Activity 1. 2진수, 10진수, 16진수의 관계

1. 8비트 짜리 메모리를 각 비트로 표현하면 다음과 같이 8개의 자리로 표현할 수 있다. 각 자리는 0이나 1로 채울 수 있다. 즉, 2진수로 표현된다. 또한 4개 비트씩 모아 16진수로 표현이 가능하다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7번 비트  (MSB) | 6 번 비트 | 5 번 비트 | 4 번 비트 | 3 번 비트 | 2 번 비트 | 1 번 비트 | 0 번 비트  (LSB) |
| 0~1 | 0~1 | 0~1 | 0~1 | 0~1 | 0~1 | 0~1 | 0~1 |
| 0~F | | | | 0~F | | | |

다음 메모리에 채워진 값을 보고 16진수 자리에 맞는 값을 채우시오.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 비트 # | 7번 비트  (MSB) | 6 번 비트 | 5 번 비트 | 4 번 비트 | 3 번 비트 | 2 번 비트 | 1 번 비트 | 0 번 비트  (LSB) |
| 2진수 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 16진수 | 0xD **(B)** | | | | 0x9 **(9)** | | | |

1. 다음 10진수의 값들을 2진수로 채우시오. 자리를 넘어가면(overflow) 무시하시오.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10진수 |  | 2진수 | | | |
| 3번 비트 | 2번 비트 | 1번 비트 | 0번 비트 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 0 | 0 |
| 11 | 1 | 0 | 0 | 1 |
| 12 | 1 | 0 | 1 | 0 |
| 13 | 1 | 0 | 1 | 1 |
| 14 | 1 | 1 | 0 | 0 |
| 15 | 1 | 1 | 0 | 1 |
| 16 | 1 | 1 | 1 | 0 |
| 17 | 1 | 1 | 1 | 1 |
|  |  |  |  |  |  |

1. 다음 10진수의 값들을 16진수 값으로 채우시오

|  |  |  |
| --- | --- | --- |
| 10진수 | 16진수는 ‘0x’ 접두사를 붙여서 표현하며, ‘0x’ 뒤에 0~9 까지의 숫자와 A~F 까지의 알파벳으로 구성됨 | 16진수 |
| 0 | 0x0 |
| 1 | 0x1 |
| 2 | 0x2 |
| 3 | 0x3 |
| 4 | 0x4 |
| 5 | 0x5 |
| 6 | 0x6 |
| 7 | 0x7 |
| 8 | 0x8 |
| 9 | 0x9 |
| 10 | 0xA |
| 11 | 0xB |
| 12 | 0xC |
| 13 | 0xD |
| 14 | 0xE |
| 15 | 0xF |
| 16 | 0x10 |
| 17 | 0x11 |
| 18 | 0x12 |

1. 다음 2진수를 16진수로 채우시오.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 16진수 |  | 2진수 | | | |
| 3번 비트 | 2번 비트 | 1번 비트 | 30 비트 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| A | 1 | 0 | 1 | 0 |
| B | 1 | 0 | 1 | 1 |
| C | 1 | 1 | 0 | 0 |
| D | 1 | 1 | 0 | 1 |
| E | 1 | 1 | 1 | 0 |
| F | 1 | 1 | 1 | 1 |

Activity 2. C언어 비트 연산자 연습문제

1. 다음 빈칸을 채우시오

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 연산 종류 | AND | OR | XOR | NOT | 오른쪽으로 쉬프트 | 왼쪽으로 쉬프트 |
| C언어 기호 | && | **||** | **^** | **!** | >> | << |

1. 다음은 32비트 메모리의 비트를 채울 수 있는 표이다. 예를 보고 빈칸들을 채우시오.

예) 0xBAB0BAB0

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 16진수 | **B** | | | | A | | | | B | | | | 0 | | | | B | | | | A | | | | B | | | | 0 | | | | |
| 2진수 | **1** | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

* 0xFFFFFFFF (32비트로 구성된 정수 상수로 모든 비트가 1로 설정된 값)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| F | | | | F | | | | F | | | | F | | | | F | | | | F | | | | F | | | | F | | | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

* 0x12345678 (32비트로 구성된 정수 상수로, 16진수로 표현되는 값)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 |  | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | | | | 2 | | | | 3 | | | | 4 | | | |  | 5 | | | | 6 | | | | 7 | | | | 8 | | | | |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |  | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

* 0x12345678 & 0xFFFFFFFF =

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | | | | 2 | | | | 3 | | | | 4 | | | | 5 | | | | 6 | | | | 7 | | | | 8 | | | | |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| F | | | | F | | | | F | | | | F | | | | F | | | | F | | | | F | | | | F | | | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | | | | 2 | | | | 3 | | | | 4 | | | | 5 | | | | 6 | | | | 7 | | | | 8 | | | | |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

1. 다음 문제를 푸시오.

* 0xA & 0xFF =

10 & 255 (10진수), 00001010 & 11111111 (2진수), **결과는 10(16진수: 0x0A)?**

0XA,0XFF 는 16진수 표기법

0Xa는 10진수로 10, 0Xff는 10진수로 255

0xA 의 2진수 00001010, 0Xff 의 2진수 11111111

* 0xA && 0xFF = 참(true)

0XA,0XFF 는 16진수 표기법

0Xa는 10진수로 10, 0Xff는 255

0xA 의 2진수 00001010, 0Xff 의 2진수 11111111

C언어에서 ‘&&’는 논리 AND 연산자이며, ‘&’는 비트 AND연산자 다.

\*비트 AND연산은 숫자의 각 비트들을 비교하여 비트 단위로 AND 연산을 수행

\*논리 AND연산은 논리적인 참과 거짓을 판단하여 참과 거짓을 반환 수행

&& 은 두 개의 피연산자가 모두 참(true)일 때 결과가 참이 된다. ‘0xA’ 와 ‘0Xff’는 둘 다 10진수 이므로 결과는 참(true)이다.

* 0xAA & 0xFF = 170 & 255 (10진수), 10101010 & 11111111 (2진수), **결과는 170(16진수: 0xAA)**

0xAA & 0xFF 는 16진수

10진수로 0xAA 는 170, 0xFF 는 255

2진수로 0xAA 는 10101010, 0xFF 11111111

* 0xAA && 0xFF = 참(true)
* 0xA | 0xFF = 10 | 255 (10진수), 00001010 | 11111111 (2진수), 결과는 255(16진수: 0xFF)

C언어에서 ‘|’ 는 비트 OR연산자 다.

* 0xA || 0xFF = 10 || 255 (10진수), 00001010 || 11111111 (2진수), **결과는 참**

C언어에서 ‘||’ 는 논리 OR연산자 다.

‘||’ 은 두 개의 피연산자 중 하나 이상이 참(true)일 때 결과가 참이 된다.

‘||’ 은 논리적인 참과 거짓을 판단하여 참과 거짓을 반환하며, ‘| (비트OR연산)’ 과는 개념이 다름. ‘|’ 은 숫자의 각 비트들을 비교하여 비트 단위로 OR연산을 수행하는 반면, ‘||’ 은 논리적인 참과 거짓을 판단.

* 0xA ^ 0xFF = 10 ^ 255 (10진수), 00001010 ^ 11111111 (2진수), **결과는 245 (10진수), 11110101 (2진수)**

C언어에서 ‘^’ 는 XOR연산자 다.

* ~0xA = 10진수 5, 2진수 0101

C언어에서 ‘~’ 는 NOT(반전) 연산자 다. 피연산자의 각 비트를 반전시킨다.

0Xa = 10진수 10 = 2진수 1010

~0xA = ~1010 = 0101 의 10진수는 5

* !(0xA) = 참(true)

C언어에서 ‘!’ 는 논리 NOT(논리 부정) 연산자로, 피연산자가 참(true) 이면 거짓(false) 로, 거짓이면 참으로 바꾼다.

* 0xA << 1 = 20 (10진수)

C언어에서 ‘<<’ 연산자는 비트 왼쪽 쉬프트 연산자로, 피연산자의 비트들을 왼쪽으로 지정된 비트 수만큼 이동시킨다.

0Xa (16진수), 10 (10진수), 1010 (2진수)

0Xa 를 왼쪽으로 1비트 쉬프트한다. 0Xa << 1

1010의 모든 비트를 왼쪽으로 1칸씩 이동하면 다음과 같다. 1010 << 1 = 10100

10100 의 결과는 20 (10진수)

* 0xA >> 1 = 5 (10진수)

C언어에서 ‘>>’ 연산자는 비트 오른쪽 쉬프트 연산자로, 피연산자의 비트들을 오른쪽으로 지정된 비트 수만큼 이동시킨다.

1010 (2진수) >> 0101

0101 은 5 (10진수)

* 0x1010 & 0xFF = 16 (10진수)

0x1010 (16진수) = 4112 (10진수) = 0001000000010000 (2진수)

0Xff (16진수) = 255 (10진수) = 11111111 (2진수

* 0b1010 & 0xFF = 10 (10진수) = 00001010 (2진수)

0b1010 (2진수) = 10 (10진수)

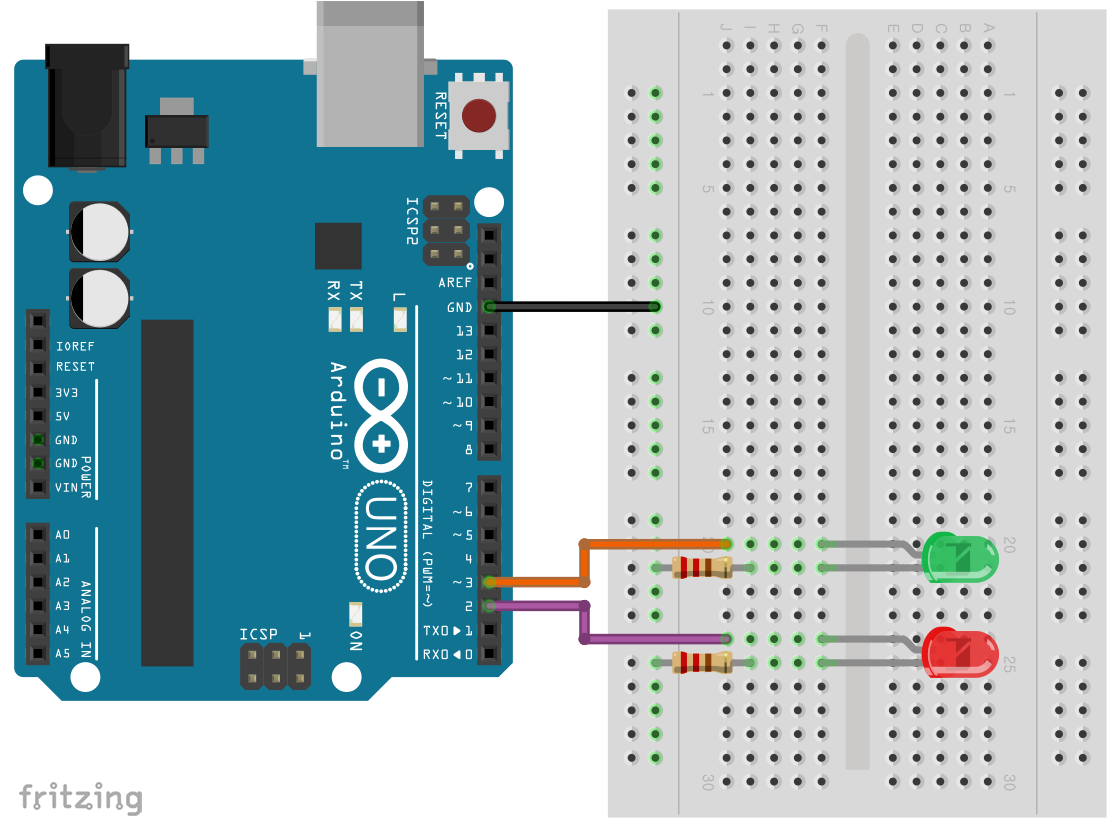
0xFF (16진수) = 255 (10진수)

0b1010 & 0Xff = 10 & 255

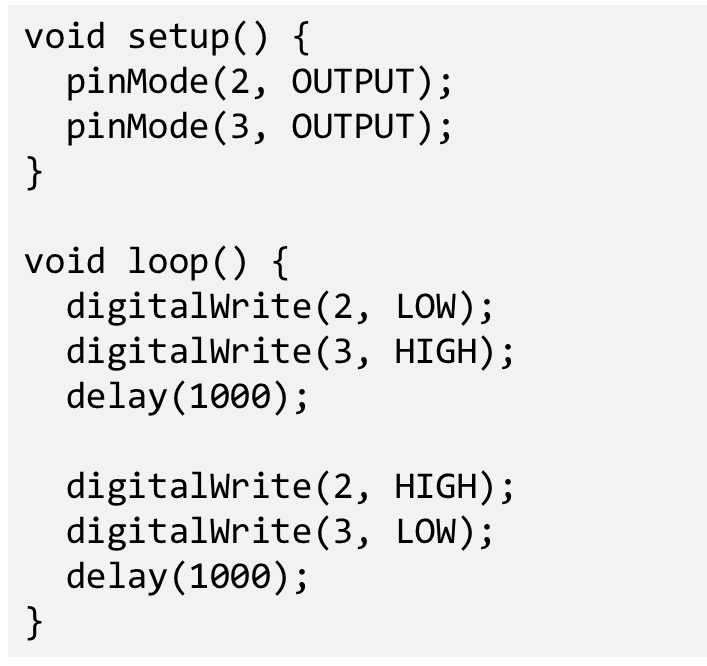
Activity 3.

임베디드 시스템에서 bit operation이 많이 사용되는 이유중의 하나는 CPU의 레지스터들을 bit별별 제어하기 위함이다.

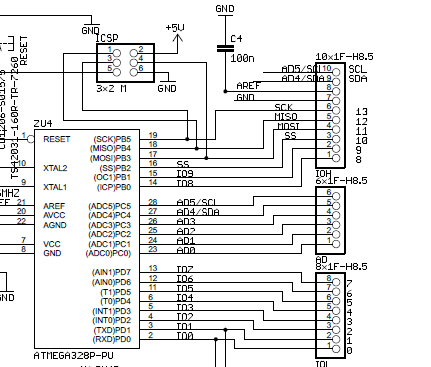
예를 들어 아두이노의 디지털 핀 2,3번의 LED를 on/off하는 예를 들어보자.



다음과 같이 pinMode라는 함수로 2,3번핀을 output으로 설정하고 digitalWrite라는 함수로 on/off한다.



여기서 아두이노의 회로도를 살펴보면 ATMEGA238 CPU의 PD2, PD3번 핀이 디지털 핀 2,3번과 연결되어 있음을 알 수 있다.



PD2,3번을 output으로 설정하는 레지스터는 ATMEGA238 CPU 내부에 있는 DDRD라는 8비트 레지스터이고 on/off를 설정하는 레지스터는 PORTD라는 8비트 레지스터이다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | | | |  | | | |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

기본적으로 모든 비트가 0이고 0이면 input이다. PD2번핀을 output으로 하려면 2번비트를 1로 서설정하고 PD3번핀도 마찬가지이다.

C언어로 표현하면 DDRD = 0x0C;

PD2번핀을 on하려면 PORTD의 2번 비트를 1, off하려면 그 비트를 0으로 하면 됨. PBD번 핀도 마찬가지임.

PORTD

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | | | |  | | | |
|  |  |  |  | 1 | 1 |  |  |

C언어로 표현하면 PORTD = 0x0C;

PD2,3번핀을 모두 off하려면 PORTD = 0x00;

이와 같은 코드를 좀더 알아보기 쉽고 세련되게 표현하기 위해 #define를 많이 이용함.

교재 제 15장 참조.

#define PI 3.141592

#define AND &&

와 같이 단순 상수나 대체용도로 쓸수도 있으나 다음과 같이 함수 매크로로 사용할 수도 있다.

<https://kartikmohta.com/tech/avr/tutorial/codes/port_test.c>

#define \_BV(bit) (1 << (bit))

#define sbi(x, y) x |= \_BV(y) //set bit - using bitwise OR operator

#define cbi(x,y) x &= ~(\_BV(y)) //clear bit - using bitwise AND operator

#define tbi(x,y) x ^= \_BV(y) //toggle bit - using bitwise XOR operator

#define is\_high(x,y) (x & \_BV(y) == \_BV(y)) //check if the y'th bit of register 'x' is high ... test if its AND with 1 is 1

&%^

1. 다음 수식의 값을 채우시오.

* (1 << 2)
* 비트 시프트 연산을 수행, 1을 왼쪽으로 2비트 이동.
* 0001 을 0100 으로 이동해서 결과는 4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | | | | 4 | | | |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

unsigned char port;

port = 0xAA;

sbi(port,2);

변수 port의 결과값을 적으시오

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | | | |  | | | |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |

**Sbi 는 ‘Set Bit In’ 의 약어로 특정 비트를 1로 설정하기 위한 매크로.**

Port 변수의 2번째 비트를 설정, ‘0Xaa’ 의 이진표현은 ‘10101010’

2번째 비트는 0부터 세어보면 ‘010’ 이며 이 비트를 1로 설정하면 결과적으로 이진 표현은 ‘10101110’ 이 되고 이 값을 16진수로 표현하면 ‘0Xae’ 가 된다.

unsigned char port;

port = 0xAA;

cbi(port,1);

변수 port의 결과값을 적으시오

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | | | |  | | | |
| **1** | **0** | **1** | **0** | **1** | **0** | **0** | **0** |

**Cbi 는 “Clear Bit In” 의 약어로 특정 비트를 0으로 설정하기 위한 매크로이다.**

Port 의 1번째 비트를 0으로 클리어하는 코드인 ‘cbi (port, 1)’ 을 사용하면 해당 비트가 1로 설정된다.

0Xaa(16진수) = 10101010 (2진수)

10101000(2진수) = 0xA8(16진수) = 168(10진수)

**결과 값은 0xA8 이다.**

unsigned char port;

port = 0xAA;

tbi(port,1); 🡪 가

tbi(port,1); 🡪 나

1. 순간의 변수 port의 결과값을 적으시오

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | | | |  | | | |
| **1** | **0** | **1** | **0** | **1** | **0** | **0** | **0** |

1. 순간의 변수 port의 결과값을 적으시오

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | | | |  | | | |
| **1** | **0** | **1** | **0** | **1** | **0** | **1** | **0** |

**Tbi 는 “Toggle Bit In” 의 약어로 특정 비트를 토글(반전) 시키기 위한 매크로이다.**

주어진 변수의 특정 비트를 반전시킴으로써 1이면 0으로, 0 이면 1로 변경된다.

0xAA 이진 표현은 10101010

**가 : 10101000 (2진수), 0xA8 (16진수)**

가의 10101000 을 다시 반전시키면

**나 : 10101010 (2진수), 0xAA (16진수)**

ATMEGA328 CPU내부의 DDRD, PORTD레지스터들은 앞의 예제와 같은 변수 port와 같은 방법으로 셋팅이 가능하다.