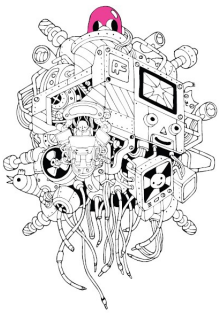


Chapter 04. 데이터 표현방식의 이해

4-1 컴퓨터의 데이터 표현

- 진법에 대한 이해
  - n 진수 표현 방식 : n개의 문자를 이용해서 데이터를 표현

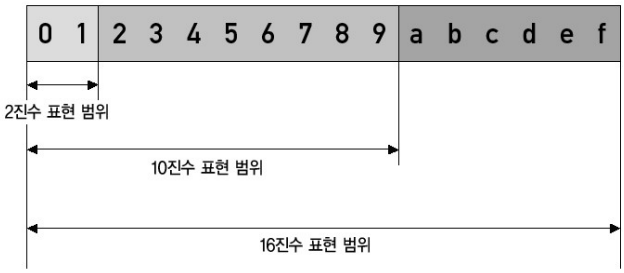


그림 4-1

2

4-1 컴퓨터의 데이터 표현

- **2진수와 10진수**
  - 10진수 : 0~9를 이용한 데이터의 표현
  - 2진수 : 0과 1을 이용한 데이터의 표현
  - 컴퓨터는 내부적으로 모든 데이터 2진수로 처리

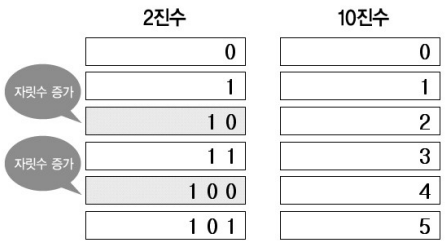


그림 4-2

3

4-1 컴퓨터의 데이터 표현

- **16진수와 10진수**
  - 16진수 : 0~9, a, b, c, d, e, f를 이용한 데이터의 표현

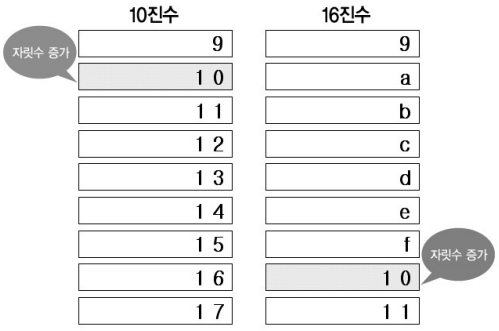


그림 4-3

4

4-1 컴퓨터의 데이터 표현

- 데이터의 표현 단위인 비트(bit)와 바이트(byte)
  - 비트 : 데이터 표현의 최소 단위, 2진수 값 하나 (0 or 1)을 저장
  - 바이트 : 8비트 == 1바이트

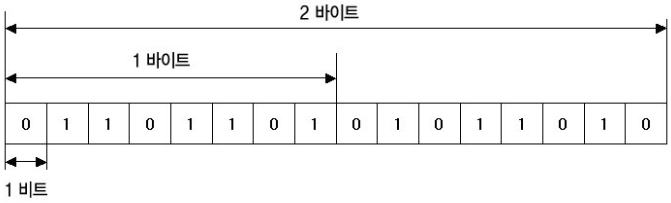


그림 4-4

5

4-1 컴퓨터의 데이터 표현

- 연습문제 4-2

0	0	0	0	0	0	0	0	[ ]
0	0	0	0	0	0	0	1	[ ]
0	0	0	0	0	0	1	0	[ ]
0	0	0	0	0	1	0	0	[ ]
0	0	0	0	1	0	0	0	[ ]
0	0	0	1	0	0	0	0	[ ]
0	0	1	0	0	0	0	0	[ ]
0	1	0	0	0	0	0	0	[ ]
1	0	0	0	0	0	0	0	[ ]

6

4-1 컴퓨터의 데이터 표현

- 프로그램상에서의 8진수, 16진수 표현
  - 8진수 : 0으로 시작
  - 16진수 : 0x로 시작
  - 예제 notation.c 참조

Notation.c

```
int a = 10;          // 10진수. 아무런 표시도 없으므로…
int b = 0xa;         // 16진수. 0x로 시작하므로…
int c = 012;         // 8진수. 0으로 시작하므로…
```

7

4-2 정수와 실수의 표현 방식

- 정수의 표현 방식
  - MSB : 가장 왼쪽 비트, 부호를 표현
  - MSB를 제외한 나머지 비트 : 데이터의 크기 표현

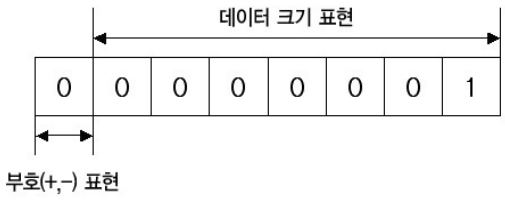


그림 4-7

8

4-2 정수와 실수의 표현 방식

- 잘못된 음의 정수 표현 방식
  - 양의 정수 표현 방식을 적용한 경우

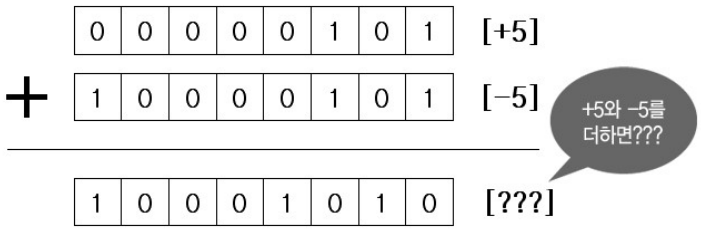


그림 4-8

9

4-2 정수와 실수의 표현 방식

- 정확한 음의 정수 표현 방식
  - 2의 보수를 이용한 음의 정수 표현 방식

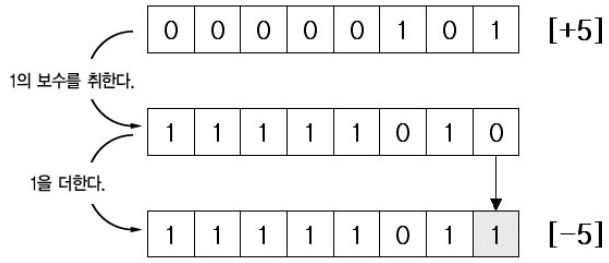


그림 4-9

10

4-2 정수와 실수의 표현 방식

- **음수 표현 방식의 증명**

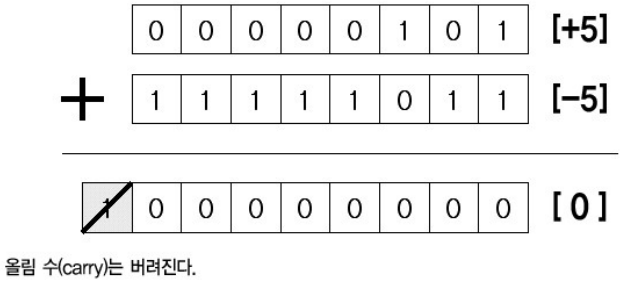


그림 4-10

11

4-2 정수와 실수의 표현 방식

- **연습문제 4-3**
  - 양의 정수
  - 01001111 [ ]
  - 00110011 [ ]
  - 음의 정수
  - 10101001 [ ]
  - 11110000 [ ]

12

4-2 정수와 실수의 표현 방식

- **잘못된 실수의 표현 방식**
  - 정수를 표현하는 방식을 실수 표현에 적용
  - 작은 수를 표현하는데 있어서 한계를 지님

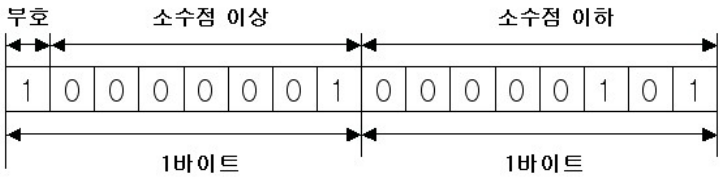
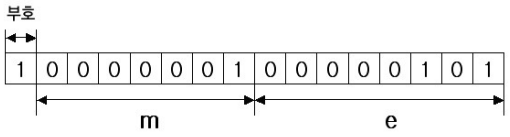


그림 4-11

13

4-2 정수와 실수의 표현 방식

- **정확한 실수 표현 방식**
  - 오차가 존재하는 단점을 지님, 그러나 효율적인 표현 방식 → **근사치를 표현 (100% 정확하지는 않음)**
  - 예제 float_error.c 참조 → **"부동 소수점 오차"**



FloatError.c

$$\pm(1.m)\times 2^{e-127}$$

그림 4-12

14

4-3 비트 단위 연산

비트 단위 연산자의 종류

연산자	연산자의 의미	결합성
&	비트 단위 AND ex) a & b	→
|	비트 단위 OR ex) a | b	
^	비트 단위 XOR ex) a ^ b	
~	비트 단위 NOT ex) ~a	
<<	왼쪽으로 이동 ex) a << 2	
>>	오른쪽으로 이동 ex) a >> 2	

표 4-1

15

4-3 비트 단위 연산

& 연산자 : 비트 단위 AND

0 & 0 → 0을 반환
0 & 1 → 0을 반환
1 & 0 → 0을 반환
1 & 1 → 1을 반환

BitAndOperation.c

```
int main(void)
{
    int a=15;       // 00000000 00000000 00000000 00001111
    int b=20;       // 00000000 00000000 00000000 00010100
    int c = a&b;

    printf("AND 연산 결과 : %d", c);     // 출력 결과 4
```

16

4-3 비트 단위 연산

- **& 연산자 : 비트 단위 AND**

```
    00000000 00000000 00000000 00001111
AND 00000000 00000000 00000000 00010100
-------------------------------------------
    00000000 00000000 00000000 00000100
```

그림 4-13

17

4-3 비트 단위 연산

- **| 연산자 : 비트 단위 OR**

```
0 | 0  →  0을 반환
0 | 1  →  1을 반환
1 | 0  →  1을 반환
1 | 1  →  1을 반환
```

BitOrOperation.c

```
int main(void)
{
    int a=15;   // 00000000 00000000 00000000 00001111
    int b=20;   // 00000000 00000000 00000000 00010100
    int c = a|b;

    printf("OR 연산 결과 : %d", c);        // 출력 결과 31
```

18

4-3 비트 단위 연산

- | 연산자 : 비트 단위 OR

```
  00000000 00000000 00000000 00001111
OR 00000000 00000000 00000000 00010100
-------------------------------------
  00000000 00000000 00000000 00011111
```

그림 4-14

19

4-3 비트 단위 연산

- ^ 연산자 : 비트 단위 XOR

```
0 ^ 0  →  0을 반환
0 ^ 1  →  1을 반환
1 ^ 0  →  1을 반환
1 ^ 1  →  0을 반환
```

BitXorOperation.c

```
int main(void)
{
   int a=15;   // 00000000 00000000 00000000 00001111
   int b=20;   // 00000000 00000000 00000000 00010100
   int c = a^b;

   printf("XOR 연산 결과 : %d", c);     // 출력 결과 27
```

20

4-3 비트 단위 연산

- ^ 연산자 : 비트 단위 XOR

```
    00000000 00000000 00000000 00001111
XOR 00000000 00000000 00000000 00010100
-------------------------------------------
    00000000 00000000 00000000 00011011
```

그림 4-15

21

4-3 비트 단위 연산

- ~ 연산자 : 비트 단위 NOT

~0 → 1을 반환
~1 → 0을 반환

BitNotOperation.c

```
int main(void)
{
    int a=15;
    int b=~a;

    printf("NOT 연산 결과 : %d", b);     // 출력 결과 -16
```

보수를 계산하는 방법은??

22

4-3 비트 단위 연산

- ~ 연산자 : 비트 단위 NOT

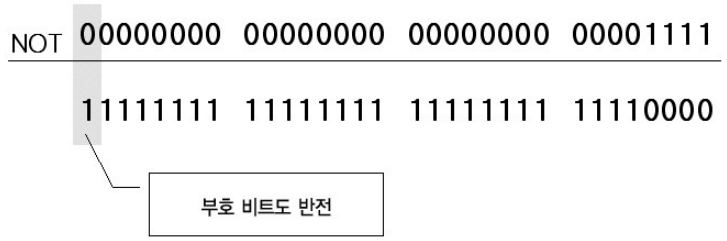


그림 4-16

23

4-3 비트 단위 연산

- << 연산자 : 왼쪽 쉬프트(shift) 연산

a<<b → a의 비트들을 b칸씩 왼쪽으로 이동한 값을 반환
8<<2 → 8의 비트들을 왼쪽으로 2칸씩 이동한 값을 반환

BitLeftShift.c

```
int main(void)
{
   int a=15;   // 00000000  00000000  00000000  00001111
   int b= a<<2;    // a의 비트들을 왼쪽으로 2칸씩 이동

   printf("<<2 연산 결과 : %d", b);      // 출력 결과 60
```

24

4-3 비트 단위 연산

- **<< 연산자 : 왼쪽 쉬프트(shift) 연산**

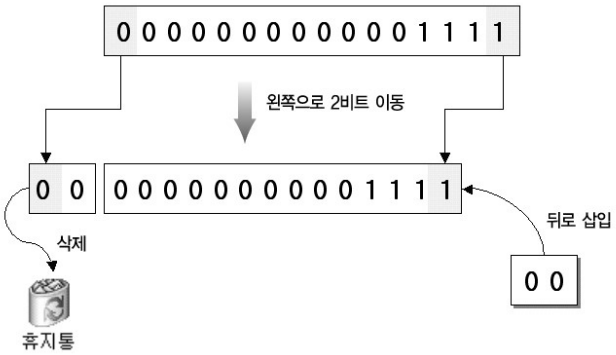


그림 4-17

4-3 비트 단위 연산

- **>> 연산자 : 오른쪽 쉬프트(shift) 연산**

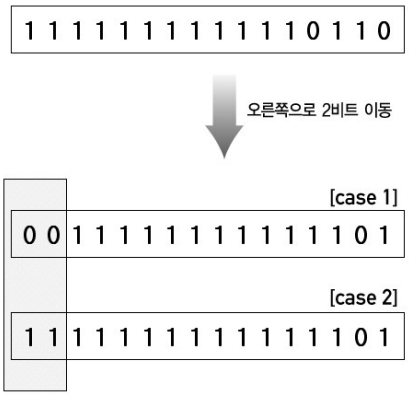
a>>b → a의 비트들을 b칸씩 오른쪽으로 이동한 값을 반환
8>>2 → 8의 비트를 왼쪽으로 2칸씩 이동한 값을 반환

BitRightShift.c

```
a=-10;
b=a>>2;   // a의 비트들을 2칸씩 오른쪽으로 이동한 값을 b에 저장
```

4-3 비트 단위 연산

- **>> 연산자 : 오른쪽 쉬프트(shift) 연산**



BitLeftShift.c

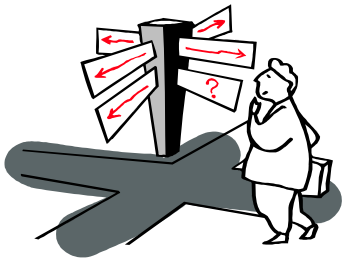
- 왼쪽에 채우는 값은 CPU에 따라 다름
- [case 1] 부호에 관계없이 '0'으로 채움
- [case 2] 부호를 고려하여 '1'로 채움

27

실습시간 (2019년 3월 27일)

- 교재 예제: (8개)
  - 표현: Notation.c, FloatError.c
  - 비트 연산: BitAndOperation.c, BitOrOperation.c, BitXorOperation.c, BitNotOperation.c, BitLeftShift.c, BitRightShift.c
- Lab1
  - 키보드로부터 정수 한 개를 **입력**하면, 입력된 정수 값의 **부호를 바꿔서** 출력하는 프로그램을 작성하라. (**비트 연산자**를 사용)
  - 결과 예: 입력한 수 10의 보수는 -10입니다.
- Lab2
  - 다음 연산을 **Shift 연산자**를 사용하여 구현하라.
    $15 \times 16 \div 8$

28



Chapter 04이 끝났습니다. 질문 있으신지요?