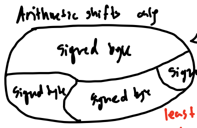
2.71. *Scratchwork

Arithmetic shifts only




← 32-bit unsigned

least significant

byte within word = {0, 1, 2, 3}

$2^3$ $2^2$ $2^1$ $2^0$

most significant

Desired Output: -1

bytenum: 3

bytenum << 3 : $3 \cdot 2^3 = 24$

(-1)

Word: 11111111 11111111 11111111 11111111

Word>>24: 000 00000 000 00000 000 00000 11111111

& 0xFF: 000 00000 000 00000 000 00000 11111111

000 00000 000 00000 000 00000 11111111

Output : 255

/* Declaration of type for the unsigned */

typedef unsigned packed_t

/* Extract byte from word. Return as signed integer */

int xbyte (packed_t word, int bytenum);

Signed - x,x,x ...., x,x $\xrightarrow{\text{packed}}$ unsigned x,x,x, ...., x,x

return as signed integer.

packed_t - unsigned

bytenum - 1011....11

3 2 1 0

11111111 11111111 11111111 11111111

word byte #

xbyte

byte sign extended

$3 \cdot 2^3 = 24$

3

bytenum 00000000 00000000 00000000 00000011

word>>bytenum 00000011 00000000 00000000 00000000

&0xFF 00000000 00000000 00000000 11111111

Output: 00000000 00000000 00000000 00000000

a. The problem with this code is that it uses the AND operator on the shifted bits and does not take care of the sign encoding. In other words, the code only works for nonnegative integers. For example, the code does not work for -1 as shown below.

Desired Output: -1

bytenum: 1

bytenum << 3 : $1 \cdot 2^3 = 8$

3 2 1 0

Word: 11111111 11111111 11111111 11111111 (-1)

Word>>8 : 11111111 11111111 11111111 000 00000

& 0xFF: 000 00000 000 00000 000 00000 11111111 *

000 00000 000 00000 000 00000 000 00000

Output : 0

The output should be -1 if you're taking the third byte, but its 0

b.

3 2 1 0

11111111 11111111 11111111 11111111

Significance

#3: leftshifts: $3-3=0 \cdot 2^3 = 0$ 11111111 11111111 11111111 11111111

#2: leftshifts: $3-2 = 1 \cdot 2^3 = 2^3 = 8$ 11111111 11111111 11111111 00000000

#1: leftshift: $3-1 = 2 \cdot 2^3 = 2^4 = 16$ 11111111 11111111 00000000 00000000

#0: leftshift: $3-0 = 3 \cdot 2^3 = 24$ 11111111 00000000 00000000 00000000

Full file is under 271b.c in submission file

```c
#include <stdio.h>
#include <assert.h>

typedef unsigned packed_t;

int xbyte(packed_t word, int bytenum) {
    //Holds amount needed to push the desired byte to the end of the 32-bit series
    int arith_left_shift = (3-bytenum) << 3;
    //Holds amount needed to push the byte that is at the end of the series to the least significant placement of 0
    int arith_right_shift = 24;
    //Explicit casting to signed encoding
    int output = (int)(word << arith_left_shift) >> arith_right_shift;
    return output;
}
```

2.82 a) $(x<y) == (-x>-y)$

$\boxed{\text{False}}$

<u>Counter example:</u> $x + {\sim}x = -1$

$x = -1 - {\sim}x$

$-x = 1 + {\sim}x$

Suppose ${\sim}x$ is TMIN, then, in bit form:

$$\begin{array}{r} {\sim}x = 111\ldots111 \\ +\,000\ldots001 \\ \hline \underbrace{1000\ldots000}_{\text{overflow}} \end{array}$$

The result is 0, which does not satisfy the above statement if $y$ is a positive integer

b) $((x+y) << 4) + y - x == 17^*y + 15^*x$

$\boxed{\text{True}}$

$$(x+y)\cdot 2^4 + y - x = 16x + 16y + y - x = 17^*y + 15^*x$$

c. ${\sim}x + {\sim}y + 1 = {\sim}(x+y)$

It is true that, if $a$ is a signed int, then: $\boxed{\text{The statement is true}}$

$a + {\sim}a = -1$

$a + {\sim}a + 1 = 0$

let $a = x+y$

$x+y + {\sim}(x+y) + 1 = 0$

${\sim}(x+y) + 1 = -x - y$

$${\sim}(x+y) + 1 = 1 + {\sim}x + 1 + {\sim}y$$
$$-1 \qquad\qquad -1$$

${\sim}(x+y) = {\sim}x + {\sim}y$ ■

* $x + {\sim}x = -1$

$x = -1 - {\sim}x$

$-x = 1 + {\sim}x$

d $(ux - uy) == -((\text{unsigned})(y-x))$

$ux \to$ unsigned $uy \to$ unsigned

$-(\text{unsigned})(y-x) \to$ explicit casting

$-(uy - ux) = (ux - uy)$ $\boxed{\text{True}}$

e. $((x>>2)<<2) <= x$

$x \to$ signed $\boxed{\text{True}}$

11 0....00 ✓

00...11...00 ✓

00...00...01 ✓

111111... 11 ✓

>>2

11111....11

<<2

$\underbrace{11111....00}_{\text{bigger negative num}}$ ✓