기초프로그래밍

제5장 연산자

Sangsoo Lim
CSAI
Dongguk University

차례

- 변수와 변수의 시작 주소
- 연산자의 종류
- 비트 연산자

변수와 변수의 시작 주소

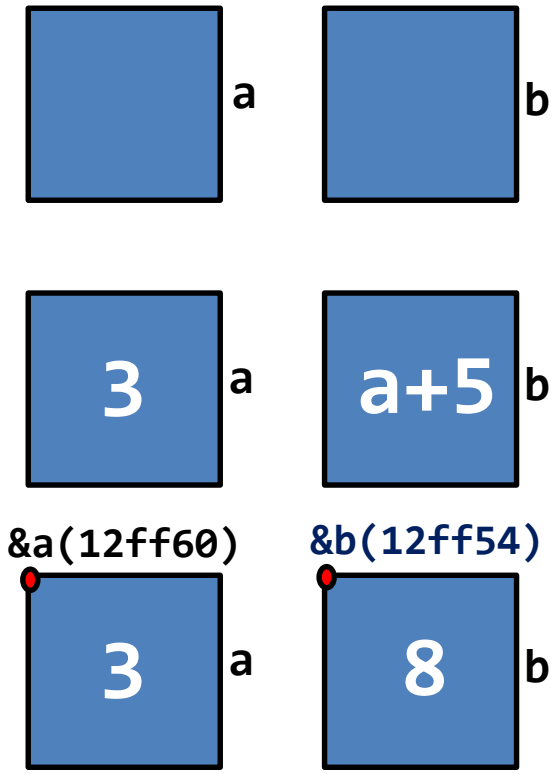
```c
/* 5-1.c */
#include<stdio.h>
int main(void)
{
   int a;
   int b;

   a = 3;
   b = a + 5;

   printf("a의 값: %d \n", a);
   printf("b의 값: %d \n", b);

   printf("변수a의 주소: %x \n", &a);
   printf("변수b의 주소: %x \n", &b);

   return 0;
}
```



연산자의 종류

- **연산자와 피연산자**
  - **연산자(Operator):** 연산을 수행하는 기호
  - **피연산자(Operand) :** 연산에 참여하는 변수 또는 상수 값

연산자의 종류

- **연산자의 종류**

분류	연산자
① 대입 연산자	=
② 산술 연산자	+, -, *, /, %
③ 복합 대입 연산자	+=, -=, *=, /=, %=
④ 증감연산자	++, --
⑤ 관계 연산자	>, <, ==, !=, >=, <=
⑥ 논리 연산자	&&, ||, !
⑦ 조건 연산자	? :
⑧ 비트 논리 연산자	&, !, ^, ~
⑨ 비트 이동 연산자	>>, <<

연산자의 종류

① 대입 연산자(=)

- 데이터를 저장하는 연산자
- 오른쪽에 있는 **값**을 왼쪽에 있는 **변수 i에 대입(저장)**하라는 의미

연산자의 종류

```c
/* 5-2.c */
#include<stdio.h>
int main(void)
{
   int i=0, j=0, k=0;

   printf("i = %d, j = %d, k = %d \n", i, j, k);

   i = 1;
   j = 5;
   k = 7;

   printf("i = %d, j = %d, k = %d\n", i, j, k);

   return 0;
}
```

```
i = 0, j = 0, k = 0
i = 1, j = 5, k = 7
```

연산자의 종류

② 산술 연산자

- 덧셈(+), 뺄셈(-), 곱셈(*), 나눗셈(/), 나머지(%)

산술 연산자	예	설명
+(덧셈 연산자)	a = 6+2	피연산자 6과 피연산자 2의 덧셈 연산
-(뺄셈 연산자)	a = 6-2	피연산자 6과 피연산자 2의 뺄셈 연산
*(곱하기 연산자)	a = 6*2	피연산자 6과 피연산자 2의 곱셈 연산
/(나누기 연산자)	a = 6/2	피연산자 6과 피연산자 2의 나눗셈 연산
%(나머지 연산자)	a = 6%2	피연산자 6과 피연산자 2를 나눈 나머지 연산

연산자의 종류

```c
/* 5-3.c */
#include<stdio.h>
int main(void)
{
   int a, b;
   a = 6;
   b = 2;

   printf("덧셈 연산 결과: %d \n", a+b);
   printf("뺄셈 연산 결과: %d \n", a-b);
   printf("곱셈 연산 결과: %d \n", a*b);
   printf("나누기 연산 결과: %d \n", a/b);
   printf("나머지 연산 결과: %d \n", a%b);

   return 0;
}
```

```c
/* 5-4.c */
#include<stdio.h>
int main(void)
{
   int num1, num2;

   num1 = 10/3;        // '몫' 출력
   num2 = 10%3;        // '나머지' 출력

   printf("몫: %d \n", num1);
   printf("나머지: %d \n", num2);

   return 0;
}
```

연산자의 종류

③ 복합 대입 연산자

– 산술 연산자와 대입 연산자를 하나로 나타내는 기호

복합 대입 연산자	같은 표현	설명
a = a + b	a += b	a + b를 먼저 수행한 후에 a에 값을 저장
a = a - b	a -= b	a - b를 먼저 수행한 후에 a에 값을 저장
a = a * b	a *= b	a * b를 먼저 수행한 후에 a에 값을 저장
a = a / b	a /= b	a / b를 먼저 수행한 후에 a에 값을 저장
a = a % b	a %= b	a % b를 먼저 수행한 후에 a에 값을 저장

연산자의 종류

```c
/* 5-5.c */
#include<stdio.h>
int main(void)
{
    int num1=1, num2=2, num3=3, num4=4, num5=5;
    num1 = num1 + num2;       // num1 += num2;
    num2 = num2 - 2;          // num2 -= 2;
    num3 = num3 * 2;          // num3 *= 2;
    num4 = num4 / 2;          // num4 /= 2;
    num5 = num5 % 2;          // num5 %= 2;

    printf("%d, %d, %d, %d, %d \n",num1,num2,num3,num4,num5);

    return 0;
}
```

연산자의 종류

④ 증감 연산자

- ++, -- 기호를 이용하는 연산자 (1증가 또는 1감소 시키는 연산자)

증감 연산자	설명
++a	선 증가, 후 연산 (먼저 증가하고 그 다음 연산)
a++	선 연산, 후 증가 (먼저 연산하고 그 다음 증가)
--a	선 감소, 후 연산 (먼저 감소하고 그 다음 연산)
a--	선 연산, 후 감소 (먼저 연산하고 그 다음 감소)

연산자의 종류

```c
/* 5-6.c */
#include<stdio.h>
int main(void)
{
   int num1=10;
   printf("%d \n", num1);    // 결과는 10

   num1++;     // num1 = num1 + 1;
   printf("%d \n", num1);    // 결과는 11

   ++num1;     // num1 = num1 + 1;
   printf("%d \n", num1);    // 결과는 12

   --num1;     // num1 = num1 - 1;
   printf("%d \n", num1);    // 결과는 11

   num1--;     // num1 = num1 - 1;
   printf("%d \n", num1);    // 결과는 10
   return 0;
}
```

```
10
11
12
11
10
```

연산자의 종류

```c
/* 5-7.c */
#include<stdio.h>
int main(void)
{
   int num1=10, num2=10;
   int a, b;

   a = ++num1;  // 전위 방식, 선 증가 후 연산
   printf("%d, %d \n", a, num1); // 11, 11

   b = num2++;  // 후위 방식, 선 연산 후 증가
   printf("%d, %d \n", b, num2); // 10, 11

   return 0;
}
```

```c
/* 5-8.c */
#include<stdio.h>
int main(void)
{
   int num1=10, num2=10;
   int a, b;

   num1 = num1 + 1;   // 선 증가
   a = num1;          // 후 연산
   printf("%d, %d \n", a, num1); // 11, 11

   b = num2;          // 선 연산
   num2 = num2 + 1;   // 후 증가
   printf("%d, %d \n", b, num2); // 10, 11

   return 0;
}
```

연산자의 종류

```c
/* 5-9.c */
#include<stdio.h>
int main(void)
{
   int num1=10, num2=10;

   printf("%d \n", ++num1); // 결과는 11
   printf("%d \n", num1);   // 결과는 11

   printf("%d \n", num2++); // 결과는 10
   printf("%d \n", num2);   // 결과는 11

   return 0;
}
```

연산자의 종류

⑤ 관계 연산자

– 관계를 비교하여 참(True)과 거짓(False)으로 결론짓는 연산자

관계 연산자	예	설명	결과
>	a>b	a가 b보다 클지를 비교	1(참), 0(거짓)
<	a<b	a가 b보다 작을지를 비교	1(참), 0(거짓)
>=	a>=b	a가 b보다 크거나 같을지를 비교	1(참), 0(거짓)
<=	a<=b	a가 b보다 작거나 같을지를 비교	1(참), 0(거짓)
==	a==b	a가 b보다 같을지를 비교	1(참), 0(거짓)
!=	a!=b	a가 b보다 같지 않을지를 비교	1(참), 0(거짓)

연산자의 종류

```c
/* 5-10.c */
#include<stdio.h>
int main(void)
{
   int num1=2, num2=4;
   int result1, result2, result3, result4;

   result1 = (num1 > num2);
   result2 = (num1 <= num2);
   result3 = (num1 == num2);
   result4 = (num1 != num2);

   printf("result1에 저장된 값 %d \n", result1);   // 0(거짓)
   printf("result2에 저장된 값 %d \n“, result2);   // 1(참)
   printf("result3에 저장된 값 %d \n", result3);   // 0(거짓)
   printf("result4에 저장된 값 %d \n", result4);   // 1(참)

   return 0;
}
```

연산자의 종류

⑥ 논리 연산자

- **&&** : AND 연산자 (논리곱)
- **||** : OR 연산자 (논리합)
- **!** : NOT 연산자 (논리 부정)

A	B	A && B	A || B	!A	!B
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

연산자의 종류

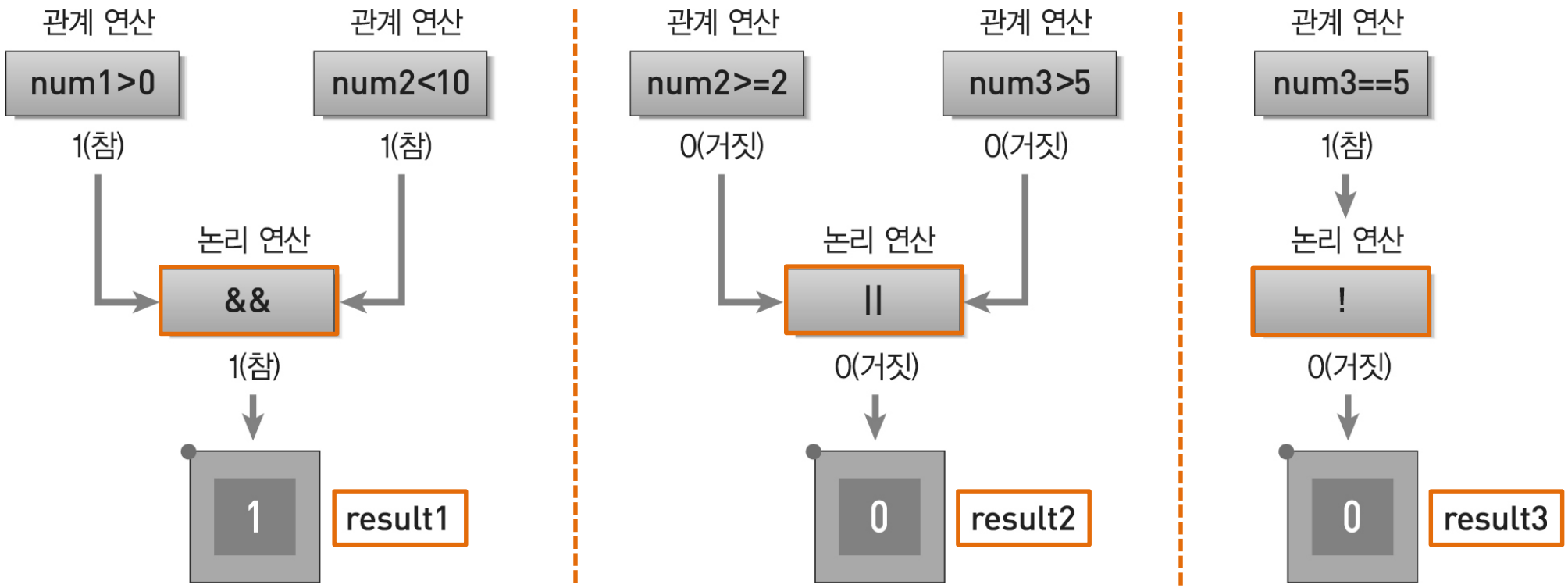
```c
/* 5-11.c */
#include<stdio.h>
int main(void)
{
   int num1=2, num2=3, num3=5;
   int result1, result2, result3;

   result1 = (num1>0) && (num2<10);
   result2 = (num2<=2) || (num3>5);
   result3 = !num3;

   printf("result1에 저장된 값 %d \n", result1);  // 1(참)
   printf("result2에 저장된 값 %d \n", result2);  // 0(거짓)
   printf("result3에 저장된 값 %d \n", result3);  // 0(거짓)

   return 0;
}
```

연산자의 종류

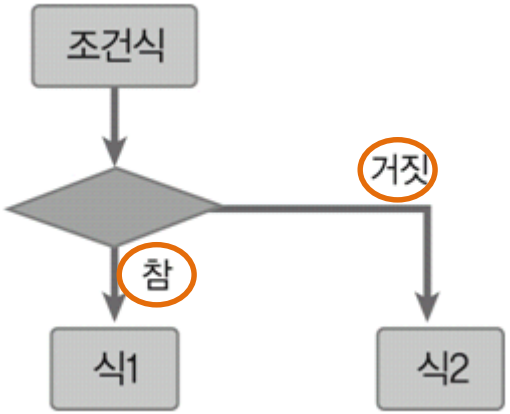

관계 연산
num1>0
1(참)
관계 연산
num2<10
1(참)
논리 연산
&&
1(참)
1
result1
관계 연산
num2>=2
0(거짓)
관계 연산
num3>5
0(거짓)
논리 연산
||
0(거짓)
0
result2
관계 연산
num3==5
1(참)
논리 연산
!
0(거짓)
0
result3

연산자의 종류

⑦ 조건 연산자

– '?' 와 ':' 로 이루어진 연산자

– **조건식** 판단의 결과로 **참**과 **거짓**을 수행



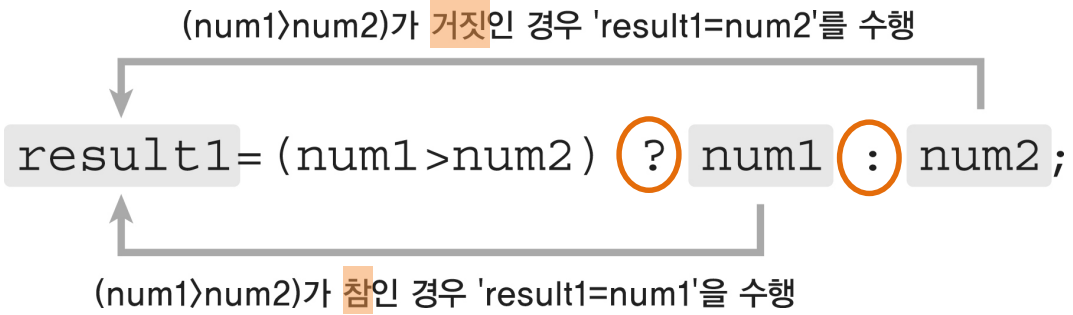
연산자의 종류

```c
/* 5-12.c */
#include<stdio.h>
int main(void)
{
    int num1=2, num2=3;
    int result1;

    result1 = (num1>num2) ? num1 : num2;
    printf("result1에 저장된 값 %d \n",result1);

    return 0;
}
```

```
result1에 저장된 값 3
```



비트 연산자

- **비트와 바이트**
  - **비트(Bit) : 2진수 값 하나(0 또는 1)를 저장할 수 있는 최소 메모리 공간**

1비트	2비트	3비트	…	n비트
$2^1 = 2$개	$2^2 = 4$개	$2^3 = 8$개	…	2^n개

  - **1바이트는 8비트**

비트 연산자

- **2진수, 10진수, 16진수, 8진수**
  - **2진수:** 0~1까지의 숫자를 사용
  - **10진수:** 0~9까지의 숫자를 사용
  - **16진수:** 0~9까지의 숫자를 사용하고, 9 이후부터 a, b, c, d, e, f 문자 사용
  - **8진수:** 0~7까지의 숫자를 사용

비트 연산자

- 데이터 표현 방법 (2진수, 10진수, 16진수, 8진수)

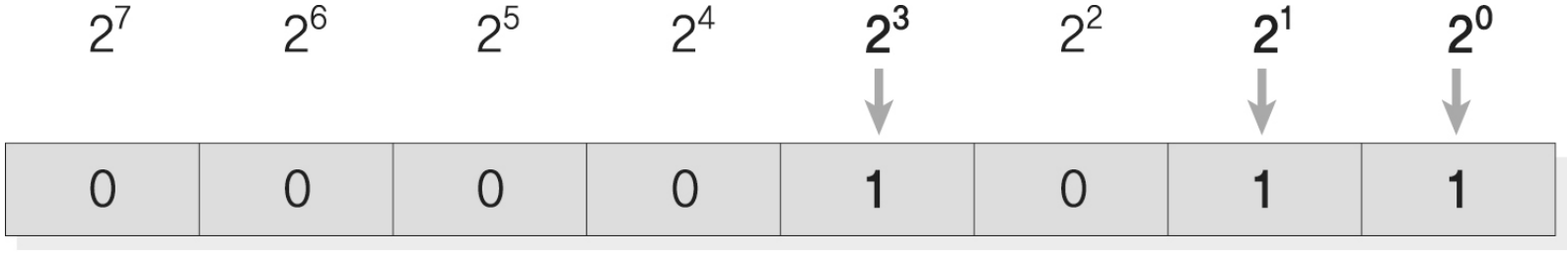
2진수	10진수	16진수	8진수
0000 0000	0	0	0
0000 0001	1	1	1
0000 0010	2	2	2
0000 0011	3	3	3
0000 0100	4	4	4
0000 0101	5	5	5
0000 0110	6	6	6
0000 0111	7	7	7
0000 1000	8	8	10

2진수	10진수	16진수	8진수
0000 1001	9	9	11
0000 1010	**10**	**a**	**12**
0000 1011	11	**b**	13
0000 1100	12	**c**	14
0000 1101	13	**d**	15
0000 1110	14	**e**	16
0000 1111	15	**f**	17
0001 0000	16	**10**	20
0001 0001	17	11	21

10 = 0xa = 012

비트 연산자

- **2진수를 10진수로 표현하는 방법**

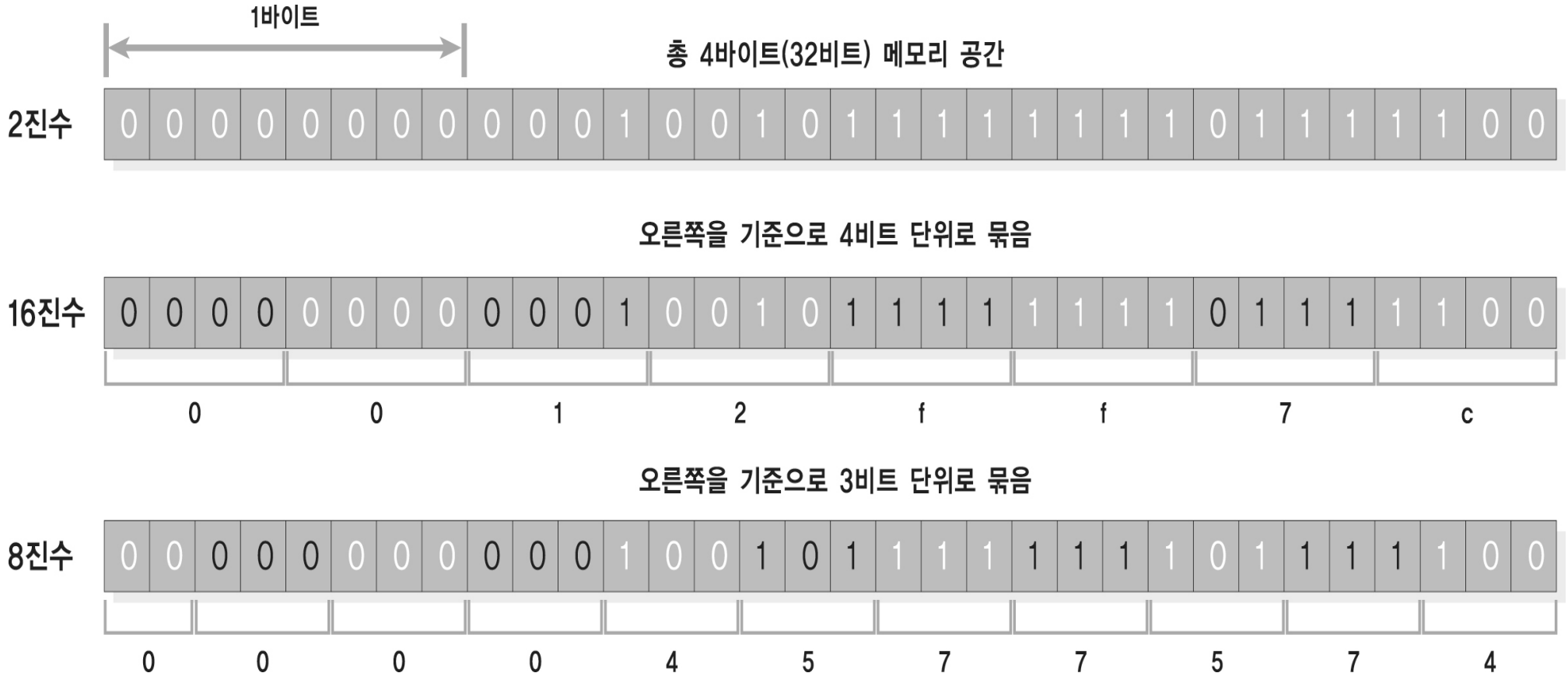


$2^3 + 2^1 + 2^0$ == 10진수로 11

2진수	10진수	2진수	10진수	2진수	10진수
0000 0000		0000 0110		0000 1100	
0000 0001		0000 0111		0000 1101	
0000 0010		0000 1000		0000 1110	
0000 0011		0000 1001		0000 1111	
0000 0100		0000 1010		0001 0000	
0000 0101		0000 1011		0001 0001	

비트 연산자

- **2진수를 16진수와 8진수로 표현하는 방법**



비트 연산자

⑧ 비트 연산자

– 데이터를 비트 단위로 처리하는 연산자

비트 연산자	연산식	설명
&	a & b	비트 단위 AND 연산
|	a | b	비트 단위 OR 연산
^	a ^ b	비트 단위 XOR 연산
~	~a	비트 단위 NOT 연산
<<	a << 3	왼쪽으로 세 칸 이동
>>	a >> 1	오른쪽으로 한 칸 이동

비트 연산자

- **& 연산자**
  - 비트 단위로 AND 연산을 수행
  - 두 개의 비트가 모두 1일 때 1을 반환

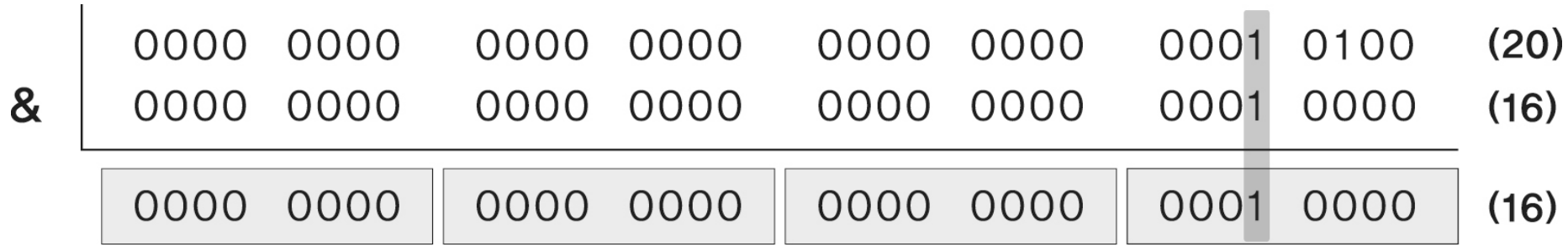
피연산자	연산자	피연산자	결과
0	&	0	0
0	&	1	0
1	&	0	0
1	&	1	1

비트 연산자

```c
/* 5-13.c */
#include<stdio.h>
int main(void)
{
    int num1=20, num2=16;
    int result1;

    result1 = num1 & num2;
    printf("비트단위 & 연산의 결과 %d \n", result1);    // 결과는 16

    return 0;
}
```



비트 연산자

- **| 연산자**
  - 비트 단위로 OR 연산을 수행
  - 두 개의 비트 중의 하나가 1일 때 1을 반환

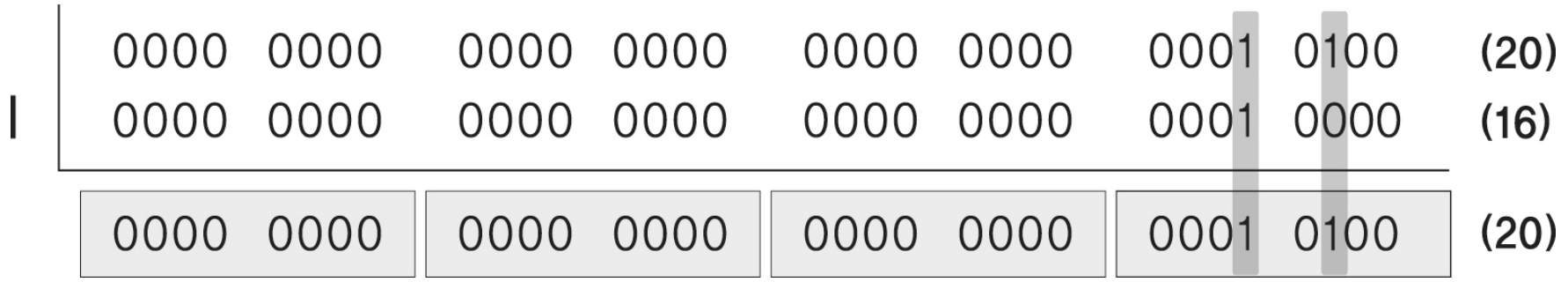
피연산자	연산자	피연산자	결과
0	|	0	0
0	|	1	1
1	|	0	1
1	|	1	1

비트 연산자

```c
/* 5-14.c */
#include<stdio.h>
int main(void)
{
   int num1=20, num2=16;
   int result1;

   result1 = num1 | num2;
   printf("비트단위 | 연산의 결과 %d \n",result1);

   return 0;
}
```



비트 연산자

- ^ 연산자
  - 비트 단위로 XOR 연산을 수행
  - 두 개의 비트가 서로 같지 않을 경우 1을 반환

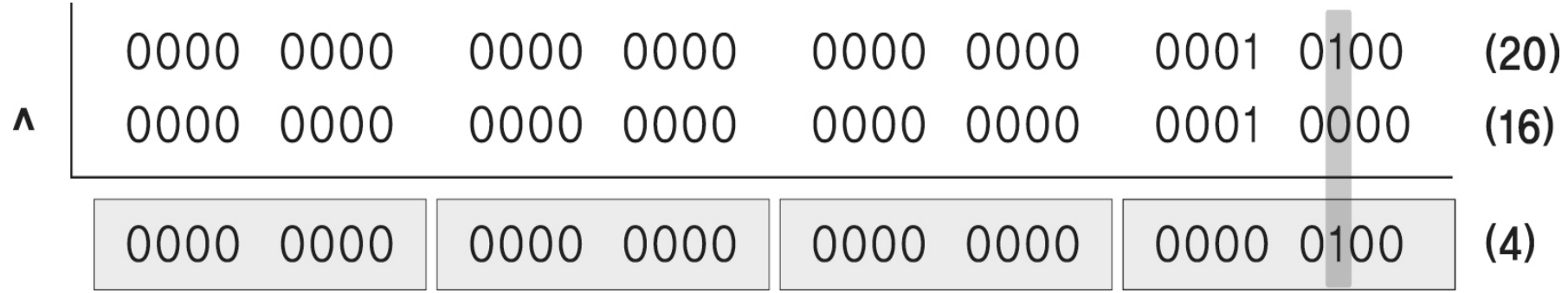
피연산자	연산자	피연산자	결과
0	^	0	0
0	^	1	1
1	^	0	1
1	^	1	0

비트 연산자

```c
/* 5-15.c */
#include<stdio.h>
int main(void)
{
   int num1=20, num2=16;
   int result1;

   result1 = num1 ^ num2;
   printf("비트단위 ^ 연산의 결과 %d \n",result1);

   return 0;
}
```



비트 연산자

- ~ 연산자
  - 비트 단위로 NOT 연산을 수행
  - 보수 연산으로 비트를 반전 시킴

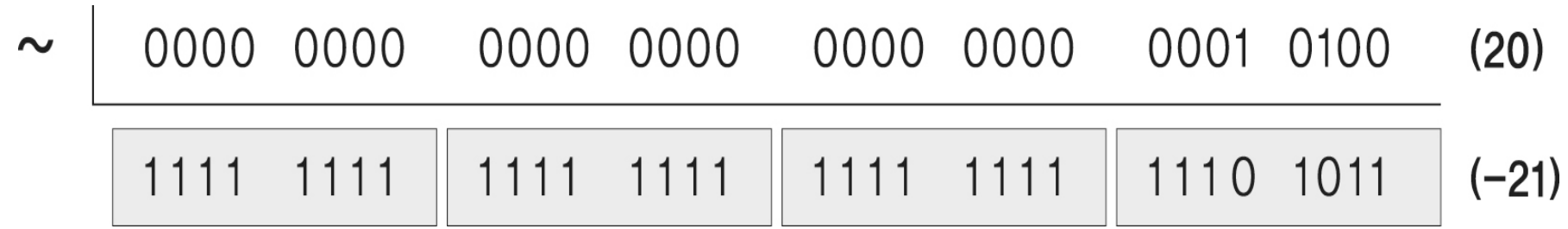
연산자	피연산자	결과
~	0	1
~	1	0

비트 연산자

```c
/* 5-16.c */
#include<stdio.h>
int main(void)
{
   int num1=20;
   int result1;

   result1 = ~num1;
   printf("비트단위 ~ 연산의 결과 %d \n",result1);   // 결과는 -21

   return 0;
}
```



비트 연산자

- **⑨ 비트 이동 연산자**
  - **<<연산자 (왼쪽 시프트 연산자)**
    - 비트를 왼쪽으로 이동시키는 연산자
  - **>>연산자 (오른쪽 시프트 연산자)**
    - 비트를 오른쪽으로 이동시키는 연산자

비트 연산자

```c
/* 5-17.c */
#include<stdio.h>
int main(void)
{
        int num1=10;
        int result1;

        result1 = num1 << 2;
        printf("비트단위 << 연산의 결과 %d \n", result1);

        return 0;
}
```

비트 연산자

```c
/* 5-18.c */
#include<stdio.h>
int main(void)
{
    int num1=10;
    int num2=-10;

    int result1;
    int result2;

    result1 = num1 >> 1;
    result2 = num2 >> 1;

    printf("비트단위 >> 연산의 결과 %d \n", result1);
    printf("비트단위 >> 연산의 결과 %d \n", result2);

    return 0;
}
```

연산자 우선순위

우선순위	연산자	연산 방향
1	() [] -> .	왼쪽에서 오른쪽
2	~ ~ ++ -- + - * &	오른쪽에서 왼쪽
3	* / %	왼쪽에서 오른쪽
4	+ -	왼쪽에서 오른쪽
5	<< >>	왼쪽에서 오른쪽
6	< <= > >=	왼쪽에서 오른쪽
7	== !=	왼쪽에서 오른쪽

우선순위	연산자	연산방향
8	&	왼쪽에서 오른쪽
9	^	왼쪽에서 오른쪽
10	|	왼쪽에서 오른쪽
11	&&	왼쪽에서 오른쪽
12	||	왼쪽에서 오른쪽
13	?:	오른쪽에서 왼쪽
14	= += -= *= /= %= &= ^= != <<= >>=	오른쪽에서 왼쪽
15	,	왼쪽에서 오른쪽

Summary

- 변수와 변수의 시작 주소
- 다양한 연산자
  - 대입 연산자, 산술 연산자, 복합 대입 연산자, 증감 연산자
  - 관계 연산자, 논리 연산자, 조건 연산자
- 비트, 바이트, 2진수, 10진수, 16진수
- 비트 연산자
  - & 연산자, | 연산자, ^ 연산자, ~ 연산자, << 연산자, >> 연산자
- 연산자 우선순위

1. 산술·대입·증감 연산자 심화

- **예제 1-1: 증가·감소 연산을 이용한 카운트다운**

```c
#include <stdio.h>

int main(void)
{
    int countdown = 5;

    printf("Start!\n");
    while(countdown > 0)
    {
        printf("%d...\n", countdown);
        countdown--; // 후위 감소 연산: 1씩 줄여서 0이 되면 종료
    }
    printf("Go!\n");

    return 0;
}
```

1. 산술·대입·증감 연산자 심화

- **예제 1-2: 복합 대입 연산과 누적 합**

```c
#include <stdio.h>

int main(void)
{
    int sum = 0;
    int input;

    printf("정수를 5번 입력하면, 누적 합계를 보여줍니다.\n");
    for(int i = 1; i <= 5; i++)
    {
        printf("%d번째 정수 입력: ", i);
        scanf("%d", &input);
        sum += input;  // 복합 대입 (sum = sum + input)
    }

    printf("입력한 5개 정수의 합은 %d입니다.\n", sum);
    return 0;
}
```

2. 관계·논리 연산자 확장

- **예제 2-1: 간단 자격 조건 판별**

```c
#include <stdio.h>

int main(void)
{
    int age;
    int hasTicket; // 0(없음), 1(있음)

    printf("나이를 입력하세요: ");
    scanf("%d", &age);

    printf("티켓이 있습니까? (1=예, 0=아니오): ");
    scanf("%d", &hasTicket);

    // 조건: 나이가 20 이상이고(hasTicket == 1)이면 입장 가능
    if(age >= 20 && hasTicket == 1)
        printf("입장 가능합니다.\n");
    else
        printf("입장 불가입니다.\n");

    return 0;
}
```

2. 관계·논리 연산자 확장

- **예제 2-2: 3개 중 최댓값(조건 연산자 중첩)**

```c
#include <stdio.h>

int main(void)
{
    int a, b, c;
    printf("정수 3개를 입력하세요: ");
    scanf("%d %d %d", &a, &b, &c);

    // 삼항 연산자(조건 연산자) 중첩 사용
    int max = (a > b) ?
                ((a > c) ? a : c) :
                ((b > c) ? b : c);

    printf("최댓값: %d\n", max);
    return 0;
}
```

3. 비트 연산자 확장

- **예제 3-1: XOR 스왑**

```c
#include <stdio.h>

int main(void)
{
    int x = 5, y = 10;
    printf("초기값: x=%d, y=%d\n", x, y);

    // XOR 연산을 이용해 임시 변수 없이 교환
    x = x ^ y;
    y = x ^ y; // (x ^ y) ^ y = x
    x = x ^ y; // (x ^ y) ^ x = y

    printf("교환 후: x=%d, y=%d\n", x, y);
    return 0;
}
```

3. 비트 연산자 확장

- **예제 3-2: 비트 마스킹**

```c
#include <stdio.h>

int main(void)
{
    unsigned int color = 0x123456;
    // RGB (24비트) 값 중에서 각각 R, G, B 추출
    // color = 0x RR GG BB

    unsigned int red   = (color >> 16) & 0xFF; // 최상위 8비트
    unsigned int green = (color >> 8)  & 0xFF; // 중간 8비트
    unsigned int blue  = color & 0xFF;         // 최하위 8비트

    printf("원본 color = 0x%x\n", color);
    printf("red = %u (0x%x)\n", red, red);
    printf("green = %u (0x%x)\n", green, green);
    printf("blue = %u (0x%x)\n", blue, blue);

    return 0;
}
```

4. 종합 문제

- **예제 4-1: 간단한 암호화/복호화**

```c
#include <stdio.h>

#define KEY 0x5A // 임의 비트 패턴 (암호화 키)

int main(void)
{
    unsigned int original;
    printf("양의 정수를 입력하세요: ");
    scanf("%u", &original);

    unsigned int encrypted = original ^ KEY;
    unsigned int decrypted = encrypted ^ KEY;

    printf("원본: %u, 암호화: %u, 복호화: %u\n", original, encrypted, decrypted);
    return 0;
}
```

1. 간단한 계산기 (산술, 복합 대입, 증감 연산자)

1. 기능
   - 사용자로부터 정수 두 개를 입력받는다(예: x, y).
   - 아래 연산들을 수행한 뒤, 결과를 각각 출력한다.
     ① x + y, x - y, x * y, x / y, x % y (산술 연산)
     ② x += y (복합 대입 후 x 출력)
     ③ x++, ++y (증감 연산 전후 값을 출력)

2. 요구사항
   - 정수 나눗셈(몫), 나머지 연산(%) 주의.
   - 복합 대입 연산(+=) 수행 전과 후를 비교해본다.
   - x++(후위), ++y(전위)를 각각 적용하여 결과를 관찰한다.

```
정수 2개를 입력하세요: 10 3

[산술 연산 결과]
10 + 3 = 13
10 - 3 = 7
10 * 3 = 30
10 / 3 = 3
10 % 3 = 1

[복합 대입]
x += y 실행 후, x = 13

[증감 연산]
x++ 실행 결과: x(출력 후) = 13, 실제 저장된 x = 14
++y 실행 결과: y = 4
```

2. 관계·논리 연산자 실습 (조건 판별)

1. 기능
   - 사용자로부터 점수(0 이상 100 이하)를 입력받는다.
   - 관계 연산자를 사용해 다음을 판별하고, 결과를 출력한다.
     1. 60점 이상인지? (합격/불합격)
     2. 90점 이상 100점 이하인지? (A 구간인지 여부)
     3. 0점 미만 또는 100점 초과인지? (입력 오류)
   - 논리 연산자(&&, ||, !)를 적극 활용한다.

2. 요구사항
   - 점수가 0 미만이거나 100 초과인 경우에는 "잘못된 점수"라는 경고 문구를 출력한다.
   - (선택) ?: 조건 연산자를 활용해 합격/불합격을 간단히 표현해도 좋다.

```
점수를 입력하세요: 95

입력받은 점수: 95
60점 이상인가? -> 합격
90점 이상 100점 이하인가? -> 네, A 구간입니다.
0점 미만 혹은 100점 초과인가? -> 아니오
```

3. 비트 연산자 (비트 단위 조작)

1. 기능
   - 사용자로부터 양의 정수(예: 10진수)를 입력받는다.
   - 아래와 같은 비트 연산 결과를 출력한다.
     1. ~num (비트 NOT)
     2. num << 1 (왼쪽 시프트 1)
     3. num >> 2 (오른쪽 시프트 2)
     4. (선택) num & 0xF, num | 0xF, num ^ 0xF 등 간단한 AND, OR, XOR 연산
2. 요구사항
   - 원본 수(num)와 결과 값을 10진수, 16진수 등으로 출력해보자.
   - 음수가 입력되면 어떻게 되는지도 실험해볼 수 있지만, 일단 양의 정수를 가정한다.

```
정수를 입력하세요: 10

입력값 (10진수) = 10, (16진수) = 0xa

~num (비트 NOT) = -11 (10진수)
num << 1 (왼쪽 시프트) = 20 (10진수), 0x14 (16진수)
num >> 2 (오른쪽 시프트) = 2 (10진수), 0x2 (16진수)
```