Supplementary: Chap.2-2

- Ex-1
- Ex-2
- Ex-3
- Review: Power-of-2 division

Ex-1

Unsigned negation

- *w*-bit negation

$$-_w^u x = \begin{cases} x, & x = 0 \\ 2^w - x, & x > 0 \end{cases}$$

- Example) 6-bit unsigned
  - 001110 (14) →
  - 111101 (61) →

Ex-2

- Signed (2's-complement) addition (5-bits)

x	y	$x + y$	$x +_5^t y$	Case
[10100]	[10001]			
[11000]	[11000]			
[10111]	[01000]			
[00010]	[00101]			
[01100]	[00100]			

Ex-3

- Signed (2's-complement) multiplication (3-bits)

Mode	x		y		$x \cdot y$	Truncated $x \cdot y$
Unsigned	_____	[100]	_____	[101]	_____	_____
Two's comp.	_____	[100]	_____	[101]	_____	_____
Unsigned	_____	[010]	_____	[111]	_____	_____
Two's comp.	_____	[010]	_____	[111]	_____	_____
Unsigned	_____	[110]	_____	[110]	_____	_____
Two's comp.	_____	[110]	_____	[110]	_____	_____

Review

- **Division: Correct power-of-2 division**
  - We want $\lceil x / 2^k \rceil$ (Round Toward 0) when $x < 0$
  - Compute as $\lfloor (x + 2^k - 1) / 2^k \rfloor$
    - In C, **(x + (1 << k) – 1) >> k**
    - Using the property $\lceil x / y \rceil = \lfloor (x + y - 1) / y \rfloor$
    - Biases dividend toward 0

Expression	Division	Result	Hex	Binary
x	-12340	-12340	CF CC	11001111 11001100
x >> 1	-6170.0	-6170	E7 E6	11100111 11100110
x >> 4	-771.25	-771	FC FC	11111100 11111101
x >> 8	-48.203125	-48	FF CF	11111111 11010000

Review

- **Division: Correct power-of-2 division**
  - Compiled signed division code
    - Uses arithmetic shift for signed

Compiled code

```
        testl   %eax, %eax
        js      L4
L3:
        sarl    $3, %eax
        ret
L4:
        addl    $7, %eax
        jmp     L3
```

C function

```
int idiv8 (int  x)
{
    return x / 8;
}
```

Explanation

```
if  (x  <  0)
    x  += 7;
return  x  >> 3;
```

Summary