

# Words, bits and pieces

## Lecture B05



Lecture notes section B05

2002-01-11

CSE1303 Part B lecture notes

1

## Last time

- Floating point
  - Anatomy of floating point
    - sign
    - exponent
    - mantissa
  - Floating point arithmetic
    - multiplication
    - addition
  - Limitations of floating point

2002-01-11

CSE1303 Part B lecture notes

2

## In this lecture

- Working with large data
  - byte order
- Working with small data
  - bitwise operations
    - AND, OR, NOT, XOR
    - masking
  - shifting
    - relationship to multiplication and division

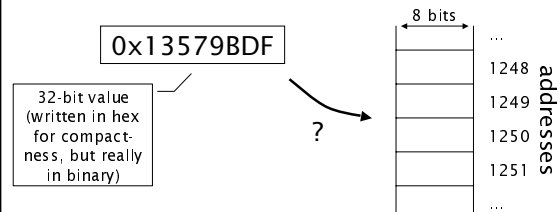
2002-01-11

CSE1303 Part B lecture notes

3

## Byte order

- Problem
  - memory is addressed by bytes (8 bits)
  - how to store a 32-bit value in memory?



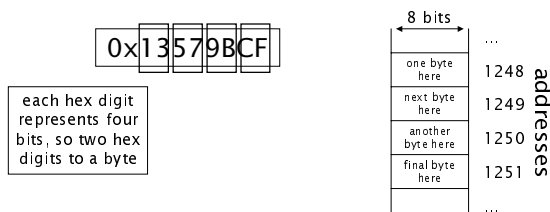
2002-01-11

CSE1303 Part B lecture notes

4

## Byte order

- Solution
  - break up value into byte-sized chunks
  - store each chunk in successive bytes



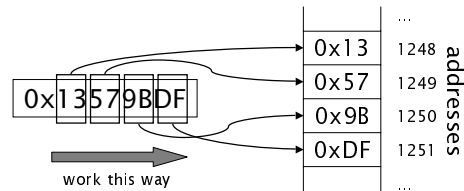
2002-01-11

CSE1303 Part B lecture notes

5

## Byte order

- Which order to store the bytes?
  - starting at the most significant ("big") end?



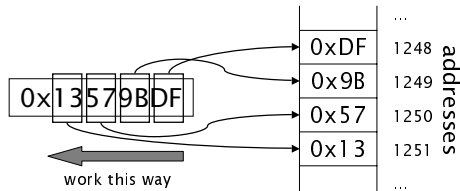
2002-01-11

CSE1303 Part B lecture notes

6

## Byte order

- Which order to store the bytes?
  - starting at the least significant ("little") end?



2002-01-11

CSE1 303 Part B lecture notes

7

## Byte order

- Two common ways of storing word into memory
  - Big Endian
    - store most significant byte in lowest address
    - store least significant byte in highest address
  - Little Endian
    - store least significant byte in lowest address
    - store most significant byte in highest address

2002-01-11

CSE1 303 Part B lecture notes

8

## Byte order

- Which byte order is used depends on the computer's microprocessor
  - Big Endian
    - Motorola 680x0 (Amiga/Atari ST/Mac classic)
    - IBM/Motorola PowerPC (current Macintosh)
    - MIPS (SGI Indy/Nintendo 64)
  - Little Endian
    - Intel 80x86/Pentium (IBM PC compatible)
    - Motorola 6800 (as seen in CSE1102)
    - Rockwell 6502 (Commodore 64)
    - MIPS (Sony Playstation/Digital DECstation)
  - Some processors (e.g., MIPS) can be configured to be either Big Endian or Little Endian

2002-01-11

CSE1 303 Part B lecture notes

9

## Logic operations

- Boolean logic operations
  - operate on one or two binary values (true/false, 1/0)
  - logical AND (on two values)
    - C operator: `&&`
    - result is true if both operands are true, else false
  - logical OR (on two values)
    - C operator: `||`
    - result is true if either operand is true, else false
  - logical NOT (on one value)
    - C operator: `!`
    - invert operand: true  $\Leftrightarrow$  false

2002-01-11

CSE1 303 Part B lecture notes

10

## Bitwise operations

- Bitwise logic operations
  - operate on each bit of one or two values
  - bitwise AND (on two values)
    - for each bit: result is 1 if both operands are 1, else 0
  - bitwise OR (on two values)
    - for each bit: result is 1 if either operand is 1, else 0
  - bitwise Exclusive OR ("XOR") (on two values)
    - for each bit: result is 1 if either operand (but not both) is 1, else 0
  - bitwise NOT (on one value)
    - for each bit: invert bit: 1  $\Leftrightarrow$  0
    - like flip in "flip-and-add-1" negation of signed value

2002-01-11

CSE1 303 Part B lecture notes

11

## Bitwise operations: NOT

- Bitwise NOT
  - Invert every bit in a value

0 1 1 0 1 0 1 0



1 0 0 1 0 1 0 1

in	not
0	1
1	0

truth table

each bit inverted independently of the others

2002-01-11

CSE1 303 Part B lecture notes

12

## Bitwise operations: AND

### Bitwise AND

- performs logical AND on corresponding bits of two values

```

0 1 1 0 1 0 1 0
1 1 0 0 0 0 1 1
-----
0 1 0 0 0 0 1 0
    
```

in 1	in 2	out
0	0	0
0	1	0
1	0	0
1	1	1

truth table

result is 1 if both corresponding input bits are 1, 0 otherwise

2002-01-11

CSE1 303 Part B lecture notes

13

## Bitwise operations: OR

### Bitwise OR

- performs logical OR on corresponding bits of two values

```

0 1 1 0 1 0 1 0
1 1 0 0 0 0 1 1
-----
1 1 1 0 1 0 1 1
    
```

in 1	in 2	out
0	0	0
0	1	1
1	0	1
1	1	1

truth table

result is 1 if either corresponding input bit (or both) are 1, 0 otherwise

2002-01-11

CSE1 303 Part B lecture notes

14

## Bitwise operations: XOR

### Bitwise Exclusive OR (XOR)

- performs logical Exclusive OR (XOR) on corresponding bits of two values

```

0 1 1 0 1 0 1 0
1 1 0 0 0 0 1 1
-----
1 0 1 0 1 0 0 1
    
```

in 1	in 2	out
0	0	0
0	1	1
1	0	1
1	1	0

truth table

result is 1 if either corresponding input bit (but not both) are 1, 0 otherwise

2002-01-11

CSE1 303 Part B lecture notes

15

## Bitwise operations

- Some operations don't affect bits
  - $x \text{ AND } 1 = x$
  - $x \text{ OR } 0 = x$
  - $x \text{ XOR } 0 = x$
- Some operations force bits to 0 or 1
  - $x \text{ AND } 0 = 0$
  - $x \text{ OR } 1 = 1$
- Some operations change bits
  - $x \text{ XOR } 1 = \text{not-}x$

2002-01-11

CSE1 303 Part B lecture notes

16

## Extracting bits

- Sometimes need to isolate one or more bits in a value
  - e.g., find sign in signed int/float

10011010

but need to ignore all the other bits

extract this bit to find out if value is positive or negative

2002-01-11

CSE1 303 Part B lecture notes

17

## Extracting bits

- Use bitwise AND to extract bits
  - for desired bits, AND with 1
    - $0 \text{ AND } 1 \Rightarrow 0$ ,  $1 \text{ AND } 1 \Rightarrow 1$
    - bit remains unchanged
  - for other bits, AND with 0
    - $0 \text{ AND } 0 \Rightarrow 0$ ,  $1 \text{ AND } 0 \Rightarrow 0$
    - bit is cleared (set to 0)
  - only desired bits remain, others are cleared

2002-01-11

CSE1 303 Part B lecture notes

18

### Extracting bits

- Example: extract sign bit of C float

AND

resulting value is zero:  
number is positive

2002-01-11 CSE1 303 Part B lecture notes 19

### Extracting bits

- Example: extract sign bit of C float

AND

a value like this which is used to extract, set or clear individual bits is usually called a mask

2002-01-11 CSE1 303 Part B lecture notes 20

### Clearing bits

- Sometimes need to set a bit to zero
  - e.g., clear sign bit of a float, to make positive (like fabs function)
- Use bitwise AND to clear bits
  - for bits to clear, AND with 0
    - $0 \text{ AND } 0 \Rightarrow 0$ ,  $1 \text{ AND } 0 \Rightarrow 0$
    - bit is cleared (set to 0)
  - for other bits, AND with 1
    - $0 \text{ AND } 1 \Rightarrow 0$ ,  $1 \text{ AND } 1 \Rightarrow 1$
    - bit is unchanged
  - desired bits are cleared to zero, others remain unchanged

2002-01-11 CSE1 303 Part B lecture notes 21

### Clearing bits

- Example: set sign bit of C float to 0

AND

resulting number is positive: 0 in MSB

2002-01-11 CSE1 303 Part B lecture notes 22

### Setting bits

- Sometimes need to set a bit to one
  - e.g., set sign bit of a float, to make negative
- Use bitwise OR to set bits
  - for bits to set, OR with 1
    - $0 \text{ OR } 1 \Rightarrow 1$ ,  $1 \text{ OR } 1 \Rightarrow 1$
    - bit is set (to 1)
  - for other bits, OR with 0
    - $0 \text{ OR } 0 \Rightarrow 0$ ,  $1 \text{ OR } 0 \Rightarrow 1$
    - bit is unchanged
  - desired bits are set to one, others remain unchanged

2002-01-11 CSE1 303 Part B lecture notes 23

### Setting bits

- Example: set sign bit of C float to 1

OR

resulting number is negative: 1 in MSB

2002-01-11 CSE1 303 Part B lecture notes 24

## Toggleing bits

- Sometimes need to flip a bit's value
  - e.g., toggle sign bit to negate a float
- Use bitwise XOR (exclusive OR) to toggle bits
  - for bits to toggle, XOR with 1
    - $0 \text{ XOR } 1 \Rightarrow 1$ ,  $1 \text{ XOR } 1 \Rightarrow 0$
    - bit is flipped ( $0 \Leftrightarrow 1$ )
  - for other bits, XOR with 0
    - $0 \text{ XOR } 0 \Rightarrow 0$ ,  $1 \text{ XOR } 0 \Rightarrow 1$
    - bit is unchanged
  - desired bits are flipped, others remain unchanged

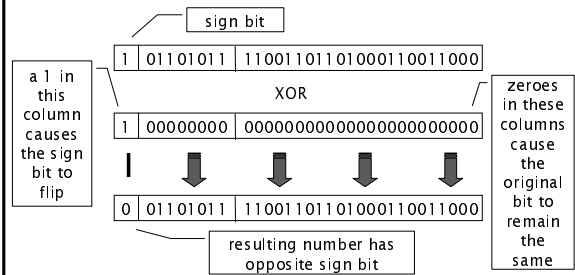
2002-01-11

CSE1 303 Part B lecture notes

25

## Toggleing bits

- Example: negate a C float



2002-01-11

CSE1 303 Part B lecture notes

26

## Bitwise operations in C

- bitwise AND (on two values)
  - C operator: & (ampersand)
- bitwise OR (on two values)
  - C operator: | (vertical bar)
- bitwise Exclusive OR (XOR) (on two values)
  - C operator: ^ (caret)
- bitwise NOT (on one value)
  - C operator: ~ (tilde)

2002-01-11

CSE1 303 Part B lecture notes

27

## Bitwise operations in C

```

/* Examples of bitwise operations. */
int main()
{
    /* 16-bit value, binary 0000001101011110. */
    unsigned short value = 0x035E;
    /* 16-bit mask, binary 0000000011111111. */
    unsigned short mask = 0x00FF;

    /* %X printf format prints in hex */
    printf("%04X\n", value); /* 035E */
    printf("%04X\n", mask); /* 00FF */
    printf("%04X\n", ~value); /* FCA1 */
    printf("%04X\n", value & mask); /* 005E */
    printf("%04X\n", value | mask); /* 03FF */
    printf("%04X\n", value ^ mask); /* 03A1 */
    return 0;
}
    
```

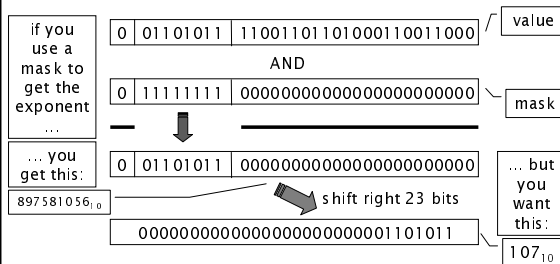
2002-01-11

CSE1 303 Part B lecture notes

28

## Shifting

- Example: extracting exponent from a C float



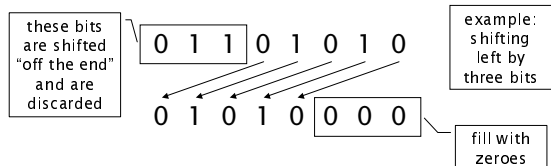
2002-01-11

CSE1 303 Part B lecture notes

29

## Shifting

- Shifting is the moving of patterns of bits left or right within a word
  - left or right
  - $0 \leq \text{amount to shift} < \text{word size}$



2002-01-11

CSE1 303 Part B lecture notes

30

## Shifting in C

- Binary (two value) operators
- Shift left operator: <<
- Shift right operator: >>
- Value to shift goes on left of operator
- Amount to shift goes on right of operator

2002-01-11

CSE1 303 Part B lecture notes

31

## Shifting in C

```

/* Example using C shift operators. */
int main()
{
    /* 16-bit value, binary 1000111100000010. */
    unsigned short value = 0x8F02;
    int i = 4; /* used as a shift amount */

    /* Shift right or left by 4 bits. */
    printf("%04X\n", value >> 4); /* 08F0 */
    printf("%04X\n", value << 4); /* F020 */
    /* Either side can be variable or literal. */
    printf("%04X\n", value >> i); /* 08F0 */
    /* Shifting by 1 harder to see in hex;
       write these in binary to see what's happening. */
    printf("%04X\n", value >> 1); /* 4781 */
    printf("%04X\n", value << 1); /* 1E04 */

    return 0;
}
    
```

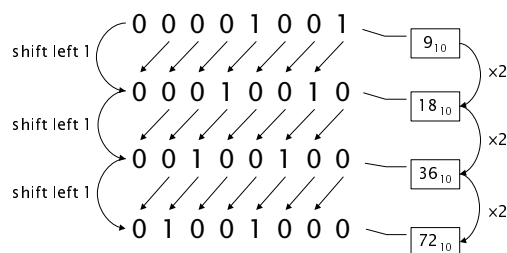
2002-01-11

CSE1 303 Part B lecture notes

32

## Shifting and multiplication

- 8-bit unsigned binary



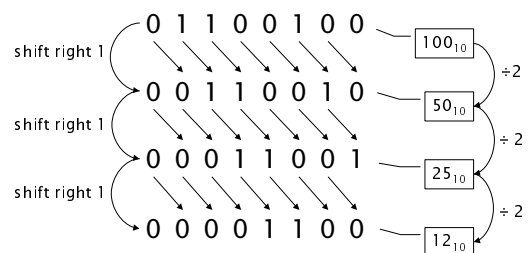
2002-01-11

CSE1 303 Part B lecture notes

33

## Shifting and division

- 8-bit unsigned binary



2002-01-11

CSE1 303 Part B lecture notes

34

## Shifting

- Shifting left a value by n places is equivalent to multiplying by 2<sup>n</sup>
  - << 1 ⇔ × 2; << 2 ⇔ × 4; << 3 ⇔ × 8; etc.
- Shifting right a value by n places is equivalent to dividing by 2<sup>n</sup>
  - >> 1 ⇔ ÷ 2; etc.
- Shifting is quicker than multiplication/division
  - for efficiency, compiler will replace multiplication or division by constant power of two with a shift left or right

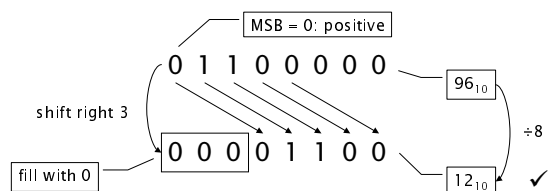
2002-01-11

CSE1 303 Part B lecture notes

35

## Shifting signed values

- 8-bit signed (two's complement)



2002-01-11

CSE1 303 Part B lecture notes

36

## Shifting signed values

- 8-bit signed (two's complement)

MSB = 1: negative

1 0 1 0 0 0 0 0  $-96_{10}$

shift right 3

fill with 0

0 0 0 1 0 1 0 0  $+20_{10}$   $\div 8$  ✗

got 00010100<sub>2</sub>, wanted -12<sub>10</sub> = 11110100<sub>2</sub>; differs in first three bits

2002-01-11 CSE1303 Part B lecture notes 37

## Shifting signed values

- 8-bit signed (two's complement)

MSB = 1: negative

1 0 1 0 0 0 0 0  $-96_{10}$

shift right 3

fill with copies of original MSB

1 1 1 1 0 1 0 0  $-12_{10}$   $\div 8$  ✓

by filling with copies of the MSB, positive numbers stay positive, and negative numbers stay negative

2002-01-11 CSE1303 Part B lecture notes 38

## Shifting

- Need to distinguish two kinds of right shift
  - logical: always fill left bits with 0
  - arithmetic: fill left bits with copies of MSB
  - C >> operator is logical for unsigned values, unknown for signed values
    - gcc uses arithmetic right shift on signed values
- Not a problem with left shift
  - logical left shift same as arithmetic left shift
  - if MSB changes, overflow occurred

2002-01-11 CSE1303 Part B lecture notes 39

## Covered in this lecture

- Byte order
- bitwise operations
  - AND, OR, NOT, XOR
  - masking
- shifting
  - relationship to multiplication and division
  - right shifts
    - arithmetic
    - logical

2002-01-11 CSE1303 Part B lecture notes 40

## Going further

- Boolean algebra
  - the mathematics of binary values
- Digital logic
  - hardware for doing AND, OR, ...
- Bit fields
  - another way of referring to bits in C

2002-01-11 CSE1303 Part B lecture notes 41

## Next time

- Pointers
- Aggregation of data
  - arrays
  - structs
- Characters
  - ASCII
- Strings

Reading:  
Lecture notes section B06

2002-01-11 CSE1303 Part B lecture notes 42

## **Copyright**

Copyright © 2001 Deborah Pickett.  
No part of this presentation may be  
duplicated without permission from  
the author.