Busan Software Meister High School

MICROPROCESSOR

2309 양유빈

20230316

마이크로프로세서

손정웅선생님

OVERVIEW

- 구조체와 패딩
- 구조체와 공용체
- 열거형
- 비트필드
- stdint.h
- 비트연산
- 통신방법

구조체와 패딩

structs and padding

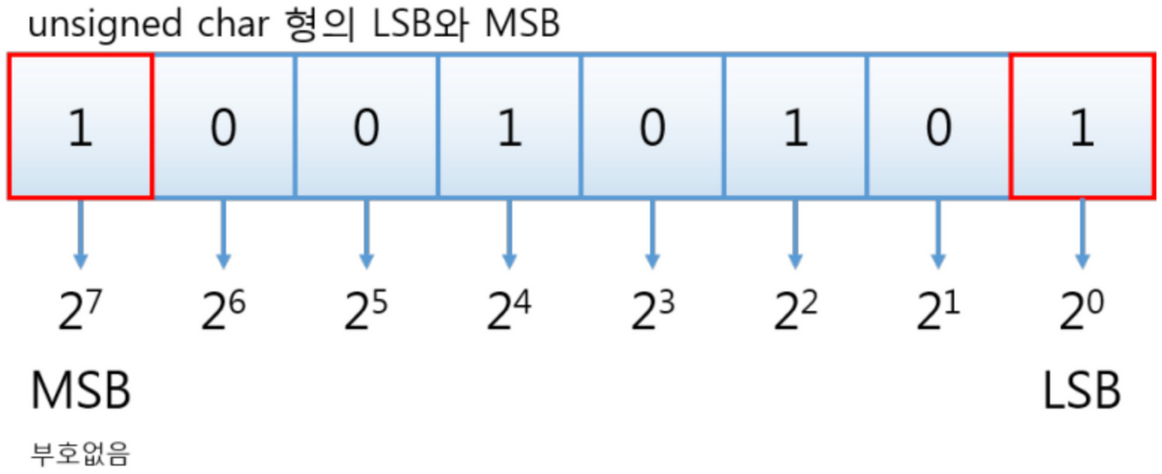
📌 구조체에서 패딩을 사용하지 않고자 할 때는 다음과 같은 속성 사용

```
__attribute__((__packed__)) test_t;
```

구조체와 공용체

structs and unions

- 16진수 한 자리는 4비트
- MCU에서는 리틀 엔디안(Little Endian) 방식 사용
- 리틀 엔디안은 낮은 주소에 LSB(Least Significant Bit)가 저장
- 빅 엔디안은 낮은 주소에 MSB가 저장




```c
#include<stdio.h>
typedef union num{
    char a;
    short b;
    int c;
} Num;

int main(void){
    Num n;
    n.c = 0x12345678;
    printf("%x\n", n.a);
    printf("%x\n", n.b);
    prinf("%x\n", n.c);
    return 0;
}
```

구조체와 공용체

structs and unions

```c
//공용체 안에 구조체 집어 넣음
#include<stdio.h>
typedef union{
    unsigned int val;
    struct  {
        unsigned char r;
        unsigned char g;
        unsigned char b;
        unsigned char a; //투명도 00~ff(불투명)
    } rgba;
} color;

int main(){
    color sample;
    sample.val=0xffffffff;
    sample.rgba.b=0xff;
    printf("%x\n", sample.val);
    return 0;
}
```

열거형

enum

- 정수형 상수에 이름을 붙혀 코드 이해를 도움
- 열거형을 사용하면 상수를 편리하게 정의 가능
- 열거형의 각 값들은 ,(콤마)로 구분하며 =(할당 연산자)를 사용하여 값을 할당
- 열거형의 값은 처음에만 할당해주면 그 아래에 오는 값들은 1씩 증가하면서 자동 할당 (아무 값도 할당하지 않으면 0부터 시작)
- 열거형 이름이나 값을 정의할 때 대문자만 사용하는 경우 多 , 대문자면 열거형일 가능성 높음
- 단어와 단어 사이는 _를 주요 사용

  (ex. WEEK_SUN)

```
enum 열거형명
{
    값1 = 초기값,
    값2,
    값3,
    값4,
    ...
}; // typedef 사용 가능 (별칭 줘야함)
```

```
5   #include<stdio.h>
6   typedef enum week{
7       SUN = 0,
8       MON,
9       TUE,
10      WED,
11      THD,
12      FRI,
13      SAT
14  } Week;
15  int main(void){
16      Week ju;
17      ju = TUE;
18      printf('"%d\n', ju);
19      return 0;
20  }
```

열거형

enum

```c
// 열거형 3
// Scanf로 교과 코드를 입력하면 과목명이 출력되도록 프로그래밍 하시오.
#include<stdio.h>
typedef enum subject_code{
    LINUX = 3,
    MICRO = 5,
    PROJECT = 8
} Code;
int main(void){
    Code emb;
    scanf("%d", &emb);
    emb = LINUX;
    switch(emb){
        case 3:
            printf("LINUX\n");
            break;
        case 5:
            printf("MICRO\n");
            break;
        case 8:
            printf("PROJECT\n");
            break;
        default:
            printf("default\n");
            break;
    }
    return 0;
}
```

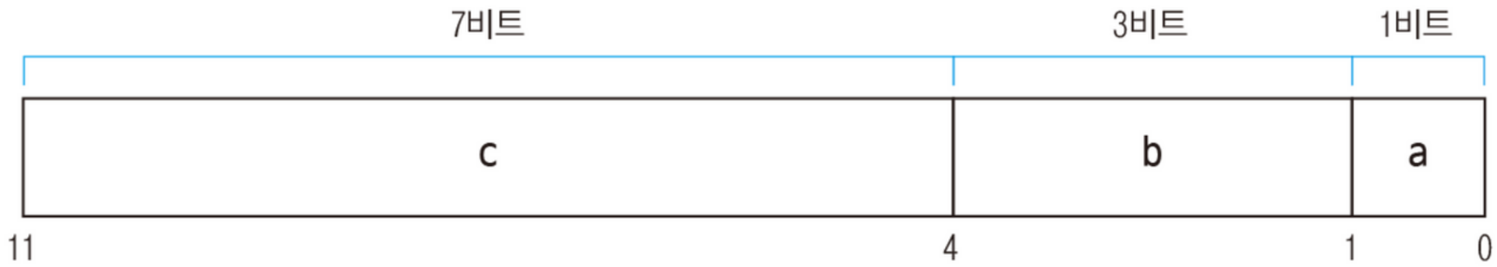
비트필드

bitfield

- 구조체의 멤버는 각 자료형 크기만큼 공간차지
- 구조체 비트 필드를 사용하면 구조체 멤버를 비트 단위로 저장 가능
- 비트필드는 정수형 자료형만 사용하므로, 실수는 사용할 수 없음
- 보통 비트 필드는 부호 없는 자료형(unsigned)을 사용
- CPU나 기타 칩의 플래그를 다루는 저수준(low level) 프로그래밍을 할 때 기본 자료형보다 더 작은 비트 단위로 값을 가져오거나 저장하는 경우가 많으므로 구조체 비트 필드가 유용하게 사용

```
 8  struct 구조체명
 9  {
10      정수형 멤버명 1 : 비트수;
11      정수형 멤버명 2 : 비트수;
12      ...
13  };
```

▼ 그림 56-1 구조체 비트 필드

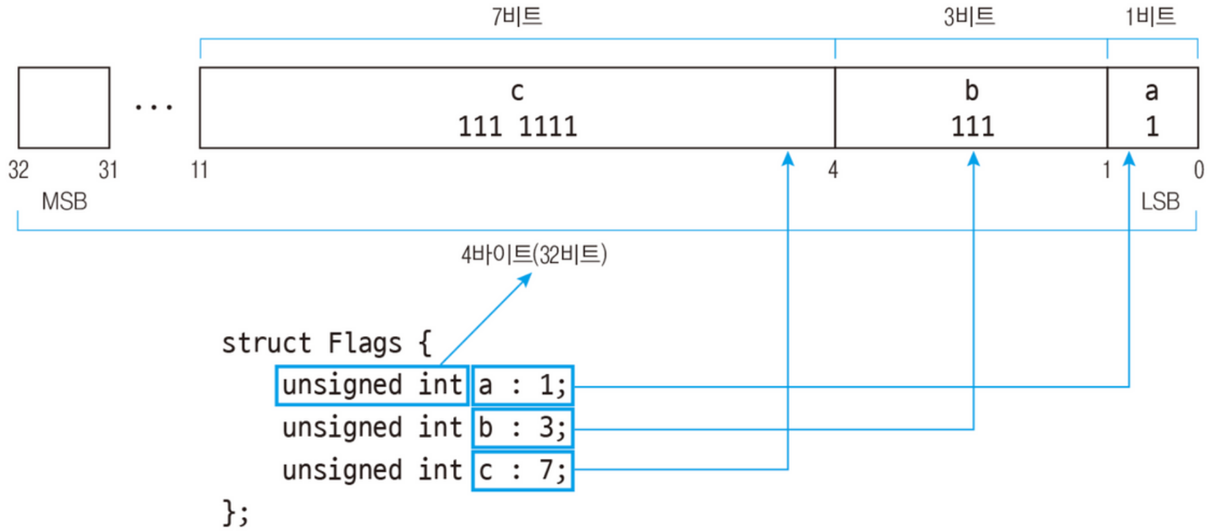


비트필드

bitfield

// 비트 필드의 각 멤버는 최하위(LSB)부터 차례대로 배치

▼ 그림 56-2 구조체 비트 필드의 구성

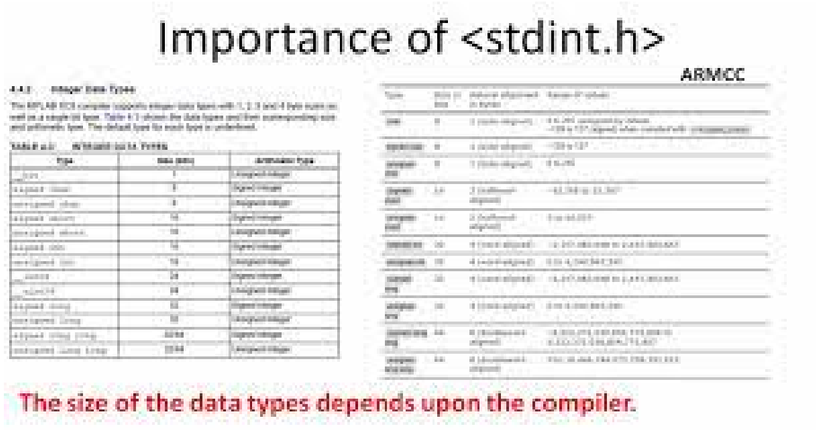



```c
#include <stdio.h>

struct Flags {
    unsigned int a : 1;     // a는 1비트 크기
    unsigned int b : 3;     // b는 3비트 크기
    unsigned int c : 7;     // c는 7비트 크기
};

int main()
{
    struct Flags f1;    // 구조체 변수 선언

    f1.a = 1;       //   1: 0000 0001, 비트 1개
    f1.b = 15;      //  15: 0000 1111, 비트 4개
    f1.c = 255;     // 255: 1111 1111, 비트 8개

    printf("%u\n", f1.a);     //   1:         1, 비트 1개만 저장됨
    printf("%u\n", f1.b);     //   7:       111, 비트 3개만 저장됨
    printf("%u\n", f1.c);     // 127: 111 1111, 비트 7개만 저장됨, 사이즈: 4

    return 0;
}

// 실행 결과
// 1
// 7
// 127
```

stdint.h

bitfield

📌C언어 표준은 1990년대 후반에 개정되어 C99에서는 비트 길이를 고유하게 정의한 데이터 형식을 추가하여 헤더파일을 <stdint.h> 에 정의함



비트연산

bit operation

- 비트 연산자
  - 비트 단위로 논리 연산을 할 때 사용하는 연산자
  - 비트 단위로 전체 비트를 왼쪽이나 오른쪽으로 이동시킬 때도 사용
  - & : 대응되는 비트가 모드 1이면 1을 반환 앤드 연산 등등
- selective Clear: 0으로 AND
- selcetive Set: 1로 OR
- selective Complement: 1로 XOR

```c
#include<stdio.h>
int main(){
    int num1 = 15;
    int result1 = num1 << 1;
    int result2 = num1 << 2;
    int result3 = num1 << 3; // 값 커짐
    printf("1칸 이동 결과: %d\n", result1); // 땡기면 2배씩
    printf("2칸 이동 결과: %d\n", result2); // 승수배로 올라감!!
    printf("3칸 이동 결과: %d\n", result3);
    return 0;
}

#include<stdio.h>
int main(){
    int num1 = 15;
    int result1 = num1 >> 1;
    int result2 = num1 >> 2;
    int result3 = num1 >> 3;
    printf("1칸 이동 결과: %d\n", result1); // 승수배로 내려감!!
    printf("2칸 이동 결과: %d\n", result2);
    printf("3칸 이동 결과: %d\n", result3);
    return 0;
}
// 출력: 7 3 1


// Num = Num | (0x01 << i)
```

통신 방법(단방향, 반이중, 전이중)

Communication method (one-way, half-duplex, full-duplex)

- 단방향통신: 한 방향으로만 전송이 가능한 경우로서 수신측에서는 송신측에 대답할 수 없는 경우(라디오, 티비 등)
- 반이중 통신: 양쪽방향으로 신호의 전송이 가능하기는 하나 어떤 순간에는 반드시 한쪽방향으로만 전송이 이루어지는 경우(무전기 등)
- 전이중 통신: 양쪽방향으로 동시에 신호의 전송이 가능한 방식이며 전이중통신은 대체로 정해진 시간에 많은 전량이 수신 되어야 할 때 사용(전화 등)

통신 방법(직렬, 병렬)

Communication method (serial, parallel)

- 이론적으론 병렬이 빠름
- but 효율: 직렬>병렬 (그래서 직렬이 더 많이 쓰임)
- usb->직렬 (시리얼 == 직렬)

통신 방법(동기, 비동기)

Communication method (synchronous, asynchronous)

동기식(Synchronous) 통신: 한 글자가 아니고 미리 정해진 수만큼의 문자열을 한 그룹으로 만들어 일시에 전송

장점: 많은 양의 데이터를 신속히 전송하려 할 때 적합

단점: 비동기 전송보다 더 복잡하고 비용이 많이 들며 송수신사이의 주의 깊은 타이밍이 필요(서로 타이밍 중요!)

비동기식(Asynchronous) 통신: 스타트-스톱 전송이라 하며 한 번에 한 글자씩 전송

장점: 발신자가 편할 때 데이터를 전송

단점: 상대적인 속도가 느림

통신 방법(UART 통신)

Communication method (UART communication)

- UART // 범용 비동기화 송수산기
- 비동기 직렬 통신
- 1 비트씩 전송, 두 시스템 통신 속도 일치 해야함 (예) 9600bp)
- 출력되는 TxD핀은 상대 시스템의 입력 핀인 RxD와 서로 연결
- 참고) USAT

통신 방법(I2C)

Communication method (I2C)

- I2C(Inter-Ingtegrated Circuit)
- 반이중 통신, 동기 통신
- 장점: 선 2개만 있으면 가능
- I2C는 필립스에서 개발한 직렬 버스로 저속의 장치 연결에 사용
- 직렬 데이터 전송을 위한 SDA(Serial Dada Line)와 클럭을 전달하는 SCL(Serial Clock) 두 개의 핀으로 양뱡향 통신 가능 (주소 탑재해서 보냄)

통신 방법(SPI)

Communication method (SPI)

- SPI(Serial Peripheral Interface)
- 전이중 통신, 동기 통신
- 마스터와 슬레이브가 1:1 통신을 할 수 있고, ss 신호가 여러개 제공되는 경우, 1:n의 통신도 가능

통신 방법 비교(UART, SPI, I2C)

Comparison of communication methods (UART, SPI, I2C)

	UART	I2C	SPI
동기/비동기	비동기	동기	동기
관계 수	1:1	N:N	1:N
선 수	2 (TX, RX)	2 (SDA, SCL)	4 (SCK, MISO, MOSI, SS)
이중통신	전이중	반이중	전이중
전송 거리	김	UART 대비 짧음	UART 대비 짧음
전송 속도	느림	SPI 대비 느림	빠름