

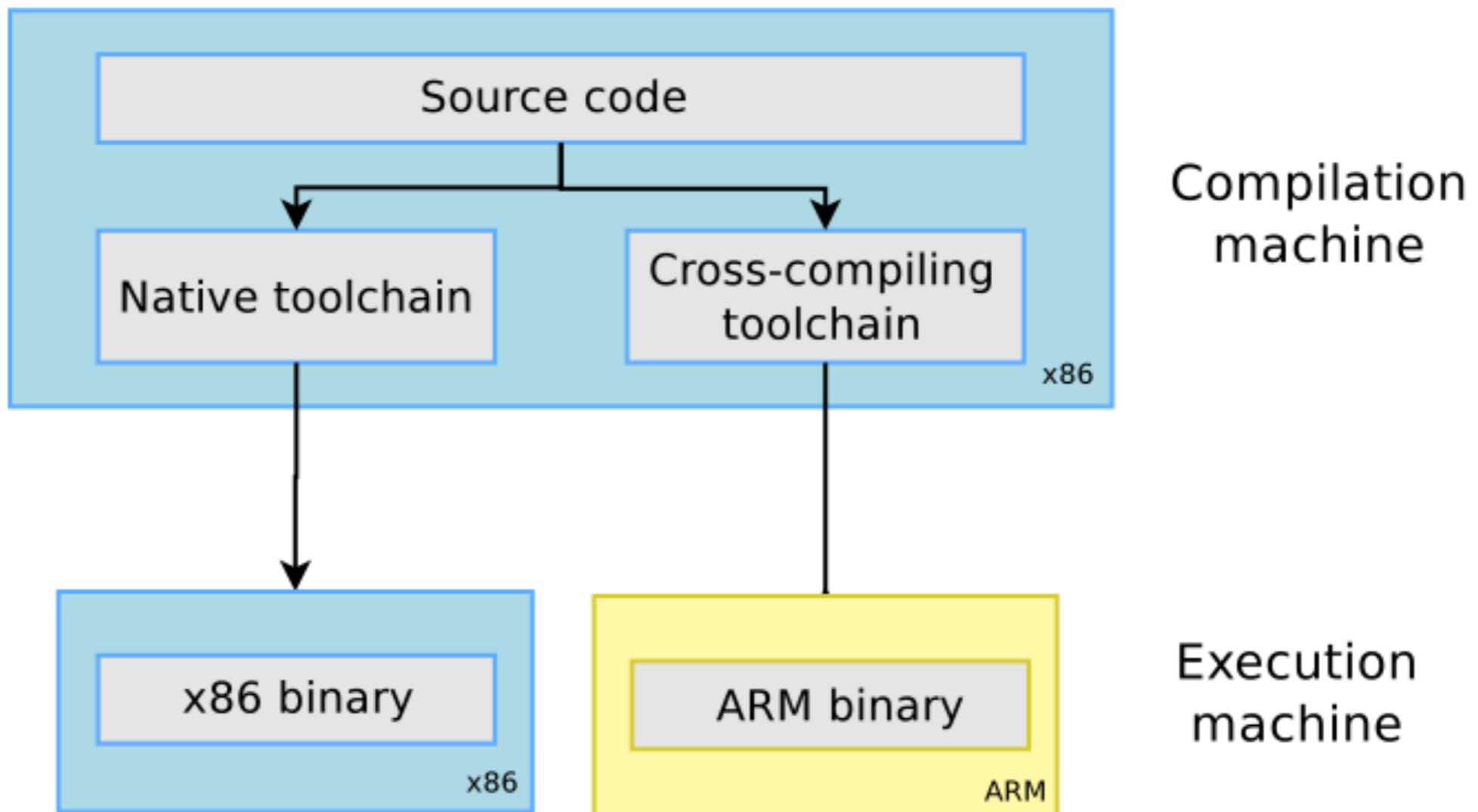
Linker and Startup Code

Lesson 04

Toolchain

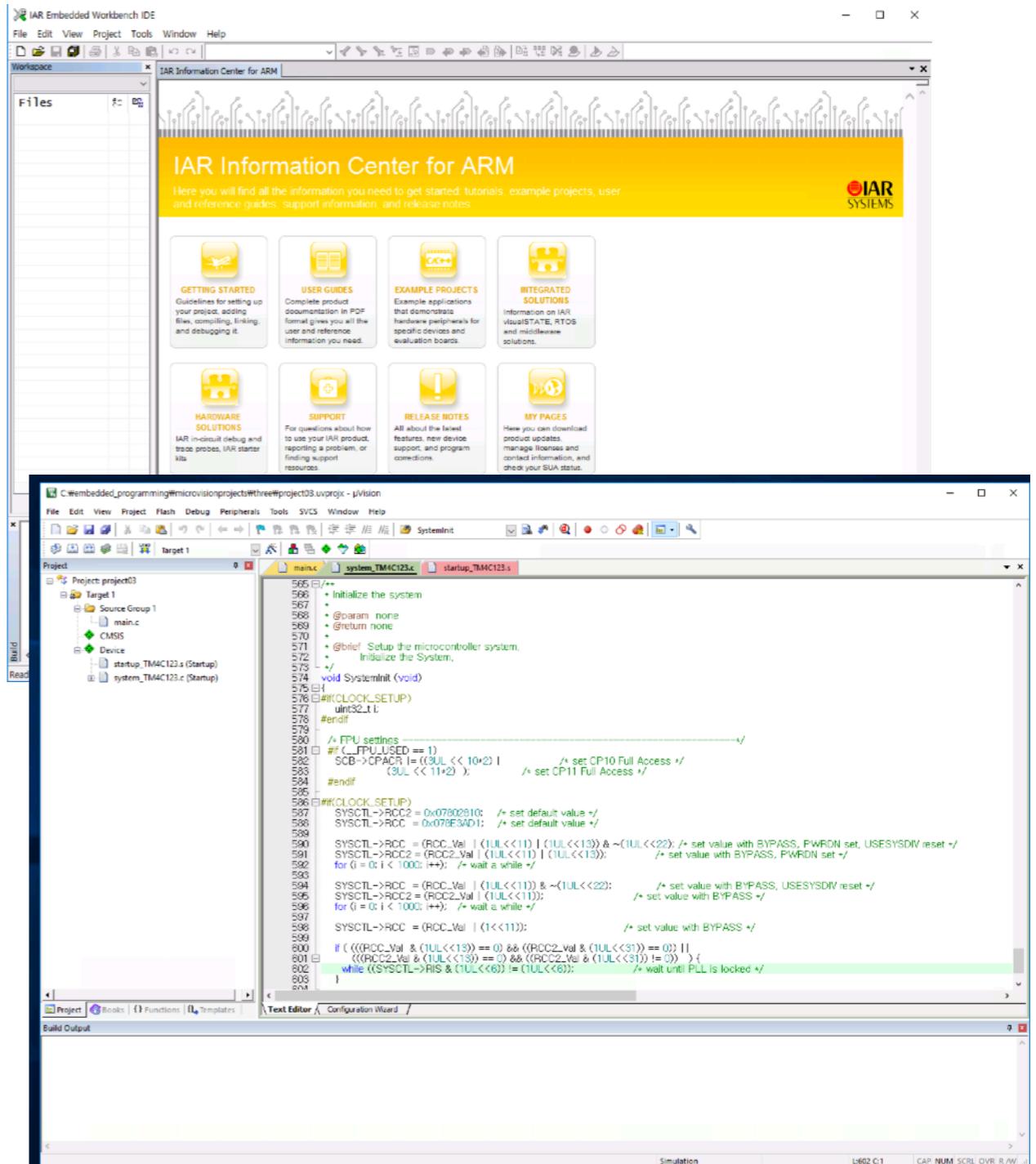
- ⦿ 소스코드를 타겟 디바이스를 위한 실행파일로 컴파일하기 위한 도구들: 컴파일러, 링커, C 라이브러리 등
- ⦿ 보통의 개발 툴들은 native toolchain
 - ⦿ 툴이 실행되는 머신(host)과 컴파일된 코드가 실행되는 머신(target)이 동일 (x86용 실행코드를 생성하는 x86용 컴파일러)
- ⦿ 임베디드 시스템 개발에서는 native toolchain을 사용하기 어려운 경우가 많음
- ⦿ 따라서 cross-compiling toolchain이 사용됨. 즉 host에서 실행되면서 타겟 보드를 위한 코드를 생성
컴파일 한 곳과는 다른 곳에서 작동할 때

크로스 컴파일 툴체인

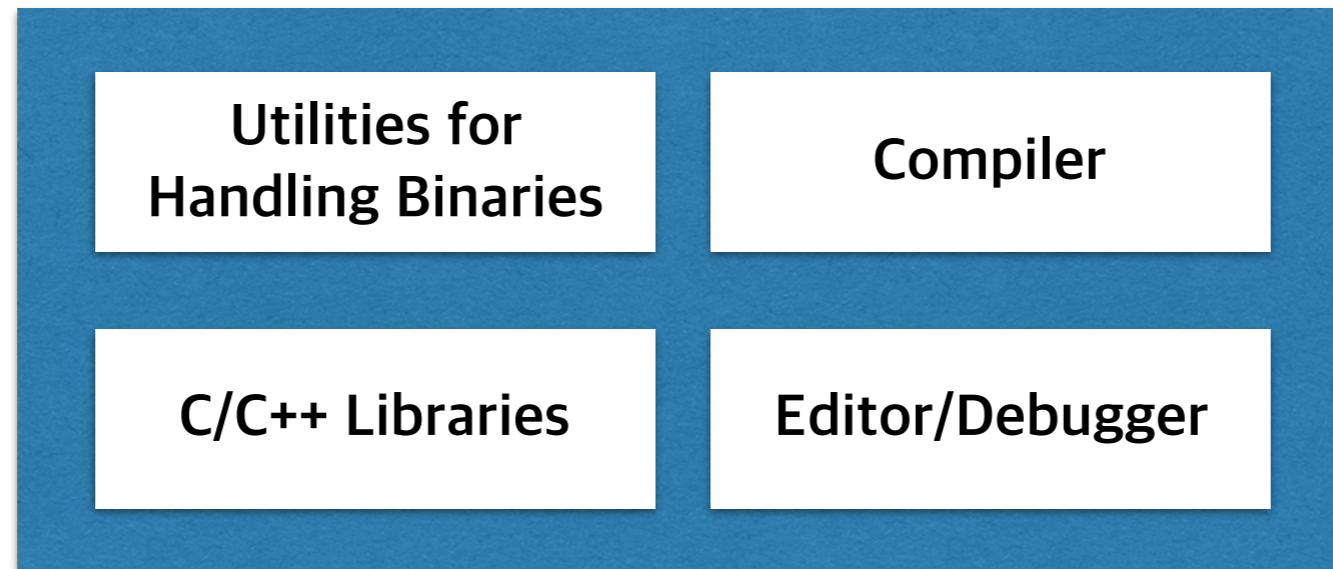


Cross-Development Tools for ARM Cortex M4

- ⦿ **IAR Embedded Workbench**
- ⦿ **Keil MDK-ARM**
- ⦿ **DS-5 Development Studio**
- ⦿ **Code Composer Studio (only for TI processors)**
- ⦿ **GNU Toolchain**



크로스 툴체인의 구성요소



- ⦿ 컴파일러 (**GCC compiler**, ARM C compiler, IAR compiler 등)
- ⦿ 어셈블러, 링커를 비롯한 바이너리를 생성하고 조작하기 위한 **다양한 유틸리티들**
- ⦿ **C/C++ library**
- ⦿ **Editor/Debugger** 등

GCC (GNU Compiler Collection)

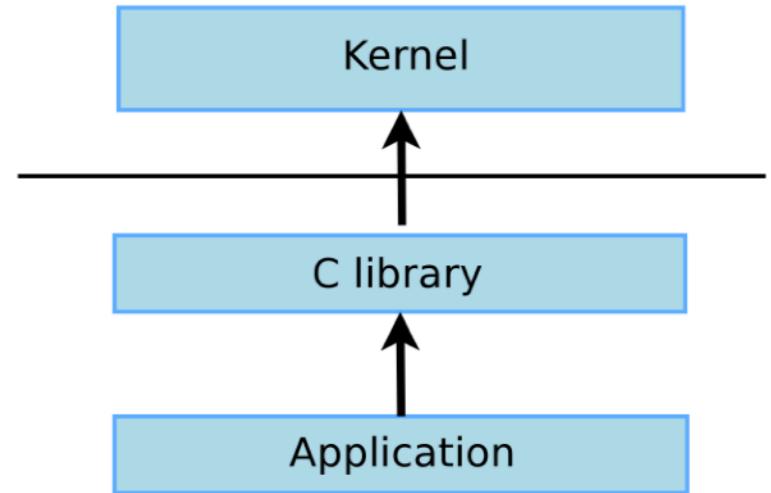
GNU : GNU is not unix

- ⦿ Free, Open source
- ⦿ 다양한 언어들을 지원: C, C++, Ada, Fortran, Java, Objective-C, Objective-C++
- ⦿ 다양한 CPU 아키텍처를 지원: ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86_64, IA64, Xtensa 등.
- ⦿ 다양한 IDE에서 사용 가능 (Keil MDK, Code Composer Studio 등)
- ⦿ <http://gcc.gnu.org/>

- ⦿ Binutils는 타겟 CPU 아키텍처를 위한 바이너리를 생성하고 조작하기 위한 다양한 툴들의 집합체
 - ⦿ as: 어셈블러(assembler) cc: 컴파일러(compiler)
 - ⦿ ld: 링커(linker)
 - ⦿ ar, ranlib: 라이브러리를 생성하고 다루기 위한 도구들
 - ⦿ objdump, readelf, size, nm, strings: 바이너리 파일을 검사/분석하는 도구들
 - ⦿ strip: 파일 크기를 줄이기 위해 바이너리로 부터 불필요한 부분을 제거하는 유틸리티
- ⦿ <http://www.gnu.org/software/binutils/>
- ⦿ GPL 라이센스

C library

- C 라이브러리는 Linux 시스템의 필수 요소
- 응용 프로그램과 커널간의 인터페이스
- 표준 C API를 응용 프로그램에게 제공
- **가용한 C 라이브러리들 :**
 - ^{Light} glibc, uClibc, eglIBC, dietlibc, newlib 등
- cross toolchain을 생성하는 시점에 C 라이브러리를 선택해야 함



Toolchain 옵션: ABI

- ⦿ 툴체인을 빌드할 때 생성할 바이너리의 ABI를 지정해야
- ⦿ Application Binary Interface (ABI):
 - ⦿ 데이터 타입의 크기, layout, alignment 등을 지정
little endian, big endian
 - ⦿ 함수 호출 방법을 지정: 매개변수 전달 방법, 리턴값 전달방식 등
 - ⦿ 시스템호출 방식 등
- ⦿ 하나의 시스템에서 모든 바이너리는 동일한 ABI를 가짐
- ⦿ ARM의 경우 두 가지 ABI가 존재: OABI와 EABI
 - ⦿ 최근엔 대부분 EABI 사용
- ⦿ MIPS의 경우 몇몇 ABI가 존재: o32, o64, n32, n64
- ⦿ http://en.wikipedia.org/wiki/Application_Binary_Interface

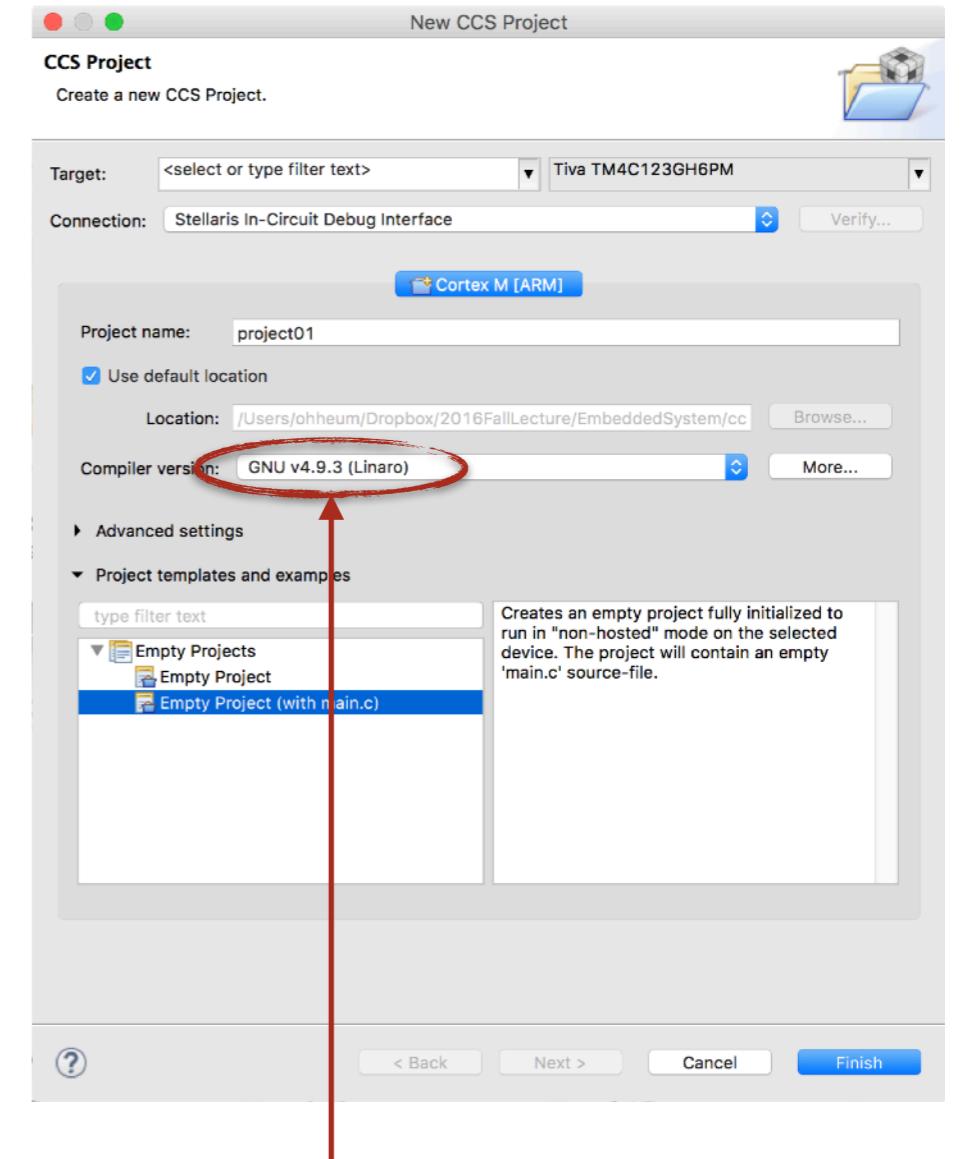
프로그램과 운영체제간의 낮은 수준의
인터페이스 (기계어 수준에서)

Toolchain 옵션: 부동소수점 연산 지원 여부

- ⦿ 프로세서에 따라 floating point unit이 있을 수도 있고 없을 수도 있음
 - ⦿ 예를 들어 다수의 ARMv4 및 ARMv5 CPU들은 floating point unit이 없음. ARMv7 이후에는 모두 VFP (Vector Floating-Point) unit을 가짐
- ⦿ FPU를 가진 프로세서의 경우 툴체인은 floating point 연산 명령어를 직접 사용하는 hard float 코드를 생성해야
- ⦿ FPU가 없는 프로세서의 경우:
 - ⦿ hard float code를 생성한 후 커널 수준에서 floating point 연산 명령어를 emulate하거나 (매우 느림)
 - ⦿ soft float code를 사용. 즉 user space 라이브러리를 호출하여 연산

GCC 툴체인을 구하는 3가지 방법

- ⦿ GCC 소스코드로부터 수작업으로 구성하는 방법
 - ⦿ 복잡하고 어려움
- ⦿ 미리 컴파일되어 배포되는 버전을 사용하는 방법
 - ⦿ SoC 혹은 보드 제작사에 의해서 제공되는 버전
 - ⦿ Sourcery CodeBench toolchains
 - ⦿ **Linaro toolchains (<https://www.linaro.org>)**
 - ⦿ 참조 : <http://elinux.org/Toolchains>
- ⦿ 툴체인 빌드 유틸리티 사용하는 방법



이 수업에서는 CCS + Linaro 컴파일러를 사용하고 있음

리나로

[숨기기]

위키백과, 우리 모두의 백과사전.



이 문서는 [위키백과의 편집 지침](#)에 맞춰 다듬어야 합니다.

더 좋은 문서가 되도록 [문서 수정을 도와주세요](#). 내용에 대한 의견이 있으시다면 [토론 문서](#)에서 나누어 주세요. (2012년 1월 25일)

리나로(Linaro)는 ARM 기반의 리눅스 오픈 소스를 지향하는 엔지니어 중심의 비영리 단체로서 2010년 6 월에 대만의 Computex에서 후원 멤버인 ARM, Freescale Semiconductor, IBM, Samsung, ST-Ericsson과 Texas Instruments가 연합하여 설립되었다 [1]

리나로는 ARM 기반의 시스템 온 칩 (SoC) 관련 리눅스 open source의 통합 및 최적화를 목적으로 매월 마지막주 목요일 단위로 전 세계의 숙련된 엔지니어들에 의하여 제작된 솔루션 및 툴체인을 배포하며 리눅스 mainline으로 upstream 과정을 통해 Linaro 멤버들의 시스템 온 칩 (SoC) 기반으로 작업하는 OEM/ODM에게 항상 최신의 안정적인 버전의 리눅스 소프트웨어를 사용할 수 있도록 지원하고 있다. [2]

리나로는 ARM 기반의 모든 core용 소프트웨어를 개발, 배포한다. 특히 Cortex-A8 또는 dual-core Cortex-A9 processor(s)기반의 시스템 온 칩 (SoC)를 위한 리눅스 코드의 최적화를 지향하고 있다. 세계적으로 숙련된 엔지니어 조직은 매월 단위로 Technical Steering Committee. [3]에서 정의되는 요구사항들을 기반으로 리눅스 upstream project를 진행하고 있으며, 매월 마지막주 목요일에 1개월 단위의 결과물과 toolchain을 wiki.linaro.org를 통해 배포하고 있다. 이를 위해 Kernel, Multimedia, Power Management, Graphics, Tool Chain Working Group의 코어 엔지니어링 그룹들이 ARM 기반의 임베디드 Linux software의 통합 및 최적화 엔지니어링을 통해 안정적이고, 최적화된 솔루션 및 ToolChain 뿐 아니라 Test Tool 제공과 실시간 software 무결성을 검증하는 것을 그 목적으로 하고 있다. [4]

목차 [숨기기]

1 역사

2 배포시기

리나로

종류	비영리단체
설립시기	2010년 6월
회원사	Samsung, ARM, Freescale Semiconductor, IBM, ST-Ericsson, Texas Instruments
프로그램방법	C, C++, ASM
운영 체제	리눅스
현상황	12.10 공개
최초 배포	2010년 11월 10일 (10.11)
지원범위	Mobile, Mobile computing, Digital Home, Infotainment
지원하는 플랫폼	ARMv7A
리눅스 종류	Linux
웹사이트	www.linaro.org

target, kernel

arm-linux-gnueabi

Toolchains for little-endian, soft-float, 32-bit ARMv7 (and earlier) for GNU/Linux systems

- gcc-linaro-*x86_64_arm-linux-gnueabi.tar.xz
 - Linux 64-bit binaries for the ARMv7 Linux soft float cross-toolchain
- gcc-linaro-*i686-mingw32_arm-linux-gnueabi.tar.xz
 - Windows 32-bit binaries for the ARMv7 Linux soft float cross-toolchain

arm-linux-gnueabihf

Toolchains for little-endian, hard-float, 32-bit ARMv7 (and earlier) for GNU/Linux systems

- gcc-linaro-*x86_arm-linux-gnueabihf.tar.xz
 - Linux 32-bit binaries for the ARMv7 Linux hard float cross-toolchain
- gcc-linaro-*x86_64_arm-linux-gnueabihf.tar.xz
 - Linux 64-bit binaries for the ARMv7 Linux hard float cross-toolchain
- gcc-linaro-*i686-mingw32_arm-linux-gnueabihf.tar.xz
 - Windows 32-bit binaries for the ARMv7 Linux hard float cross-toolchain

arm-none-eabi

Toolchains for little-endian, soft-float, 32-bit ARMv7 (and earlier) for bare-metal systems

- gcc-linaro-*x86_64_arm-eabi.tar.xz
 - Linux 64-bit binaries for the ARMv7 bare-metal cross-toolchain
- gcc-linaro-*i686-mingw32_arm-eabi.tar.xz
 - Windows 32-bit binaries for the ARMv7 bare-metal cross-toolchain

↑
이 수업에서 사용하는 컴파일러이다. bare-metal system은 운영체제가 없는 시스템을 의미한다.

<https://wiki.linaro.org/WorkingGroups/ToolChain/FAQ>

Does Linaro gcc toolchain target glibc or eglibc?

- eglibc is a fork of glibc, but they are mostly inter-changeable.
- eglibc allows you to select which features you want in your C library, and so be able to produce a smaller C library.
- We do not currently provide a “Linaro glibc”, and instead use upstream directly for binary releases.
- Currently, we base our releases on eglibc-2.19 which is compatible with glibc2.19. Android releases target Bionic (Android’s (incompatible) replacement for (e)glibc) instead.
- We distribute binaries using eglibc because that is what most distributions use for historical reasons, and also because it allows users to build a more slimmed down C library.

Due to the end of development of eglibc, we will transition to glibc beginning version 2.20.

What is the differences between “arm-none-eabi-” and “arm-linux-gnueabihf”? Can I use “arm-linux-gnueabihf” tool chain in bare-metal environment? How do you know which toolchain binary to use where?

The general form of compiler/linker prefix is as follows:

A-B-C

Where:

- A indicates the target (arm for AArch32 little-endian, aarch64 for AArch64 little-endian).
- B indicates the vendor (none or unknown for generic). Note that this is optional (Eg: not present in arm-linux-gnueabihf)
- C indicates the ABI in use (linux-gnu* for Linux, linux-android* for Android, elf or eabi for ELF based bare-metal).

컴파일러의 명칭의 의미를 설명한다.

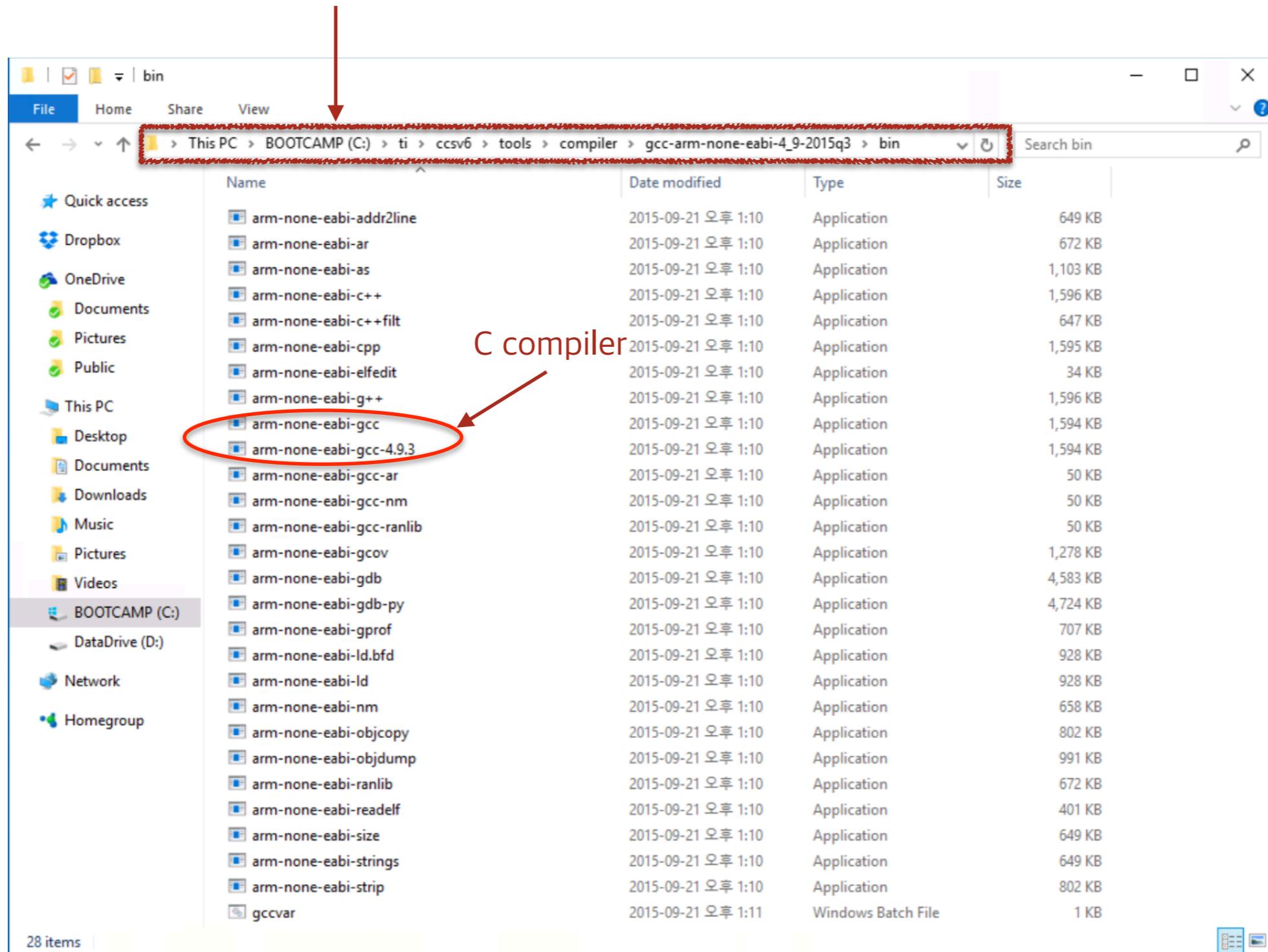
C has values which seem odd until you understand the history behind it (basically AArch32 used to have a linux-gnu ABI which got changed so needed a new name so we have linux-gnueabi). For AArch32 we have linux-gnueabi and linux-gnueabihf which indicate soft float, and hard float respectively.

The bare-metal ABI will assume a different C library (newlib for example, or even no C library) to the Linux ABI (which assumes glibc). Therefore, the compiler may make different function calls depending on what it believes is available above and beyond the Standard C library.

Also the bare-metal ABI and Linux ABI for the 32-bit Instruction sets make different assumptions about the storage size of enums and wchar_t which you have to be careful of (not a complete list). And the difference between the 32-bit and 64-bit ABIs are also numerous and subtle (the obvious example being pointer sizes).

GCC 설치 디렉토리

CCS 설치 디렉토리 내에 GCC가 설치되어 있다.



Binary Tools

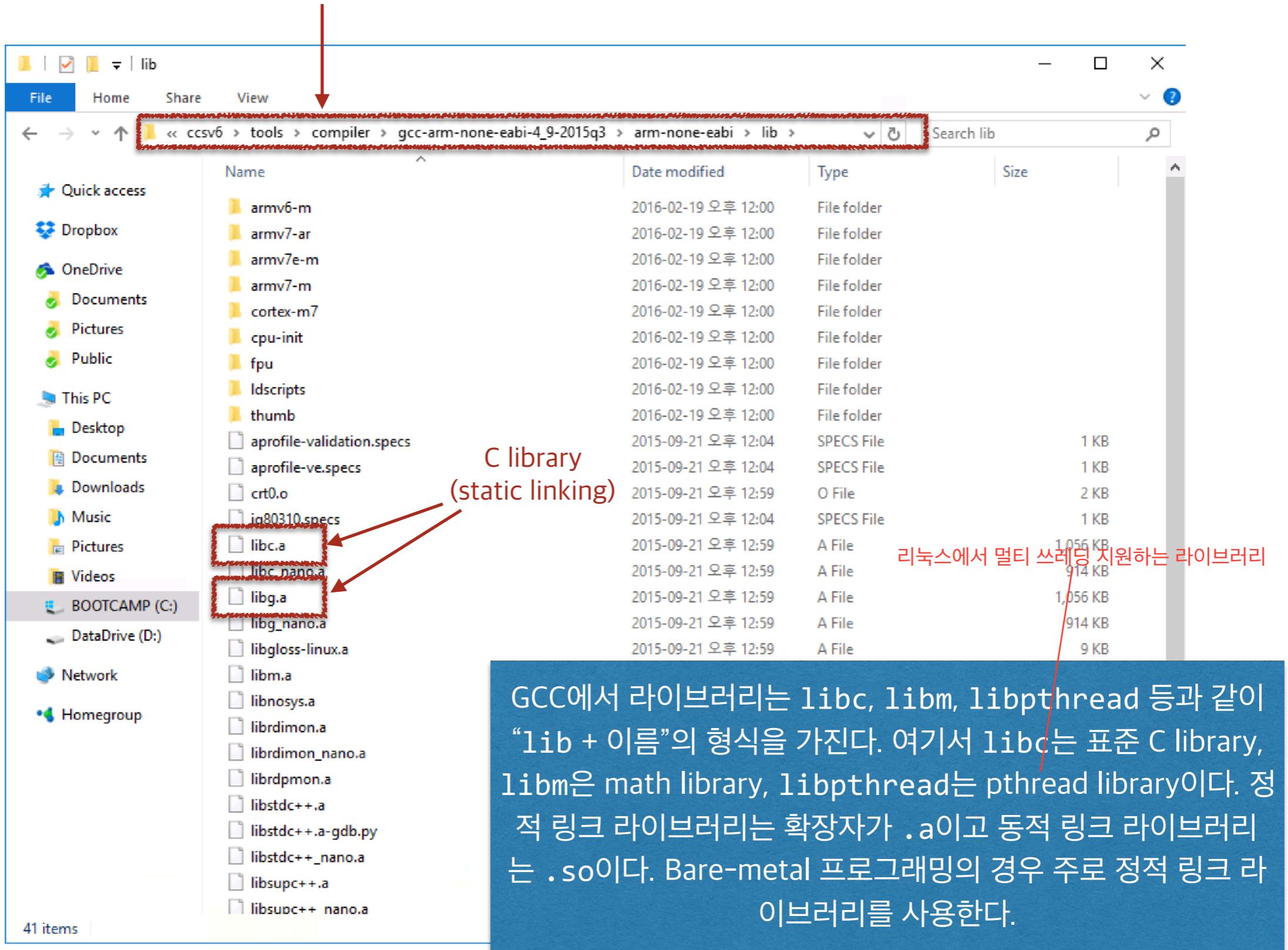
Command	Description
addr2line	Converts program addresses into filenames and numbers by reading the debug symbol tables in an executable file. It is very useful when decoding addresses printed out in a system crash report.
ar	The archive utility is used to <u>create static libraries</u> .
as	This is the GNU <u>assembler</u> .
c++filt	This is used to demangle C++ and Java symbols.
cpp	This is the C preprocessor, and is used to expand #define, #include, and other similar directives. You seldom need to use this by itself.
elfedit	This is used to update the ELF header of ELF files.
g++	This is the GNU C++ front-end, which assumes source files contain C++ code.
gcc	This is the GNU C front-end, which assumes source files contain C code.

Binary Tools (계속)

Command	Description
gcov	This is a code coverage tool.
gdb	This is the GNU debugger.
gprof	This is a program profiling tool.
ld	This is the GNU <u>linker</u> .
nm	This lists symbols from object files.
objcopy	This is used to <u>copy and translate object files</u> .
objdump	This is used to <u>display information from object files</u> .
ranlib	This creates or modifies an index in a static library, making the linking stage faster.
readelf	This <u>displays information about files in ELF object format</u> . 컴파일된 object code들은 ELF 포맷을 따른다.
size	This lists section sizes and the total size.
strings	This display strings of printable characters in files.
strip	This is used to strip an object file of debug symbol tables, thus making it smaller. Typically, you would strip all the executable code that is put onto the target.

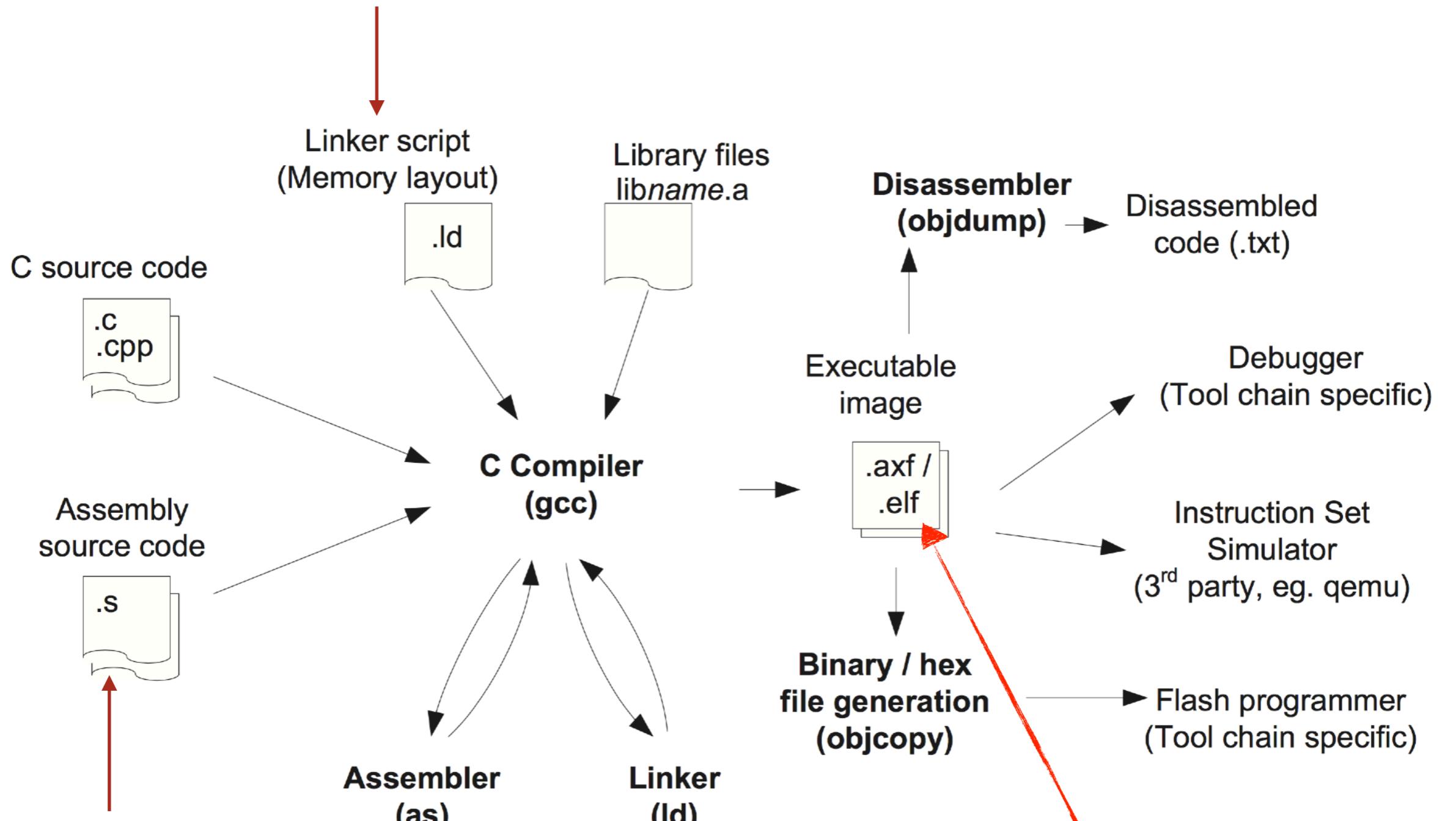
C Libraries

C Library가 설치된 디렉토리이다.



전형적인 프로그램 빌드 과정

Linker script 파일은 memory layout을 지정한다.



startup code는 일반적으로 어셈블리 언어로 작성되지만 Cortex M4에서는 C언어로 작성 할 수 있다.
런치패드에 전원이 들어가고 나서 사용할 수 있도록 정리해주는 코드 os와 비슷함

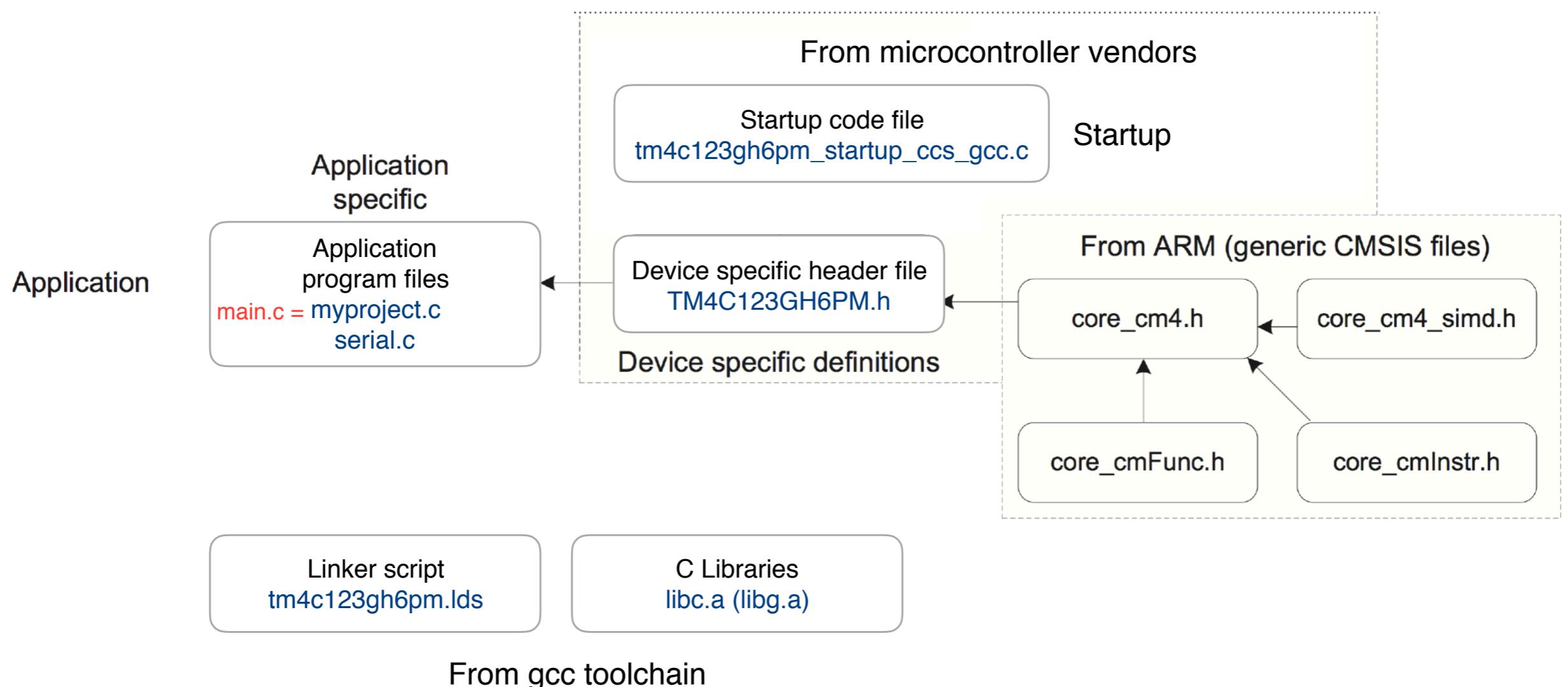
ELF (Executable and Linkable Format)

code composer에서는 .o가 .elf이다.

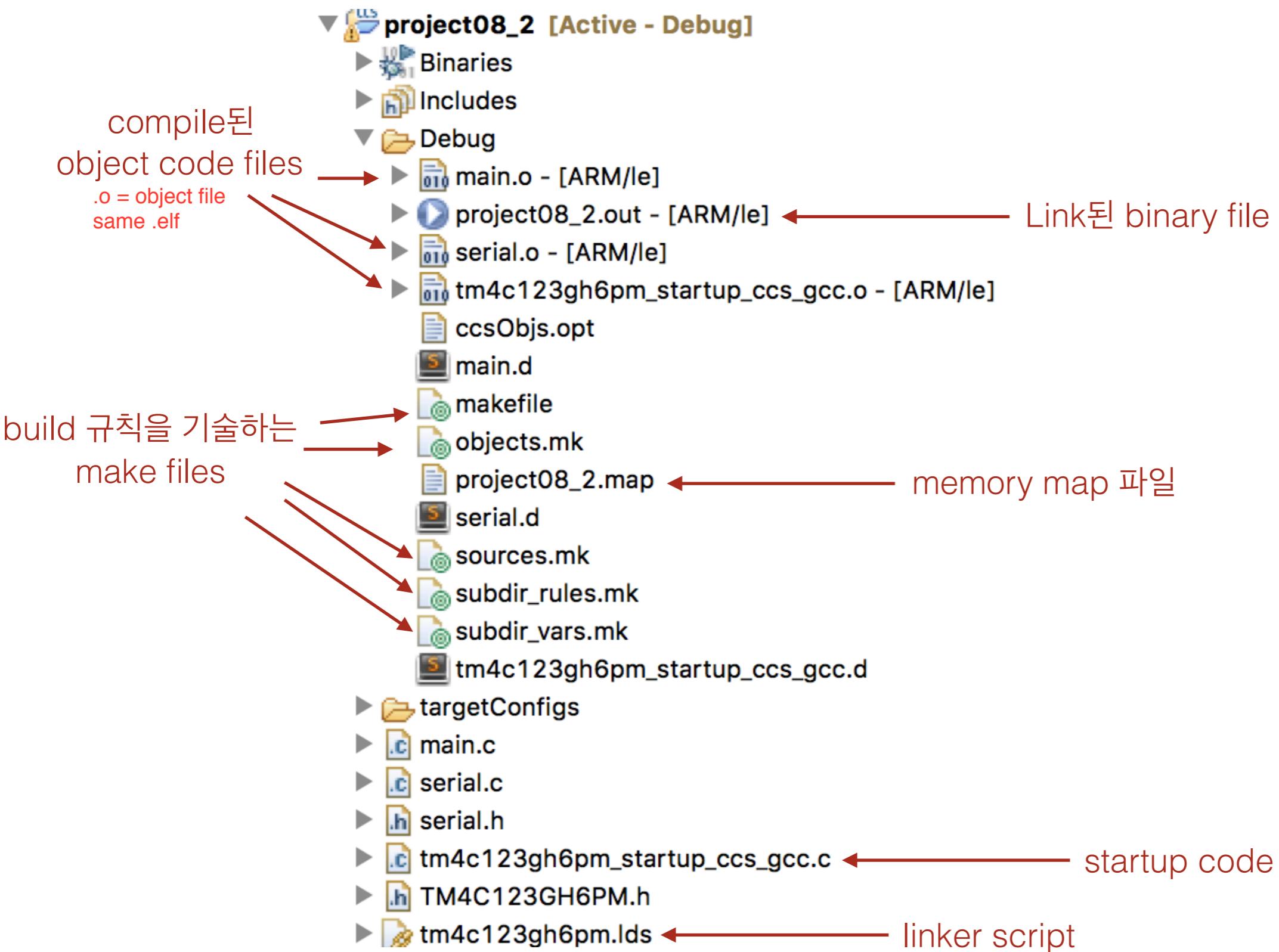
Typical Required Files for the Project

File Type	Descriptions
Application code	Source code of your application.
Device-specific CMSIS Header files	The definition header files for the microcontroller you use. This is provided by the microcontroller vendor.
Device-specific startup code for gcc	The device specific startup code for the microcontroller you use. This is provided by the microcontroller vendor.
Device-specific system initialization files	This contains the SystemInit() function (system initialization) which is specified by CMSIS-Core, and additional functions for system clock updates. This is provided by the microcontroller vendor.
Generic CMSIS Header files	This is typically included in the device driver library package or included in tool installation. Or you can download it from ARM (www.arm.com/cmsis)
Linker script	The linker script is device specific. The complete linker script for a project can be composed of several files, with one file to specify the memory layout of the device and other files to define the settings required for gcc itself. The installation of GNU Tools for ARM Embedded Processors already provided an example linker script to make it easier.
Library files	This included the runtime libraries provided by the toolchain (typically included in the installation). You can also add additional custom libraries if needed.

Example project



Files in the example project



실습: binutils

- ☞ 지난 시간의 **serial project**를 연다.
- ☞ objdump로 main.o나 serial.o를 **disassemble**해서 텍스트 파일로 저장한다.
- ☞ elf 포맷임을 확인하고 최종 **binary**와 비교해본다.

Linker script and Startup Code

Memory Layout

• .text section

- 실행 가능한 명령어 코드와 Read-Only 데이터를 저장

hard code data
ex : x=100;

• .data section

- 초기화되어 있고 수정가능한 전역(global) 혹은 static 변수들
- 이 변수들의 초기값은 처음에는 read-only memory (.text section)에 저장되어 있으며 start-up routine에 의해서 .data 섹션에 있는 각 변수들에게 복사됨

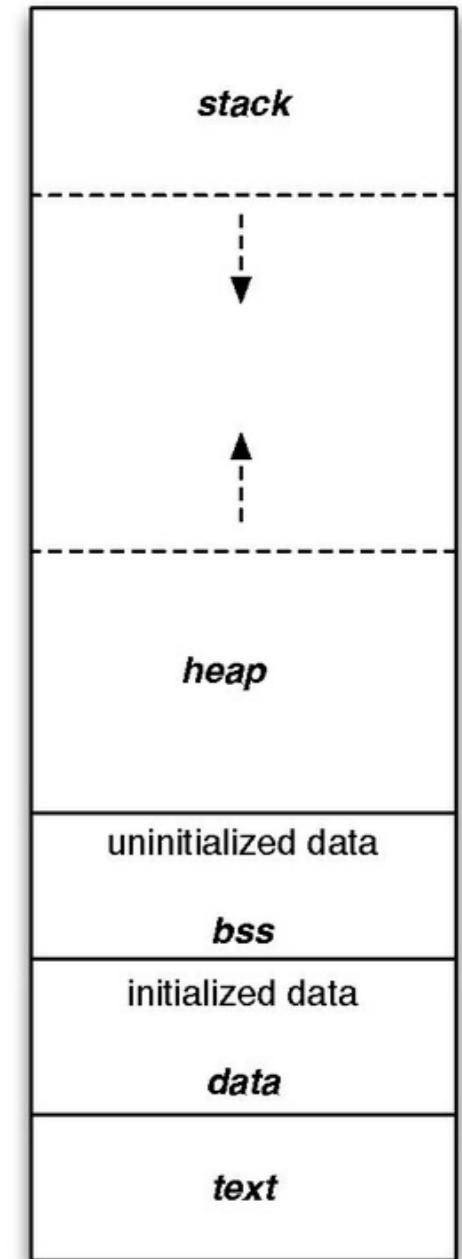
• .bss section

- 0으로 초기화되어 있거나 혹은 초기화되지 않은 수정가능한 전역(global) 혹은 static 변수들

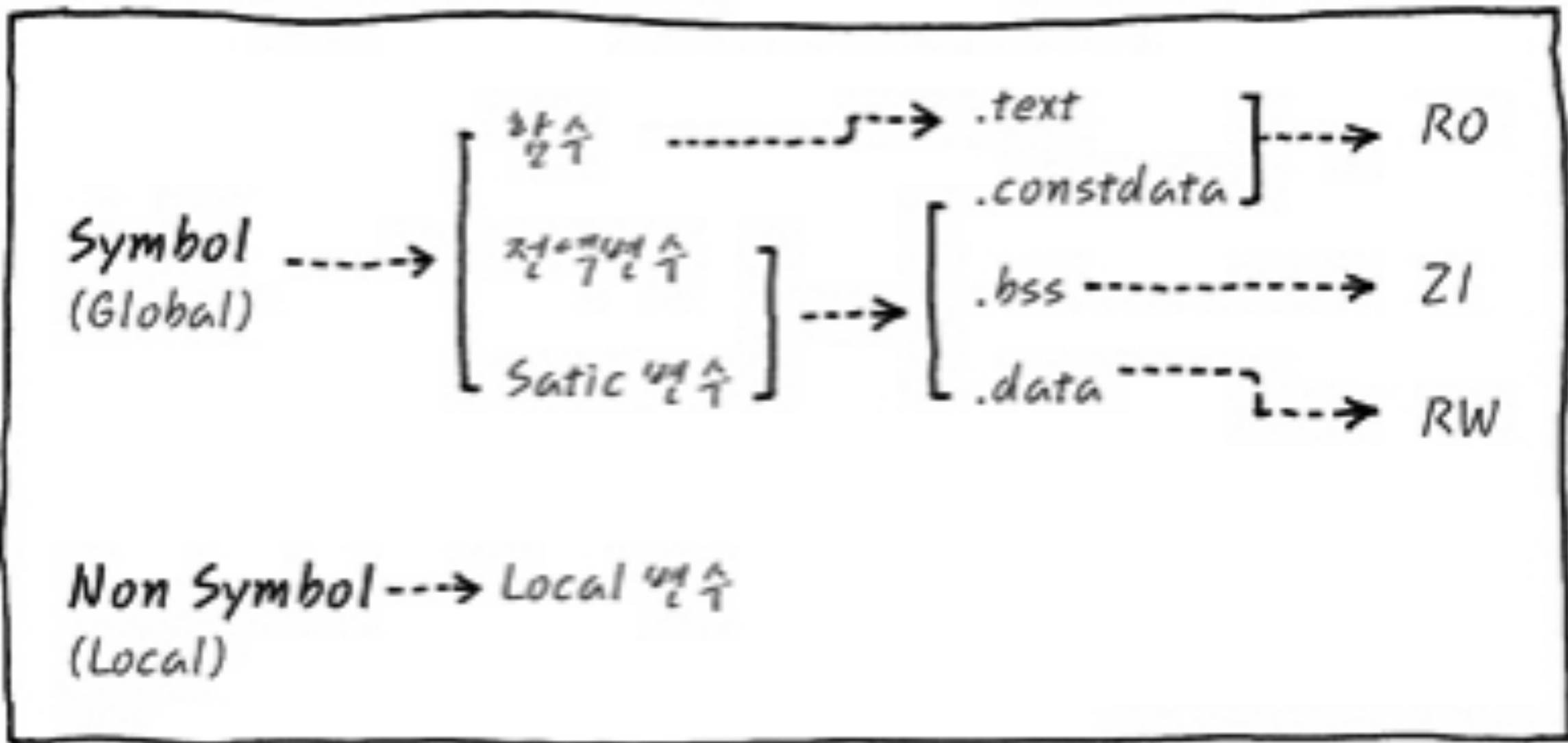
일반적으로 0으로 초기화 시킨다.. loader 입장에서

- Start-up routine에 의해서 0으로 초기화됨

• heap and stack



Memory Layout



인용: <http://recipes.egloos.com/5009181>

Memory Layout: 예

```
#include <stdio.h>
```

```
int a = 10;  
int b = 0;  
int array[10];  
int const img[5] = {0,1,2,3,4};  
char *pData = "Hieonn"
```

수정 불가해서 read only

```
main()  
{  
    static int c;  
    static int d = 15;  
    char label [100];  
    char pLabel;
```

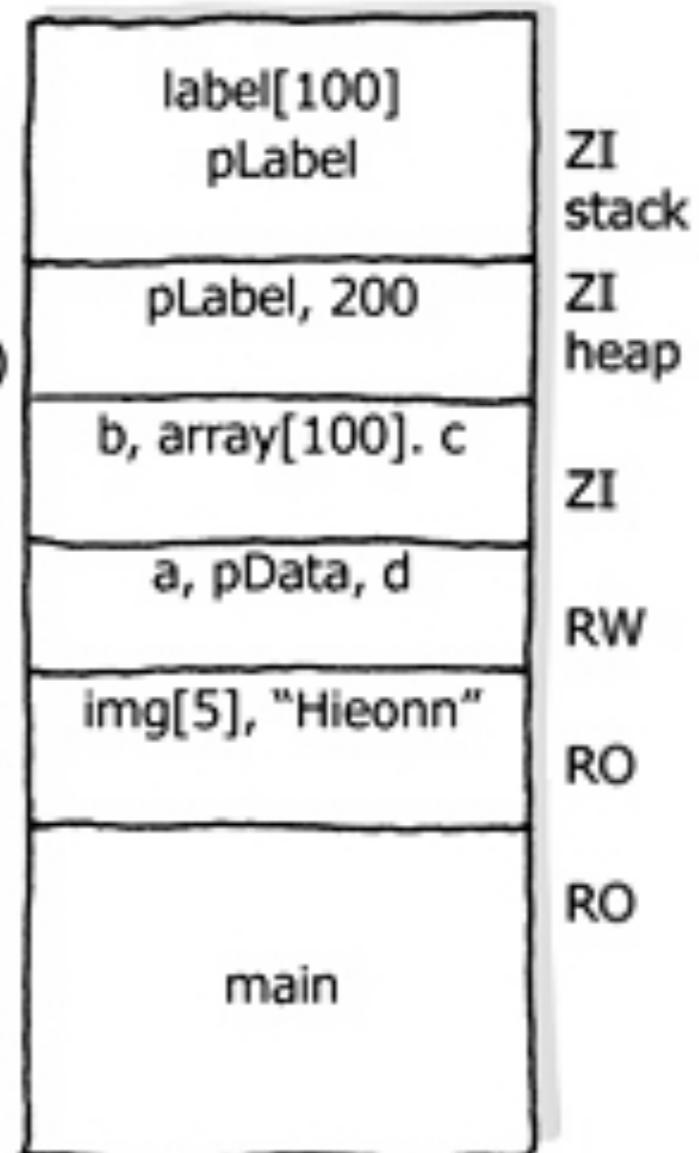
```
pLabel =  
    (char *)malloc(sizeof(char)*200);  
.....  
free (pLabel);  
}
```

RW(.data)
ZI(.bss)
ZI(.bss)
RO(.constdata)
pData → RW(.data),
"Hieonn" → RO(.constdata)

RO (.text)

ZI(.bss)
RW(.data)
Stack
Stack

Heap allocation
Heap Free

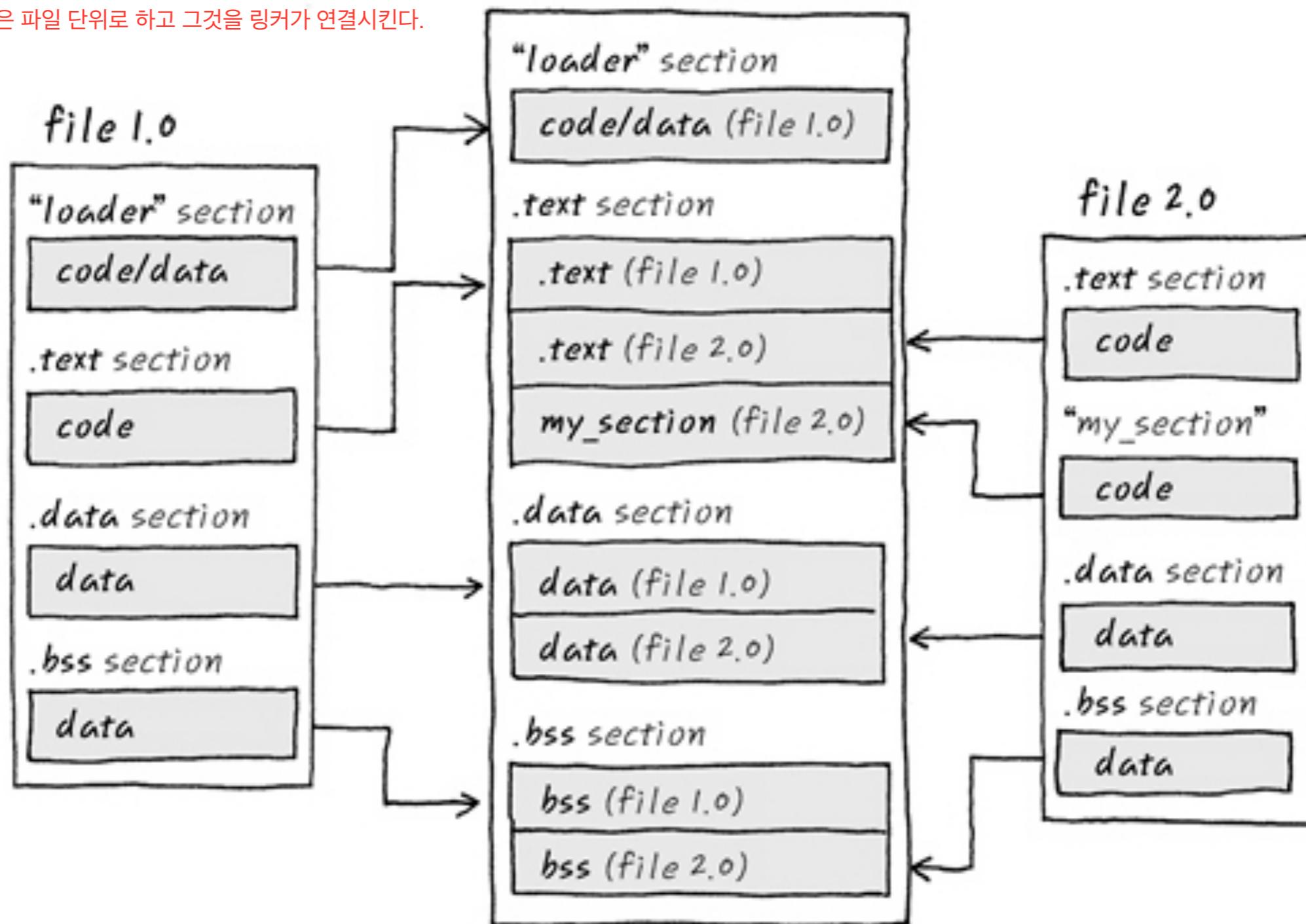


인용: <http://recipes.egloos.com/5009181>

Linking

Executable Image

컴파일은 파일 단위로 하고 그것을 링커가 연결시킨다.



링커는 같은 그룹끼리 모은다.

Linker Script: tm4c123gh6pm.lds

메모리 블록의 위치와 크기를 지정하는 명령어



```
MEMORY
{
    FLASH (RX) : ORIGIN = 0x00000000, LENGTH = 0x00040000
    SRAM (WX)  : ORIGIN = 0x20000000, LENGTH = 0x00008000
}
                                ~ 0x20008000

REGION_ALIAS("REGION_TEXT", FLASH);
REGION_ALIAS("REGION_BSS", SRAM);
REGION_ALIAS("REGION_DATA", SRAM);
REGION_ALIAS("REGION_STACK", SRAM);
REGION_ALIAS("REGION_HEAP", SRAM);
REGION_ALIAS("REGION_ARM_EXIDX", FLASH);
REGION_ALIAS("REGION_ARM_EXTAB", FLASH);
```

Linker Script: tm4c123gh6pm.lds

인터럽트 핸들러들이 있다. 무한하지 않음 I/O가 제한적이라서

(인터럽트 서비스 루틴)

인터럽트 핸들러와 그 주소를 저장해놓은 테이블이 인터럽트 벡터 테이블이다.

SECTIONS {

인터럽트 벡터

PROVIDE (_intvecs_base_address = 0x0);

메모리의 첫부분에 들어갈것

```
.intvecs (_intvecs_base_address) : AT (_intvecs_base_address) {  
    KEEP (*(.intvecs)) Keyword  
} > REGION_TEXT
```

PROVIDE (_vtable_base_address = 0x20000000);

startup code is c language

```
.vtable (_vtable_base_address) : AT (_vtable_base_address) {  
    KEEP (*(.vtable))  
} > REGION_DATA
```

.text : { .text로 시작하는 모든 섹션을 .text영역에 배치해라

```
CREATE_OBJECT_SYMBOLS  
*(.text)  
*(.text.* )  
. = ALIGN(0x4);  
KEEP (*(.ctors))  
. = ALIGN(0x4);  
KEEP (*(.dtors))  
. = ALIGN(0x4);  
_init_array_start = .;  
KEEP (*(.init_array*))  
_init_array_end = .;  
*(.init)  
*(.fini*)  
} > REGION_TEXT
```

```
68 //*****  
69 __attribute__ ((section(".intvecs")))  
70 void (* const g_pfnVectors[])(void) = 지금부터 나오는 것을 .intvecs 섹터에 배치해라  
71 {  
    (void (*) (void))((uint32_t)pui32Stack + sizeof(pui32Stack)),  
    // The initial stack pointer  
    ResetISR,  
    // The reset handler  
    NmiISR,  
    // The NMI handler  
    FaultISR,  
    // The hard fault handler  
    IntDefaultHandler,  
    // The MPU fault handler  
    IntDefaultHandler,  
    // The bus fault handler  
    IntDefaultHandler,  
    // The usage fault handler  
    0,  
    // Reserved  
    0,  
    // Reserved  
    0  
};
```

tm4c123gh6pm_startup_ccs_gcc.c

PROVIDE (__etext = .);
PROVIDE (_etext = .);
PROVIDE (etext = .);

Linker Script: tm4c123gh6pm.lds

```
.rodata : {  
    *(.rodata)  
    *(.rodata*)  
} > REGION_TEXT
```

```
.data : ALIGN(4) { 4의 배수가 되도록 유지해라  
    __data_load__ = LOADADDR (.data);  
    __data_start__ = .; data 섹션의 시작주소  
    *(.data)  
    *(.data*)  
    . = ALIGN(4); 4의 배수가 되도록 유지해라  
    __data_end__ = .;  
} > REGION_DATA AT> REGION_TEXT
```

```
.ARM.exidx : {  
    __exidx_start = .;  
    *(.ARM.exidx*.gnu.linkonce.armexidx.*)  
    __exidx_end = .;  
} > REGION_ARM_EXIDX
```

```
.ARM.extab : {  
    *(.ARM.extab*.gnu.linkonce.armextab.*)  
} > REGION_ARM_EXTAB
```

Linker Script: tm4c123gh6pm.lds

```
.bss : {  
    __bss_start__ = .;  
    *(.shbss)  
    *(.bss)  
    *(.bss.*)  
    *(COMMON)  
    . = ALIGN(4);  
    __bss_end__ = .;  
} > REGION_BSS
```

```
.heap : {  
    __heap_start__ = .;  
    end = __heap_start__;  
    _end = end;  
    __end = end;  
    KEEP(*(.heap))  
    __heap_end__ = .;  
    __HeapLimit = __heap_end__;  
} > REGION_HEAP
```

```
.stack : ALIGN(0x8) {  
    _stack = .;  
    __stack = .;  
    KEEP(*(.stack))  
} > REGION_STACK
```

}

end, _end, __end는
힙(heap)의 시작위치를 나타냄



Map File

건너뛰

❷ 프로젝트의 Debug 디렉토리에 생성된 .map 파일을 살펴본다.

194				0x00000F54	4770469E
195	.rodata	0x000000000000f58	0x24	0x00000F58	6F656948
196	*(.rodata)			0x00000F5C	00006E6E
197	.rodata	0x000000000000f58	0x7	./main.o	img
198	*(.rodata*)			0x00000F60	00000000
199	*fill*	0x000000000000f5f	0x1	0x00000F60	00000001
200	.rodata.img	0x000000000000f60	0x14	./main.o	00000002
201		0x000000000000f60		img	00000003
202	.rodata.str1.4			0x00000F6C	00000004
203		0x000000000000f74	0x4	/Application	00000043
204			0x2	(size before	_global_im
205	.rodata._global_impure_ptr	0x000000000000f78	0x4	/Application	20000010
206				_global_	_EH_FRAME_
207		0x000000000000f78			00000000
208					

213	.data	0x0000000020000000	0x84c	load address 0x0000000000000000
214		0x000000000000f80		__data_load__ = LOAD
215		0x0000000020000000		__data_start__ = .
216	*(.data)		0x0	/Applications/ti/ccsv6/tc
217	.data	0x0000000020000000	0x4	/Applications/ti/ccsv6/tc
218	.data	0x0000000020000000		_dso_handle
219		0x0000000020000000	0x0	/Applications/ti/ccsv6/tc
220	.data	0x0000000020000004	0x4	/Applications/ti/ccsv6/tc
221	.data	0x0000000020000004		./main.o
222	.data	0x0000000020000004	0x0	./tm4c123gh6pm_startup_c
223	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
224	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
225	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
226	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
227	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
228	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
229	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
230	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
231	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
232	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
233	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
234	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
235	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
236	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
237	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
238	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
239	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
240	.data	0x0000000020000004	0x0	/Applications/ti/ccsv6/tc
241	*(.data*)		0x4	./main.o
242	.data.a	0x0000000020000004		a
243		0x0000000020000004	0x4	./main.o
244	.data.pData	0x0000000020000008		pData
245		0x0000000020000008	0x4	./main.o
246	.data.d.4348	0x000000002000000c		d
247	.data.impure_data		0x4	./main.o
248		0x0000000020000010	0x428	/Applications/ti/ccsv6/tc
249	data impure_ptr			impure_data

Interrupt Vector Table (.intvecs)

- ☞ 인터럽트 핸들러의 주소를 저장하는 테이블
- ☞ 메모리의 최하위 1024 바이트 (**0~1023번지**)에 위치
- ☞ **256개**의 인터럽트에 대한 핸들러의 주소를 저장

각각 4바이트이기 때문에

Interrupt Vector Table (.intvecs)

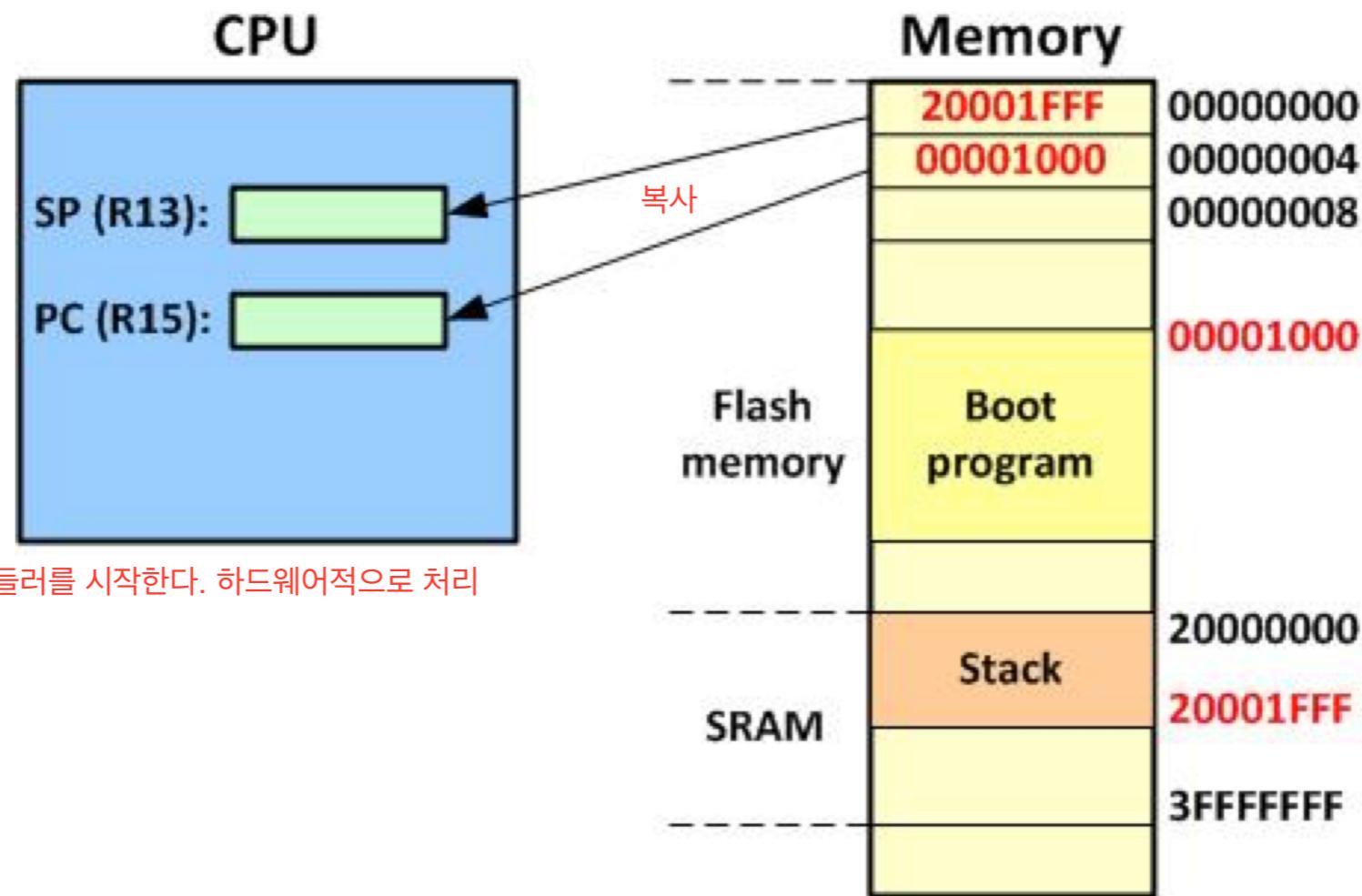
Interrupt #	Interrupt	Memory Location (Hex)
	<i>Stack Pointer initial value</i> 스택포인터의 이니셜 값	0x00000000
1	Reset	0x00000004
2	NMI non mask interrupt	0x00000008
3	Hard Fault	0x0000000C
4	Memory Management Fault	0x00000010
5	Bus Fault	0x00000014
6	Usage Fault (undefined instructions, divide by zero, unaligned memory access,...)	0x00000018
7	Reserved	0x0000001C
8	Reserved	0x00000020
9	Reserved	0x00000024
10	Reserved	0x00000028
11	SVCall	0x0000002C
12	Debug Monitor	0x00000030
13	Reserved	0x00000034
14	PendSV	0x00000038
15	SysTick	0x0000003C
16	IRQ for peripherals	0x00000040
17	IRQ for peripherals	0x00000044
...
255	IRQ for peripherals	0x000003FC

인터럽트 처리 중에 인터럽트가 들어왔을 때
..우선순위가 높은 순위로 처리

predefined
by Cortex-M

defined
by SOC chip

Cortex M4의 Booting



1. 전원이 들어오면 리셋 인터럽트 핸들러를 시작한다. 하드웨어적으로 처리

전원이 들어오면 하드웨어적으로 메모리 주소 0x00000000의 값이 SP로 복사되고
0x00000004의 값이 PC로 복사된다.

Startup Code: tm4c123gh6pm_startup_ccs_gcc.c

```
#include <stdint.h>

void ResetISR(void);
static void NmiISR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);

#ifndef HWREG
#define HWREG(x) (*((volatile uint32_t *)(x)))
#endif

//***** The entry point for the application.
//***** Reserve space for the system stack.
static uint32_t pui32stack[128];
```

시스템 스택을 배열로 설정한것은 일반적이지는 않다.

Startup Code: tm4c123gh6pm_startup_ccs_gcc.c

```
/*
 * The vector table. Note that the proper constructs must be placed on this to
 * ensure that it ends up at physical address 0x0000.0000 or at the start of
 * the program if located at a start address other than 0.
 */

__attribute__((section(".intvecs")))
void (* const g_pfnVectors[])(void) =
{
    함수 포인터 타입의 배열인데 리턴타입과 매개변수가 void인 함수

    (void (*)(void))((uint32_t)pui32Stack + sizeof(pui32Stack)), = 배열의 끝주소
                                            배열의 시작주소 // The initial stack pointer
    ResetISR, // 두번째 값 // The reset handler
    function pointer일 경우에는 앞에&를 적지 않아도 된다.
    NmiISR, // The NMI handler
    FaultISR, // The hard fault handler
    IntDefaultHandler, // The MPU fault handler
    IntDefaultHandler, // The bus fault handler
    IntDefaultHandler, // The usage fault handler
    0, // Reserved
    0, // Reserved
    0, // Reserved
    0, // Reserved
    IntDefaultHandler, // SVCall handler
    IntDefaultHandler, // Debug monitor handler
    0, // Reserved
    IntDefaultHandler, // The PendSV handler
    IntDefaultHandler, // The SysTick handler
    :
};

};
```

Startup Code:
tm4c123gh6pm_startup_ccs_gcc.c

```
//*****  
// The following are constructs created by the linker, indicating where the  
// the "data" and "bss" segments reside in memory. The initializers for the  
// for the "data" segment resides immediately following the "text" segment.  
//*****  
extern uint32_t __data_load__;    링커 스크립트에서 온 변수이다.  
extern uint32_t __data_start__;  
extern uint32_t __data_end__;  
extern uint32_t __bss_start__;  
extern uint32_t __bss_end__;
```

다른 파일에 있는 변수다. 변수 선언만 한다.

Startup Code:

tm4c123gh6pm_startup_ccs_gcc.c

```

//*****
// This is the code that gets called when the processor first starts execution
// following a reset event. Only the absolutely necessary set is performed,
// after which the application supplied entry() routine is called.
//*****
```

void ResetISR(void) 텍스트

```

{
    uint32_t *pui32Src, *pui32Dest;
    // Copy the data segment initializers from flash to SRAM.
    pui32Src = &__data_Load__;
    for(pui32Dest = &__data_Start__; pui32Dest < &__data_End__; )
    {
        *pui32Dest++ = *pui32Src++;
    }
    // Zero fill the bss segment. 0으로 초기화하는 영역 굳이 어셈블리로 하지 않아도 된다.
    __asm__(
        "    ldr    r0, =__bss_Start__\n"
        "    ldr    r1, =__bss_End__\n"
        "    mov    r2, #0\n"
        ".thumb_func\n"
        "zeroLoop:\n"
        "    cmp    r0, r1\n"
        "    it    lt\n"
        "    strlt    r2, [r0], #4\n"
        "    blt    zeroLoop");
}

// Enable the floating-point unit.
HWREG(0xE000ED88) = ((HWREG(0xE000ED88) & ~0x00F00000) | 0x00F00000);
// Call the application's entry point.
main(); main을 호출 c언어에서 메인문의 이름을 바꾸고 여기서 바꾸면 실행된다. 굳이 이름이 메인일 필요는 없다.
}
```

The diagram illustrates the flow of data from Flash memory to SRAM. It shows the assembly code for copying data from flash to SRAM, with annotations for variables like x and y, and sections like .text and .data.

- .text**: A red line points from the `int x = 1024;` line to the `__data_Load` label in the assembly code.
- Flash**: A red line points from the `int y = 512;` line to the `__data_End` label in the assembly code.
- 1024**: A red line points from the `1024` value to the `__data_Load` label in the assembly code.
- copy**: A red bracket labeled "copy" spans the entire assembly loop from `zeroLoop:` to `zeroLoop`.
- 0이 아닌 초기화된 전역변수는 데이터 섹션에 있다.**: A red annotation points to the `__data_Start` label in the assembly code.
- Store R2 to word addressed by R0. Increment R0 by 4**: A red annotation points to the `strlt r2, [r0], #4` instruction in the assembly code.
- main을 호출 c언어에서 메인문의 이름을 바꾸고 여기서 바꾸면 실행된다. 굳이 이름이 메인일 필요는 없다.**: A red annotation points to the `main();` call in the C code.

Startup Code:
tm4c123gh6pm_startup_ccs_gcc.c

```
static void
NmiISR(void)
{
    // Enter an infinite loop.
    // while(1)
    //

}

static void
FaultISR(void)
{
    // Enter an infinite loop.
    // while(1)
    //

    무책임하다.. 적절한 답변, 대응을 해야한다.

}

static void
IntDefaultHandler(void)
{
    // Go into an infinite loop.
    // while(1)
    //

}
```

```
#include "TM4C123GH6PM.h"
#include "serial.h"

char strBuffer[BUFFER_SIZE];

char *promptMsg = "Type something and press enter: ";
char echoMes[] = "You typed: ";

const char delim = '\r';    const가 있으면 read only된다. ram에 있을 필요가 없다.
char * const newlineMsg = "\n\r";

int main(void)
{
    init_serial();

    while(1)
    {
        printString(promptMsg);

        readString(delim);

        printString(newlineMsg);
        printString(echoMes);
        printString(strBuffer);
        printString(newlineMsg);
    }
}
```

serial.h

```
#include "TM4C123GH6PM.h"

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define BUFFER_SIZE 100

extern char strBuffer[];

void init_serial(void);
char readChar(void);
void printChar(char c);
void printInt(int n);
void printString(char *);
void readString( char delimiter );
```

```
#include "serial.h"

/* init_serial(), readChar(void), printChar(char c)은 그대로 ... */

void printString(char *str)
{
    static int lineCount = 1;
    int i = 0;
    for (i=0; i < strlen(str); i++)
        printChar( str[i] );
    lineCount = (lineCount + 1) % 10;
}

void readString( char delimiter )
{
    int size = 0;
    char c = readChar();
    printChar(c);

    while(c != delimiter && size < BUFFER_SIZE-1)
    {
        strBuffer[size++] = c; // put the new character at the end of the array
        c = readChar();
        printChar(c); // display the character the user typed
    }
    strBuffer[size] = '\0';
}
```

Map 파일: .rodata 섹션

.rodata	0x0000000000000928	0x34
*(.rodata)		
.rodata	0x0000000000000928	0x27 ./main.o
(.rodata)		
.rodata.delim	0x000000000000094f	0x1 ./main.o
	0x000000000000094f	delim
.rodata.newlineMsg	0x0000000000000950	0x4 ./main.o
	0x0000000000000950	newlineMsg

Map 파일: .data 섹션

(.data)		
.data.promptMsg	0x0000000020000004	0x4 ./main.o promptMsg
	0x0000000020000004	
.data.echoMes	0x0000000020000008	0xc ./main.o echoMes
	0x0000000020000008	

Startup Coded 실행과정

- ④ **startup code**의 실행과정을 살펴본다.

Linker Script와 startup code의 설정

<http://state-machine.com/quickstart/>

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-bigarm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(Reset_Handler) /* entry Point */

MEMORY { /* memory map of Tiva TM4C123GH6PM */
    ROM (rx) : ORIGIN = 0x00000000, LENGTH = 256K
    RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 32K
}

/* The size of the stack used by the application. NOTE: you need to adjust */
STACK_SIZE = 256;

/* The size of the heap used by the application. NOTE: you need to adjust */
HEAP_SIZE = 100;

SECTIONS {

    .isr_vector : { /* the vector table goes FIRST into ROM */
        KEEP(*(.isr_vector)) /* vector table */
        . = ALIGN(4);
    } >ROM

    .text : { /* code and constants */
        . = ALIGN(4);
        *(.text)
        *(.text*)
        *(.rodata)
        *(.rodata*)
        KEEP (*(.init))
        KEEP (*(.fini))

        . = ALIGN(4);
    } >ROM
}
```

[Download zip file.](#)

Linker Script와 startup code의 설정

<http://state-machine.com/quickstart/>

```
.preinit_array : {  
    PROVIDE_HIDDEN (__preinit_array_start = .);  
    KEEP (*(.preinit_array*))  
    PROVIDE_HIDDEN (__preinit_array_end = .);  
} >ROM  
  
.init_array : {  
    PROVIDE_HIDDEN (__init_array_start = .);  
    KEEP (*(SORT(.init_array.*)))  
    KEEP (*(.init_array*))  
    PROVIDE_HIDDEN (__init_array_end = .);  
} >ROM  
  
.fini_array : {  
    PROVIDE_HIDDEN (__fini_array_start = .);  
    KEEP (*(.fini_array*))  
    KEEP (*(SORT(.fini_array.*)))  
    PROVIDE_HIDDEN (__fini_array_end = .);  
} >ROM  
  
_etext = .; /* global symbols at end of code */
```

Linker Script와 startup code의 수정

<http://state-machine.com/quickstart/>

```
.stack : {  
    __stack_start__ = .;  
    . = . + STACK_SIZE;  
    . = ALIGN(4);  
    __stack_end__ = .;  
} >RAM  
  
.data : AT (_etext) {  
    __data_load = LOADADDR (.data);  
    __data_start = .;  
    *(.data)          /* .data sections */  
    *(.data*)         /* .data* sections */  
    . = ALIGN(4);  
    __data_end__ = .;  
    _edata = __data_end__;  
} >RAM  
  
.bss : {  
    __bss_start__ = .;  
    *(.bss)  
    *(.bss*)  
    *(COMMON)  
    . = ALIGN(4);  
    _ebss = .;        /* define a global symbol at bss end */  
    __bss_end__ = .;  
} >RAM
```

Linker Script와 startup code의 수정

<http://state-machine.com/quickstart/>

```
PROVIDE ( end = _ebss );
PROVIDE ( _end = _ebss );
PROVIDE ( __end__ = _ebss );

.heap : {
    __heap_start__ = .;
    . = . + HEAP_SIZE;
    . = ALIGN(4);
    __heap_end__ = .;
} >RAM

/* Remove information from the standard libraries */
/DISCARD/ : {
    libc.a ( * )
    libm.a ( * )
    libgcc.a ( * )
}
}
```

Linker Script와 startup code의 설정

<http://state-machine.com/quickstart/>

```
/* start and end of stack defined in the linker script -----*/
extern int __stack_start__;
extern int __stack_end__;

/* weak prototypes for error handlers -----*/
/***
* \note
* The function assert_failed defined at the end of this file defines
* the error/assertion handling policy for the application and might
* need to be customized for each project. This function is defined in
* assembly to avoid accessing the stack, which might be corrupted by
* the time assert_failed is called.
*/
__attribute__((naked)) void assert_failed(char const *module, int loc);

/* Function prototypes -----*/
void Default_Handler(void); /* Default empty handler */
void Reset_Handler(void); /* Reset Handler */
void SystemInit(void); /* CMSIS system initialization */

/*-----
* weak aliases for each Exception handler to the Default_Handler.
* Any function with the same name will override these definitions.
*/
/* Cortex-M Processor fault exceptions... */
void NMI_Handler      ((void)) __attribute__((weak));
void HardFault_Handler((void)) __attribute__((weak));
void MemManage_Handler((void)) __attribute__((weak));
void BusFault_Handler((void)) __attribute__((weak));
void UsageFault_Handler((void)) __attribute__((weak));

/* Cortex-M Processor non-fault exceptions... */
void SVC_Handler      ((void)) __attribute__((weak, alias("Default_Handler")));
void DebugMon_Handler((void)) __attribute__((weak, alias("Default_Handler")));
void PendSV_Handler   ((void)) __attribute__((weak, alias("Default_Handler")));
void SysTick_Handler  ((void)) __attribute__((weak, alias("Default_Handler")));
```

Linker Script와 startup code의 수정

<http://state-machine.com/quickstart/>

```
/* external interrupts... */  
void GPIOPortA_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void GPIOPortB_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void GPIOPortC_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void GPIOPortD_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void GPIOPortE_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void UART0_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void UART1_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void SSI0_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void I2C0_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void PWMFault_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void PWMGen0_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void PWMGen1_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void PWMGen2_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void QEI0_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void ADCSeq0_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void ADCSeq1_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void ADCSeq2_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void ADCSeq3_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void Watchdog_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void Timer0A_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void Timer0B_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void Timer1A_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void Timer1B_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void Timer2A_IRQHandler __attribute__((weak, alias("Default_Handler")));  
void Timer2B_IRQHandler __attribute__((weak, alias("Default_Handler")));  
...
```

Linker Script와 startup code의 수정

<http://state-machine.com/quickstart/>

Reset Handler

```
/* reset handler -----*/
void Reset_Handler(void) {
    extern int main(void);
    extern int __libc_init_array(void);
    extern unsigned __data_start; /* start of .data in the linker script */
    extern unsigned __data_end__; /* end of .data in the linker script */
    extern unsigned const __data_load; /* initialization values for .data */
    extern unsigned __bss_start__; /* start of .bss in the linker script */
    extern unsigned __bss_end__; /* end of .bss in the linker script */
    extern void software_init_hook(void) __attribute__((weak));

    unsigned const *src;
    unsigned *dst;

    //SystemInit(); /* CMSIS system initialization */

    /* copy the data segment initializers from flash to RAM... */
    src = &__data_load;
    for (dst = &__data_start; dst < &__data_end__; ++dst, ++src) {
        *dst = *src;
    }

    /* zero fill the .bss segment in RAM... */
    for (dst = &__bss_start__; dst < &__bss_end__; ++dst) {
        *dst = 0;
    }
}
```

Reset Handler

```
/* init hook provided? */
if (&software_init_hook != (void (*)(void))(0)) {
    /* give control to the RTOS */
    software_init_hook(); /* this will also call __libc_init_array */
}
else {
    /* call all static constructors in C++ (harmless in C programs) */
    __libc_init_array();
    (void)main(); /* application's entry point; should never return! */
}

/* the previous code should not return, but assert just in case... */
assert_failed("Reset_Handler", __LINE__);
}
```

Linker Script와 startup code의 수정

<http://state-machine.com/quickstart/>

```
/* fault exception handlers -----*/
__attribute__((naked)) void NMI_Handler(void);
void NMI_Handler(void) {
    __asm volatile (
        "    ldr r0,=str_nmi\n\t"
        "    mov r1,#1\n\t"
        "    b assert_failed\n\t"
        "str_nmi: .asciz \"NMI\"\n\t"
    );
}
/*-----*/
__attribute__((naked)) void MemManage_Handler(void);
void MemManage_Handler(void) {
    __asm volatile (
        "    ldr r0,=str_mem\n\t"
        "    mov r1,#1\n\t"
        "    b assert_failed\n\t"
        "str_mem: .asciz \"MemManage\"\n\t"
    );
}
/*-----*/
__attribute__((naked)) void HardFault_Handler(void);
void HardFault_Handler(void) {
    __asm volatile (
        "    ldr r0,=str_hrd\n\t"
        "    mov r1,#1\n\t"
        "    b assert_failed\n\t"
        "str_hrd: .asciz \"HardFault\"\n\t"
    );
}
...
```

assert_failed

```
/* Board Support Package */
#include "TM4C123GH6PM.h"
#include "bsp.h"

__attribute__((naked)) void assert_failed (char const *file, int line) {
    /* TBD: damage control */
    NVIC_SystemReset(); /* reset the system */
}

void SysTick_Handler(void) {
    GPIOF_AHB->DATA_Bits[LED_BLUE] ^= LED_BLUE;
}
```