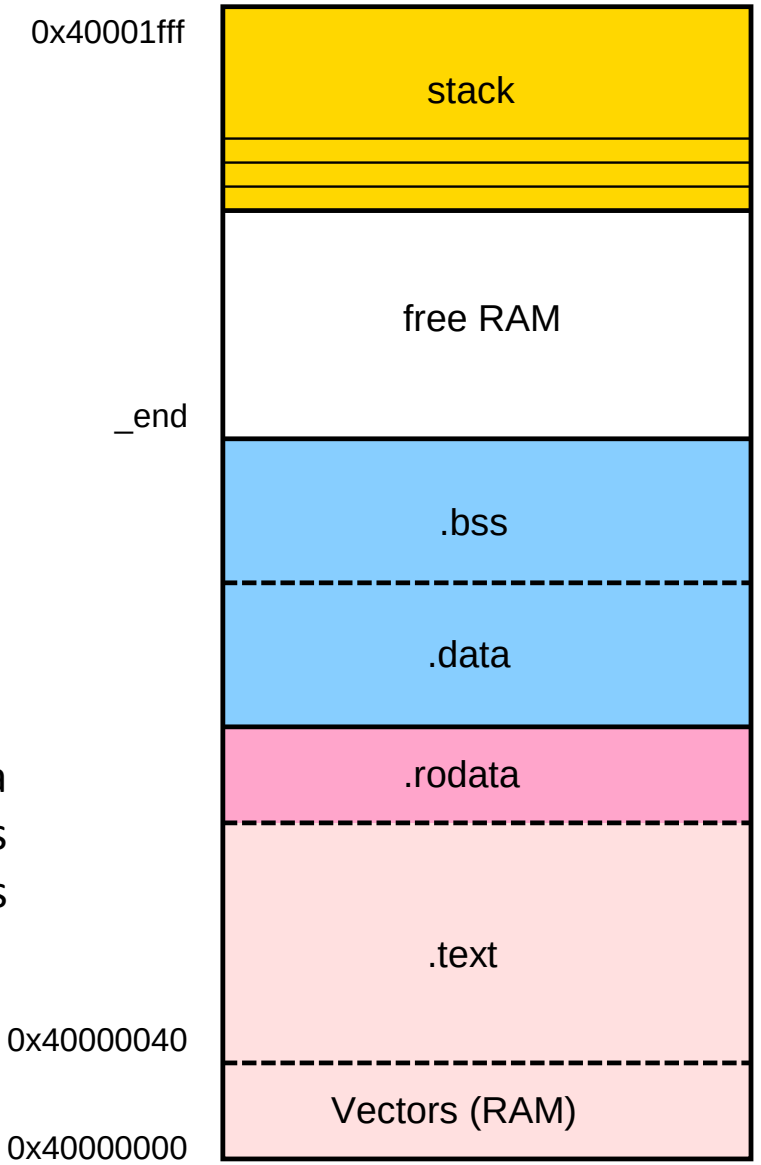
C compiler. Memory map. Program in RAM

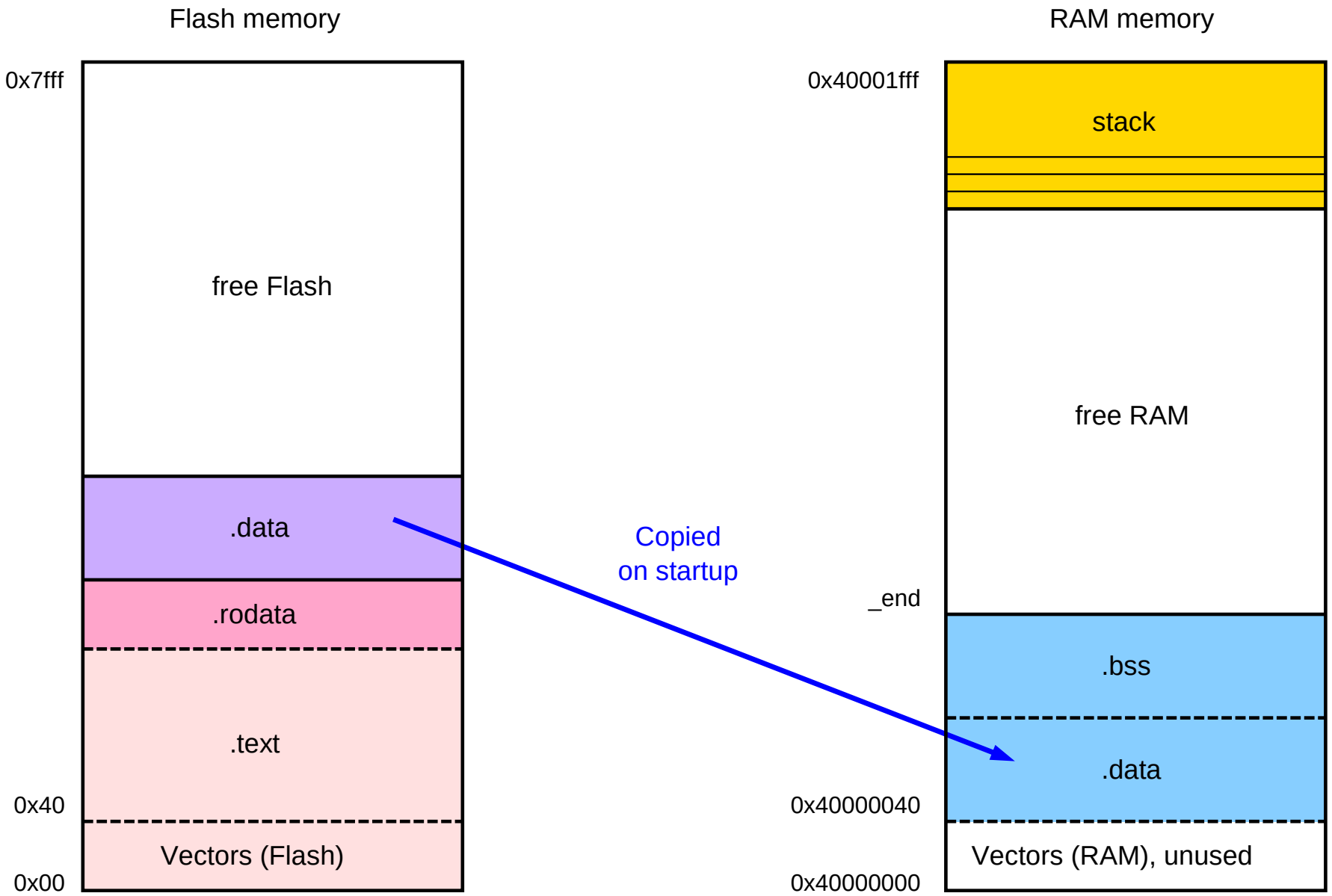
Sections:

- .text: Program code. Read only
- .rodata: constants (const modifier) and strings. Read only
- .data: Initialized global and static variables (startup value ≠ 0)
- .bss: Uninitialized global and static variables (zero value on startup)

The bootloader (bt2.exe) places the .text, .rodata and .data sections into the RAM and then orders the ARM CPU to jump to the reset vector (address 0x40000000)



C compiler. Memory map. Program in Flash



C compiler. Memory map. Program in Flash

- The Flasher program (lpc21isp_148x.exe) writes the .text, .rodata and .data sections into the Flash. This storage is non volatile

- The C runtime startup code (crt.S file) does some processing before calling the “main” routine. In particular it:
  - Copies the .data section into RAM, because variables have to be stored in a read/write memory
  - Fills the .bss section with zeroes

- The free RAM area can be dynamically allocated by providing suitable “malloc” and “free” functions

C compiler. Code components

Explicitly stated:

- crt.S: First code to be executed, written in assembler, including:
  - Reset, interrupt and exception vectors
  - Basic I/O and system initialization (clocks, UART, etc)
  - .data and .bss initialization
  - Call to "main"
  - Simple I/O library (printf like, provided by author, non standard)
- User source code. Must include the "main" function

Implicit:

- libgcc.a: Compiler helper library
  - Missing hardware arithmetic (division,…)
  - Floating point emulation
  - …
- libc.a: Standard C library (stdlib, stdio, string,…)

Libraries:

- libm.a: Mathematical functions (log, sin,…)
- …

C compiler. Command line options

arm-none-eabi-gcc -O2 -g -mcpu=arm7tdmi -nostartfiles –static -Wl,-Tlinker_ram.ld -o code.elf -DCRLF crt.S main.c

- -O2: Optimizer level 2 (0=no optimization, 3=maximum optimization)
- -g: Generate debug information
- -mcpu=arm7tdmi: Generate code for this particular processor
- -nostartfiles: Do not link standard start files. We are providing our own start file (crt.S)
- -static: Do not link dynamic libraries (use .a libraries instead of .so)
- -Wl,-Tlinker_ram.ld: Use the linker_ram.ld linker script. The memory layout, in this case for a RAM stored program, is stated in this file.
- -o code.elf: name of the generated objet file (in ELF format)
- -DCRLF: Define symbol CRLF for preprocessor (used in crt.S)

Linker script (program stored in RAM)

Specify the available memory blocks:

```
MEMORY
{
        flash   : ORIGIN = 0x00000000, LENGTH = 32K   /* FLASH area  */
        ram     : ORIGIN = 0x40000000, LENGTH = 8K    /* RAM area  */
}
```

Specify the section location:

```
SECTIONS
{
        .text :                 /* collect all code related sections */
        {
                *(.text)        /* all .text sections (code)  */
                *(.rodata)      /* all .rodata sections (constants,…) */
                _etext = .;     /* define a global symbol _etext */
        } >ram                  /* put all the above into RAM */

        .data :                 /* all .data sections to RAM  */
        {
                _data = .;      /* symbol marking the .data start  */
                *(.data)        /* all .data sections  */
                _edata = .;     /* symbol marking the .data end  */
        } >ram                  /* put all the above into RAM */
```

Linker script (program stored in RAM)

```
        .bss :                          /* .bss sections to RAM  */
        {
                _bss_start = .;         /*.bss section start*/
                *(.bss)                 /* all .bss sections  */
        } >ram                          /* put all the above in RAM */

        . = ALIGN(4);                   /* align to 32 bit */

        _bss_end = . ;                  /* .bss end symbol */
        _end = .;                       /* end of program RAM symbol */
}
```

Define other symbols:

```
_stack_end = 0x40000000+8K;             /* STACK at RAM end */
```

Linker script (program stored in Flash)

Specify the available memory blocks:

```
MEMORY
{
        flash   : ORIGIN = 0x00000000, LENGTH = 32K   /* FLASH area  */
        ram     : ORIGIN = 0x40000040, LENGTH = 8K-64 /* RAM area   */
}
```

Specify the section location:

```
SECTIONS
{
        .text :                 /* collect all code related sections */
        {
                *(.text)        /* all .text sections (code)  */
                *(.rodata)      /* all .rodata sections (constants,…) */
                _etext = .;     /* define a global symbol _etext */
        } >flash                /* put all the above into flash */

        .data :                 /* all .data sections to RAM  */
        {
                _data = .;      /* symbol marking the .data start  */
                *(.data)        /* all .data sections  */
                _edata = .;     /* symbol marking the .data end  */
        } >ram AT >flash        /* .data will reside in RAM, but it is
                                   stored in the flash */
```

(The rest is the same as in a RAM stored program)

Object file formats

- ELF (Executable and Linkable Format)
  - Default object format for .o, .a and executable files
  - Can include symbol, section, and debug information
  - Standard executable file format in Linux and Solaris
- .hex (hexadecimal file format)
  - Used mainly with flasher programs
  - Specifies block address and data content with hexadecimal digits. Example:

    :10001000 18F09FE50000A0E1F0FF1FE510F09FE5 5C

    Number of data bytes (0x10 = 16)

    Address (0x0010)

    Type of record (0x00: 16-bit address plus data)

    Data (0x18,0xF0,0x9F,0xE5,0x00,0x00,0xA0,0xE1,
    0xF0,0xFF,0x1F,0xE5,0x10,0xF0,0x9F,0xE5)

    Checksum (0x5C)
- .bin (binary file)
  - Single block of binary data
  - No address specified
  - Used with bootloader (bt2.exe)

objcopy -O <format> file.elf file.<format> can convert elf files into ihex or binary

Makefiles

- "make" is an standard Unix tool for compilation. Refer to the make man page
- Specifies the dependences between files and the actions to be taken in order to generate the target files from sources
- Format:

  target: <tab> source1 source2 … sourceN
  <tab> command 1
  <tab> command 2
  …
- It also allows the definition and use of variables. Example:

```
CC = arm-none-eabi-gcc
CFLAGS=-I./ -O2 -g -mcpu=arm7tdmi -nostartfiles –static
…
code.elf:<tab>     crt.S  main.c
<tab>    $(CC)  $(CFLAGS) -Wl,-Tlinker_ram.ld -o $@ -DCRLF crt.S main.c
```

- Special variables:
  - $< input source
  - $@ output target

Note: do not replace <tab> characters with spaces