펭권 스터디 내용05

작성자 : 류성수
작성일자 : 24.07.01

url : https://www.youtube.com/watch?v=h4YXQxKcpwI&t=1s

● 비트연산자

기본적으로 GPIO의 핀이 1바이트 단위였던 것으로 기억한다.
그래서, 각 핀에 HW가 하나씩 물려져 있기 때문에, GPIO를 다룰 때는 비트연산이 필수적이다.
오랜만에 char 형 아스키코드를 하나 받아서, 비트쉬프팅 연산을 이용해서, 문자 하나가 아스키 코드로 어떻게 표현되는지를 살펴보았다.

```c
#include <stdio.h>

void main()
{
        char GPIO;

        printf("Input your ASCII ch : ");
        scanf("%c",&GPIO);

        putchar('\n');

        char printVal = 0;

        printVal = GPIO & (1<<7); // 0x80 & ch 를 하면, MSB만 결과로 출력되게 된다.
        printVal = printVal >> 7; // 그리고 1만을 출력하기 위해,  해당 MSB를 LSB로 내려서 1또는 0으로 만든다.
        printf("%d ",printVal);

        printVal = GPIO & (1<<6);
        printVal = printVal >> 6;
        printf("%d ",printVal);

        printVal = GPIO & (1<<5);
        printVal = printVal >> 5;
        printf("%d ",printVal);

        printVal = GPIO & (1<<4);
        printVal = printVal >> 4;
        printf("%d ",printVal);

        printVal = GPIO & (1<<3);
        printVal = printVal >> 3;
        printf("%d ",printVal);

        printVal = GPIO & (1<<2);
        printVal = printVal >> 2;
        printf("%d ",printVal);

        printVal = GPIO & (1<<1);
        printVal = printVal >> 1;
```

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy05# ./bit_operating.out
Input your ASCII ch : 5

0 0 1 1 0 1 0 1
root@LAPTOP-BQ4AK11N:/project/24-Cstudy05#
```

5라는 문자를 입력하였을 때, 0x35 라는 걸 확인할 수 있다. 십진수로 변환하면, 53이 나온다.
이는 아스키 코드표의 숫자 5의 번호와 일치한다.

이번에는 비트연산자가 임베디드환경에서 GPIO 핀을 제어한다는걸 확인하기 위해, 0x15를 0xfb으로 만들어 보는 실습을 하였다.

먼저, ~(0xfb)을 하면, 0x04가 된다. 일단 0x15를 0x04로 만든 후, 자료형 전체 not 연산을 하여 0xfb를 만들어주었다.

```
char GPIO = 0b00010101; // 해당 GPIO핀을 11111011로 만들어야 한다.

// 아이디어
// ~(11111011) 연산시, 00000100 으로 나오게 된다.
// 그럼, 이 GPIO를 전체NOT 연산을 거친 비트로 만들고, NOT을 걸면 나오게 된다.

GPIO &= 0x0;
GPIO |= (1 << 2);
GPIO ~= GPIO;
```

● 삼항연산자

삼항연산자는 작은 논리 연산자를 한줄로 처리하기 위해 만든 문법이다.
이전에 임베디드 수업때, 해당 기능이 잘 작동하는지를 판단하기 위해, 자주 썼던 것으로 기억한다.

```
#include <stdio.h>
#include <stdbool.h>

bool isRunning()
{
        return true;
}

void main()
{
        (isRunning()) ? printf("is running now\n"):printf("is not running now\n");
}
```

이러한 점을 반영하여, 이러한 문법을 구현해서 복습해 보았다.

```
root@LAPTOP-BQ4AK11N:/project/24-Cstudy05# ./conditional_operator2.out
is running now
```

● 형변환

C/C++이 다른언어보다 어려운 이유는 포인터의 개념도 있겠지만, 제일 많이 실수를 유발하고, 찾기 어려운 버그가 형변환 오류라고 생각된다.
그래서, 리눅스 환경에서 다시 이러한 형변환이 어떻게 되는지를 확인하였다.

```
#include <stdio.h>

int main()
{
        double d = 3.14;
        int i =2;

        printf("%lf\n",(d+i)); // 이런식으로 사용시 의도치 않는 값이 나올 수 있다.

        printf("%lf\n",(double)i +d); // 형변환에 유의할 것.

        int integer = 500;
        printf("%u\n",(unsigned char)integer);

        return 0;
}
```

```
5.140000
5.140000
244
```

결과를 보면, 첫 번째, 두 번째 print 문은 잘 실행이 되었지만, int문을 부호없는 char로 출력하니, 244가 나왔다.
이는 리틀엔디안과 형변환의 합작으로 볼 수 있다.

리틀엔디안은 비트열에서 제일 낮은값을 가리키는 byte가 시작주소에 써지는 메모리 저장구조이다.
여기서 unsigned char로 형변환 하여 가져오니,

500 = 0x000001f4 로 볼 수 있다. 여기서 첫주소에 f4가 써지니, 이것만 취해서 부호없는 char 형으로 출력하면,
f4 = 244가 되어, 이것이 출력되는 것으로 볼 수 있다.