armlink 第一章 鏡像結構

原創 安仔都有人用 2020-06-18 23:25

第一章 鏡像結構

注意：本文章只針對，裸機開發.至於 SysV,BPABI,BP 的鏈接模型請參考《armlink_user_guide》

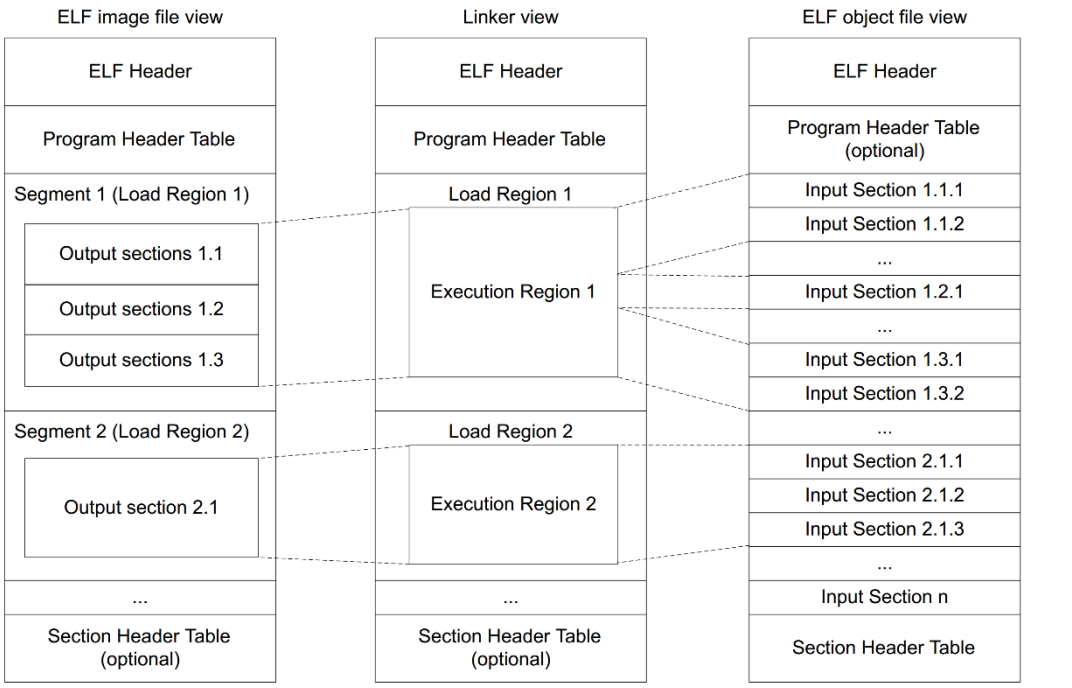
1.1 重要的概念

1.1.1 鏡像的構成

編譯器將源文件編譯成中間文件。鏈接器將中間文件最終生成鏡像文件。

在中間文件中，邏輯最小單元稱為 section。鏈接器將所有中間文件的 section 收集起來，然後按照一定的規則，進行重新組織。將具有相同屬性的 section，組織在一起，稱為輸出 section。相應的，中間文件的 section 稱為輸入 section。然後再將不同的輸出 section 組織在一起，起個名字叫做 region。

他們之間的關係如下圖



下面分別介紹輸入 section，輸出 section，region 和 segment

1. 輸入 section：輸入 section 來自於中間文件。它含有代碼，初始化數據或者描述信息。描述信息用於表示未被初始化的段或者鏡像執行之前必須設置為 0 的段。輸入 section 具有 RO,RW，XO,ZI 這樣的屬性。這些屬性被 armlink 用於組織成 region 或者輸出 section
2. 輸出 section：輸出 section 其實是具有相同屬性的輸入 section 的集合，因此輸出 section 的屬性和輸入 section 的屬性相同。這些輸入 section 被連續的放置於內存中。
3. region：一個 region 最多包含四個不同屬性的輸出 section。默認情況下，在 region 中的輸出 section 根據屬性進行排序。XO 輸出 section 總是第一個，接著是 RO 輸出 section，然後是 RW 輸出 section，最後是 ZI 輸出 section。通常一個 region 被映射成一個物理內存設備，例如 ROM，RAM。可以使用 scatter 改變輸出 section 的順序。
4. segment：等同於 region，之所以叫 segment 是因為在 arm elf 標準中使用了這個術語。

注意：armlink 的單個 segment 的最大值為 2GB

RO——read only

RW——read write

ZI——zero initialized

XO——execute only

1.1.2 鏡像的地址

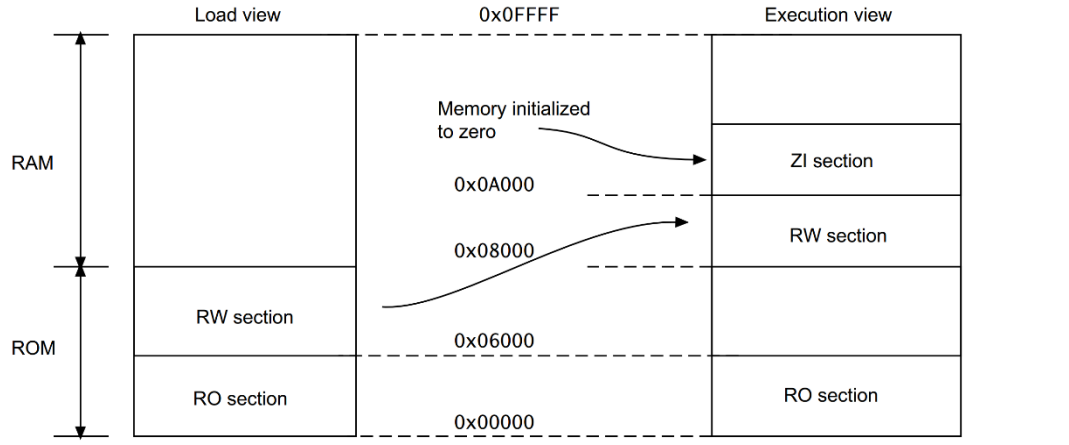
在執行鏡像之前，可能必須要把部分 region 移動到它的執行地址處，或者創建 ZI 輸出 section。例如，對於已經初始化的 RW 數據可能必須將其從 ROM 移動到 RAM 處。

這樣就會導致一個鏡像具有兩種不同的內存視圖：加載視圖，執行視圖

1. 加載視圖：描述了鏡像執行之前的地址
2. 執行視圖：描述了鏡像執行時的地址

當 region 的加載地址和執行地址一樣時，稱為 root region

下圖展示了這兩者的區別。

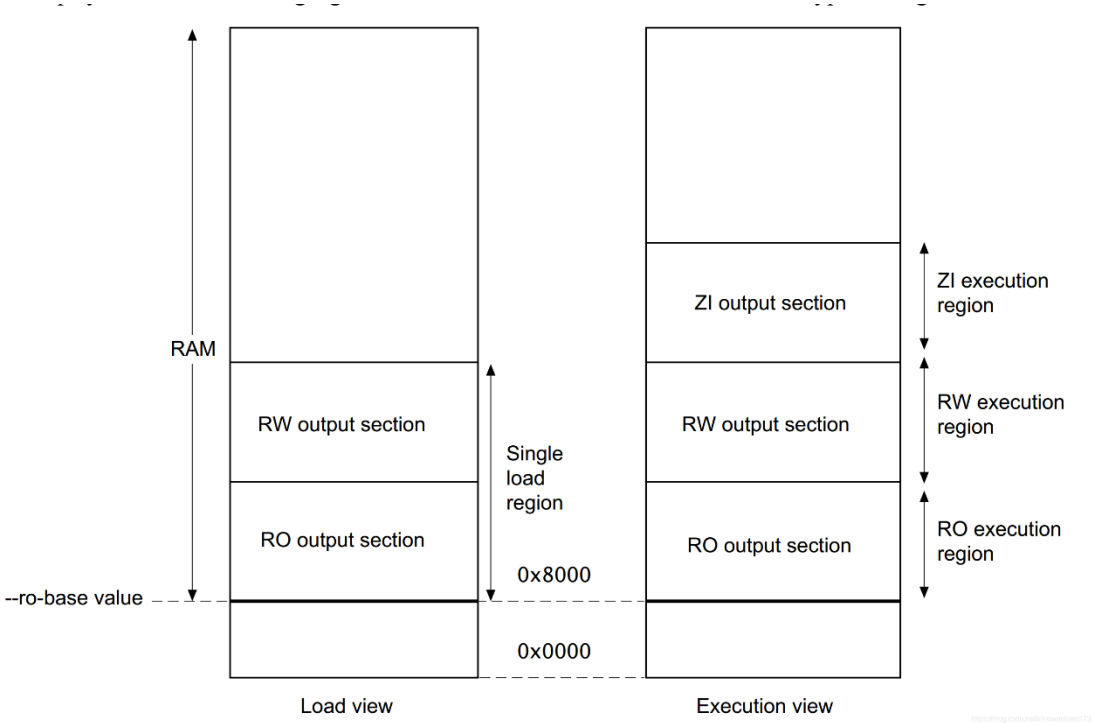


1.2 三種簡單的鏡像結構

為了幫助理解上述概念，這裡介紹三種簡單鏡像。

1.2.1 類型 1

類型 1：一個加載 region，三個執行 region。如下圖所示：



為了指定 ro 的地址，可以使用命令行選項–ro_base.命令如下：

```
armlink --ro_base 0x8000
```

注意：0x8000 是默認地址，因此在這個例子中，也可以不用指定

加載視圖：這個唯一的加載 region 由 RO 和 RW 兩個輸出 section 組成。ZI 輸出 section 在加載視圖中不存在，他是在執行之前創建的。

執行視圖：由三個連續的執行 region 組成，他們分別包含 RO，RW 和 ZI 輸出 section。RO 和 RW 的執行地址與加載地址一樣，因此無需做任何移動。但是，ZI 執行 region 必須在運行時創建。

使用--ro_base address 指定 RO region 的加載和執行地址。

使用--zi_base address 指定 ZI region 的執行地址。

包含 XO 的加載視圖

如果一個鏡像包含 XO section。那麼 XO 輸出 section 就被放置於--ro_base 指定的位置處。RO 和 RW 則連續排在其後

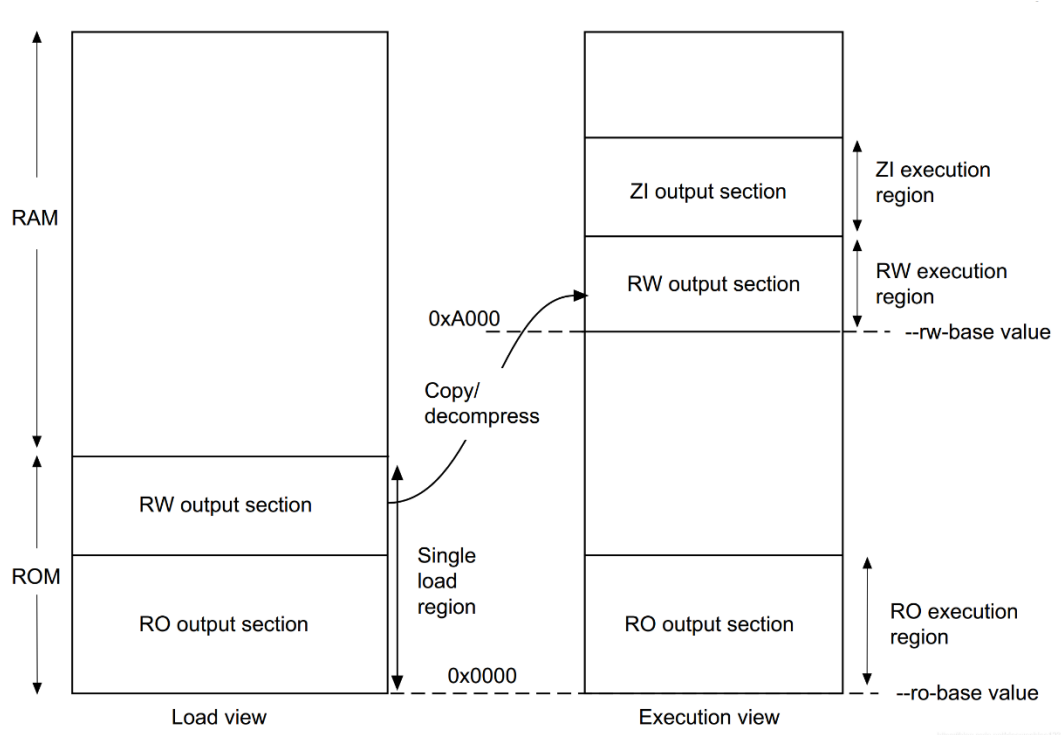
包含 XO 的執行視圖

如果一個鏡像包含 XO section，那麼 XO 執行地址為--ro_base 指定的地址處，RO，RW 和 ZI 緊隨其後

1.2.2 類型 2

類型 2：一個加載 region，三個執行 region。但是 RW 執行 region 的地址和 RO 的執行地址不連續。

如下圖：



使用下面的命令創建這種類型的鏡像：

```
armlink --ro_base 0x0 --rw_base 0xA000
```

加載視圖：一個加載 region，包含 RO 和 RW 輸出 section，且這兩者連續放置。

執行視圖：第一執行 region 包含 RO 輸出 section。第二個執行 region 包含 RW 和 ZI 輸出 section

第一個執行 region 和加載 region 地址相同，因此不需要移動。第二個執行 regoin 和加載 region 的地址不同，因此它需要進行搬運。ZI 執行 region 在運行時創建

使用--ro_base address 指定 RO 輸出 section 加載和執行地址。

使用--rw_base address 指定 RW 輸出 section 的執行地址。

使用--zi_base address 指定 ZI 執行 region 的地址

含有 XO region 的加載視圖

如果一個鏡像含有 XO section。那麼 XO section 被放置在--ro_base 指定的地址處。RO 和 RW 則緊隨其後

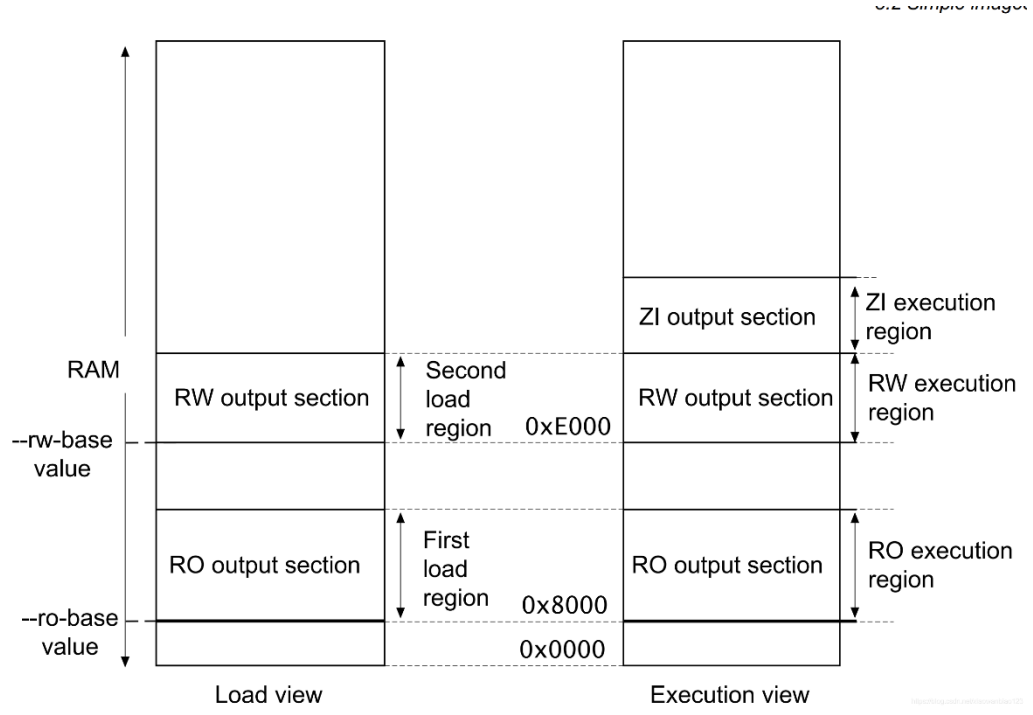
含有 XO region 的執行視圖

如果一個鏡像含有 XO section。那麼 XO section 的執行 region 放置在--ro_base 指定的地址處。RO 則緊隨其後

如果使用了--xo_base address,那麼 XO 執行 region 被放置在這個指定的地址處

1.2.3 類型 3

類型 3：該類型類似於類型 2，但是跟類型 2 不同的是，類型 3 的加載視圖，被分隔成多個 root region。如下圖：



使用下面的命令，生成這種類型的鏡像：

```
armlink --split --ro_base 0x8000  --rw_base 0xE000
```

加載視圖：第一個 加載 region 包含 RO 輸出 section 。第二個 加載 region 包含 RW 輸出 section。

執行視圖：第一個執行 region 包含 RO 輸出 section，第二個執行 region 包含 RW 輸出 section。第三個執行 region 包含 ZI 輸出 section。

RO 和 RW 輸出 section 都是 root region,因此不需要移動，只有 ZI 需要在運行時創建。

使用–split,將默認的單個加載 region 分隔開。

含有 XO 的加載視圖

如果一個鏡像含有 XO section，則 XO 被放置在--ro_base 所在的地址處，RO 和 RW 緊隨其後

含有 XO 的執行視圖

如果一個鏡像含有 XO section，則放置在--ro_base 所指的地址處。

如果你指定了--split 選項，XO 和 RO 被放置在第一個加載 regioin 中，RW 和 ZI 執行 region 放在第二個加載 region 中

如果你指定了--xo_base,XO 的執行地址則是此處指定的地址。

本章完，下章，scatter 文件語法