



C언어 – HW1

임베디드스쿨1기

Lv1과정

2020. 07. 21

손표훈

2. C언어의 필요성

- (1) C언어란? 기계어(1과 0으로 구성된 명령어 체계)를 인간이 직관적으로 이해할 수 있게 해주는 언어
- (2) C언어 그대로 사람이 의도한 내용을 전달하지 못함
"컴파일러"를 통해 C -> 기계어로 변환하여 전달
- (3) C언어 말고도 어셈블리어라고 기계어와 1:1 대응 되는 언어가 있다.
- (4) 앞에 나온 임베디드 시스템에서 **MCU**에 C언어로 설계된 펌웨어를 탑재하여 시스템을 제어한다.

1. Microprocessor? Microcontroller?

- (1) Microprocessor : 그림1과 같이 **범용** 컴퓨터 시스템에 사용되는 칩을 말한다.
- (2) Microprocessor는 칩 자체만으로 아무것도 할 수 없다.
- (3) **외부에서** 여러 주변기기들이 연결 되어야만 사용자의 목적에 맞는 시스템을 구성할 수 있다.



그림1. 범용 컴퓨터 시스템

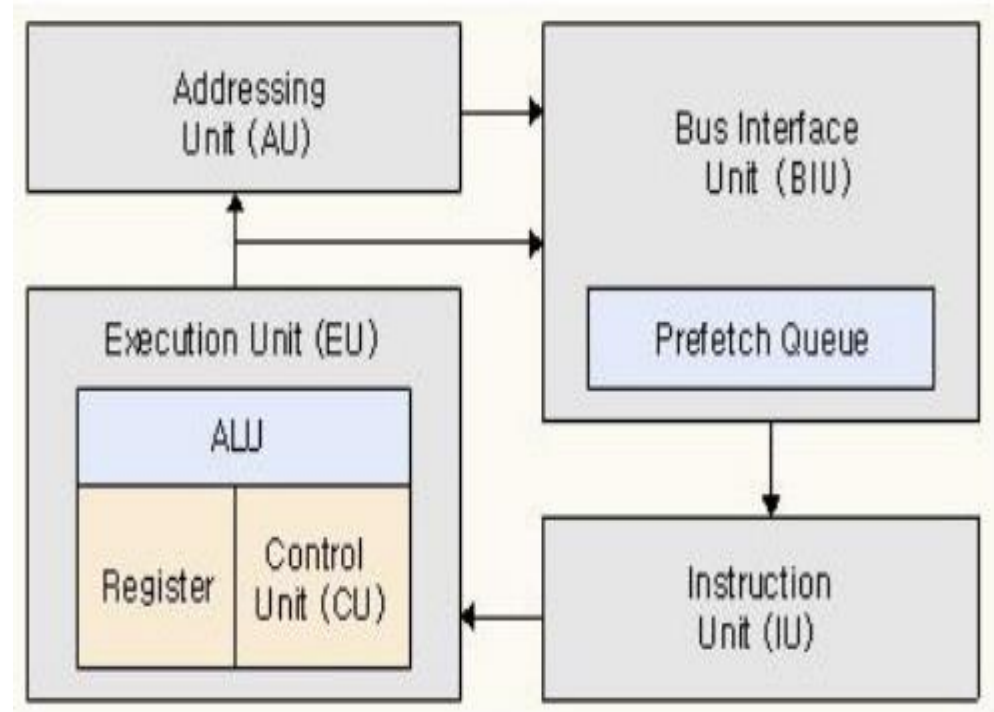


그림2. Microprocessor 구조

1. Microprocessor? Microcontroller?

- (3) Microcontroller : 아래 그림1과 같이 “특정기능”을 위한 임베디드 시스템에 사용되는 칩을 말한다.
- (4) Microcontroller는 그림2와 같이 주변장치들이 “하나의 칩”에 CPU와 연결되어있다.

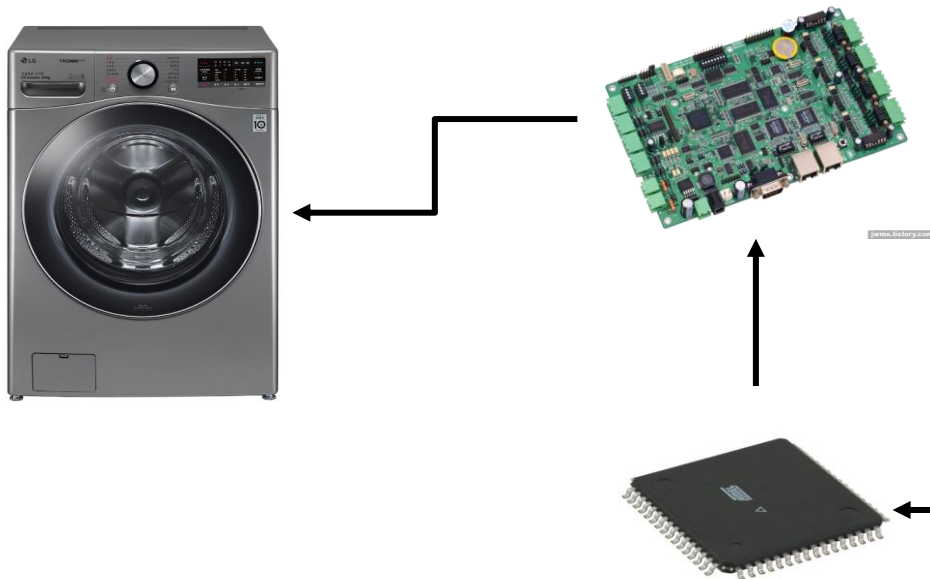


그림1. 임베디드 시스템 예시(세탁기)

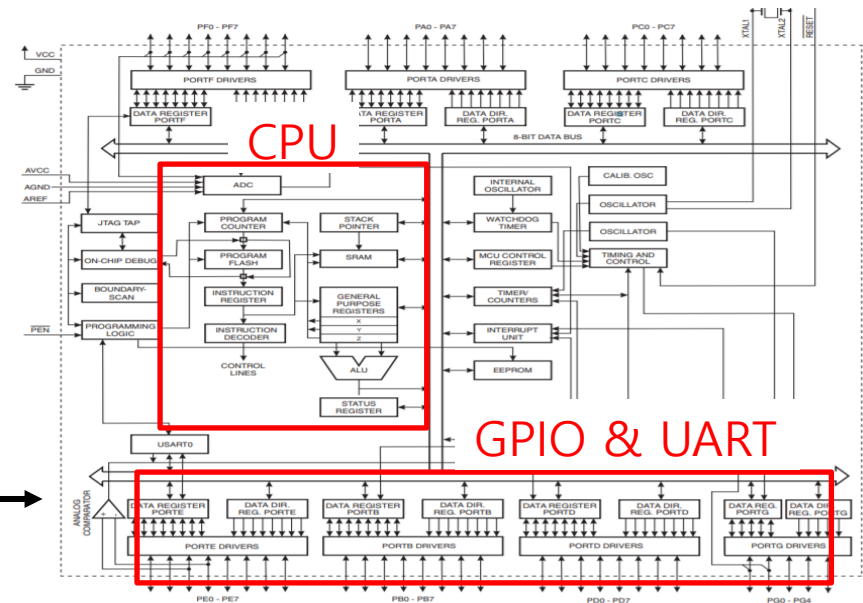
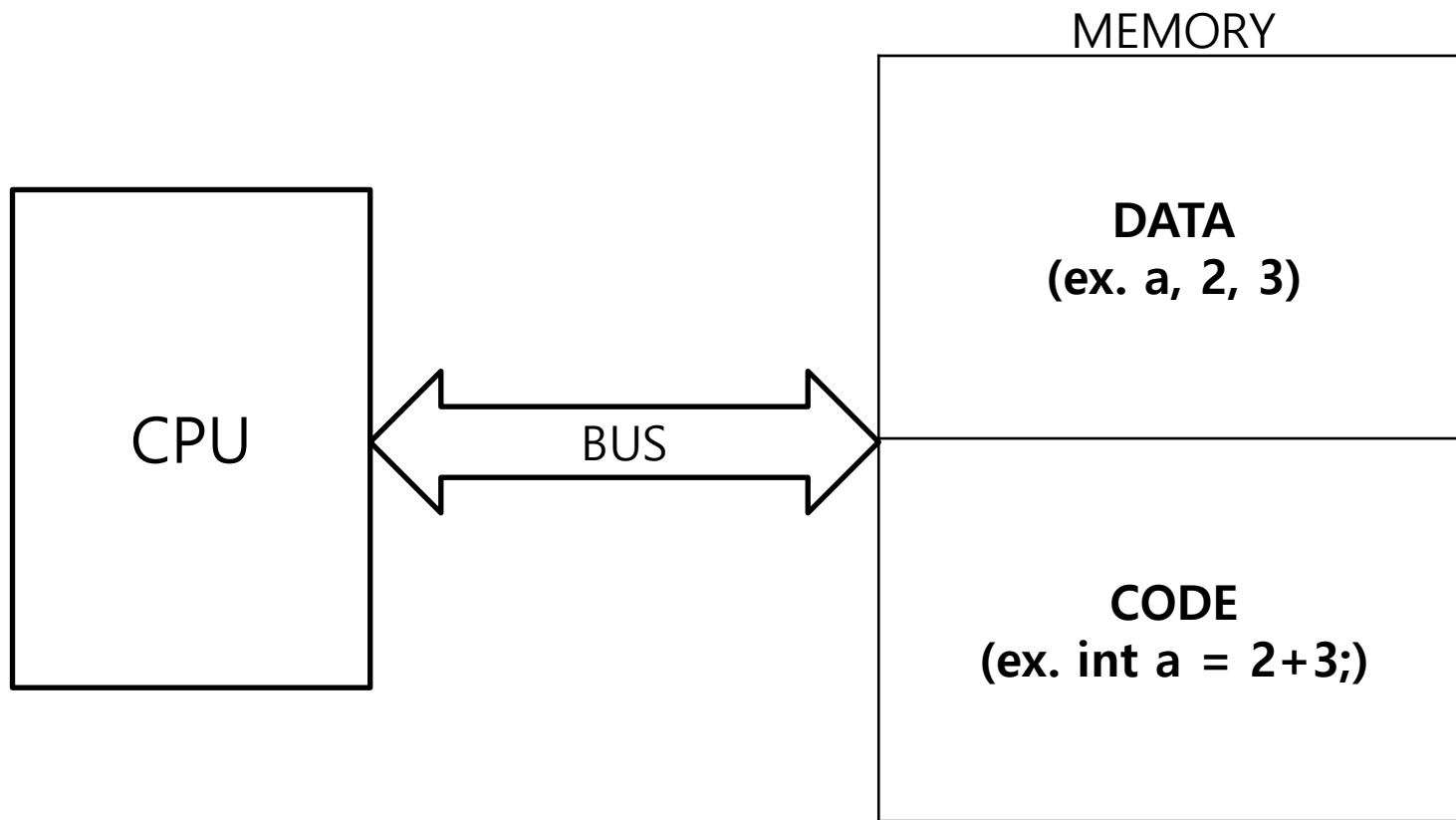


그림2. Microcontroller(Atmega128)

2. 폰 노이만? 하버드? 구조

- (1) 폰 노이만 구조 : 동일 메모리내에 데이터와 명령어가 같이 들어있다.
프로그램 내장 메모리 순차 처리방식
하나의 버스로 데이터와 명령어를 CPU에서 처리한다



2. 폰 노이만? 하버드? 구조

(2) 폰 노이만 구조 단점 : 메모리 지연

Ex. $2+3$ (컴퓨터한테 2를 불러와라 -> 3을 불러와라 -> 더하기는 모두 명령어)

* 명령어에 따라 4단계를 거치지 않음

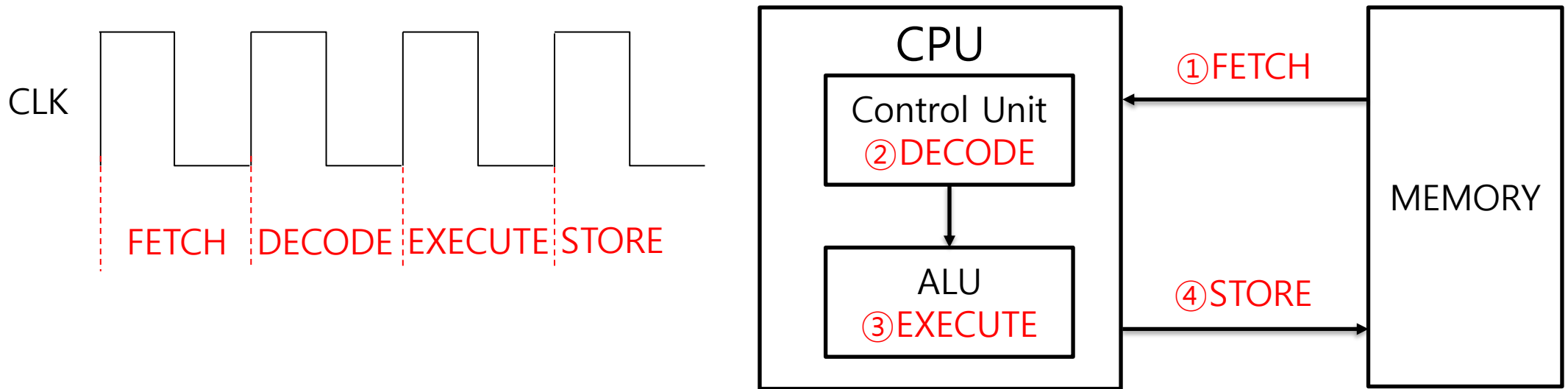


그림1. 명령어 실행 단계

(3) 명령은 각각 그림1과 같이 실행됨 순차 실행으로 하나의 명령을 실행하는데 4클럭이나 소모

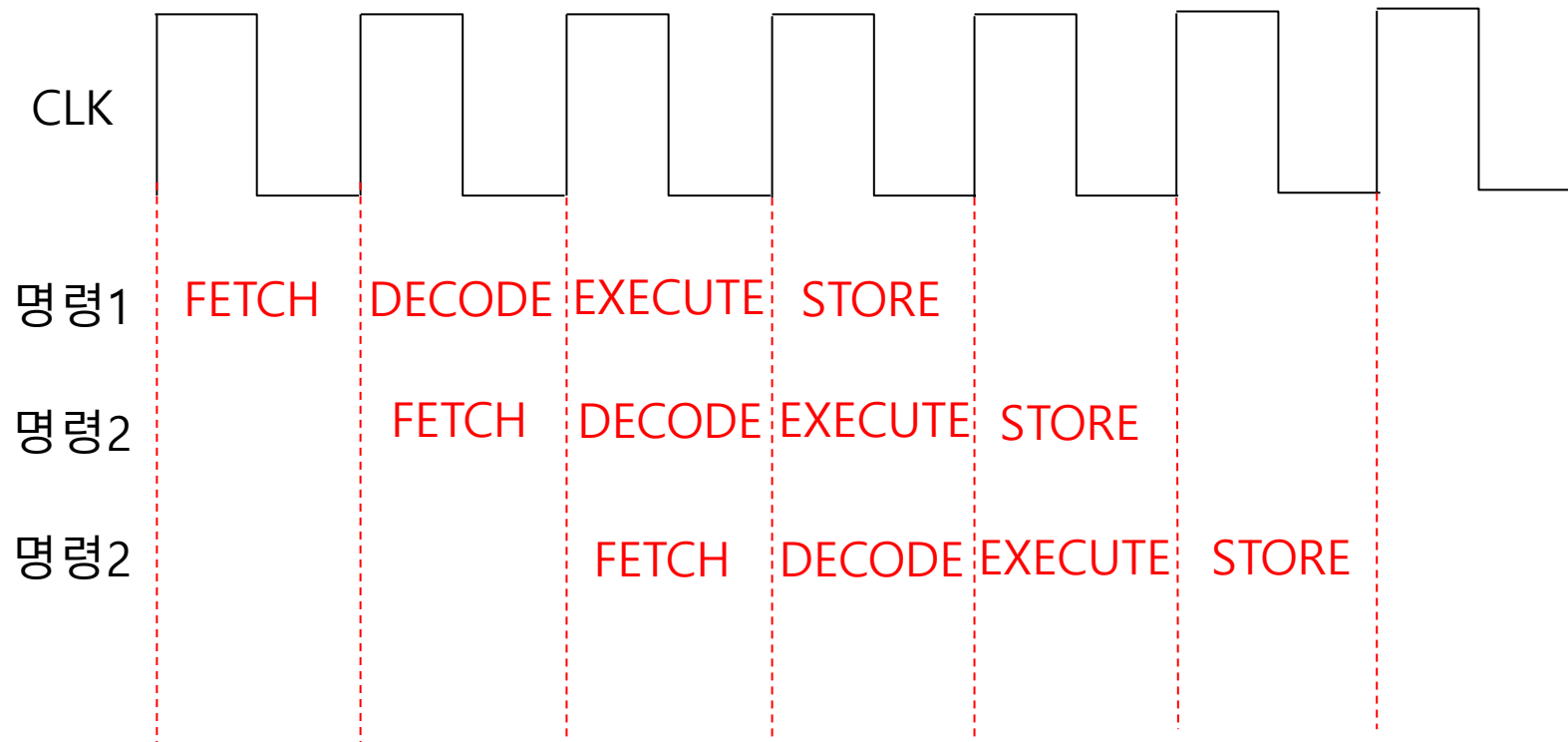
(4) 먼저 실행된 명령이 끝날 때 까지 다음 명령어 실행을 못함
-> 해결은 파이프 라이닝 기법(하버드 구조)

2. 폰 노이만? 하버드? 구조

(5) 파이프 라이닝 기법

Ex. $2+3$ (명령1 : 2를 불러와라 -> 명령2 : 3을 불러와라 -> 명령3 : 더하기)

* 명령어에 따라 4단계를 거치지 않음



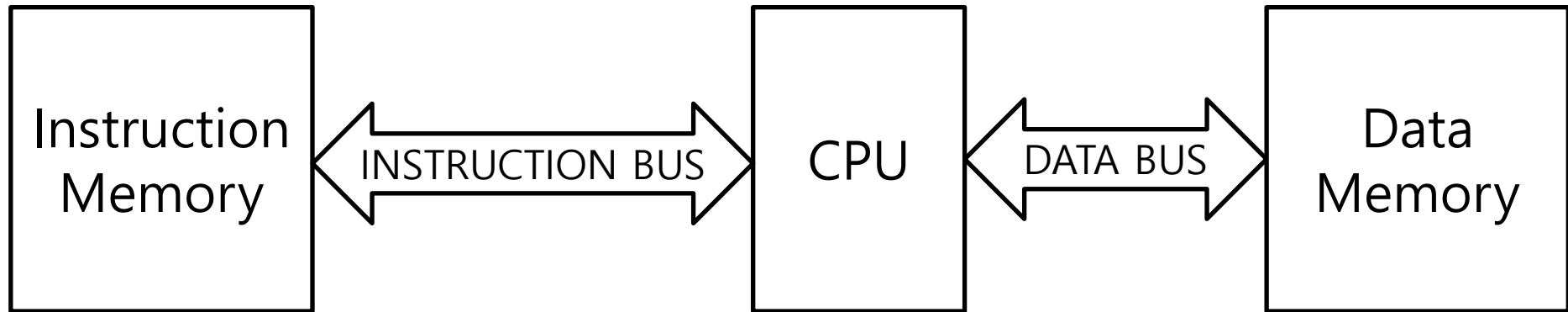
(6) 7클럭에 $2+3$ 연산이 끝남(파이프라이닝 없이 하면 $4*3 = 12$ 클럭 소요)

(7) 파이프 라이닝 기법을 위한 하드웨어 구조 = 하버드 구조

2. 폰 노이만? 하버드? 구조

(8) 하버드구조 : 데이터 메모리와 명령어 메모리가 **물리적으로 분리**되어 있음
데이터버스와 명령어 버스가 **각각** 구성됨

(9) 버스구조가 복잡하여 설계가 어려움



(10) 임베디드에선 폰 노이만? 하버드?

-> 임베디드는 제어 위주의 프로그램이 많다 보니 CPU가 처리하는 명령어도 많아진다..
폰 노이만 구조를 사용하게 되면 명령어 처리 시간이 늘어나고 결과 값을 얻는데 그
만큼 지연되기 때문에 고반응 시스템에 사용하기에 힘들지 않을까 싶다..(경험해보면
 좋을 것 같다)

3. 변수

- (1) 변수란? 특정 데이터가 저장되지 않은 메모리 영역의 지정된 이름!
- (2) 변수는 문자 또는 숫자로 구성 될 수 있다(변수명은 반드시 문자나 밑줄로 시작해야함)

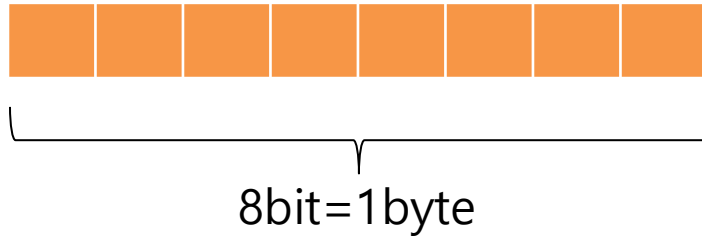
```
//char RefAxis;
//Axis Position = { 0, };
//double Angle=0.0F;
float Matrix[MatrixSize][MatrixSize] = { 0, };
//float DeterminantResult;
float Value[3] = { 0, };

float yn=0;
static float yn_1=0;
const float LPF_CONST1 = 0.083026;
const float LPF_CONST2 = 0.916974;
```

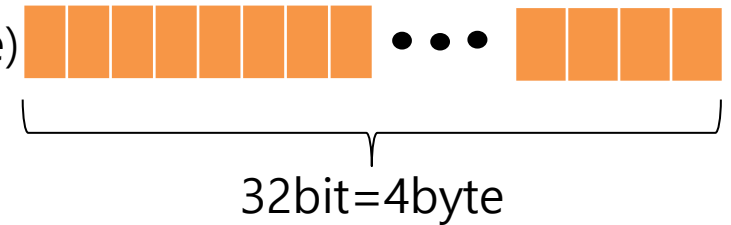
3. 변수

- (3) C의 변수에는 특정 유형(Data Type)이 존재, 메모리 크기와 레이아웃(메모리계층)을 결정
※ int크기는 CPU마다 다름!

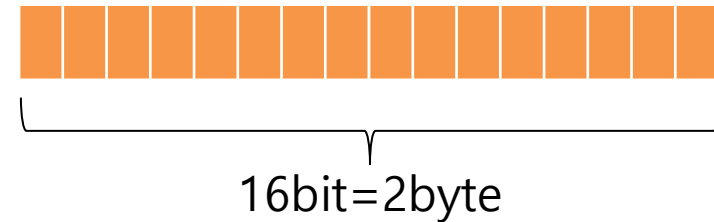
char형



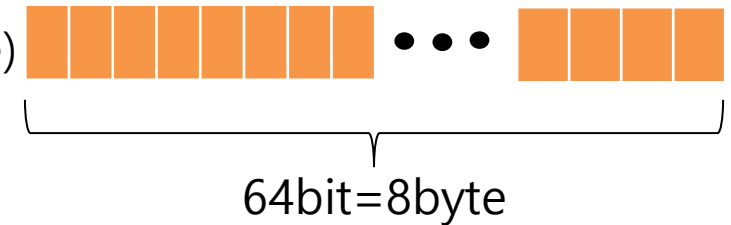
float형(4byte)



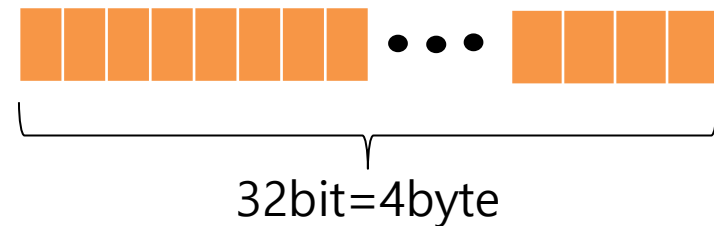
int형(2byte)



double형(8byte)



int형(4byte)



void형(?byte)

아무런 Type도아님
(크기도 정해지지 않음)

3. 변수

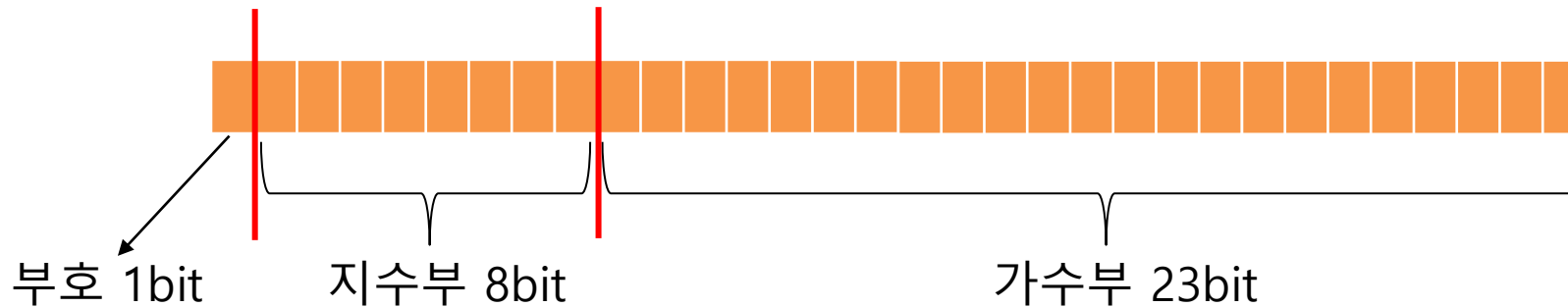
(4) float(single-precision), double(double-precision)이다.

single-precision : 단정밀도 32bit

double-precision : 배정밀도 64bit(단정밀도의 두 배)

(5) 둘 다 실수를 표현함.

(6) 자세한 실수표현 계산법은 <https://whatisthenext.tistory.com/146> 참조



4. 데이터 유형(Data Type)

- 데이터 유형에 따라 메모리에 차지하는 공간과 저장된 비트 방식이 다르다
(signed or unsigned)
- unsigned가 없으면 signed

유형	크기	범위
char	1byte	-126 ~ 127
unsigned char	1byte	0 ~ 255
int	1byte	-32,768 ~ 32,767 or -2,147,483,648 ~ 2,147,483,647
unsigned int	2 or 4byte	0 ~ 65,535 or 0 ~ 4,294,967,295
short	2 or 4byte	-32,768 ~ 32,767
unsigned short	2byte	0 ~ 65,535
long	2byte	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
unsigned long	8byte	0 ~ 16,446,744,073,709,551,615
Float	4byte	1.2E-38 ~ 3.4E+38
double	4byte	2.3E-308 ~ 1.7E+308
long double	8byte	3.4E-4932 ~ 1.1E+4932

4. 데이터 유형(Data Type)

- Signed 데이터형과 Unsigned 데이터형의 범위를 초과하면?

```
char a = 127;
unsigned char b = 255;

printf("unsigned char = %d\n", a);
printf("signed char = %d\n", b);

a = a + 1;
b = b + 1;

printf("!!!!Over Flow!!!!\n");
printf("unsigned char = %d\n", a);
printf("signed char = %d\n", b);

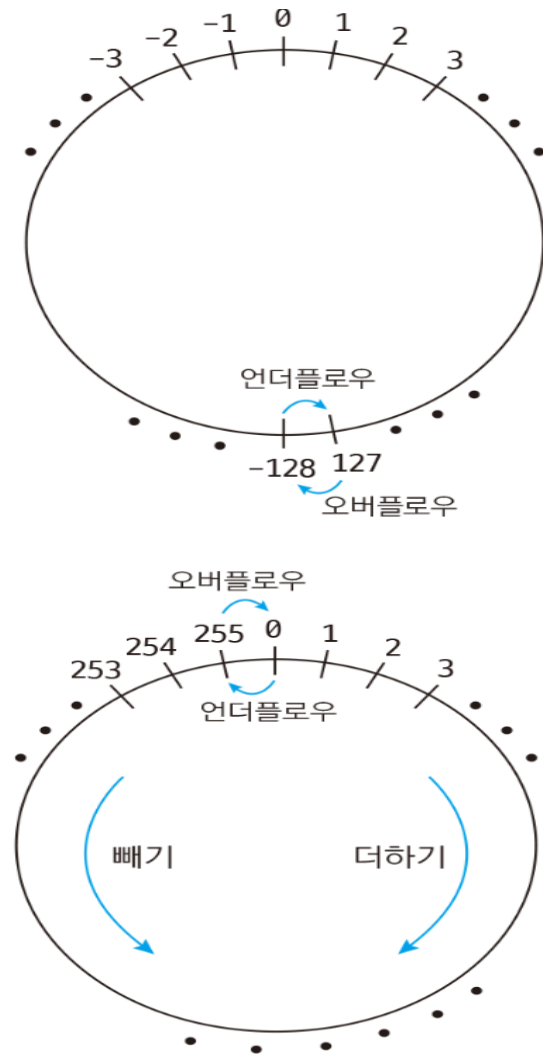
a = -129;
b = -1;

printf("!!!!Under Flow!!!!\n");
printf("unsigned char = %d\n", a);
printf("signed char = %d\n", b);

return 0;
```

```
unsigned char = 127
signed char = 255
!!!!Over Flow!!!!
unsigned char = -128
signed char = 0
!!!!Under Flow!!!!
unsigned char = 127
signed char = 255
```

C:\Users\SON\source\repos\Test
이 창을 닫으려면 아무 키나 누르



5. 데이터 유형 변환(Type Casting)

- (1) Type Casting은 데이터 타입을 다른 데이터 타입으로 바꾸는 것
- 표현하고자 하는 자료의 크기를 확장 또는 축소 시 사용한다

(1) 아래 그림1과 같이 변수명 앞에 "(원하는 데이터 유형)"를 붙여 사용

(2) Casting 지시를 하지 않으면 그림2처럼 컴파일러는 범위가 큰 자료형으로 강제로 Casting한다

```
#include <stdio.h>

int main(void)
{
    char a = 127;

    printf("a = %f\n", (float)a);

    return 0;
}
```

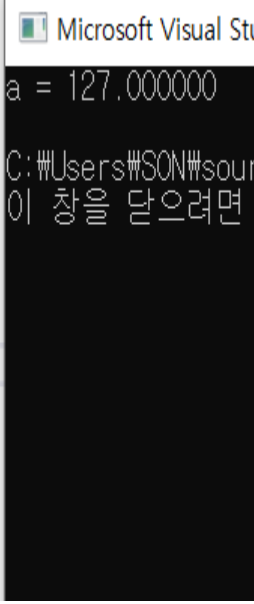


그림1

```
#include <stdio.h>

int main(void)
{
    char a = 127;
    float b = a;

    printf("a = %d\n", a);
    printf("b = %f\n", b);

    return 0;
}
```

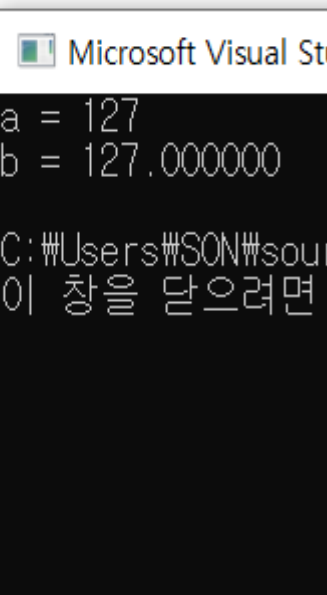


그림2