C언어 멘토링 3주차

데이터 표현방식

0x MSB 0101 >> & | ^ << ~ /2

9/17일 목 작성자 : 이승준 inspired by MONDRIAN, BAUHAUS

컴퓨터가 데이터를
표현하는 방식

10101010011

11010100011

00010101010

01010101001

10010101000

10101010011

2진수, 8진수, 10진수, 16진수

- N진수 : N개의 부호를 이용하여 값을 표현하는 방식
- 2진수 : 2개(0,1)의 부호를 이용해서 값을 표현하는 방식
- 8진수 : 8개(0~7)의 부호를 이용해서 값을 표현하는 방식
- 10진수 : 10개(0~9)의 부호를 이용해서 값을 표현하는 방식
- 16진수 : 16개(0~9,A,B,C,D,E,F)의 부호를 이용해서 값을 표현하는 방식

자릿수 증가에 따른 각 진수의 표기 방법

직접 작성해 보면서 규칙성 파악해보기

- 과제 1) 직접 계산해보기
  (word 문서 참조)
- 10진수 127, 245, 178를
  각 진수로 나타내기
- 16진수 C5는 각 다른 진수 별로 얼마인가
- 2진수 1000110011은 각 진수 별로 얼마인가

2진수	8진수	10진수	16진수
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10

진수 변환법

10 <-> (2, 8, 16)　　(시험문제에서 나올수 있다. 실상에서는 무쓸모)

- 10진수를 2진수로 변환 (10진수에서 8,16진수는 2로 나눌걸 8,16으로 나누거나, 10=>2=>8or16 순서가 빠르다.)
  1. N의 몫이 0이 될때 까지 2로 계속 나눈다. (몫, 나머지)
  2. 정리된 나머지의 역순이 2진수

- ex) N = 87
  87/2 => (43,1) ‘c언어에서 / (나누기) 연산을 생각하면 됩니다.’
  43/2 => (21,1)
  21/2 => (10,1)
  10/2 => (5,0)
  5/2 => (2,1)
  2/2 => (1,0)
  1/2 => (0,1) STOP! => 1010111　　결론 : 87 = 1010111

- 2진수를 10진수로 변환
  1. 각각 2진수의 자릿수를 10진수로 변환하고 1이 있는 자리의 모든 값을 더한다.
  2진수의 k번째 자리수는 2^(k-1) 이다.

진수 변환법

2 <-> (8, 16)

- 2진수를 3or4칸 씩 첫째 자리부터 끊어주고 각 칸을 8or16진수로 바꿔준다.
- ex) 1010010111111
  2 to 8 : 101 001 011 111 => 6137
  2 to 16 : 1010 0101 1111 => A5F
- 8 or 16을 2진수로 바꾸려면 각 자리를 3,4자리의 2진수로 바꿔준다.
- ex) 1010010111111
  8 to 2 : 6137 => 101 001 011 111
  16 to 2 : A5F => 1010 0101 1111

C 내에서 8,16 진수 표현해보기 CODE

변수를 초기화 해줄 때 결정된다.

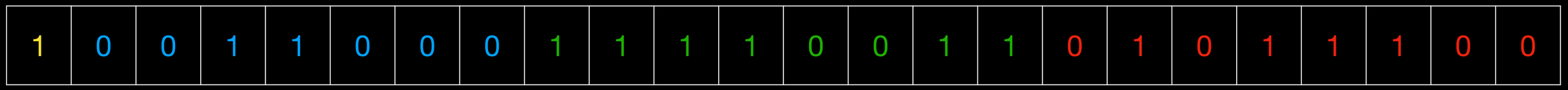
```c
#include <stdio.h>

int main (void){

    int num1 = 010;
    //정수 앞에 0을 붙이면 8진수이다.
    int num2 = 0xA1;
    //정수 앞에 0x를 붙이면 16진수 이다.
    printf("num1 is %d\n",num1);
    printf("num1 is %d\n",num2);
    //8, 16 진수가 10진수로 변환되어 출력됨을 볼 수 있다.

    return 0;
}
```

```
num1 is 8
num1 is 161
```

데이터의 표현 단위 비트와 바이트

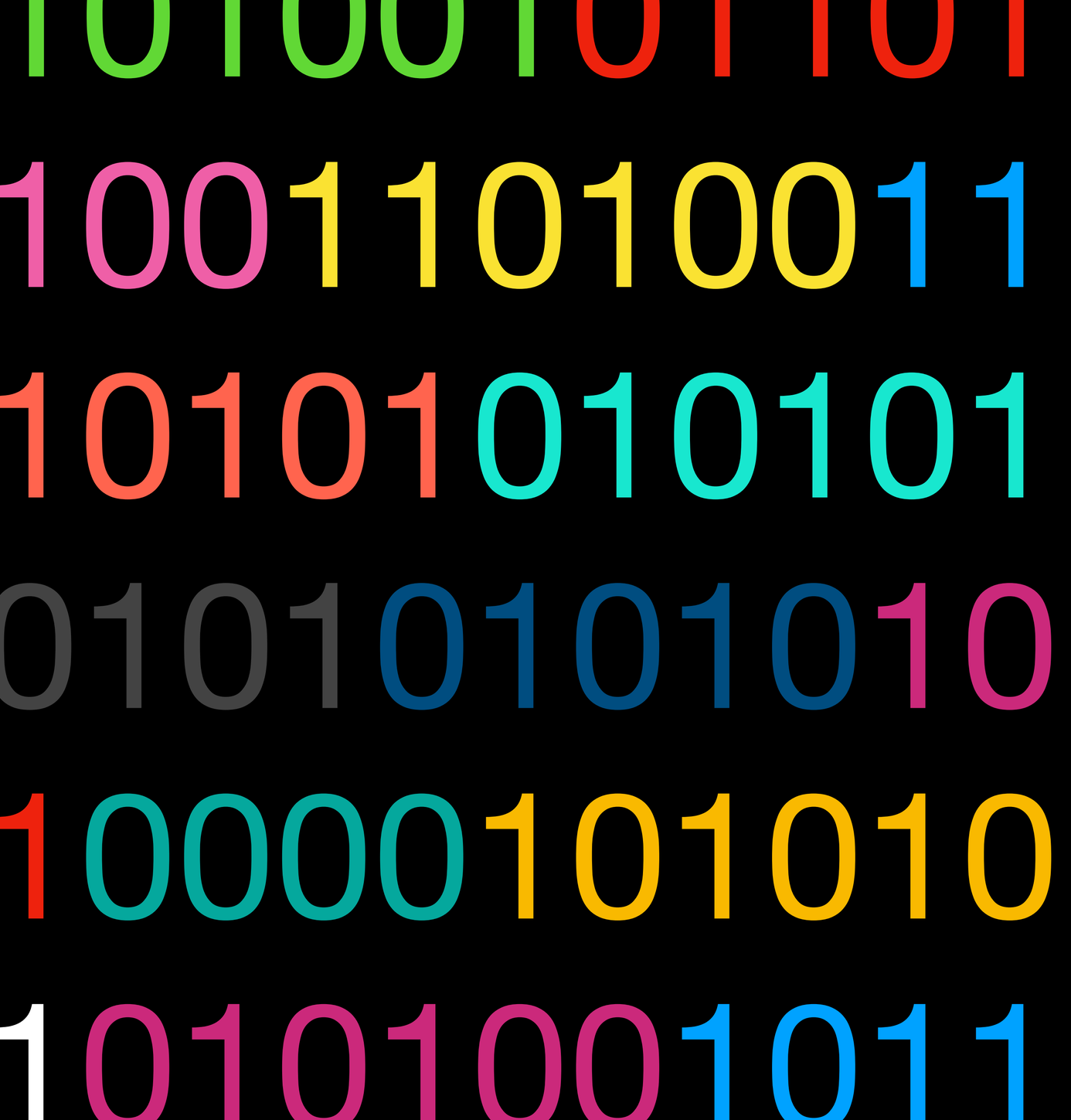
1 bit

- 1 byte(= 8 bits)



- 2 bytes (= 16bits)

정수와 실수의 표현방식


10100101011
10011010011
10101010101
01010101010
10000101010
10101001011

정수형 데이터의 표현(저장)방식

MSB 에 위치한 공간에 SIGN(부호) 을 의미하는

MSB	0	1	1	0	1	0	1

부호
0 : + 1 : -

숫자의 크기

0	0	1	1	0	1	0	1

0	0	0	1	1	1	0	0

- 10000001 + 00000001 = 1000010
  -1 + 1 = -2 라는 납득 불가능한 결과가 나온다
- 따라서 10000001 이 -1은 아니다!
- 1의 보수, 2의 보수라는
  새로운 방법으로 음의 정수를 표현한다

보수법

1의 보수 , 2의 보수

정수형에서 음수를 표현할 때는 2의 보수법을 사용한다.
1의 보수법과 2의 보수법에 대해 알아보자

1의 보수법 : (0과 1 반전)
    0001 의 부호반대수는 1110으로 0과 1 바꾸기
    1000 의 부호반대수는 0111으로 0과 1 바꾸기
2의 보수법 : (0과 1 반전 후 +1)
    0001을 정반대 (1110 +1) = 1111이 부호반대수
    1001을 (0110 + 1) = 0111

<1의 보수와 2의 보수의 차이점>

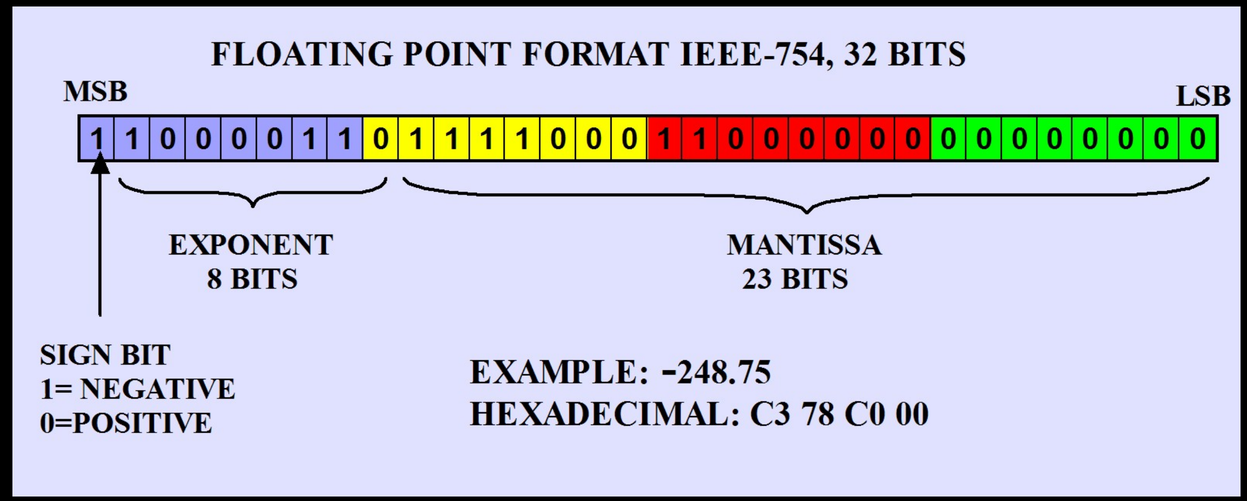
• 1의 보수법에서의 0	○ 2의 보수법에서의 0
• 11111111	○ 00000000

실수 표현방식

부동 소수점 표현 방식

MSB : 부호부
EXPONENT : 지수부
MANTISSA : 가수부

정밀도 < 표현가능 범위
이 때문에 완벽하게 정밀한 실수를 표현할 수 없다

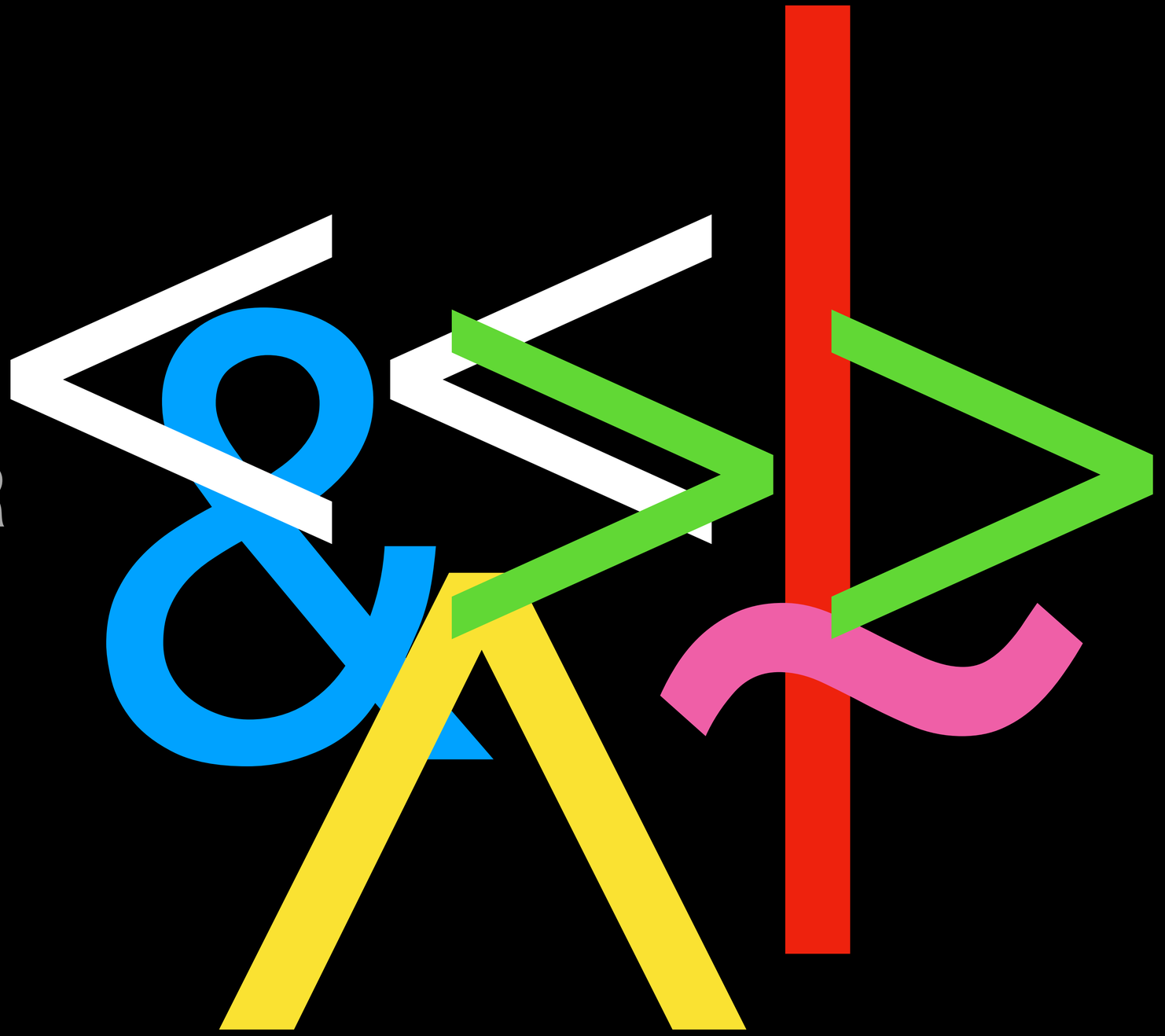


실수 표현의 한계 CODE

```c
#include <stdio.h>

int main (void){

  int i;
  float num = 0.0; //실수는 0으로 저장 되어있지만 실제로는 아니다!
  for (i=0; i< 100; i++){
    num += 0.1;
  }
  printf(" 0.1 을 100번 더한 결과 : %f\n", num);
  return 0;
}

```

```
 0.1 을  100번  더 한  결 과  : 10.000002
```

BIT OPERATOR
SHIFT OPERATOR



BIT OPERATOR

&	|	^	~
1과1일 때만 1	1이 하나라도 있으면 1	같으면 0 다르면 1	반전

BIT OPERATOR

X	Y	X&Y	X|Y	X^Y	~X
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

BIT OPERATOR

	1	0	0	1	1	0	1	0
	1	1	0	1	0	1	1	0
&	1	0	0	1	0	0	1	0
|	1	1	0	1	1	1	1	0
^	0	1	0	0	1	1	0	0
~	0	1	1	0	0	1	0	1

BIT OPERATOR CODE

```c
#include <stdio.h>

int main (void){

  int num1 = 15;  //00001111
  int num2 = 20;  //00010100

  int result1 = num1&num2;
  int result2 = num1|num2;
  int result3 = num1^num2;
  int result4 = ~num1;

  printf("&연산의 결과%d\n", result1);
  printf("|연산의 결과%d\n", result2);
  printf("^연산의 결과%d\n", result3);
  printf("~연산의 결과%d\n", result4);

  //과제 직접 손으로 왜 이런 결과가 나왔는지 써보기
  return 0;
}
```

```
&연산의 결과4
|연산의 결과31
^연산의 결과27
~연산의 결과-16
```

SHIFT OPERATOR

- << n 왼쪽으로 N만큼의 bit 이동
  값은 2배씩 증가　　왼쪽으로 이동할 때 마다 0으로 채움
- >> n 오른쪽으로 N만큼의 bit 이동
  값은 절반으로 감소　　오른쪽으로 이동할 때 마다 양수는 0 음수는 CPU에 따라 다르다.

00000000 00000000 00000000 00011110 30

<< 00000000 00000000 00000000 00111100 60

>> 00000000 00000000 00000000 00001111 15

11111111 11111111 11111111 11110000

>> 00111111 11111111 11111111 11111100

>> 11111111 11111111 11111111 11111100

CPU에 따라 결과가 다르다

SHIFT OPERATOR CODE

```c
#include < stio.h>

int main (void){

  int num1 = 15;
  int result1 = num1<<1;
  int result2 = num1<<2;
  int result3 = num1<<3;

  printf("왼쪽으로 1칸 이동한 결과 : %d\n", result1);
  printf("왼쪽으로 2칸 이동한 결과 : %d\n", result2);
  printf("왼쪽으로 3칸 이동한 결과 : %d\n", result3);

  int num2 = -16;
  int result4 = num1>>2;
  int result5 = num1>>4;

  printf("오른쪽으로 2칸 이동한 결과 : %d\n", result4);
  printf("오른쪽으로 3칸 이동한 결과 : %d\n", result5);

  return 0;
}
```

```
왼쪽으로 1칸 이동한 결과 : 30
왼쪽으로 2칸 이동한 결과 : 60
왼쪽으로 3칸 이동한 결과 : 120
오른쪽으로 2칸 이동한 결과 : 3
오른쪽으로 3칸 이동한 결과 : 0
```