

Sistemas Empotrados

Tema 2: Procesador y mapa de memoria

Lección 7: El mapa de memoria



Contenidos

Tema 2: Procesador y mapa de memoria

El procesador

- Motivación

- Introducción a la arquitectura ARMv4T

- Repertorio de instrucciones de la arquitectura ARMv4T

- El hola mundo de un sistema empotrado

- Application Binary Interface* de la arquitectura ARM

El mapa de memoria

Introducción

- Direcciones de carga y de ejecución

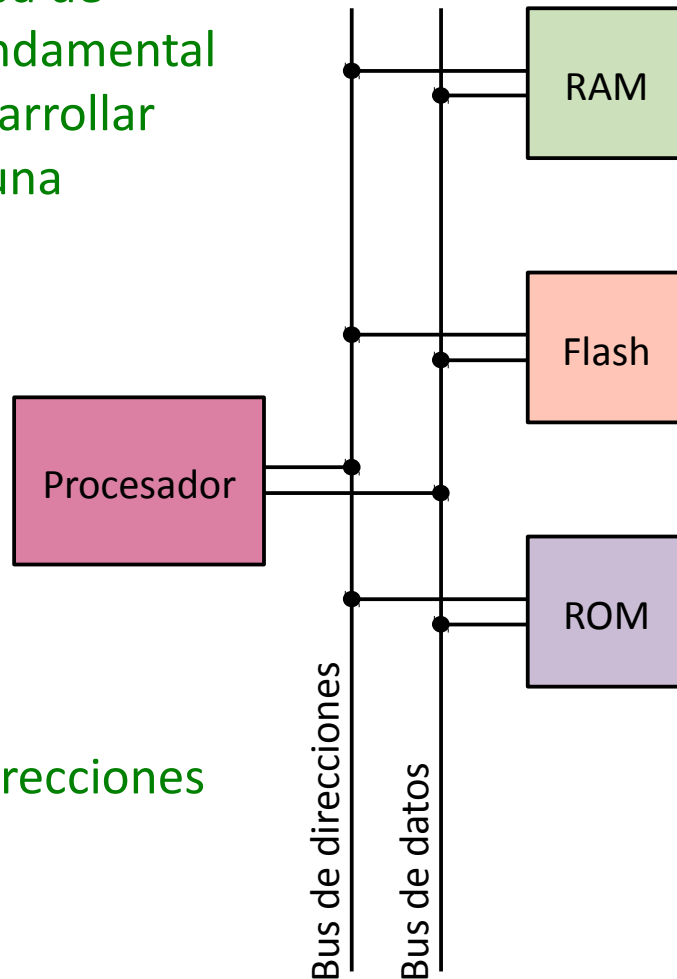
- El formato ELF

- El script de enlazado

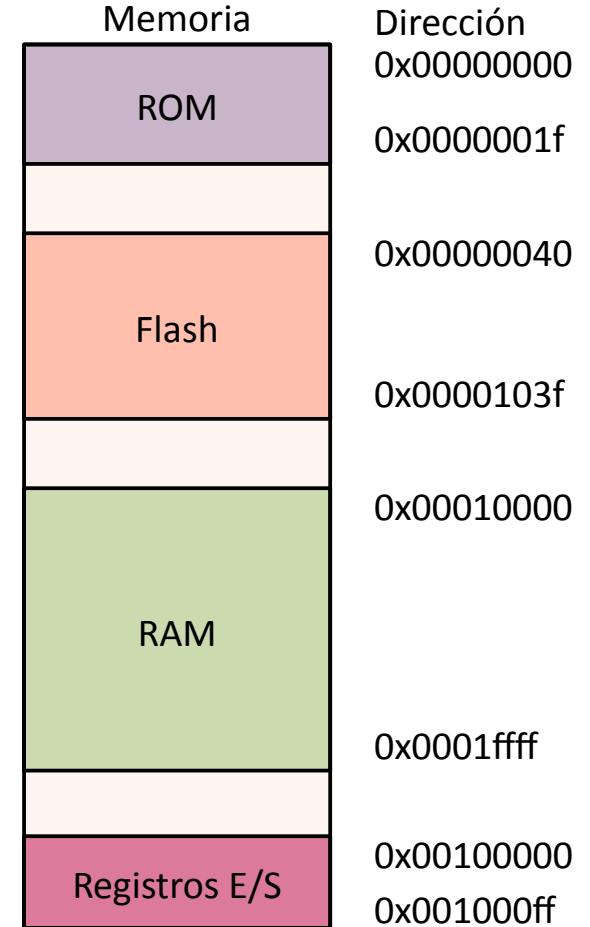
- Ejemplos

El mapa de memoria

Conocer el mapa de memoria es fundamental para poder desarrollar software para una plataforma



No todas las direcciones son válidas

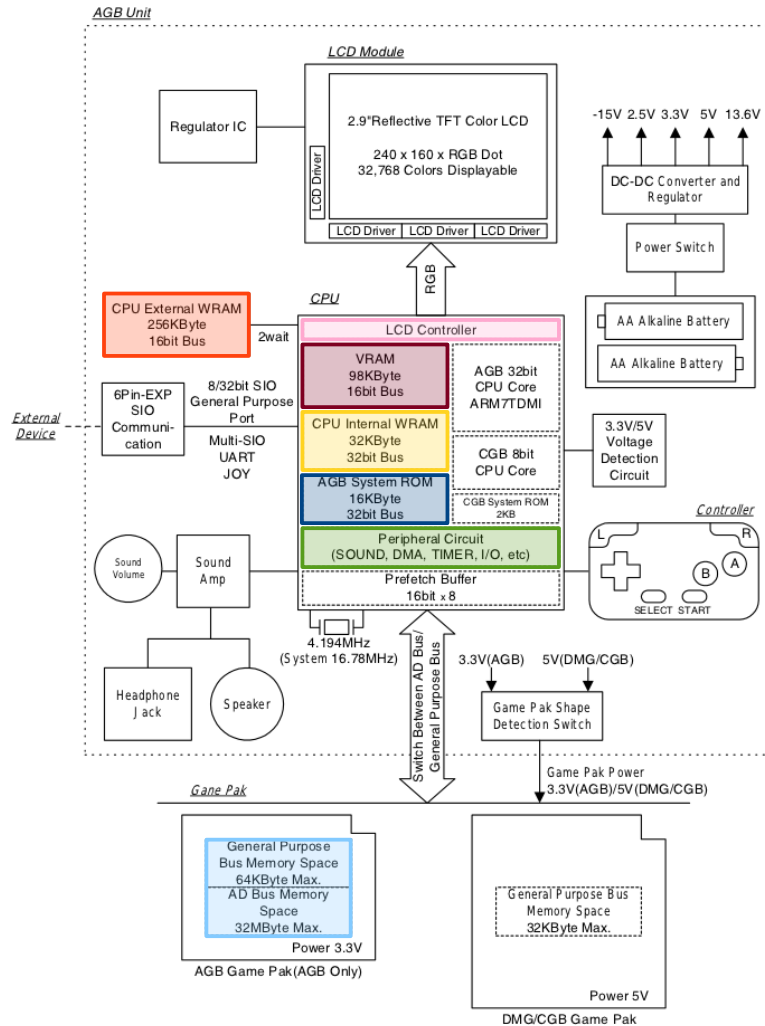


Cada plataforma tendrá un mapa de memoria diferente

Diferentes periféricos, tipos de memoria, tamaños, direcciones, etc.

El mapa de memoria

Mapa de memoria de la Gameboy Advance de Nintendo



0FFFFFFh		
0E00FFFh	Game Pak RAM (0 - 512 Kbits)	Images
0E00000h		
0DFFFFFFh	Game Pak ROM Wait State 2 (32 MB)	
0C00000h		
0BFFFFFFh	Game Pak ROM Wait State 1 (32 MB)	Flash Memory (1 Mbit)
0A00000h		
09FFFFFFh	Game Pak ROM Wait State 0 (32 MB)	
0800000h		
070003FFh	OAM (1 Kbyte)	Mask ROM (255 Mbits)
0700000h		
06017FFFh	VRAM (96 Kbytes)	
0600000h		
050003FFh	Palette RAM (1 Kbyte)	Flash Memory (1 Mbit)
0500000h		
0400000h	I/O, Registers	
03007FFFh	CPU Internal Working RAM (32 Kbytes)	
0300000h		Mask ROM (255 Mbits)
0203FFFFh	CPU External Working RAM (256 Kbytes)	
0200000h		
00003FFFh	System ROM (16 Kbytes)	
0000000h		



Fuente:

Nintendo of America Inc. AGB Programming Manual, version 1.22, 1999 – 2001.

El mapa de memoria

Debemos saber dónde alojar el cargador, el código, las variables, las pilas, el heap,...

09FFFFFFh	Game Pak ROM Wait State 0 (32 MB)
08000000h	
070003FFh	OAM (1 Kbyte)
07000000h	
06017FFFh	VRAM (96 Kbytes)
06000000h	
050003FFh	Palette RAM (1 Kbyte)
05000000h	
	I/O, Registers
04000000h	
03007FFFh	CPU Internal Working RAM (32 Kbytes)
03000000h	
0203FFFFh	CPU External Working RAM (256 Kbytes)
02000000h	
00003FFFh	System ROM (16 Kbytes)
00000000h	

El código de la aplicación debe almacenarse en una memoria no volátil. El cargador necesita saber en qué direcciones de memoria está el código, para saltar a ejecutarlo directamente o para copiarlo a la RAM antes de ejecutarlo.

La ejecución desde la ROM es más lenta.

La ejecución desde la RAM implica más cantidad de RAM

El código que debe ejecutarse rápidamente (ISRs, drivers, ...), las pilas y los datos que se usen habitualmente deberían estar en una RAM de rápido acceso para no penalizar la ejecución

El resto de datos y código pueden almacenarse en RAM externas (si la plataforma dispone de ellas) durante la ejecución

El cargador debe estar en una memoria no volátil mapeada en la dirección 0x00000000

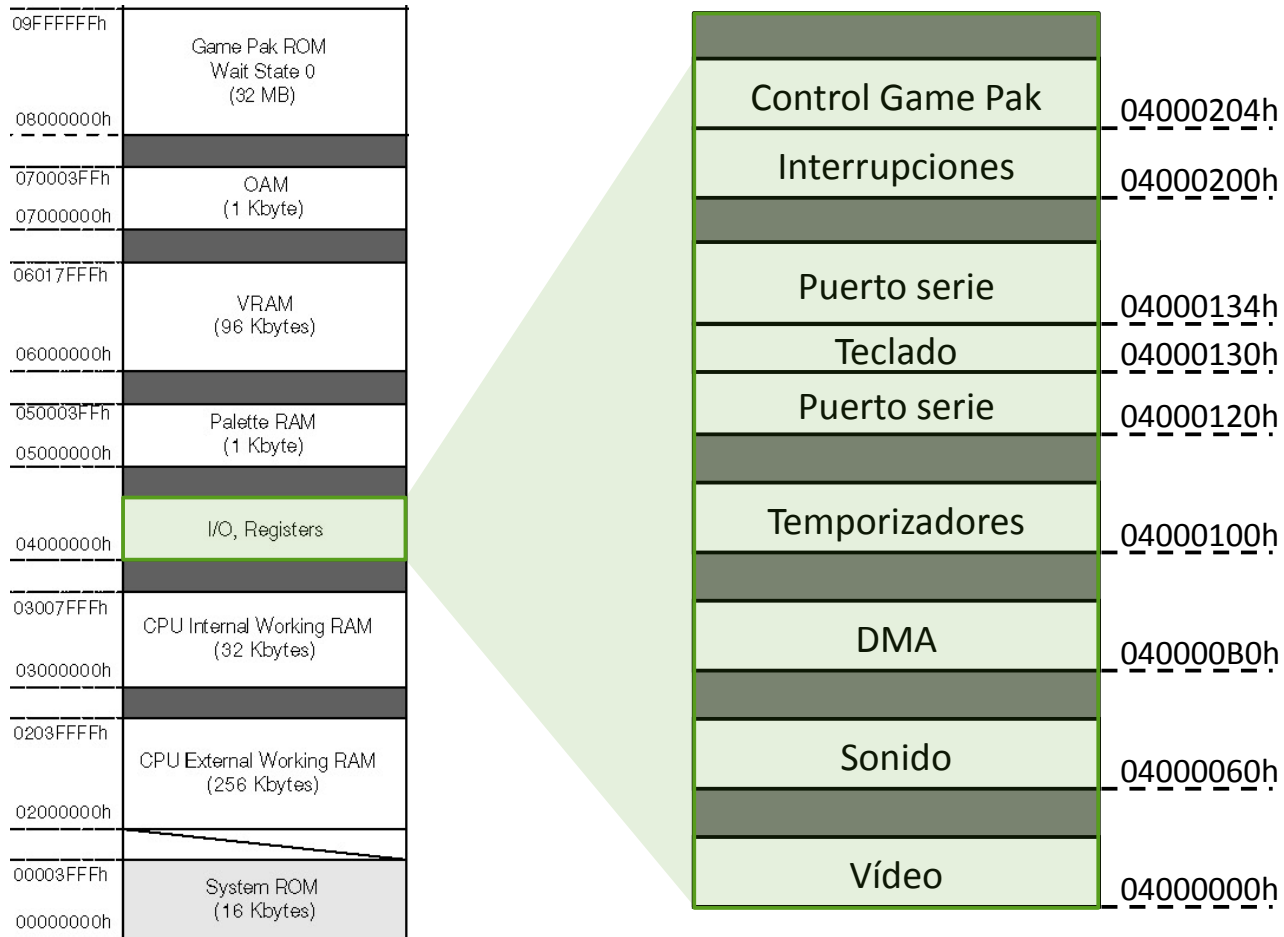


Fuente:

Nintendo of America Inc. AGB Programming Manual, version 1.22, 1999 – 2001.

El mapa de memoria

Normalmente, los registros de control/estado están mapeados en memoria



Fuente:

Nintendo of America Inc. AGB Programming Manual, version 1.22, 1999 – 2001.



El mapa de memoria

Registros de control/estado del controlador de DMA

04000204h	Control Game Pak
04000200h	Interrupciones
04000134h	Puerto serie
04000130h	Teclado
04000120h	Puerto serie
04000100h	Temporizadores
040000B0h	DMA
04000060h	Sonido
04000000h	Vídeo

040000F2h	DMA3CNT H	DMA3 control
040000F0h	DMA3CNT L	DMA3 word count
040000E6h	DMA3DAD	DMA3 Dest. Address
040000E2h	DMA3SAD	DMA3 Source Address
040000E0h	DMA2CNT H	DMA2 control
040000DEh	DMA2CNT L	DMA2 word count
040000DAh	DMA2DAD	DMA2 Dest. Address
040000C6h	DMA2SAD	DMA2 Source Address
040000C4h	DMA1CNT H	DMA1 control
040000C2h	DMA1CNT L	DMA1 word count
040000BEh	DMA1DAD	DMA1 Dest. Address
040000BAh	DMA1SAD	DMA1 Source Address
040000B8h	DMA0CNT H	DMA0 control
040000B6h	DMA0CNT L	DMA0 word count
040000B4h	DMA0DAD	DMA0 Dest. Address
040000B0h	DMA0SAD	DMA0 Source Address

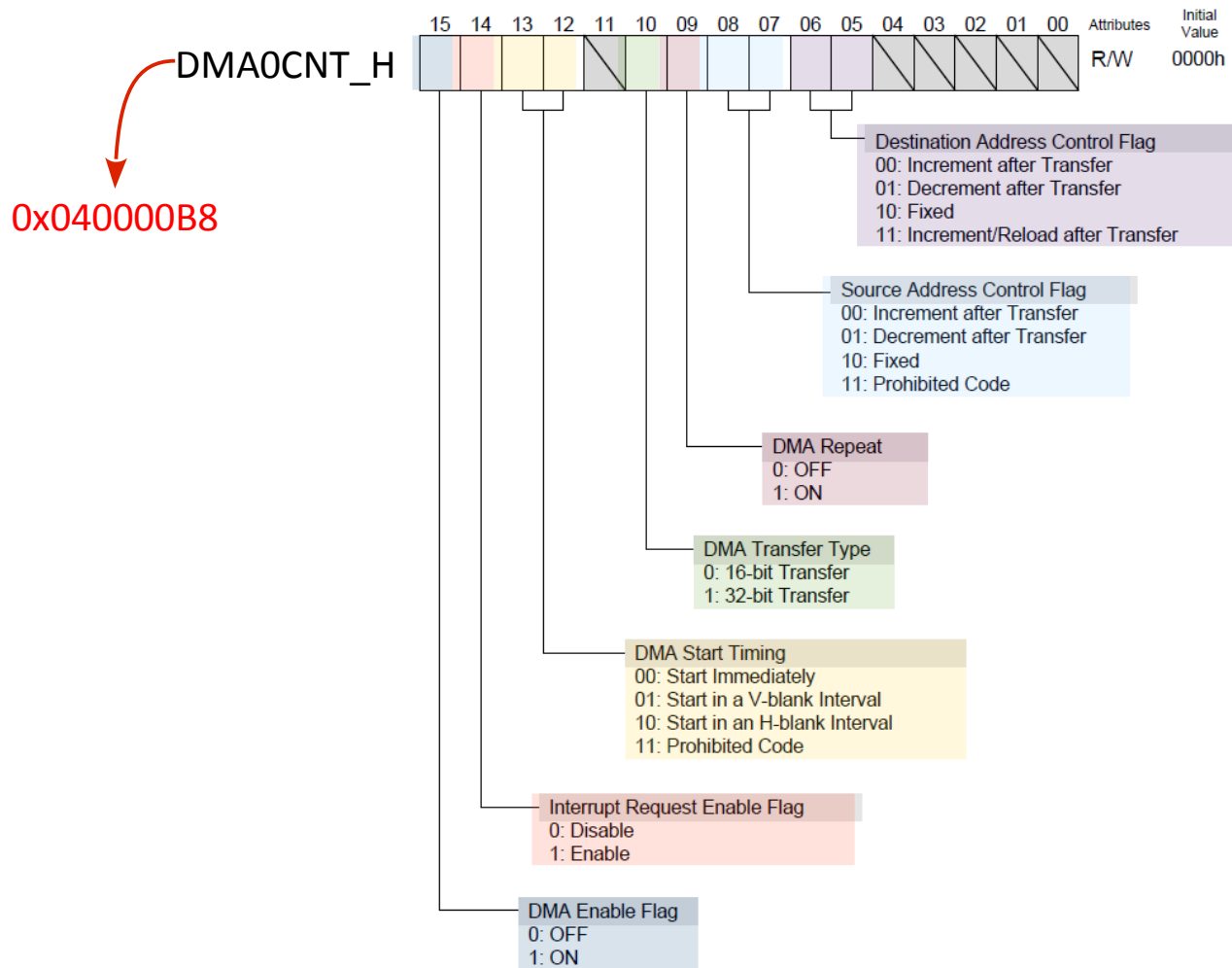


Fuente:

Nintendo of America Inc. AGB Programming Manual, version 1.22, 1999 – 2001.

El mapa de memoria

Detalle de un registro de control de DMA



El controlador de DMA implementa todas las funciones de gestión del dispositivo mediante hardware

El programador debe conocer qué bits debe modificar en qué registros de E/S para dar las órdenes adecuadas al dispositivo

Conocer el mapa de memoria es fundamental para poder manejar los dispositivos de la plataforma



Fuente:

Nintendo of America Inc. AGB Programming Manual, version 1.22, 1999 – 2001.

Contenidos

Tema 2: Procesador y mapa de memoria

El procesador

- Motivación

- Introducción a la arquitectura ARMv4T

- Repertorio de instrucciones de la arquitectura ARMv4T

- El hola mundo de un sistema empotrado

- Application Binary Interface* de la arquitectura ARM

El mapa de memoria

- Introducción

- Direcciones de carga y de ejecución

- El formato ELF

- El script de enlazado

- Ejemplos

Direcciones de carga y ejecución

Dirección de carga (*Load Memory Address, LMA*):

Dirección de memoria en la que se almacena permanentemente el código o los datos

Dirección de ejecución (*Virtual Memory Address, VMA*):

Dirección de memoria en la que reside el código o los datos en tiempo de ejecución

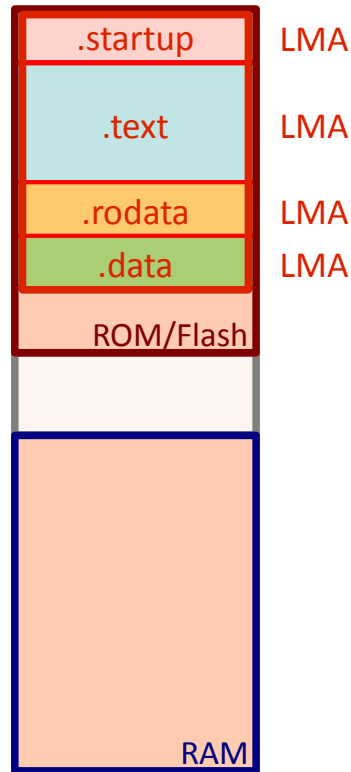
El linker genera direcciones VMA para todos los accesos a memoria de la aplicación

Posibilidades:

- Si el código se ejecuta desde la ROM/Flash, su VMA coincide con su LMA
- Si el código se ejecuta desde la RAM, tendrá una LMA en la ROM/Flash donde estará almacenado y una VMA en la RAM durante su ejecución
- Si las constantes permanecen en la ROM/Flash durante la ejecución del programa, su VMA coincide con su LMA
- Las variables globales inicializadas tienen una LMA donde está su valor inicial en la ROM/Flash y una VMA en la RAM durante la ejecución del programa
- Las variables globales sin inicializar sólo tienen VMA en la RAM

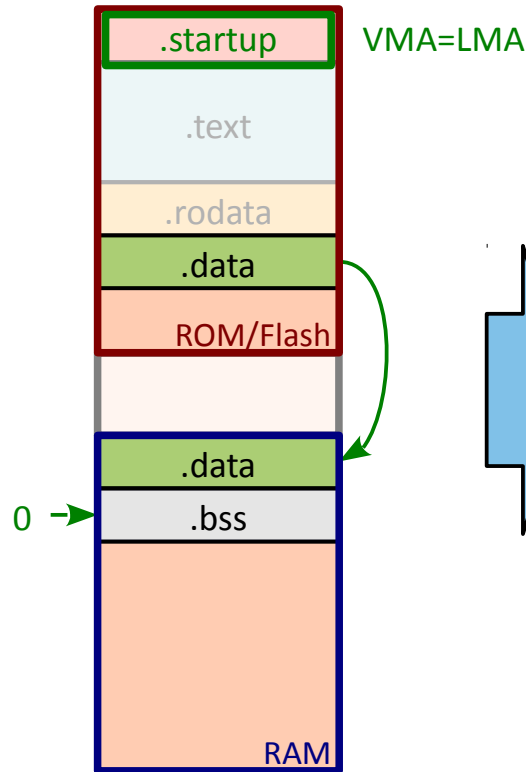
Ejemplo: Ejecución de la aplicación desde la ROM/Flash (más barato)

Sistema apagado



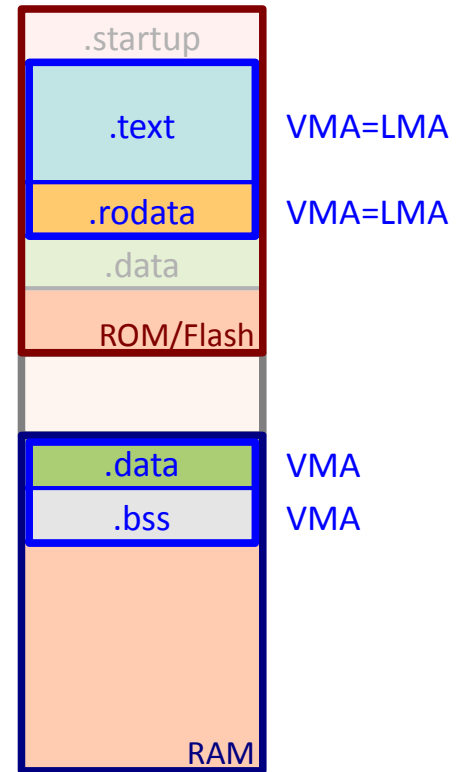
La imagen del firmware se almacena en las direcciones LMA, que estarán mapeadas a una ROM o una Flash

Arranque del sistema



El cargador prepara el runtime de C y salta a la VMA de la función main de la aplicación

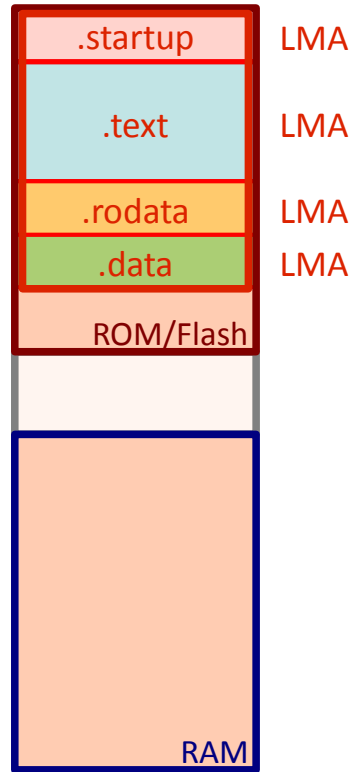
Ejecución de la aplicación



La aplicación se ejecuta desde su VMA, en este caso mapeada a la ROM/Flash

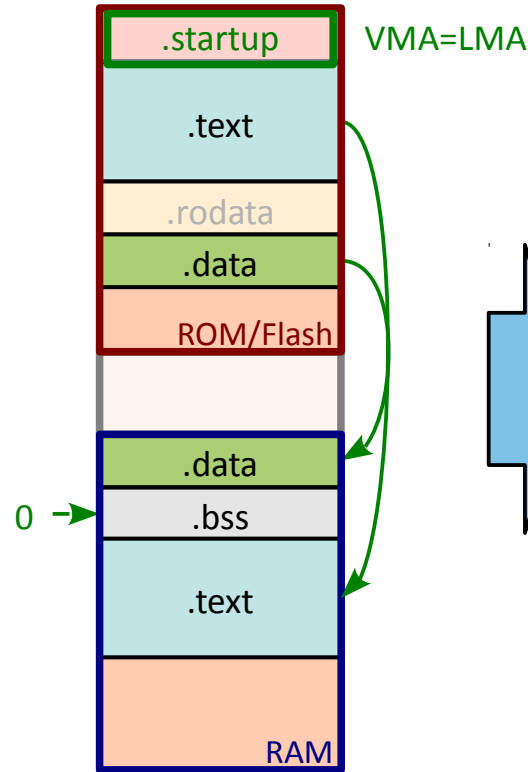
Ejemplo: Ejecución de la aplicación desde la RAM (más rápido)

Sistema apagado



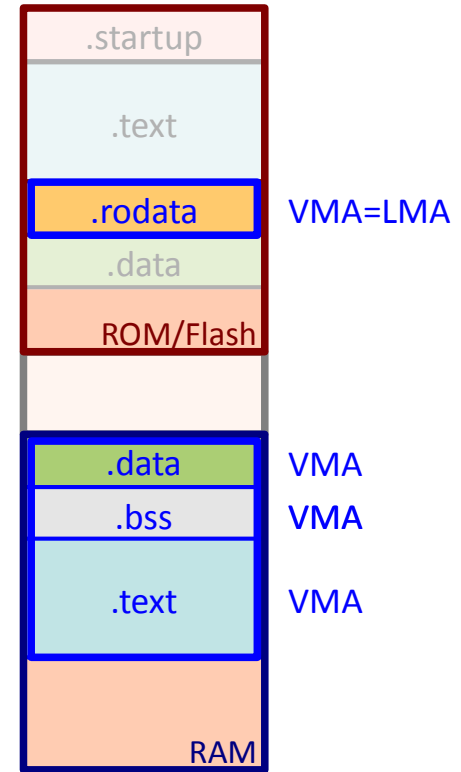
La imagen del firmware se almacena en las direcciones LMA, que estarán mapeadas a una ROM o una Flash

Arranque del sistema



El cargador prepara el runtime de C y salta a la VMA de la función main de la aplicación

Ejecución de la aplicación



La aplicación se ejecuta desde su VMA, en este caso mapeada a la RAM

Contenidos

Tema 2: Procesador y mapa de memoria

El procesador

- Motivación

- Introducción a la arquitectura ARMv4T

- Repertorio de instrucciones de la arquitectura ARMv4T

- El hola mundo de un sistema empotrado

- Application Binary Interface* de la arquitectura ARM

El mapa de memoria

- Introducción

- Direcciones de carga y de ejecución

- El formato ELF

- El script de enlazado

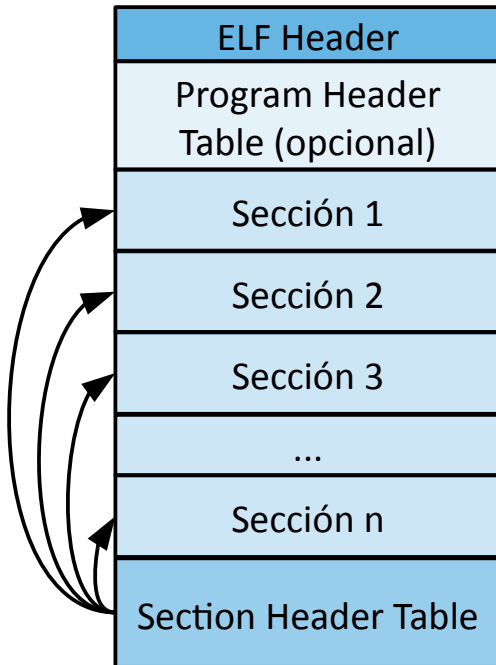
- Ejemplos

Ficheros ELF (*Executable & Linkable Format*)

Los ficheros objeto y los ejecutables siguen el formato ELF

El enlazador ve a los ficheros ELF como un conjunto de secciones

Visión del enlazador



Tipos de sección

NULL	Cabecera inactiva sin sección
PROGBITS	Código o datos inicializados
SYMTAB	Tabla de símbolos para enlazado estático
STRTAB	Tabla de cadenas (nombres de los símbolos)
RELA/REL	Entradas de reubicación
HASH	Tabla hash símbolos en ejecución
DYNAMIC	Información para el enlazado dinámico
NOBITS	Datos sin inicializar
DYNSYM	Tabla de símbolos para enlazado dinámico

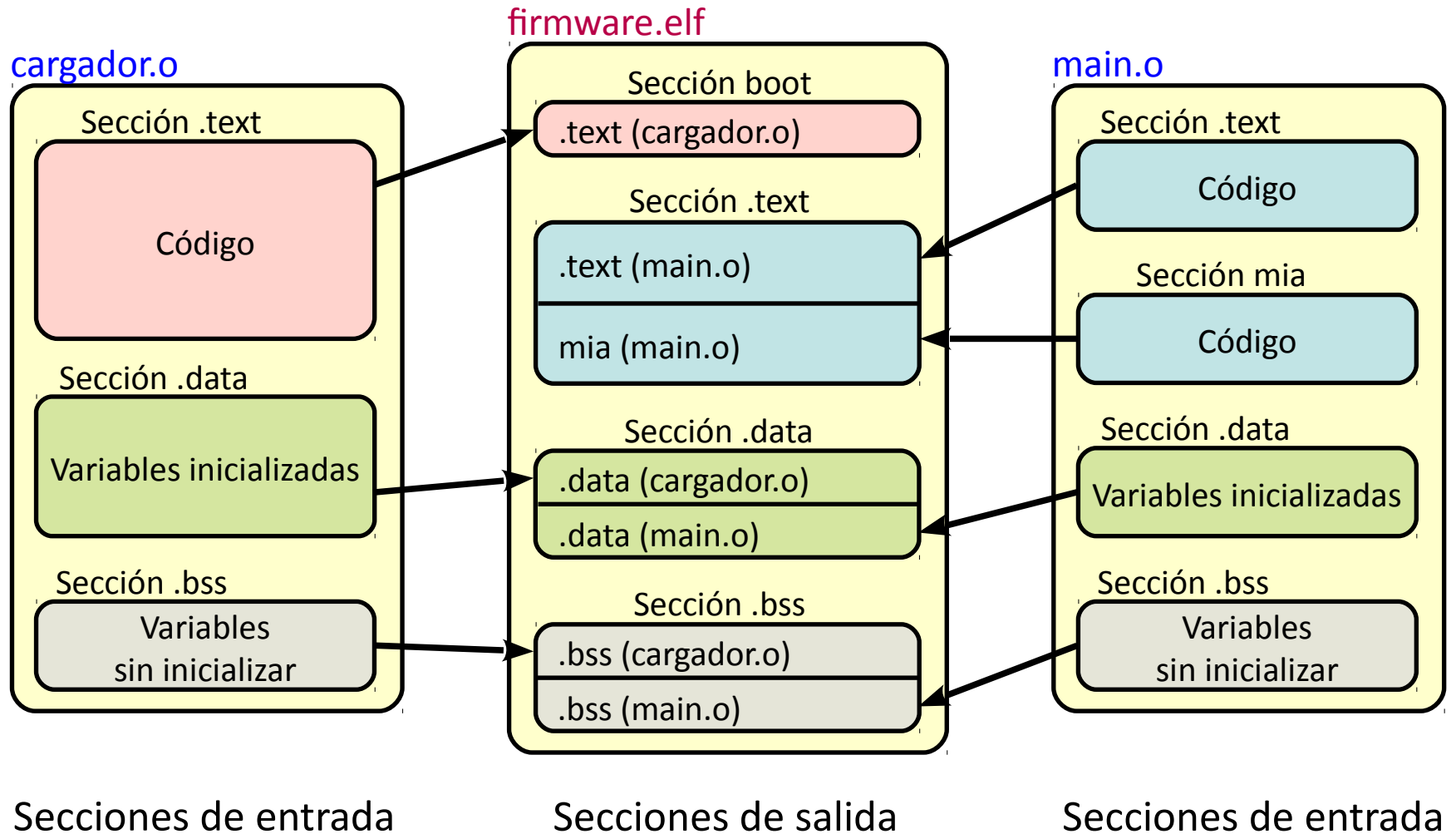
Atributos de sección

WRITE	La sección es modificable
ALLOC	La sección contiene datos
EXEC	La sección contiene instrucciones

Secciones generadas por defecto al compilar

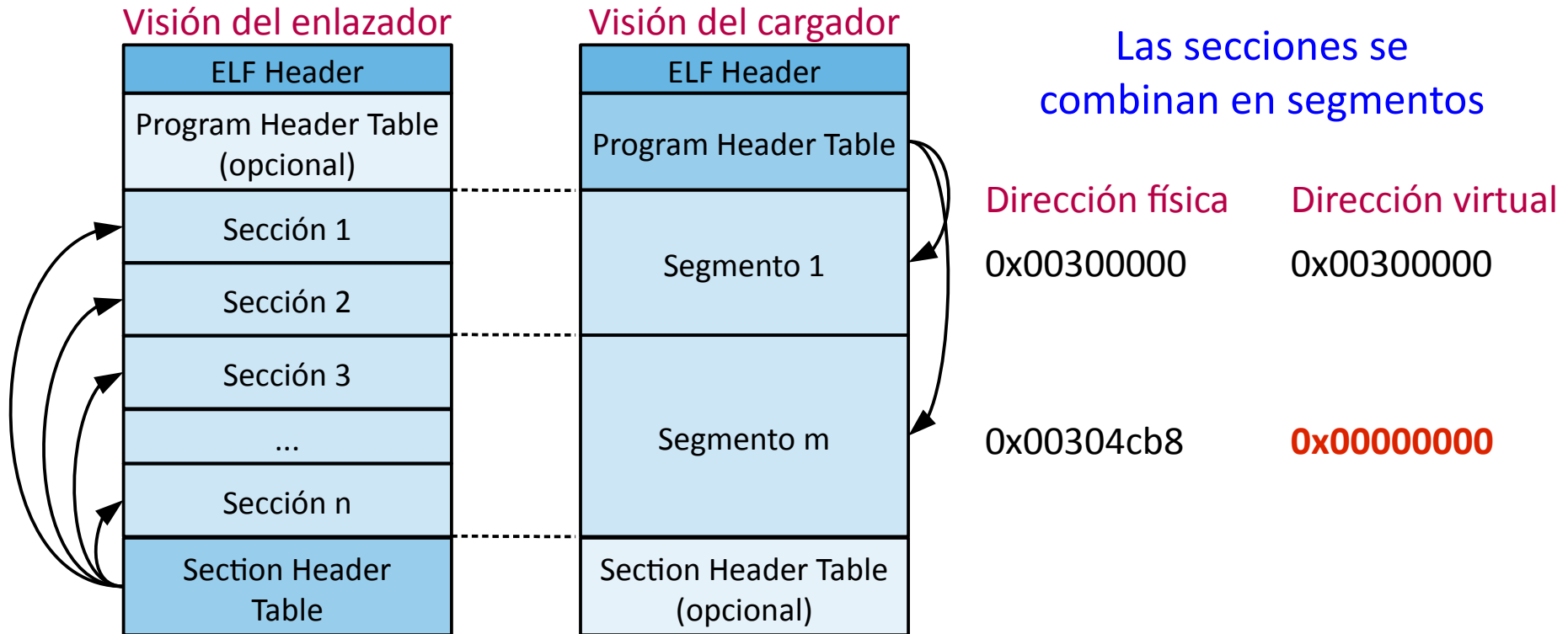
Nombre	Descripción	Tipo	Atributos
.text	Código del programa	PROGBITS	ALLOC & EXEC
.data	Datos globales inicializados	PROGBITS	ALLOC & WRITE
.bss	Datos globales sin inicializar	NOBITS	ALLOC & WRITE

Proceso de enlazado



Obtención del archivo ejecutable que
contiene el firmware a partir de los ficheros objeto

Visión del sistema operativo



Cada segmento tiene una dirección física que indica dónde se almacenará en ROM/Flash

Cada segmento tiene una dirección virtual que indica dónde estará cuando el programa se esté ejecutando

La dirección virtual no tiene por qué coincidir con la física

El cargador copia los segmentos a sus direcciones virtuales antes de pasar a ejecutar el programa

Contenidos

Tema 2: Procesador y mapa de memoria

El procesador

- Motivación

- Introducción a la arquitectura ARMv4T

- Repertorio de instrucciones de la arquitectura ARMv4T

- El hola mundo de un sistema empotrado

- Application Binary Interface* de la arquitectura ARM

El mapa de memoria

- Introducción

- Direcciones de carga y de ejecución

- El formato ELF

- El script de enlazado

- Ejemplos

Descripción del mapa de memoria

La directiva MEMORY se utiliza para definir las regiones de memoria del sistema

MEMORY

{

Definición de región #1

Definición de región #2

...

}

Podemos definir cuantas regiones necesitemos

Definición de una región de memoria

nombre : org = origen , len = longitud

Nombre
de la región

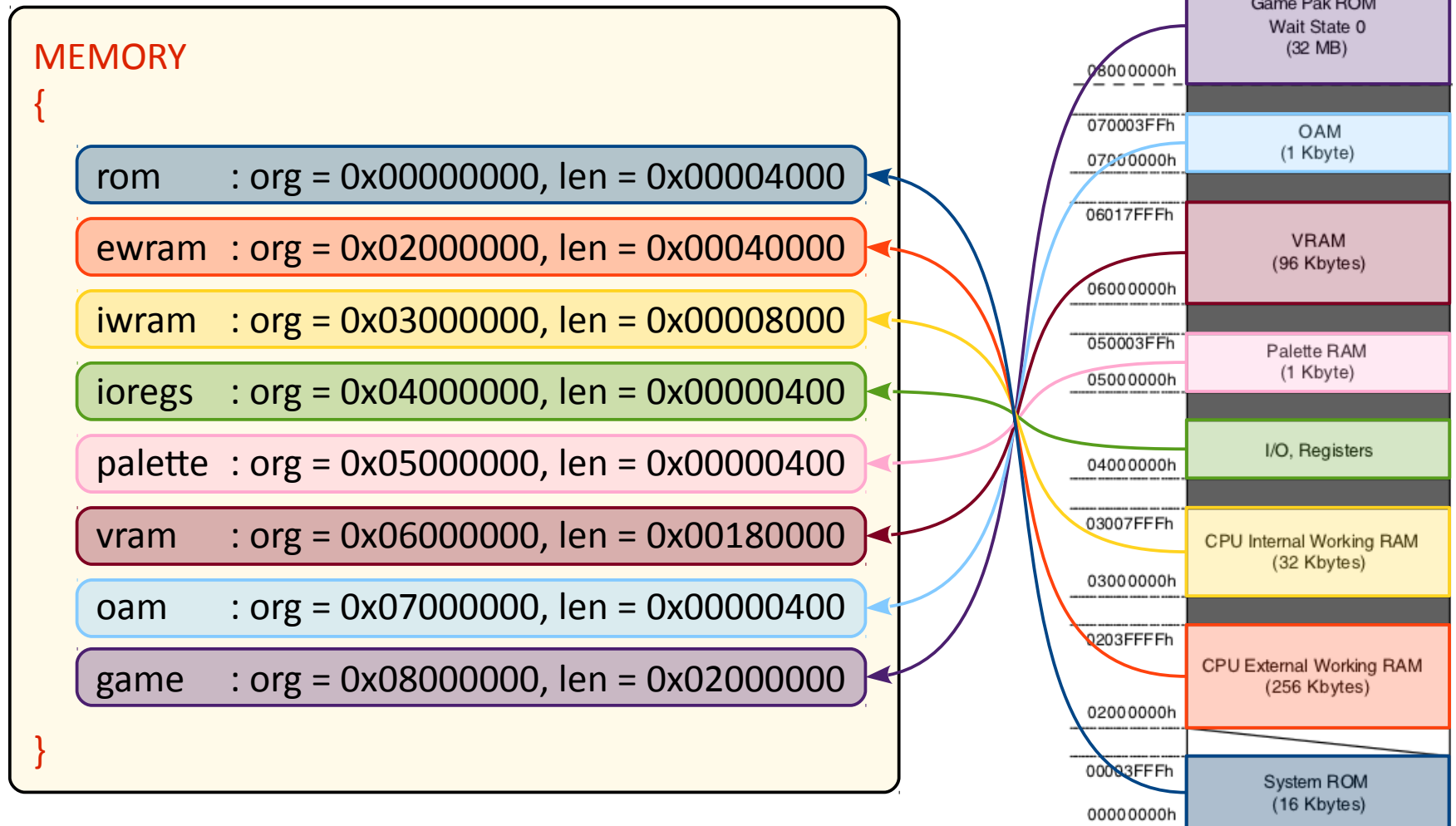
Dirección
de origen

Longitud
en bytes

El desarrollador necesita conocer los diferentes tipos de memorias conectadas al sistema, sus tamaños y sus direcciones. Esta información suele estar en los esquemáticos y las datasheet de los fabricantes

Ejemplo

Mapa de memoria de la Gameboy Advance de Nintendo

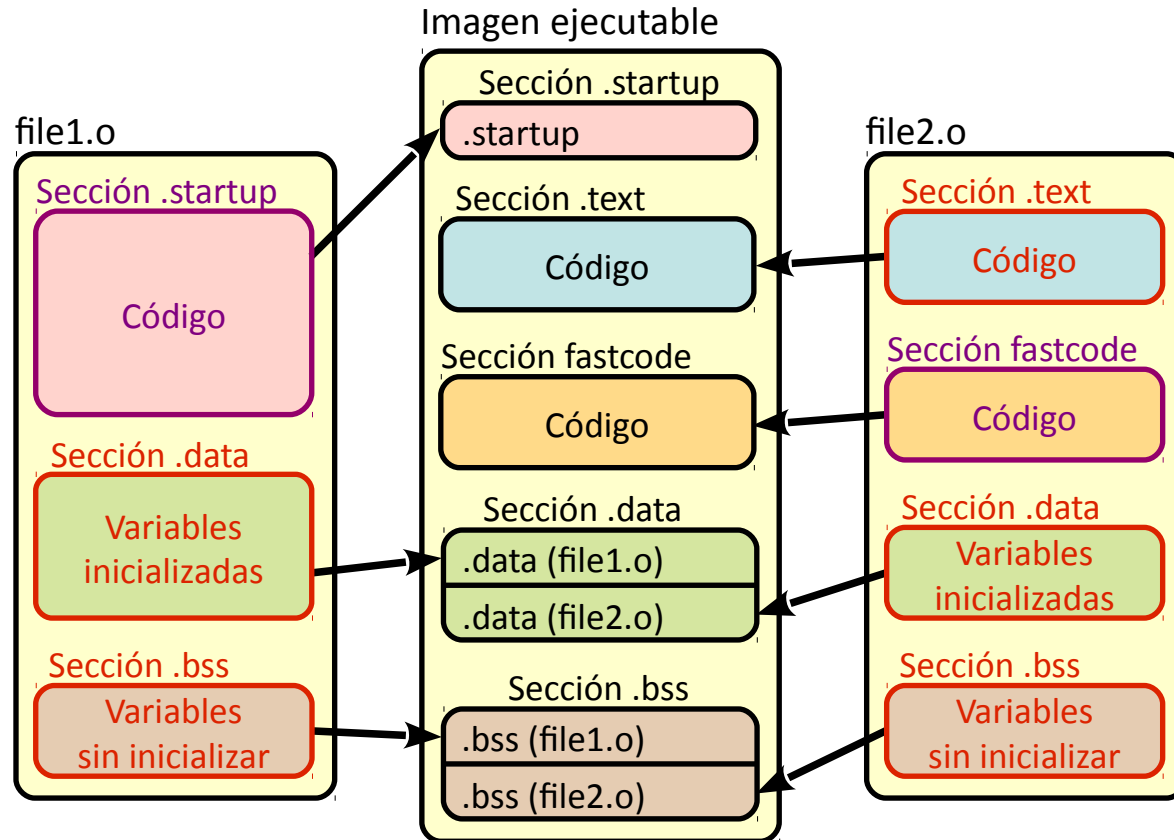


Fuente:

Nintendo of America Inc. AGB Programming Manual, version 1.22, 1999 – 2001.

Mapeo del firmware a la plataforma

Secciones de entrada (input sections)



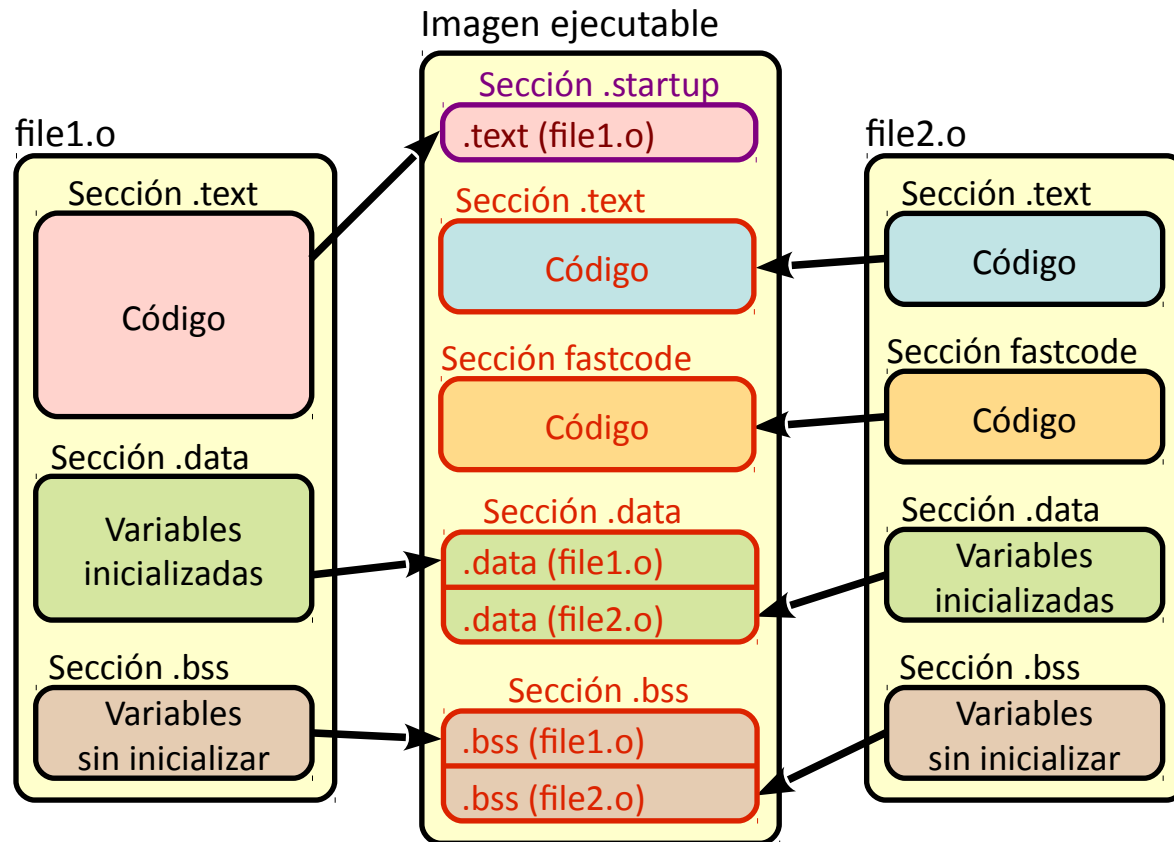
Componen los ficheros objeto que se enlazarán en la imagen final

Normalmente el compilador las genera por defecto (`.text`, `.data`, `.bss`, `.sdata`, `.sbss`)

El desarrollador puede generar secciones a medida para mapear parte de los datos o el código a regiones de memoria particulares, como por ejemplo el código del cargador a la ROM, o las rutinas que se ejecutan habitualmente a la RAM interna al chip

Mapeo del firmware a la plataforma

Secciones de salida (output sections)

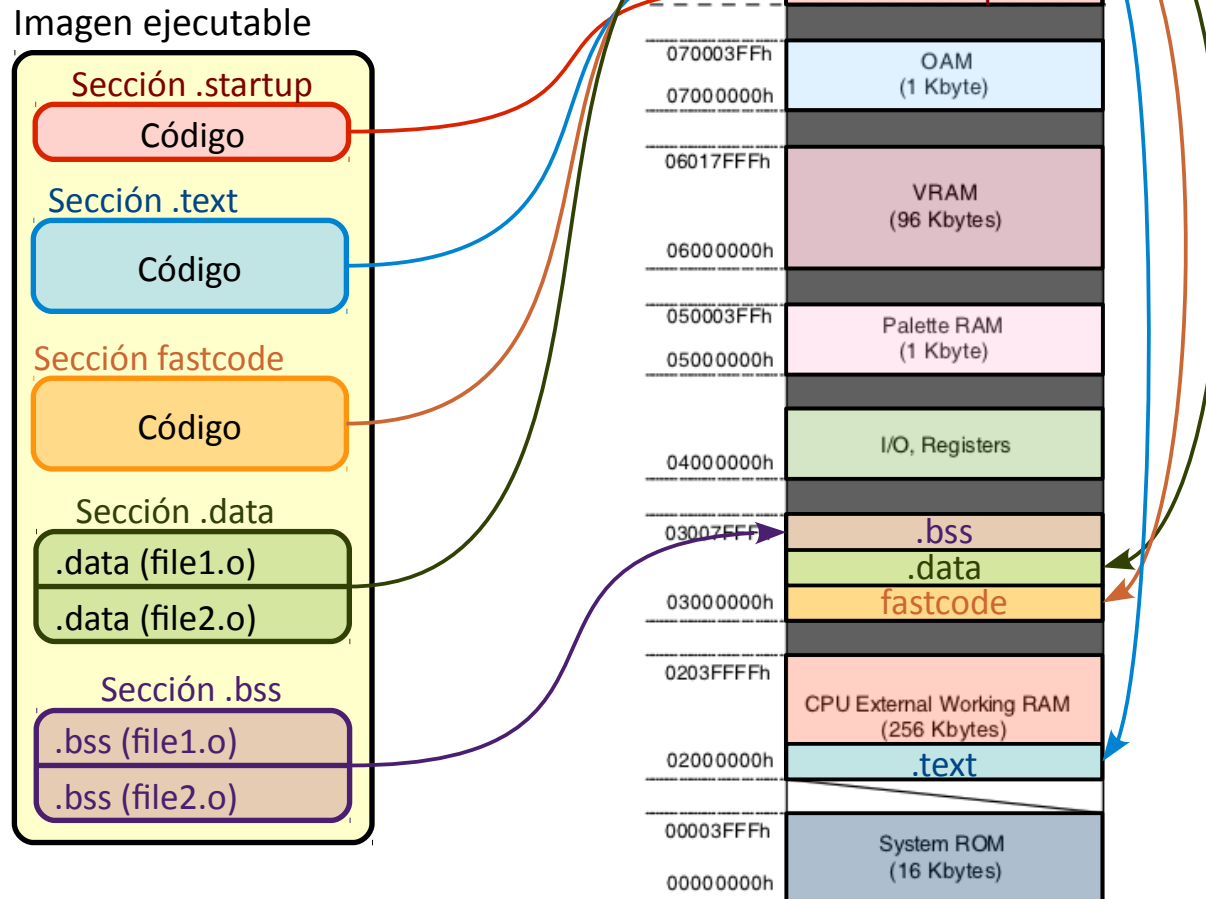


Agrupar las secciones de entrada de los diferentes ficheros objeto

Por defecto, el enlazador agrupa todas las secciones de entrada con el mismo nombre en una única sección de salida, aunque a veces es necesario cambiar este comportamiento

Mapeo del firmware a la plataforma

Secciones de salida (output sections)



Están mapeadas a direcciones concretas del mapa de memoria de la plataforma

A veces es necesario definir secciones de salida a medida para conseguir mapeos adecuados para ciertas secciones de entrada de la imagen

Mapecto del firmware a la plataforma

Secciones de salida (output sections)

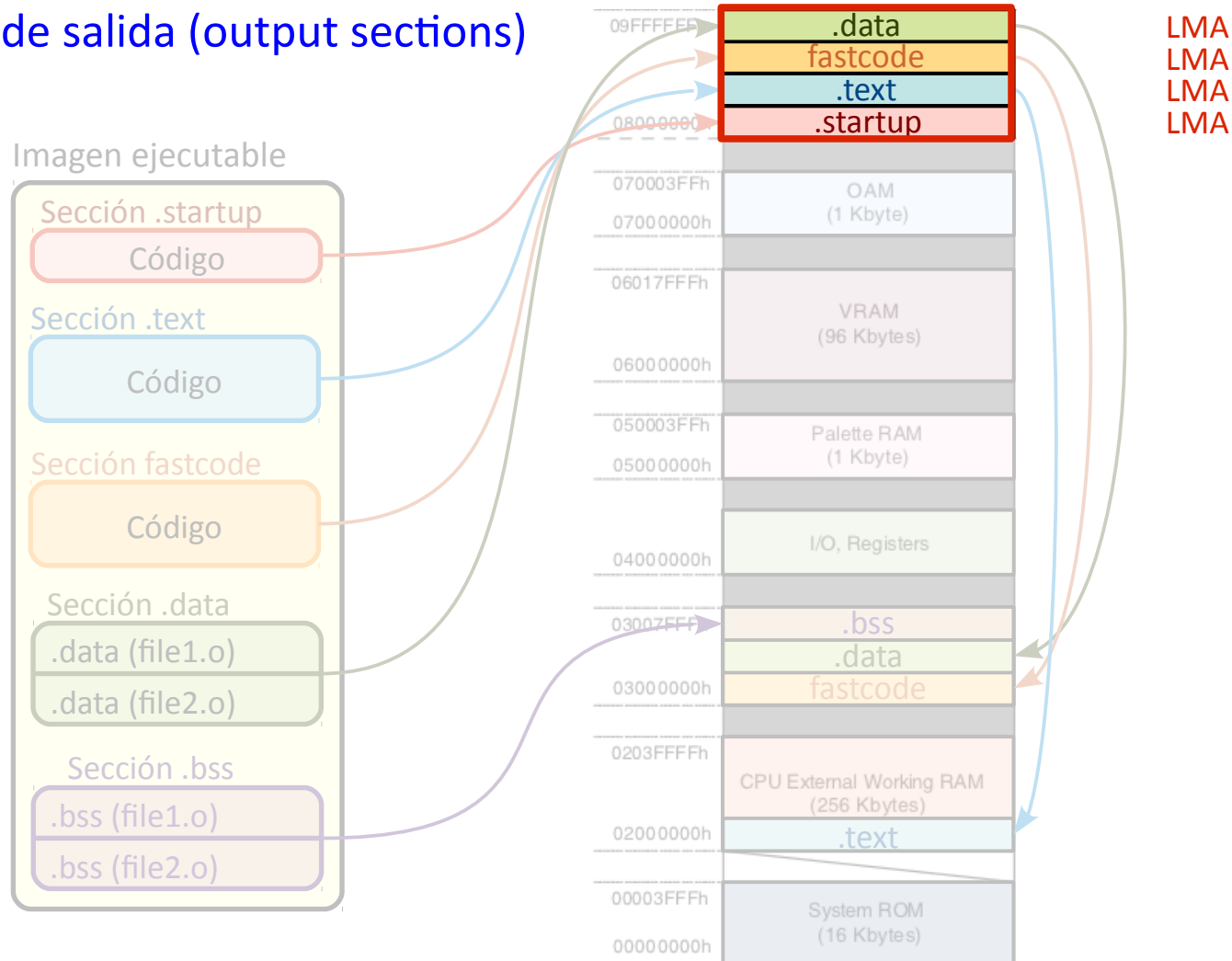
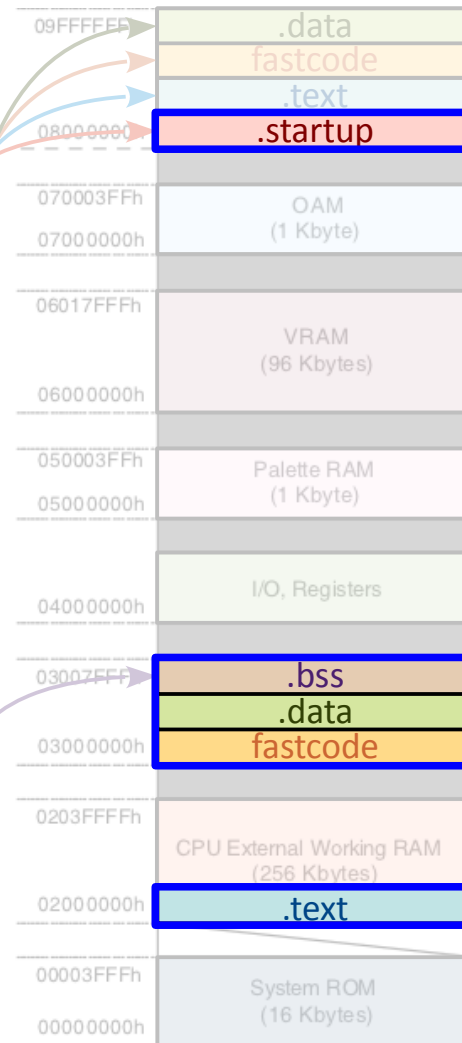
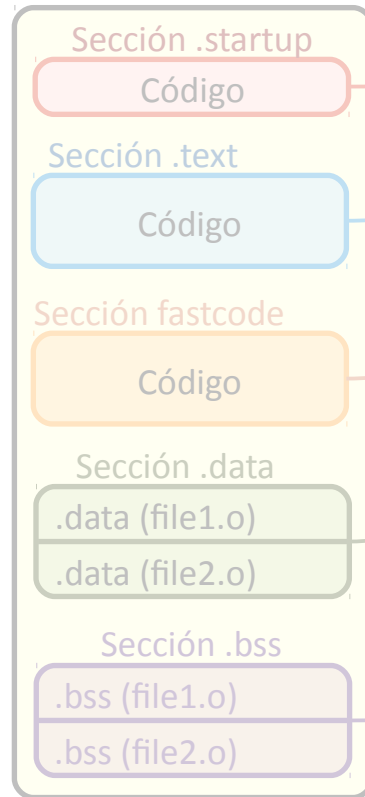


Imagen del firmware que se copiará a la ROM / Flash

Mapecto del firmware a la plataforma

Secciones de salida (output sections)

Imagen ejecutable



LMA
LMA
LMA
LMA = VMA

VMA
VMA
VMA

VMA

Imagen de la aplicación en ejecución

Maapeo del firmware a la plataforma

La directiva `SECTIONS` se utiliza para definir las secciones de salida de la aplicación

```
SECTIONS
{
    sections-command #1
    sections-command #2
    . . .
}
```

Cada línea dentro de la directiva `SECTIONS` puede ser

Una asignación de una dirección a un símbolo

Asignación de símbolos (globales) a direcciones para su uso tanto en el linker script como en el cargador o nuestra aplicación

Una definición de sección de salida

Qué secciones de entrada se deben combinar, en qué orden, y sus direcciones LMA y VMA

Una sección overlay

Si deseamos mapear varias secciones de salida a la misma zona de memoria RAM. Se asume que habrá un gestor de overlays externo que irá copiando cada sección de su LMA a la VMA cuando toque

Mapeo del firmware a la plataforma

Definición de símbolos

`simbolo = valor ;` El punto y coma es obligatorio

Todos los símbolos son globales

Pueden usarse tanto en el cargador como en nuestra aplicación

Expresiones

Asignación: `expresión1 = expresión2;` **Desp. izquierda:** `expresión1 <<= expresión2;`

Suma: `expresión1 += expresión2;` **Desp. derecha:** `expresión1 >>= expresión2;`

Resta: `expresión1 -= expresión2;` **Producto lógico:** `expresión1 &= expresión2;`

Multiplicación: `expresión1 *= expresión2;` **Suma logica:** `expresión1 |= expresión2;`

División: `expresión1 /= expresión2;`

El símbolo '.'

Siempre está definido

Indica el valor actual de la dirección a la que se está emitiendo código

Se puede usar en cualquier expresión y se le puede asignar cualquier valor, siempre que se mueva hacia adelante

Mapeo del firmware a la plataforma

Ejemplos de expresiones

```
. = 0x02000000 ;
```

```
.text :
```

```
{
```

```
*(.text) ;
```

```
}
```

```
.data :
```

```
{
```

```
*(.data) ;
```

```
}
```

```
.bss :
```

```
{
```

```
_bss_start = . ;
```

```
*(.bss) ;
```

```
_bss_end = . ;
```

```
}
```

La primera instrucción empieza en 0x02000000

El cargador puede usar estos símbolos para saber qué zona de memoria debe inicializar a cero

```
. = 0x02000000 ;
```

```
.text :
```

```
{
```

```
*(.text) ;
```

```
. = 0x1000 ;
```

```
}
```

```
.data :
```

```
{
```

```
*(.data) ;
```

```
PROVIDE (edata = .) ;
```

```
}
```

```
.bss :
```

```
{
```

```
*(.bss) ;
```

```
_stack_bottom = . ;
```

```
. += 0x200 ;
```

```
_stack_top = . ;
```

```
}
```

Desplaza el contador 0x1000 a partir del comienzo de la sección

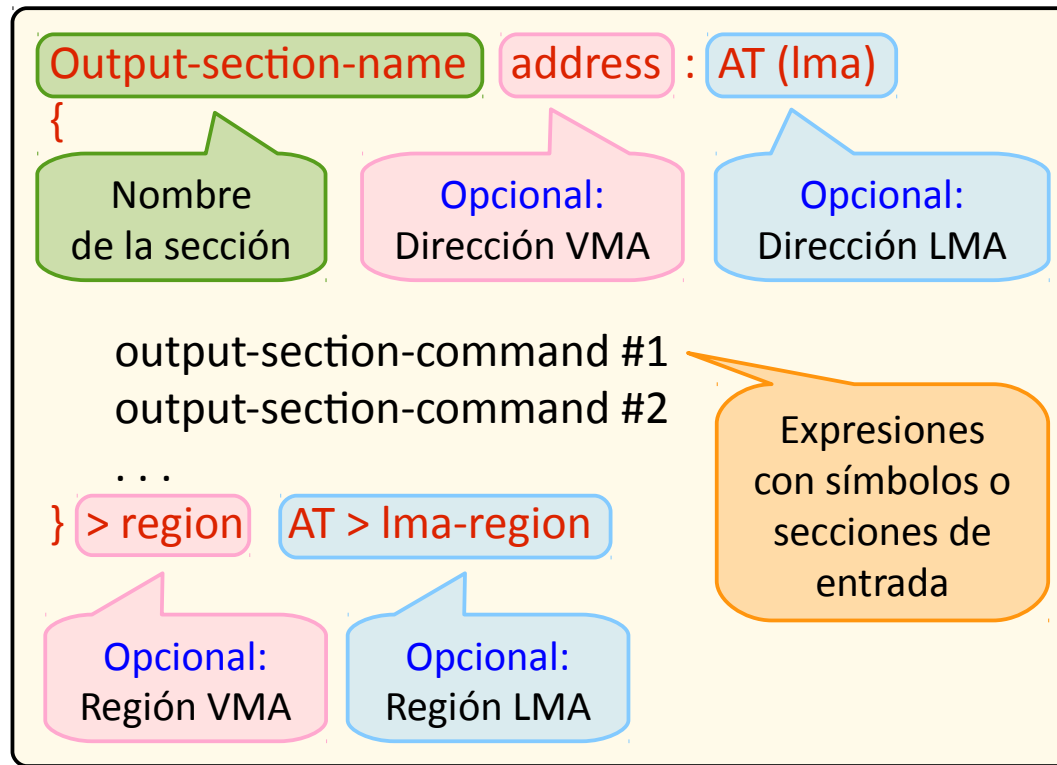
El primer dato empieza en 0x02001000

Define edata sólo si se usa pero no se ha definido en el código

Reservamos una zona de memoria para la pila

Mapeo del firmware a la plataforma

Definición de secciones de salida



Equivalencias

address y region para la dirección VMA

Ima y Ima-region para la dirección LMA

Precedencias

Si se define una dirección y una región para la sección, tendrá prioridad la dirección

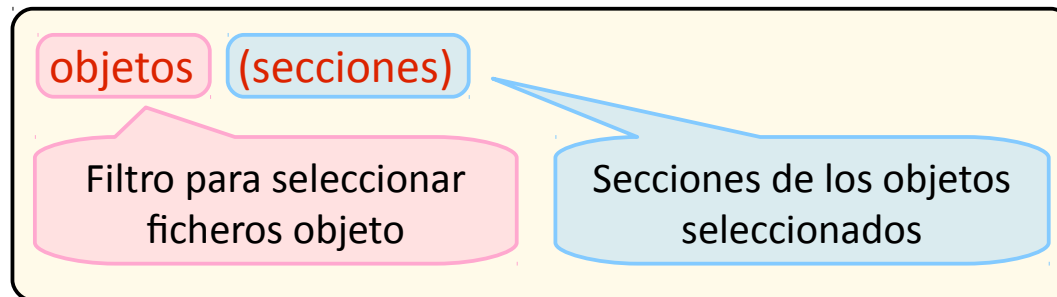
Valores por defecto

Si no se define LMA, se asume que LMA = VMA

Si no se define VMA, se usa el valor actual del contador de posiciones

El valor inicial por defecto para el contador de posiciones es 0

Definición de secciones de entrada



Ejemplos

```
main.o (.text)
```

```
*(.data .bss)
```

```
*(.rodata*)
```

Contenidos

Tema 2: Procesador y mapa de memoria

El procesador

- Motivación

- Introducción a la arquitectura ARMv4T

- Repertorio de instrucciones de la arquitectura ARMv4T

- El hola mundo de un sistema empotrado

- Application Binary Interface* de la arquitectura ARM

El mapa de memoria

- Introducción

- Direcciones de carga y de ejecución

- El formato ELF

- El script de enlazado

- Ejemplos

Definición del mapa de memoria de la plataforma

ENTRY(reset)

MEMORY

```
{  
  rom : org = 0x00000000, len = 0x00000020  
  flash : org = 0x00000040, len = 0x00001000  
  ram : org = 0x00010000, len = 0x00010000  
}
```

SECTIONS

```
{  
  boot :  
  {  
    cargador.o(.text);  
  } > rom  
  .text :  
  {  
    *(.text);  
    mia;  
  } > ram AT > rom  
  .data :  
  {  
    *(.data);  
  } > ram AT > rom  
  .bss :  
  {  
    *(.bss);  
  } > ram  
}
```

Memoria

Dirección

ROM

0x00000000

0x0000001f

Flash

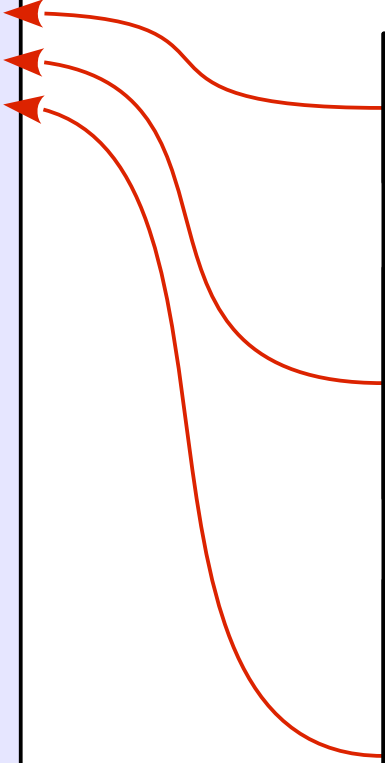
0x00000040

0x0000103f

RAM

0x00010000

0x0001ffff



Definición de las secciones del ejecutable

ENTRY(reset)

MEMORY

```
{  
  rom : org = 0x00000000, len = 0x00000020  
  flash : org = 0x00000040, len = 0x00001000  
  ram : org = 0x00010000, len = 0x00010000  
}
```

SECTIONS

```
{  
  boot :  
  {  
    cargador.o(.text);  
  } > rom  
  .text :  
  {  
    *(.text);  
    mia;  
  } > ram AT > rom  
  .data :  
  {  
    *(.data);  
  } > ram AT > rom  
  .bss :  
  {  
    *(.bss);  
  } > ram  
}
```

cargador.o

Sección .text

Código

Sección .data

Variables
inicializadas

Sección .bss

Variables
sin inicializar

firmware.elf

Sección boot

.text (cargador.o)

Sección .text

.text (main.o)

mia (main.o)

Sección .data

.data (cargador.o)

.data (main.o)

Sección .bss

.bss (cargador.o)

.bss (main.o)

main.o

Sección .text

Código

Sección mia

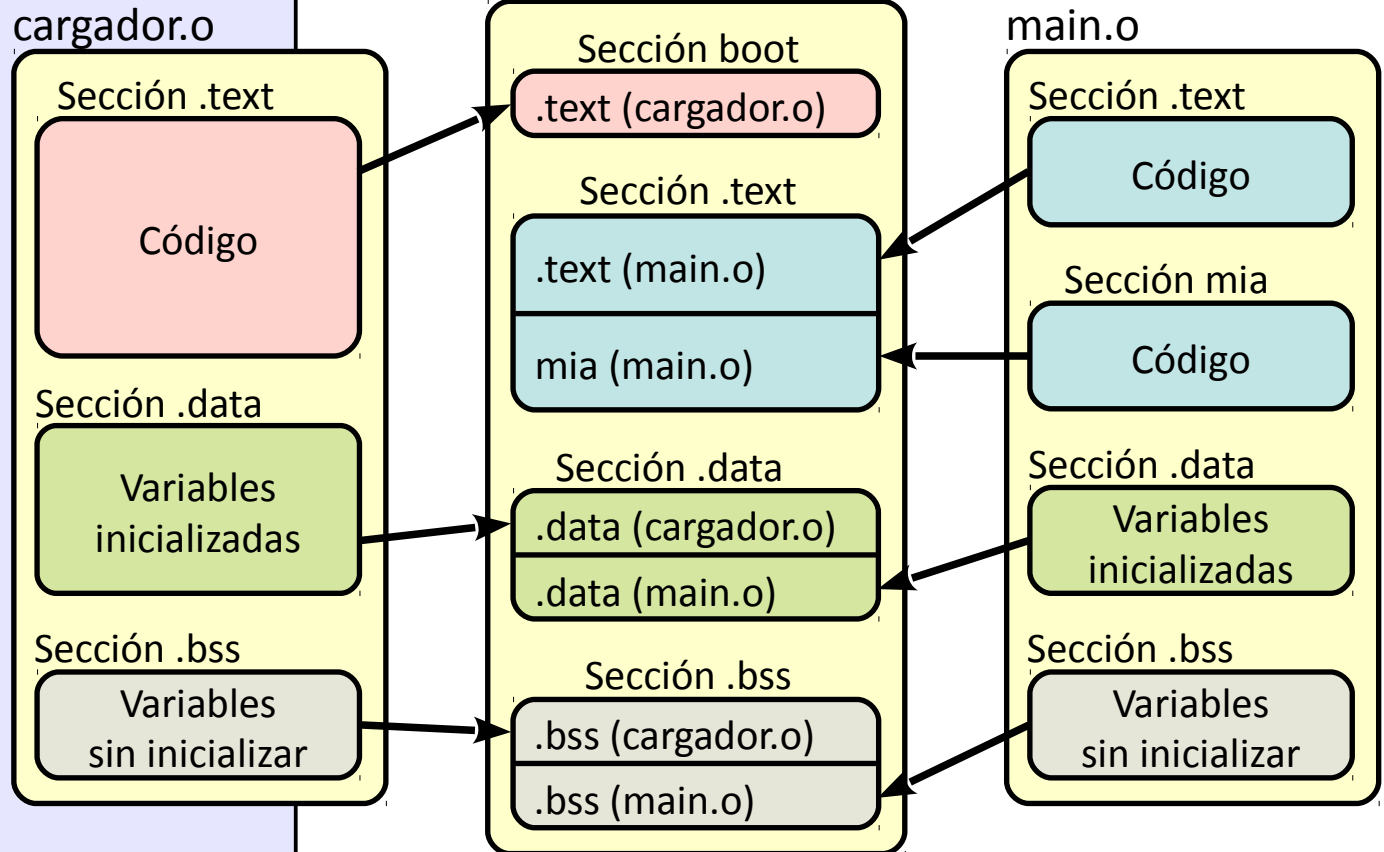
Código

Sección .data

Variables
inicializadas

Sección .bss

Variables
sin inicializar



Definición de las direcciones físicas y virtuales

ENTRY(reset)

MEMORY

```
{
  rom : org = 0x00000000, len = 0x00000020
  flash : org = 0x00000040, len = 0x00001000
  ram : org = 0x00010000, len = 0x00010000
}
```

SECTIONS

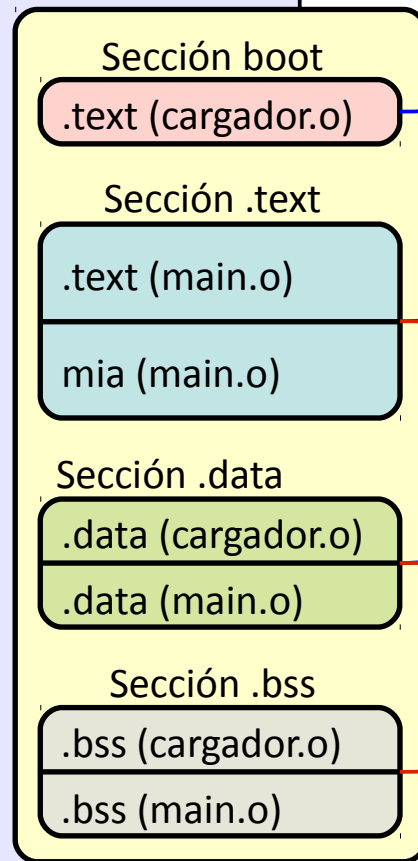
```
{
  boot :
  {
    cargador.o(.text);
  } > rom

  .text :
  {
    *(.text);
    mia;
  } > ram AT > rom

  .data :
  {
    *(.data);
  } > ram AT > rom

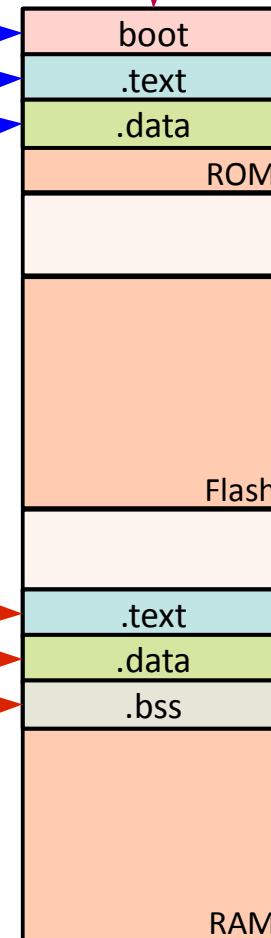
  .bss :
  {
    *(.bss);
  } > ram
}
```

firmware.elf



Direcciones físicas

Direcciones virtuales

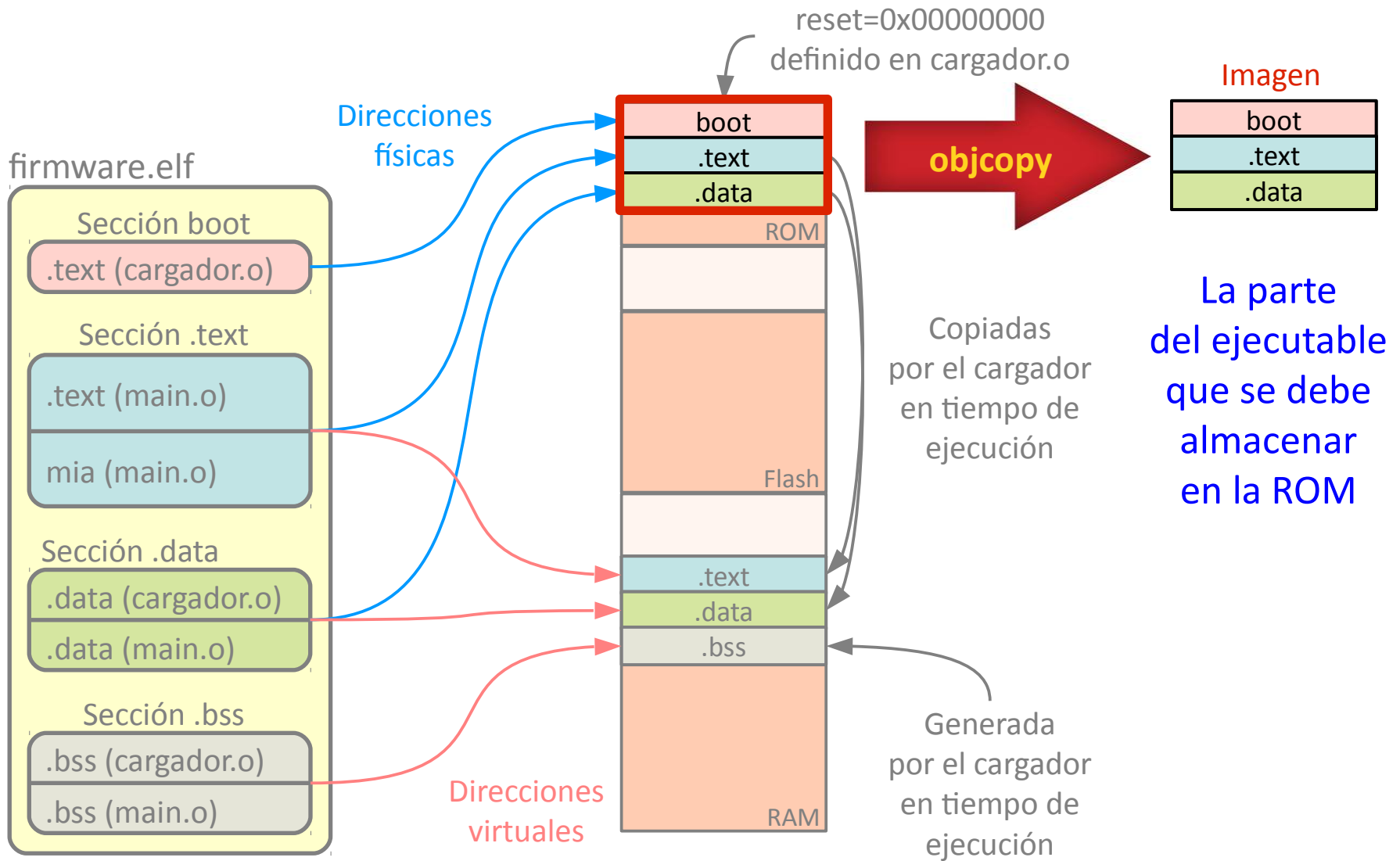


reset=0x00000000
definido en cargador.o

Copiado
por el cargador
en tiempo de
ejecución

Generada
por el cargador
en tiempo de
ejecución

Obtención de la imagen del firmware



Ejemplo: Linker script para la placa AT91EB40A

```
ENTRY(_reset)
```

```
MEMORY
```

```
{
    ram    : org = 0x00000000, len = 0x00040000
    flash  : org = 0x01000000, len = 0x00100000
}
```

```
SECTIONS
```

```
{
    .startup :
    {
        _startup_start = . ;
        *(.startup);
        *(.rodata*);
        . = ALIGN(4);
        _startup_end = . ;
    } > flash

    .vectors : { . += 0x40 ; } > ram

    .text :
    {
        _text_start = . ;
        *(.text);
        _text_end = . ;
    } > ram AT > flash
    _text_flash_start = LOADADDR(.text);
}
```

```
.data :
{
    _data_start = . ;
    *(.data);
    . = ALIGN(4);
    _data_end = . ;
} > ram AT > flash
_data_flash_start = LOADADDR(.data);

.bss :
{
    _bss_start = . ;
    *(.bss);
    . = ALIGN(4);
    *(COMMON);
    . = ALIGN(4);
    _bss_end = . ;
} > ram

_ram_limit = ORIGIN(ram) + LENGTH(ram);
_stack_size = 0x800;

.stack _ram_limit - _stack_size :
{
    _stack_bottom = . ;
    . += _stack_size ;
    _stack_top = . ;
}
}
```

Lecturas recomendadas

Formato ELF:

- Q. Li, C. Yao. *Real-Time Concepts for Embedded Systems*. CMP Books, 2003. Capítulo 2
- Tool Interface Standard (TIS) Comitee. *Executable and Linking Format (ELF) Specification, version 1.2*, 1995. <http://refspecs.linuxbase.org/elf/elf.pdf>
- E. Youngdale. *The ELF Object File Format by Dissection*, 1995
<http://www.linuxjournal.com/node/1060/print>

Mapa de memoria y enlazado:

- L. Edwards. *Embedded System Design on a Shoestring*. Newness, 2003. Capítulo 3
- M. Samek. *Building Bare-Metal ARM Systems with GNU: Part 3 The Linker Script*. Embedded.com, 2007.
- GNU. *GNU Linker Documentation*. <http://sourceware.org/binutils/docs/ld/index.html>
- B. Gatliff. *Embedding with GNU: The GNU Compiler and Linker*. Embedded Systems Programming, 13(2), 2000. <http://www.embedded.com/design/other/4227399/Embedding-with-GNU--The-GNU-Compiler-and-Linker>