C – HW4

임베디드스쿨1기

Lv1과정

2020. 08. 21

강경수

1. USB_REGFLAG분석

▣ USB_REGFLAG분석

```
typedef union      //익명 유니온을 USBREGFlag로 이름 바꿔 사용
{
        UINT16 ALL;     //유니온은 ALL이라는 unsigned int 변수로 접근 가능.
                        //상단에 #define UINT16 unsigned int 선언되어 있을듯.
                sruct   //익명 구조체 변수 Byte 선언
                {
                        UINT16 Hibyte: 8;           //16bit 를 8bit씩 나누어 사용
                        UINT16 Lowbyte: 8;
                } Byte;

                struct  //익명 구조체 변수 Bit 선언
                {
                        UNIT16 VAUXLow: 1;                  //상위 8bit
                        UNIT16 VAUXoverCUR: 1;
                        UNIT16 CAN5VThShutdown: 1;
                        UNIT16 CAN5VUV: 1;
                        UNIT16 CAN5VOC: 1;
                        UNIT16 VSENSELow: 1;
                        UNIT16 VSUPUV: 1;
                        UNIT16 IDDOcNorm: 1;
```

1. USB_REGFLAG분석

```
UNIT16: 3;                          //하위 8bit
UNIT16 VDDThShutdown: 1;
UNIT16: 1;
UNIT16 RSTLow: 1;
UNIT16 VSUPBatFail: 1;
UNIT16 IDDOcLp: 1;
}Bit;
```

이때 ALL, Byte, Bit의 경우 union이므로 메모리 공간을 공용으로 사용하고
struct 내부의 변수들만 각각 메모리 공간할당

1. USB_REGFLAG분석

즉 위의 주석에 설명해둔것 처럼 위 USBCREGFlag를 사용하기 위해서는

1) USBCREGFlag 변수를 선언하여야함.
2) union 내부의 경우 세가지 방법으로 접근 할 수 있음
3) union안의 unsigned int ALL 변수 접근
4) union 안의 struct 변수 Byte내 변수 접근
5) union 안의 struct 변수 Bit내 변수 접근

```
int main(void)
{
    USBCREGFlag my_usb_register;

    my_usb_register.ALL= 0x0000; //Hibyte LowByte 모두 0으로 초기화

    my_usb_register.Bit.CAN5VThShutdown = 1; //Bit내 변수로 접근
    my_usb_register.Bit.VDDThShutdown = 0;

    my_usb_register.Byte.Hibyte = 0x00;      //Byte내 변수로 접근
    my_usb_register.Byte.LoByte = 0xFF;



    return 0;
}
```

2. C언어 기초 및 변수

▣ C언어 기초 및 변수

1. Compiler
   1) 기계어와 인간이 이해할 수 있는 C언어를 서로 변환해주는 것.
   2) 기계어는 10101001010...과 같이 사람이 직관적으로 이해하기 힘듦.
   3) 컴파일러의 종류 다양하다.

(SOURCE CODE가 CPU INSTRUCTION을 제어하는 과정)

   4) CPU INSTRUCTION란 각 CPU가 사용하는 기계어를 이야기한다.
      CPU INSTRUCTION는 제조사별로 다르다.
      하지만 어느정도는 ISA라는 공통규격을 갖는다. (Instruction Set Architecture)
   5) 어셈블리어란 이 ISA 를 인간이 이해가능한 언어로 만든것이다.
      어셈블리어는 제조사별로 다르기 때문에 모두 배우기 힘들다.

2. C언어 기초 및 변수

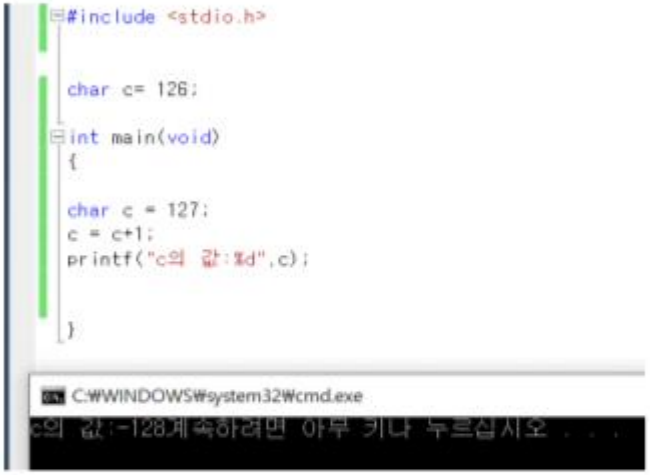
2. Microprocessor VS Microcontroller

1) 마이크로 프로세서는 범용적이며 (PC 등) 마이크로 컨트롤러는 기능이 정해져있다.
(세탁기,비데,청소기 등)

2) 마이크로 프로세서는 RAM ROM등 주변장치를 사용자가 선정할 수 있다.

3. 아래 소스코드 결과가 128이 아닌이유?

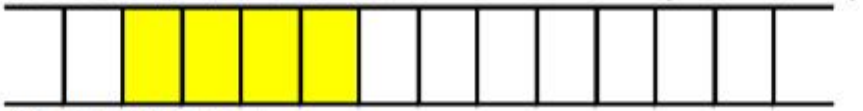
```
#include <stdio.h>

int main(void)
{
    char c= 127;
    c = c+1;

    printf("%d₩n",c)
    return 0;
}
```



위와같은 코드의 결과값은 -128이 나온다.
이를 명확하게 설명하려면 2의 보수에 대해서 알아야한다. →6번 목차로 이동.

4. 변수(Variable)

컴퓨터가 데이터를 기억하는 방법
RAM이라는 공간에 이를 기록한다. (RAM 메모리)



한 칸 한칸은 1byte
0x12345678 0x1234567B

2. C언어 기초 2 연산자

▣ C언어 기초2

1. Operators
- 컴퓨터에게 특정한 수학 or 논리기능을 수행하도록 지시

1) Arithmetic operator

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

2. C언어 기초 2 연산자

2) Relational Operator

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.

3) Logical Operator

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

2. C언어 기초2 연산자

4) Assign Operator

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

2. C언어 기초2 연산자

5) Bitwise Operator

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

2. C언어 기초2 연산자

6) 연산자 우선순위

연산에는 우선순위가 존재한다. 따라서 연산자 사용시순위를 명확하게 알고 사용

Left to right 와 같은 방향은 연산이 어느방향으로 진행되는지 나타내기 위함.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

2. C언어 기초3 음수 표현법

▣ C언어가 음수를 표현하는 법

-. 2의 보수

C언어에서 음수를 나타내는방법은 최상단비트가 1일 경우 음수 0일 경우 양수이다.
이때 이렇게 표현하면 두가지 문제점이 발생한다.
1) 0의 경우 '0000' '1000' 두가지 표현방법이 생긴다
2) 덧셈이 복잡해진다. EX) 0001 + 1001 을 할경우 덧셈이 아닌 뺄셈을 해야함.

위 해결 방법이 2의 보수 표현법이다.
2의 보수 표현법은 간단함. -3을 만들고 싶으면
1) 0011 에서 비트를 반전시킴 → 1100
2) 이값에 1을 더해줌. 그러므로 결과는 1101

위와같이 2의 보수를 취하면 0의 경우 +0 이던 -0이던 0000이며
뺄셈도 덧셈으로 해결가능함.
EX) 0000 부호 반전 1111 → 1111+1 = (1)0000
5 + (-3) = 0101 + 1101 = (1)0010

2. C언어 기초4 조건문

▣ 조건문

- if (조건이 참이면 실행
- else(조건 만족하지 않을때 실행
- else if (else if라는 문법이 따로 있는게 아니라. else+ if 를 연이어 사용한것)

```
#include <stdio.h>

int main(void)
{
    int score = 0;

    scanf("%d",&score);

    if(score > 90)
    {
        printf("최고우등생입니다");
    }
    else if(score>80)
    {
        printf("상위권입니다");
    }
    else if(score>70)
    {
        printf("중위권입니다");
    }
    else
    {
        printf("더 열심히하세요");
    }

    return 0;
}
```

2. C언어 기초4 조건문

-switch(case)
-변수의 수가 적을때 사용
- if문보다 간결한 특징을 가짐

```
int main(void)
{
    int num = 0;
    scanf("%d",num);
    switch(num)
    {
        case 1: printf("입력이 1입니다");
                break;
        case 2: printf("입력이 2입니다");
                break;
        case 3: printf("입력이 3입니다");
                break;
        default:break;
    }

    return 0;
}
```

- 반드시 각 케이스에 대해서 break; 해줄것!
- 또한 default의 경우가 발생하지 않더라도 적어줄것!

2. C언어 기초4 조건문

-switch(case)
-변수의 수가 적을때 사용
- if문보다 간결한 특징을 가짐

```
int main(void)
{
    int num = 0;
    scanf("%d",num);
    switch(num)
    {
        case 1: printf("입력이 1입니다");
                break;
        case 2: printf("입력이 2입니다");
                break;
        case 3: printf("입력이 3입니다");
                break;
        default:break;
    }

    return 0;
}
```

- 반드시 각 케이스에 대해서 break; 해줄것!
- 또한 default의 경우가 발생하지 않더라도 적어줄것!

2. C언어 기초5 함수

▣ 함수

1. 사용법

2. C언어 기초5 함수

▣ 함수

1. 사용법

2. C언어 기초5 함수

-함수 유형1) 입력 반환 모두 존재하는 경우

```c
int sum(int a,int b); //함수의 프로토 타입 선언 없으면 컴파일 오류남

int main(void)
{
    sum(1,5);
    return 0;
}

int sum(int a, int b) // 반환데이터타입 함수명 매게변수
{                     // 중괄호 내 함수 시작
    int c = 0;
    c = a + b;
    return c;         // c값 반환
}
```

-함수 유형2) 반환이 void

```c
void sum_plus1(int *a); //함수의 프로토 타입 선언 없으면 컴파일 오류남

int main(void)
{
    int a = 3;
    sum_plus1(&a);
    printf("%d₩n",a);
    return 0;
}

void sum_plus1(int *num) // 반환되는 값 x 매게변수만 존재
{
    *num += 1;
}
```

2. C언어 기초5 함수

-함수 유형3) 입력이 void

```
int error_function(); //함수의 프로토 타입 선언 없으면 컴파일 오류남

int main(void)
{
    if(error_function() == 1)
    {
        printf("함수가 error값을 반환하였습니다!₩n");
    }
    return 0;
}

int error_function()// 입력 x
{
    printf("error 발생₩n");
    return 1;
}
```

-함수 유형4) 입력반환 모두 void

```
int a = 0x0001;
int b = 0x0011;

void LCD_initalize(); //함수의 프로토 타입 선언 없으면 컴파일 오류남

int main(void)
{
     LCD_initalize();
    //이하 LCD구동문
    return 0;
}

void LCD_initalize()// 입력 x
{
    a |= (1<<3);
    b ^= ~b;1;
}
```

2. C언어 기초6 배열

▣ 배열

1) 배열이란 무엇인가?

변수를 각각 선언하지 않고 편리하게 관리 할 수 있다.

```
#include <stdio.h>

int main(void)
{
    int arr1[] = {1,2,3}; //대괄호 내에 변수 개수 생략
    int arr2[3] = {1,2,3}; // 대괄호 내에 변수 개수 선언
    int arr3[3] = {0,0,0}; // 모든 변수 0으로 대입
    int arr4[3] = {0,};    // 위와같이 모든 변수가 0으로 대입된다.

    printf("%d₩n",arr1[1]);  //변수 출력시 arr1[0],arr1[1],arr[2] 순으로 저장된다.
                              //선언은 [3]으로 하고 접근은 n-1로 하니 주의할것!
    return 0;

}
```

2) 2차원 배열

```
int arr1[2][3] = {{1,2,3},{4,5,6}}; //2차원 배열 선언 [행][열]이 된다.
int arr2[2][3] = {1,2,3,4,5,6}; //옆과 같이 선언해도 위와 같다.

printf("%d₩n",arr1[1][1]);

return 0;
```

3) 배열의 크기

int 형 배열의 경우 int형 변수가 연속적으로 저장되어 있음.

char 형 배열의 경우 char형 변수가 연속적으로 저장되어 있음.

따라서 int형 int arr[2], char arr_char[3] 은 각각 4byte x 2 = 8byte, 1byte x 3 = 3byte

2. C언어 기초7 포인터

▣ Pointer

1) 포인터란 무엇인가?

포인터는 단순히 주소값을 저장하는 변수

int가 4byte char가 1byte 저장공간을 갖는것처럼 포인터 변수는 4byte의 크기를 갖는다.

왜 4byte인가? 32bit 기준 표현할 수 있는 주소의 가짓수가 2^32개이기 때문

2) 포인터 사용법

int a = 3;

위와 같이 int *a = 3; 변수명 뒤에 *을 붙여주면 주소값을 저장 할 수 있는 포인터 변수임

이때 (int*) a =3; 즉 int와*을 함께 묶어 int형변수를 가르키는 주소값

이라고 생각하면 다른 내용을 이해하기 편하다.

3) 포인터와 함수

```
void sum_plus1(int *a); //함수의 프로토 타입 선언 없으면 컴파일 오류남

int main(void)
{
    int a = 3;
    sum_plus1(&a);
    printf("%d₩n",a);
    return 0;
}

void sum_plus1(int *num) // 반환되는 값 x 매게변수만 존재
{
    *num += 1;
}
```

포인터 변수에는 메모리 주소를 저장할 수 있다. 따라서 함수에서 포인터 변수를

매게변수로 사용시 main변수를 직접 컨트롤 할 수 있다.

2. C언어 기초7 포인터

4) 포인터와 2차원배열

```
#include <stdio.h>

int main(void)
{
    int arr[2][3]={1,2,3,4,5,6};
    printf("%p₩n",arr); //배열의 시작주소
    printf("%p₩n",&arr); //2차원 배열 전체
    printf("%p₩n",(&arr)+1); //24byte차이 4*6 즉 이때 &arr은 [2X3]행렬 전체를 말함.*
    printf("%p₩n",(&arr+1));
    printf("%p₩n",*(arr));
    printf("%p₩n",*(arr)+1); //4byte 즉 int형 변수 1개 차이
    printf("%p₩n",*(arr)+2); //8byte차이 즉 int형 변수 2개 차이
    printf("%p₩n",*((arr)+1)); //12BYTE차이
    printf("%p₩n",arr+1); //12BYTE차이
    printf("%d₩n",**arr); //배열내의 변수

    return 0;
}
```

즉 arr그 자체는 3열의 크기를 갖는 배열

&arr 2x3행렬 그 자체

*arr 2x3 행렬 하나하나 라고 생각 할 수 있다.

2. C언어 기초8 구조체

▣ Structer

1) 구조체란 무엇인가?

다양한 데이터 타입을 하나로 묶을 수 있음

2) 구조체 사용법

```
typedef struct DATA
{
    char name[50];
    char address[50];
    int age;
    int birthday;
}data;
```

혹은

```
struct DATA
{
    char name[50];
    char address[50];
    int age;
    int birthday;
};
```

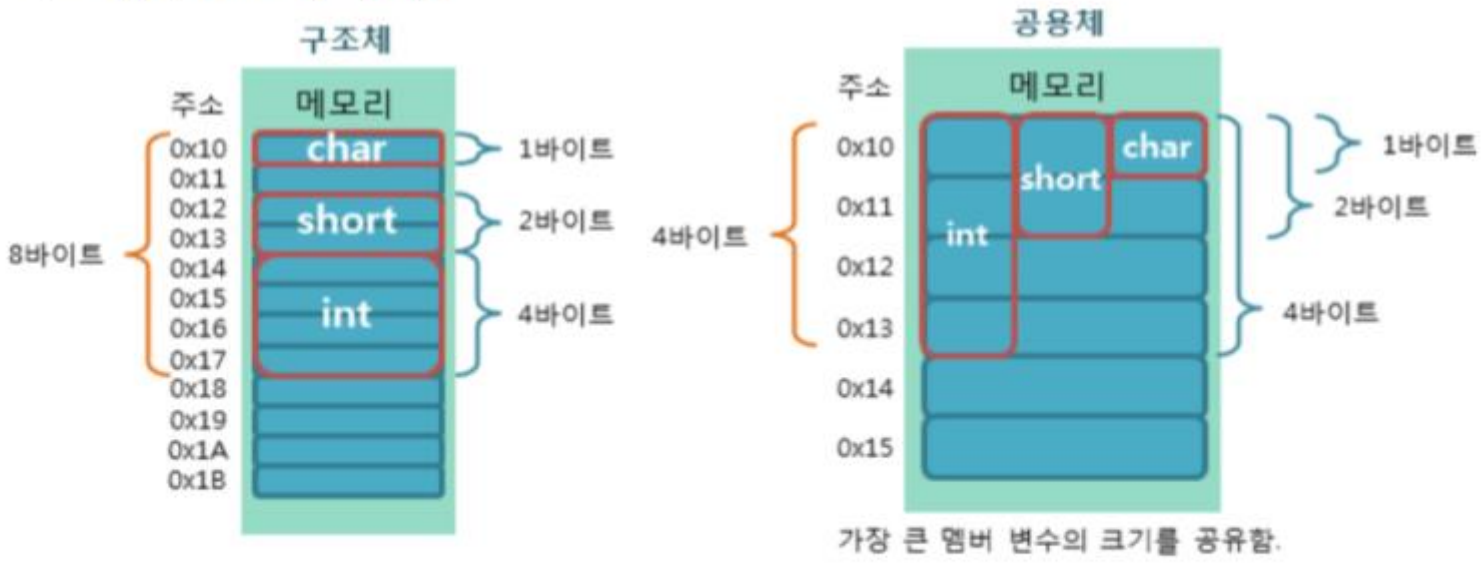
```
#include <stdio.h>
#include <string.h>

typedef struct DATA
{
    char name[50];
    char address[50];
    int age;
    int birthday;
}data;
```

2. C언어 기초8 구조체

```c
int main(void)
{
    struct DATA man1; //이렇게 선언해도 되고
    data book2; //typdef를 사용하여 이렇게 선언하여도됨.
    strcpy(man1.name,"김민수"); //구조체 변수 사용시 온점.을 사용

    return 0;
}
```

3) Union

- 구조체와 Union의 차이점



유니온의 경우 메로리를 공유한다! 절대 잊지말것!

2. C언어 기초8 구조체

```c
typedef union{  //익명 유니온을 사용하여 별도의 이름을 지정하지 않음
    float a;
    int b;
    double c;
} my_union;


int main(void)
{
    my_union my_num;

    my_num.a = 1.3;
    my_num.b = 5;
    my_num.c = 5.27;

    printf("%d₩n",my_num.b);

}
```

이때의 출력값은 5가아닌 이상한값이 나오게 된다.
이유는 유니온의 경우 공용으로 메모리 공간을 사용하기에 my_num.c기준으로 5.27이 저장되어 있기 때문이다.

2. C언어 기초8 구조체

4) 구조체 선언 방법 차이점

```
struct _INF1            //정석적인 구조체 정의
{
    int age;
    int name;
}

typedef struct _INF02   //struct _INF02 -> info2로 대체함
{
    int age;
    int name;
} info2;

typedef struct          //struct (익명구조체) -> info3로 대체함
{
    int age;
    int name;
} info3;

struct _INF04           //struct _INF04변수 info4; 정의
{
    int age;
    int name;
} info04;
struct                  //struct (익명구조체) 변수 info5; 정의;
{
    int age;
    int name;
} info5;
```