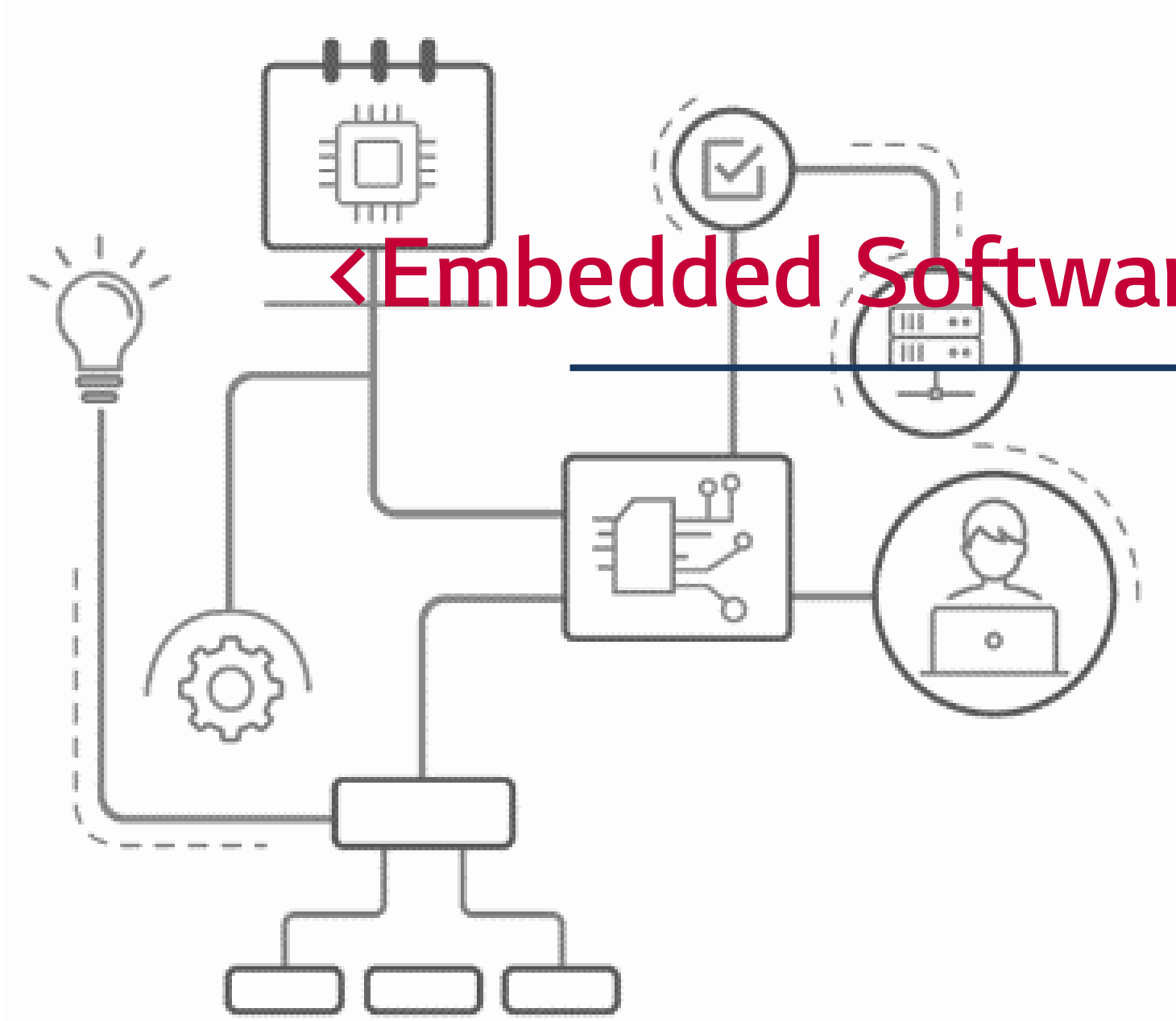


# <Embedded Software의 이해>

# 이번 교육에서 배울 내용은...


**MCU 구조를 이해하고 Embedded Software 개발 기초 이론을 습득**

## M1 Embedded System

- 임베디드 시스템이란?
- 임베디드 소프트웨어 특징
- 임베디드 개발 환경

## M2 MCU 구성요소 및 CPU 동작 원리

- MCU 구조
- CPU 구성 요소
  (CPU, 메모리, 보조모듈)
- CPU 동작 원리

## M3 Memory

- Memory 종류
- Memory 영역별 개념

## M4 Compile과 Link

- Compile stage
- Linking과 Symbol table

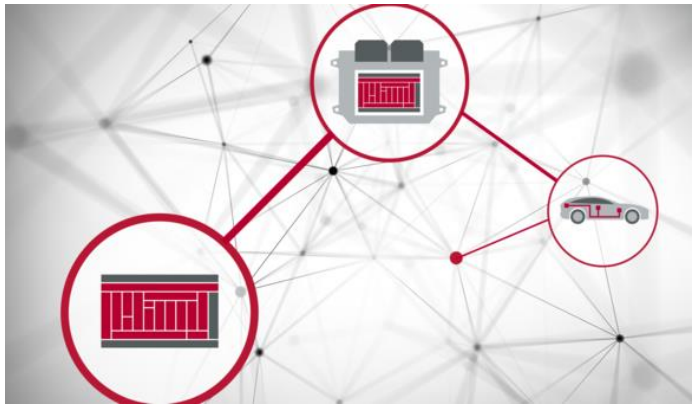
## M5 Firmware 동작의 이해

- 폴링과 인터럽트
- Reset flow
- Interrupt handling flow

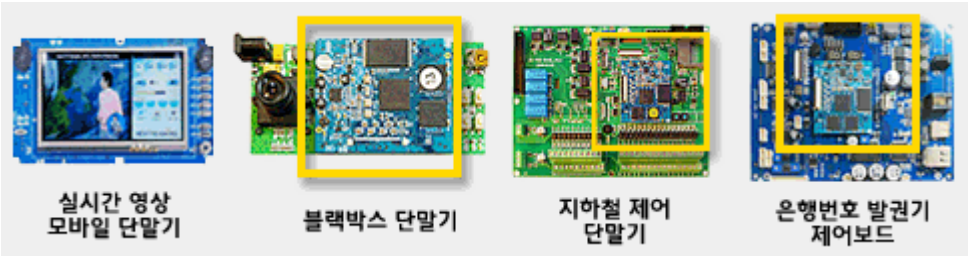
M1

# Embedded System

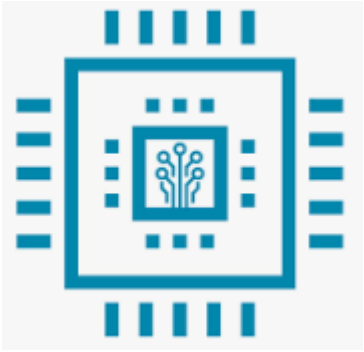
# Embedded System



'임베디드' = '내장된'



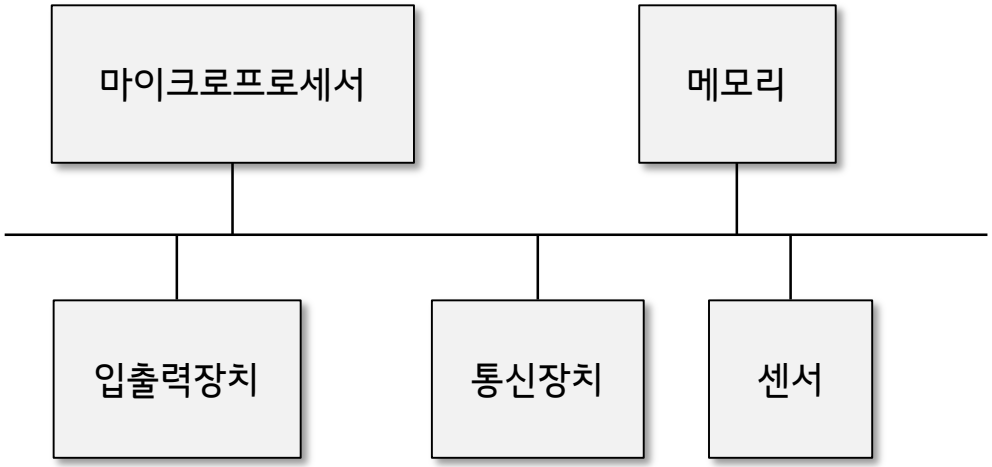

특수 목적을 위한 시스템



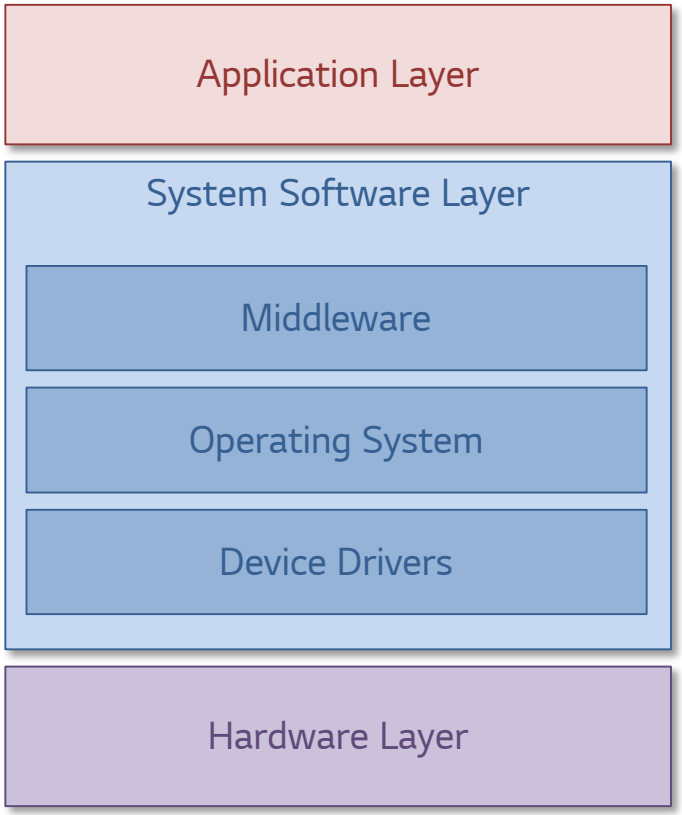
하드웨어 + 소프트웨어

최적화, 실시간성 보장

# Embedded System




임베디드 하드웨어




임베디드 소프트웨어

# Embedded Software

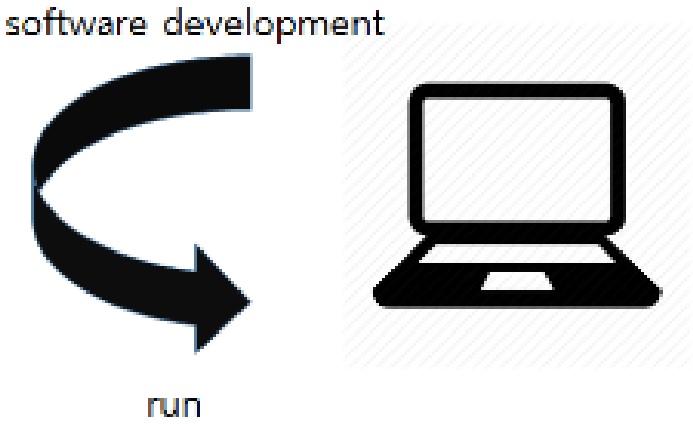
## 임베디드 소프트웨어 특징

- 특정 제품에서만 동작하는 S/W
- 실시간성, 자원제한성, 고 신뢰성 요구
- H/W에 대한 이해가 필요
- 같은 기능이라도 다양한 H/W에 맞추어 개발되어야 함
- 교차 개발 (Cross compile)

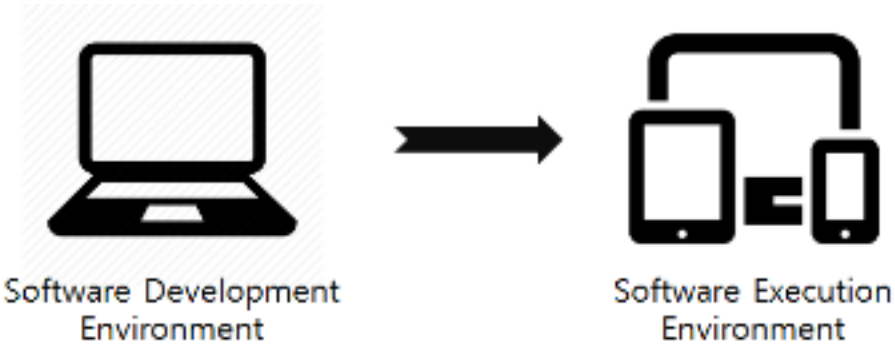

*** Host System : 교차 개발 환경에서 소프트웨어를 개발하는 시스템**

*** Target System : 소프트웨어가 작동될 환경, 핸드폰이나, PDA 시스템과 같은 것**

# Embedded Software




네이티브(Native) 컴파일러



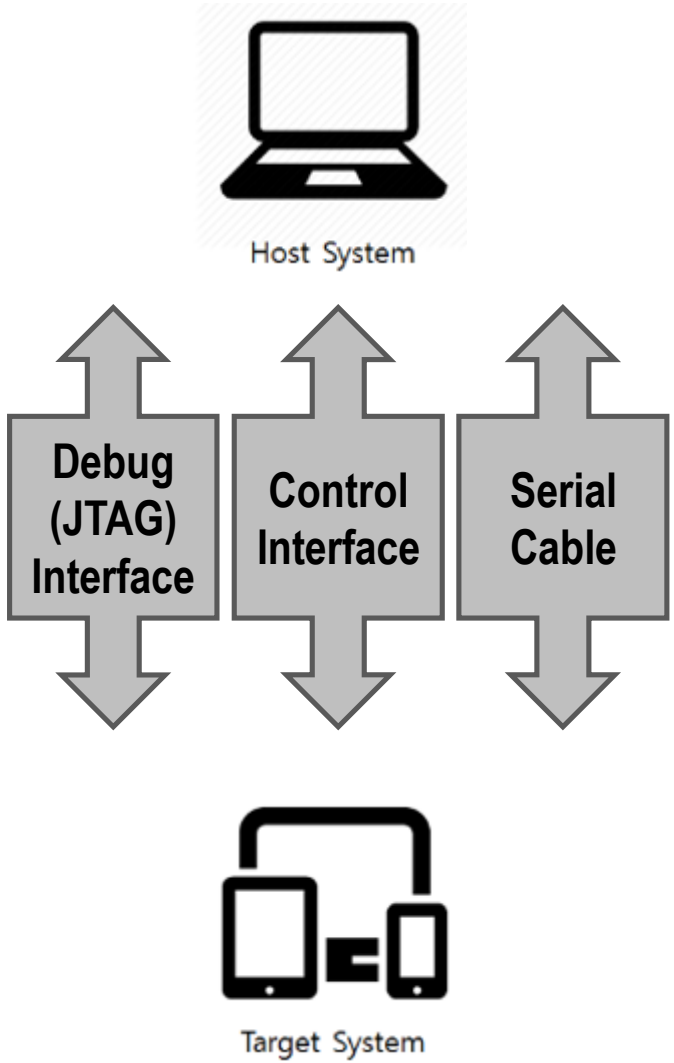

크로스(Cross) 컴파일러

## 임베디드 소프트웨어 개발의 특징

- S/W 개발 시스템과 S/W 동작 시스템이 다름 (크로스 컴파일러 필요)
- 개발 및 컴파일 시, 타겟 시스템 구성에 대해 상세히 알아야 함
- 프로그램 실행 시의 메모리 구성에 대해 충분히 고려해야 함

# Embedded Software 개발 환경

1. *툴체인(Tool-chain) 설치
2. 디버거(Debugger), 디버깅 툴 설치
3. 다운로드 유틸리티
4. 통신 케이블 연결

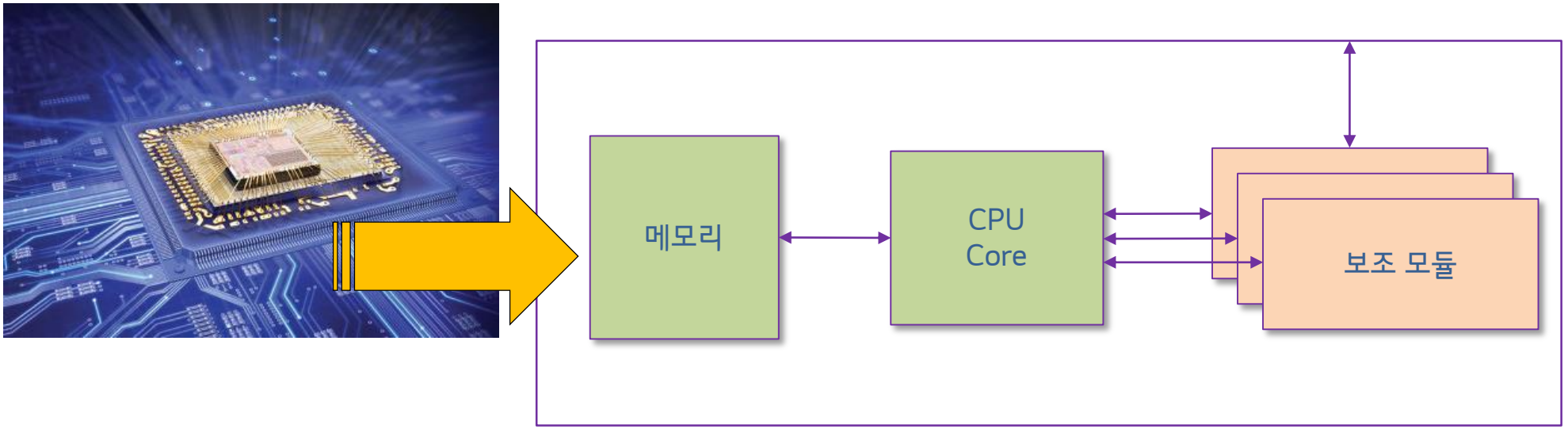



* 툴체인(Tool-chain): Target System에서 동작하는 프로그램 개발에 필요한 Host System의 소프트웨어들 또는 개발 환경을 통칭

M2

# MCU 구성요소 및 CPU 동작 원리

# MCU (Micro Controller Unit)



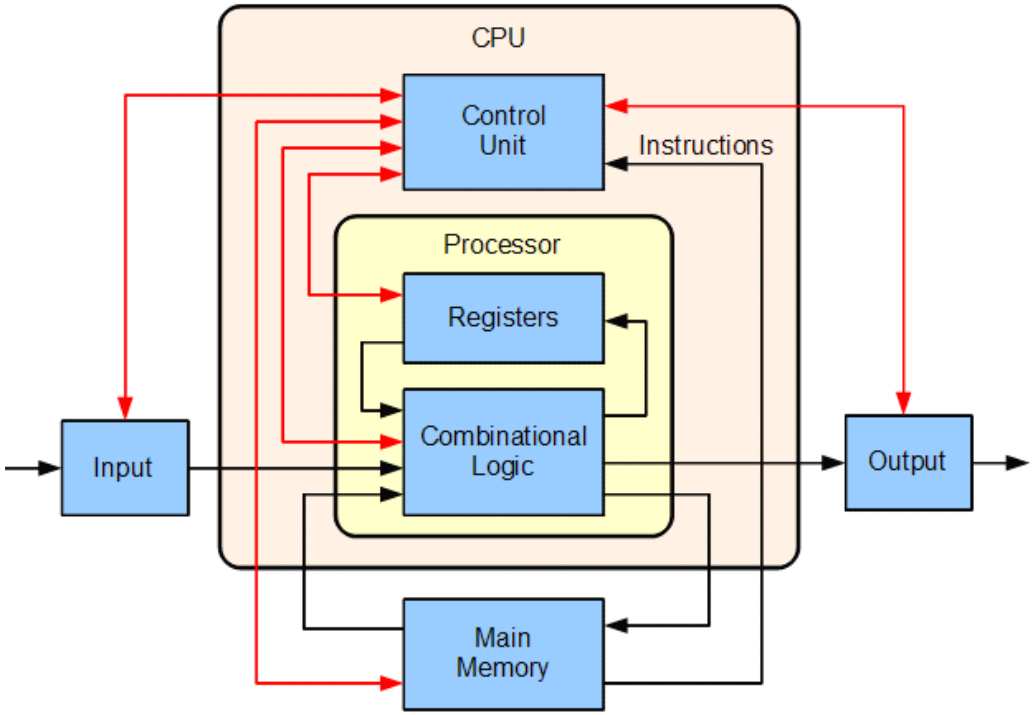

**Micro Controller Unit = CPU Core + 메모리 + 보조 모듈**

## MCU 란?

- CPU(Central Processing Unit), 입출력 모듈, 여러 종류의 메모리 및 주변장치들을 함께 집적하여 정해진 기능을 수행하는 칩

# MCU 구성요소 - CPU

## CPU Architecture



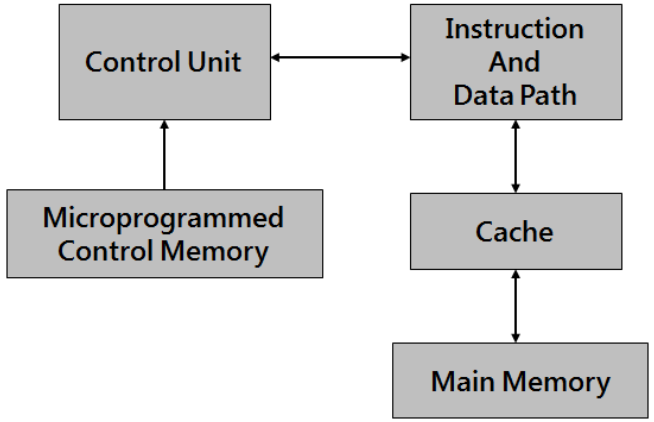

**Arithmetic / Logic Unit**
→ 비교, 판단, 연산을 담당

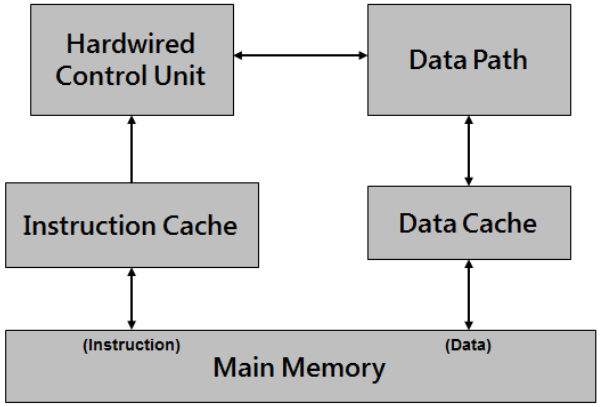
**Registers**
→ CPU에서 처리할 명령어, 연산에 필요한 데이터를 임시로 저장

**Control Unit**
→ ALU, Register와 통신
명령어 실행을 위해 CPU를 내부적으로 제어

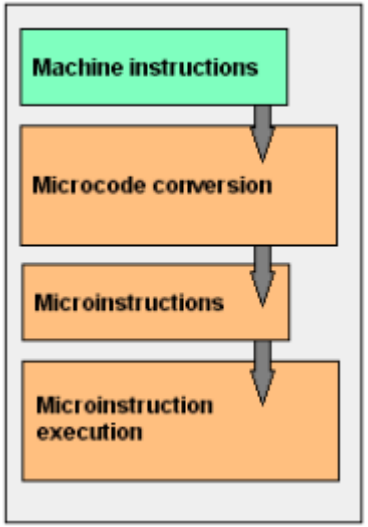
# MCU 구성요소 - CPU

## CISC & RISC Architecture



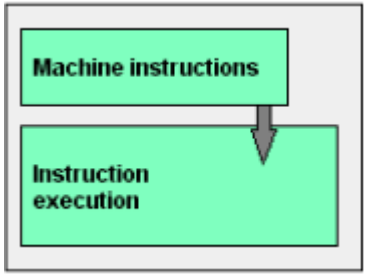




**CISC** (Complex Instruction Set Computer)




```
r1 ← [addr(A) + 4 * r2] + 1
ret
```

**RISC** (Reduced Instruction Set Computer)




```
r3 ← 4 * r2
r3 ← addr(A) + r3
r5 ← [r4]
r1 ← r5 + 1
ret
```

# MCU 구성요소 - CPU

## CISC & RISC 비교

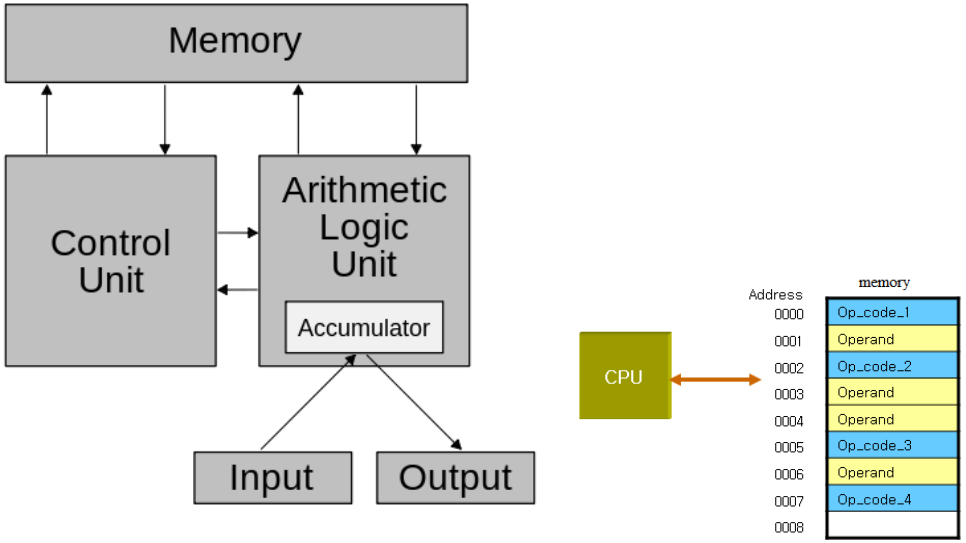
| 구분 | CISC | RISC |
|---|---|---|
| 구성 | 명령어 복잡, 명령어 해석에 소모되는 시간이 큼 | CPU 명령어 개수를 줄이고 명령어가 단순함 |
| 명령어 길이 | 명령어 길이가 가변적 | 명령어의 길이가 고정 (Pipeline 처리 고속화 ) |
| 명령어 효율 | 복잡한 명령이더라도 microcode이므로 실행 효율이 좋음, 작은 코드 크기 | Microcode logic을 사용하지 않으므로 단위시간동안 낮은 사이클수, 큰 코드 크기 |
| 메모리 접근 | 대부분의 명령어가 메모리의 operand에 접근 | 많은 수의 레지스터를 사용, 메모리 접근 줄임 메모리 접근은 load / store 명령어로 제한 (회로 단순) |
| 연산 | 레지스터-레지스터 연산, 레지스터-메모리 연산, 메모리-메모리 연산 모두 갖춤 | 레지스터 사이의 연산만 실행 |

**프로그래밍 용이, 설계 복잡**

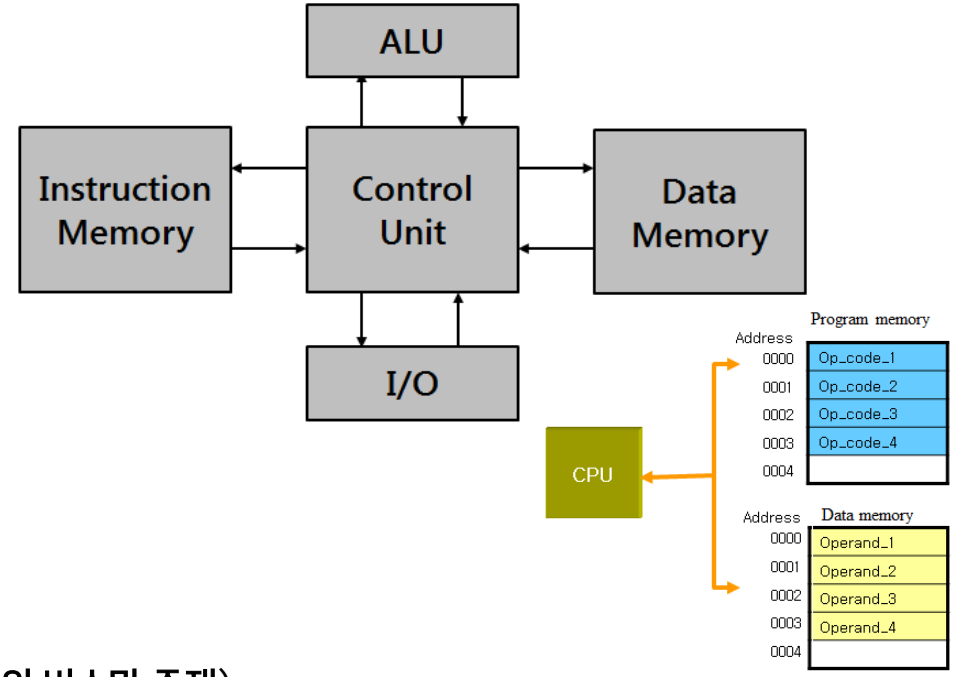
**설계 용이 , 프로그래밍 복잡**

# 폰노이만과 하버드 구조

## Von Neumann Architecture




## Harvard Architecture




## ‖ 폰노이만 구조 (Von Neumann Architecture)

- 프로그램과 데이터가 메모리를 공유 (CPU와 메모리 사이에 **하나의 버스만 존재**)
- 명령어를 읽을 때 데이터를 읽거나 쓸 수 없음
- 명령어 실행을 위해 프로그램과 데이터 2번의 Fetch cycle이 필요해 상대적으로 저속

## ‖ 하버드 구조 (Harvard Architecture)

- 프로그램 메모리와 데이터 메모리가 분리 (CPU는 데이터 메모리와 프로그램 메모리 **버스를 분리**하여 사용)
- 명령어를 읽을 때 데이터를 읽거나 쓸 수 있음
- 동시에 프로그램 메모리와 데이터 메모리 접근이 가능하여 상대적으로 고속처리가 가능

# MCU 구성요소 – Memory

## Instruction Memory

- 프로그램이 Running중에는 Read만 가능한 논리적인 개념의 메모리
- CPU가 Fetch하고 실행할 수 있는 명령어(Instruction)가 저장된 메모리
- Byte Addressable 해야 함
- PC(Program Counter)에 Set되는 영역

## Data Memory

- CPU가 OP Code와 함께 사용하는 Operand data가 저장된 메모리
- 프로그램이 Running중에 Data를 Read하거나 임의의 값으로 Write할 수 있음
- 일반적으로 R/W가 가능한 RAM 영역을 사용

# MCU 구성요소 – Peripheral Module

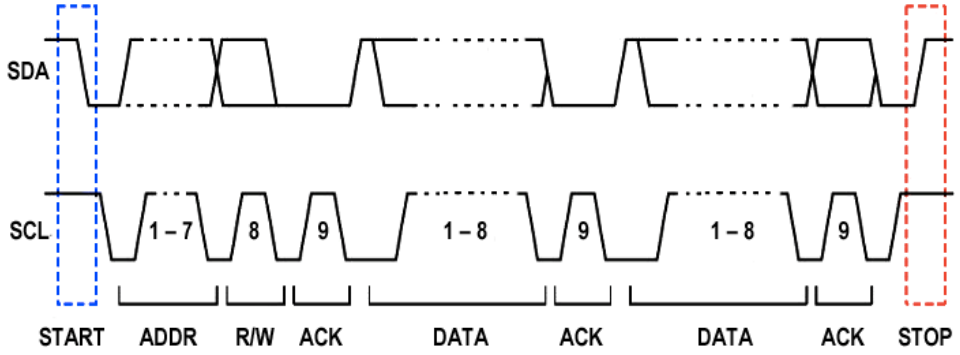
## 보조 모듈

통신 모듈
- MCU 외부 장치와 데이터를 주고 받음 (I2C, SPI, UART)

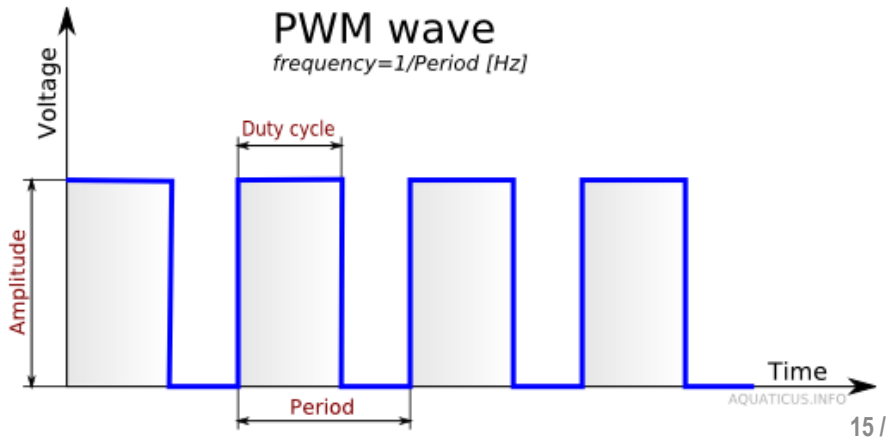
ADC/DAC
- 디지털 데이터를 처리하는 MCU가 아날로그 데이터를 처리하는 외부 장치와 함께 사용되었을 때, 아날로그 데이터를 디지털 데이터로 전환하는 작업을 수행

타이머 모듈
- MCU의 System Clock을 기준으로 일정한 주기로 작업을 수행

PWM
- Pulse Width Modulation (펄스 폭 변조)의 약자로, 일정한 주기 내에서 Duty 비를 변화시켜 평균 전압을 제어하는 방법. 주로 LED조명제품의 광량이나 DC모터의 속도제어 조절 컨트롤러에 사용됨.

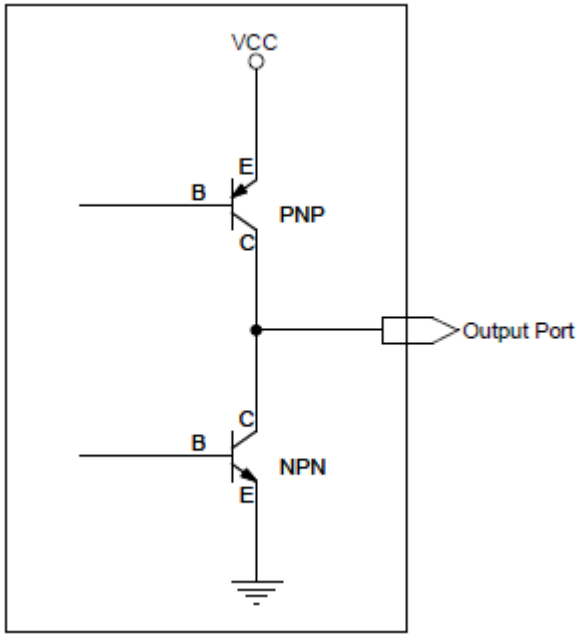



*The I²C Protocol*

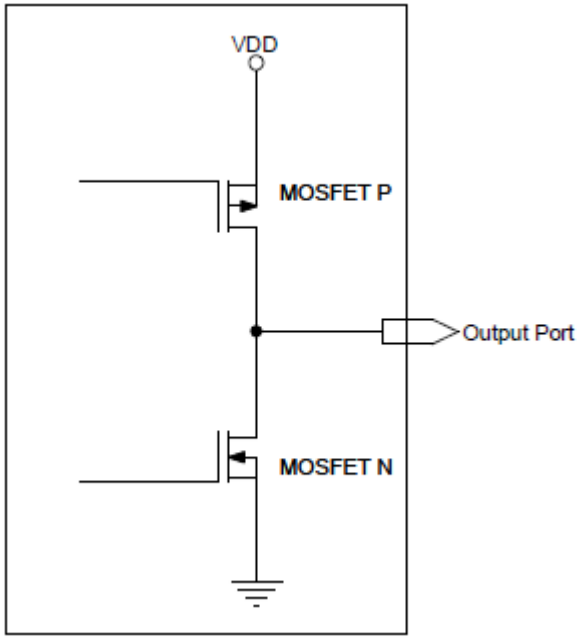


# MCU 입출력회로 구성 – Push pull Vs Open drain

## Push-pull 출력




BJT 회로

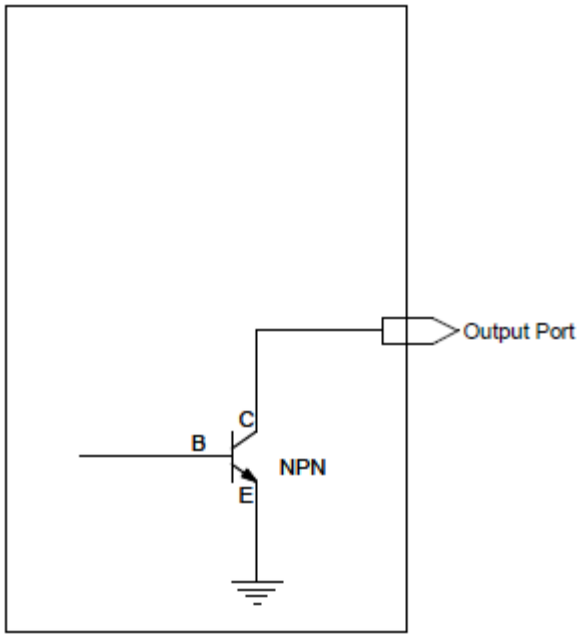



CMOS 회로

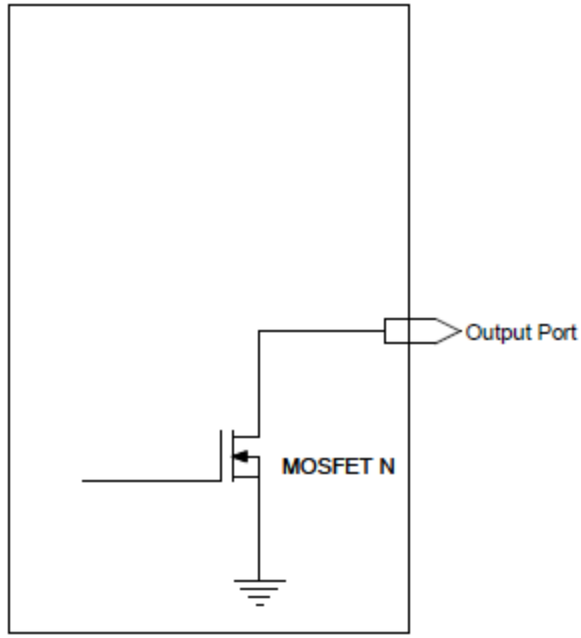
- 2 개의 TR로 구성
- NPN TR은 GND로 끌어 당기는(Pull) 역할
- PNP TR은 VCC쪽으로 밀어 올리는(Push) 역할

# MCU 입출력회로 구성 – Push pull Vs Open drain

## Open-collector, Open-drain 출력




Open Collector

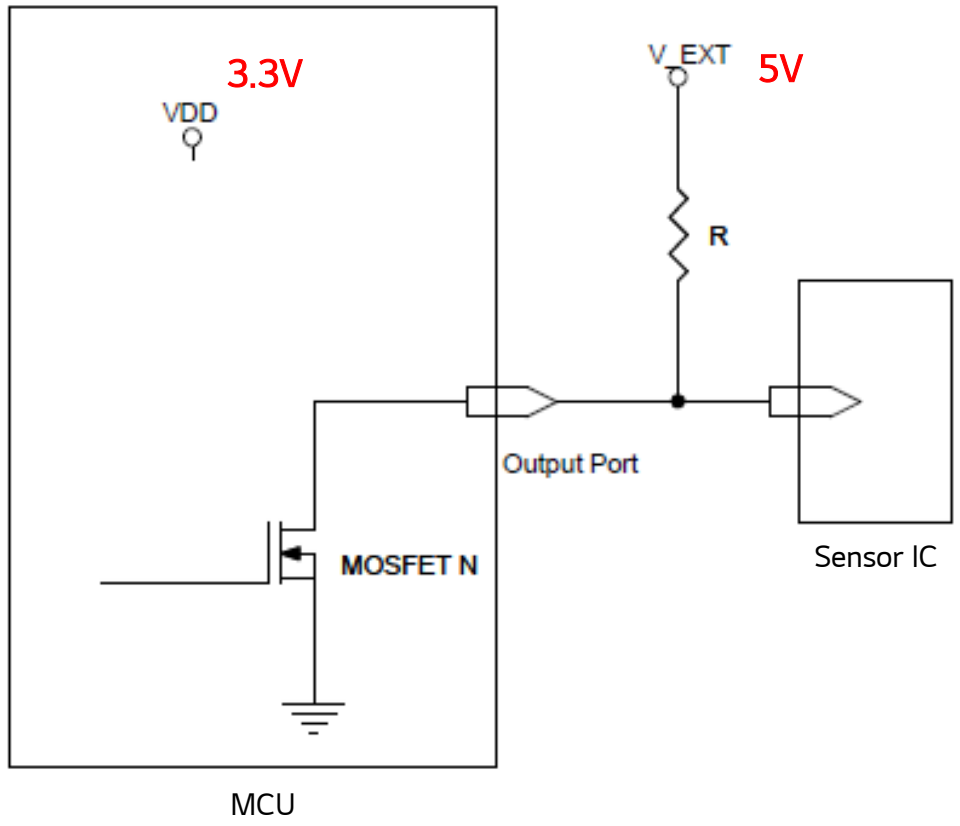



Open Drain

- VCC로 밀어 올려주는 위쪽 회로가 없음
- NPN TR 혹은 N-MOS로 Output을 GND로 당김 (0V)
- NPN TR이나 N-MOS가 OFF시 Output은 Unknown state가 됨
- 외부에 부가 회로가 필요
- Level converter(Level shifter)나 Bus network 구성을 위해 활용

# MCU 입출력회로 구성 – Push pull Vs Open drain

## ‖ Open-drain을 활용한 Level converter

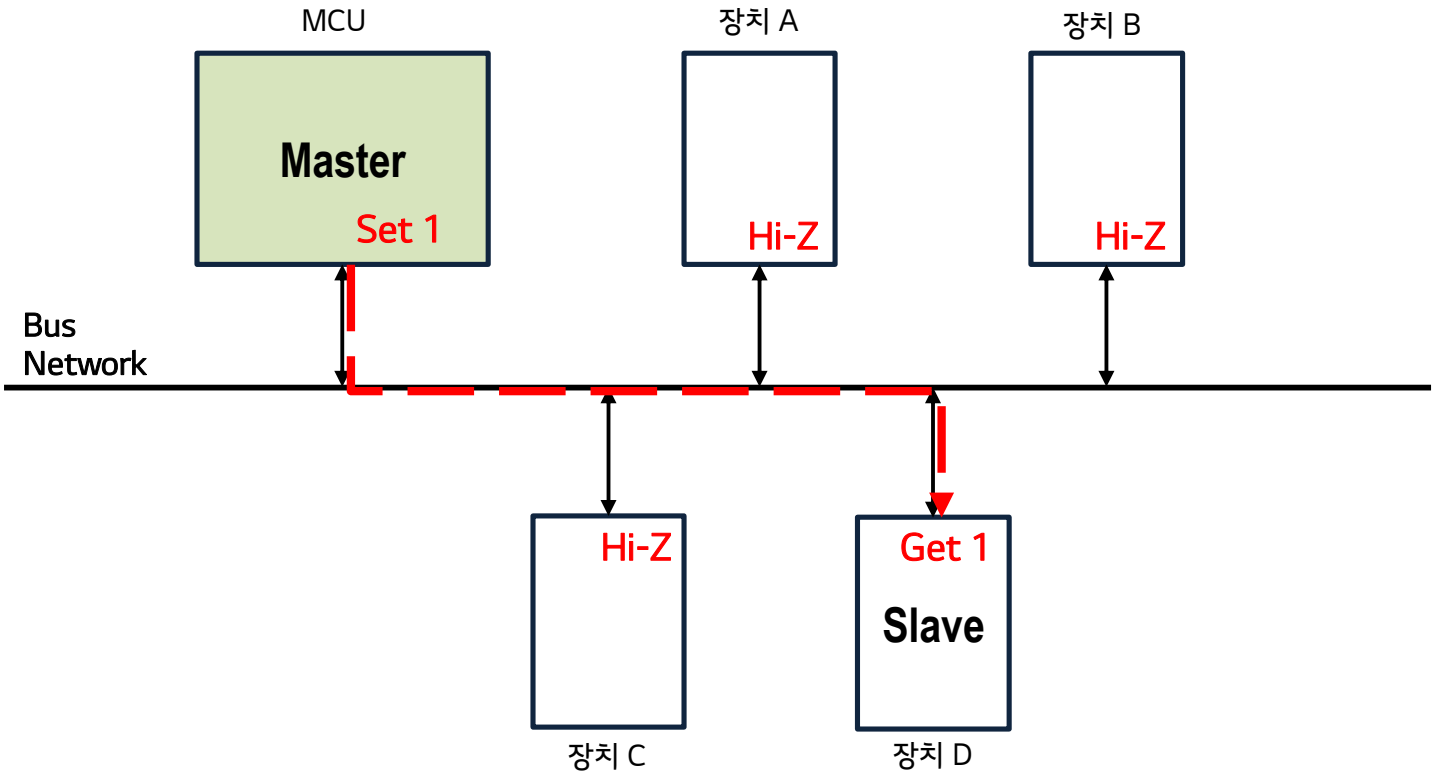



- MCU VDD = 3.3V
- 센서 칩이 사용하는 전원(V_EXT)는 5V
- Open-drain 출력회로와 외부 풀업 저항으로 Level shifter 구성

# MCU 입출력회로 구성 – 하이 임피던스

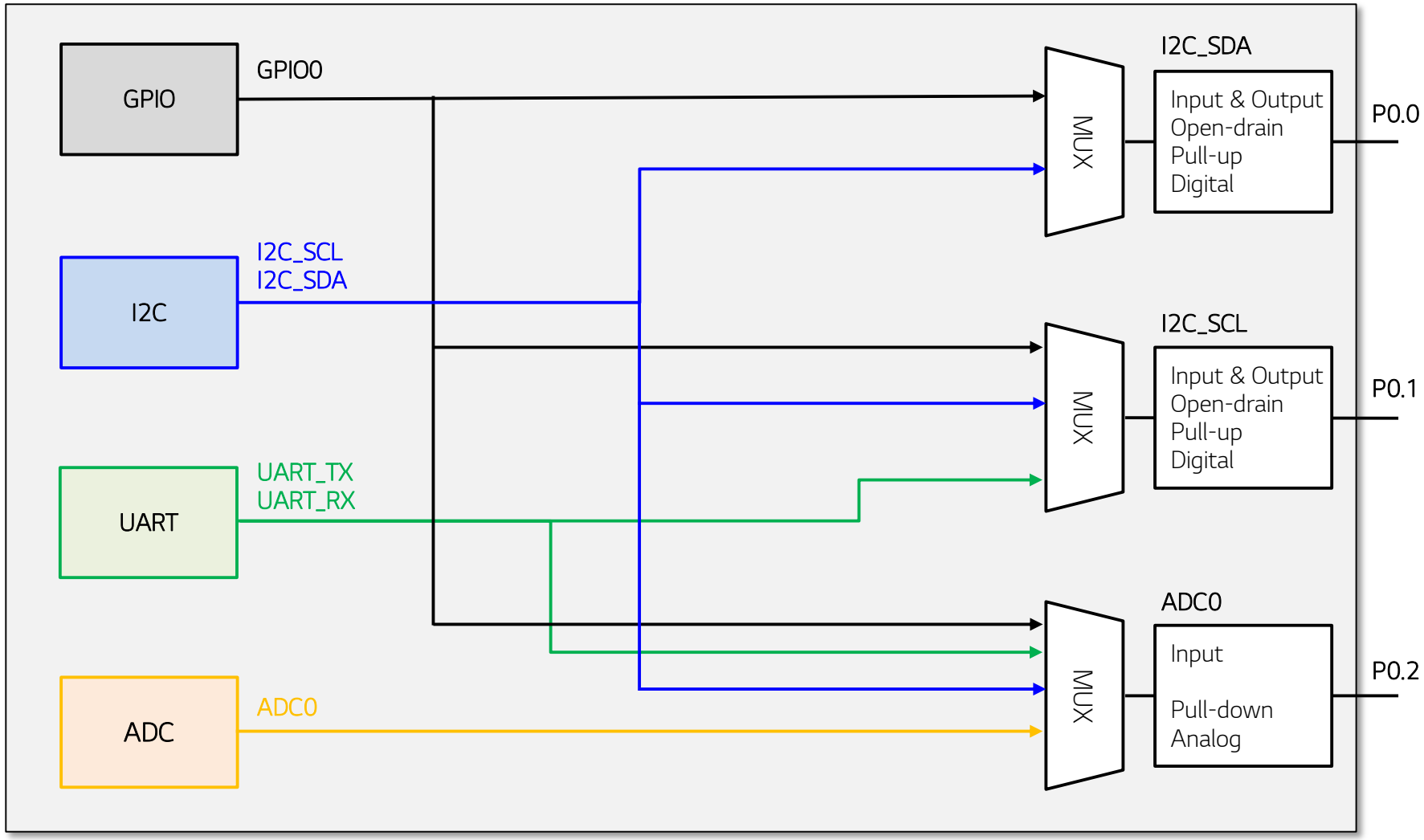
## ‖ 하이 임피던스란(Hi-Z)?

- 0, 1 논리 상태가 아닌 끊어진 상태 (tri-state)
- 말 뜻 그대로 임피던스(교류저항)가 크다는 뜻
- 버스 시스템 환경에서 Unknown state인 일부 버스들에 의한 전기적 문제가 발생하지 않도록 함



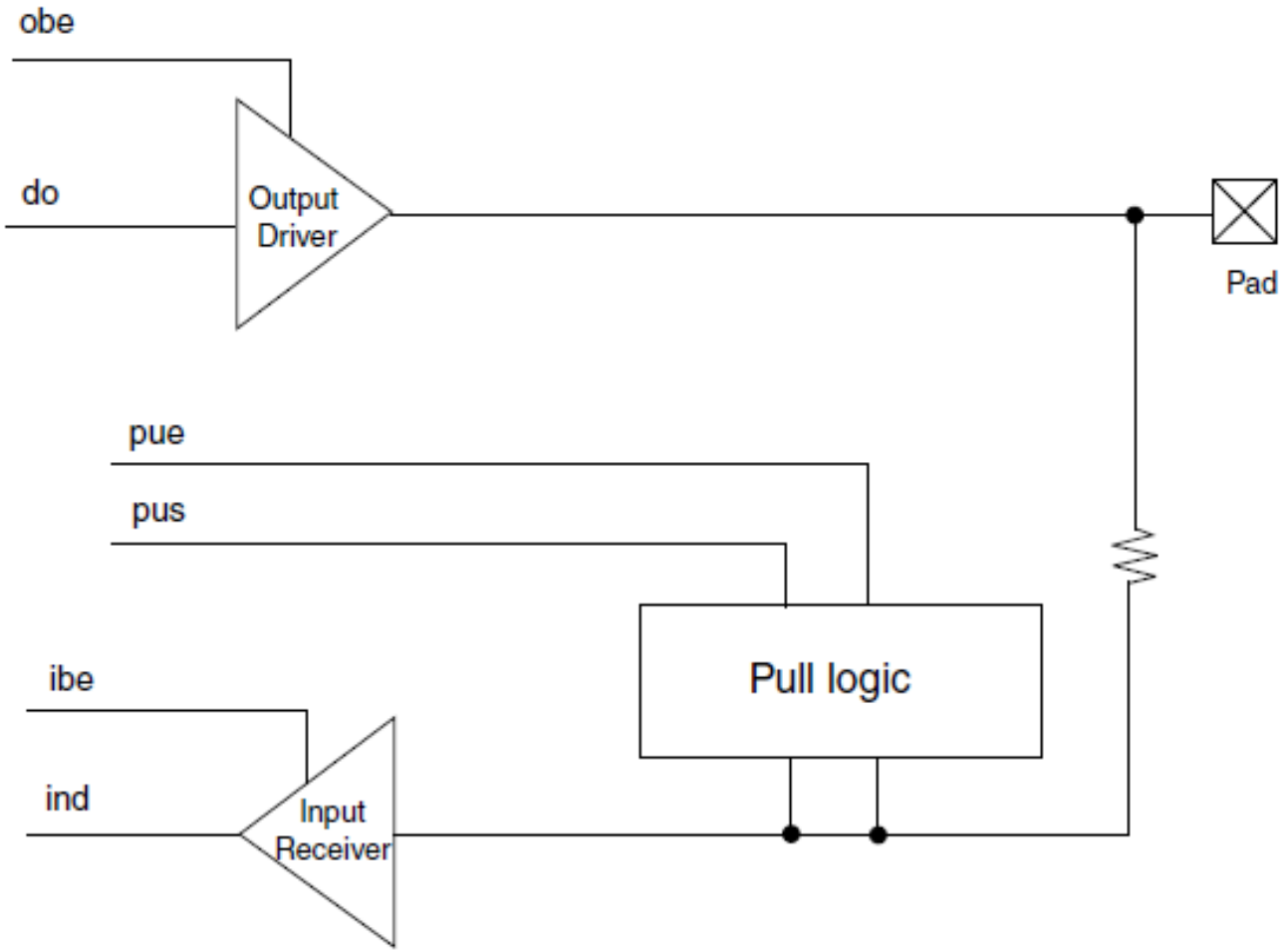
# MCU 입출력회로 구성 – Port 설정

## Pin Mux & IO Pad Configuration



# MCU 입출력회로 구성 – Port 설정

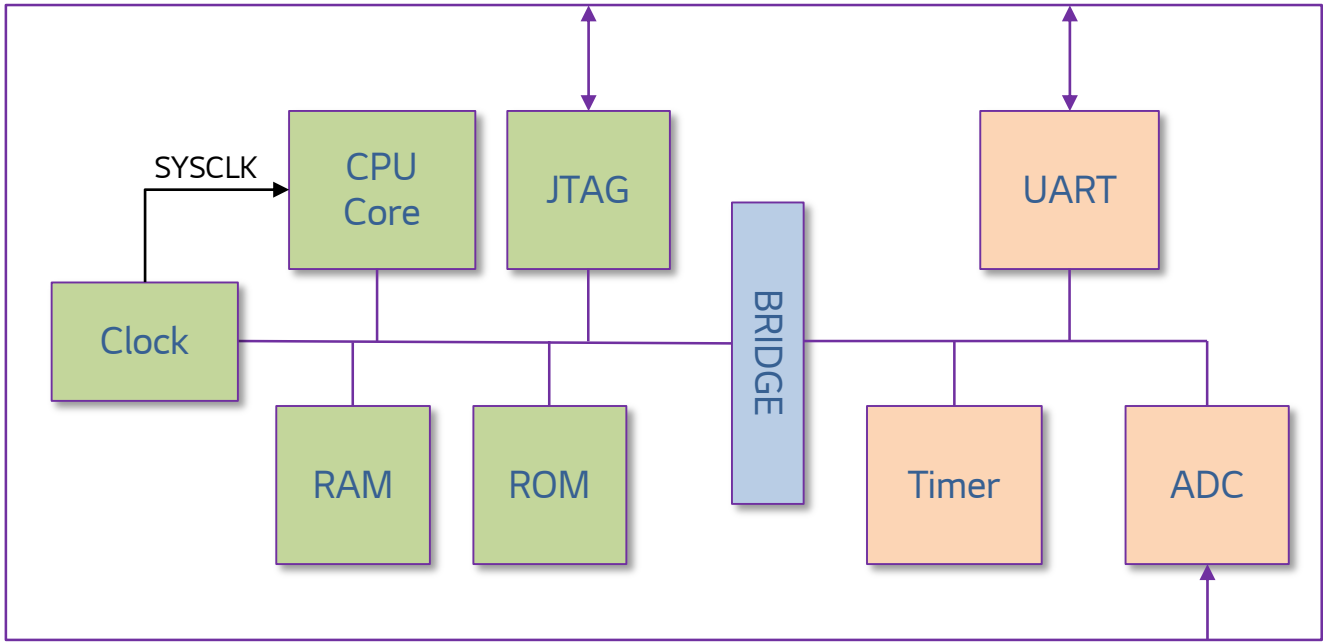
## Example of a GPIO Pad Configuration (NXP S32K)




*obe: Enable output driver
*pue: Enable/Disable Internal pullup or pulldown resistor
*ibe: Enable input receiver

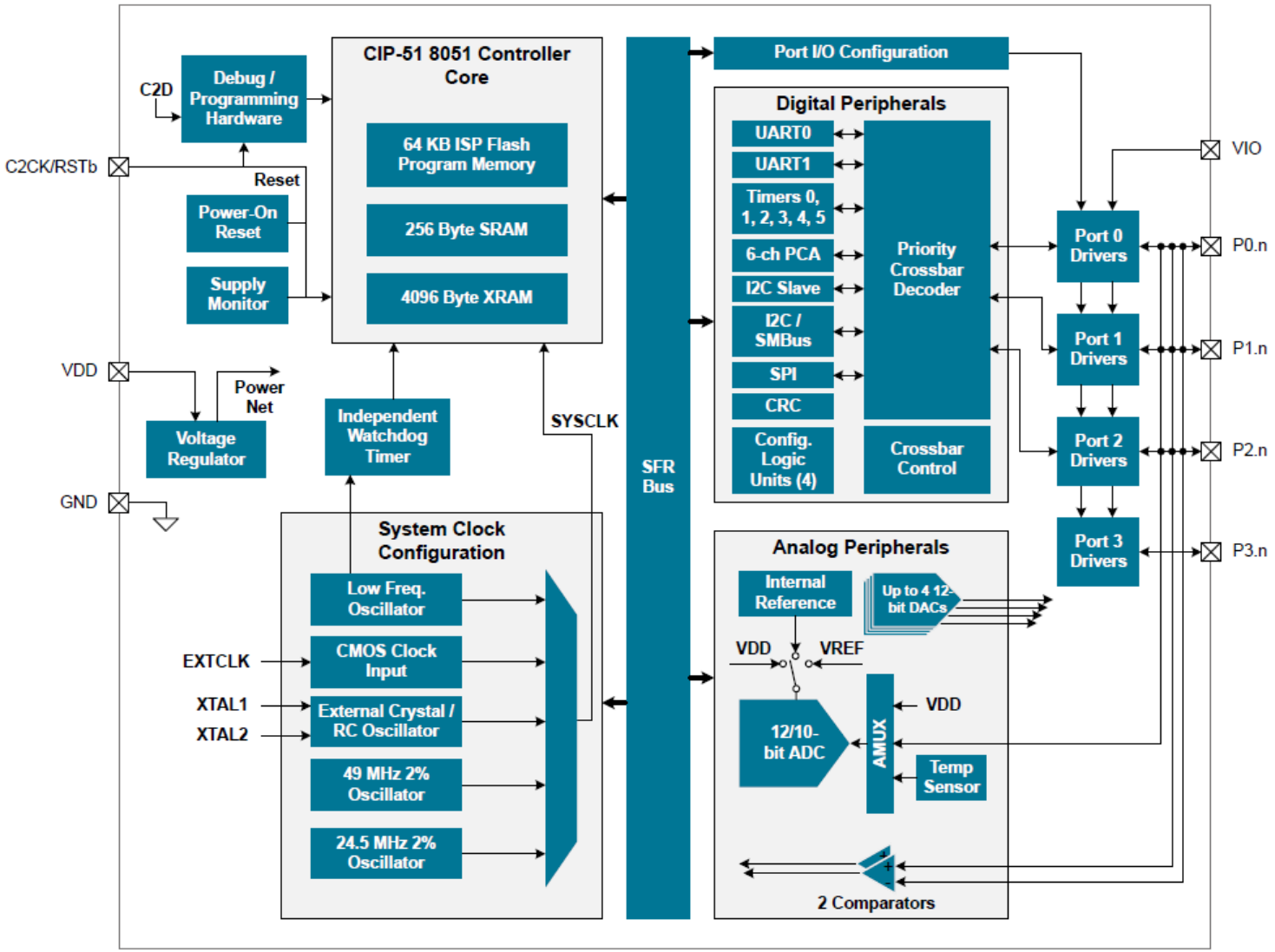
*do: Data coming from the core into the pad
*pus: Enable pullup or pulldown if pue is set
*ind: Data coming out of the pad into the core

# MCU 구조

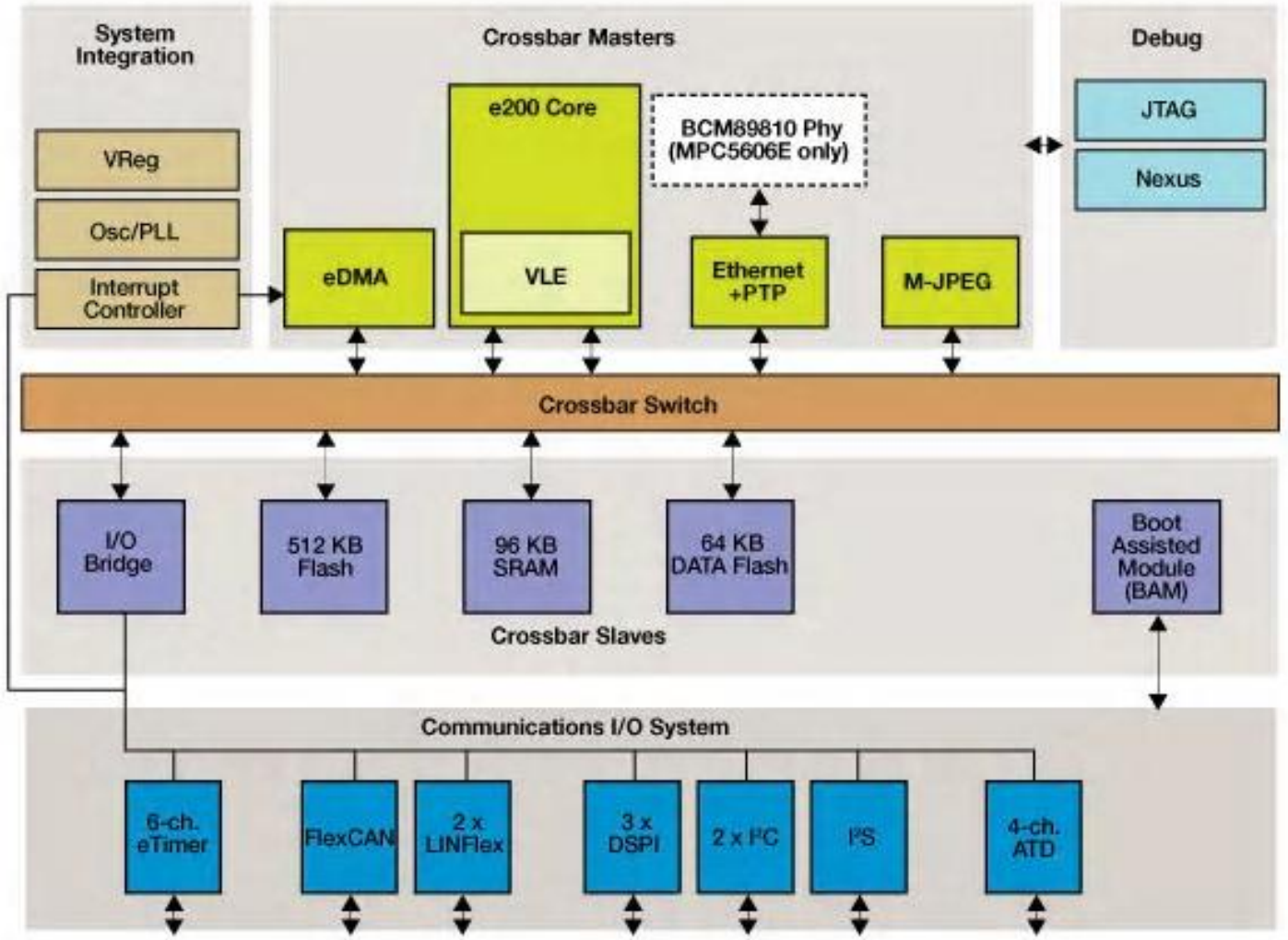



* JTAG(Joint Test Action Group) : Embedded System 개발 시 사용되는 Debugging Interface 표준
* UART(Universal Asynchronous Receiver/Transmitter) : 범용 비동기화 송수신기, 병렬 데이터를 직렬 방식으로 전환하여 데이터를 전송
* ADC(Analog to Digital Converter): 연속적인 아날로그 신호를 샘플링 및 양자화를 통해 디지털 신호로 변환하는 모듈

# MCU 구조 – Silicon Labs EFM8BB3



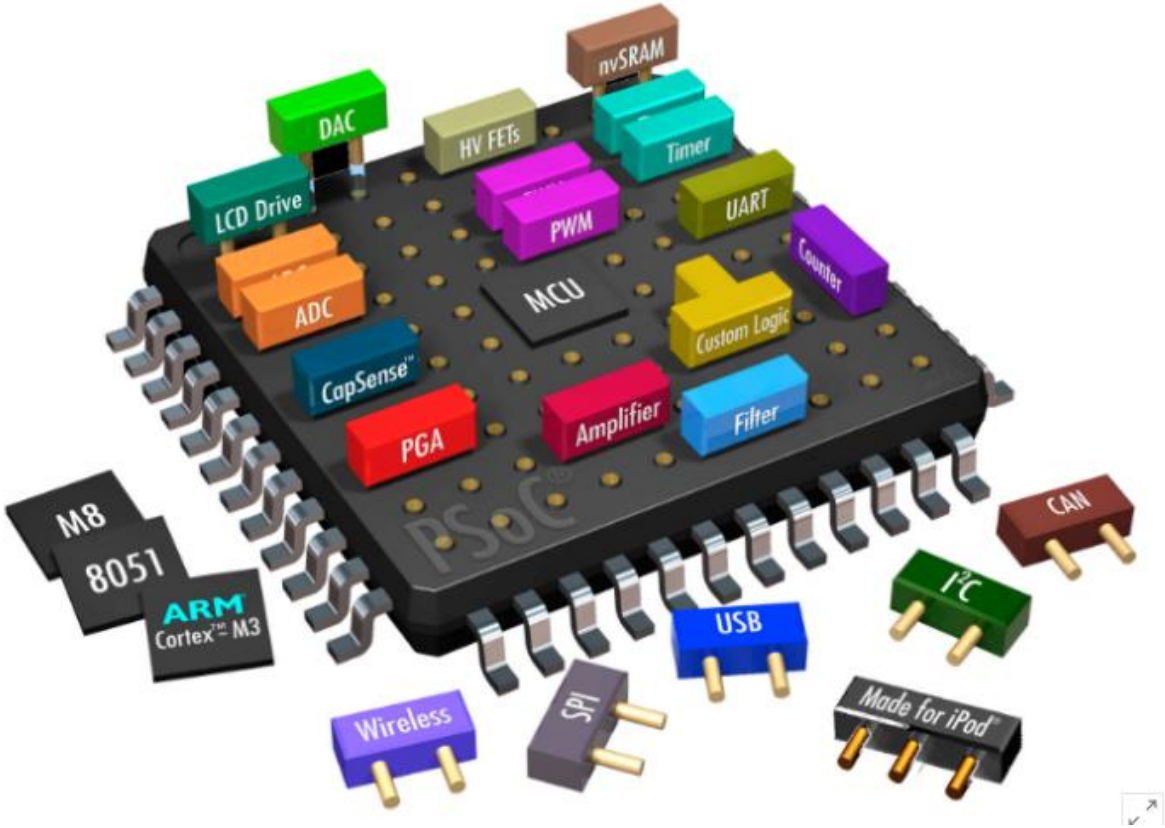
# MCU 구조 – NXP MPC560xE



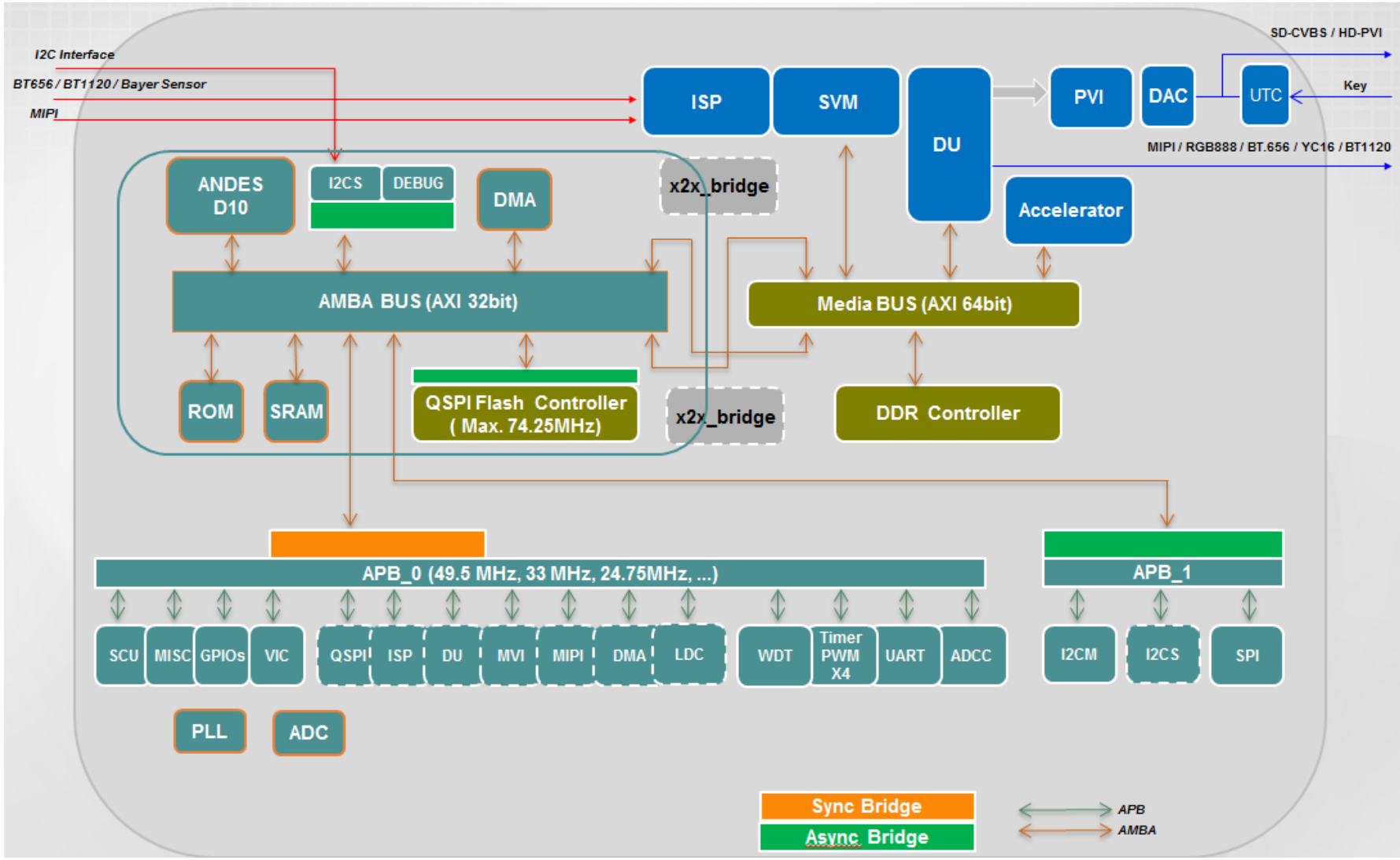
# SOC

## SOC 란?

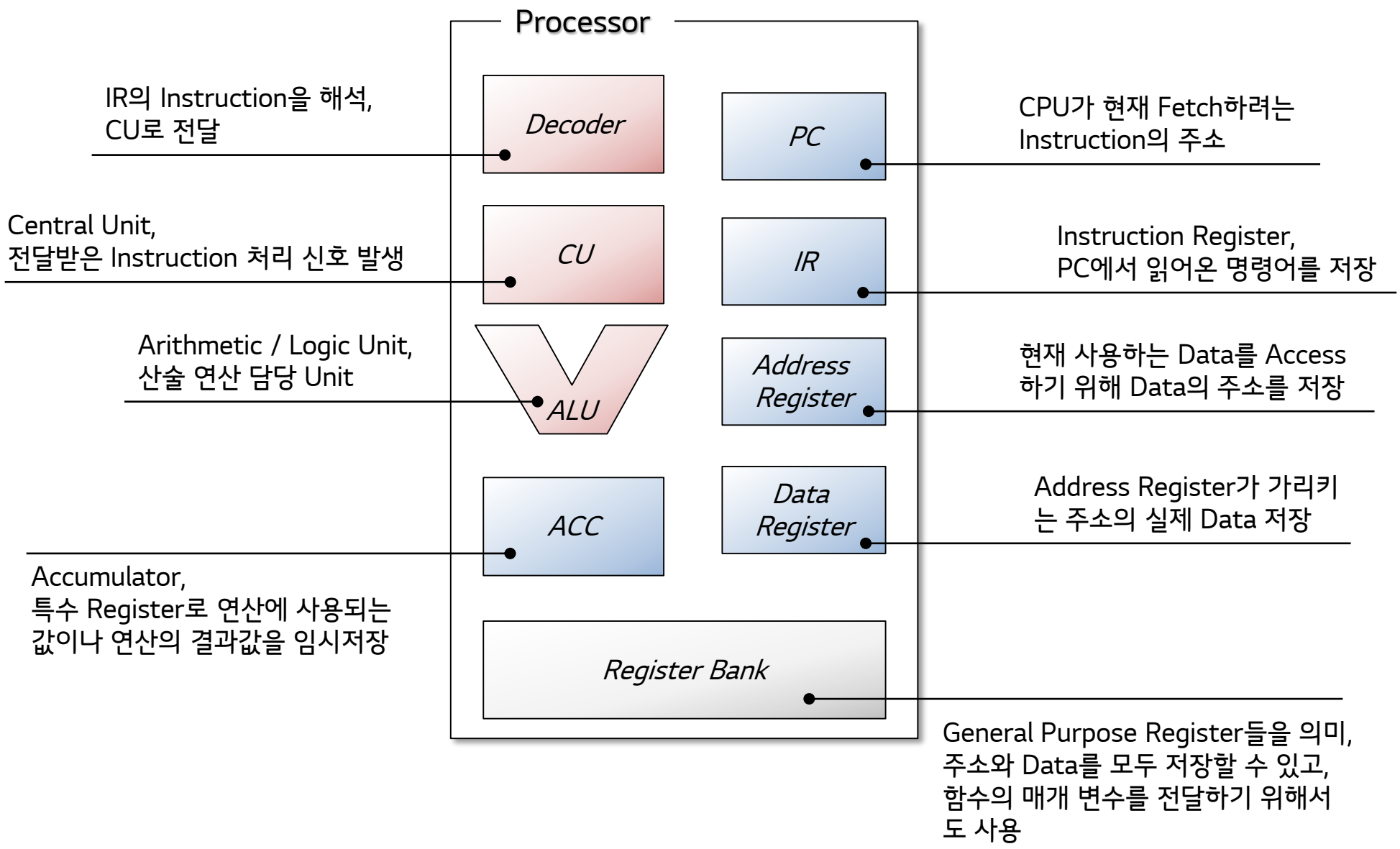
- System On Chip의 약자로, SOC 혹은 SoC로 표현
- 단일 칩 시스템으로, 여러 기능을 가진 기기들로 구성된 시스템을 하나의 칩으로 만드는 기술
- 여러 기능을 가진 반도체가 하나의 칩으로 통합, 제품 소형화가 가능



# SOC 구조 – Pixel Plus PI5008K

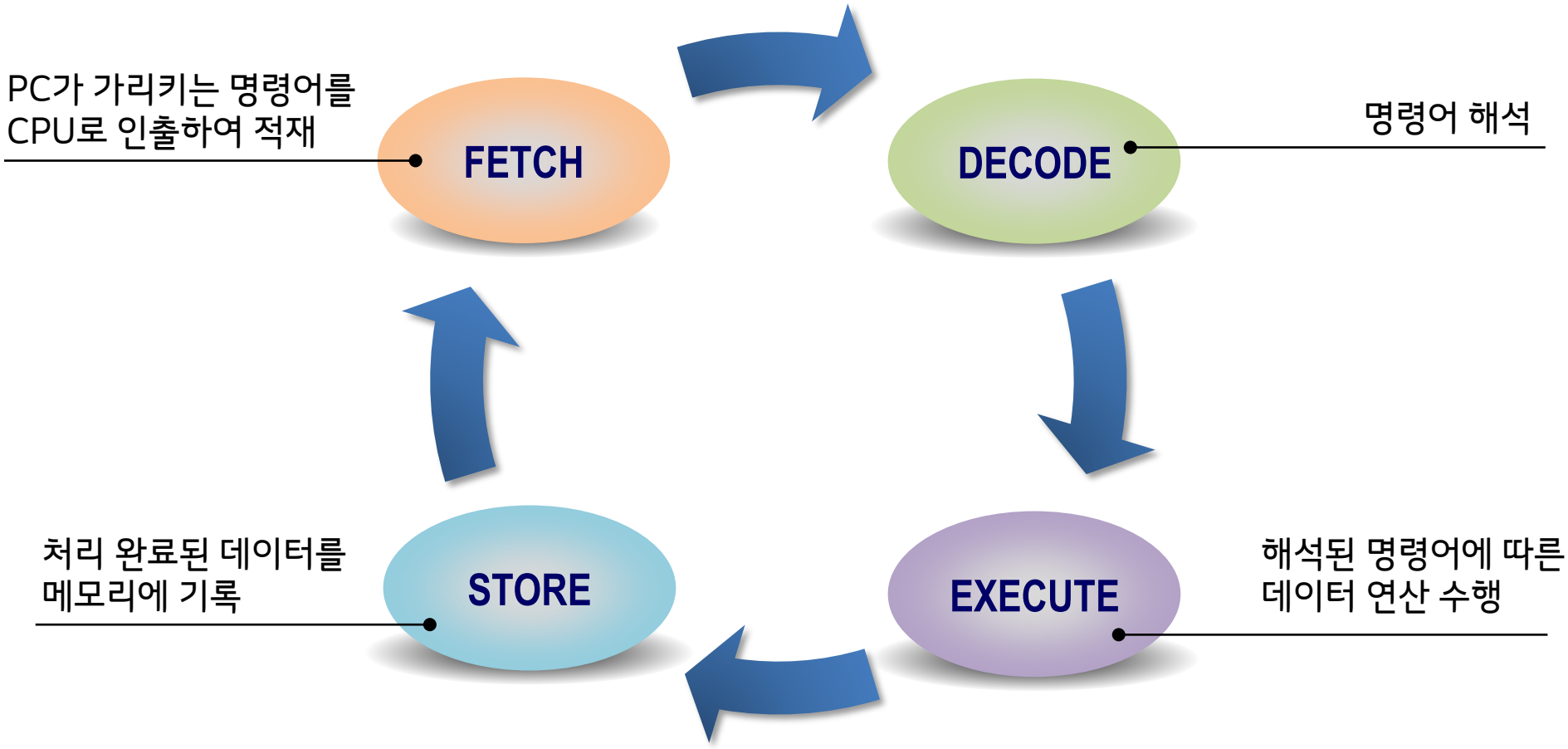


# CPU 구조



# CPU 동작 원리

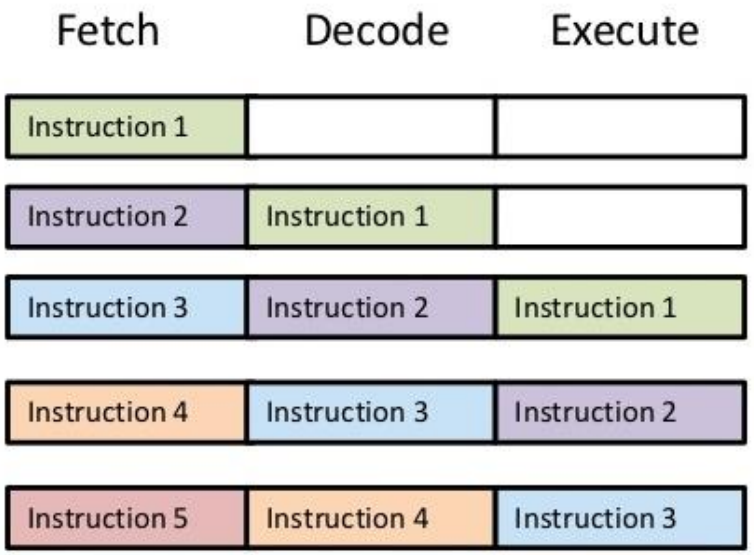
## CPU Operation



# CPU 동작 원리 – Pipeline

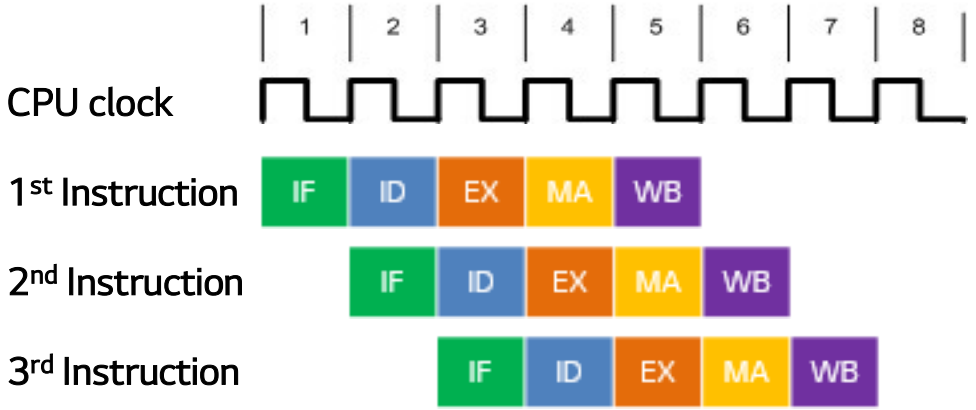
## Pipeline 기법

- CPU또는 프로세서가 이전 명령어의 수행 완료 전 다음 명령어를 수행하는 기법
- 여러 개의 작업들이 동시 처리, CPU 속도 향상 가능
- Instruction 크기가 일정할수록 Pipeline 효과 극대화




## Pipeline Stage

- Instruction Fetch (IF)
- Instruction Decode (ID)
- Execute (EX)
- Memory Access (MA)
- Writeback (WB)



# CPU 동작 원리 – 명령어

## 명령어(Instruction)의 구성

- 명령어는 크게 두 부분, 연산자(OP Code)부분과 피연산자 (Operand)부분으로 구성됨

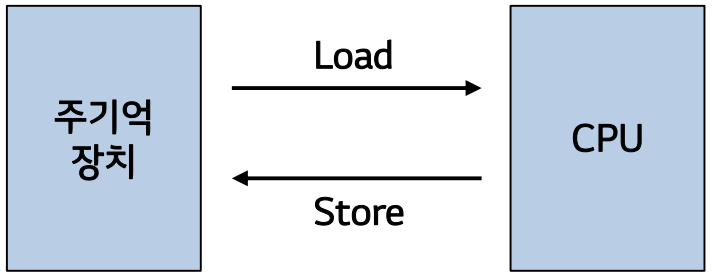

- 연산자(OP Code)
  : 명령어에서 연산 동작을 지정하는 부분으로 명령어의 종류를 표현

- 피연산자(Operand)
  : 연산의 대상이 되는 데이터의 위치를 나타내는 부분 (주소)

## 연산자(OP Code)의 기능

- 함수연산 기능
  : 산술 연산( + , - , ×, ÷) 및 논리 연산(AND, OR, NOT)을 수행

- 자료전달 기능
  : 주기억장치와 CPU 간의 자료 이동을 수행하는 기능 (Load, Store)




- 제어 기능
  : 프로그램의 실행순서를 변경하는 기능 (IF, GOTO)

- 입/출력 기능
  : Input, Output 등의 연산자들로 주변장치와의 입출력 작업을 수행

# CPU 동작 원리 – 명령어

## 명령어(Instruction)의 형식 – 8051 Instruction Set

| Hex Code | Bytes | Mnemonic | Operands |
|---|---|---|---|
| 22 | 1 | RET | |
| 20 | 3 | JB | bit, offset |
| 02 | 3 | LJMP | addr16 |
| 04 | 1 | INC | A |

**RET**

- The RET instruction pops the high-order and low-order bytes of the PC from the stack (and decrements the stack pointer by 2). Program execution resumes from the resulting address which is typically the instruction following an ACALL or LCALL instruction. No flags are affected by this instruction.

**Operation**

```
RET
PC15-8 = (SP)
SP = SP - 1
PC7-0 = (SP)
SP = SP - 1
```

# CPU 동작 원리 – 명령어

## ‖ 명령어(Instruction)의 형식 – 8051 Instruction Set

### JB

- The JB instruction branches to the address specified in the second operand if the value of the bit specified in the first operand is 1. The bit that is tested is not modified. No flags are affected by this instruction.

**Operation**

```
JB
PC = PC + 3
IF (bit) = 1
  PC = PC + offset
```

### LJMP

- The LJMP instruction transfers program execution to the specified 16-bit address. The PC is loaded with the high-order and low-order bytes of the address from the second and third bytes of this instruction respectively. No flags are affected by this instruction.

**Operation**

```
LJMP
PC = addr16
```

### INC

- The INC instruction increments the specified operand by 1. An original value of 0FFh or 0FFFFh overflows to 00h or 0000h. No flags are affected by this instruction.

**Operation**

```
INC
A = A + 1
```

# CPU 동작 원리

## CPU의 소스코드 실행

```
static U8 val1 = 1;
static U8 val2 = 2;
static U8 val3;

void add(void)
{
    U8 tmp;

    tmp = val1;
    val3 = tmp + val2;

}
```

| | Address | Data |
|---|---|---|
| val1 | 0x59B | 1 |
| val2 | 0x59C | 2 |
| val3 | 0x59D | 0 |
| | ... | ... |

Data 영역

| | Address | Instruction Set |
|---|---|---|
| | 0x20D2 | MOV R7 #59BH |
| tmp = val1; | 0x20D6 | MOV R5 #59CH |
| | 0x20DA | MOV A R7 |
| | 0x20DD | ADD A R5 |
| val3 = tmp + val2; | 0x20E0 | MOVX #59DH, A |
| | ... | ... |

Text(Code) 영역

# CPU 동작 원리

## CPU의 소스코드 실행

```
10              tmp = val1;
              add:
000020d2:       MOV     DPTR, #59BH
000020d5:       MOVX    A, @DPTR
000020d6:       MOV     R7, A
11              val3 = tmp + val2;
000020d7:       INC     DPTR
000020d8:       MOVX    A, @DPTR
000020d9:       MOV     R5, A
000020da:       MOV     A, R7
000020db:       ADD     A, R5
000020dc:       MOV     R6, A
000020dd:       CLR     A
000020de:       RLC     A
000020df:       INC     DPTR
000020e0:       MOVX    @DPTR, A
000020e1:       INC     DPTR
000020e2:       XCH     A, R6
000020e3:       MOVX    @DPTR, A
12            }
000020e4:       RET
```

(x)= Variables | Breakpoints | Registers | Expressions

| Name | Type | Value | Location |
|---|---|---|---|
| (x)= tmp | unsigned char | 0x0 (Hex) | RAM:0x7 |

```
---          MODULE    ---       ---       TEST
010020D2H    PUBLIC    CODE      ---       add
0200059BH    SYMBOL    XDATA     BYTE      val1
0200059CH    SYMBOL    XDATA     BYTE      val2
0200059DH    SYMBOL    XDATA     WORD      val3
```

CODE:0x20d2 : 0x20D2 <Traditional> | New Renderings...

```
0x000020D2   90059BE0 FFA3E0FD EF2DFEE4 33A3F0A3 CEF0227D
```

RET

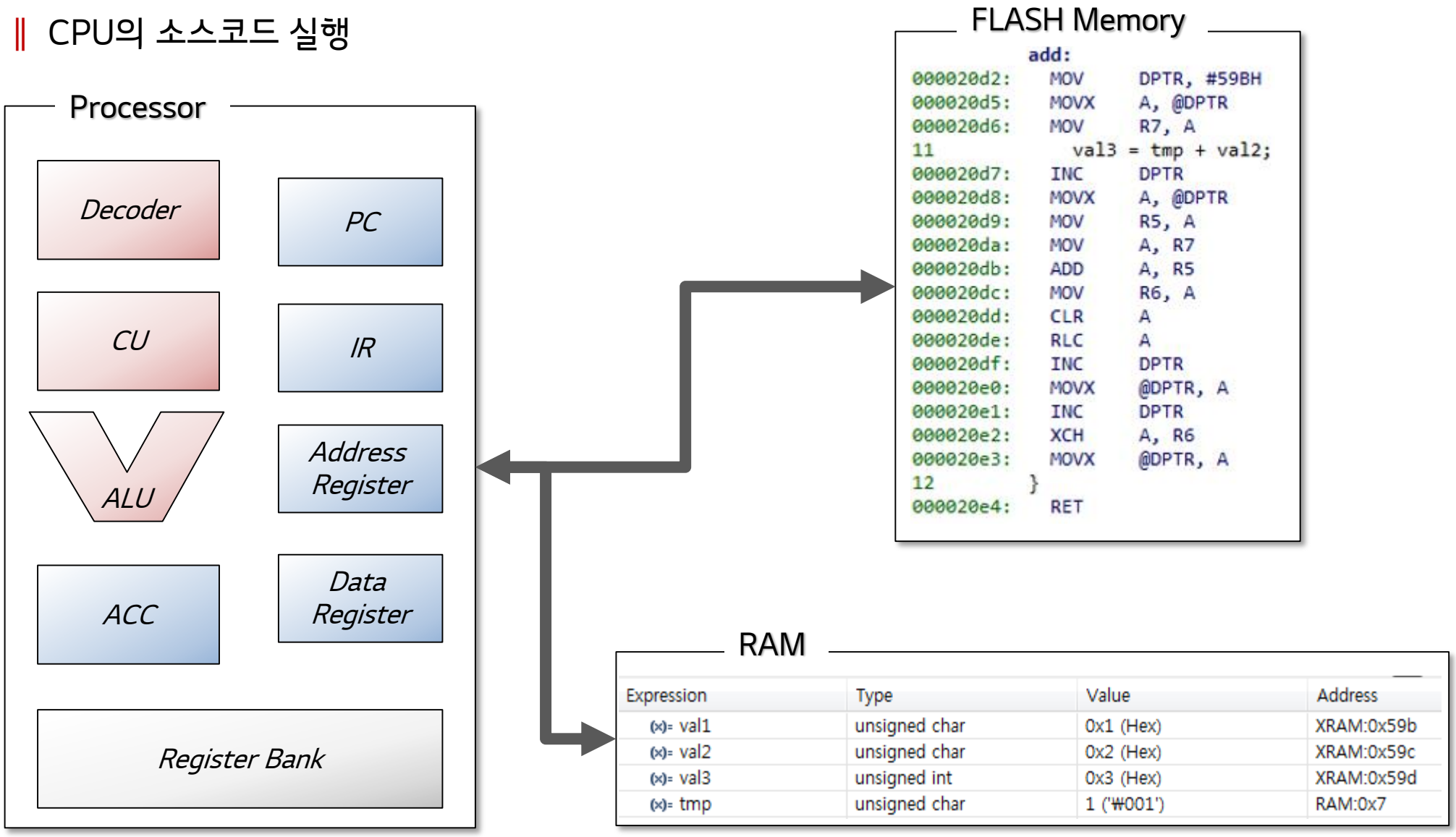
XRAM:0x59B : 0x59B <Tra

```
0x0000059B   01020003
```

val1 val2 val3

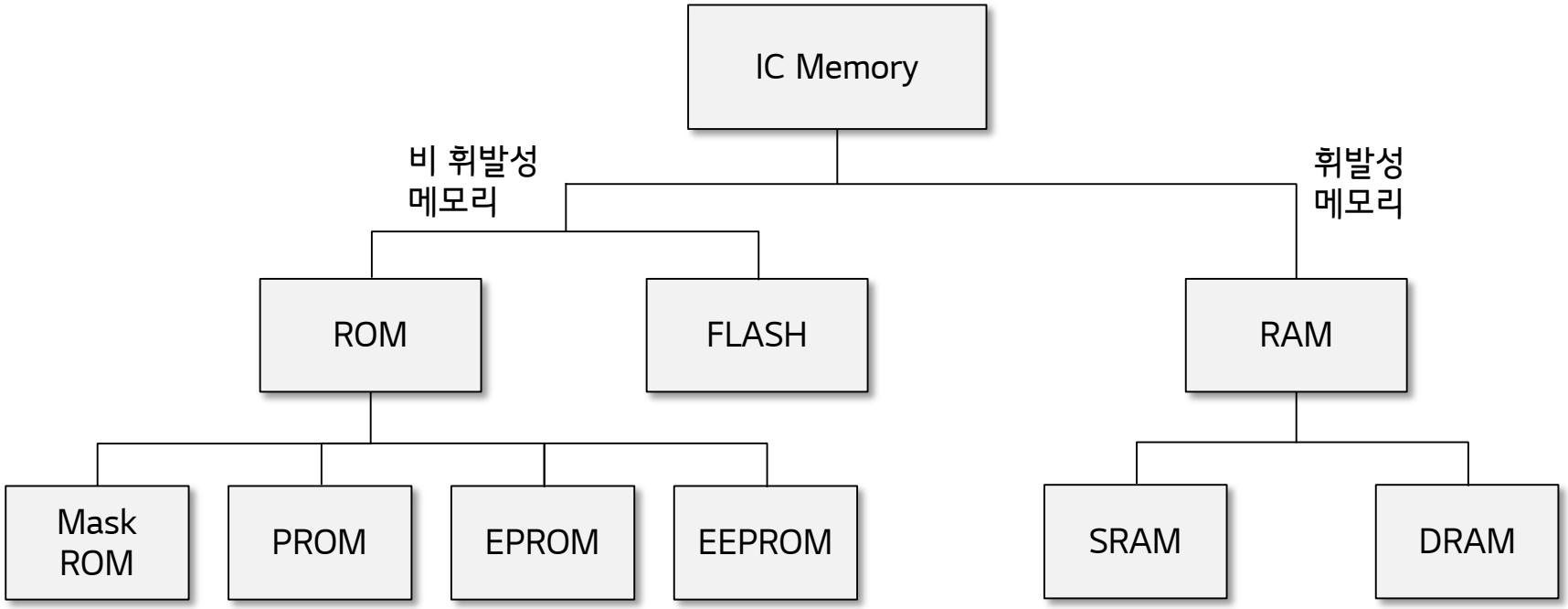
# CPU 동작 원리

## CPU의 소스코드 실행



# M3 Memory

# Memory 종류




## ROM (Read-Only Memory)

- Read만 가능한 메모리, 임의로 새로운 데이터 입력 불가
- 전원 공급이 중단되어도 데이터 영구 보관 (non-volatile)

- Mask ROM: 제조 공정에서 Mask로 데이터를 미리 만들어서 ROM에 기억시킴. 데이터 수정 불가.
- PROM (Programmable ROM): ROM Writer 등으로 내용을 한 번 저장할 수 있음. 단, 저장한 후에는 재사용이 불가능함.
- EPROM (Erasable PROM): 자외선 신호로 데이터 삭제 가능, 새로운 프로그램으로 업데이트 가능
- EEPROM (Electrically EPROM): 전기적 신호로 데이터 삭제 가능, 동작 중 쓰기 가능.

# Memory 종류

## RAM (Random Access Memory)

- 전원이 끊어지면 데이터 소멸 (volatile)
- 데이터 읽는 속도와 기록하는 속도가 같음
- CPU가 작동 중에 일시적으로 데이터 Read/Write, 응용 프로그램 로딩 등과 같은 곳에 사용됨

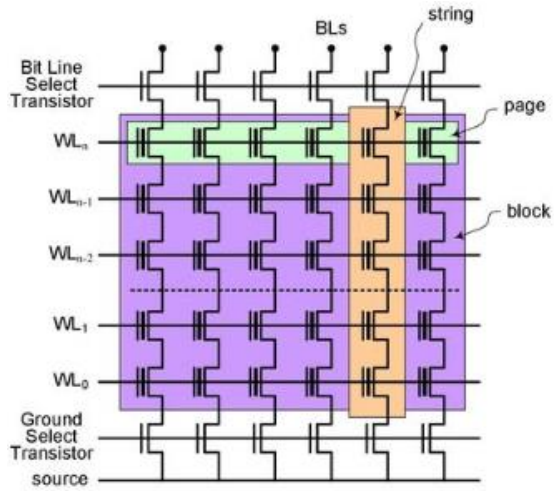
| SRAM (Static RAM) | DRAM (Dynamic RAM) |
|---|---|
| 전원에 공급되는 전압이 일정 | 전원에 공급되는 전압이 일정하지 않음 |
| 전원 공급 동안에는 데이터 유지 | 일정 시간 후 데이터 사라짐, Reflesh 필요<br>(저장공간마다 캐패시터를 사용하기 때문) |
| DRAM에 비해 용량은 적으나<br>속도가 빠름 | SRAM에 비해 IC집적도가 높아 용량은 크지만<br>속도가 느림 |
| 비교적 가격이 비쌈 | 비교적 가격이 저렴 |
| CPU cache로 사용 | Main memory로 사용 |

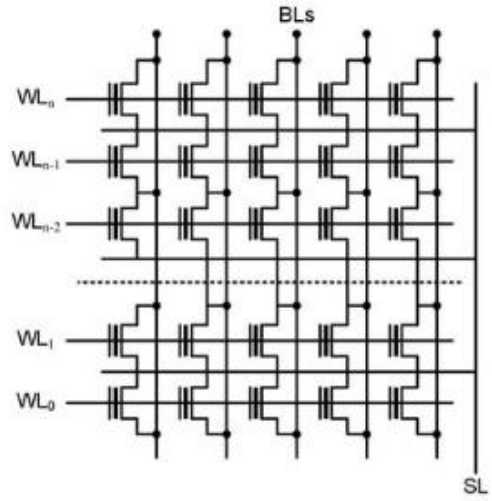
# Memory 종류

## FLASH

- RAM처럼 Read / Write가 쉬움
- ROM처럼 전원이 없어도 데이터가 사라지지 않음 (non-volatile)

| NAND Flash | NOR Flash |
|---|---|
| 직렬 회로로 연결 (데이터 접근 속도 느림) | 병렬 회로로 연결 (데이터 접근 속도 빠름) |
| write 성능 우수 / read 성능 느림 | read 성능 우수 / write 성능 느림 |
| 대용량 데이터 저장이 가능 | 대용량 데이터 저장 한계 |
| XIP 방식에 적합하지 않음 | XIP 방식에 적합<br>(RAM처럼 byte단위로<br>Random Access 가능) |

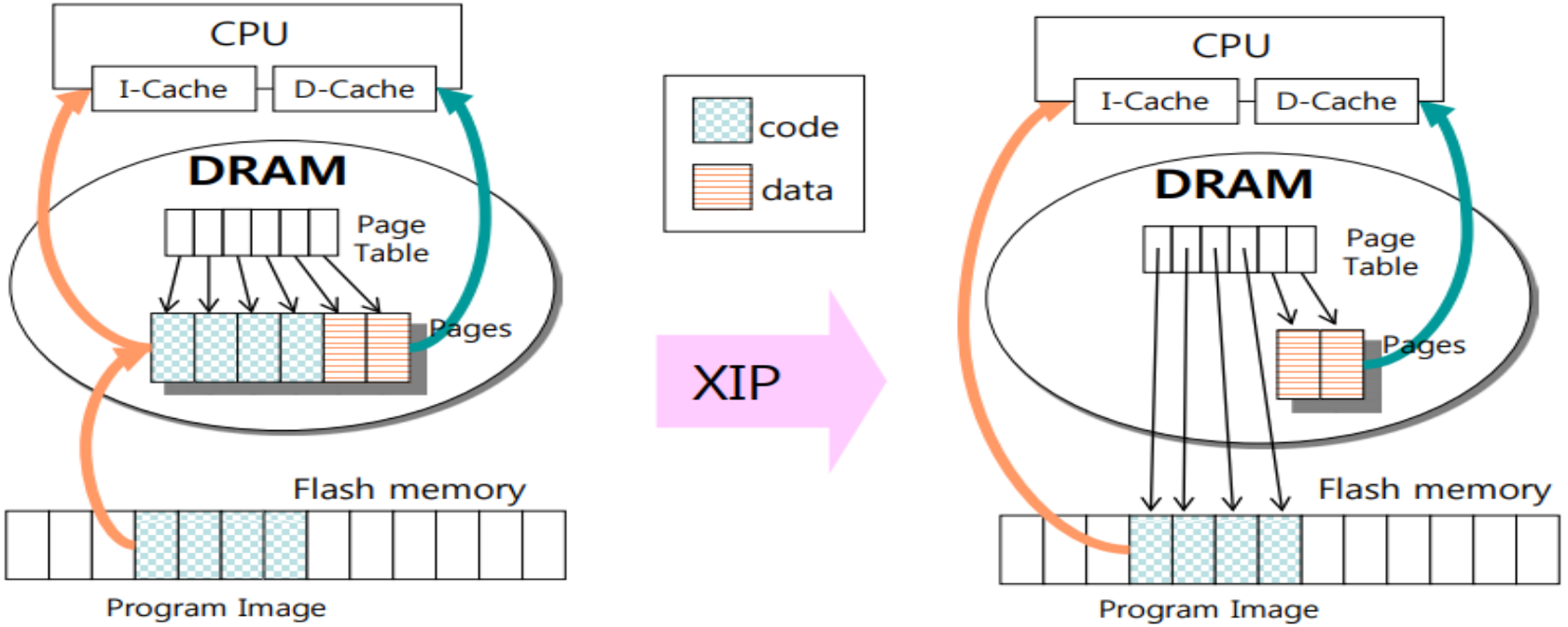


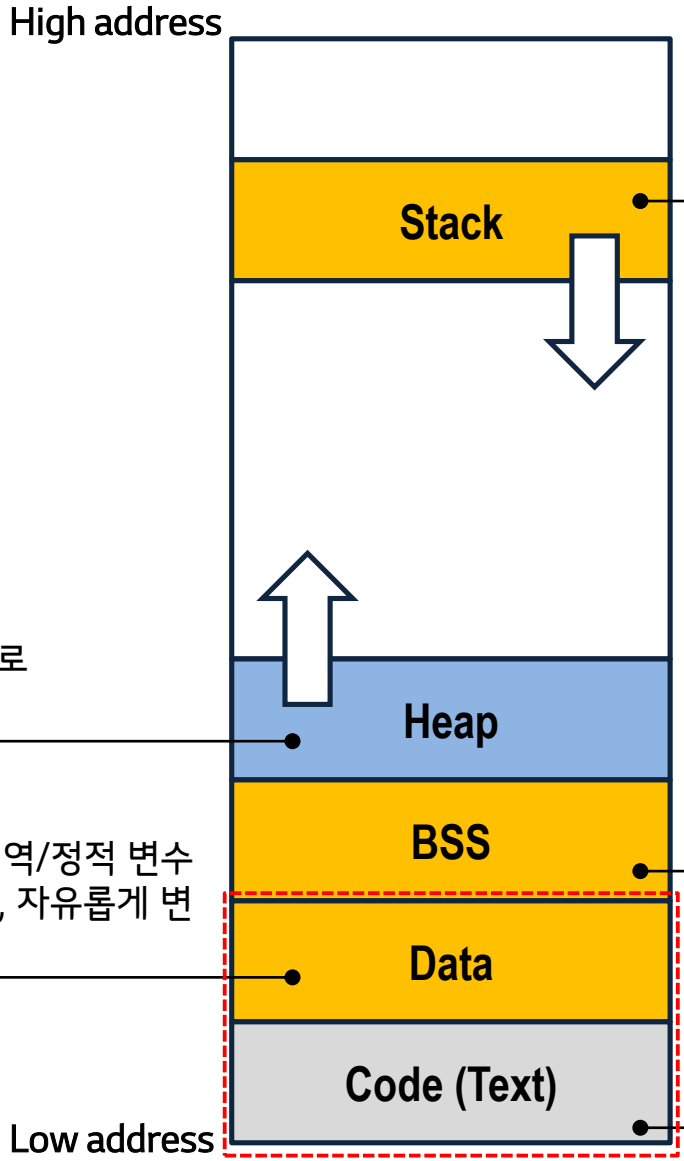
# Memory

## XIP (eXecute-In-Place)

- 프로그램을 RAM에 Load하지 않고 FLASH Memory상에서 직접 Program/Code 실행
- Byte단위의 Access가 가능 (Random Access)
- CPU가 메모리 주소를 할당하고 직접 접근하므로 메모리 절약 가능
- 프로그램을 RAM Loading할 필요 없으므로 Booting시간이 적게 소모됨



# Memory 영역별 개념


**Stack 영역**
함수 호출 시 마다 자동으로 할당하여 사용되는 영역
Local 변수, Parameter, Return 값 등 실행 중 자동으로 증가/감소하기 때문에 보통 메모리의 마지막 번지에 위치함

**Heap 영역**
실행 중 필요 시, 동적으로 할당 및 수거됨

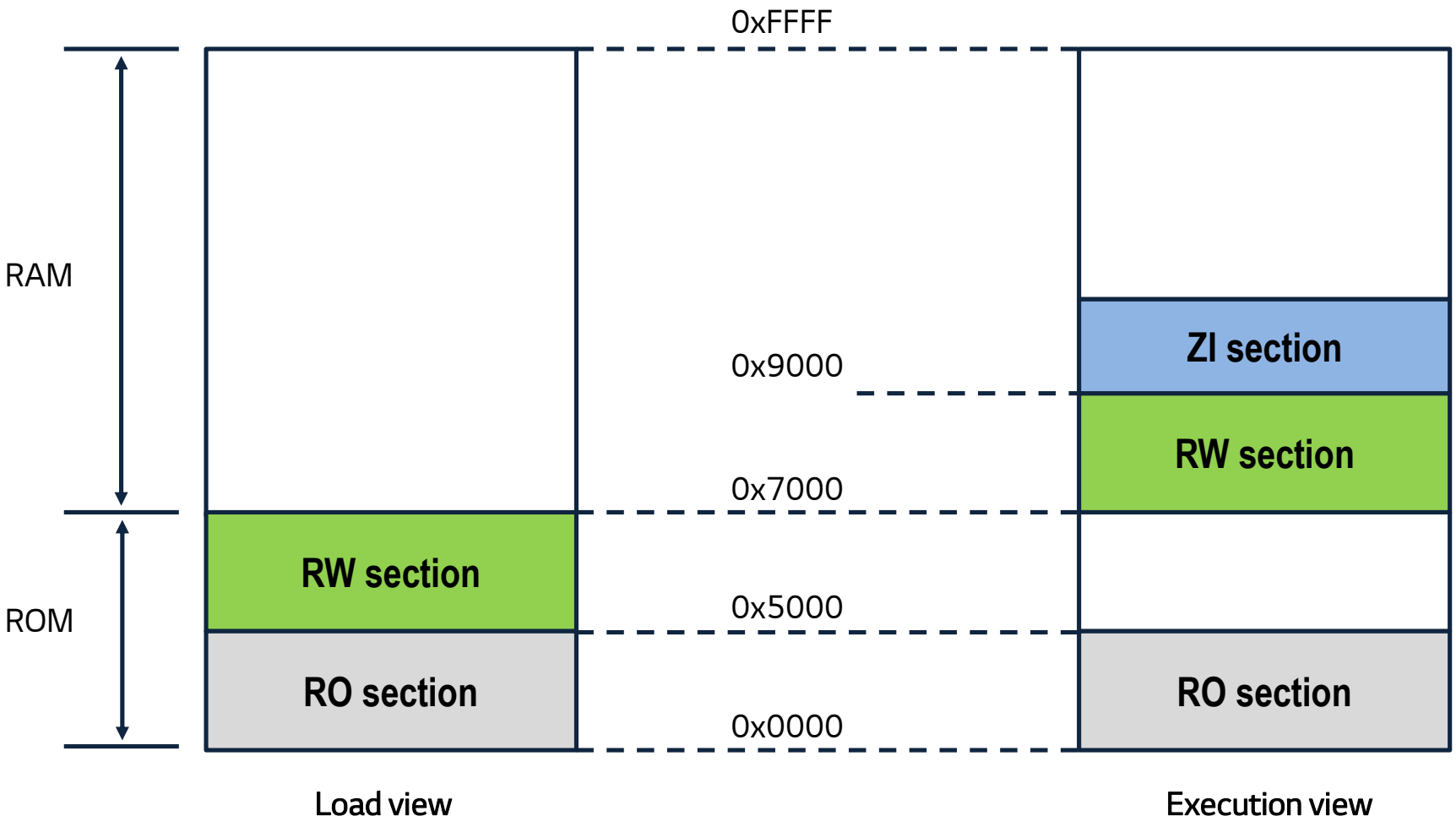
**BSS(ZI) 영역**
초기값이 0 혹은 없는 전역 변수, Memory map에 공간만 할당하며 실제 값이 없어 Firmware에 포함되지 않음
(실행 시, Scatter file을 loading하여 0으로 채워진 BSS영역을 할당)

**Data 영역**
RW영역, 특정 초기값을 가진 전역/정적 변수
실행 중에는 RAM Loading되며, 자유롭게 변경 가능

**Code (Text) 영역**
RO영역, 프로그램 실행 중 변경 불가
Constant 및 프로그램 명령어가 포함됨

# Memory 영역별 개념

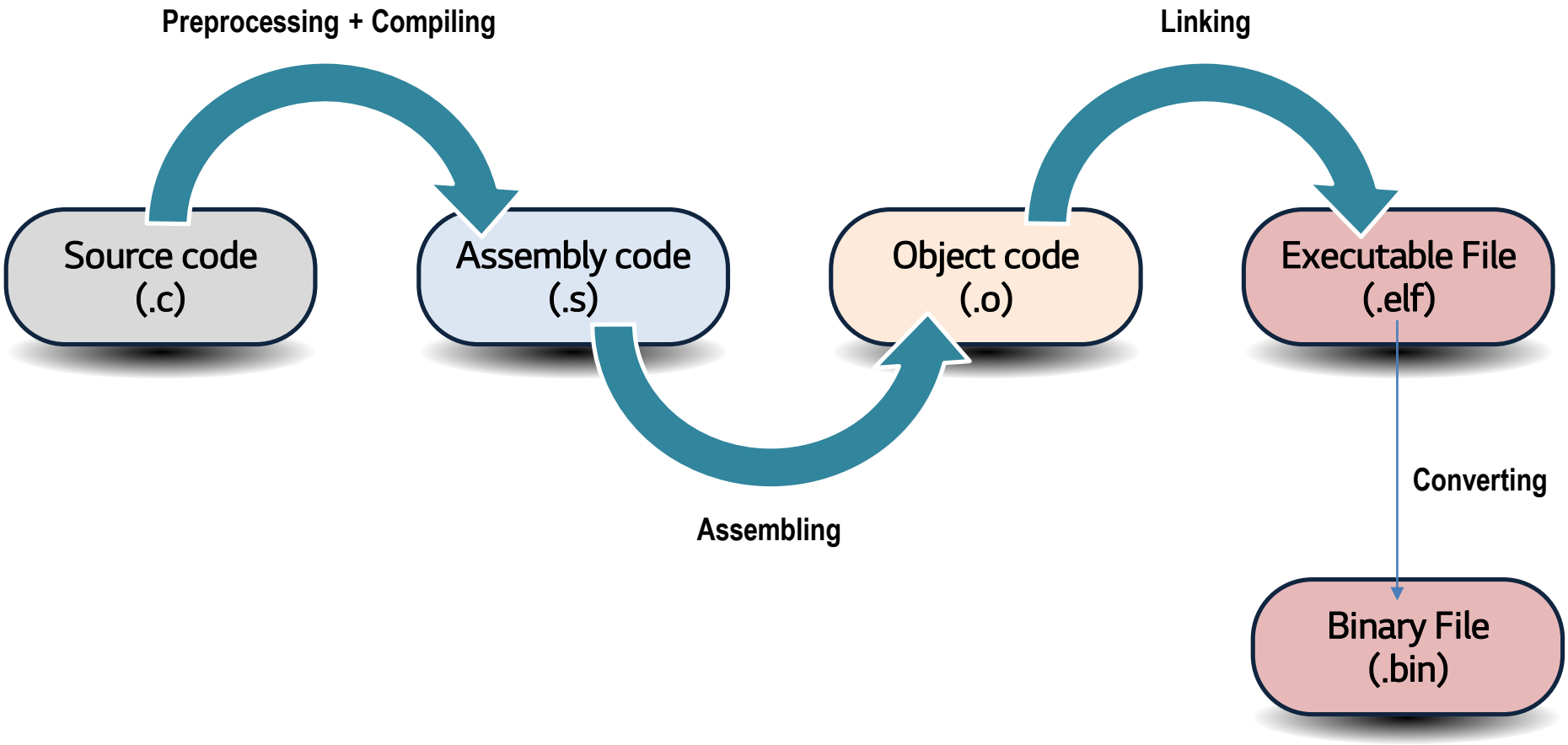
## Load view와 Execution view



# M4 Compile과 Link

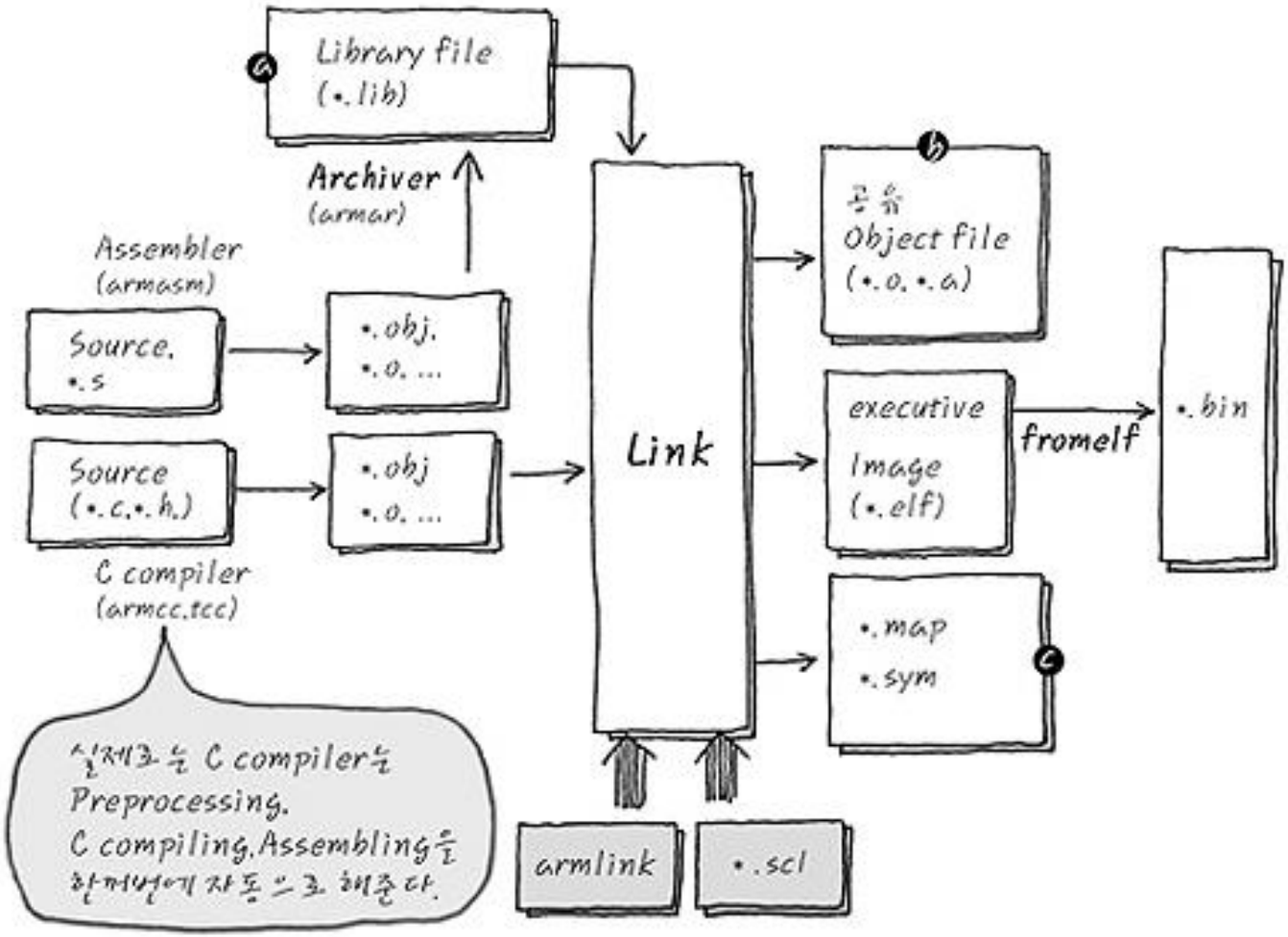
# Compile과 Link

## Compile Stages



# Compile과 Link

## Link



# Compile과 Link

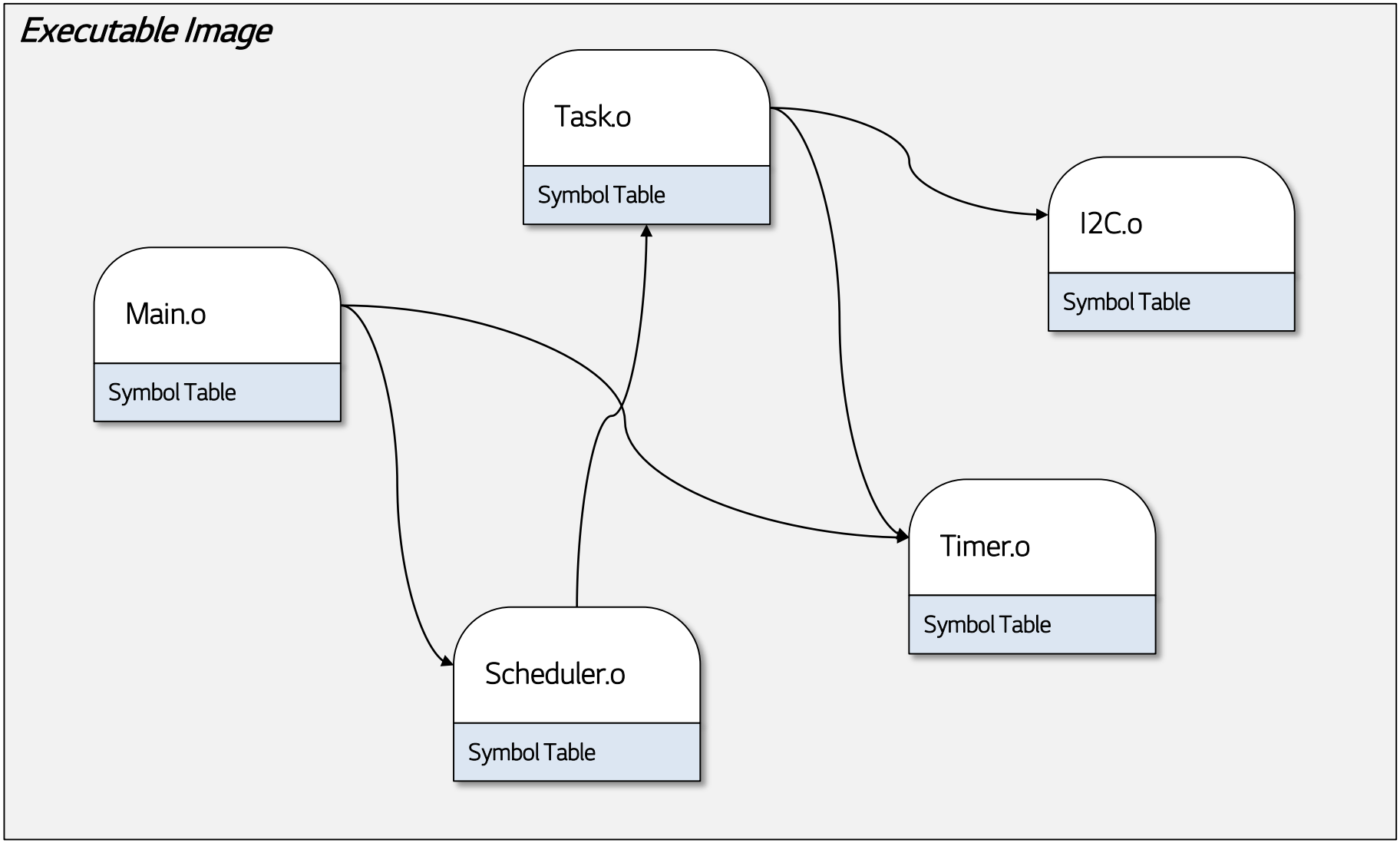
## Symbol Table이란?

```
// Declare an external function
extern double bar(double x);

// Define a public function
double foo(int count)
{
    double  sum = 0.0;

    // Sum all the values bar(1) to bar(count)
    for (int i = 1;  i <= count;  i++)
        sum += bar((double) i);
    return sum;
}
```

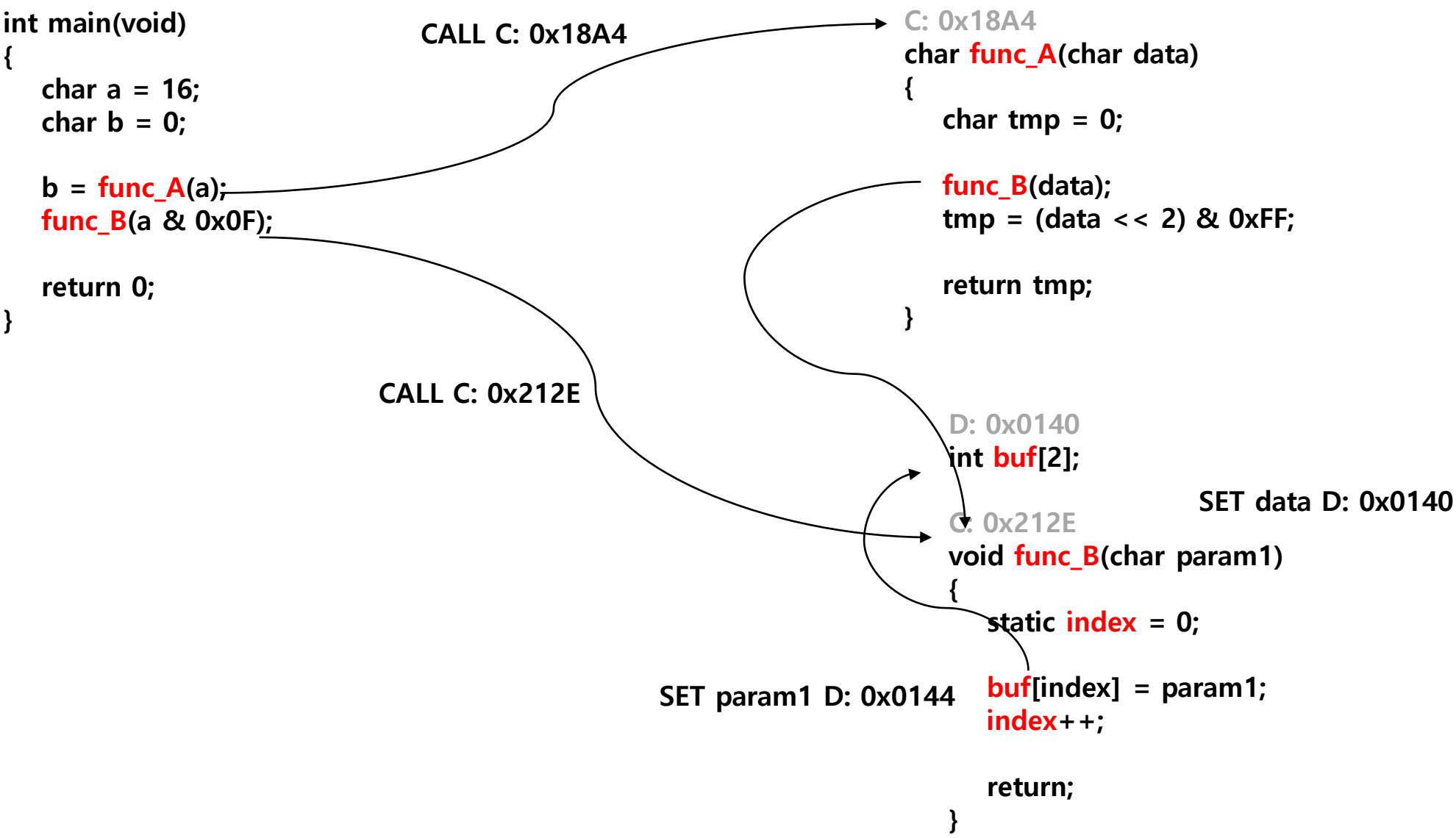
| 심볼 이름 | 타입 | 스코프 |
|---|---|---|
| bar | function, double | extern |
| x | double | function parameter |
| foo | function, double | global |
| count | int | function parameter |
| sum | double | block local |
| i | int | for-loop statement |

Symbol Table

# Compile과 Link



# Compile과 Link



# Compile과 Link

## EFM8BB3(C8051) Memory map

```
BASE         START        END          USED       MEMORY CLASS
=============================================================
C:000000H    C:002800H    C:0077FFH    002384H    CODE
C:000000H    C:002800H    C:0077FFH    000122H    CONST
X:000000H    X:000000H    X:000FFFH    000404H    XDATA
X:000000H    X:000000H    X:000FFFH               HDATA
I:000020H.0  I:000020H.0  I:00002FH.7  000001H.6  BIT
I:000000H    I:000000H    I:00007FH    00000EH    DATA
I:000000H    I:000000H    I:0000FFH    000001H    IDATA
```

```
* * * * * * * * * * *   C O D E   M E M O R Y   * * * * * * * * * * * * * *
002800H   002802H   000003H   ---    OFFS..   CODE          ?CO?SILABS_STARTUP?3
002803H   002BEEH   0003ECH   BYTE   UNIT     CODE          ?C?LIB_CODE
002BEFH   002E44H   000256H   BYTE   UNIT     CODE          ?PR?_SYSTEMMANAGER_SWAPTASK?SYSTEMMANAGER
002E45H   003069H   000225H   BYTE   UNIT     CODE          ?PR?GPIOI2C_MASTER_ISR?GPIOI2C_MASTER
00306AH   0031B8H   00014FH   BYTE   UNIT     CODE          ?PR?SMBUS0_ISR?SMBUS
0031B9H   00328AH   0000D2H   BYTE   UNIT     CODE          ?PR?_SYSTEMMANAGER_REGISTERTASK?SYSTEMMANAGER
00328BH   003355H   0000CBH   BYTE   UNIT     CODE          ?PR?SYSTEMMANAGER_UPDATETASKSTATE?SYSTEMMANAGER
003356H   00341EH   0000C9H   BYTE   UNIT     CODE          ?PR?_PACKETMANAGER_SETREGISTERATTRIBUTE?PACKETMANAGER
00341FH   0034D2H   0000B4H   BYTE   UNIT     CODE          ?PR?_REGISTERTABLE_CLASSIFYCOMMAND?REGISTERTABLE
0034D3H   003579H   0000A7H   BYTE   UNIT     CODE          ?PR?_IOBUFFER_REMOVE?IOBUFFER
00357AH   00361DH   0000A4H   BYTE   UNIT     CODE          ?PR?ISP_STREAMINGSTOPVIDEO?ISP
00361EH   0036BBH   00009EH   BYTE   UNIT     CODE          ?PR?_SMBUS_MASTERREAD?SMBUS
0036BCH   003757H   00009CH   BYTE   UNIT     CODE          ?PR?ISP_STREAMINGSTARTVIDEO?ISP
003758H   0037E5H   00008EH   BYTE   UNIT     CODE          ?PR?_IOBUFFER_ADD?IOBUFFER
0037E6H   003871H   00008CH   BYTE   UNIT     CODE          ?C_C51STARTUP
003872H   0038F2H   000081H   BYTE   UNIT     CODE          ?PR?I2CSLAVEPACKETSERVICE_PROCESSBUFFER?I2CSLAVEPACKETSERV
FFLX51 LINKER/LOCATER V4.66.30.0                                    05/13/2019  10:42:05  PAGE 4


                                                                    -ICE
0038F3H   00396EH   00007CH   BYTE   UNIT     CODE          ?PR?_UDS_CHANGESESSION?UDS
00396FH   0039E4H   000076H   BYTE   UNIT     CODE          ?PR?_SMBUS_MASTERWRITE?SMBUS
0039E5H   003A56H   000072H   BYTE   UNIT     CODE          ?PR?PACKETMANAGER_INITTABLE?PACKETMANAGER
003A57H   003AC5H   00006FH   BYTE   UNIT     CODE          ?PR?_SYSTEMMANAGER_INITSCHEDULER?SYSTEMMANAGER
```

# Compile과 Link

## PI5008K(AndeSight) Memory map

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .nds32_init   00000290  20040000  20040000  00001000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .BLANK4KB     00001000  20041000  20041000  00002000  2**12
                  CONTENTS, ALLOC, LOAD, DATA
  2 .text         00081e88  20042000  20042000  00003000  2**8
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       000102c4  200c3f00  200c3f00  00084f00  2**8
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .STACK        00002000  200d41c4  200d41c4  0
                  CONTENTS, ALLOC, LOAD, DATA
  5 .data         000096c8  200d61e0  200d61e0  0
                  CONTENTS, ALLOC, LOAD, DATA
  6 .sbss_b       0000000a  200df8a8  200df8a8  0
                  ALLOC
  7 .sbss_h       00000002  200df8b2  200df8b2  0
                  ALLOC
  8 .sbss_w       000002cc  200df8b4  200df8b4  0
                  ALLOC
  9 .bss          0090e744  200dfc00  200dfc00  0
                  ALLOC
 10 .ucHeap       00000004  21000000  200df8a8  0
                  CONTENTS, ALLOC, LOAD, DATA
```

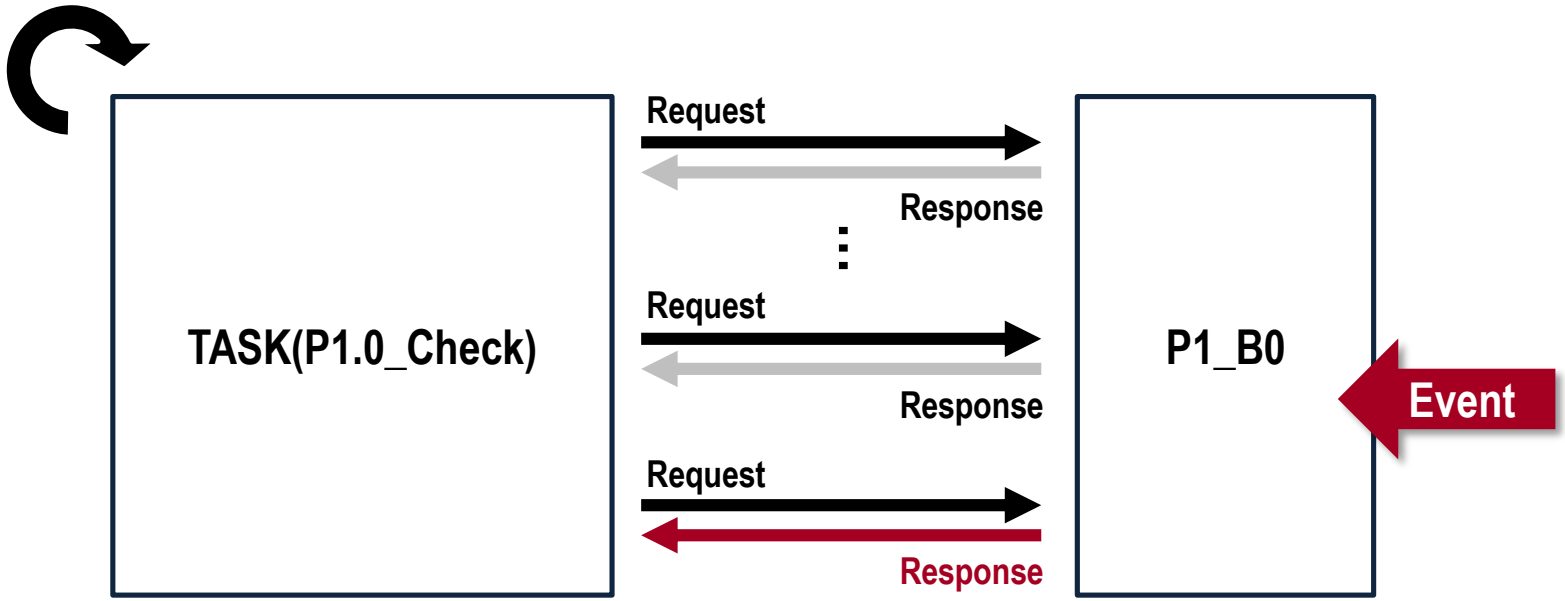
```
200a0fb8 l     F .text  00000140 AppTask_AppQueueInit
200a10f8 l     F .text  000000a0 AppTask_AutoCalTaskStart
200a1198 l     F .text  0000095a AppTask_AutoCalTaskProcess
20425c50 l     O .bss   00000024 viewPoint3D_last.13413
20425c74 l     O .bss   00000001 steadyState.13412
00000000 l    df *ABS*  00000000 AutoCalCtrl.c
20425c75 l     O .bss   00000001 gCalStatus
20425c78 l     O .bss   00000004 gNumStability
20425c7c l     O .bss   00000008 gCircleIdx
20425c84 l     O .bss   00000010 gUpdateParamCnt
20425c94 l     O .bss   00000010 gSerialNoOnCalROM
20425ca4 l     O .bss   00000010 gSerialNoDID
200a58c4 l     F .text  00000038 AutoCalCtrl_SetCalStatus
200a5bdc l     F .text  00000100 AutoCalCtrl_CheckCameraChange
200a5b74 l     F .text  00000068 AutoCalCtrl_CheckECUChange
200a5988 l     F .text  000001ec AutoCalCtrl_GetCamNumStatus
200a47e8 l     F .text  000001bc AutoCalCtrl_AutoCalLoop
200a62d0 l     F .text  0000031c AutoCalCtrl_DetermOutofTolerance
200a5cdc l     F .text  00000388 AutoCalCtrl_CheckMatureValinTolerance
200a626c l     F .text  00000062 AutoCalCtrl_CalcSpeedtokmh
200a6064 l     F .text  0000013c AutoCalCtrl_DrivDirect
200a61a0 l     F .text  000000cc AutoCalCtrl_CheckCANActiveCondi
20425cb4 l     O .bss   00000002 prtWorkStat.12311
```

M5

# Firmware 동작의 이해

# Firmware 동작의 이해 – 폴링과 인터럽트

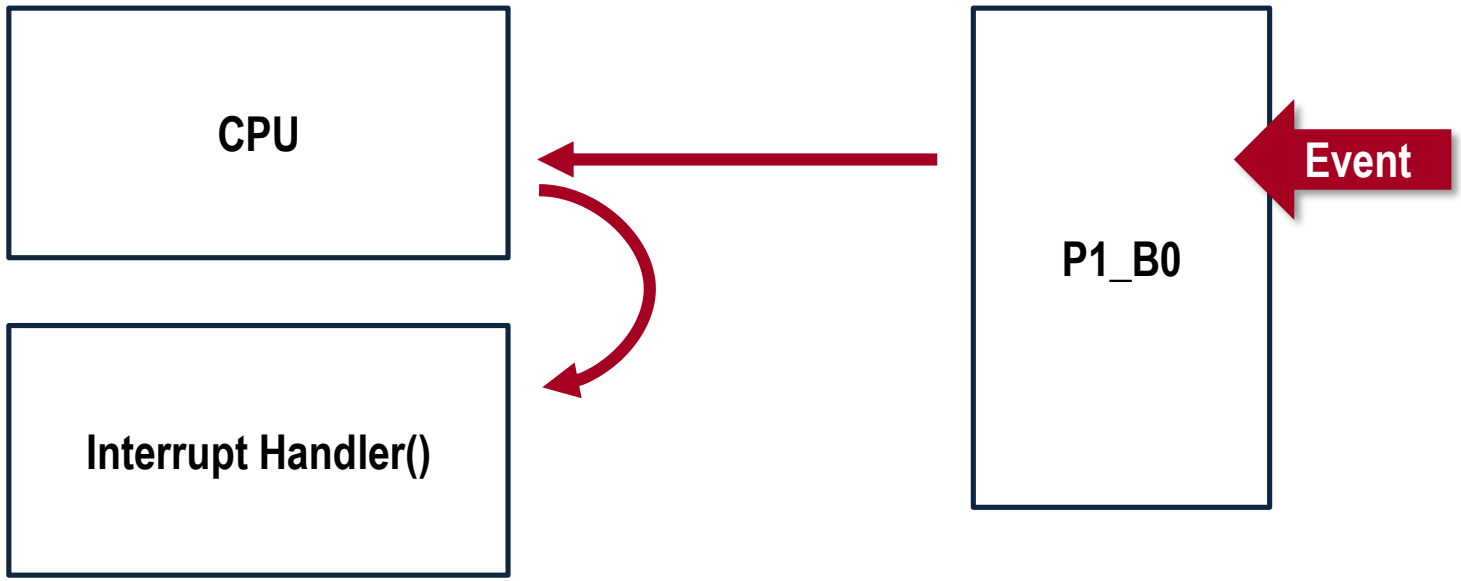
## ‖ 폴링(Polling) 방식




- 폴링(Polling)은 CPU가 정해진 시간 또는 순번에 장치의 상태를 확인하는 방식
- Software 구현이 간단함
- 인터럽트보다는 실시간성 보장이 어렵다.
- 실시간성 보장을 위해 TASK의 주기를 짧게 하면, CPU 점유율이 증가해 성능이 하락할 수 있음

# Firmware 동작의 이해 – 폴링과 인터럽트

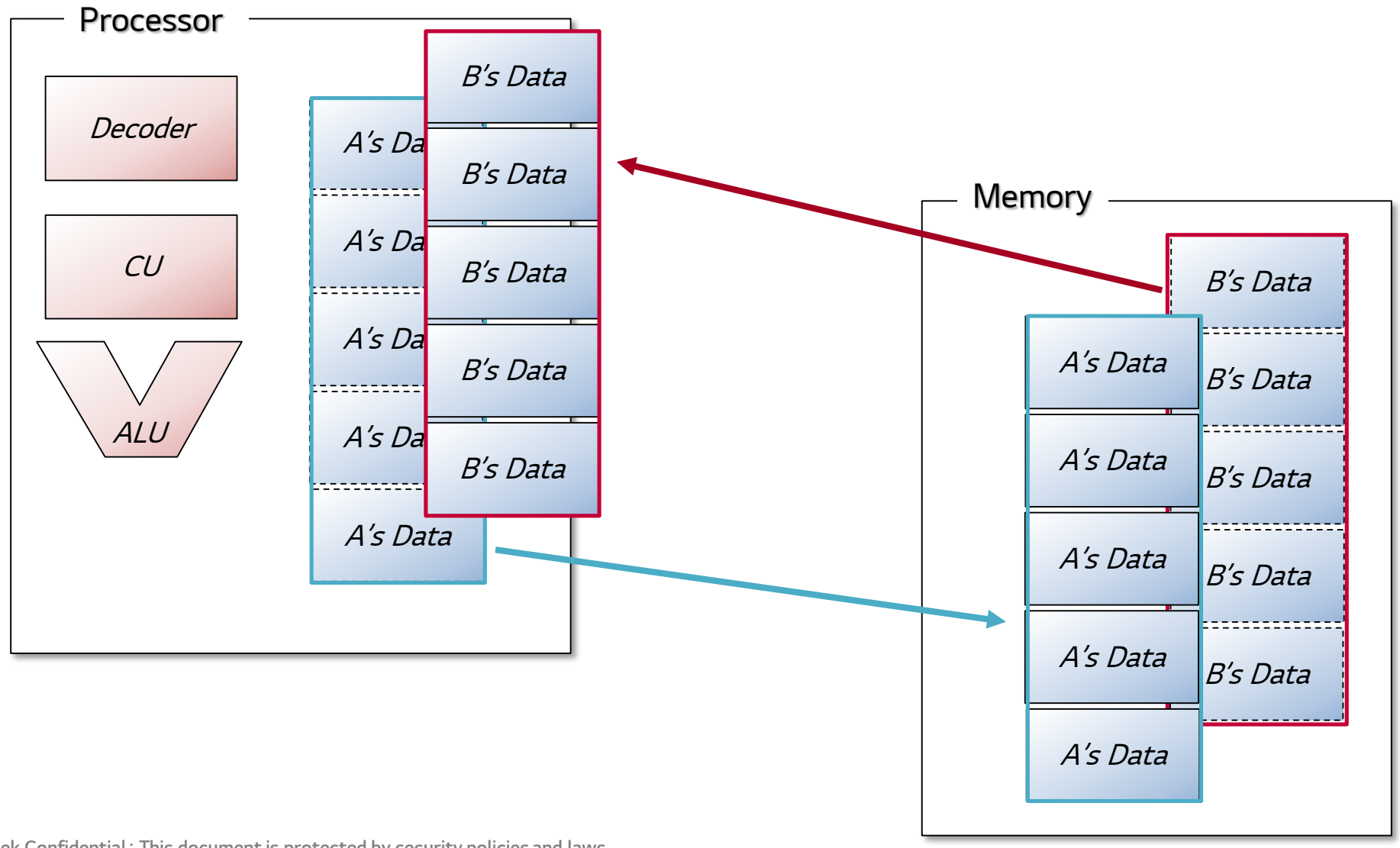
## ‖ 인터럽트(Interrupt) 방식




- 인터럽트(Interrupt)는 장치 상태가 변경되면 CPU가 처리 중인 동작을 멈추고 즉시 해당 신호를 처리하는 것
- Software 구현이 복잡함
  (CPU마다 인터럽트 처리방식이 다르고 인터럽트 우선순위도 고려해야 함)
- 입출력 장치의 상태가 변경되자마자 인터럽트 신호가 발생하므로 실시간성이 좋음

# Firmware 동작의 이해

## ‖ 문맥교환(Context Switching)



# Firmware 동작의 이해

## ‖ 인터럽트 벡터 테이블(Interrupt Vector Table, IVT)

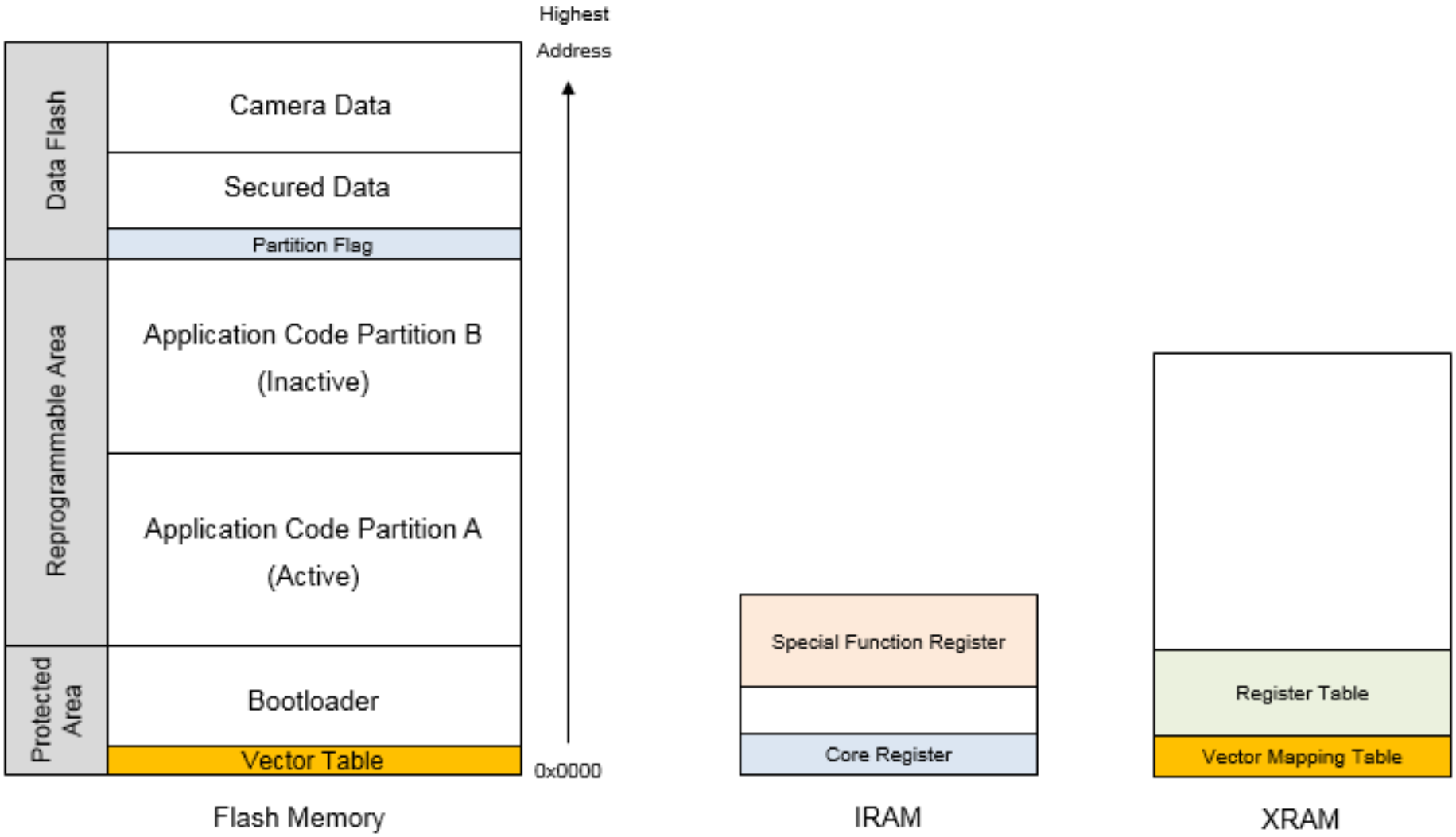
- 인터럽트 요청이 발생했을 때, CPU는 인터럽트 소스가 무엇인지 해당 인터럽트 ISR이 어디에 적재되어 있는지 확인해야 함
- 인터럽트들의 우선 순위와 ISR 위치 정보를 포함

| Address | Interrupt Vector Table | Priority |
|---|---|---|
| 0x0000 0000 | Reset | Top |
| 0x0000 0003 | External Interupt0 | 0 |
| 0x0000 000B | Timer0 Overflow | 1 |
| 0x0000 0013 | External Interrupt1 | 2 |
| 0x0000 001B | Timer1 Overflow | 3 |
| 0x0000 0023 | UART0 | 4 |
| 0x0000 002B | Timer2 Overflow | 5 |
| 0x0000 0033 | SPI0 | 6 |
| 0x0000 003B | SMBus0 | 7 |
| 0x0000 0043 | ADC0 End of Conversion | 8 |
| 0x0000 004B | … | 9 |

EFM8BB3 MCU IVT

# Firmware 동작의 이해

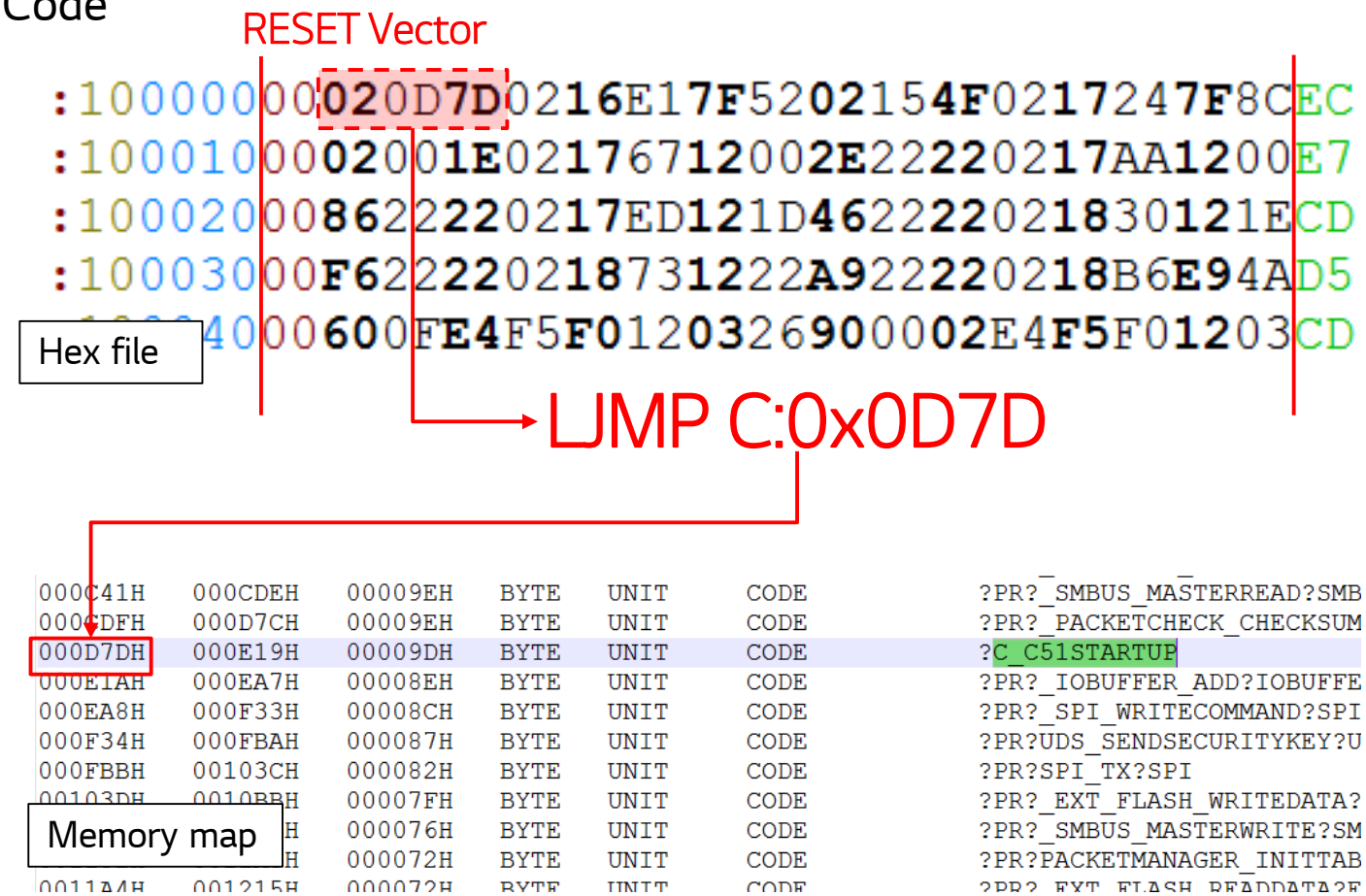
## EFM8BB3 Memory map



# Firmware 동작의 이해 – Reset Flow

## Reset Vector → Startup Code

| Interrupt Source | Vector | Priority |
|---|---|---|
| Reset | 0x0000 | Top |
| External Interrupt 0 | 0x0003 | 0 |
| Timer 0 Overflow | 0x000B | 1 |
| External Interrupt 1 | 0x0013 | 2 |
| Timer 1 Overflow | 0x001B | 3 |
| UART0 | 0x0023 | 4 |
| Timer 2 Overflow / Capture | 0x002B | 5 |
| SPI0 | 0x0033 | 6 |
| SMBus 0 | 0x003B | 7 |
| Port Match | 0x0043 | 8 |
| ADC0 Window Compare | 0x004B | 9 |



# Firmware 동작의 이해 – Reset Flow

## Startup Code → C_MAIN

Startup Code

```
;------------------------------------------------------------------
;sion control flag to RAM that is shared with application
; Absolute BIT segment at 0x20.0 with the name SESSION_FLAG
; [Ryan_180822] Deleted the below ASM code to resolve the RAM uninitializing issue.
;PUBLIC SESSION_FLAG
;           BSEG  AT 20H.0
;SESSION_FLAG:   DBIT  1
;------------------------------------------------------------------


                NAME    ?C_STARTUP


?C_C51STARTUP   SEGMENT   CODE
?STACK          SEGMENT   IDATA

                RSEG    ?STACK
                DS      1

                EXTRN CODE (?C_START)
                PUBLIC  ?C_STARTUP

                CSEG    AT      0
?C_STARTUP:     LJMP    STARTUP1

                RSEG    ?C_C51STARTUP

STARTUP1:

$IF (
EXTRN

$ENDI

IF ID

;====
; [Ry
```

```
01000E12H   CODE      ---        ?C_START
01000000H   CODE      NEAR LAB   ?C_STARTUP
01000016H   CODE      ---        _CallHalService_FlashByteRead
0100008EH   CODE      ---        _CallHalService_FlashByteWrite
01000086H   CODE      ---        _CallHalServi
```

Memory map

```
; This code is required if you use L51_BANK.A51 with Banking Mode 4
;<h> Code Banking
; <q> Select Bank 0 for L51_BANK.A51 Mode 4
$IF (USE_BANKING = 1)
;     <i> Initialize bank mechanism to code bank 0 when using L51_BANK.A51 with Banking Mode 4.
EXTRN CODE (?B_SWITCH0)
                CALL    ?B_SWITCH0
$ENDIF
;</h>

                LJMP    ?C_START

                END
```

LJMP ?C_START

# Firmware 동작의 이해 – Reset Flow

## Startup Code → C_MAIN

```
233 EXTRN CODE (?B_SWITCH0)
234                     CALL    ?B_SWITCH0
235 $ENDIF
236 ;</h>
237                     LJMP    ?C_START
238
239                     END
```

Disassembly

```
                        MOV     81H, #20H
237                             LJMP    ?C_START
00000d97:               LJMP    ?C_START(0DD5H)
00000d9a:               LJMP    Main(22F9H)
00000d9d:               CLR     A
00000d9e:               MOVC    A, @A+DPTR
00000d9f:               INC     DPTR
00000da0:               MOV     R0, A
```

```
59              Main_Init();
              Main:
000022f9:       LCALL   22FFH
60              SystemManager_Running();
000022fc:     LJMP    SystemManager_Running(1D1BH)
65              CallHalService_Init();
              Main_Init:
000022ff:     LCALL   CallHalService_Init(22BBH)
67              TF_FA = FLAG_SET;
```

```
void Main (void)
{
    Main_Init();
    SystemManager_Running();
}

static void Main_Init(void)
```

# Firmware 동작의 이해 – SMBus0 Interrupt Flow

## SMBus0 Interrupt Vector → SMBus0_ISR

| Interrupt Source | Vector | Priority |
|---|---|---|
| Reset | 0x0000 | Top |
| External Interrupt 0 | 0x0003 | 0 |
| Timer 0 Overflow | 0x000B | 1 |
| External Interrupt 1 | 0x0013 | 2 |
| Timer 1 Overflow | 0x001B | 3 |
| UART0 | 0x0023 | 4 |
| Timer 2 Overflow / Capture | 0x002B | 5 |
| SPI0 | 0x0033 | 6 |
| SMBus 0 | 0x003B | 7 |
| Port Match | 0x0043 | 8 |

```
:1000000020D7D0216E17F5202154F0217247F8CEC
:1000100002001E0217671200 2E22220217AA1200E7
:1000200086222202 17ED1 2                21ECD
:1000300F62222021873122 2A92222 0218B6E94AD5
:1000400600FE4F5F0120326900002E4F5F01203CD
```

SMBUS0 Interrupt Vector

LJMP C:0x18B6

```
001830H   001872H   000043H   BYTE   UNIT   CODE   ?PR?TIMER2_ISR?INTERRUPT_HANDLER
001873H   0018B5H   000043H   BYTE   UNIT   CODE   ?PR?SPI0_ISR?INTERRUPT_HANDLER
0018B6H   0018F8H   000043H   BYTE   UNIT   CODE   ?PR?SMBUS0_ISR?INTERRUPT_HANDLER
0018F9H   00193BH   000043H   BYTE   UNIT   CODE   ?PR?ADC0_ISR?INTERRUPT_HANDLER
00193CH   00197EH   000043H   BYTE   UNIT   CODE   ?PR?TIMER3_ISR?INTERRUPT_HANDLER
00197FH   0019C1H   000043H   BYTE   UNIT   CODE   ?PR?I2C0_ISR?INTERRUPT_HANDLER
0019C2H   001A04H   000043H   BYTE   UNIT   CODE   ?PR?TIMER4_ISR?INTERRUPT_HANDLER
001A05H   001A47H   000043H   BYTE   UNIT   CODE   ?PR?TIMER5_ISR?INTERRUPT_HANDLER
001A48H   001A89H   000042H   BYTE   UNIT   CODE   ?PR?TIMER_INIT?TIMER
001A8AH   001AC8H   00003FH   BYTE   UNIT   CODE   ?PR?UDS_REQUESTDOWNLOAD?UDS
```

# Firmware 동작의 이해 – SMBus0 Interrupt Flow

## SMBus0_ISR → ISR Mapping Table

```
78  SI_INTERRUPT (SMBUS0_ISR, SMBUS0_IRQn)
79  {
80      IE_EA = FLAG_SET;
81      IR_TABLE[IR_TABLE_SMB0]();
82  }
83
```

```
02000000H    XDATA    ---      IR_TABLE
01002224H    CODE     ---      IR_Table_Init
000000E4H    DATA     BYTE     IT01CF
00002800H    NUMBER   ---      Jump_Application
000000B1H    DATA     BYTE     LFO0CN
010025B7H    CODE     ---      Main
00000080H    DATA     BYTE     P0
00000080H    DATA     BIT      P0_B0
00000080H.1  DATA     BIT      P0_B1
00000080H.2  DATA     BIT      P0_B2
00000080H.3  DATA     BIT      P0_B3
```

```
25  typedef enum
26  {
27      IR_TABLE_EX0,       /* 0x0003   External Interrupt 0 */
28      IR_TABLE_ET0,       /* 0x000B   Timer0 Interrupt */
29      IR_TABLE_EX1,       /* 0x0013   External Interrupt 1 */
30      IR_TABLE_ET1,       /* 0x001B   Timer1 Interrupt */
31      IR_TABLE_UART0,     /* 0x0023   UART0 Interrupt */
32      IR_TABLE_GPIOI2C,   /* 0x002B   GPIO I2C by Timer2 */
33      IR_TABLE_SPI0,      /* 0x0033   SPI */
34      IR_TABLE_SMB0,      /* 0x003B   SMBus */
35      IR_TABLE_ADC0,      /* 0x0053   ADC0 End of Conversion */
36      IR_TABLE_ET3,       /* 0x0073   Timer3 Interrupt */
37      IR_TABLE_I2C0,      /* 0x0083   Slave I2C */
38      IR_TABLE_ET4,       /* 0x008B   Timer4 Interrupt */
39      IR_TABLE_ET5,       /* 0x0093   Timer5 Interrupt */
40      IR_TABLE_SIZE,
41  } InterruptTableIndexes_e;
```

21 Bytes = 7 * 3 Bytes (Instruction set size of function pointer)

IR_TABLE[*IR_TABLE_SMB0*]의 Address = XDATA: 0x0015

XRAM:0x0 : 0x0 <Traditional> | New Renderings...

LCALL C:0x064B

```
0x00000000  FF003AFF 2033FF00 3AFF1F16 FF003AFF 003AFF1F 72FF064B FF003AFF 210DFF00 3AFF003A
0x00000024  FF003A00 0000FFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00000048  FFFF00FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFF00
0x0000006C  000000FF 1B0700FF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

# Firmware 동작의 이해 – SMBus0 Interrupt Flow

## ISR Mapping Table[SMBus0] → SMBus0 ISR Function

```
000093H    000095H    000003H    BYTE    OFFS..    CODE          ?INTERRUPT_HANDLER?00093
000096H    00043DH    0003A8H    BYTE    UNIT      CODE          ?C?LIB_CODE
00043EH    00064AH    00020DH    BYTE    UNIT      CODE          ?PR?UDS_DOWNLOADCTRL?UDS
00064BH    000799H    00014FH    BYTE    UNIT      CODE          ?PR?SMBUS0_ISR?SMBUS
```

```
289 static void SMBUS0_ISR(void)
290 {
291     static bool isFisrtRxData = true;
292     static RegAddr_t startRegAddr = VALUE_CLR;
293     static uint8_t slaveRxType = DATA_R_TYPE;
294     bool bFail = false;
295
296     if (SMB0CN0_ARBLOST == FLAG_CLR)
297     {
298         switch (SMB0CN0 & 0xF0U)
299         {
300         //-->> [MASTER MODE]
301             // Master Transmitter/Receiver: START condition transmitted.
302             case SMB_MTSTA:
303                 SMB0DAT = gSlaveAddr8; // Load address of the target slave
304                 SMB0DAT &= 0xFEU;      // Clear the LSB of the address for the R/W bit
305                 SMB0DAT |= gSMBusRW;   // Load R/W bit
306                 SMB0CN0_STA = FLAG_CLR;    // Manually clear START bit
307                 break;
308
309             // Master Transmitter: Data byte transmitted
```

# Firmware 동작의 이해

## SMBus0 Interrupt Context Switching

EFM8BB3 Core Register
Context Switching

| Core Register | Stack Top |
|---|---|
| IDATA: 0x07 | R7 |
| IDATA: 0xF0 | R6 |
| … | … |
| IDATA: 0x02 | R2 |
| IDATA: 0x01 | R1 |
| IDATA: 0x00 | R0 |
| IDATA: 0xD0 | PSW |
| IDATA: 0x82 | DPL |
| IDATA: 0x83 | DPH |
| IDATA: 0xF0 | B |
| IDATA: 0xE0 | ACC |

```
 78          SI_INTERRUPT (SMBUS0_ISR, SMBUS0_IRQn)
             SMBUS0_ISR:
000018b6:      PUSH    0E0H
000018b8:      PUSH    0F0H
000018ba:      PUSH    83H
000018bc:      PUSH    82H
000018be:      PUSH    0D0H
000018c0:      MOV     0D0H, #00H
000018c3:      PUSH    00H
000018c5:      PUSH    01H
000018c7:      PUSH    02H
000018c9:      PUSH    03H
000018cb:      PUSH    04H
000018cd:      PUSH    05H
000018cf:      PUSH    06H
000018d1:      PUSH    07H
 80              IE_EA = FLAG_SET;
000018d3:      SETB    IE.7
 81              IR_TABLE[IR_TABLE_SMB0]();
000018d5:      MOV     DPTR, #15H
000018d8:      LCALL   ?C?PLDXDATA(3BDH)
000018db:      LCALL   ?C?ICALL(40CH)
 82          }
000018de:      POP     07H
000018e0:      POP     06H
000018e2:      POP     05H
000018e4:      POP     04H
000018e6:      POP     03H
000018e8:      POP     02H
000018ea:      POP     01H
000018ec:      POP     00H
000018ee:      POP     0D0H
000018f0:      POP     82H
000018f2:      POP     83H
000018f4:      POP     0F0H
000018f6:      POP     0E0H
000018f8:      RETI
```

SMBus0 인터럽트 처리를 위한
Context Switching

SMBus0 인터럽트 처리 후
Context Switching

# Embedded System에서의 Firmware는 ...

- RSTb pin으로 전원이 공급되면, MCU Power-up
- Vector Table의 Reset Vector로 PC(Program Counter)가 SET
- Reset Vector의 Startup Code Address로 JUMP
- Startup Code 수행 후, C_MAIN으로 JUMP
- MAIN Application 처리 중 Interrupt 발생 시, Vector Table을 참조해 ISR 처리 완료 후 MAIN Application을 다시 수행

감사합니다.

끝