

# 시스템 프로그래밍 (2016)

장경선

충남대학교 컴퓨터공학과

Bryant and O'Hallaron, Computer Systems: Programmer's Perspective에서  
발췌해 온 저작권이 있는 내용들이 포함되어 있으므로,  
시스템프로그래밍 강의 수강 이외 용도로 사용할 수 없음.



# 시스템 프로그래밍

---

강의 2 : 2장. 정보의 표현 및 처리 I  
2.1, 2.2, 2.3 정수의 표현 및 연산



# 강의 일정

주	날짜	강의실	날짜	실습실
1	9월 1일(목)	소개 강의	9월 6일(화)	리눅스 개발환경 익히기 (VI, 셸 기본명령어들)
2	9월 8일(목)	정수 표현 방법	9월 13일(화)	GCC & Make, shell script
3	9월 15일(목)	추석 휴강	9월 20일(화)	C 복습과 GDB 사용하기 1 (소스 수준 디버깅)
4	9월 22일(목)	실수 표현 방법	9월 27일(화)	Data lab (GDB활용)
5	9월 29일(목)	어셈1 - 데이터이동	10월 4일(화)	어셈1 - move(실습),
6	10월 6일(목)	어셈2 - 제어문	10월 11일(화)	어셈2- 제어문 (실습)
7	10월 13일(목)	어셈3 - 프로시저	10월 18일(화)	어셈3-프로시저(실습)
8	10월 20일(목)	어셈보충/중간시험	10월 25일(화)	GDB 사용하기2(어셈수준)
9	10월 27일(목)	보안(buffer overflow)	11월 1일(화)	Binary bomb 1 (GDB활용)
10	11월 3일(목)	프로세스 1	11월 8일(화)	Binary bomb 2 (GDB활용)
11	11월 10일(목)	프로세스 2	11월 15일(화)	Tiny shell 1
12	11월 17일(목)	시그널	11월 22일(화)	Tiny shell 2
13	11월 24일(목)	동적메모리 1	11월 29일(화)	Malloc lab1
14	12월 1일(목)	동적메모리 2	12월 6일(화)	Malloc lab2
15	12월 8일(목)	기말시험	12월 13일(화)	Malloc lab3



# 배울 내용: 정수의 표현

- 바이트 인코딩, 워드, 데이터 크기(C에서)
- 바이트 순서(endianism, 2byte이상의 data)
- C 연산자, 비트수준연산자, 논리연산자, 쉬프트 연산자
- 정수의 표현 방법(2,8,10,16진수, 2의 보수, signed, unsigned)
- Casting (unsigned, signed 간 캐스팅)
- 제로/부호 확장, 덧셈(signed, unsigned)
- 곱셈과 쉬프트, 나눗셈과 쉬프트




기초...

# 바이트 인코딩, 워드, 데이터의 크기

# 바이트 값의 인코딩

- 1바이트 Byte = 8 bits
  - 이진수  $00000000_2$  to  $11111111_2$
  - 십진수:  $0_{10}$  to  $255_{10}$
  - 16진수  $00_{16}$  to  $FF_{16}$ 
    - '0' to '9' and 'A' to 'F' 사용
    - $FA1D37B_{16}$  는 C에서 다음과 같이 표시
      - $0xFA1D37B$
      - $0xfa1d37b$



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

# 컴퓨터의 워드길이

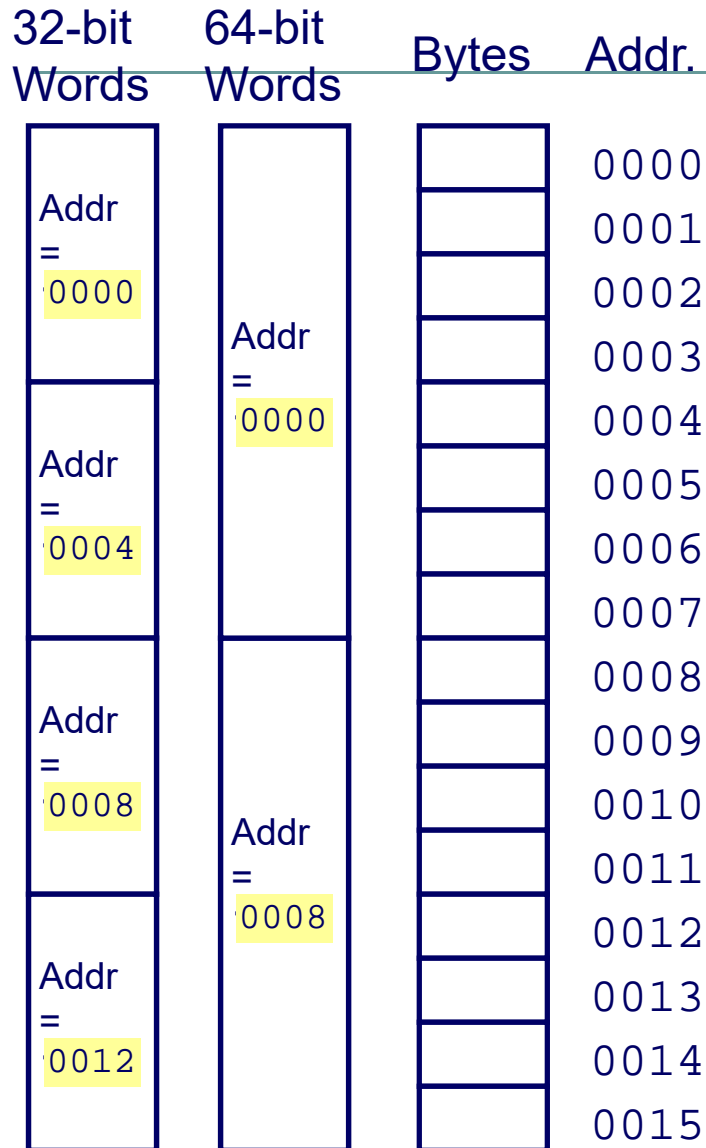
## Machine Words



- 컴퓨터는 워드길이 Word Size
  - 정수 값의 크기를 말한다
    - 주소의 길이가 되기도 한다
  - 대부분의 요즘 컴퓨터는 32비트(4바이트) 워드이다
    - 이로 인해 주소범위가 4GB로 제한된다.
    - 메모리가 많이 필요한 프로그램에서는 제약이 될 수 있다
  - 최근 데스크 탑 컴퓨터는 64비트 워드를 사용한다
    - 가용 주소 공간  $\approx 1.8 \times 10^{19}$  bytes
    - x86-64 컴퓨터는 48비트 주소를 지원한다: 256 테라바이트
  - 컴퓨터는 다양한 데이터 타입을 지원한다
    - 워드의 일부분 또는 여러 워드 길이의 데이터 타입
    - 모든 데이터 타입은 바이트의 배수를 길이로 갖는다.



# 워드 기반 메모리 구조



- 주소는 메모리에서 바이트의 위치를 지정
  - 워드의 첫번째 바이트의 위치를 지정
  - 연속된 워드의 주소는 4 또는 8씩 증가한다





# 데이터의 표시/크기

C Data Type	Typical 32-bit	Intel IA32	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	4	8
long long	8	8	8
float	4	4	4
double	8	8	8
long double	8	10/12	10/16
pointer	4	4	8



바이트 순서(2바이트 이상의 데이터 표현에서...)

# 바이트 저장 순서 Byte Ordering



- 여러 바이트로 이루어진 데이터는 어떤 순서로 저장되는가의 문제
- Sun, Mac : "Big Endian"
  - LSB가 최대 주소의 위치에 기록된다
- Alpha, PC : "Little Endian"
  - LSB가 최소 주소의 위치에 기록된다

# Byte Ordering 예제

Byte Ordering 은  
언제 문제가 될까?

- Big Endian

- Least significant byte 가 최대 주소에 저장됨

- Little Endian

- Least significant byte 가 최소 주소에 저장됨

- Example

- 변수  $x$  는 다음과 같은 4 바이트의 워드이다

0x01234567

- $x$ 의 주소  $\&x$  는 현재 0x100 이다

Big Endian

		0x100	0x101	0x102	0x103		
		01	23	45	67		

Little Endian

		0x100	0x101	0x102	0x103		
		67	45	23	01		



# 기계어 해독하기 Disassembly

## ● Disassembly 예제

Address	Instruction Code	Assembly Rendition
8048365:	5b	pop %ebx
8048366:	81 c3 <u>ab 12 00 00</u>	add \$0x12ab,%ebx
804836c:	83 bb 28 00 <u>00 00 00</u>	cmpl \$0x0,0x28(%ebx)

### Little Endian의 해독과정

- 값: 0x12ab
- 4 바이트로 패딩 padding 하기 : 0x000012ab
- 바이트로 나누기 : 00 00 12 ab
- 뒤집기 (왜?): ab 12 00 00

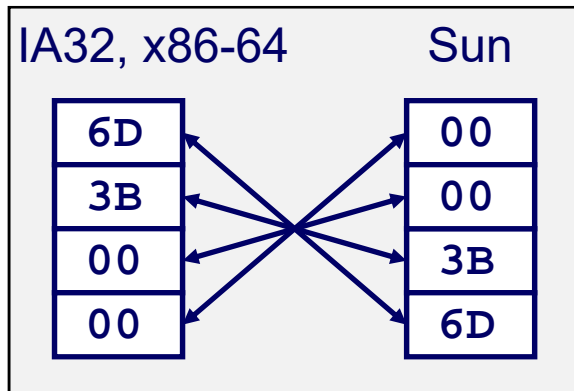
# 정수 표현하기

Decimal: 15213

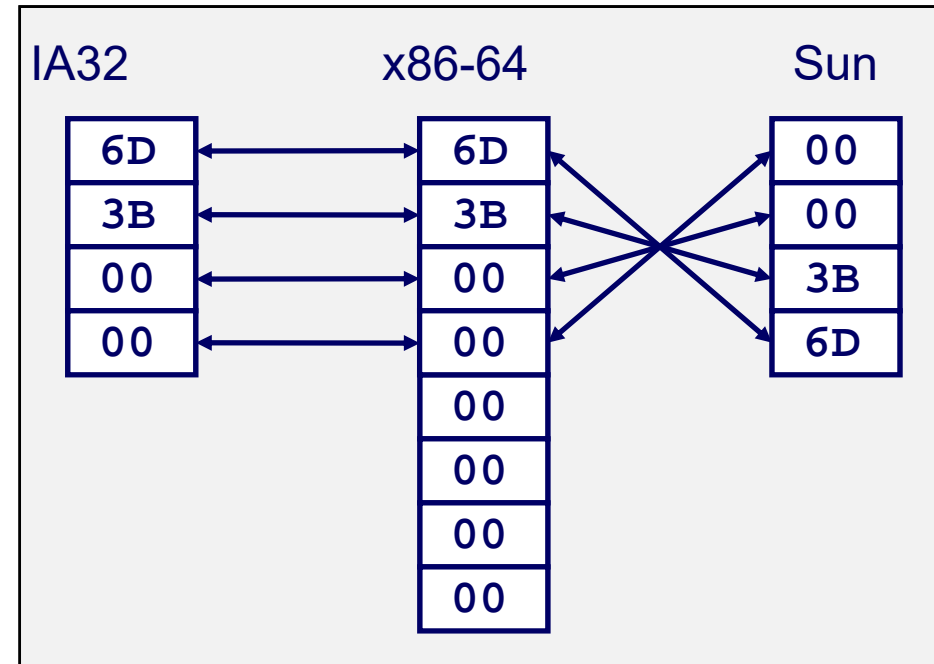
Binary: 0011 1011 0110 1101

Hex: 3 B 6 D

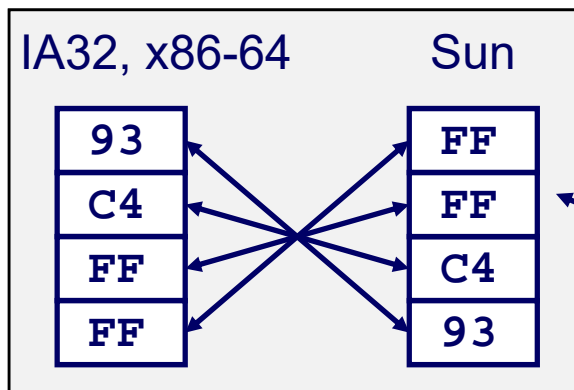
```
int A = 15213;
```



```
long int C = 15213;
```



```
int B = -15213;
```



Two's complement representation



# 리눅스에서 바이트의 출력

- 데이터를 바이트로 출력해주는 프로그램
  - unsigned char \* 는 바이트 배열을 만든다

```
typedef unsigned char *pointer;

void show_bytes(pointer start, int len)
{
    int i;
    for (i = 0; i < len; i++)
        printf("0x%p\t0x%.2x\n",
               start+i, start[i]);
    printf("\n");
}
```

**Printf directives:**

**%p:** Print pointer

**%x:** Print Hexadecimal



# show\_bytes 실행결과

```
int a = 15213;  
printf("int a = 15213;\n");  
show_bytes((pointer) &a, sizeof(int));
```

## Result (Linux x86-64):

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





# 포인터 표현

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

Sun

EF
FF
FB
2C

IA32

AC
28
F5
FF

x86-64

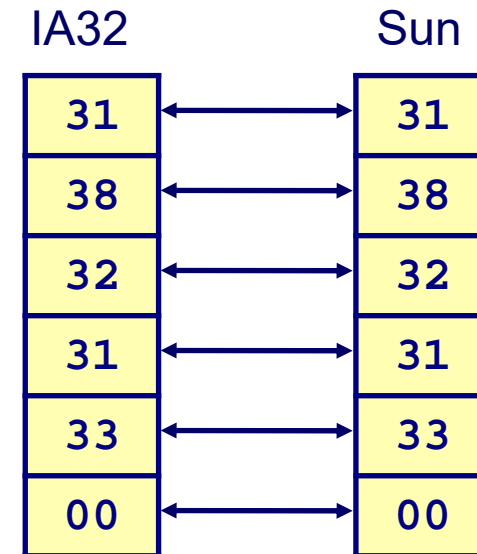
3C
1B
FE
82
FD
7F
00
00

컴파일러와 기계에 다른 다른 주소를 인쇄함.  
프로그램을 동작시킬 때마다 다른 주소가 인쇄될 수도 있음.

# 문자열 표시

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

- C에서 문자열은 ...
  - 문자들의 배열로 표시됨.
  - 각 문자는 ASCII 형식으로
    - 표준 7-비트 인코딩 방식
    - 문자 "0" 은 ASCII 코드 0x30
      - 문자 i는 0x30+i 코드
  - 문자열은 ASCII 코드 0으로 마쳐
  - Big/little endian은 차이 없음.
  - 문자열은 바이트 단위라서...





# Practice : Byte ordering

- 다음과 같은 값이 주어졌다. show\_bytes 함수를 호출할 때, 실행하는 머신의 바이트 정렬법에 따라 화면에 출력되는 값을 쓰시오.(주소 제외)
  - `int val = 0x87654321;`
  - `byte_pointer valp = (byte_pointer) &val;`
  - `show_bytes(valp, 1); /* A */`
  - `show_bytes(valp, 2); /* B */`
- |                    |              |
|--------------------|--------------|
| A. Little endian : | Big endian : |
| B. Little endian : | Big endian : |



**C 연산자, 비트수준연산자, 논리  
연산자, 쉬프트 연산자**

# 부울 대수 (C 언어에서 연산자)



a	b	a & b	a   b	~a	a ^ b
0	0	0	0	1	0
1	0	0	1	0	1
0	1	0	1	1	1
1	1	1	1	0	0

# Practice 2 : Boolean bit operation



- Operate on bit vectors

01101001	01101001	01101001	
& 01010101	01010101	^ 01010101	~ 01010101
<u>          </u>	<u>          </u>	<u>          </u>	<u>          </u>
0100	0111		



# 논리연산자(C) – 부울 연산자?

- 세가지: &&, ||, !
  - 0은 “False”
  - 0이 아닌 것은 “True”
  - 0이나 1를 리턴 함.
  - 일찍 마치기도 함.  $1 || 0 || 0 || 1 \dots \rightarrow$  처음에 바로 1
- 예제들
  - $!0x41 \rightarrow 0x00$ ,  $!0x00 \rightarrow 0x01$
  - $!!0x41 \rightarrow 0x01$
  - $0x69 \&\& 0x55 \rightarrow 0x01$
  - $0x69 || 0x55 \rightarrow 0x01$
  - $p \&\& *p \rightarrow$  (널 포인터 접근 방지표현)



# 쉬프트 연산

- Left Shift:  $x \ll y$ 
  - **x를 왼쪽으로 y 위치만큼 쉬프트**
    - 좌단 비트들은 사라짐, 오른쪽에서는 '0'이 들어옴.
- Right Shift:  $x \gg y$ 
  - **x를 오른쪽으로 y 위치만큼 쉬프트**
    - 우측 비트들은 사라짐.
  - 논리 쉬프트 : '0'으로 채움.
  - 산술 쉬프트 : 최좌단 비트의 반복 채움
- y 가  $< 0$  거나,  $\geq$  워드크기 → 정의안됨!



# Practice :

## 쉬프트연산, 비트연산의 활용



- 8비트로 표시한 값  $x$  에 대해 다음과 같이 left 또는 right shift를 수행한 결과를 빈 칸에 쓰시오.

연산	값	
$x$	0110 0011	1001 0101
$x \ll 4$	0011 0000	
$x \gg 4$ (논리)	0000 0110	
$x \gg 4$ (산술)		1111 1001

- 정수  $x$ 의 마지막 비트가 0/1인지 알려면?
- 정수  $y$ 의 최상위 비트가 0/1인지 알려면?
- 정수  $z$ 의 네번째 비트가 0/1인지 알려면?



정수의 표현 방법

**수의 표현**

**2,8,10,16진수**

# Practice 1: 2진수, 16진수, 10진수



- 다음의 숫자들을 지시한 대로 변환하십시오.
  - A. 0x39A7F8 => 2진수로
  - B. 2진수 1100 0011 0101 0001 을 16진수로 => 0x
  - C. 0xD5E4C 를 32비트 이진수로
  - D. 십진수 255를 이진수로
  - E. 16진수 0x010E 를 십진수로
  - F. 10진수 314,156을 16진수로?



# 314,156 = 0x???

- $314,156 = 19,634 * 16 + 12$  (C)
  - $19,364 = 1,227 * 16 + 2$  (2)
  - $1,227 = 76 * 16 + 11$  (B)
  - $76 = 4 * 16 + 12$  (C)
  - $4 = 0 * 16 + 4$  (4)
- ➔ 0x4CB2C



# w비트 워드 x의 정수화

비부호형 정수

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

부호형 정수 (2의 보수)

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

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

MSB는  
부호비트

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

- 2의 보수방식에서 MSB는 부호-크기 방식처럼 부호를 나타낸다
  - 0 이면 양수
  - 1 이면 음수

# 정수 표시의 예

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



**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
<b>Sum</b>	<b>15213</b>		<b>-15213</b>	



# 표현 가능한 정수의 범위

## ■ 비부호형 Unsigned Values

● **UMin** = 0

000...0

● **UMax** =  $2^w - 1$

111...1

## ■ 2의 보수 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		01111111 11111111
TMin	-32768	80 00	
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

# Signed 와 Unsigned 수의 비교



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

- 관찰

- $|TMin| = TMax + 1$

- 범위가 대칭이 아니다

- $UMax = 2 * TMax + 1$

- 동일성

- 양수부분에 있어서는 signed 와 unsigned의 표현은 동일하다

- $UMax = 2^w - 1$

- $Tmax = 2^{w-1} - 1$





## Practice 5 : 정수의 표현

- 8비트로 표시된 16진수 x를 다음의 표에 맞게 비부호형 및 부호형 값으로 계산해서 채우시오

x		정수	
16진수	2진수	비부호형	부호형
0x 03	0000 0011	3	3
0x 51			
0x 8A	1000 1010	$2^7+2^3+2^1 = 128+8+2 = 138$	$-2^7+2^3+2^1 = -128+8+2 = -118$
0x D9			

# Why TMin32 as -2147483647-1 ??



```
// limits.h
```

```
/* Minimum and maximum values a 'signed int'  
can hold. */
```

```
#define INT_MAX 2147483647
```

```
#define INT_MIN (-INT_MAX - 1)
```

Word Size	ISO C90		ISO C99	
Expression	-2147483648	0x80000000	-2147483648	0x80000000
32	unsigned	unsigned	long long	unsigned
64	long	unsigned	long	unsigned



# CASTING



# 데이터 타입변환 casting 하기

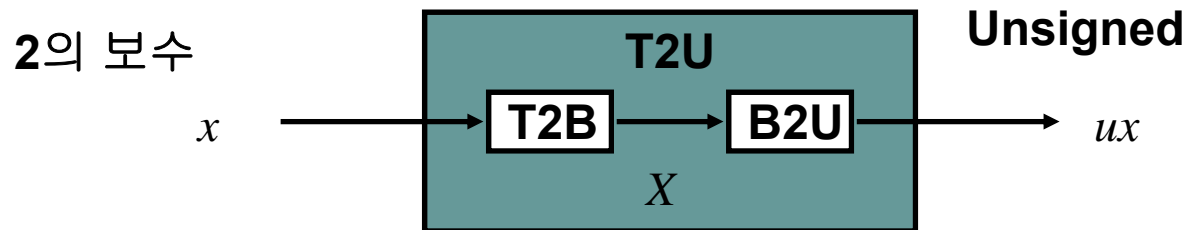
- C 언어에서는 signed로부터 unsigned로의 변환을 허용한다

```
short int          x = 15213;
unsigned short int ux = (unsigned short) x;
short int          y = -15213;
unsigned short int uy = (unsigned short) y;
```

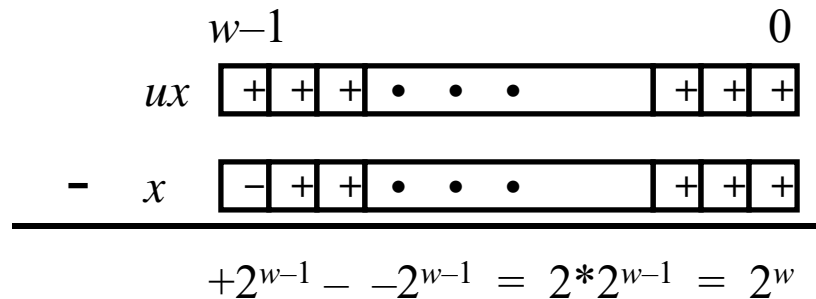
- 결과값
  - 비트 표현에는 변화가 없다
  - 양수는 변화가 없다 (당연)
    - $ux = 15213$
  - 음수는 양수로 변환된다
    - $uy = 50323$



# Signed와 Unsigned와의 관계



비트패턴은 동일하게 유지된다



$$ux = \begin{cases} x & x \geq 0 \\ x + 2^w & x < 0 \end{cases}$$



# Signed와 Unsigned와의 관계

$$ux = \begin{cases} x & x \geq 0 \\ x + 2^w & x < 0 \end{cases}$$

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

Weight	-15213		50323	
1	1	1	1	1
2	1	2	1	2
4	0	0	0	0
8	0	0	0	0
16	1	16	1	16
32	0	0	0	0
64	0	0	0	0
128	1	128	1	128
256	0	0	0	0
512	0	0	0	0
1024	1	1024	1	1024
2048	0	0	0	0
4096	0	0	0	0
8192	0	0	0	0
16384	1	16384	1	16384
32768	1	-32768	1	32768
Sum	-15213		50323	

●  $uy = y + 2 * 32768 = y + 65536$  (16비트의 경우)

# C 언어에서 signed, unsigned 변환



- 상수
  - 아무 명시가 없으면 signed integers 임
  - U를 숫자 끝에 붙이면 Unsigned  
0U, 4294967259U
- 타입변환 Casting
  - 명시적으로 casting을 하는 경우

```
int tx, ty;
unsigned ux, uy;
tx = (int) ux;
uy = (unsigned) ty;
```
  - 묵시적 캐스팅 Implicit casting 을 이용할 수도 있다

```
tx = ux; // unsigned를 signed로 변환
uy = ty; // signed를 unsigned로 변환
```



# Casting 충격

- 수식계산시
  - signed와 unsigned 값들이 한 개의 수식 내에 섞여 있는 경우 implicit 하게 unsigned로 바뀌어 진다
  - 비교연산자에서도 발생한다 <, >, ==, <=, >=
  - Examples for W = 32

Constant 1	Constant 2	Relation	Evaluation
0	0U		
-1	0		
-1	0U		
2147483647	-2147483648		
2147483647U	-2147483648		
-1	-2		
(unsigned)-1	-2		
2147483647	2147483648U		
2147483647	(int) 2147483648U		



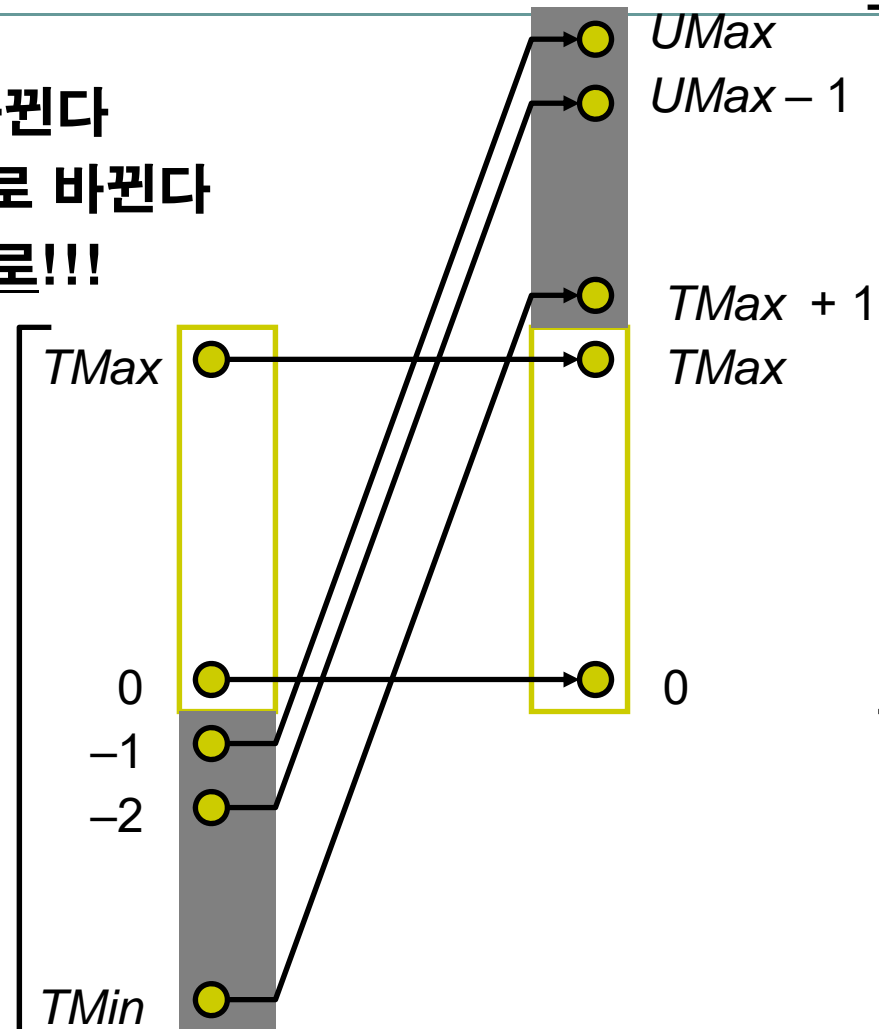
# Casting 충격에 대한 설명



## ■ 2의 보수 → 비부호형

- 크기 순서가 뒤바뀐다
- 음수 → 큰 양수로 바뀐다
- 비트표현은 그대로!!!
- 해석만 다르게!

2's Comp.  
Range



Unsigned  
Range



## Practice 7 : Casting (casting.c)

- 다음 표의 식들이 정수(int)의 2의 보수를 사용하는 연산을 수행한다고 할 때, 비교연산의 결과(Y/N)와 사용되는 정수의 타입을 채우시오.

Expression	Type	Evaluation	구체적이유?
<code>-2147483647-1 == 2147483648U</code>	_____	_____	
<code>-2147483647-1 &lt; 2147483647</code>	_____	_____	
<code>-2147483647-1U &lt; 2147483647</code>	_____	_____	
<code>-2147483647-1 &lt; -2147483647</code>	_____	_____	
<code>-2147483647-1U &lt; -2147483647</code>	_____	_____	

Casting: 비트표현은 그대로, 해석만 달라짐!!



더 큰 변수로 casting되는 경우...

## 부호확장, 제로확장 (SIGN/ZERO EXTENSION)



# 부호 확장 sign extension

- 목적

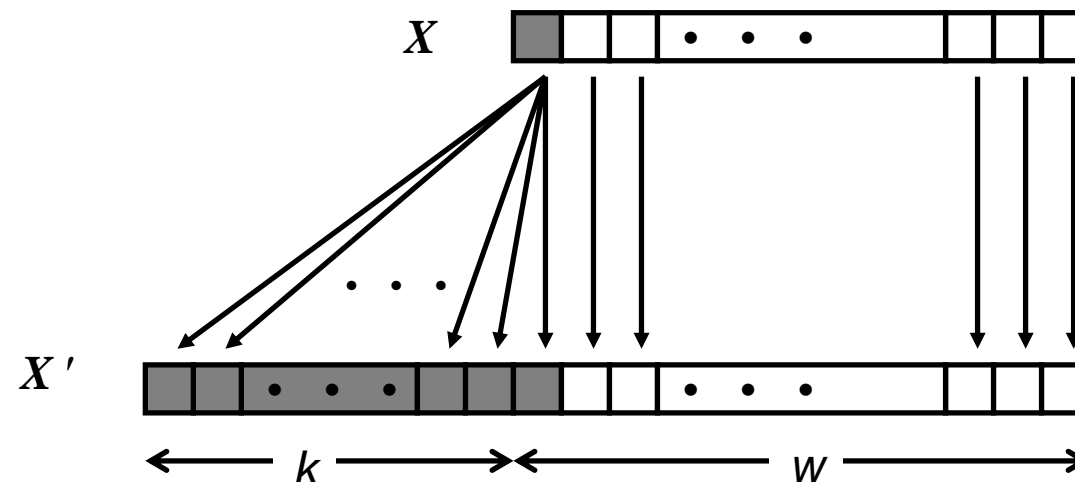
- W 비트의 부호형 정수 x가 주어질 때 x를 w+k 비트의 보다 길이가 긴 정수로 변환시킨다

- 규칙

- x의 부호비트를 k 개 복사한다
- $X' = \underbrace{x_{w-1}, \dots, x_{w-1}}_{k \text{ copies of MSB}}, x_{w-1}, x_{w-2}, \dots, x_0$

Zero extensio은  
항상 0로만 확장. Unsigned의 경우

어셈블리어 명령어:  
`movsx %bx, %eax`  
`movzx %ax, %ebx`





# 부호 확장 예제

```
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

- C 언어에서는 부호확장을 자동으로 해준다
- Unsigned의 경우에는 zero extension을...



---

**덧셈(SIGNED, UNSIGNED)**



# 정수의 연산

- 비 부호형의 덧셈

- 일반적인 덧셈연산과 동일
- Carry 는 무시
- mod 함수로 표시가능
- $s = \text{UAdd}_w(u, v) = (u + v) \bmod 2^w$

Operands:  $w$  bits

$u$



+  $v$



참값:  $w+1$  bits

$u + v$

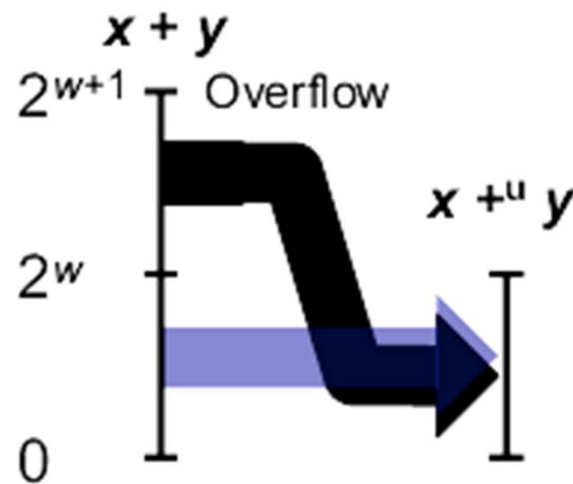


캐리 버림:  $w$  bits

$\text{UAdd}_w(u, v)$



# 비부호형 정수의 덧셈



- $x+y$ 의 값이  $2^w - 1$ 보다 크면 Overflow가 발생한다





# 부호형(2의 보수)에서의 덧셈

Operands:  $w$  bits

$u$

+  $v$

참값:  $w+1$  bits

$u + v$

Discard Carry:  $w$  bits

$\text{TAdd}_w(u, v)$

- 비부호형에서의 덧셈과 동일하게 수행
  - C에서 부호형과 비부호형의 덧셈 Signed vs. unsigned

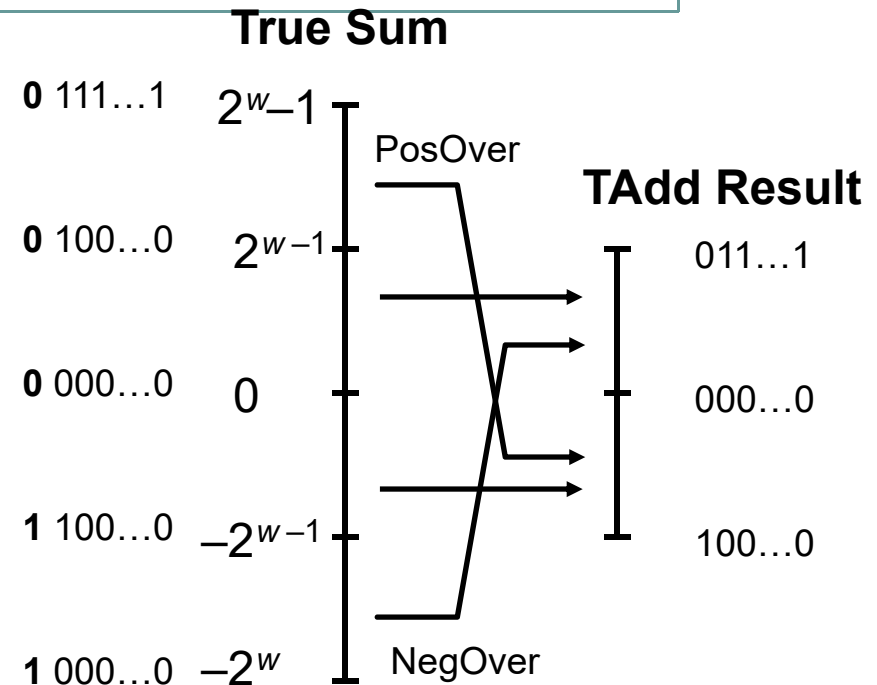
```
int s, t, u, v;  
s = (int) ((unsigned) u + (unsigned) v);  
t = u + v
```
  - $s == t$  동일한 결과를 얻는다



## 2의 보수 덧셈에서 Overflow 찾아내기

- 목표
  - $s = TAdd_w(u, v)$  일 때
  - $s = Add_w(u, v)$  성립여부 체크
  - Example
 

```
int s, u, v;
s = u + v;
```
- 판단방법
  - Overflow iff either:
    - $u, v < 0, s \geq 0$  (NegOver)
    - $u, v \geq 0, s < 0$  (PosOver)



$$TAdd_w(u, v) = \begin{cases} u + v + 2^{w-1} & u + v < TMin_w \text{ (NegOver)} \\ u + v & TMin_w \leq u + v \leq TMax_w \\ u + v - 2^{w-1} & TMax_w < u + v \text{ (PosOver)} \end{cases}$$



## Practice 6 : Overflow(tadd.c)

- 두 정수의 덧셈의 결과 오버플로우가 발생하지 않으면 1을 리턴하는 함수 `tadd_ok(int x, int y)`를 작성하시오.
- `int tadd_ok(int x, int y) {`
- `}`



# 곰셈과 쉬프트연산

# 비부호형 곱셈

Operands:  $w$  bits

$u$

\*

$v$

True Product:  $2w$  bits

$u \cdot v$

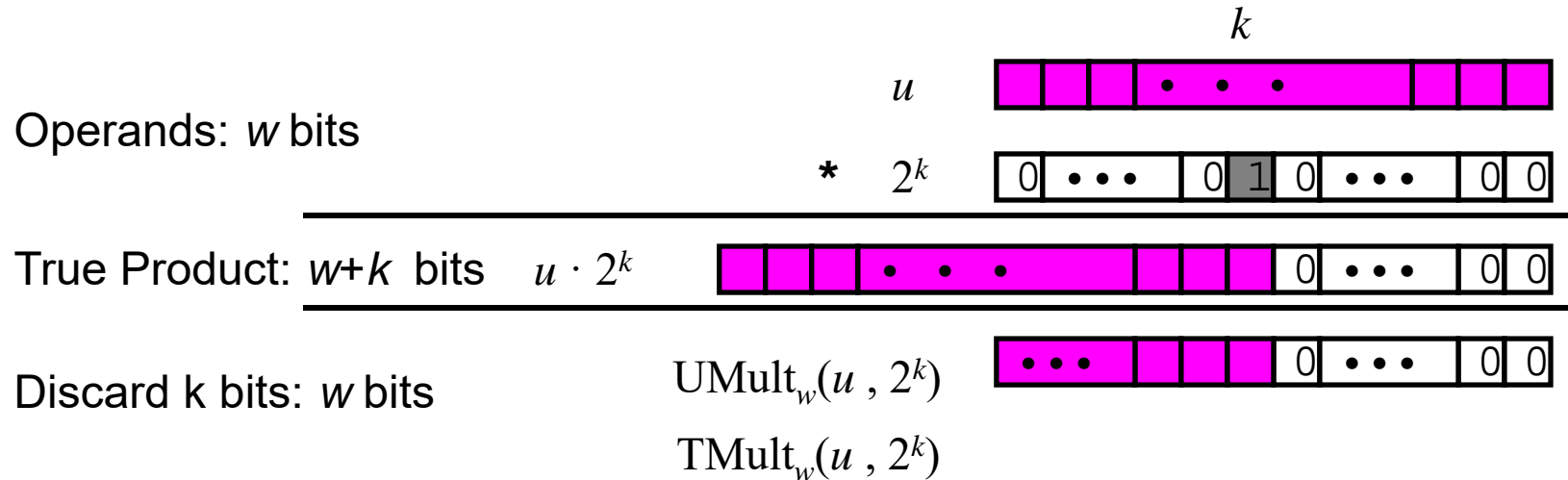
Discard  $w$  bits:  $w$  bits

$\text{UMult}_w(u, v)$

- 표준 곱셈함수와 동일
  - 상위  $w$  비트는 무시
  - mod 로 표시할 수 있음
  - $\text{UMult}_w(u, v) = (u \cdot v) \bmod 2^w$
- 부호형 곱셈은 비부호형과 동일하게 수행



# Shift 연산을 이용한 ( $2^k$ ) 곱셈



## ● 연산

- $u \ll k$  gives  $u * 2^k$
- signed 와 unsigned 모두 적용
  - $u \ll 3 \quad == \quad u * 8$
  - $(u \ll 5) - (u \ll 3) \quad == \quad u * 24$
  - 쉬프트와 덧셈/뺄셈이 곱셈보다 훨씬 빠름!!



# 컴파일러의 곱셈번역

## C Function

```
int mul12(int x)
{
    return x*12;
}
```

### Compiled Arithmetic Operations

```
leaq (%rax,%rax,2), %rax
salq $2, %rax
```

### Explanation

```
t <- x+x*2
return t << 2;
```

- C 컴파일러는 상수의 곱셈을 자동으로 shift와 덧셈으로 번역한다



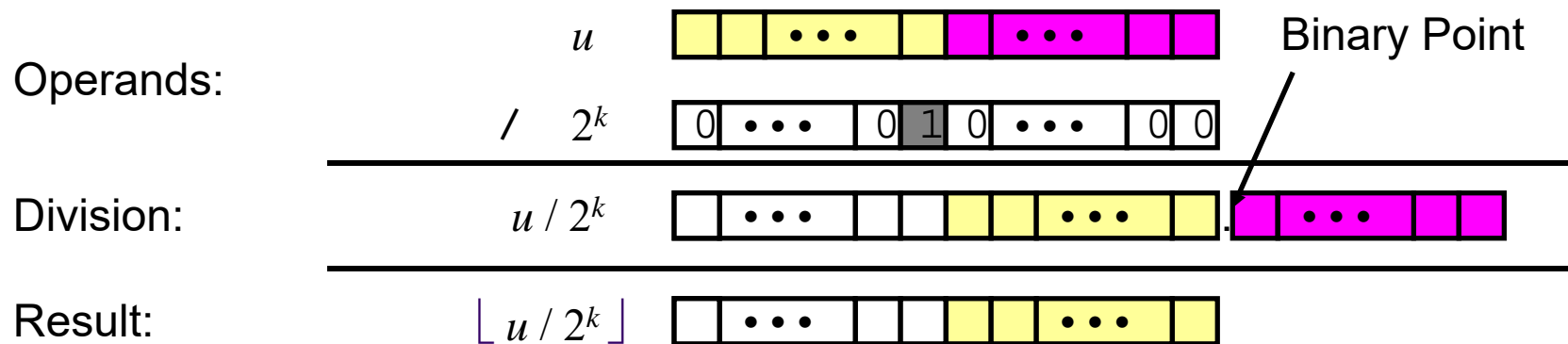
# 나눗셈과 쉬프트 연산





# 비부호형 정수의 shift 이용 나눗셈( $/2^k$ )

- 몫:  $u \gg k$  하면  $\lfloor u / 2^k \rfloor$  가 된다
- 논리 쉬프트 연산을 사용 logical shift



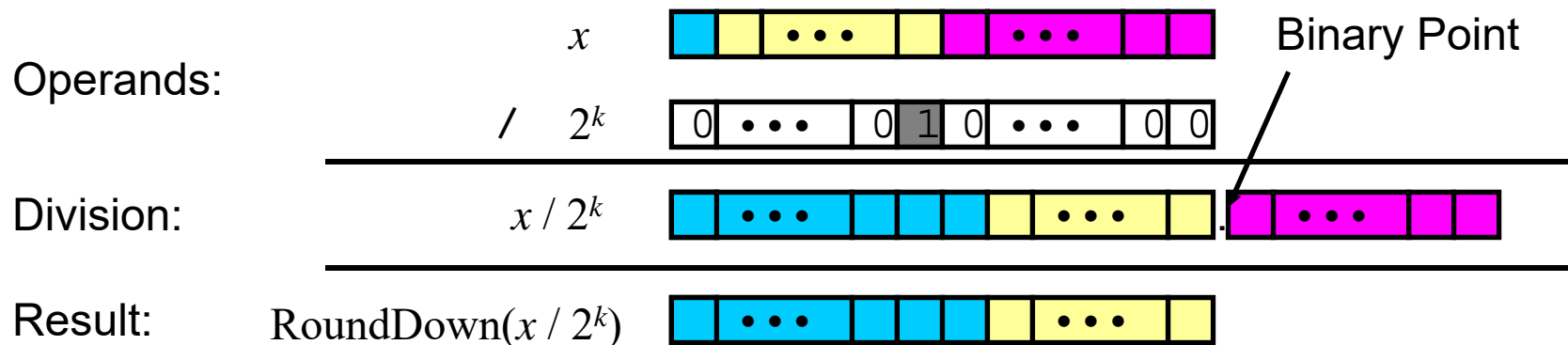
	Division	Computed	Hex	Binary
x	15213	15213	3B 6D	00111011 01101101
x >> 1	7606.5	7606	1D B6	00011101 10110110
x >> 4	950.8125	950	03 B6	00000011 10110110
x >> 8	59.4257813	59	00 3B	00000000 00111011

소수점 이하는 절삭됨!!

# 부호형 정수의 shift 이용 나눗셈 (shift.c) ( / 2^k )



- $x \gg k$  gives  $\lfloor x / 2^k \rfloor$
- 산술 쉬프트 연산을 사용 arithmetic shift
- $x < 0$  이면, 잘못된 방향으로 절삭이 일어난다



	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

음수일 때 보정식 :  $(x < 0 ? x + (1 \ll k) - 1 : x) \gg k$



## 요약: 정수의 표현

- 바이트 인코딩, 워드, 데이터 크기(C에서)
- 바이트 순서(intel CPU : little endian)
- C 연산자들:비트,논리,쉬프트: data lab에서
- Signed(2의 보수), unsigned : 해석의 문제
- Signed, unsigned 혼합 → unsigned로
- 제로/부호 확장,
- 덧셈(signed, unsigned):해석의 문제, flag설정
- 곱셈과 쉬프트, 나눗셈과 쉬프트



# 실습과 다음 주 준비

- 실습: GCC, MAKE, Shell Script
- 다음주 강의 시간(9월15일) – 추석 연휴 휴강  
(예습 동영상 없음)
- 다음 실습 (9월20일화) **C 복습과 GDB 사용  
하기 1(소스 수준 디버깅)**
- 그 다음주 (9월22일) 강의 동영상: 실수 표현  
방법
- 예습 질문은 개인과제로 올림.