데이터 표현방식의 이해

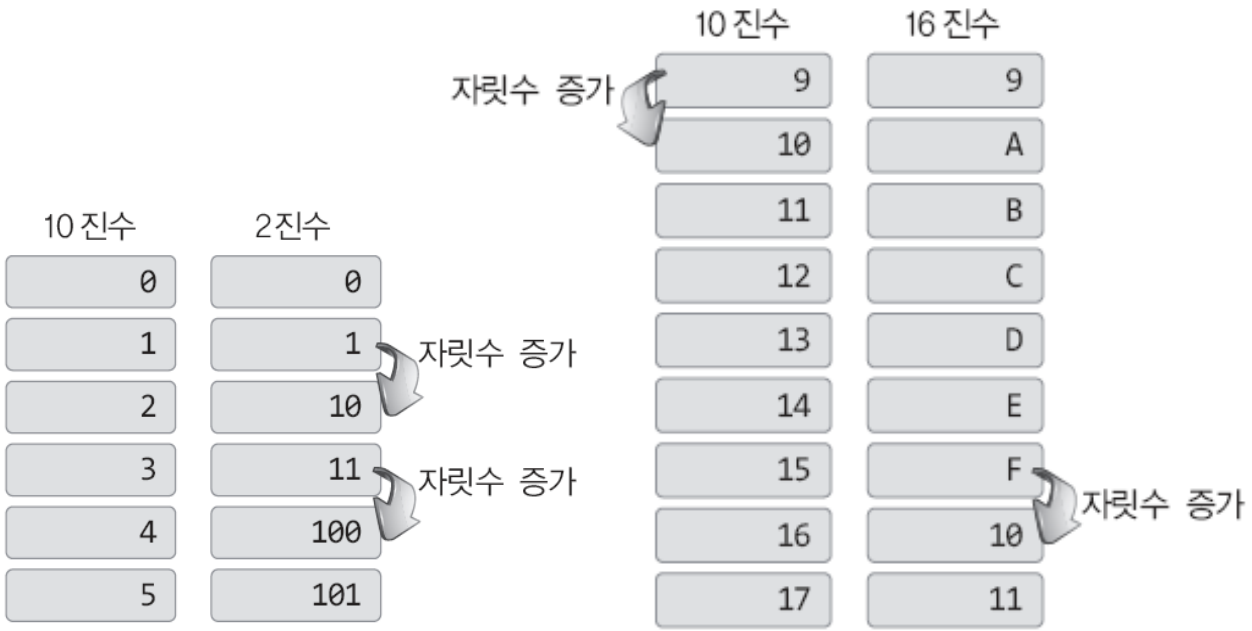
- 진법, 정수와 실수의 표현 방식, 비트 연산자 -

성공회대학교 IT융합자율학부
소프트웨어공학전공
홍 성 준

컴퓨터가 데이터를 표현하는 방식

◎ 2진수? 10진수? 16진수?

- 2진수 : 2개의 기호를 이용해서 데이터를 표현하는 방식 (0, 1)
- 10진수 : 10개의 기호를 이용해서 데이터를 표현하는 방식 (0, 1, ⋯, 9)
- 16진수 : 16개의 기호를 이용해서 데이터를 표현하는 방식 (0, 1, ⋯, 9, A, B, ⋯, F)



컴퓨터가 데이터를 표현하는 방식

◎ Quiz : 진법의 이해

- 10진수 8부터 20까지를 2진수와 16진수로 나타내보자.

- 10진수 5부터 18까지를 8진수로 나타내보자.

컴퓨터가 데이터를 표현하는 방식

◎ 비트(bit)

- 컴퓨터가 표현하는 데이터의 최소단위로서 2진수 값 하나를 저장할 수 있는 메모리의 크기를 의미

◎ 바이트(byte)

- 8개의 비트를 묶어낸 데이터의 표현 단위
- 컴퓨터 메모리에서 1 바이트 당 하나의 주소가 할당
- 1 byte = 8 bits
- 1 KB = 1,024 B = 2^{10} B
- 1 MB = 1,024 KB = 2^{10} KB
- 1 GB = 1,024 MB = 2^{10} MB
- 1 TB = 1,024 GB = 2^{10} GB

컴퓨터가 데이터를 표현하는 방식

◎ Quiz : 데이터 표현의 이해

- 4 bits, 1 byte 그리고 4 bytes로 표현할 수 있는 데이터의 수는 몇 가지인가?

- 8 비트 00000001은 10진수로 1이고, 8 비트 00000010은 10진수 2일 때, 다음 비트열은 10진수로 몇 인가?

  00000100

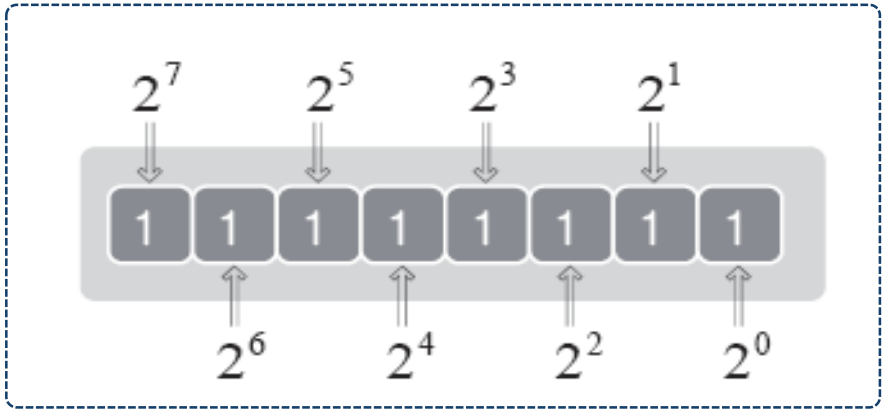
  00001000

  00010000

  00100000

  01000000

  10000000



컴퓨터가 데이터를 표현하는 방식

◎ 8진수와 16진수를 이용한 데이터 표현

- C언어는 10진수 이외에 8진수와 16진수의 데이터 표현도 허용
- Notation.c

```
int main(void)
{
    int num1=0xA7, num2=0x43;
    int num3=032, num4=024;

    printf("0xA7의 10진수 정수 값: %d \n", num1);
    printf("0x43의 10진수 정수 값: %d \n", num2);
    printf(" 032의 10진수 정수 값: %d \n", num3);
    printf(" 024의 10진수 정수 값: %d \n", num4);

    printf("%d-%d=%d \n", num1, num2, num1-num2);
    printf("%d+%d=%d \n", num3, num4, num3+num4);
    return 0;
}
```

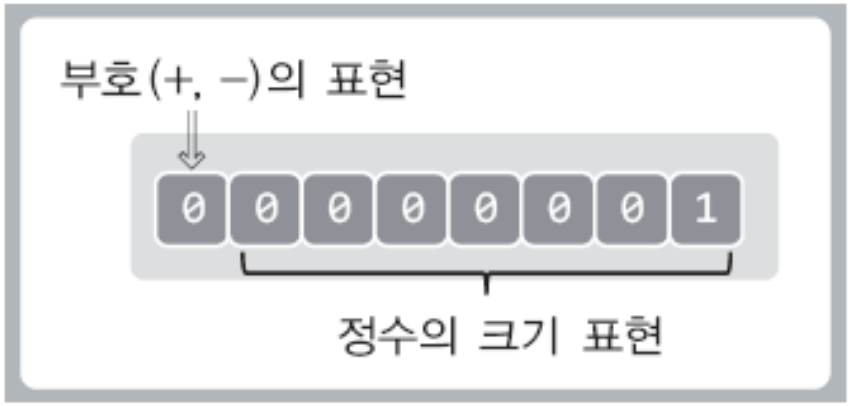
```
int num1 = 10;      // 특별한 선언이 없으면 10진수의 표현
int num2 = 0xA;     // 0x로 시작하면 16진수로 인식
int num3 = 012;     // 0으로 시작하면 8진수로 인식
```

```
0xA7의 10진수 정수 값: 167
0x43의 10진수 정수 값: 67
 032의 10진수 정수 값: 26
 024의 10진수 정수 값: 20
167-67=100
26+20=46
```

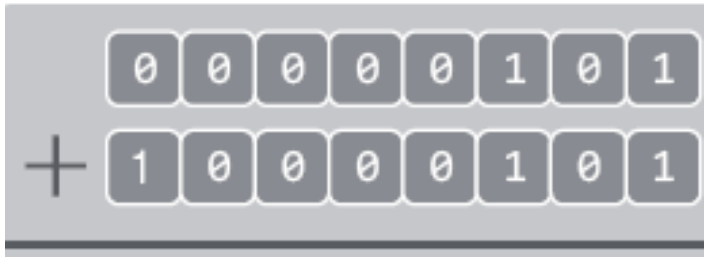
정수와 실수의 표현방식

◎ 정수의 표현방식

- 부호를 나타내는 MSB(Most Significant Bit)와 크기를 나타내는 나머지로 구성



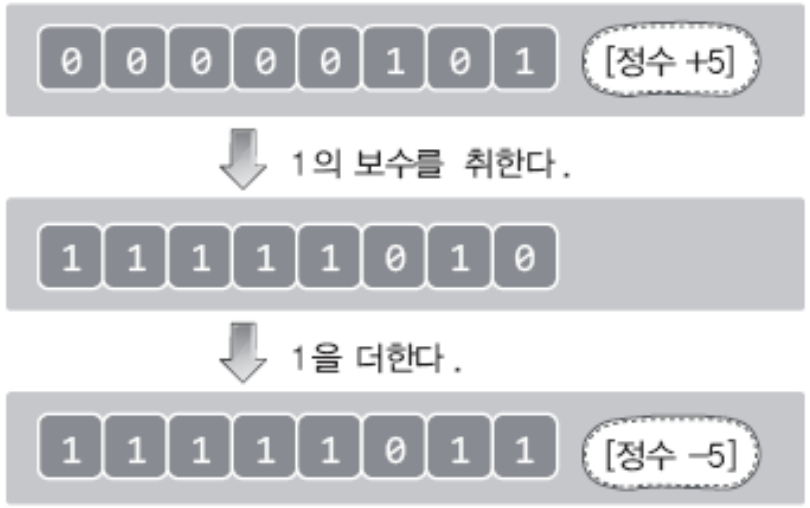
- 양의 정수 : MSB=0, 나머지는 크기 (예) $00000101_{(2)} = +5$
- 음의 정수 : 양의 정수의 2의 보수

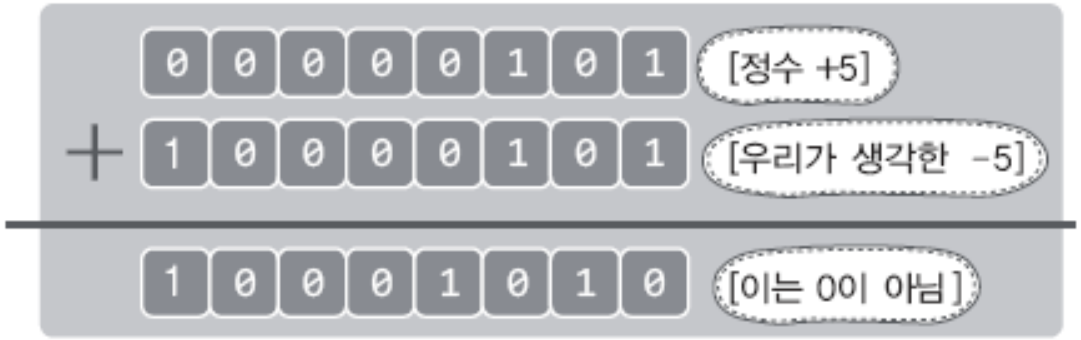


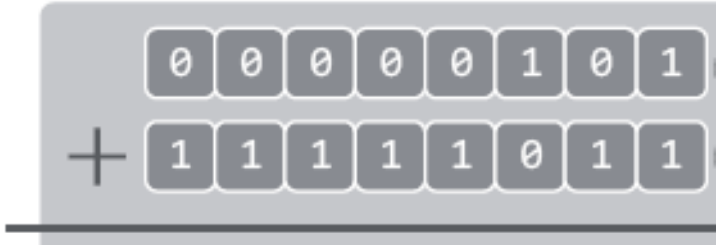
정수와 실수의 표현방식

◎ 정수의 표현방식 (cont.)

- 2의 보수(two's complement)
  - 어떤 커다란 2의 제곱수에서 뺀 수
  - 1의 보수를 구한 후 1을 더하여 계산







정수와 실수의 표현방식

◎ Quiz : 음의 정수 표현하기

- 양의 정수 01001111과 00110011은 각각 10진수로 얼마인가?

- 음의 정수 10101001과 1110000은 각각 10진수로 얼마인가?

정수와 실수의 표현방식

◎ 실수의 표현방식

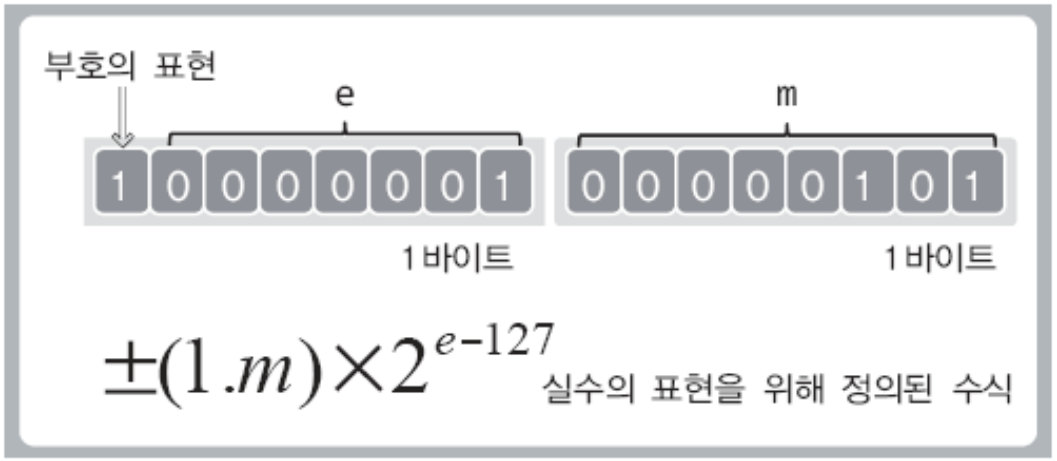
- 정수와 같이 표현?

- 표현할 수 있는 실수의 수가 몇 개 되지 않음
- 0.1과 0.2 사이에 존재하는 수많은 실수를 제대로 표현할 수 없음

정수와 실수의 표현방식

◎ 실수의 표현방식 (cont.)

- 부동 소수점 방식 (floating point)



  - 부호 비트, 지수 비트(e)와 가수 비트(m)으로 구성
  - 적은 수의 비트로 넓은 범위의 실수를 표현할 수 있지만, 실수의 표현에는 오차가 존재한다.
  - 정확한 영(0.0)을 표현할 수 없음
- 읽을 거리: 컴퓨터에서의 실수 표현: 고정소수점 vs 부동소수점 (https://gsmesie692.tistory.com/94)

정수와 실수의 표현방식

◎ 실수 표현의 오차 확인하기

- FloatError.c

```
int main(void)
{
    int i;
    float num=0.0;

    for(i=0; i<100; i++)
        num+=0.1;   // 이 연산을 총 100회 진행하게 됩니다.

    printf("0.1을 100번 더한 결과: %f \n", num);
    return 0;
}
```

```
0.1을 100번 더한 결과: 10.000002
```

비트 연산자

◎ 비트 연산자

- 비트 단위로 연산을 진행하는 단항 또는 이항 연산자
- 메모리 공간의 효율성을 높이고 연산의 수를 줄이는데 용이하여 하드웨어 프로그래밍에 활용

연산자	연산자의 기능	결합방향
&	비트단위로 AND 연산을 한다. 예) num1 & num2;	→
|	비트단위로 OR 연산을 한다. 예) num1 | num2;	→
^	비트단위로 XOR 연산을 한다. 예) num1 ^ num2;	→
~	단항 연산자로서 피연산자의 모든 비트를 반전시킨다. 예) ~num; // num은 변화 없음, 반전 결과만 반환	←
<<	피연산자의 비트 열을 왼쪽으로 이동시킨다. 예) num<<2; // num은 변화 없음, 두 칸 왼쪽 이동 결과만 반환	→
>>	피연산자의 비트 열을 오른쪽으로 이동시킨다. 예) num>>2; // num은 변화 없음, 두 칸 오른쪽 이동 결과만 반환	→

비트 연산자

◎ & 연산자 : 비트단위 AND

- 두 개의 비트가 모두 1일 때 1을 반환하고, 아니면 0을 반환하는 이항 연산자

- 0 & 0 0을 반환
- 0 & 1 0을 반환
- 1 & 0 0을 반환
- 1 & 1 1을 반환

```
        00000000 00000000 00000000 00001111
& 연산  00000000 00000000 00000000 00010100
-------------------------------------------
```

- BitAndOperation.c

```c
int main(void)
{
    int num1 = 15;     // 00000000 00000000 00000000 00001111
    int num2 = 20;     // 00000000 00000000 00000000 00010100
    int num3 = num1 & num2;     // num1과 num2의 비트단위 & 연산
    printf("AND 연산의 결과: %d \n", num3);
    return 0;
}
```

```
AND 연산의 결과: 4
```

비트 연산자

◎ | 연산자 : 비트단위 OR

- 두 개의 비트 중에 하나라도 1일 때 1을 반환하고, 아니면 0을 반환하는 이항 연산자

• 0 & 0	0을 반환
• 0 & 1	1을 반환
• 1 & 0	1을 반환
• 1 & 1	1을 반환

```
        00000000 00000000 00000000 00001111
| 연산   00000000 00000000 00000000 00010100
```

- BitOrOperation.c

```c
int main(void)
{
    int num1 = 15;    // 00000000 00000000 00000000 00001111
    int num2 = 20;    // 00000000 00000000 00000000 00010100
    int num3 = num1 | num2;
    printf("OR 연산의 결과: %d \n", num3);
    return 0;
}
```

```
OR 연산의 결과: 31
```

비트 연산자

◎ ^ 연산자 : 비트단위 XOR

- 두 개의 비트가 다르면 1을 반환하고, 같으면 0을 반환하는 이항 연산자

• 0 & 0	0을 반환
• 0 & 1	1을 반환
• 1 & 0	1을 반환
• 1 & 1	0을 반환

```
        00000000 00000000 00000000 00001111
^ 연산  00000000 00000000 00000000 00010100
```

- BitXorOperation.c

```c
int main(void)
{
    int num1 = 15;    // 00000000 00000000 00000000 00001111
    int num2 = 20;    // 00000000 00000000 00000000 00010100
    int num3 = num1 ^ num2;
    printf("XOR 연산의 결과: %d \n", num3);
    return 0;
}
```

```
XOR 연산의 결과: 27
```

비트 연산자

◎ ~ 연산자 : 비트단위 NOT

- 비트 1을 0으로, 비트 0을 1로 반환하는 단항 연산자 (1의 보수)

• ~ 0	1을 반환
• ~ 1	0을 반환

```
      00000000 00000000 00000000 00001111
     ------------------------------------
```

- BitNotOperation.c

```c
int main(void)
{
    int num1 = 15;    // 00000000 00000000 00000000 00001111
    int num2 = ~num1;
    printf("NOT 연산의 결과: %d \n", num2);
    return 0;
}
```

```
NOT 연산의 결과: -16
```

비트 연산자

◎ << 연산자 : 비트의 왼쪽 이동(shift)

- a << b : 피연산자 a의 비트열을 b칸씩 왼쪽으로 이동시킨 결과를 반환
- 비트 이동으로 생기는 오른쪽 빈 칸은 0으로 채워지고, 밀려나는 왼쪽 비트는 버려짐
- BitLeftShift.c

```
int main(void)
{
    int num = 15;     // 00000000 00000000 00000000 00001111

    int result1 = num<<1;     // num의 비트 열을 왼쪽으로 1칸씩 이동
    int result2 = num<<2;     // num의 비트 열을 왼쪽으로 2칸씩 이동
    int result3 = num<<3;     // num의 비트 열을 왼쪽으로 3칸씩 이동

    printf("1칸 이동 결과: %d \n", result1);
    printf("2칸 이동 결과: %d \n", result2);
    printf("3칸 이동 결과: %d \n", result3);
    return 0;
}
```

```
1칸 이동 결과: 30
2칸 이동 결과: 60
3칸 이동 결과: 120
```

- 정수의 비트 열을 왼쪽으로 1칸씩 이동시킬 때마다 정수의 값은 두 배가 됨

비트 연산자

◎ >> 연산자 : 비트의 오른쪽 이동(shift)

- a >> b : 피연산자 a의 비트열을 b칸씩 오른쪽으로 이동시킨 결과를 반환
- 비트 이동으로 밀려나는 오른쪽 비트는 버려지고, 양수(MSB=0)라면 왼쪽 비트는 0으로 채워짐[1)]
- BitRightShift.c

```
int main(void)
{
    int num = -16;    // 11111111 11111111 11111111 11110000
    printf("2칸 오른쪽 이동의 결과: %d \n", num>>2);
    printf("3칸 오른쪽 이동의 결과: %d \n", num>>3);
    return 0;
}
```

```
2칸 오른쪽 이동의 결과: -4
3칸 오른쪽 이동의 결과: -2
```

  - 부호 비트를 유지하는 시스템에서의 실행결과임.

- 양의 정수 비트 열을 오른쪽으로 1칸씩 이동시킬 때마다 정수의 값은 2로 나누어짐

1) 음수(MSB=1)인 경우, CPU에 따라 음수를 유지하기 위해 1을 채우기도 하고, 음수 여부에 상관없이 0을 채우기도 함. 따라서 컴퓨터 간의 CPU 호환성을 유지하기 위해서는 음수에서 >> 연산자 사용을 제한해야함