CPS104
Computer Organization and Programming
Lecture 3 : Data representations

Robert Wagner

http://www.cs.duke.edu/~raw/cps104/Lectures.html



## Overview of Today's Lecture:

- Introduction: Decimal, Binary, Octal and Hexadecimal numbers.
- Storage types: Byte, Word, Double-word.
- ASCII Characters.
- 2's complement numbers.
- Floating-point numbers.

Sections 3,7, 4.1-4.3, 4.8 In the textbook



### Binary , Octal and Hexidecimal numbers

- Decimal numbers are used by people! Most computers use Binary numbers and Binary Arithmetic!!
- Computers can input and output decimal numbers but they convert them to internal binary number representation.
- Binary numbers use only two different digits: {0,1}
  - Example: $1200_{10} = 0000\ 0100\ 1011\ 0000_{2}$
- Octal numbers use 8 digits: {0 - 7}
  - Example: $1200_{10} = 04260_{8}$
- Hexadecimal numbers use 16 digits: {0-9, A-F}
  - Example: $1200_{10} = 0X04B0_{16}$

## Binary and Hex

- Use Hexadecimal numbers:
  - To convert to and from binary: group binary digits in groups of four and convert according to table:

| Bin | Hex | Bin | Hex |
|---|---|---|---|
| 0 | 0000 | 8 | 1000 |
| 1 | 0001 | 9 | 1001 |
| 2 | 0010 | A | 1010 |
| 3 | 0011 | B | 1011 |
| 4 | 0100 | C | 1100 |
| 5 | 0101 | D | 1101 |
| 6 | 0110 | E | 1110 |
| 7 | 0111 | F | 1111 |

**Example:**

$1100\ 0010\ 0110\ 0111\ 0100\ 1111\ 1101\ 0101_2$

$C\ 2\ 6\ 7\ 4\ F\ D\ 5_{16}$


©RW Fall 2000

## Why grouping digits works

$101111000101_2$

1011 | 1100 | 0101

$1\times2^{11}+0\times2^{10}+1\times2^9+1\times2^8 \ + \ 1\times2^7+1\times2^6+0\times2^5+0\times2^4 \ + \ 0\times2^3+1\times2^2+0\times2^1+1\times2^0$

$(1\times2^3+0\times2^2+1\times2^1+1\times2^0)\times2^8 + (1\times2^3+1\times2^2+0\times2^1+0\times2^0)\times2^4 + (0\times2^3+1\times2^2+0\times2^1+1\times2^0)$

$B\times16^2 \ + \ C\times16^1 \ + \ 5\times16^0$

0xBC5


©RW Fall 2000

## Powers of 2 Table

| n | $2^n$ | n | $2^n$ |
|---|---|---|---|
| 0 | 1 | 6 | 64 |
| 1 | 2 | 7 | 128 |
| 2 | 4 | 8 | 256 |
| 3 | 8 | 9 | 512 |
| 4 | 16 | 10 | 1024 (1K) |
| 5 | 32 | | |


©RW Fall 2000

## Computer Memory

- Memory is a large linear array of 8 bit bytes.
  - Each byte has a unique address (location)
- Bytes are grouped into longer sequences, some of which can be addressed as one unit. Their addresses must be multiples of their byte lengths.
  - Word (4 bytes)
  - Double (8 bytes)

Processor
Memory
CPU
r0
r1
hi
lo
Address.
Data
0
1
2
3
4
0110110
0001100
$2^{n-1}$


©RW Fall 2000

## Basic Data Types

Bit: 0, 1
Bit String: sequence of bits of a particular length
4 bits is a nibble
8 bits is a byte
16 bits is a half-word
32 bits is a word
64 bits is a double-word
Character:
ASCII 7 bit code
Decimal: (BCD code)
digits 0-9 encoded as 0b0000 thru 0b1001
two decimal digits packed per 8 bit byte
Integers:
2's Complement (32-bit representation).
Floating Point:
Single Precision (32-bit representation).
Double Precision (64-bit representation).
Extended Precision(128 -bit representation).

$M \times R^{E}$
exponent
base
mantissa

- How many +/- #'s?
- Where is decimal pt?
- How are +/- exponents represented?


©RW Fall 2000

## ASCII Characters representation

- Each character is represented by a 7-bit ASCII code.
- It is packed into an 8-bit byte.

| Oct. | Chr. | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | nul | 001 | soh | 002 | stx | 003 | etx | 004 | eot | 005 | enq | 006 | ack | 007 | bel |
| 010 | bs | 011 | ht | 012 | nl | 013 | vt | 014 | np | 015 | cr | 016 | so | 017 | si |
| 020 | dle | 021 | dc1 | 022 | dc2 | 023 | dc3 | 024 | dc4 | 025 | nak | 026 | syn | 027 | etb |
| 030 | can | 031 | em | 032 | sub | 033 | esc | 034 | fs | 035 | gs | 036 | rs | 037 | us |
| 040 | sp | 041 | ! | 042 | " | 043 | # | 044 | $ | 045 | % | 046 | & | 047 | ' |
| 050 | ( | 051 | ) | 052 | * | 053 | + | 054 | , | 055 | - | 056 | . | 057 | / |
| 060 | 0 | 061 | 1 | 062 | 2 | 063 | 3 | 064 | 4 | 065 | 5 | 066 | 6 | 067 | 7 |
| 070 | 8 | 071 | 9 | 072 | : | 073 | ; | 074 | < | 075 | = | 076 | > | 077 | ? |
| 100 | @ | 101 | A | 102 | B | 103 | C | 104 | D | 105 | E | 106 | F | 107 | G |
| 110 | H | 111 | I | 112 | J | 113 | K | 114 | L | 115 | M | 116 | N | 117 | O |
| 120 | P | 121 | Q | 122 | R | 123 | S | 124 | T | 125 | U | 126 | V | 127 | W |
| 130 | X | 131 | Y | 132 | Z | 133 | [ | 134 | \ | 135 | ] | 136 | ^ | 137 | _ |
| 140 | ` | 141 | a | 142 | b | 143 | c | 144 | d | 145 | e | 146 | f | 147 | g |
| 150 | h | 151 | i | 152 | j | 153 | k | 154 | l | 155 | m | 156 | n | 157 | o |
| 160 | p | 161 | q | 162 | r | 163 | s | 164 | t | 165 | u | 166 | v | 167 | w |
| 170 | x | 171 | y | 172 | z | 173 | { | 174 | \| | 175 | } | 176 | ~ | 177 | del |


©RW Fall 2000

## Hex ASCII Character representation

- More useful representation:
- (Shows what each byte looks like).

| Hex | Chr. | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | nul | 01 | soh | 02 | stx | 03 | etx | 04 | eot | 05 | enq | 06 | ack | 07 | bel |
| 08 | bs | 09 | ht | 0A | nl | 0B | vt | 0C | np | 0D | cr | 0E | so | 0F | si |
| 10 | dle | 11 | dc1 | 12 | dc2 | 13 | dc3 | 14 | dc4 | 15 | nak | 16 | syn | 17 | etb |
| 18 | can | 19 | em | 1A | sub | 1B | esc | 1C | fs | 1D | gs | 1E | rs | 1F | us |
| 20 | sp | 21 | ! | 22 | " | 23 | # | 24 | $ | 25 | % | 26 | & | 27 | ' |
| 28 | ( | 29 | ) | 2A | * | 2B | + | 2C | , | 2D | - | 2E | . | 2F | / |
| 30 | 0 | 31 | 1 | 32 | 2 | 33 | 3 | 34 | 4 | 35 | 5 | 36 | 6 | 37 | 7 |
| 38 | 8 | 39 | 9 | 3A | : | 3B | ; | 3C | < | 3D | = | 3E | > | 3F | ? |
| 40 | @ | 41 | A | 42 | B | 43 | C | 44 | D | 45 | E | 46 | F | 47 | G |
| 48 | H | 49 | I | 4A | J | 4B | K | 4C | L | 4D | M | 4E | N | 4f | O |
| 50 | P | 51 | Q | 52 | R | 53 | S | 54 | T | 55 | U | 56 | V | 57 | W |
| 58 | X | 59 | Y | 5A | Z | 5B | [ | 5C | \ | 5D | ] | 5E | ^ | 5F | _ |
| 60 | ` | 61 | a | 62 | b | 63 | c | 64 | d | 65 | e | 66 | f | 67 | g |
| 68 | h | 69 | i | 6A | j | 6B | k | 6C | l | 6D | m | 6E | n | 6F | o |
| 70 | p | 71 | q | 72 | r | 73 | s | 74 | t | 75 | u | 76 | v | 77 | w |
| 78 | x | 79 | y | 7A | z | 7B | { | 7C | \| | 7D | } | 7E | ~ | 7F | del |


©RW Fall 2000

## Decimal ASCII Character representation

- More readable representation:

| Dec | Chr. | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | nul | 1 | soh | 2 | stx | 3 | etx | 4 | eot | 5 | enq | 6 | ack | 7 | bel |
| 8 | bs | 9 | ht | 10 | nl | 11 | vt | 12 | np | 13 | cr | 14 | so | 15 | si |
| 16 | dle | 17 | dc1 | 18 | dc2 | 19 | dc3 | 20 | dc4 | 21 | nak | 22 | syn | 23 | etb |
| 24 | can | 25 | em | 26 | sub | 27 | esc | 28 | fs | 29 | gs | 30 | rs | 31 | us |
| 32 | sp | 33 | ! | 34 | " | 35 | # | 36 | $ | 37 | % | 38 | & | 39 | ' |
| 40 | ( | 41 | ) | 42 | * | 43 | + | 44 | , | 45 | - | 46 | . | 47 | / |
| 48 | 0 | 49 | 1 | 50 | 2 | 51 | 3 | 52 | 4 | 53 | 5 | 54 | 6 | 55 | 7 |
| 56 | 8 | 57 | 9 | 58 | : | 59 | ; | 60 | < | 61 | = | 62 | > | 63 | ? |
| 64 | @ | 65 | A | 66 | B | 67 | C | 68 | D | 69 | E | 70 | F | 71 | G |
| 72 | H | 73 | I | 74 | J | 75 | K | 76 | L | 77 | M | 78 | N | 79 | O |
| 80 | P | 81 | Q | 82 | R | 83 | S | 84 | T | 85 | U | 86 | V | 87 | W |
| 88 | X | 89 | Y | 90 | Z | 91 | [ | 92 | \ | 93 | ] | 94 | ^ | 95 | _ |
| 96 | ` | 97 | a | 98 | b | 99 | c | 100 | d | 101 | e | 102 | f | 103 | g |
| 104 | h | 105 | i | 106 | j | 107 | k | 108 | l | 109 | m | 110 | n | 111 | o |
| 112 | p | 113 | q | 114 | r | 115 | s | 116 | t | 117 | u | 118 | v | 119 | w |
| 120 | x | 121 | y | 122 | z | 123 | { | 124 | \| | 125 | } | 126 | ~ | 127 | del |


©RW Fall 2000

## Binary representations for Integers

(signed numbers)

- Binary numbers:
  - Base 2 numbers, only two digits {0, 1}
    $I = 100101_2$ = 1 0 0 1 0 1
  - $I = 37_{10} \leftarrow 1*2^5 + 0*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0$
- Sign-Magnitude := Highest order bit is the sign bit
  - Example: $010110_2 = 22_{10}$ ; $110110_2 = -22_{10}$
- 2's Complement := $I = a_{n-1} * ( - 2^{n-1} ) + a_{n-2} * 2^{n-2} + \dots a_0 * 2^0$
  - Example: $010110_2 = 22_{10}$ ; $101010_2 = -22_{10}$ (6-bit 2's comp.)
  - Examples: $0_{10} = 000000_2$; $1_{10} = 000001_2$; $-1_{10} = 111111_2$


©RW Fall 2000

## Binary representations for Integers Sign-Magnitude

- Advantages:
  - Simple extension of unsigned numbers.
  - Same number of positive and negative numbers.
- Disadvantages:
  - Two representations for 0:= 0=000000; -0=100000.
  - Algorithm (circuit) for addition depends on the arguments' signs.


©RW Fall 2000

## Binary representations for Integers 2's Complement

- Advantages:
  - Only one representation for 0:= 0 = 000000
  - Addition algorithm independent of sign bits.
- Disadvantages:
  - One more negative number than positive : Example: $100000_2 = -32_{10}$ ; but $32_{10}$ could not be represented as a 6-bit 2's complement number.


©RW Fall 2000

## Binary Arithmetic

- Addition
  - As in DECIMAL arithmetic, add low order column of digits up, forming a digits sum DS
  - Record the sum-digit for this column as DS mod 2
  - Include a CARRY of floor(DS/2) in the next column's sum
  - Example: $18_{10} + 14_{10} = 32_{10}$
    - $18_{10} = 010010_2$
    - $14_{10} = 001110_2$

$$\begin{array}{r} 010010_2 \\ +001110_2 \\ \hline 100000_2 \end{array}$$


©RW Fall 2000

## Binary Arithmetic 2

- Shifts
  - A LEFT shift of K bit positions MULTIPLIES the number by $2^K$
    - $18_{10}$ << 2 is $010010_2$ << 2 == $01001000_2$ == $64_{10}$ +$8_{10}$ == 72 == 18*4
  - A RIGHT shift of K bit positions DIVIDES the number by $2^K$ dropping the remainder (low-order digits)
    - $18_{10}$ >> 2 is $010010_2$ >> 2 == $0100_2$ == 4
  - The REMAINDER left when a number is divided by $2^K$ is the LOW-ORDER K BITS
    - $18_{10}$ % 4 is $010010_2$ % $2^2$ == $0100.10_2$ == $10_2$ == 2


©RW Fall 2000

## 2's Complement (cont.) Complementing (negating)

- To negate a number do:
  - A: complement each digit (Change 0 to 1, and 1 to 0)
  - B: add 1.
  - Example: $14_{10} = 001110_2$ ; $-14_{10} = 110001_2 + 1 = 110010_2$
- To add signed numbers use regular addition but disregard carry out.
  - Example: $18_{10} - 14_{10} = 18_{10} + (-14_{10}) = 4_{10}$

$$\begin{array}{r} 010010_2 \\ +110010_2 \\ \hline 000100_2 \end{array}$$


©RW Fall 2000

## Why 2's Complement Addition Works (1)

- Examples for 5-bit 2's complement integers

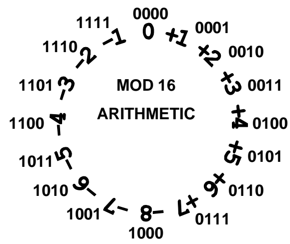
$4_{10}$ + ( - $5_{10}$ ) = - 1
00100 + ( - 00101 )
00100 + 11011 = 11111 = - ( 00001 ) = -1

(- $4_{10}$ ) + ( - $5_{10}$ ) = - 1
( - 00100 ) + ( - 00101 )
11100 + 11011 =r 1 10111
=m 10111 = - ( 01001 ) = - $9_{10}$


©RW Fall 2000

## How 2's Complement Works (4 bit)

❍






©RW Fall 2000

## Why 2's Complement Addition Works (2)

- N-bit 2's complement arithmetic is exact arithmetic mod $2^N$
  - Suppose J, K are small positive integers, J>K.
  - R(-J) = ~J + 1 = 111…111 (N 1's) -J+1 = $2^N$ -1 - J + 1.
    R(- J) = $2^N$ - J, R(- K) = $2^N$ - K, R(J) = J, R(K) = K

R(- J) + R(- K) = $2^N$ + ( $2^N$ - J - K) = ( $2^N$ - J - K) mod $2^N$
R( J) + R(- K) = $2^N$ + ( J - K) = ( J - K) mod $2^N$
( Because J-K is greater than 0 )
R(- J) + R( K) = ( $2^N$ - J + K) = ( $2^N$ - (J - K)) mod $2^N$
( Because -J+K is less than 0 )
R( J) + R( K) = ( J + K) = ( J + K) mod $2^N$



©RW Fall 2000

## 2's Complement (cont.)

Example: A = 0x0ABC; B = 0x 0FEB.

Compute: A + B and A - B in 16-bit 2s complement arithmetic.



©RW Fall 2000

## 2's Complement (cont.)
## Precision Extension

- To extend precision do: sign bit extension:

Example:

- $14_{10} = 001110_2$ in 6-bit representation.
  $14_{10} = 000000001110_2$ in 12-bit representation
- $-14_{10} = 110010_2$ in 6-bit representation
  $-14_{10} = 111111110010_2$ in 12-bit representation.


©RW Fall 2000

## Changing base representation

- Suppose you can do arithmetic in base Z, and want to convert a number N from base Z to base B
- Write $N = q \times B + r,$ where
  $$q = \sum_{i=1} N_i \times B^{i-1}, \quad r = N_0$$
  - R is the low-order digit of N in base B, and q can be converted to get the remaining digits
  - Like shifting the base-point:
    $N = N_k N_{k-1} \ldots N_1 . N_0 \times B$
  - For fractions, MULTIPLY by B; the integer part is the high-order digit, and the fraction holds the other digits:
    $N \times B = N_k . N_{k-1} \ldots N_1 N_0$
- To convert TO base Z, just evaluate the sum:
  $$N = \sum_{i=0} N_i \times B^i$$


©RW Fall 2000

## Limits on 2's Complement Arithmetic

- N-bit 2's complement represents a finite RANGE of #s
  - Numbers outside this range cannot be represented
  - Arithmetic operation can produce unrepresentable #s
  - Only values of I satisfying eqn below can be represented:
    $$-2^{N-1} \le I \le 2^{N-1} - 1$$
- When operation gives an out-of-range answer A, machine gives A', where R(A') = R(A) mod $2^N$
- Examples (6-bit 2's complement)

```
  100010 = -30         001101 =  13
+ 100001 = -31       + 011000 =  24
  000011 =   3         100101 = -27
```


©RW Fall 2000

## 2's Complement with Implicit Binary Point

- 2's complement numbers need not represent integers -- each can have a "hidden" binary point, and thus represent a number with an integral part, AND a fractional part:

| | |
|---|---|
| $010010_2 = 18_{10}$ | $010.010_2 = 2.25_{10}$ |
| $+001011 = 11$ | $+001.011 = 1.375$ |
| $011101 = 29$ | $011.101 = 3.625$ |

- Addition (IF the binary points are aligned) produces the same sequence of bits, regardless of the position of the "."
- This property underlies floating point addition
  - Uses a "significand", F, with a hidden, implicit binary point, multiplied by $2^E$, where the number representation includes fields that give values for "E" and "F".


©RW Fall 2000

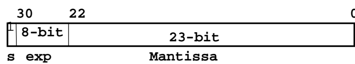
## Floating Point Representation

Numbers are represented by:

$$X = (-1)^s \times 2^{E-127} \times 1.M$$

S := 1-bit field ; Sign bit
E := 8-bit field; Exponent: Biased integer, $0 \le E \le 255$.
M:= 23-bit field; Mantissa: Normalized fraction with hidden 1.
F := 1.M = Significand

### Single precision floating point number





©RW Fall 2000

## Floating Point Representation
(cont.)

- The mantissa represents a fraction using binary notation:

$$M = .\,s_1, s_2, s_3 \ldots = 1.0 + s_1{*}2^{-1} + s_2{*}2^{-2} + s_3{*}2^{-3} + \ldots$$

- Example: $X = -0.75_{10}$ in single precision:

$-0.75_{10} = -0.11_2 = (-1) \times 1.1_2 \times 2^{-1} = (-1) \times 1.1_2 \times 2^{126-127}$
S = 1 ; Exp = $126_{10}$ = $0111\ 1110_2$ ;
M = $100\ 0000\ 0000\ 0000\ 0000\ 0000_2$

```
     31 30       23 22                         0
X = |1|0111 1110|100 0000 0000 0000 0000 0000|
```


©RW Fall 2000

## Floating Point Representation
(cont.)

**Example:**

**What floating-point number is : 0xC1580000 ?**



©RW Fall 2000

## Floating Point Representation
(cont.)

**Example:**
**What floating-point number is : 0xC1580000 ?**
**1100 0001 0101 1000 0000 0000 0000 0000**

| S | Exp 8-bit | Mantissa 23 - bit |
|---|---|---|
| 1 | 100 0001 0 | 101 1000 0000 0000 0000 0000 |

**-** E=128+2-127=3 $-1.1011_2 \times 2^3 = -1101.1_2 =$ -13.5



©RW Fall 2000

## Floating Point Arithmetic

- **To add FP #s, Equalize Exponents, then add Adjusted Significands.**
  - **Extend M's, by including the hidden leading "1": gives F's.**
  - **Shift $F_A$ of #, A, with SMALLER exponent right**
    - **Each 1-bit right-shift adds 1 to $E_A$**
    - **Shift until $E_A = E_B$**
  - **Add shifted $F_A$ to $F_B$ to get $F_S$ observing sign rules**
  - **Re-normalize (shift $F_S$ left, reducing $E_s$, till leading 1 disappears)**
  - **Significant bits may be lost on the RIGHT. Approximate answer. "Weight" of lost bits is (hopefully) small.**
- **Numbers out of range produce "special" representations, using E field of all 1's, or all 0's.**
  - **NaN's, and 0 See TABLE, end of Sect. 4.8, p 301**



©RW Fall 2000

## FP addition example

A = 0xC1D80000, B=0xC2E80000
0xC1D80000 = 1 | 100 0001 1 | 101 1000 0000 …
0xC2E80000 = 1 | 100 0010 1 | 110 1000 0000 …
$F_A$ = 1.1011000
Adjust $F_A$: 5-3=2 shifts: new $F_A$ = 0.011011000

$F_B$ = 1.1101000, $E_B$=100 0010 1
shifted $F_A$ = 0.0110110
$F_S$ =10.0011110 , $E_S$=100 0010 1
Re-normalize $F_S$ = 1.00011111, $E_S$=100 0011 0
Result = 1 | 100 0011 0 | 000 1111 1 = 0xC30F80000

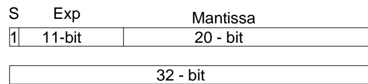

©RW Fall 2000

## Floating Point Representation
(cont.)

- Double Precision Floating point:

  64-bit representation: 1-bit sign, 11-bit (biased) exponent; 52-bit mantissa (with hidden 1).

### Double precision floating point number




$$X = (-1)^s \times 1.M \times 2^{(E-1023)}$$


©RW Fall 2000