

Code control using the GNU linker

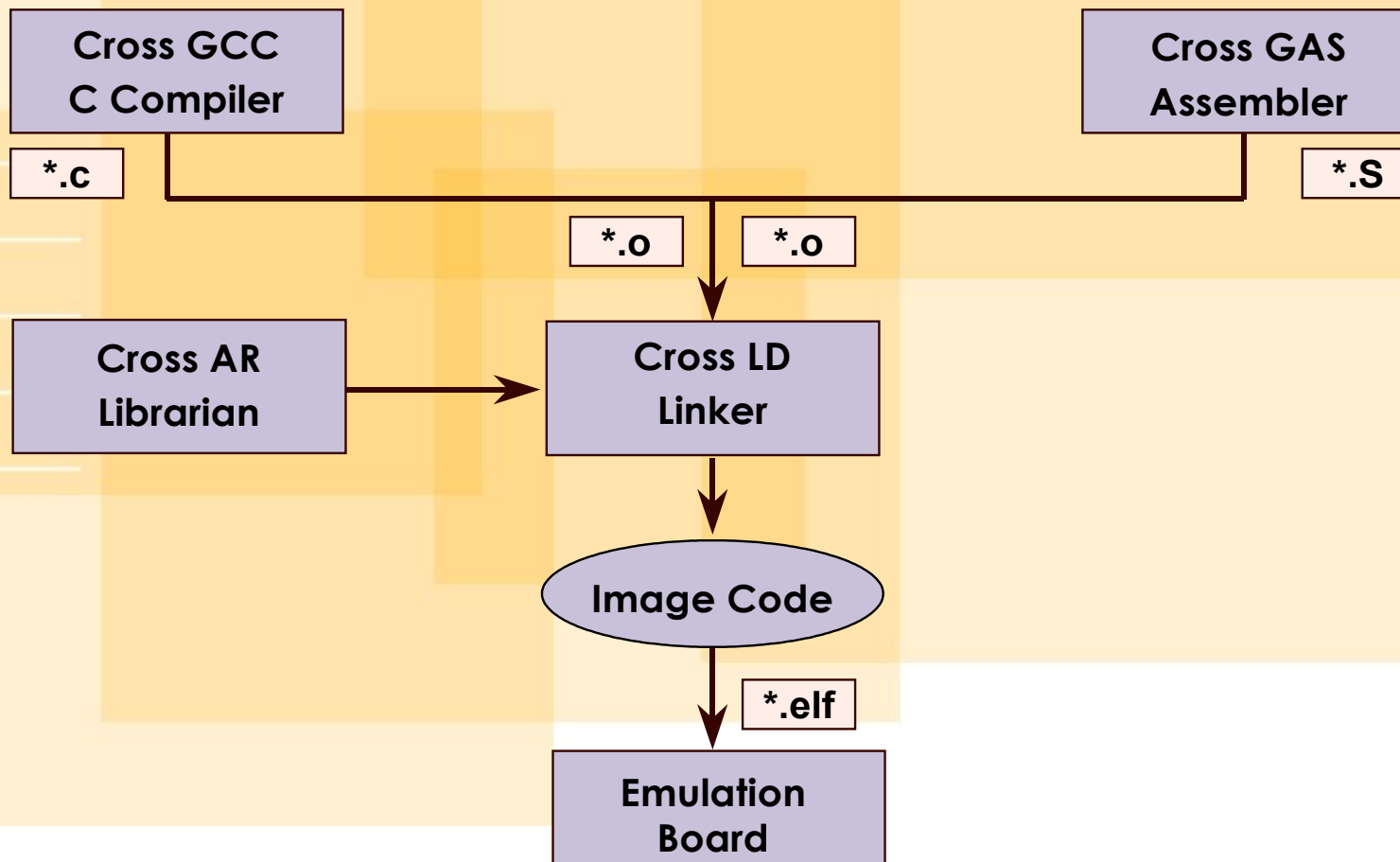
Agenda

- **GNU Linker Introduction.**
- **Typical structure of a single bootable image.**
- **GNU linker default settings.**
- **Creating a standalone image.**
- **Linker script handling.**
- **Packing the output in binary format.**

GNU Linker Introduction

- **GNU linker (or GNU ld) is the GNU project's implementation of the Unix command ld.**
- **Part of the GNU Binary utilities (binutils).**
- **GNU linker is a free software, distributed under the terms of GPL.**

GNU Linker Introduction



GNU Linker Introduction

- **<Arch>-elf-ld**
 - control the positioning and attributes of object code
 - control details using a linker-script (.ld file)
 - numerous interacting options (flexibility)

Other GNU Tools

- **<Arch>-elf-objcopy**
 - converting file format (srec, ihex, binary)
- **<Arch>-elf-objdump**
 - check an executable's layout
- **<Arch>-elf-readelf**
 - Display information about the content of an ELF format file.

A Bootable Image

Typical structure

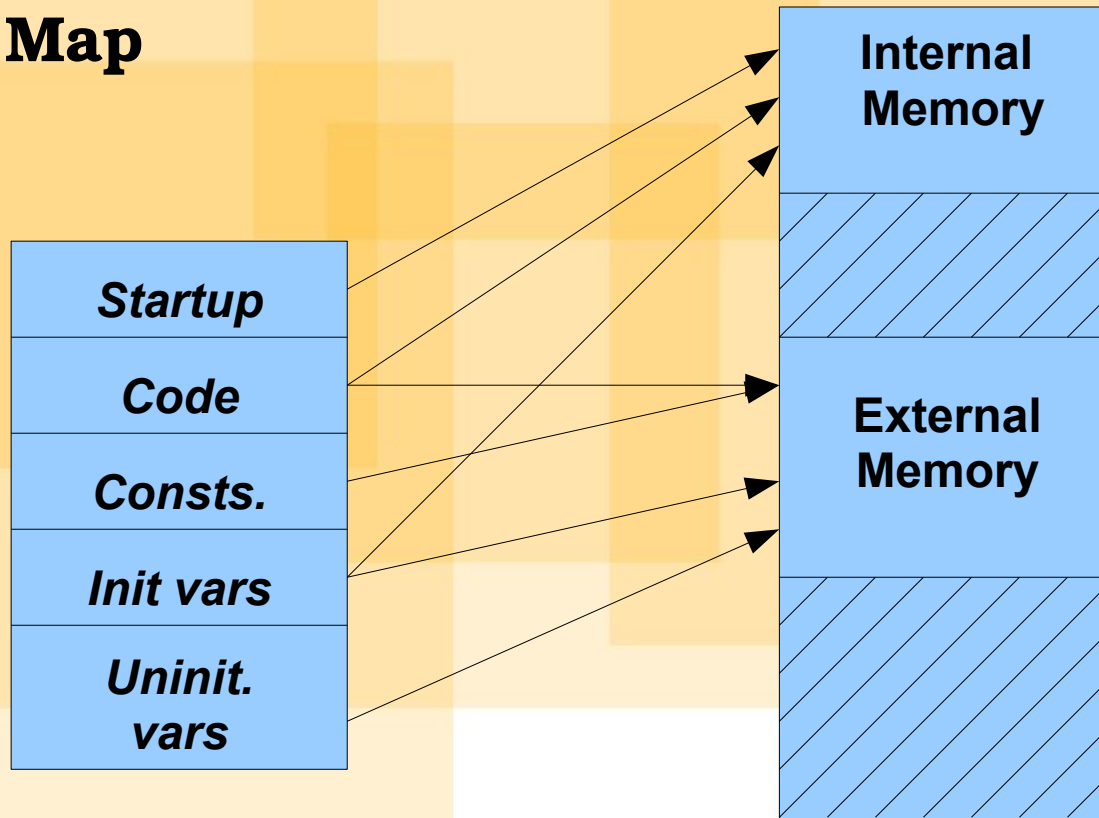
- **Startup**
 - **Some hardware and C run-time initialize code.**
 - **MUST be located at a specific (Reset vector).**
- **Application code (Instructions)**
- **Constant data**
 - **Constants and binary data**

A Bootable Image

- **Initialized variables**
 - **Initial values – must be loaded at boot.**
- **Uninitialized variables**

A Bootable Image

Memory Map



GNU linker defaults

- **The GNU Linker (ld) uses a built-in default linker script.**
- **Use this script as the basis for building your own linker script.**
- **To view this script:**
 - **<Arch>-elf-ld --verbose**

GNU linker defaults

```

Applications  Actions  [Icons]  raviv@tregolnx1:~/arm_toolchain/gnuarm-3.4.3/bin
File  Edit  View  Terminal  Tabs  Help
[raviv@tregolnx1 bin]$ ./arm-elf-ld --verbose
GNU ld version 2.15
Supported emulations:
  armelf
using internal linker script:
=====
/* Script for -z combreloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf32-littlearm", "elf32-bigarm",
              "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SEARCH_DIR("/srv/shared/gnuarm/gnuarm-3.4.3/arm-elf/lib");
/* Do we need any of these for elf?
  __DYNAMIC = 0; */
SECTIONS
{
  /* Read-only sections, merged into text segment: */
  PROVIDE (__executable_start = 0x8000); . = 0x8000;
  .interp      : { *(.interp) }
  .hash        : { *(.hash) }
  .dynsym      : { *(.dynsym) }
  .dynstr      : { *(.dynstr) }
  .gnu.version : { *(.gnu.version)}
  .gnu.version_d : { *(.gnu.version_d) }
  .gnu.version_r : { *(.gnu.version_r) }
  .rel.dyn     :
  {
    *(.rel.init)
    *(.rel.text .rel.text.* .rel.gnu.linkonce.t.*)
    *(.rel.fini)
    *(.rel.rodata .rel.rodata.* .rel.gnu.linkonce.r.*)
    *(.rel.data .rel.data.* .rel.gnu.linkonce.d.*)
    *(.rel.tdata .rel.tdata.* .rel.gnu.linkonce.td.*)
    *(.rel.tbss .rel.tbss.* .rel.gnu.linkonce.tb.*)
    *(.rel.ctors)
    *(.rel.dtors)
    *(.rel.got)
    *(.rel.bss .rel.bss.* .rel.gnu.linkonce.b.*)
  }
  .rela.dyn    :
  {
    *(.rela.init)
    *(.rela.text .rela.text.* .rela.gnu.linkonce.t.*)
    *(.rela.fini)
  }
}

```

GNU linker defaults

```

raviv@tregolnx1:~/arm_toolchain/gnuarm-3.4.3/bin
File Edit View Terminal Tabs Help
.rel.plt      : { *(.rel.plt) }
.init        :
{
  KEEP (*(init))
} =0
.plt         : { *(.plt) }
.text        :
{
  *(.text .stub .text.* .gnu.linkonce.t.*)
  /* .gnu.warning sections are handled specially by elf32.em.  */
  *(.gnu.warning)
  *(.glue_7t) *(.glue_7)
} =0
.fini        :
{
  KEEP (*(fini))
} =0
PROVIDE (__etext = .);
PROVIDE (_etext = .);
PROVIDE (etext = .);
.rodata      : { *(.rodata .rodata.* .gnu.linkonce.r.*) }
.rodata1     : { *(.rodata1) }
.eh_frame_hdr : { *(.eh_frame_hdr) }
/* Adjust the address for the data segment.  We want to adjust up to
the same address within the page on the next page up.  */
. = ALIGN(256) + (. & (256 - 1));
/* Ensure the __preinit_array_start label is properly aligned.  We
could instead move the label definition inside the section, but
the linker would then create the section even if it turns out to
be empty, which isn't pretty.  */
. = ALIGN(32 / 8);
PROVIDE (__preinit_array_start = .);
.preinit_array : { *(.preinit_array) }
PROVIDE (__preinit_array_end = .);
PROVIDE (__init_array_start = .);
.init_array   : { *(.init_array) }
PROVIDE (__init_array_end = .);
PROVIDE (__fini_array_start = .);
.fini_array   : { *(.fini_array) }
PROVIDE (__fini_array_end = .);
.data        :
{
  __data_start = . ;
  *(.data .data.* .gnu.linkonce.d.*)
  SORT(CONSTRUCTORS)
}

```

GNU linker defaults

```

Xming - 192.168.1.8
Applications Actions
raviv@tregolnx1:~/arm_toolchain/
File Edit View Terminal Tabs Help
.got : { *(.got.plt) *(.got) }
_edata = .;
PROVIDE (edata = .);
__bss_start = .;
__bss_start__ = .;
.bss :
{
*(.dynbss)
*(.bss .bss.* .gnu.linkonce.b.*)
*(COMMON)
/* Align here to ensure that the .bss section occupies space up to
_end. Align after .bss to ensure correct alignment even if the
.bss section disappears because there are no input sections. */
. = ALIGN(32 / 8);
}
. = ALIGN(32 / 8);
_end = .;
__bss_end__ = . ; __bss_end__ = . ; __end__ = . ;
PROVIDE (end = .);
/* Stabs debugging sections. */
.stab 0 : { *(.stab) }
.stabstr 0 : { *(.stabstr) }
.stab.excl 0 : { *(.stab.excl) }
.stab.exclstr 0 : { *(.stab.exclstr) }
.stab.index 0 : { *(.stab.index) }
.stab.indexstr 0 : { *(.stab.indexstr) }
.comment 0 : { *(.comment) }
/* DWARF debug sections.
Symbols in the DWARF debugging sections are relative to the beginning
of the section so we begin them at 0. */
/* DWARF 1 */
.debug 0 : { *(.debug) }
.line 0 : { *(.line) }
/* GNU DWARF 1 extensions */
.debug_srcinfo 0 : { *(.debug_srcinfo) }
.debug_sfnames 0 : { *(.debug_sfnames) }
/* DWARF 1.1 and DWARF 2 */
.debug_aranges 0 : { *(.debug_aranges) }
.debug_pubnames 0 : { *(.debug_pubnames) }
/* DWARF 2 */
.debug_info 0 : { *(.debug_info .gnu.linkonce.wi.*) }
.debug_abbrev 0 : { *(.debug_abbrev) }
.debug_line 0 : { *(.debug_line) }
.debug_frame 0 : { *(.debug_frame) }

```


GNU linker defaults

- **Simple program outputs:**

```
raviv@tregolnx1:~/linker_lect/example1
File Edit View Terminal Tabs Help

[raviv@tregolnx1 example1]$ arm-elf-readelf -h hello
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 61 00 00 00 00 00 00 00 00
  Class:                               ELF32
  Data:                                  2's complement, little endian
  Version:                               1 (current)
  OS/ABI:                                 ARM
  ABI Version:                           0
  Type:                                  EXEC (Executable file)
  Machine:                                ARM
  Version:                                0x1
  Entry point address:                   0x8110
  Start of program headers:              52 (bytes into file)
  Start of section headers:              210440 (bytes into file)
  Flags:                                  0x2, has entry point, GNU EABI
  Size of this header:                    52 (bytes)
  Size of program headers:                32 (bytes)
  Number of program headers:              2
  Size of section headers:                40 (bytes)
  Number of section headers:              24
  Section header string table index:     21
```

GNU linker defaults

- **Can this image boot standalone on my machine?**
- **Need to create a standalone image suiting my machine requirements:**
 - **Customize startup**
 - **Instructions (.text) organizing**
 - **Data (.data , .bss , ...) organizing**

Creating a standalone image

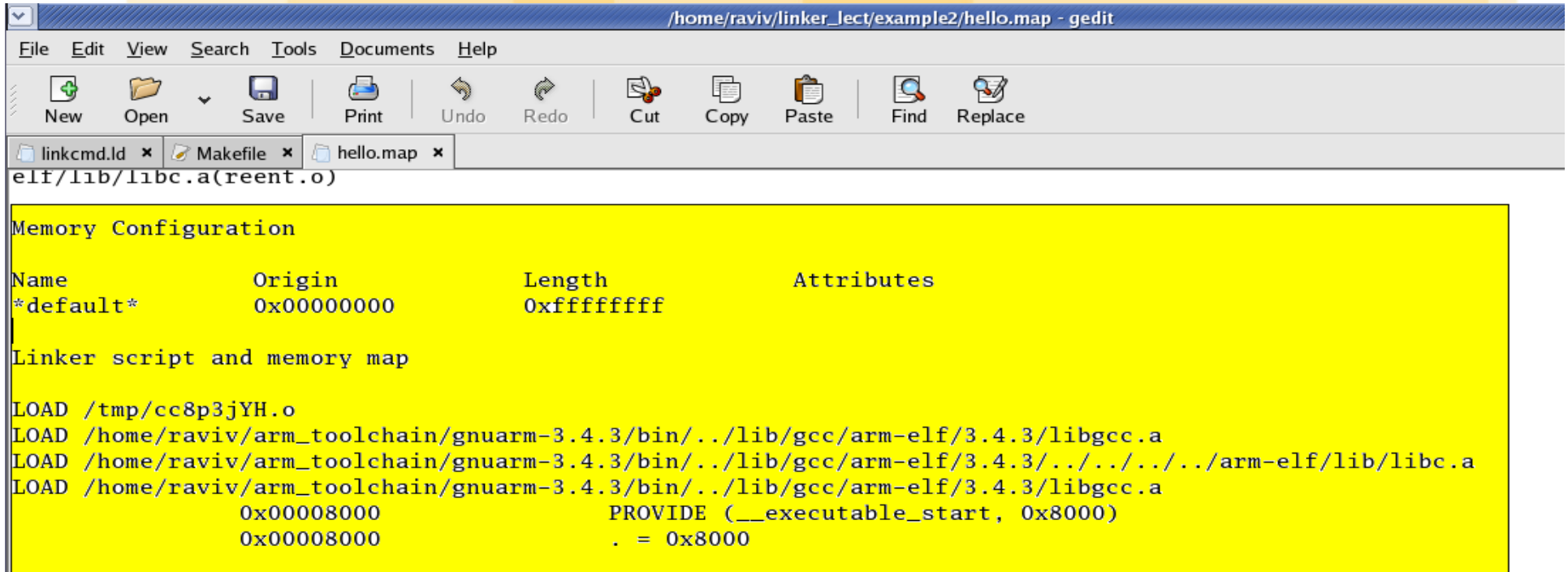
- **Link without default (tool-chain) start files.**
 - **LDFLAGS += -nostartfiles**
- **Provide your custom linker command file (.ld)**
 - **LDFLAGS += -T linkcmd.ld**
- **Setup the code entry point.**
- **Provide the linker with the machine memory segments .**
 - **Special consideration for boot memory segment.**
- **Provide your custom start file (.asm)**

Creating a standalone image

- **Step 1 - Link without default (tool-chain) start files:**

```
raviv@tregolnx1:~/linker_lect/example2
File Edit View Terminal Tabs Help
[raviv@tregolnx1 example2]$ make
arm-elf-gcc -nostartfiles -Wl,-Map -Wl,hello.map -o hello hello.c
/home/raviv/arm_toolchain/gnuarm-3.4.3/bin/../lib/gcc/arm-elf/3.4.3/../../../../arm-elf/bin/ld: warning: can
not find entry symbol _start; defaulting to 00008000
[raviv@tregolnx1 example2]$
```

Creating a standalone image



The screenshot shows a gedit editor window with the title bar "/home/raviv/linker_lect/example2/hello.map - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The active window is "hello.map" and the content is as follows:

```
elf/lib/libc.a(reent.o)

Memory Configuration

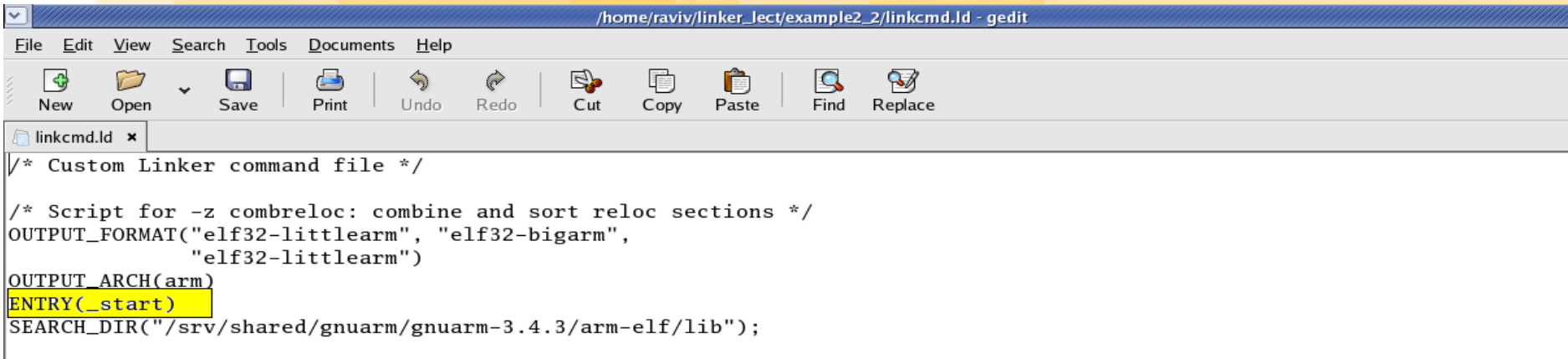
Name                Origin              Length              Attributes
*default*          0x00000000          0xffffffff

Linker script and memory map

LOAD /tmp/cc8p3jYH.o
LOAD /home/raviv/arm_toolchain/gnuarm-3.4.3/bin/../../lib/gcc/arm-elf/3.4.3/libgcc.a
LOAD /home/raviv/arm_toolchain/gnuarm-3.4.3/bin/../../lib/gcc/arm-elf/3.4.3/../../../../arm-elf/lib/libc.a
LOAD /home/raviv/arm_toolchain/gnuarm-3.4.3/bin/../../lib/gcc/arm-elf/3.4.3/libgcc.a
                                0x00008000          PROVIDE (__executable_start, 0x8000)
                                0x00008000          . = 0x8000
```

Creating a standalone image

- **Step 2 - Setup the code entry point**

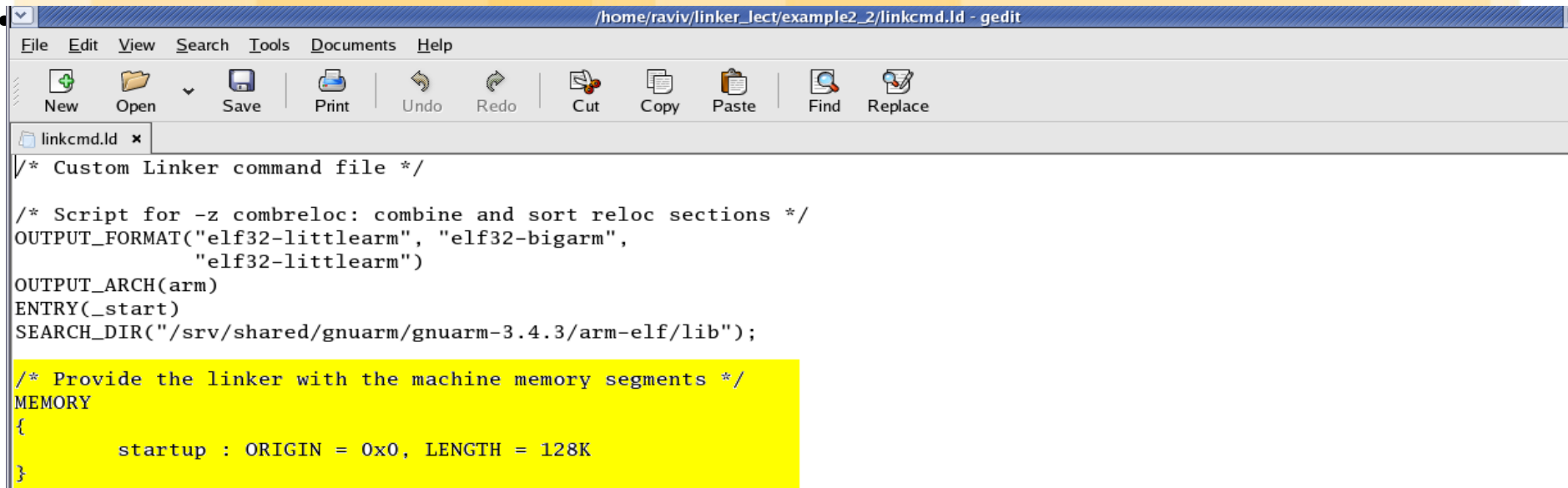


The screenshot shows a text editor window titled "/home/raviv/linker_lect/example2_2/linkcmd.ld - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for New, Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The main text area shows the following linker command file content:

```
/* Custom Linker command file */  
  
/* Script for -z combreloc: combine and sort reloc sections */  
OUTPUT_FORMAT("elf32-littlearm", "elf32-bigarm",  
              "elf32-littlearm")  
OUTPUT_ARCH(arm)  
ENTRY(_start)  
SEARCH_DIR("/srv/shared/gnuarm/gnuarm-3.4.3/arm-elf/lib");
```

Creating a standalone image

- **Step 3 - Provide the linker with the machine memory segments**

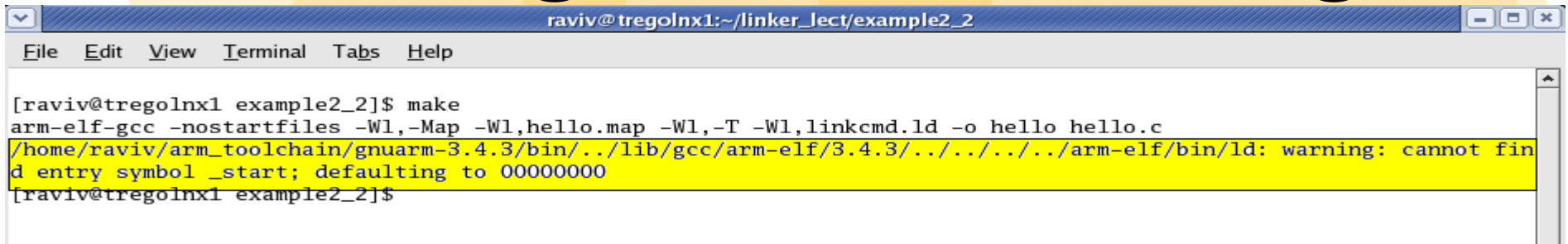


```
/* Custom Linker command file */

/* Script for -z combreloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf32-littlearm", "elf32-bigarm",
              "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SEARCH_DIR("/srv/shared/gnuarm/gnuarm-3.4.3/arm-elf/lib");

/* Provide the linker with the machine memory segments */
MEMORY
{
    startup : ORIGIN = 0x0, LENGTH = 128K
}
```

Creating a standalone image



```
raviv@tregolnx1:~/linker_lect/example2_2
File Edit View Terminal Tabs Help
[raviv@tregolnx1 example2_2]$ make
arm-elf-gcc -nostartfiles -Wl,-Map -Wl,hello.map -Wl,-T -Wl,linkcmd.ld -o hello hello.c
/home/raviv/arm_toolchain/gnuarm-3.4.3/bin/../lib/gcc/arm-elf/3.4.3/../../../../arm-elf/bin/ld: warning: cannot find entry symbol _start; defaulting to 00000000
[raviv@tregolnx1 example2_2]$
```

Creating a standalone image

- **Step 4 - Provide your custom start file**

```

/home/raviv/linker_lect/example2_3/hello.map - gedit
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy Paste Find Replace
hello.map x
errno          0x4          /home/raviv/arm_toolchain/gnuarm-3.4.3/bin/../lib/gcc/arm-elf/3.4.3/../../../../a
elf/lib/libc.a(reent.o)

Memory Configuration

Name           Origin           Length           Attributes
startup        0x00000000       0x00020000
*default*      0x00000000       0xffffffff

Linker script and memory map

LOAD start.o
LOAD hello.o

.text          0x00000000       0x7984
*(.text .stub .text.* .gnu.linkonce.t.*)
.text          0x00000000         0xc start.o
               0x00000000                _start
.text          0x0000000c       0x34 hello.o
               0x0000000c                main

```

Creating a standalone image

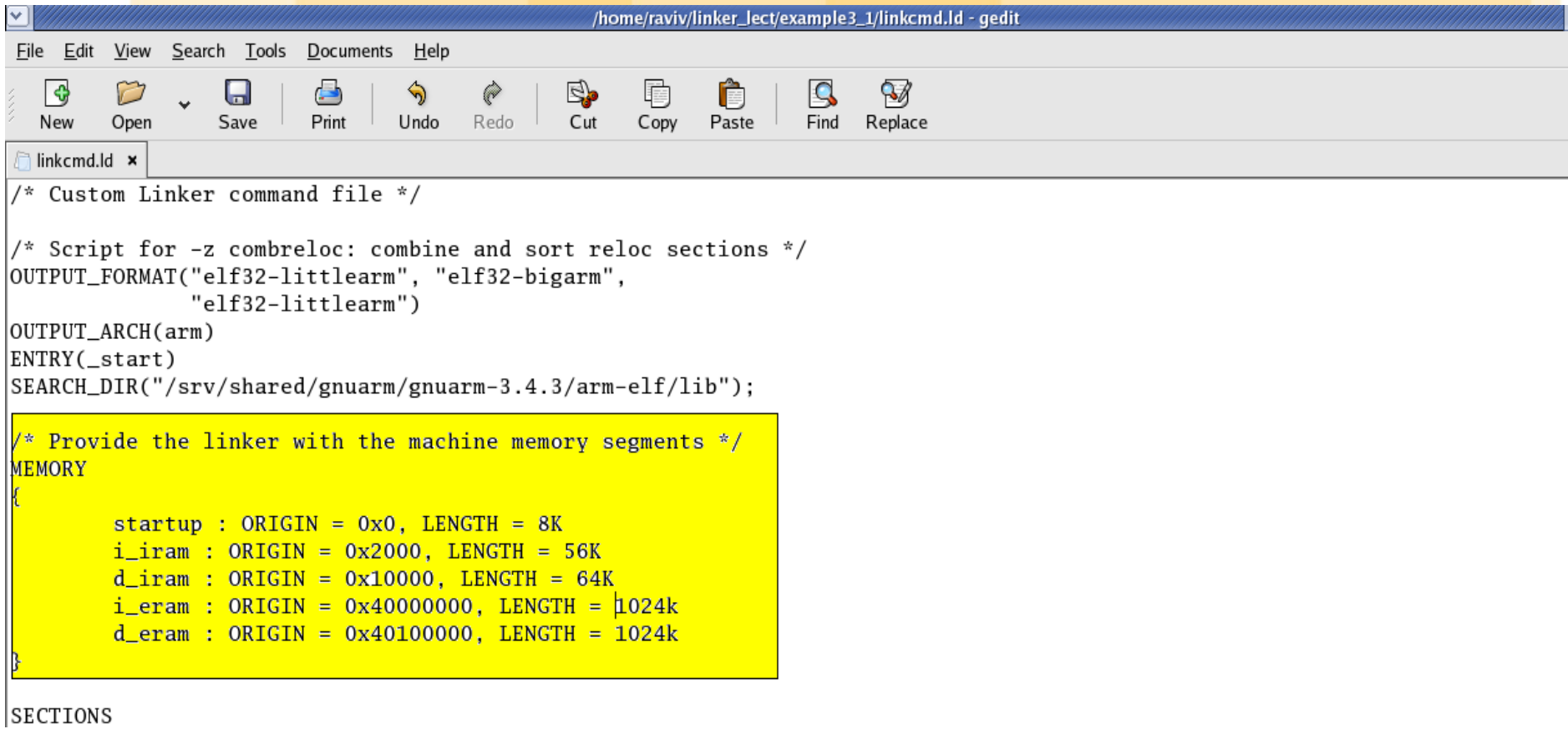
```
raviv@tregolnx1:~/linker_lect/example2_3
File Edit View Terminal Tabs Help
hello:      file format elf32-littlearm
Disassembly of section .text:
00000000 <_start>:
   0:      ea000000      b      8 <_reset>
00000004 <_main>:
   4:      0000000c      andeq  r0, r0, ip
00000008 <_reset>:
   8:      e51ff00c      ldr    pc, [pc, #-12] ; 4 <_main>
0000000c <main>:
   c:      e1a0c00d      mov    ip, sp
  10:      e92dd800      stmdb  sp!, {fp, ip, lr, pc}
  14:      e24cb004      sub    fp, ip, #4      ; 0x4
  18:      e24dd008      sub    sp, sp, #8      ; 0x8
 1c:      e50b0010      str    r0, [fp, #-16]
 20:      e50b1014      str    r1, [fp, #-20]
 24:      e59f0010      ldr    r0, [pc, #16]   ; 3c <.  
28:      eb00000d      bl     64 <printf>
 2c:      e3a03000      mov    r3, #0      ; 0x0
 30:      e1a00003      mov    r0, r3
 34:      e24bd00c      sub    sp, fp, #12    ; 0xc
 38:      e89da800      ldmia  sp, {fp, sp, pc}
 3c:      00007984      andeq  r7, r0, r4, lsl #19
:
```

Image spread to memory segments

- **Provide the linker with the machine memory segments .**
 - **Special consideration for boot memory segment.**
- **Create the sections residing in different physical memories:**
 - **Instructions (.text)**
 - **Data (.data , .bss)**
 - **Constants (.rodata)**
- **Place relevant sections in appropriate memory segment.**

Image spread to memory segments

- **Step 1 - Provide the linker with the machine memory segments**



```
/* Custom Linker command file */

/* Script for -z combreloc: combine and sort reloc sections */
OUTPUT_FORMAT("elf32-littlearm", "elf32-bigarm",
              "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SEARCH_DIR("/srv/shared/gnuarm/gnuarm-3.4.3/arm-elf/lib");

/* Provide the linker with the machine memory segments */
MEMORY
{
    startup : ORIGIN = 0x0, LENGTH = 8K
    i_iram  : ORIGIN = 0x2000, LENGTH = 56K
    d_iram  : ORIGIN = 0x10000, LENGTH = 64K
    i_eram  : ORIGIN = 0x40000000, LENGTH = 1024k
    d_eram  : ORIGIN = 0x40100000, LENGTH = 1024k
}

SECTIONS
```

Image spread to memory segments

- **Step 2 - Create the sections residing in different physical memories**

```
SECTIONS
```

```
{
```

```
.boot :
{
    start.o(.text .stub .text.* .gnu.linkonce.t.*)
    start.o(.rodata .rodata.* .gnu.linkonce.r.*)
    start.o(.data .data.* .gnu.linkonce.d.*)
    start.o(.bss .bss.* .gnu.linkonce.b.*)
} > startup
```

```
.text_iram :
{
    hello.o(.text .stub .text.* .gnu.linkonce.t.*)
    /* .gnu.warning sections are handled specially by elf32.em. */
    hello.o(.gnu.warning)
    hello.o(.glue_7t) *(.glue_7)
} > i_iram
```

```
.text_eram :
{
    *(.text .stub .text.* .gnu.linkonce.t.*)
    /* .gnu.warning sections are handled specially by elf32.em. */
    *(.gnu.warning)
    *(.glue_7t) *(.glue_7)
} > i_eram
```

Image spread to memory segments

```

/home/raviv/linker_lect/example3_1/hello.map - gedit
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy Paste Find Replace
hello.map x
default 0x00000000 0x11111111

Linker script and memory map
LOAD start.o
LOAD hello.o

.boot          0x00000000      0xc
start.o(.text .stub .text.* .gnu.linkonce.t.*)
.text          0x00000000      0xc start.o
               0x00000000      _start
start.o(.rodata .rodata.* .gnu.linkonce.r.*)
start.o(.data .data.* .gnu.linkonce.d.*)
start.o(.bss .bss.* .gnu.linkonce.b.*)

text_iram     0x00002000      0x40
hello.o(.text .stub .text.* .gnu.linkonce.t.*)
.text         0x00002000      0x40 hello.o
               0x00002000      main
hello.o(.gnu.warning)
hello.o(.glue_7t)
*(.glue_7)

text_eram     0x40000000      0x7944
*(.text .stub .text.* .gnu.linkonce.t.*)
.text         0x40000000      0x54 /home/raviv/arm_toolchain/gnuarm-3.4.3/bin/../../lib/gcc/arm-elf/3.4.3/../../../../arm
elf/lib/libc.a(sprintf.o)

```

Image packing

- **Packing the image in binary format (for flash programming)**
 - **<arch>-elf-objcopy --gap-fill 0xFF -v -S -O binary <exe.elf> <exe.bin>**

```
raviv@tregolnx1:~/linker_lect/example3_2
File Edit View Terminal Tabs Help
[raviv@tregolnx1 example3_2]$ make
arm-elf-gcc -c start.s -o start.o
arm-elf-gcc -c -mlong-calls hello.c -o hello.o
arm-elf-gcc start.o hello.o -nostartfiles -Wl,-Map -Wl,hello.map -Wl,-T -Wl,linkcmd.ld -o hello
arm-elf-objcopy --gap-fill 0xff -v -S -O binary hello hello.bin
copy from hello(elf32-littlearm) to hello.bin(binary)
[raviv@tregolnx1 example3_2]$ du -h hello.bin
1.1G    hello.bin
[raviv@tregolnx1 example3_2]$
```

VMA & LMA

- **Every loadable or allocatable output section has two addresses.**
 - **VMA, or virtual memory address. This is the address the section will have when the output file is run.**
 - **LMA, or load memory address. This is the address at which the section will be loaded.**
- **In most cases the two addresses will be the same.**
- **Provide the linker the LMA using the AT keyword.**

Image packing

```
linkcmd.ld - gedit
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy Paste Find Replace
linkcmd.ld x
SECTIONS
{
    .boot
    {
        start.o(.text .stub .text.* .gnu.linkonce.t.*)
        start.o(.rodata .rodata.* .gnu.linkonce.r.*)
        start.o(.data .data.* .gnu.linkonce.d.*)
        start.o(.bss .bss.* .gnu.linkonce.b.*)
    } > startup

    .text_iram
    {
        hello.o(.text .stub .text.* .gnu.linkonce.t.*)
        /* .gnu.warning sections are handled specially by elf32.em. */
        hello.o(.gnu.warning)
        hello.o(.glue_7t) *(.glue_7)
    } > i_iram

    .text_eram
    {
        *(.text .stub .text.* .gnu.linkonce.t.*)
        /* .gnu.warning sections are handled specially by elf32.em. */
        *(.gnu.warning)
        *(.glue_7t) *(.glue_7)
    } > i_eram
```

Image packing

```

/home/raviv/linker_lect/example4_1/hello.map - gedit
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy Paste Find Replace
hello.map x
default 0x00000000 0xFFFFFFFF

Linker script and memory map

LOAD start.o
LOAD hello.o

.boot 0x00000000 0xc load address 0x00000000
start.o(.text .stub .text.* .gnu.linkonce.t.*)
.text 0x00000000 0xc start.o
0x00000000 _start
start.o(.rodata .rodata.* .gnu.linkonce.r.*)
start.o(.data .data.* .gnu.linkonce.d.*)
start.o(.bss .bss.* .gnu.linkonce.b.*)

.text_iram 0x00002000 0x40 load address 0x0000000c
hello.o(.text .stub .text.* .gnu.linkonce.t.*)
.text 0x00002000 0x40 hello.o
0x00002000 main
hello.o(.gnu.warning)
hello.o(.glue_7t)
*(.glue_7)

.text_eram 0x40000000 0x7944 load address 0x0000004c
*(.text .stub .text.* .gnu.linkonce.t.*)

```

Image packing



```
raviv@tregolnx1:~/linker_lect/example4_1
File Edit View Terminal Tabs Help Maxim
[raviv@tregolnx1 example4_1]$ du -h hello.bin
36K hello.bin
[raviv@tregolnx1 example4_1]$
```

Image packing

- **A copy routine must be executed to copy LMA to VMA before the relevant parts are executed (Usually part of the start.s code that also clears bss sections)**

Overlays

- **Sections which are to be loaded as part of a single memory image but are to be run at the same memory address.**
- **At run time, some sort of overlay manager will copy the overlaid sections in and out of the runtime memory address as required**

Overlays

```
/home/raviv/linker_lect/example5_1/linkcmd.ld - gedit
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy Paste Find Replace
linkcmd.ld x
SECTIONS
{
    .boot                : AT(0x0)
    {
        start.o(.text .stub .text.* .gnu.linkonce.t.*)
        start.o(.rodata .rodata.* .gnu.linkonce.r.*)
        start.o(.data .data.* .gnu.linkonce.d.*)
        start.o(.bss .bss.* .gnu.linkonce.b.*)
    } > startup

OVERLAY                : NOCROSSREFS AT(LOADADDR(.boot) + SIZEOF(.boot))
{
    .text1_iram
    {
        hello.o(.text .stub .text.* .gnu.linkonce.t.*)
        /* .gnu.warning sections are handled specially by elf32.em.  */
        hello.o(.gnu.warning)
        hello.o(.glue_7t) *(.glue_7)
    }
    .text2_iram
    {
        another_hello.o(.text .stub .text.* .gnu.linkonce.t.*)
        /* .gnu.warning sections are handled specially by elf32.em.  */
        another_hello.o(.gnu.warning)
        another_hello.o(.glue_7t) *(.glue_7)
    }
} > i_iram

.text_eram             : AT(LOADADDR(.text1_iram) + SIZEOF(.text1_iram) + SIZEOF(.text2_iram))
}
```

Overlays

```

hello.map x
LOAD start.o
LOAD hello.o
LOAD another_hello.o

.boot          0x00000000          0xc load address 0x00000000
start.o(.text .stub .text.* .gnu.linkonce.t.*)
.text          0x00000000          0xc start.o
              0x00000000          _start
start.o(.rodata .rodata.* .gnu.linkonce.r.*)
start.o(.data .data.* .gnu.linkonce.d.*)
start.o(.bss .bss.* .gnu.linkonce.b.*)

.text1_iram    0x00002000          0x40 load address 0x0000000c
hello.o(.text .stub .text.* .gnu.linkonce.t.*)
.text          0x00002000          0x40 hello.o
              0x00002000          main
hello.o(.gnu.warning)
hello.o(.glue_7t)
*(.glue_7)
              0x0000000c          __load_start_text1_iram = LOADADDR (.text1_iram)
              0x0000004c          __load_stop_text1_iram = (LOADADDR (.text1_iram) + SIZEOF (.text1_iram))

.text2_iram    0x00002000          0x28 load address 0x0000004c
another_hello.o(.text .stub .text.* .gnu.linkonce.t.*)
.text          0x00002000          0x28 another_hello.o
              0x00002000          another_hello
another_hello.o(.gnu.warning)
another_hello.o(.glue_7t)
*(.glue_7)
              0x0000004c          __load_start_text2_iram = LOADADDR (.text2_iram)
              0x00000074          __load_stop_text2_iram = (LOADADDR (.text2_iram) + SIZEOF (.text2_iram))

.text_eram     0x40000000          0x7944 load address 0x00000074
*(.text .stub .text.* .gnu.linkonce.t.*)

```