# Background/Review on Integers and Bases (lecture)

ICS312

**Machine-Level and Systems Programming**

Henri Casanova (henric@hawaii.edu)

# Numbers and Computers

- Throughout this course we will
  - use binary and hexadecimal representations of numbers
  - need to understand how the computer stores numbers
- So let us go through a simple review:
  - Numbers in different bases (these slides)
  - Number representation in computers (next set of slides)

# Integers and Bases

- "Writing an integer in a base" means "writing the integer as a linear combination of powers of the base, where each coefficient is lower than the base"

$$n = \sum_{k=0}^{\infty} d_k \cdot b^k \quad \text{with} \quad \forall k \;\; 0 \leq d_k < b$$

- There is a unique way to write an integer in a base
- We call the coefficients "digits" and we typically write them from left to right by decreasing exponent, often omitting leading zeros

$$d_{\lceil \log_b n \rceil} \cdots d_1 d_0$$

# Decimal: Base 10

- We use base 10 (decimal) every day with digits between 0 and 9:

  $25 = 2*10^1 + 5*10^0$

  $136 = 1*10^2 + 3*10^1 + 6*10^0$

- We often don't write any leading 0's

$136 = \ldots + 0*10^4 + 0*10^3 + 1*10^2 + 3*10^1 + 6*10^0$

$= \ldots00000136$

- But sometimes we do, e.g., for months: 01/2027
- In this course we'll often write leading zeros because we deal with computer arithmetic (more later)

# Binary: Base 2

- Counting in binary:

  0
  1
  10
  11
  100
  101
  110
  111
  1000
  …

- A binary number with n bits corresponds to integer values between 0 and $2^n - 1$

- Example: An integer that can be encoded on at most 8 bits can only take values between 0 and 255

# Binary Addition

- Adding two binary numbers is just like adding decimal numbers with carries

| With no previous carry | | | | With a previous carry | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| + 0 | + 1 | + 0 | + 1 | + 0 | + 1 | + 0 | + 1 |
| = 0 | = 1 | = 1 | = 0 | = 1 | = 0 | = 0 | = 1 |
| | | | c | | c | c | c |

# Binary Addition

| | | | | | |
|---|---|---|---|---|---|
| | c | c | c | c | |
| | | 1 | 0 | 0 | 1 |
| + | | 1 | 1 | 1 | 1 |
| = | 1 | 1 | 0 | 0 | 0 |

$9_{10}$

$+ 15_{10}$

$= 24_{10}$

# Converting from Binary to Decimal

- Let's denote by $XXXX_2$ a binary representation of a number and by $XXXX_{10}$ a decimal representation
- Converting from binary to decimal is straightforward:

$$10010110_2 = 1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$$
$$= 128 + 16 + 4 + 2$$
$$= 150_{10}$$

- The rightmost bit of a binary number is called the least significant bit (smallest power of 2)
- The leftmost bit of a binary number is called the most significant bit (largest power of 2)
- If the least significant bit is 0, then the number is even, otherwise it's odd

# Converting from Decimal to Binary

- The algorithm proceeds in a series of integer divisions by 2, and by recording the remainder of the division
  - Integer division a/b: a = b* q + r, where all are integers and r < b
- Example: converting $37_{10}$ into binary
  - Divide 37 by 2: 37 = 2*18 + 1
  - Divide 18 by 2: 18 = 2*9 + 0
  - Divide 9 by 2: 9 = 2*4 + 1
  - Divide 4 by 2: 4 = 2*2 + 0
  - Divide 2 by 2: 2 = 2*1 + 0
  - Divide 1 by 2: 1 = 2*0 + 1
  - Result: $100101_2$
- The least significant bit is computed first
- The most significant bit is computed last
- Note that if we continue dividing, we get extraneous leading 0s
  - …$00000100101_2$

# Converting from Decimal to Binary

- Anybody can use this algorithm of course, and I don't really care that you know it
  - Online tools can do all conversions for us anyway
- However, as a computer scientist, for small numbers, we should be able to do quick, intuitive conversions without using the algorithm
- The idea is: find a power of 2 that's near the number, and then add/subtract whatever's needed

- For instance, if asked to convert $19_{10}$ to binary, you should immediately think: 19 = 16 + 3 = 16 + 2 + 1
- Therefore $19_{10} = 10011_2$ (1*16 + 0*8 + 0*4 + 1*2 + 1*1)

# Bases that are powers of 2

- Computers use binary internally because it's easy to associate two states to an electrical current
  - Low voltage = 0, high voltage = 1
- But binary is cumbersome for humans
  - The lower the base the longer the numbers!
  - It's really tedious and error-prone for a human to read/write binary
- Therefore we, as humans, like to use higher bases
- Bases that are powers of 2 make for easy translation to binary, and thus are particularly useful
- Octal: base 8 with digits 0, 1, 2, 3, 4, 5, 6, 7
  - Each digit can be represented with 3 bits
- Hexadecimal: base 16 with digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - Each digit can be represented with 4 bits

# Counting in Hexadecimal

| | | | |
|---|---|---|---|
| $0_{16}=0_{10}$ | $A_{16}=10_{10}$ | $14_{16}=20_{10}$ | $1E_{16}=30_{10}$ |
| $1_{16}=1_{10}$ | $B_{16}=11_{10}$ | $15_{16}=21_{10}$ | $1F_{16}=31_{10}$ |
| $2_{16}=2_{10}$ | $C_{16}=12_{10}$ | $16_{16}=22_{10}$ | $20_{16}=32_{10}$ |
| $3_{16}=3_{10}$ | $D_{16}=13_{10}$ | $17_{16}=23_{10}$ | $21_{16}=33_{10}$ |
| $4_{16}=4_{10}$ | $E_{16}=14_{10}$ | $18_{16}=24_{10}$ | $22_{16}=34_{10}$ |
| $5_{16}=5_{10}$ | $F_{16}=15_{10}$ | $19_{16}=25_{10}$ | $23_{16}=35_{10}$ |
| $6_{16}=6_{10}$ | $10_{16}=16_{10}$ | $1A_{16}=26_{10}$ | $24_{16}=36_{10}$ |
| $7_{16}=7_{10}$ | $11_{16}=17_{10}$ | $1B_{16}=27_{10}$ | $25_{16}=37_{10}$ |
| $8_{16}=8_{10}$ | $12_{16}=18_{10}$ | $1C_{16}=28_{10}$ | $26_{16}=38_{10}$ |
| $9_{16}=9_{10}$ | $13_{16}=19_{10}$ | $1D_{16}=29_{10}$ | $27_{16}=39_{10}$ |

# Converting from hex to decimal

- This is again straightforward

$$A203DE_{16} = 10*16^5 + 2*16^4 + 3*16^2 + 13*16^1 + 14*16^0 = 10{,}617{,}822_{10}$$

# Converting from decimal to hex

- Use the same algorithm as for binary
- Example: convert $1237_{10}$
  - 1237 = 77*16 + 5
  - 77 = 4*16 + 13 (D)
  - 4 = 0*16 + 4
  - Result: $4D5_{16}$

# Hexadecimal addition

| | | |
|---|---|---|
| | c c c | |
| | D 1 F F | $53759_{10}$ |
| + | A 4 D F | + $42207_{10}$ |
| = | 1 7 6 D E | = $95966_{10}$ |

# Why is hexadecimal useful?

- We need to think in binary because computers operate on binary quantities
- But binary is cumbersome
- However, hexadecimal makes it possible to represent binary quantities in a compact form for us to look at and reason about
  - So does octal, to a lesser extent
- Conversions back and forth from binary to hex/octal are straightforward
  - Just convert each octal/hex digit to binary
  - Just convert each group of bits to octal/hex digits
- In what follows let's just consider hex

# Converting from hex to binary

- Consider $A43FE_{16}$
- There are 16 different hex digits, so each one is represented using 4 bits (from 0000 to 1111)
- We thus convert each hex digit into a 4-bit binary number:
  - $A_{16}$: $1010_2$ $4_{16}$: $0100_2$ $3_{16}$: $0011_2$ $F_{16}$: $1111_2$ $E_{16}$: $1110_2$
- We "glue" them all together:
  - $A43FE_{16} = 10100100001111111110_2$
- Important: We must keep the leading 0's for each 4-bit numbers
  - As for the digits 4, and 3 above
- If you're not convinced:
  - In the above example, the digit 3 means that the integer's includes a term $3 * 16^2$ in the linear combination of exponents of the base
  - But $3 * 16^2 = 3 * 2^8 = (2 + 1) * 2^8 = \mathbf{0} * 2^{11} + \mathbf{0} * 2^{10} + \mathbf{1} * 2^9 + \mathbf{1} * 2^8$
  - Which is seen in the binary: $10100100\mathbf{0011}11111110_2$

# Converting from binary to hex

- Let's convert $1001010101111_2$ into hex
- We split it in 4-bit numbers, which we convert separately
- First **we add leading 0's** to have a number of bits that's a multiple of 4:

  0001 0010 1010 1111
- Then we convert
  - $0001_2$ : $1_{16}$
  - $0010_2$ : $2_{16}$
  - $1010_2$ : $A_{16}$
  - $1111_2$ : $F_{16}$
- And the result: $1001010101111_2 = 12AF_{16}$

# Conclusion

- Hopefully, what we just went through was already solid knowledge for all of you
- If it wasn't, just make sure you practice this until it is solid
  - Job interviews sometimes have a "convert this from/to hex" question…
  - It would be embarrassing to have to resort to an online tool / LMM for simple conversions
- Onward to how computers store numbers…