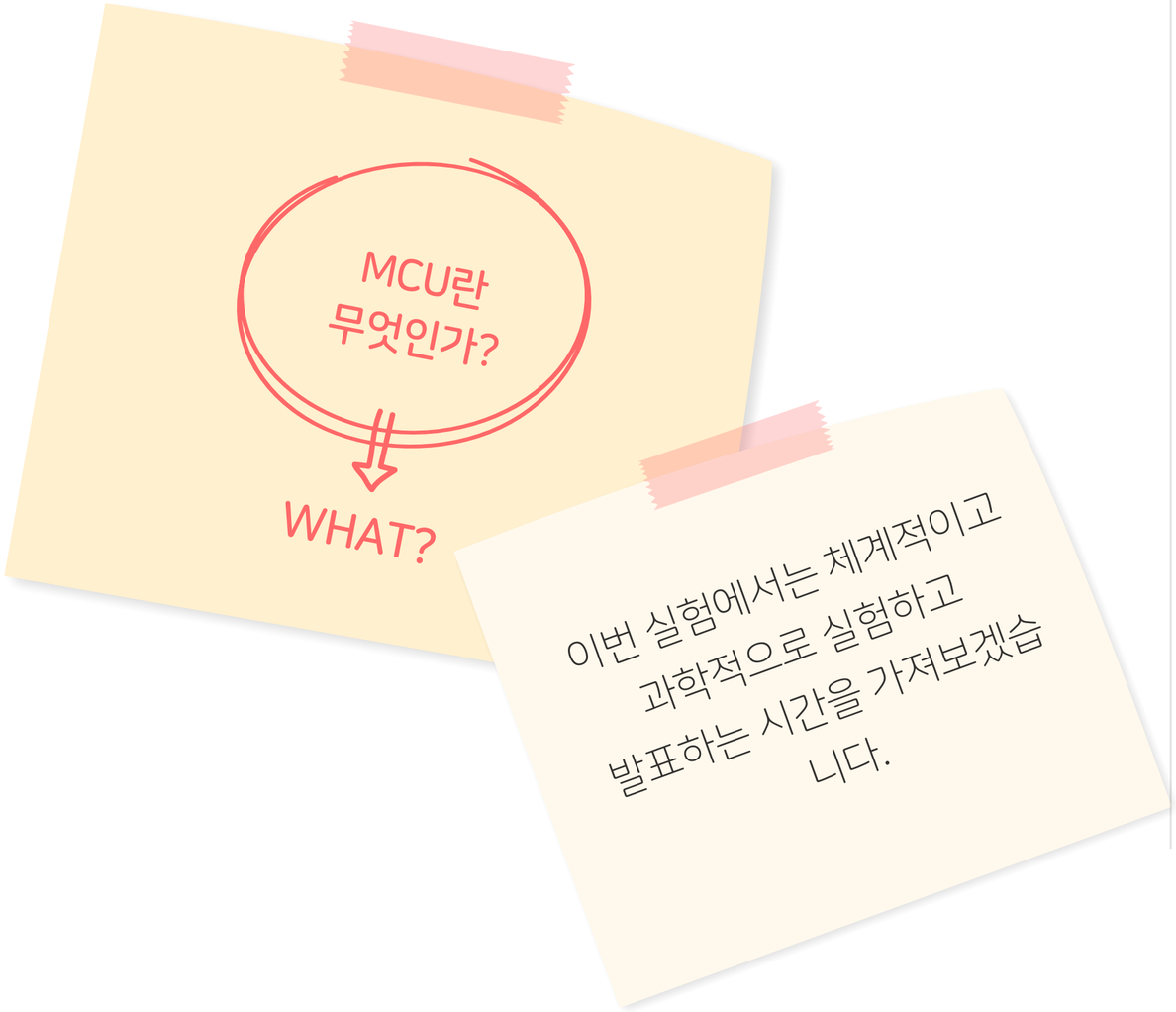
기본 실험의 정석®

임베디드시스템설계및실험 9조

김태경 최정혜
강준우 최상준
여지수

발표

차례®

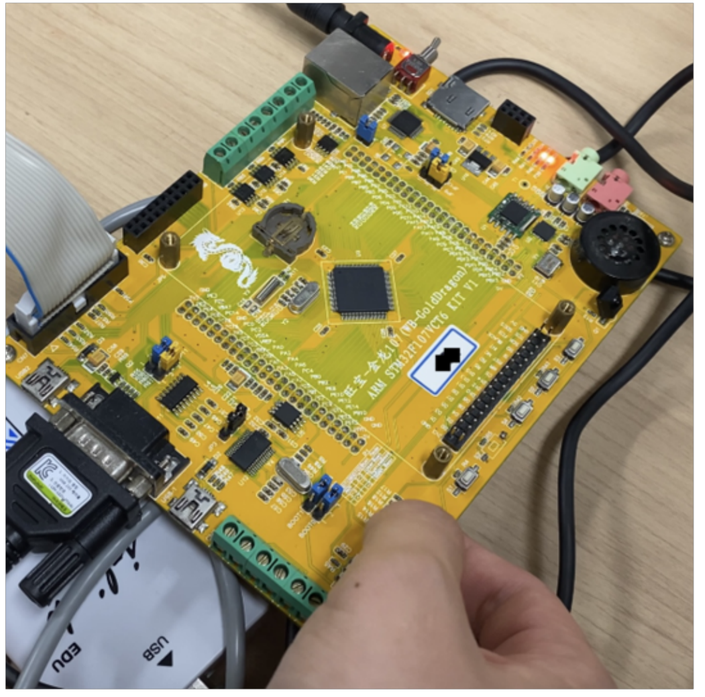


MCU

MCU

마이크로컨트롤러(microcontroller) 또는 **MCU**(microcontroller unit)는 마이크로 프로세서와 입출력 모듈을 하나의 칩으로 만들어 정해진 기능을 수행하는 컴퓨터를 말한다.

*위키백과 : 마이크로컨트롤러



Stm32 MCU를 장착한 실험 키트

GPIO functional description

Each of the general-purpose I/O ports has two 32-bit configuration registers (GPIOx_CRL, GPIOx_CRH), two 32-bit data registers (GPIOx_IDR, GPIOx_ODR), a 32-bit set/reset register (GPIOx_BSRR), a 16-bit reset register (GPIOx_BRR) and a 32-bit locking register (GPIOx_LCKR).

Subject to the specific hardware characteristics of each I/O port listed in the *datasheet*, each port bit of the General Purpose IO (GPIO) Ports, can be individually configured by software in several modes:

- Input floating
- Input pull-up
- Input-pull-down
- Analog
- Output open-drain
- Output push-pull
- Alternate function push-pull
- Alternate function open-drain

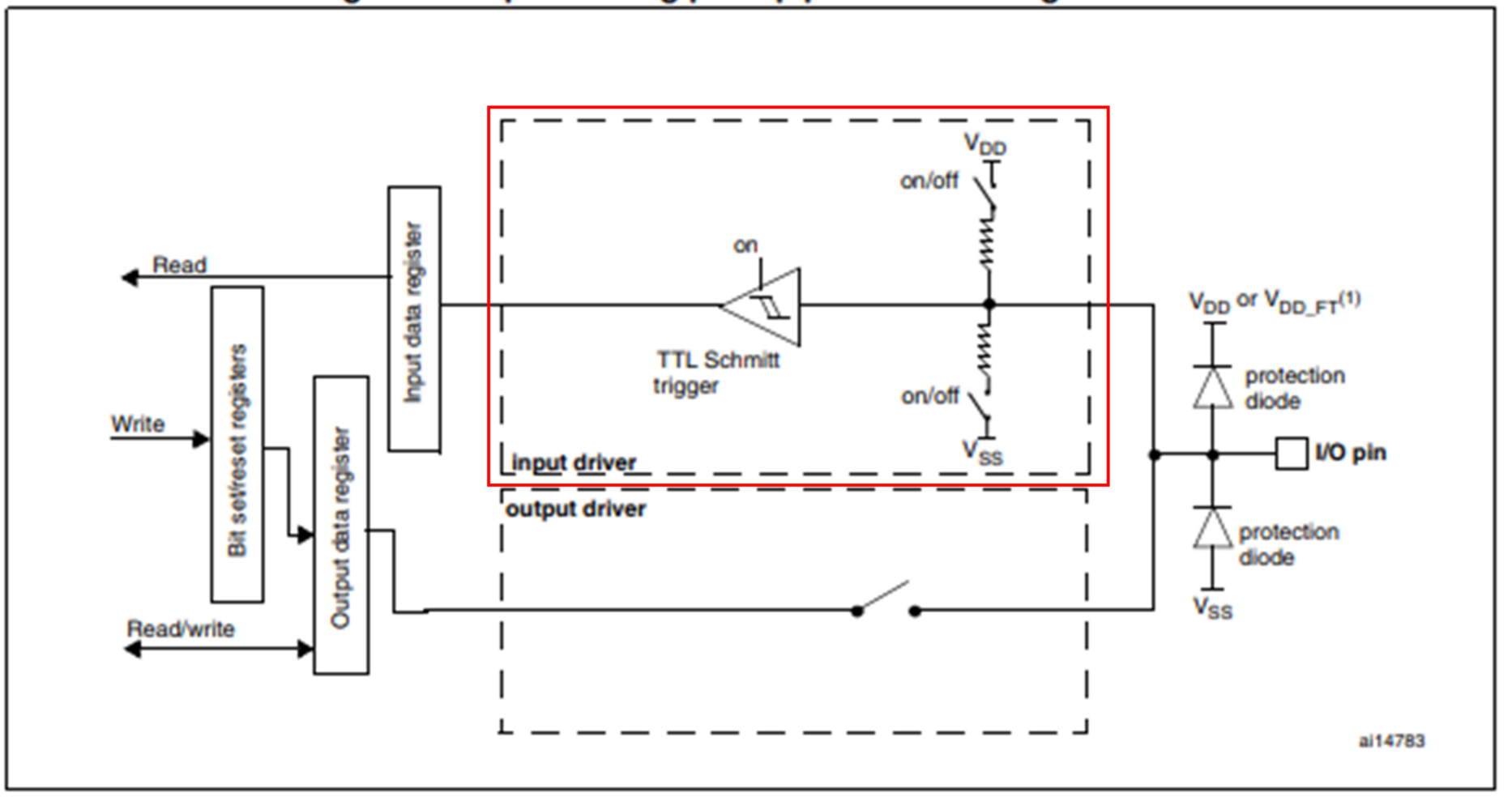
Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words (half-word or byte accesses are not allowed). The purpose of the GPIOx_BSRR and GPIOx_BRR registers is to allow atomic read/modify accesses to any of the GPIO registers. This way, there is no risk that an IRQ occurs between the read and the modify access.

Figure 13 shows the basic structure of an I/O Port bit.

*stm32_ReferenceManual_p.159

Floating / Pull-up / Pull-down

Figure 15. Input floating/pull up/pull down configurations



1. V_{DD_FT} is a potential specific to five-volt tolerant I/Os and different from V_{DD}.

*stm32_ReferenceManual_p.163

Interrupt와 Polling의 개념

01

Interrupt와 Polling의 공통점

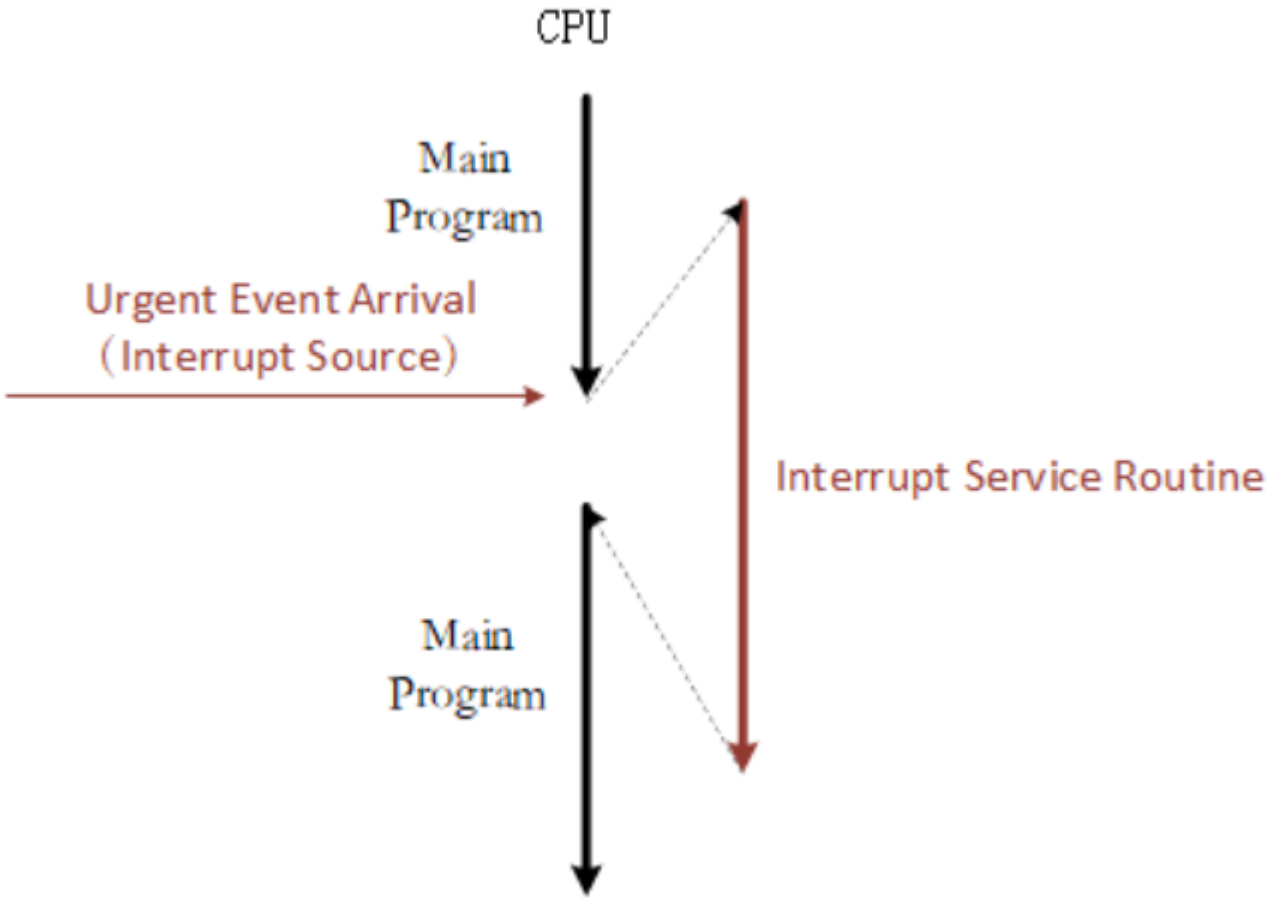
Interrupt와 Polling은 CPU가 다른 장치들로부터 발생하는 이벤트를 처리하는 **방법**을 뜻한다. 두 가지 방식 모두 CPU가 하던 일을 멈추게 하여 CPU가 더 중요한 일에 반응할 수 있는 상태로 만든다.

02

Interrupt와 Polling의 차이점

Interrupt는 **새치기**, Polling은 **반복확인**이다.

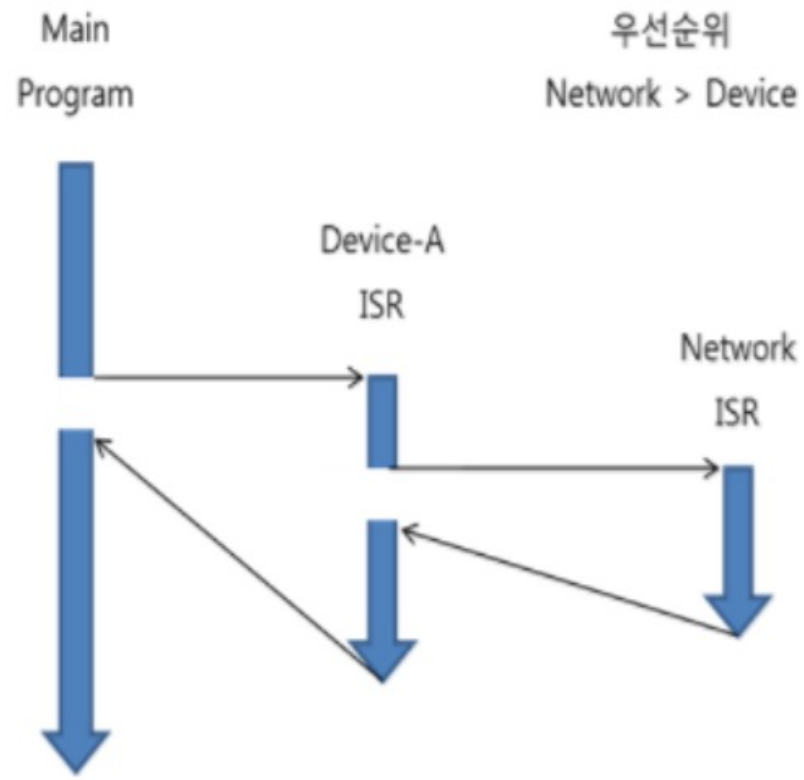
Interrupt 개념과 장점



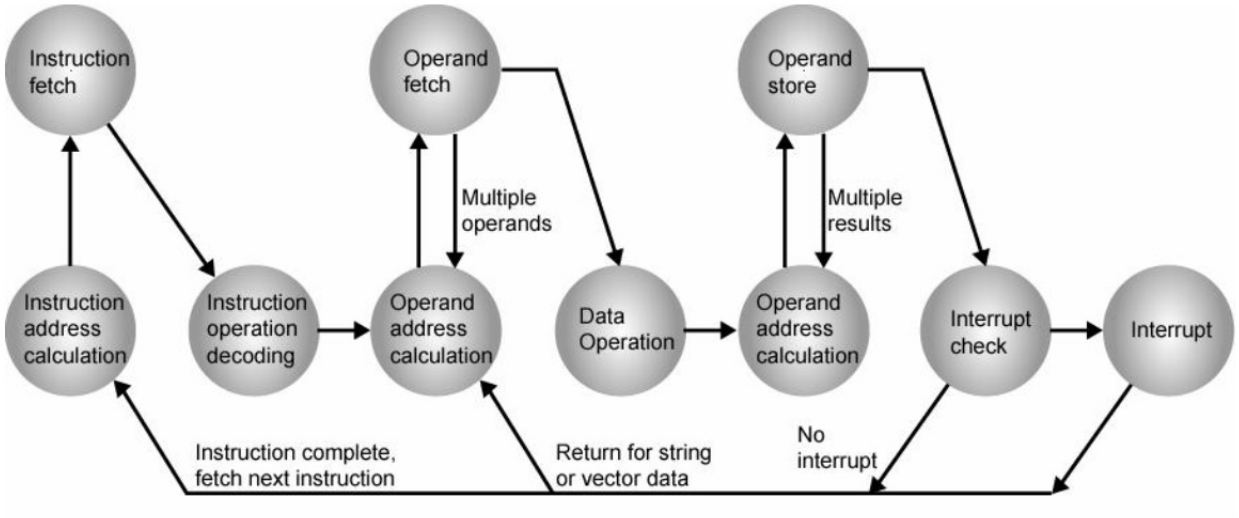
Event에 의해 Interrupt가 발생하면 Interrupt를 관리하는 장치인 Interrupt-Handler에 의해 CPU는 **하던 일을 중지하고**, Event를 우선적으로 처리한 이후 **하던 일**을 다시 한다.
(Hardware로 Event를 처리)

예를 들면, 자동차 사고시 에어백 작동과 같이 긴급하고 우선적으로 처리되어야 하는 일은 Interrupt방식으로 처리되어야 할 것이다.

Interrupt 개념과 장점

개념도	설명
	– 인터럽트의 우선순위를 정하고, 우선순위가 낮은 인터럽트 처리 중 높은 인터럽트가 들어오면 현재 인터럽트 서비스 루틴을 중단하고 새로운 인터럽트 처리 수행

Interrupt 단점

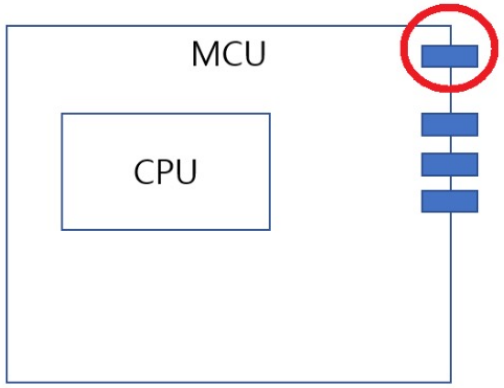


절차와 구현이 복잡하다.

Interrupt의 절차는

Device -> Handler ->
Interrupt 발생 -> CPU ->
기존 instruction 주소 저장 ->
new_instruction 실행 ->
주소 복귀 -> 기존 instruction 실행

Polling 개념과 장점



그렇다면 아래와 같이 코드가 구현될 것입니다.

```c
int main(void)
{
    while(1u)
    {
        pin_state = mcu_pin_state();
        if(pin_state == 1)
        {
            AirBag_Operation();
        }
    }

    return 0;
}
```

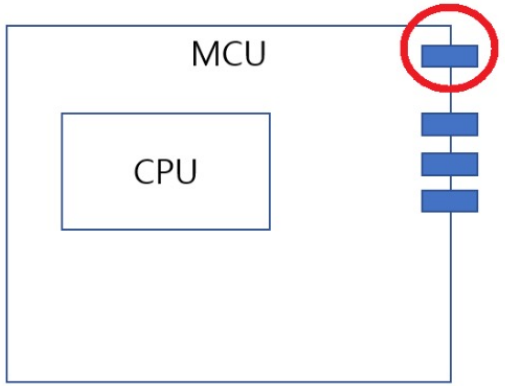
Polling은 특정 주기마다 반복해서 확인하는 것

너무나 쉬운 구현

Polling은 while loop로 구현할 수 있다.
(Software로 Event를 처리한다.)

예를 들어, Airbag이 터져야 하는 상황임을 특정 주기마다 확인해야한다면 어떻게 구현해야 할까요?

Polling의 단점



그렇다면 아래와 같이 코드가 구현될 것입니다.

```
int main(void)
{
    while(1u)
    {
        pin_state = mcu_pin_state();
        if(pin_state == 1)
        {
            AirBag_Operation();
        }
    }

    return 0;
}
```

Polling은 특정 주기마다 반복해서 확인하는 것

Polling은 while loop로 구현할 수 있다. (Software로 Event를 처리한다.)

예를 들어, Airbag이 터져야 하는 상황임을 특정 주기마다 확인해야한다면 어떻게 구현해야 할까요?

리소스를 굉장히 많이 잡아먹는다.

-> While loop를 도는 동안 CPU는 아무 일도 하지 못한다.

Interrupt가 항상 유리한가?

꼭 그렇지는 않다.

1 는 Interrupt가 발생한 순서
→는 Overhead (처리하는 일을 변경하는 데 걸리는 시간 혹은 비용)

다음 그림과 같이 진행되었을 때 1번 일은 언제 마무리가 되는가?

일은 마무리되지 않고 지역간의 이동에 쓰이는 비용(Overhead) 커진다.

Overhead 동안 아무 일을 못한다.
이동이 잦고 비용이 큰 경우 비효율적이다.

개발자로서 지향할 목표

개발자로서 지향할 목표

촌각을 다투는 100m 달리기 선수는 Interrupt로 구현해야 할까? Polling으로 구현해야 할까?

개발자로서 지향할 목표

촌각을 다투는 100m 달리기 선수는
Interrupt로 구현해야 할까?
Polling으로 구현해야 할까?

다른 일을 하다 입력이 오면
뛰게 하는 것이 맞을까?

아무것도 하지 않은 채로
출발선에 대기시키는 것이
맞을까?

Scatter file

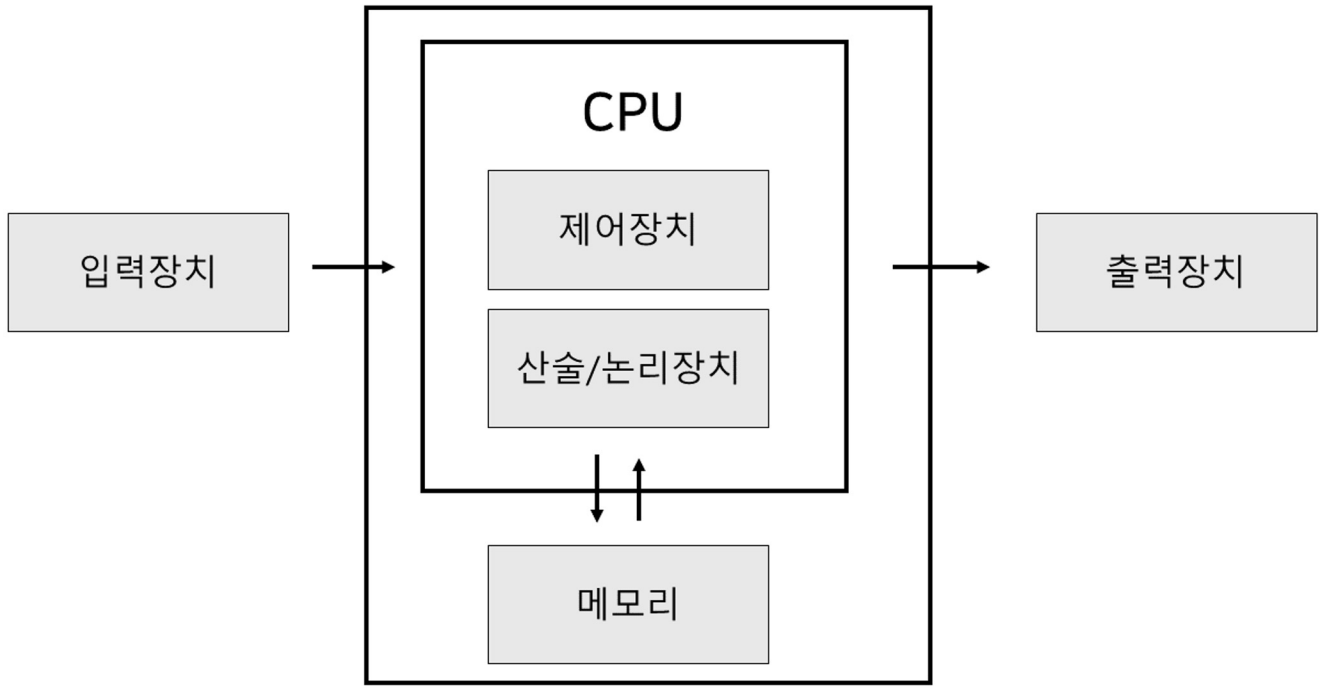
01 폰 노이만 구조

02 메모리의 종류

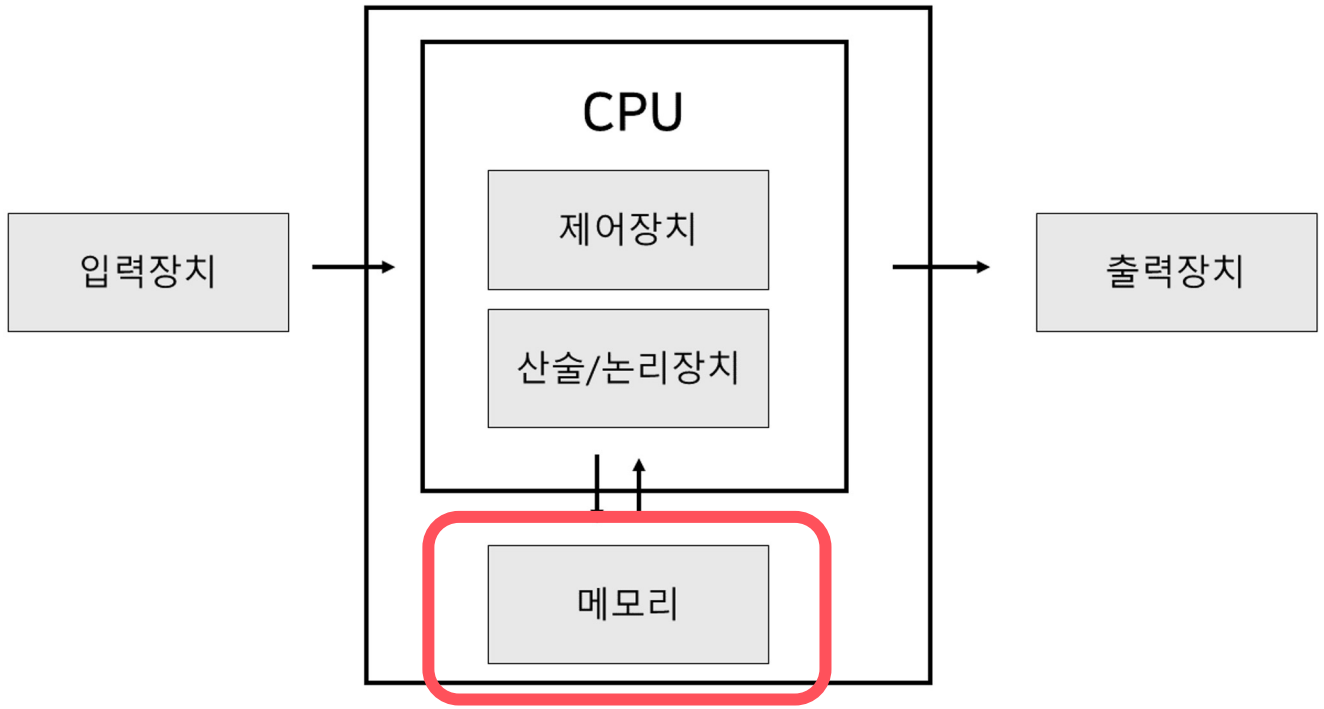
03 Scatter File

04 코드 분석

01 폰 노이만 구조



01 폰 노이만 구조



02 메모리의 종류

ROM (Read Only Memory)

비휘발성 메모리
전기가 끊겨도 데이터들이 지워지지 않도록 저장 가능

RAM (Random Access Memory)

휘발성 메모리
전원이 꺼지면 가지고 있던 데이터가 사라짐

02 메모리의 종류

플래시 메모리 Flash Memory

전기적으로 데이터를 지우고 다시 기록할 수 있는 “비휘발성” 컴퓨터 기억장치

02 메모리의 종류

플래시 메모리 Flash Memory

전기적으로 데이터를 지우고 다시 기록할 수 있는 “비휘발성” 컴퓨터 기억장치

즉, 전원을 연결하지 않아도 메모리 안에 있는 데이터가 유지 될 수 있다.

02 메모리의 종류

STM32에서도 비휘발성 메모리 사용 가능

02 메모리의 종류

STM32에서도 비휘발성 메모리 사용 가능

=> Scatter File을 이용해서

03 Scatter File

Scatter File이란

ROM과 REM에 메모리를 할당하는 파일

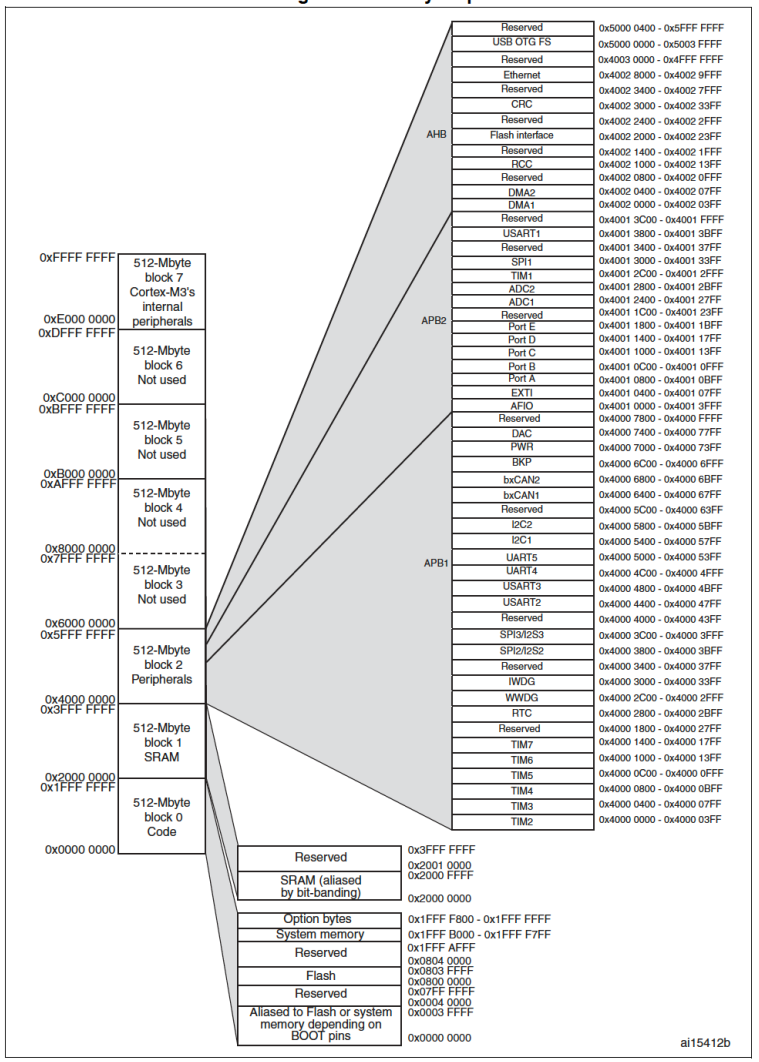
03 Scatter File

Table 2. STM32F105xx and STM32F107xx features and peripheral counts

Peripherals[1]		STM32F105Rx			STM32F107Rx		STM32F105Vx			STM32F107Vx	
Flash memory in Kbytes		64	128	256	128	256	64	128	256	128	256
SRAM in Kbytes		64									
Package		LQFP64					LQFP 100	LQFP 100, BGA 100	LQFP 100	LQFP 100	LQFP 100, BGA 100
Ethernet		No			Yes		No			Yes	
Timers	General-purpose	4									
	Advanced-control	1									
	Basic	2									

03 Scatter File

Figure 5. Memory map



04 코드 분석

메모리 할당 및 정의

Source file : stm32f10x.h

```
#define FLASH_BASE            ((uint32_t)0x08000000) /*!< FLASH base address in the alias region */
#define SRAM_BASE             ((uint32_t)0x20000000) /*!< SRAM base address in the alias region */
```

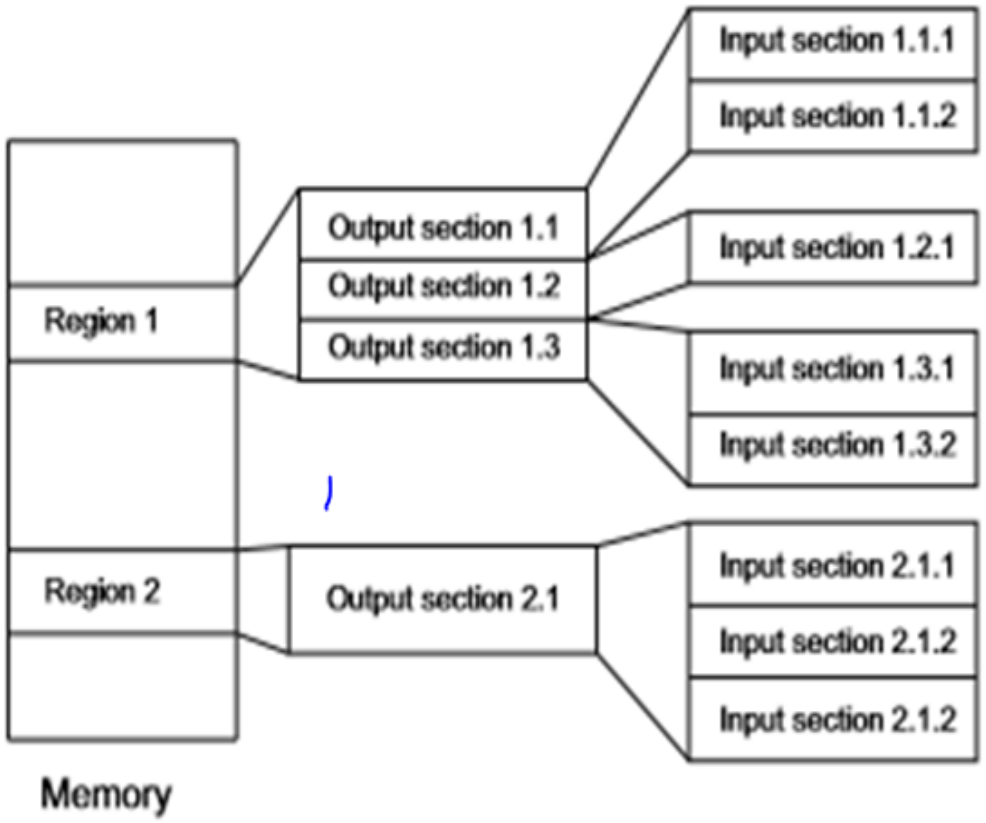
04 코드 분석

메모리 핸들링

Source file : startup_stm32f10x_cl.s

```
__vector_table
        DCD     sfe(CSTACK)
        DCD     Reset_Handler             ; Reset Handler
        DCD     NMI_Handler               ; NMI Handler
        DCD     HardFault_Handler         ; Hard Fault Handler
        DCD     MemManage_Handler         ; MPU Fault Handler
        DCD     BusFault_Handler          ; Bus Fault Handler
        DCD     UsageFault_Handler        ; Usage Fault Handler
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
        DCD     0                         ; Reserved
        DCD     SVC_Handler               ; SVCall Handler
        DCD     DebugMon_Handler          ; Debug Monitor Handler
        DCD     0                         ; Reserved
        DCD     PendSV_Handler            ; PendSV Handler
        DCD     SysTick_Handler           ; SysTick Handler
```

Scatter file 코드 분석



01

Input Section

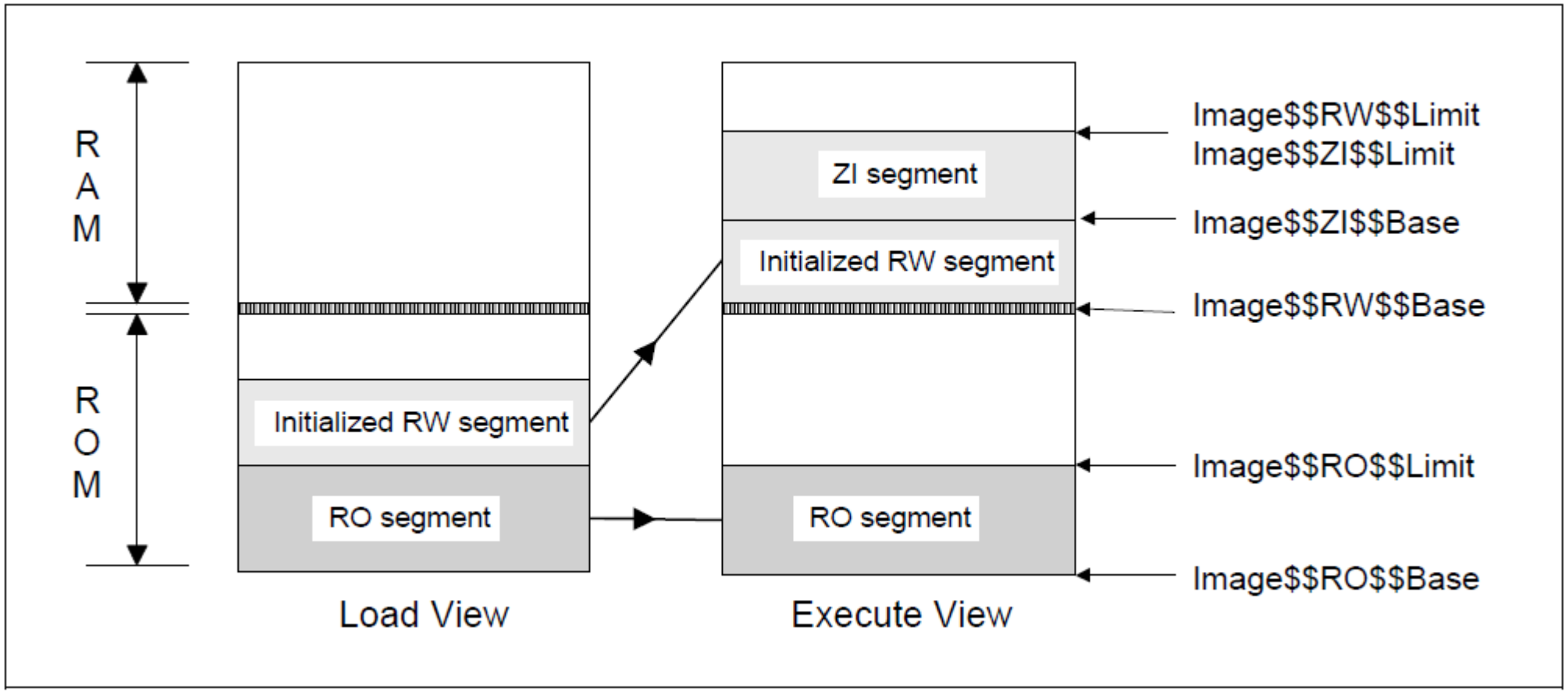
1. RO(Read Only)
2. RW(Read-Write)
3. ZI(Zero Initialized)

02

Output Section, Region

Input section 들이 모여 Output Section, Region, Image 로 불리는 더 큰 Block 을 구성

Scatter file 코드 분석

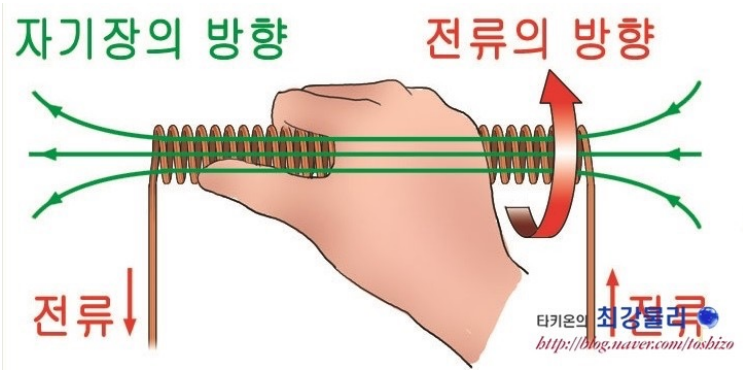


출처

출처

1. https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=pjsin865&logNo=120060750272
   Scatter description file 간단히 이해하기 _1
2. https://m.blog.naver.com/pjsin865/120060751669ZI(Zero Initialized) Scatter description file 간단히 이해하기 _2
3. https://www.ARM.com

릴레이 모듈



릴레이 모듈이란?

- 😊 릴레이(Relay)를 제어할 수 있는 모듈
- 😊 전자석을 이용한 원리
- 😊 스위치의 ON, OFF 와 같은 기능

릴레이의 NO 와 NC

- 😊 NO (Normal Open) 회로
  -> 평소 Open, 신호 오면 Close
- 😊 NC (Normal Close) 회로
  -> 평소 Close, 신호 오면 Open

The End

감사합니다.