2장. 정보의 표현과 처리

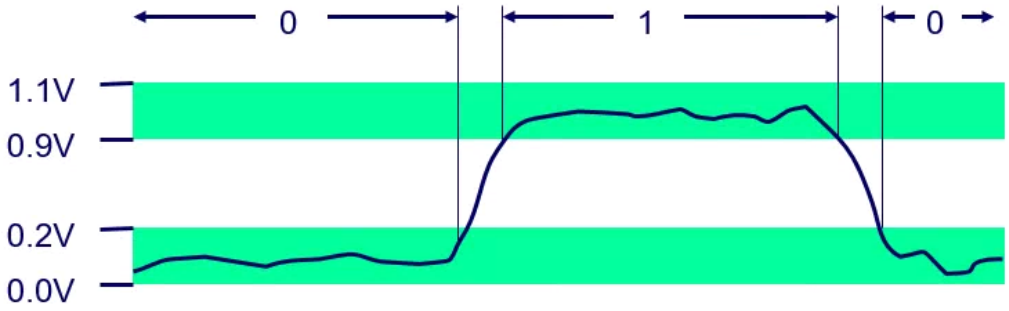
2.1 정보의 저장
2.2 정수의 표시
2.3 정수의 산술 연산

2.4 부동소수점

2.1 정보의 저장

비트 (Bits)

- 컴퓨터 시스템의 모든 정보는 비트로 저장
  - 비트들은 다양한 방식으로 인코딩/해석
  - 숫자, 문자열, 명령어
- 왜 비트로 표현하는가?
  - 전자회로는 간단하고 안정적으로 두 개의 값을 갖는 신호를 저장, 전송, 계산



바이트 (Byte)

- 바이트 = 8 비트
  - 메모리는 주소지정이 가능한 최소의 단위로 바이트를 사용
  - 2진수 (Binary)
    - 00000000_2 ~ 11111111_2
  - 10진수 (Deciaml)
    - 0_{10} ~ 255_{10}
  - 16진수 (Hexadecimal)
    - 00_{16} ~ FF_{16}
    - '0' ~ '9', 'A' ~ 'F'
    - C 언어에서 $FA1D37B_{16}$ 표현
      - 0x FA1D37B
      - 0xfa1d367b

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

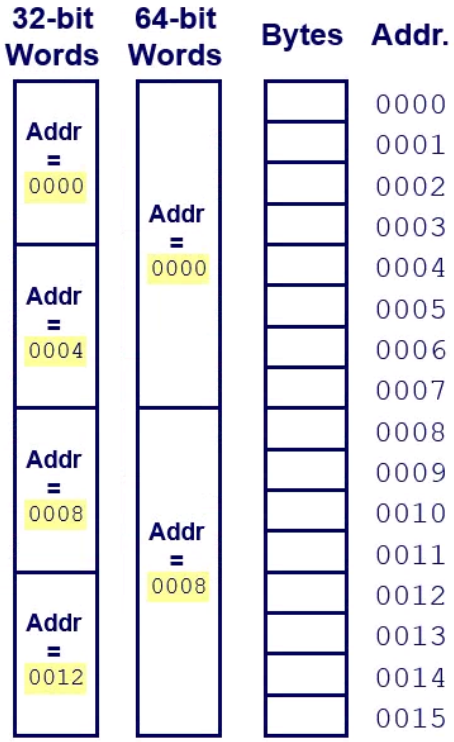
워드 (Word)

- 모든 컴퓨터는 워드 (word) 크기의 규격을 가짐
  - 하나의 가상주소가 한 개의 워드로 인코딩
    - C 포인터의 정규 크기
  - 워드의 크기는 가상 주소공간의 최대 크기를 결정
    - w 비트 워드 크기는 최대 2^w 바이트에 접근 (0 ~ 2^w-1)
  - 최근에는 32비트의 워드 크기에서 64비트의 워드 크기를 갖는 컴퓨터로의 전환이 보편화
    - 32비트 머신에서는 2^{32} (4 GB)의 가상 주소공간 크기
    - 64비트 머신에선는 2^{64} (16 EB)의 가상 주소공간 크기

Decimal term	Abbreviation	Value	Binary term	Abbreviation	Value	% Larger
kilobyte	KB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

워드 기반 메모리 구성

- 주소(address)는 바이트의 위치를 가리킴
  - 워드에서는 첫 바이트가 주소
  - 연속적인 워드의 주소는 4 (32 비트) 또는 8 (64 비트)의 배수

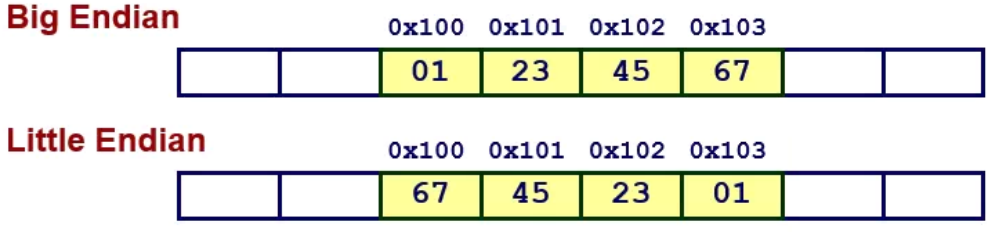


C 언어의 데이터 타입 크기

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	8	8
float	4	4	4
double	8	8	8
long double	–	–	10/16
pointer	4	8	8

주소 지정과 바이트 순서

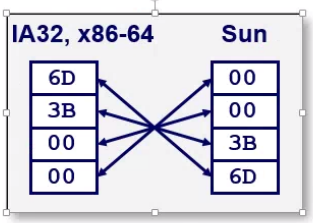
- 여러 개의 바이트들로 구성된 값들은 메모리에 연속된 바이트들로 저장하고, 주소는 사용된 바이트의 최소 주소로 지정
- 바이트의 순서 지정에 2가지 방식
  - 빅 엔디안 (Big Endian)은 첫 바이트(최소 주소)로 최상위(most significant) 바이트를 저장
    - IBM, Oracle (SUN) 머신들
  - 리틀 엔디안 (Little Endian)은 첫 바이트(최소 주소)로 최하위(least significant)바이트를 저장
    - 대부분의 Intel 호환 머신들
  - int 형의 변수 x의 주소(&x)는 0x100, 값은 0x01234567의 예



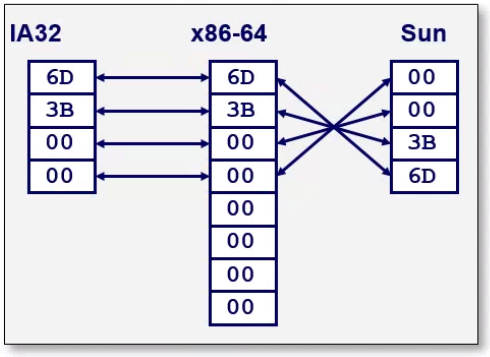
정수 바이트 순서 예

Decimal: 15213
Binary: 0011 1011 0110 1101
Hex: 3 B 6 D

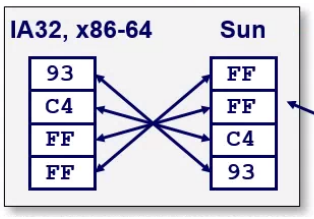
int A = 15213;



long int C = 15213;



int B = -15213;



Two's complement representation

정수 바이트 순서 실행 코드 예

```
/* show-bytes - prints byte representation of data */
#include <stdio.h>
typedef unsigned char *byte_pointer;

void show_bytes(byte_pointer start, size_t len) {
    size_t i;
    for (i = 0; i < len; i++)
            printf("%p₩t%.2x₩n", start+i, start[i]);
    printf("₩n");
}

int main()
{
   int a = 15213; // 0x3B6D
   printf("int a = 15213;₩n");
   show_bytes((pointer) &a, sizeof(int));

return 0;
}
```

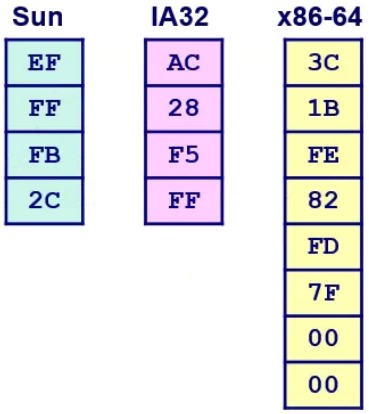
Result (Linux x86-64):

```
int a = 15213;
0x7fffb7f71dbc    6d
0x7fffb7f71dbd    3b
0x7fffb7f71dbe    00
0x7fffb7f71dbf    00
```

포인터 저장

- 데이터들의 메모리 위치(주소, 포인터)는 컴파일러, 머신들에 따라 다름
  - 프로그램 실행할 때마다 위치가 다를 수도 있음

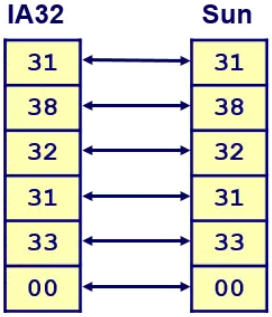
```
int B = -15213;
int *P = &B;
```



문자열(Stirng) 저장

- C 언어의 문자열
  - 문자들의 배열로 표현
    - 연속적으로 저장
  - 각 문자는 ASCII (또는 유니코드)로 인코딩
  - 문자열의 끝은 널(null, 0) 값
  - 바이트 순서와 무관
    - 항상 낮은 주소부터 첫 문자 저장

```
char S[6] = "18213";
```



과제 2-1: 데이터 객체들의 바이트 표시 (실습과제)

- 앞의 show_bytes 코드를 참조하여 아래와 같이 각 데이터들의 바이트 표시 C 프로그램을 작성, 컴파일, 실행
  - 임의의 정수, 실수(flaot, double), 문자열, 포인터 등의 데이터 객체 값에 대한 바이트를 표시
  - 64비트, 32비트 머신 버전에 대해 각각 실행
    - 64비트 머신에서 gcc 컴파일 시 '-m32' 옵션을 지정

부울 대수 (Boolean Algebra)

- 부울 대수는 TRUE와 FALSE로 표현되는 논리 추론의 기본 원리들의 대수학
  - 1850년 George Boole에 의해 개발
  - 1937년 Claude Shannon이 부울대수를 디지털 논리회로에 적용

And

- A&B = 1 when both A=1 and B=1

&	0	1
0	0	0
1	0	1

Or

- A|B = 1 when either A=1 or B=1

|	0	1
0	0	1
1	1	1

Not

- ~A = 1 when A=0

~	
0	1
1	0

Exclusive-Or (Xor)

- A^B = 1 when either A=1 or B=1, but not both

^	0	1
0	0	1
1	1	0

부울 대수 – 비트 벡터 (Bit Vectors)

- 부울 대수 연산들은 비트 벡터에도 확장 적용
  - 각 비트 단위로 연산이 적용

```
  01101001     01101001     01101001
& 01010101   | 01010101   ^ 01010101   ~ 01010101
  01000001     01111101     00111100     10101010
```


집합의 표현과 연산

- 비트 벡터를 사용하여 유한집합(set)을 표현하고 연산 적용
  - w 비트의 벡터는 {0, ..., w-1}의 부분 집합을 표시
  - $a_j = 1$ if $j \in A$
    - 01101001 { 0, 3, 5, 6 }
    - *76543210*
    - 01010101 { 0, 2, 4, 6 }
    - *76543210*
  - 연산
    - & 교집합 (intersection) 01000001 { 0, 6 }
    - | 합집합 (union) 01111101 { 0, 2, 3, 4, 5, 6 }
    - ^ 대칭차집합 (symmetric difference) 00111100 { 2, 3, 4, 5 }
    - ~ 여집합 (complement) 10101010 { 1, 3, 5, 7 }

C 언어의 비트수준 연산

- C 언어는 비트들 간의 부울 연산을 지원
  - &, |, ~, ^ 연산자
  - 정수형 데이터에 적용
    - long, int, short, char, unsigned
  - char 데이터 타입에서 수식 계산 예
    - ~0x41 → 0xBE
      - $\sim 01000001_2 \rightarrow 10111110_2$
    - ~0x00 → 0xFF
      - $\sim 00000000_2 \rightarrow 11111111_2$
    - 0x69 & 0x55 → 0x41
      - $01101001_2 \,\&\, 01010101_2 \rightarrow 01000001_2$
    - 0x69 | 0x55 → 0x7D
      - $01101001_2 \mid 01010101_2 \rightarrow 01111101_2$

과제 2-2: 비트수준 연산 (실습과제)

- 숙제문제 2.59 (p.124)
  - x의 최하위(least significant) 바이트와 y의 나머지 바이트들로 이루어진 워드를 만드는 C 수식을 작성하고 아래의 값에 대해 실행
    - 오퍼랜드 x=0x89ABCDEF, y=0x76543210
      => 0x765432EF
  - 위의 수식을 두 개의 오퍼랜드를 인수로 받는 함수로 작성하고, 임의의 10개의 값들의 오퍼랜드 쌍을 인수로 전달하고 결과를 축력하도록 수정

C 언어의 논리 연산

- C 언어 의 논리 연산은 0은 거짓(false), 0이 아닌 값은 을 참(True)을 처리
  - &&, ||, ! 연산자
  - 조기에 연산 종료
    - a = 0;
      a && 5/a; <- 0으로 나누는 에러 발생하지 않음
  - char 데이터 타입 적용 예
    - !0x41 → 0x00
    - !0x00 → 0x01
    - !!0x41 → 0x01
    - 0x69 && 0x55 → 0x01
    - 0x69 || 0x55 → 0x01
    - p && *p (널 포인터의 접근을 피함)

C 언어의 쉬프트 연산

- 왼쪽 쉬프트: x << y
  - x를 왼쪽으로 y만큼 이동
  - 좌측의 y개 비트는 버려짐
  - 우측은 y개의 0으로 채워짐
- 오른쪽 쉬프트: x >> y
  - x를 오른쪽으로 y만큼 이동
  - 우측의 y개 비트는 버려짐
  - 논리 쉬프트 (Logical Shift)
    - 좌측의 y개는 0으로 채워짐
  - 산술 쉬프트 (Arithmetic Shift)
    - 좌측의 y개는 최상위 비트 값으로 채워짐

Argument x	01100010
<< 3	00010*000*
Log. >> 2	*00*011000
Arith. >> 2	*00*011000

Argument x	10100010
<< 3	00010*000*
Log. >> 2	*00*101000
Arith. >> 2	*11*101000

2.2 정수의 표시

정수의 인코딩

- w비트 워드에 대해 비부호형(unsigned) 정수와 부호형(singned) 정수로 표시
  - 부호형 정수에서 음수는 2의 보수(two's complement)로 표시
    - 최상위 비트가 부호 비트 (0은 양수, 1은 음수)
  - <그림 2.8> 용어 참조

Unsigned

$$B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i$$

Two's Complement

$$B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$$

```
short int x =  15213;
short int y = -15213;
```

Sign Bit

- C short 2 bytes long

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
y	-15213	C4 93	11000100 10010011

2의 보수 인코딩 예

```
x =      15213: 00111011 01101101
y =     -15213: 11000100 10010011
```

Weight	15213		-15213	
1	1	1	1	1
2	0	0	1	2
4	1	4	0	0
8	1	8	0	0
16	0	0	1	16
32	1	32	0	0
64	1	64	0	0
128	0	0	1	128
256	1	256	0	0
512	1	512	0	0
1024	0	0	1	1024
2048	1	2048	0	0
4096	1	4096	0	0
8192	1	8192	0	0
16384	0	0	1	16384
-32768	0	0	1	-32768
Sum		**15213**		**-15213**

숫자의 범위

- **Unsigned Values**
  - *UMin* = 0
    000…0
  - *UMax* = $2^w - 1$
    111…1
- **Two's Complement Values**
  - *TMin* = -2^{w-1}
    100…0
  - *TMax* = $2^{w-1} - 1$
    011…1
- **Other Values**
  - Minus 1
    111…1

Values for *W* = 16

	Decimal	Hex	Binary
UMax	**65535**	FF FF	11111111 11111111
TMax	**32767**	7F FF	01111111 11111111
TMin	**-32768**	80 00	10000000 00000000
-1	**-1**	FF FF	11111111 11111111
0	**0**	00 00	00000000 00000000

워드 크기에 따른 숫자 범위

	W			
	8	16	32	64
UMax	255	65,535	4,294,967,295	18,446,744,073,709,551,615
TMax	127	32,767	2,147,483,647	9,223,372,036,854,775,807
TMin	-128	-32,768	-2,147,483,648	-9,223,372,036,854,775,808

- 주요 특징
  - $|TMin| = TMax + 1$
    - 비대칭
  - $UMax = 2 * TMax + 1$
- C 언어
  - #include <limits.h>
  - 상수로 정의
    - ULONG_MAX
    - LONG_MAX
    - LONG_MIN

비부호형과 부호형 수

X	B2U(*X*)	B2T(*X*)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	−8
1001	9	−7
1010	10	−6
1011	11	−5
1100	12	−4
1101	13	−3
1110	14	−2
1111	15	−1

비부호형과 부호형 간의 변환

Bits	Signed		Unsigned
0000	0		0
0001	1		1
0010	2		2
0011	3	=	3
0100	4		4
0101	5		5
0110	6		6
0111	7		7
1000	-8		8
1001	-7		9
1010	-6		10
1011	-5	+/- 16	11
1100	-4		12
1101	-3		13
1110	-2		14
1111	-1		15

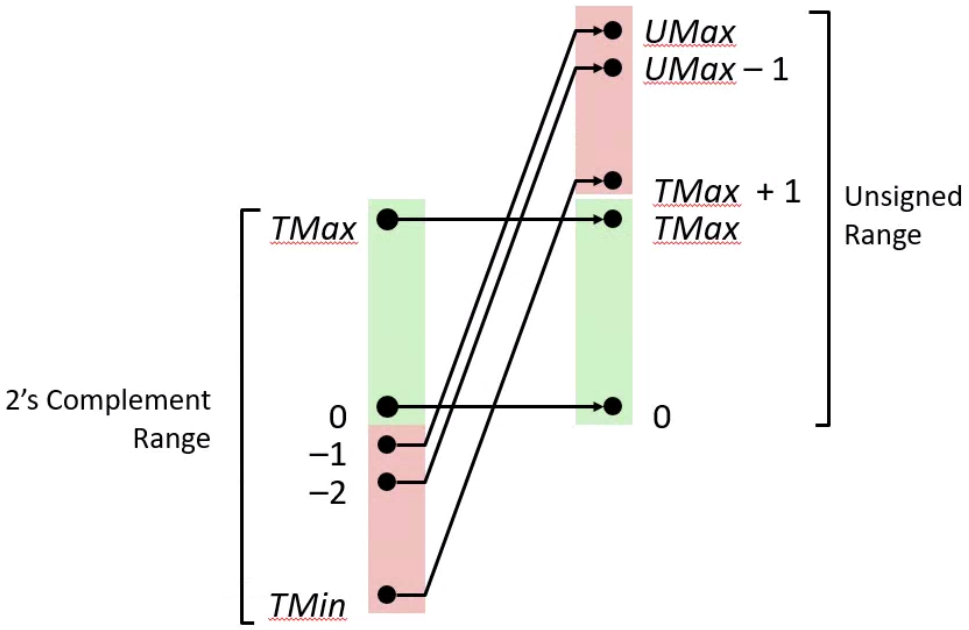
부호형과 비부호형 값 변환 예

- p.68 <그림 2.15>의 변환 예
  - 0x3039 이진값
    - 부호형 -12345, 비부호형 53191

```
short    int        v = -12345;
unsigned short uv = (unsigned short) v;
printf("v = %d, uv = %u=₩n", v, uv)
```

- v = -12345, uv = 53191

2의 보수에서 비부호형으로 변환



<그림 2.17>

C 언어의 부호형과 비부호형 비교

- 상수 (constant)
  - 디폴트로 부호형 정수로 간주
  - U 가 뒤에 오면 비부호형 싱수
    - 0U, 4294967259U
- 형변환(캐스팅, casting)
  - 부호형과 비부호형 형변환 시 기본 비트 표시는 바뀌지 않음
    - 해석은 다름

```
int  tx, ty;
unsigned ux, uy;
tx = (int) ux;
uy = (unsigned) ty;
```

  - 묵시적 형변환 (implicit casting)

```
tx = ux;
uy = ty;
```

형변환 효과

- 수식의 처리 (expression evaluation)
  - 단일 식에 부호형과 비부호형이 혼합되었을 때에는 묵시적으로 부호형이 비부호형으로 변환
  - <, >, ==, <=, >= 연산자에도 적용
  - W=32 예: TMIN = -2,147,483,648 , TMAX = 2,147,483,647

$Constant_1$	$Constant_2$	Relation	Evaluation
0	0U	==	unsigned
-1	0	<	signed
-1	0U	>	unsigned
2147483647	-2147483647-1	>	signed
2147483647U	-2147483647-1	<	unsigned
-1	-2	>	signed
(unsigned)-1	-2	>	unsigned
2147483647	2147483648U	<	unsigned
2147483647	(int) 2147483648U	>	signed

과제 2-3: 형변환 효과 (실습과제)

- 연습문제 2.25 (p.81)
  - 배열 a의 원소들의 합을 계산하는 코드 예에 대해 아래와 같이 실행
    - length > 0 인 정상적인 배열 값들에 대해 실행하여 결과 출력
    - lnegth = 0 인 값을 인수로 전달하여 실행 시 발생하는 에러를 관찰하고 이유 설명
    - 위의 에러를 수정하고 실행한 결과 출력

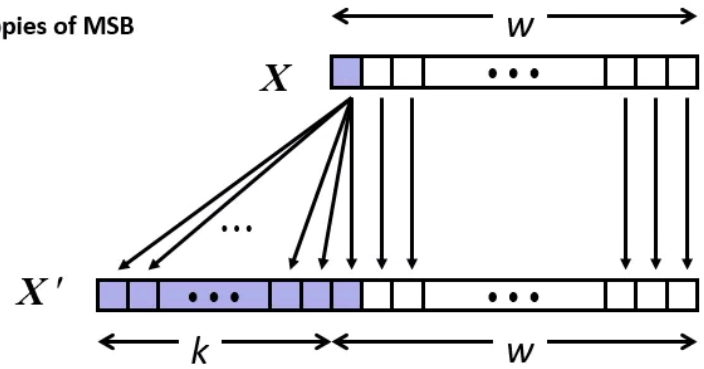
부호 확장 (Sign Extension)

❑ 부호 확장

- 부호형 변수 x의 값은 유지하면서 w 비트(작은 길이)의 x를 w+k 비트(더 큰 길이)로 변환
- 부호 비트를 k 비트만큼 복사하여 확장

$$X' = x_{w-1}, \ldots, x_{w-1}, x_{w-1}, x_{w-2}, \ldots, x_0$$

(x_{w-1} 이 k copies of MSB)



부호 확장 예

❑ 작은형의 정수 데이터 타입(short)에서 큰 형의 정수 데이터 타입(int)으로 변환

- C 언어는 자동으로 부호 확장을 수행

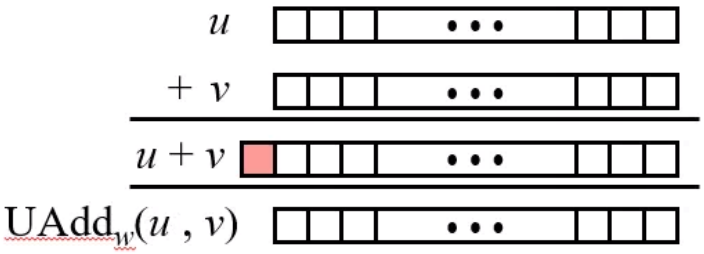
```
short int x =  15213;
int      ix = (int) x;
short int y = -15213;
int      iy = (int) y;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 3B 6D	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

2.3 정수의 산술연산

비부호형 덧셈 (Unsigned Addition)

- w 비트의 비부호형 덧셈
  - w+1 비트 크기의 합을 생성
  - 캐리 (w+1 번째 비트)는 버림

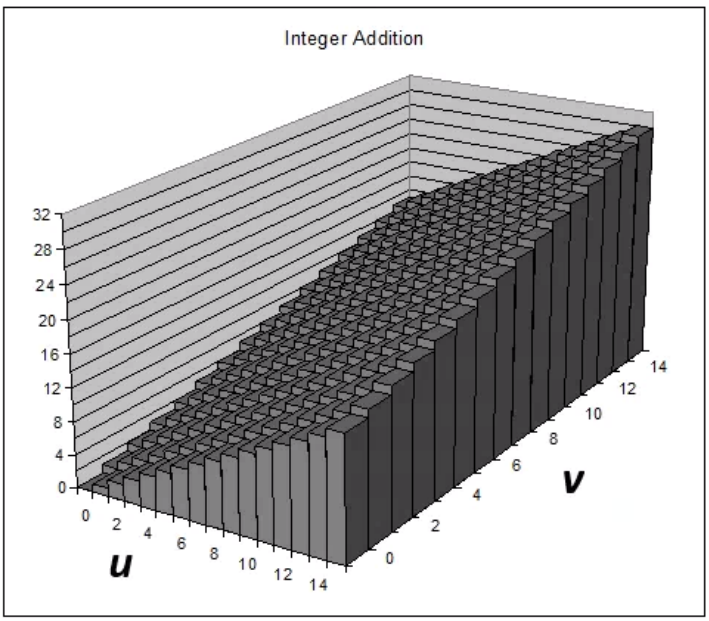


  - 모듈로 연산(Modular Arithmetic) 으로도 구현

$$s = \mathrm{UAdd}_w(u, v) = u + v \bmod 2^w$$

4비트 워드 크기의 비부호형 정수 덧셈

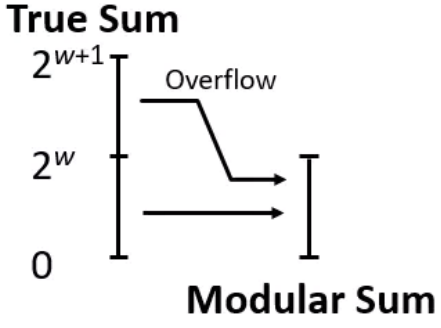
$\mathbf{Add_4}(u , v)$

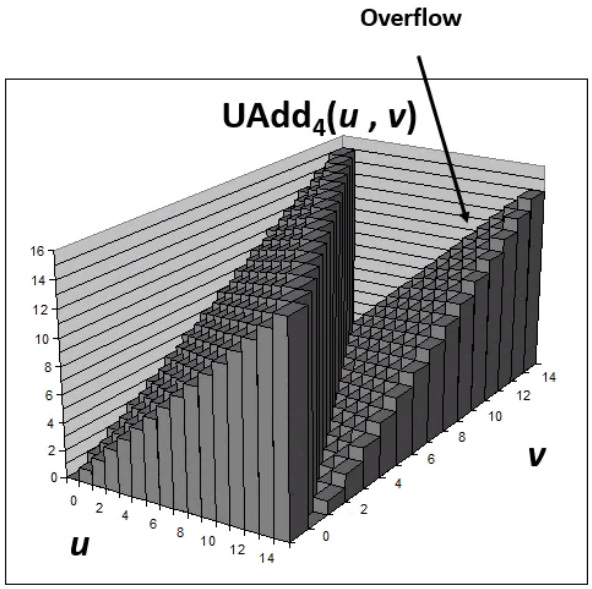


<그림 2.21>

4비트 워드 크기의 비부호형 정수 덧셈 – 오버플로우(Overflow)

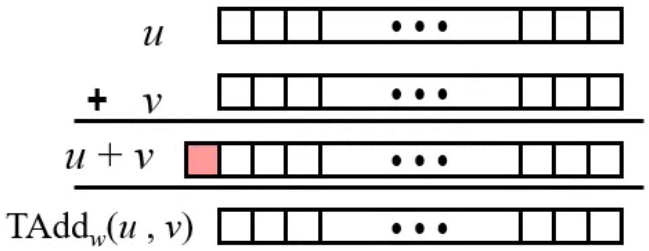
- 합의 결과가 w 비트의 워드 크기로만 표시
  - 합이 2^w 보다 크거나 같으면 수의 범위를 벗어난 오버플로우
  - 실제 합(true sum)이 모듈로 합(modular sum)으로 표시





부호형 덧셈 (Signed Addition) – 2의 보수 덧셈

- w 비트의 부호형 덧셈
  - w+1 비트 크기의 합을 생성
  - 음수는 2의 보수로 표현
  - 캐리 (w+1 번째 비트)는 버림



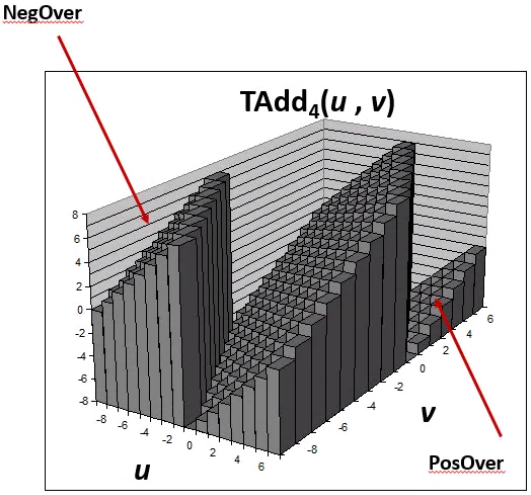
- 비부호형 덧셈과 부호형 덧셈의 비트-수준 동작은 같음

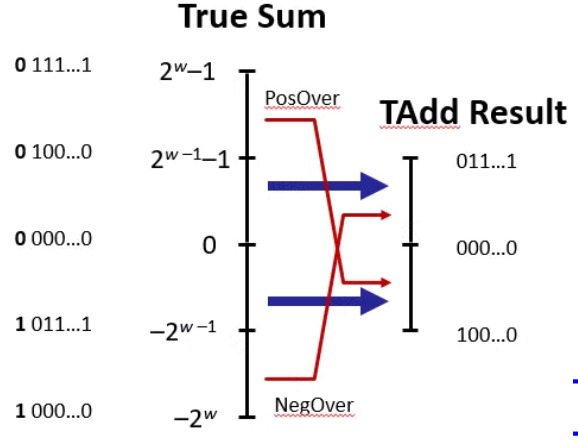
```
int s, t, u, v;
s = (int) ((unsigned) u + (unsigned) v);
t = u + v
```

  - 결과: s == t

부호형 덧셈의 오버플로우

- 두 종류의 오버플로우
  - 양의 오버플로우 (positive overflow)
    - sum >= 2^{w-1}
  - 음의 오버플로우 (negative overflow)
    - sum < -2^{w-1}





과제 2-4: 정수의 덧셈(실습과제)

- 연습문제 2.30 (p.91) 변형
  - 16비트 정수형(short int)에 대해 덧셈 결과와 <그림 2.24>의 케이스를 판별하는 코드를 작성하고 실행
    - 4가지 경우의 예를 실행
    - 덧셈의 실행 결과를 출력 (오버플로우 포함)

곱셈 (Multiplication)

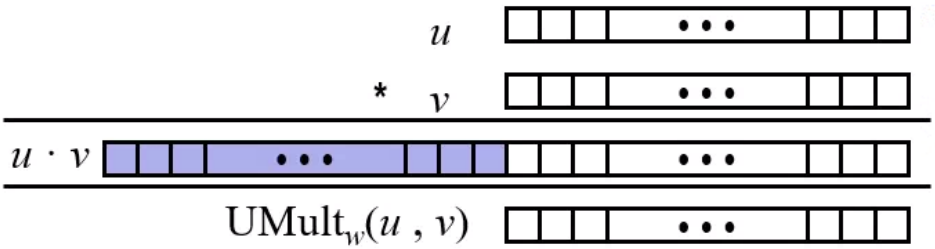
- w 비트의 두 수 x, y의 곱셈 (비부호형/부호형)

- 곱셈의 결과가 w 비트 보다 클 수 있음
  - 비부호형: 최대 2w 비트 크기
    - 결과의 범위: $0 \le x * y \le (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1$
  - 2의 보수 최소값(음수): 2w-1 비트 크기
    - 결과의 범위: $x * y \ge (-2^{w-1}) * (2^{w-1}-1) = -2^{2w-2} + 2^{w-1}$
  - 2의 보수 최대값(양수): 2w 비트 크기
    - 결과의 범위: $x * y \le (-2^{w-1})^2 = 2^{2w-2}$

- w 비트를 초과(오버프로우)하는 곱셈의 결과를 위해서는 별도의 소프트웨어 구현이 필요

C 언어의 비부호형 곱셈

- w 비트 워드 크기의 비부호형 곱셈
  - 2w 비트의 곱셈 결과를 생성
  - 상위 w비트를 버림

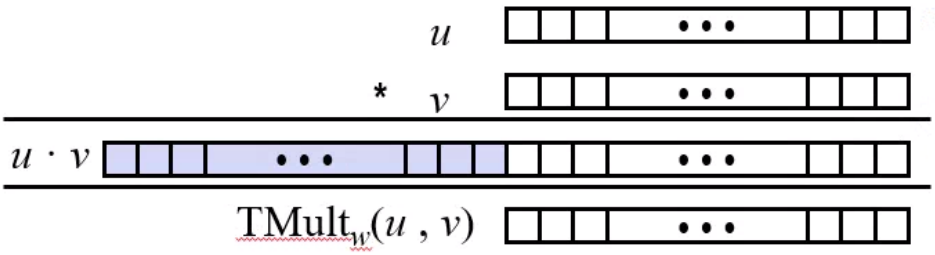


  - 모듈로 연산의 구현

$$\text{UMult}_w(u, v) = u \cdot v \bmod 2^w$$

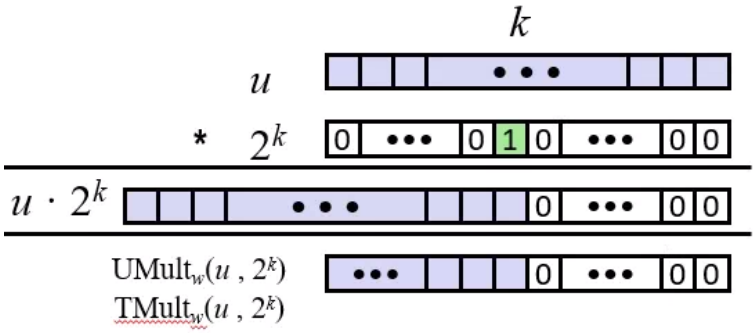
C 언어의 부호형 곱셈

- w 비트 워드 크기의 부호형 곱셈
  - 2w 비트의 곱셈 결과를 생성
  - 상위 w비트를 버림



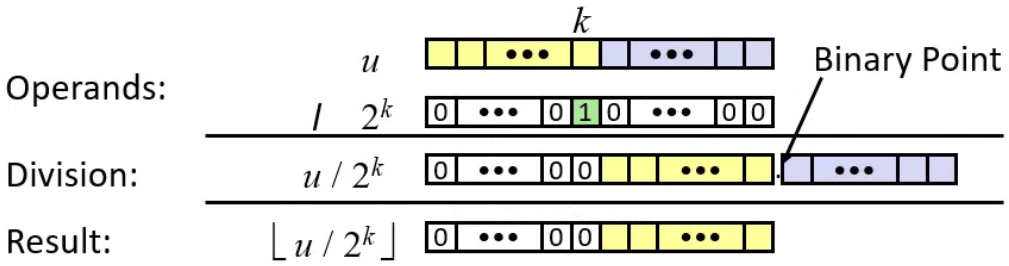
쉬프트 연산을 이용한 곱셈

- 정수의 곱셈은 10개 이상의 클럭 사이클 소요
- 2의 멱승의 곱셈은 간단히 쉬프트 연산(1 사이클)으로 구현
  - u << k 는 u * 2^k의 결과 생성
    - u << 3 = u * 2^3 = u * 8
  - 부호형/비부호형 모두 적용



쉬프트 연산을 이용한 비부호형 나눗셈

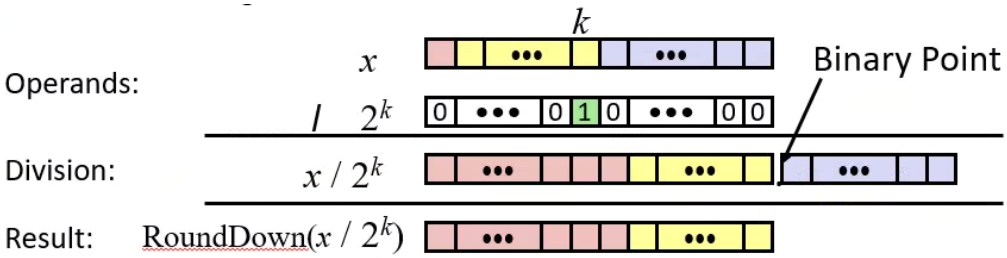
- 정수의 나눗셈은 30 클럭 사이클 이상 소요
- 2의 멱승의 나눗셈의 몫(quotient)
  - u >> k는 u/2^k의 몫 생성
  - 논리 쉬프트 적용



	Division	Computed	Hex	Binary
x	15213	15213	3B 6D	00111011 01101101
x >> 1	7606.5	7606	1D B6	00011101 10110110
x >> 4	950.8125	950	03 B6	00000011 10110110
x >> 8	59.4257813	59	00 3B	00000000 00111011

쉬프트 연산을 이용한 부호형 나눗셈

- 2의 멱승의 나눗셈의 몫(quotient)
  - u >> k는 $u/2^k$의 몫 생성
  - 산술 쉬프트 적용
  - u < 0인 경우 반올림(실질적으로는 내림,방향이 반대)



	Division	Computed	Hex	Binary
y	-15213	-15213	C4 93	11000100 10010011
y >> 1	-7606.5	-7607	E2 49	11100010 01001001
y >> 4	-950.8125	-951	FC 49	11111100 01001001
y >> 8	-59.4257813	-60	FF C4	11111111 11000100

요 약

요약

- 컴퓨터는 정보를 비트로 인코딩
  - 숫자, 문자열, 명령어로 인코딩
  - 다중 바이트의 데이터 순서 지정를 위해 빅 엔디안, 리틀 엔디안 모델
- 대부분의 머신들이 정수 인코딩에 2의 보수, 부동소수점 인코딩에 IEEE 표준 754를 사용
- C 언어는 워드 길이와 숫자 인코딩에 다양한 방법을 수용할 수 있도록 설계
  - 동일한 크기를 갖는 부호형, 비부호형 정수 캐스팅 시 내부 비트 패턴은 불변
- 컴퓨터의 정수와 실수의 산술 연산 시 표현 범위를 넘어서는 오버플로우 발생

과 제

과제 2-1: 데이터 객체들의 바이트 표시 (실습과제)

- 앞의 show_bytes 코드를 참조하여 아래와 같이 데이터들의 바이트 표시 C 프로그램을 작성, 컴파일, 실행
  - 임의의 정수, 실수(flaot, double), 문자열, 포인터 등의 데이터 객체 값에 대한 바이트를 표시
  - 64비트, 32비트 머신 버전에 대해 각각 실행
    - 64비트 머신에서 gcc 컴파일 시 '-m32' 옵션을 지정

과제 2-2: 비트수준 연산 (실습과제)

- 숙제문제 2.59 (p.124)
  - x의 최하위(least significant) 바이트와 y의 나머지 바이트들로 이루어진 워드를 만드는 C 수식을 작성하고 아래의 값에 대해 실행
    - 오퍼랜드 x=0x89ABCDEF, y=0x76543210
      => 0x765432EF
  - 위의 수식을 두 개의 오퍼랜드를 인수로 받는 함수로 작성하고, 임의의 10개의 값들의 오퍼랜드 쌍을 인수로 전달하고 결과를 축력하도록 수정

과제 2-3: 형변환 효과 (실습과제)

- 연습문제 2.25 (p.81)
  - 배열 a의 원소들의 합을 계산하는 코드 예에 대해 아래와 같이 실행
    - length > 0 인 정상적인 배열 값들에 대해 실행하여 결과 출력
    - lnegth = 0 인 값을 인수로 전달하여 실행 시 발생하는 에러를 관찰하고 이유 설명
    - 위의 에러를 수정하고 실행한 결과 출력

과제 2-4: 정수의 덧셈(실습과제)

- 연습문제 2.30 (p.91) 변형
  - 16 비트 정수형(short int)에 대해 덧셈 결과와 <그림 2.24>의 케이스를 판별하는 코드를 작성하고 실행
    - 4가지 경우의 예를 실행
    - 덧셈의 실행 결과를 출력 (오버플로우 포함)