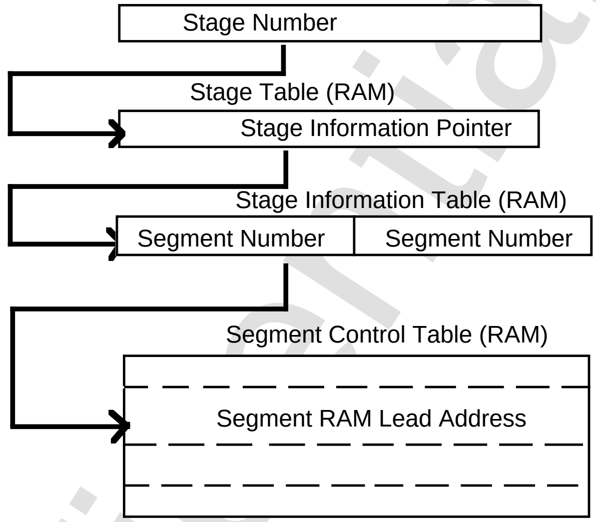
2. Library Segment Library
   Library group that should be included in the library segment. C library routines other than those described above are taken and prepared from the standard C execution library (provided by Hitachi). If others are needed, they are prepared by the librarian. This tool searches all segments and extracts only the necessary routines, and outputs library segment. This segment is made to reside in the RAM.



3.5 Source Coding Debug

3.5.1 Processing Method





If the break address is in RAM (designated by an additional command), then it searches the stage pointer table from the stage number, and acquires the corresponding segment number. The segment address is taken from the segment control table in RAM.
ROM addresses are taken from the segment control table in ROM.
The source-bound address=the original address in the debug information – the address in ROM+the address in RAM.

Source can be shown with the above method. This processing is included in the emulator.

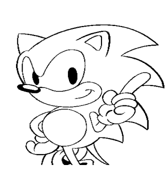
3.5.2 Additional Functions

To enable source debugging with the emulator, the following functions were added to the E7000 small evaluation board.

(1) Designation command for RAM ROM area to correspond to dynamic load.
(2) Load control table address designation command.
(3) Loading of the multi-load module. (Added the LOAD command function)

3.5.3 Other

(1) If a segment load occurs while C source trace information is being displayed, it is OK if the C source before load was incorrect.



4.0 Process Format and Command Specifications

This system is created by improving the existing H series linker. See the second edition for processing methods and command specifications.

5.0 Development Method

5.1 Development Machine

(1) IBM-PC
Device: IBM-PC (However, a PC98 can be used if testing is done on an IBM-PC)
Language: MS-C (Ver.6.0) and MS-ASM
OS: MS-DOS V3.3 or later
Testing will be done on MS-DOS V5.0.

(2) SPARC
Device: SPARC
Language: C
OS: UNIX V4.0.3 or later

(3) HP9000/700
Device: SPARC
Language: C
OS: HP-UX V8.0

5.2 Document

Create and deliver the following documents.

(1) Function Design Document: Create new
This design document

(2) Internal Processing Specification (H series linker flow areas are not described)
Describes the internal processing method. (Will not create a flow chart equivalent)

(3) Manual: Create a user's manual. However, it will be simple. (Less than 50 pages). Will have both Japanese and English versions.

6. 0 Other

Development Schedule, Guarantees and additional functions will be based on the contract.