

---

---

# ARM 펌웨어 개발환경 개요

Hancheol Cho

---

---

MCU 공부를 어떻게 해야 하나요?

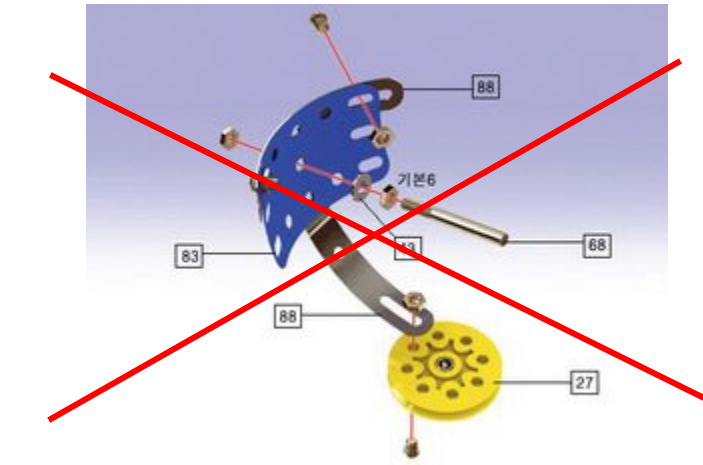


**Data Sheet**

**Reference Manual**

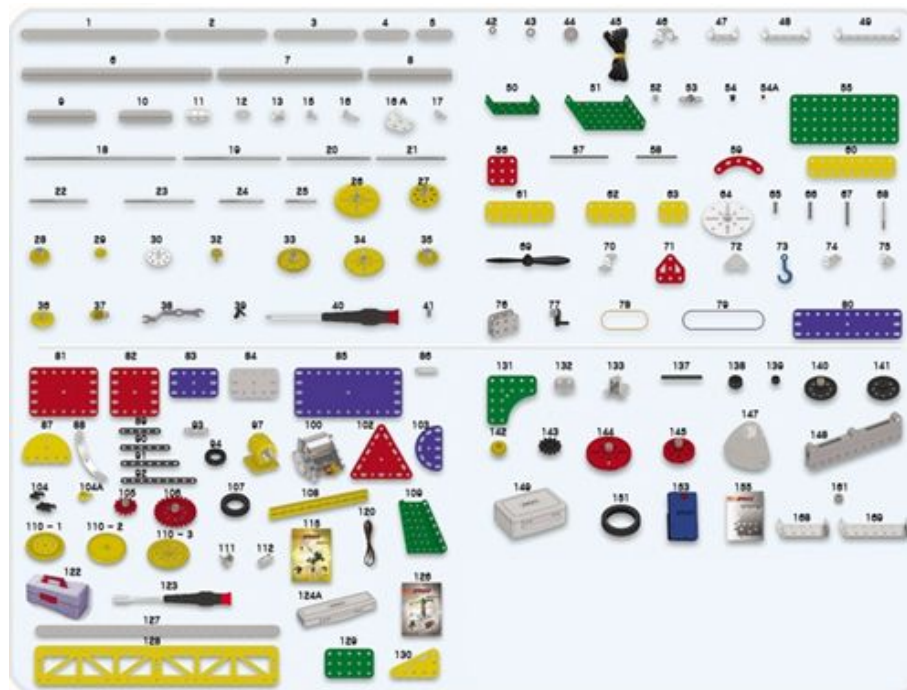
# Data Sheet가 어려운 이유

Data Sheet는 조립 메뉴얼이 아님



# Data Sheet가 어려운 이유

Data Sheet는 부품 사용법




# Data Sheet가 어려운 이유

조립 방법 및 활용은 스스로 만들어 가는 것




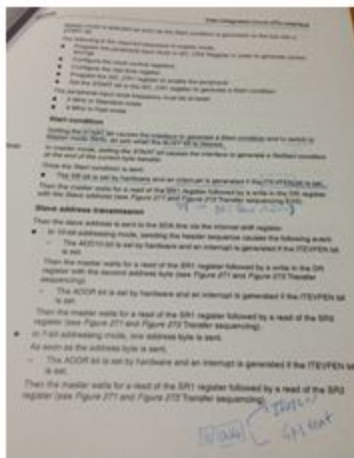
# Data Sheet Study

반복 학습하여 각 기능을 정확히 파악

 STM32F10xxx Cortex™-M3 programming manual.pdf

 STM32F103CB Reference Manual.pdf

 STM32F103CB 데이터 쉬트.pdf



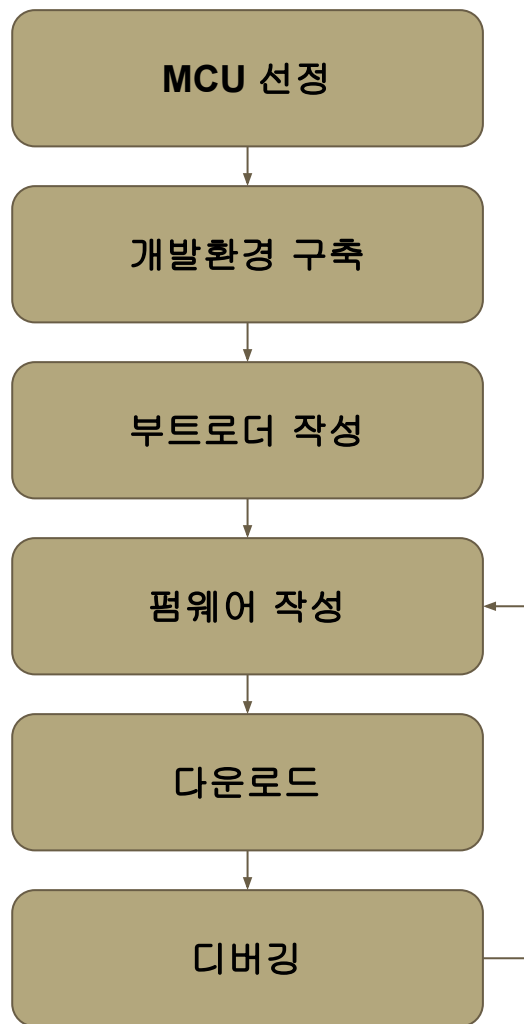
최근에는 MCU의 기능이 많아지고 펌웨어 라이브러리를 MCU 제조사에서 제공하기 때문에 예전 보다는 데이터 쉬트나 레퍼런스 메뉴얼의 중요도가 낮아지는 것은 사실

# 어떤 MCU로 공부 해야 하나요?





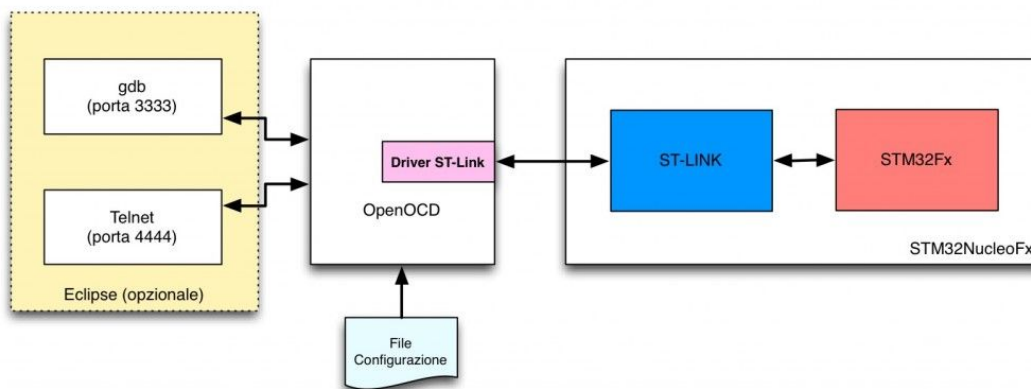
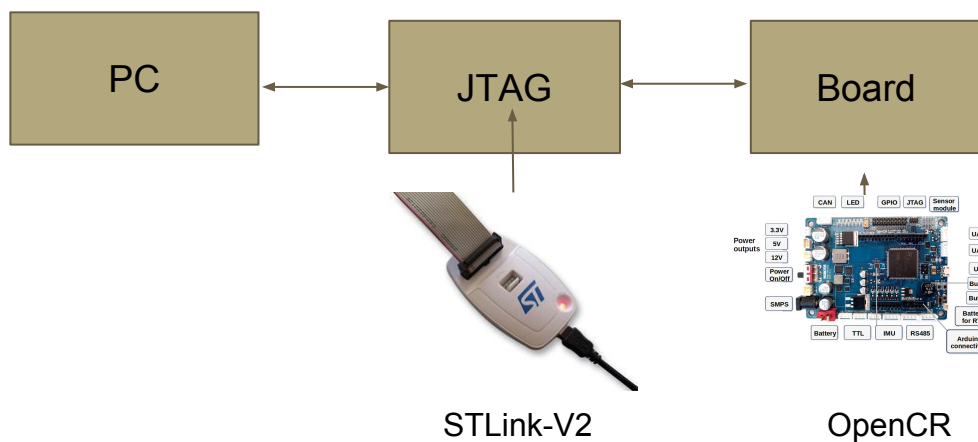
# 개발 과정



# 개발 환경 - Hardware

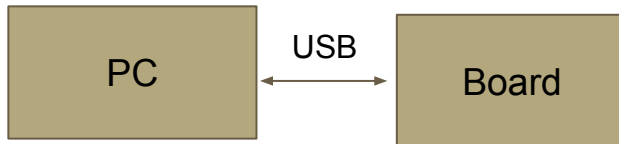
- JTAG

- JTAG을 이용한 실시간 디버깅 가능

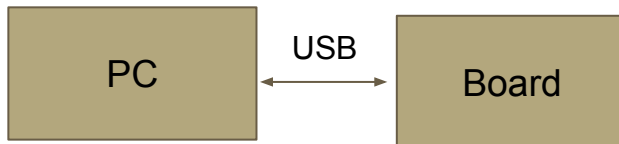


# 개발 환경 - Hardware

- DFU (Device Firmware Upgrade) 모드
  - MCU에 자체 내장된 DFU 부트로더를 이용하여 펌웨어 다운로드



- 부트로더
  - 직접 구현한 부트로더를 이용하여 펌웨어 다운로드



# 컴파일러 선정

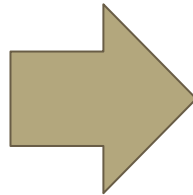
- 임베디드 프로그래밍을 위한 컴파일러 선택은 중요함

유료 컴파일러 vs 무료 컴파일러

# 컴파일러 선정

- 유료 컴파일러
  - 통합 개발 환경 제공으로 쉬운 사용
  - 문제 발생시 고객 지원 가능

비용 확보 가능



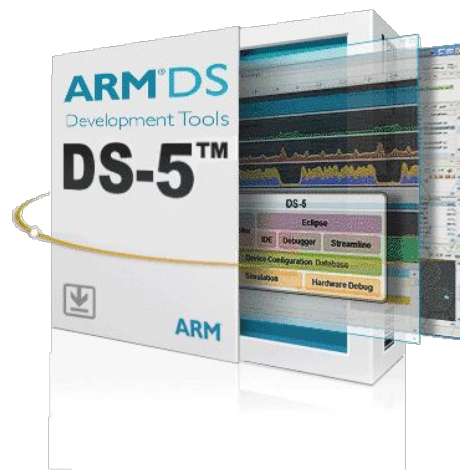
유료  
컴파일러

# 컴파일러 - 무료

- <http://www.openstm32.org/HomePage>
- <http://www.coocox.org/software/coide.php>
- <http://www.emide.org/>
- Eclipse + GCC
  - <https://eclipse.org/>
  - <https://launchpad.net/gcc-arm-embedded>



# 컴파일러 - 유료



# 컴파일러 - TrueSTUDIO

- ST사의 MCU를 사용한다면 무료로 사용 가능한 TrueSTUDIO도 좋은 대안임



---

TrueSTUDIO<sup>®</sup> for STM32



# 버전 관리 시스템

버전 관리 시스템을  
사용하십니까?

# 버전 관리 시스템

버전 관리 시스템은 소스 백업용이  
아니다.

# 버전 관리 시스템

협업과 소스 유지/관리를 위한  
선택이 아닌 필수

# 버전 관리 시스템



# 버전 관리 시스템

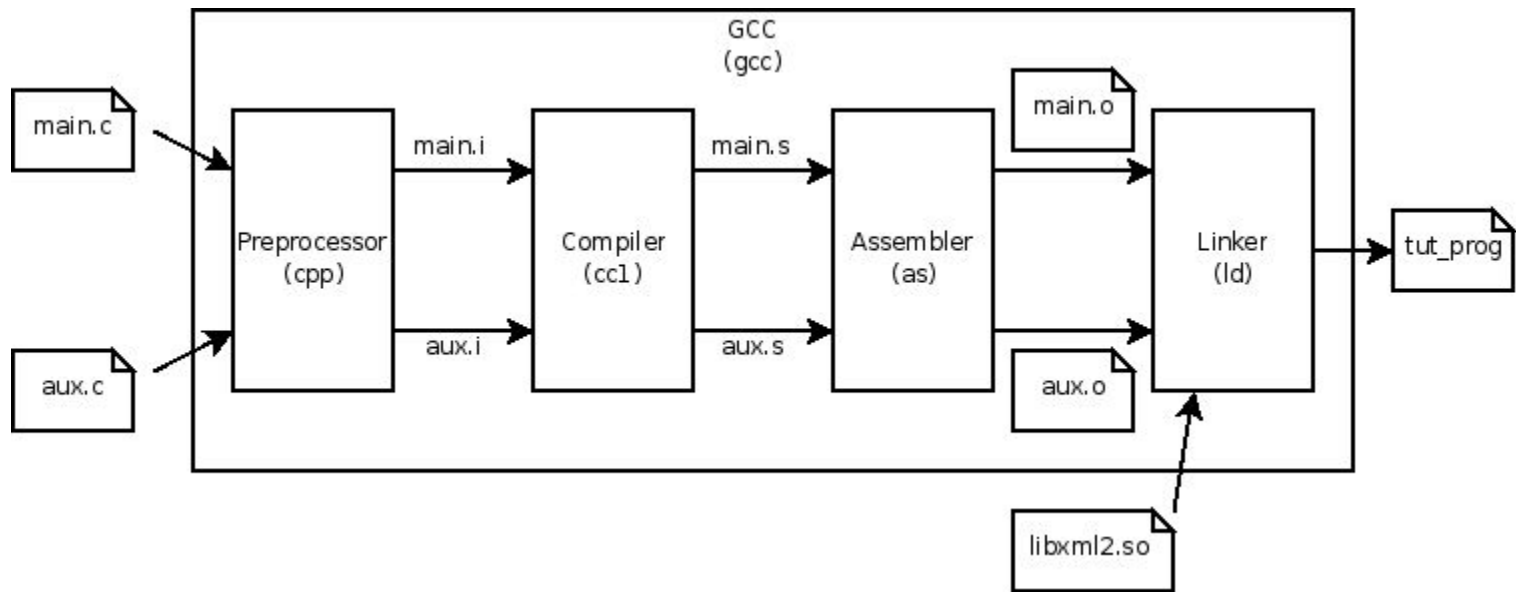


# 이슈 트래커



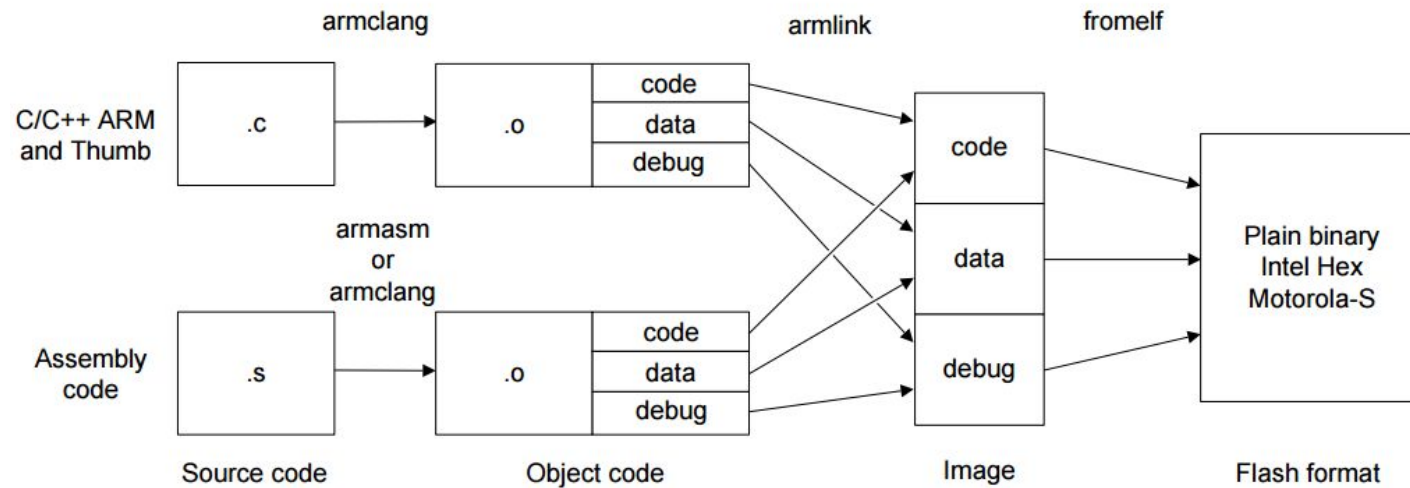
# 빌드 과정

- 소스 빌드 과정 - GCC 컴파일러



# 빌드 과정

- 소스 빌드 과정 - ARM 컴파일러

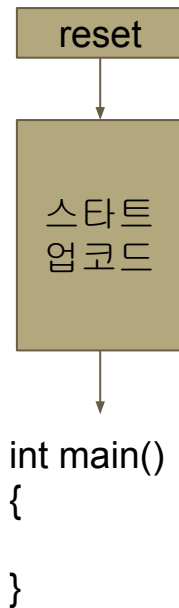




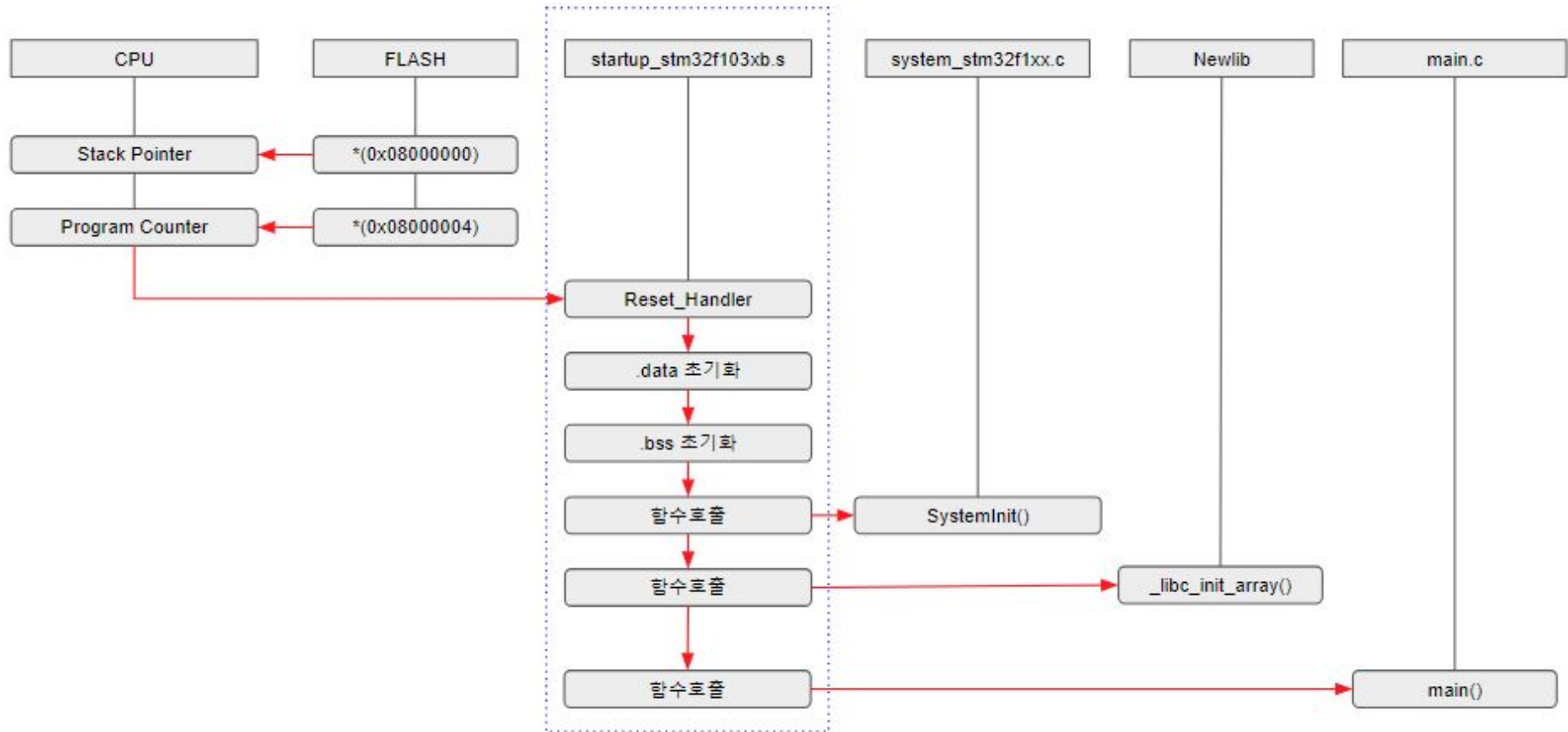
스타트업 코드 ?

# 스타트업 코드

- Reset 이후에 main함수가 실행되기 전까지의 코드를 일반적으로 스타트업 코드라고 함
  - 어셈블리어로 대부분 작성되어 있으며 Cortex-M 시리즈는 C코드만으로도 작성 가능



# 스타트업 코드



# 스타트업 코드

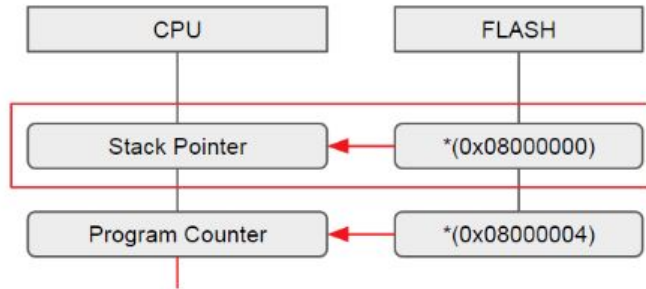


Figure 12. Vector table

Exception number	IRQ number	Offset	Vector
83	67	0x014C	IRQ67
.	.	.	.
.	.	.	.
.	.	.	.
18	2	0x004C	IRQ2
17	1	0x0048	IRQ1
16	0	0x0044	IRQ0
15	-1	0x0040	Systick
14	-2	0x003C	PendSV
13		0x0038	Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			Reserved
8			Reserved
7			Reserved
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

# 스타트업 코드

```
.section .text.Reset_Handler
.weak Reset_Handler
.type Reset_Handler, %function
Reset_Handler:
    ldr    sp, =_estack    /* set stack pointer */
```

스택포인터 초기화

```
    movs   r1, #0
    b      LoopCopyDataInit
```

```
CopyDataInit:
    ldr    r3, =_sidata
    ldr    r3, [r3, r1]
    str    r3, [r0, r1]
    adds   r1, r1, #4
```

```
LoopCopyDataInit:
    ldr    r0, =_sdata
    ldr    r3, =_edata
    adds   r2, r0, r1
    cmp    r2, r3
    bcc    CopyDataInit
    ldr    r2, =_sbss
    b      LoopFillZerobss
```

.bss, .data 섹션 초기화

```
/* Zero fill the bss segment. */
```

```
FillZerobss:
    movs   r3, #0
    str    r3, [r2], #4
```

```
LoopFillZerobss:
    ldr    r3, =_ebss
    cmp    r2, r3
    bcc    FillZerobss
```

# 스타트업 코드

```
g_pfnVectors:  
    .word _estack  
    .word Reset_Handler  
  
    .word NMI_Handler  
    .word HardFault_Handler  
    .word MemManage_Handler  
    .word BusFault_Handler  
    .word UsageFault_Handler  
    .word 0  
    .word 0  
    .word 0  
    .word 0  
    .word SVC_Handler  
    .word DebugMon_Handler  
    .word 0  
    .word PendSV_Handler  
    .word SysTick_Handler
```

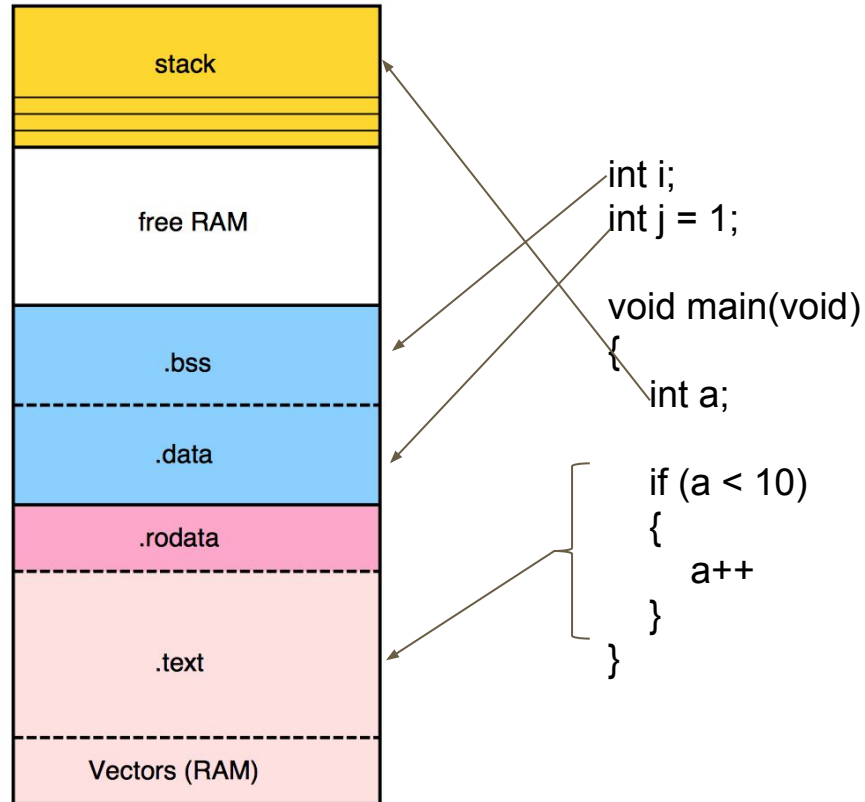
인터럽트 벡터 함수 정의

링커 스크립트 ?

# 메모리 섹션

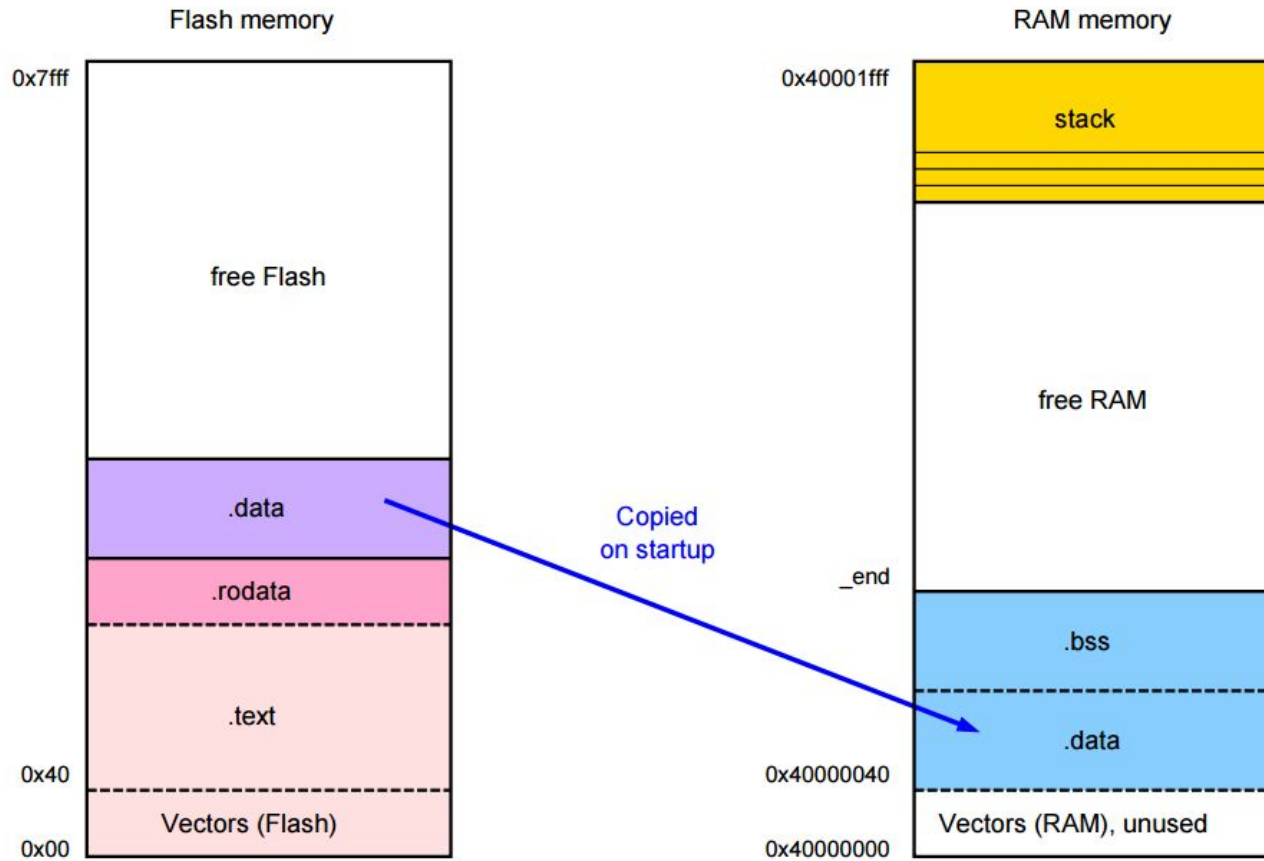
## Sections:

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





# 메모리 섹션



# 링커 스크립트

```
/* Specify the memory areas */
```

```
MEMORY
```

```
{
```

```
FLASH (rx)      : ORIGIN = 0x08000000, LENGTH = 64K
```

```
RAM_DTCM (xrw)  : ORIGIN = 0x20000000, LENGTH = 0x10000
```

```
RAM (xrw)       : ORIGIN = 0x200114EC, LENGTH = 0x3EB14
```

```
QSPI (rx)       : ORIGIN = 0x90000000, LENGTH = 16M
```

```
}
```

물리적인 메모리 영역을 정의함

```
/* Define output sections */
```

```
SECTIONS
```

```
{
```

```
/* The startup code goes first into FLASH */
```

```
.isr_vector :
```

```
{
```

```
  . = ALIGN(4);
```

```
  KEEP(*(.isr_vector)) /* Startup code */
```

```
  . = ALIGN(4);
```

```
} >FLASH
```

```
/* The program code and other data goes into FLASH */
```

```
.text :
```

```
{
```

```
  . = ALIGN(4);
```

```
*(.text)           /* .text sections (code) */
```

```
*(.text*)          /* .text* sections (code) */
```

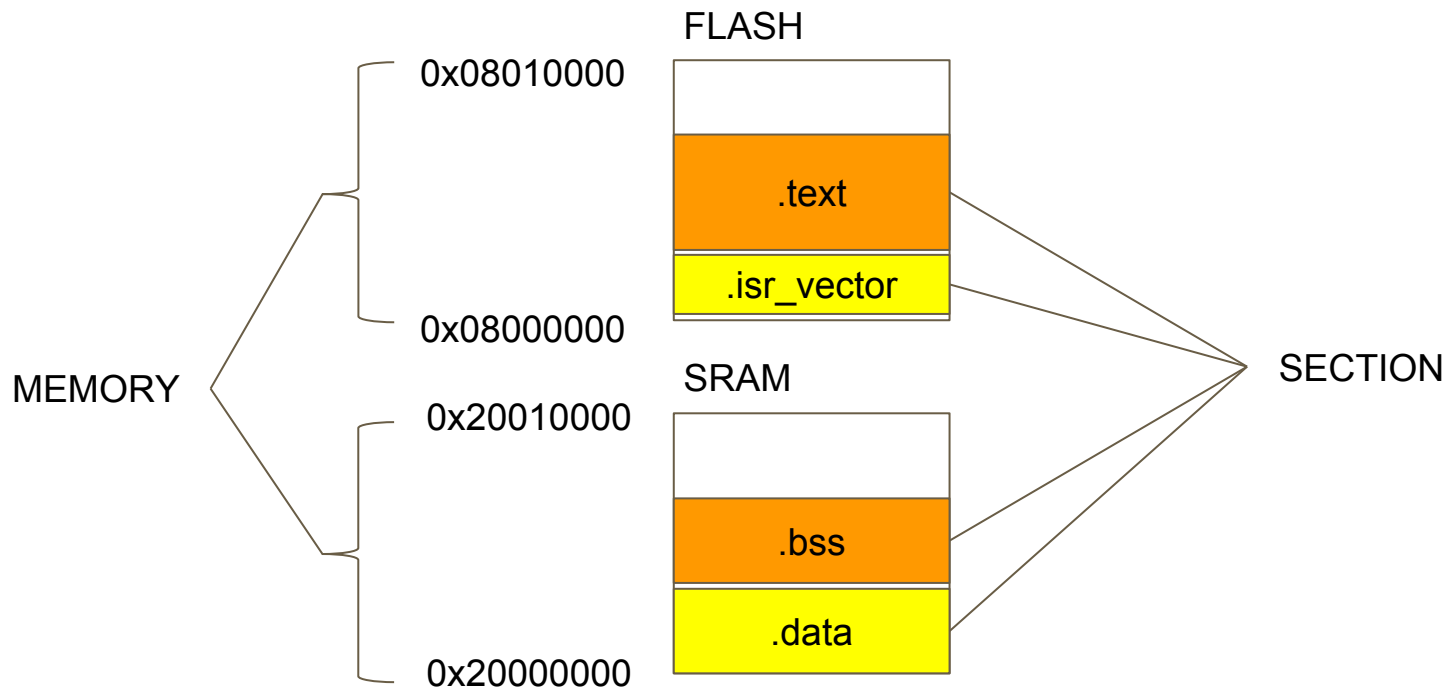
```
*(.glue_7)          /* glue arm to thumb code */
```

```
*(.glue_7t)         /* glue thumb to arm code */
```

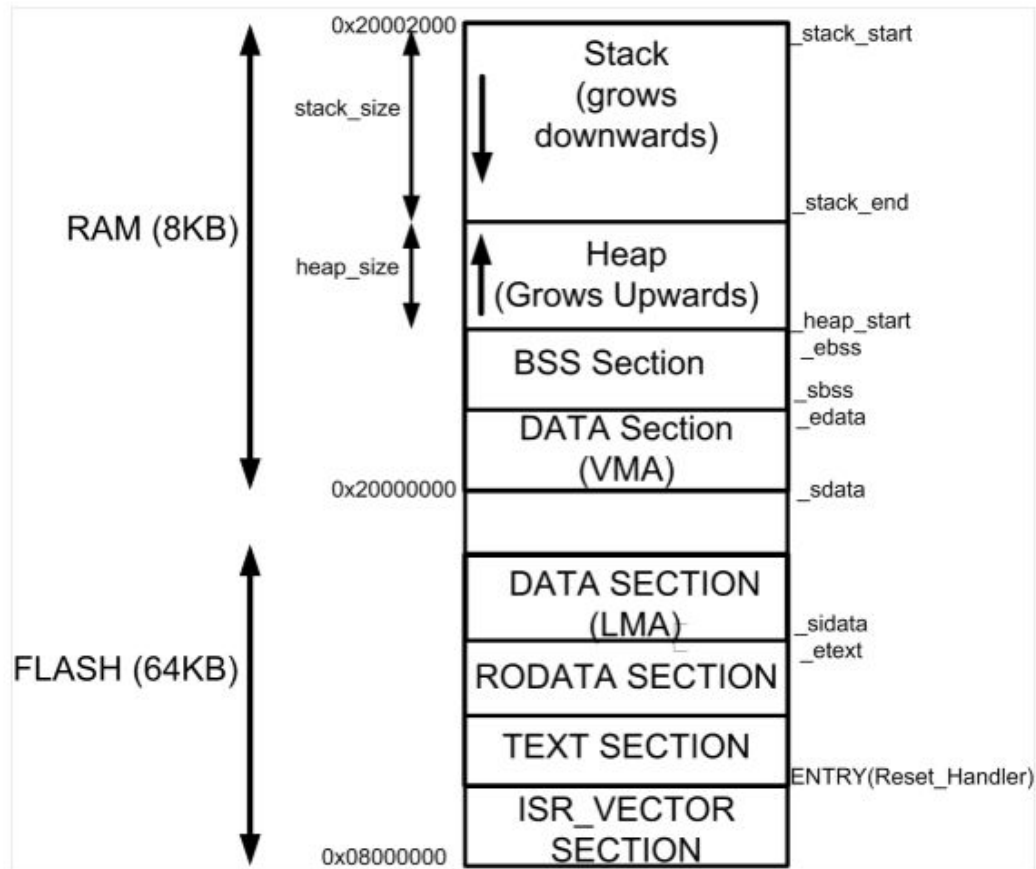
```
*(.eh_frame)
```

물리적인 메모리에 위치시킬 섹션 정의

# 링커 스크립트



# 링커 스크립트



출처 : <http://hertaville.com/a-sample-linker-script.html>

# Time to break

- Zynq 이야기
  - [Zynq FPGA 사용기 #1 - 개요 및 개발환경](#)
  - [Zynq FPGA 사용기 #2 - Hello World 출력](#)
  - [Zynq FPGA 사용기 #3 - LED 제어](#)