Chapter 1:

Integer representations in computer systems (Review)

Topics:

Unsigned representations
Signed representations
Signed binary and hexadecimal arithmetic

Reading:

5th Ed. Patterson and Hennessy 2.4, 3.2

Internal Representation of Integers

Unsigned integers (non-negative)
signed integers
Fixed number of digits

Decimal number system

* base (or radix) = 10
* digits: 0, 1, 2, ... , 9

Example:

739 = $7 \times 10^2 + 3 \times 10^1 + 9 \times 10^0$

Rules:
1)number digit positions 0,1,2..., right to left

2)multiply by radix to (digit position) power

Binary number system

* base 2
* digits: 0, 1

* binary digit: **bit**

Example:

Convert this binary number to decimal:

$$(\mathbf{1011})_2$$

$= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
$= 8 + 0 + 2 + 1$
$= 11$

Useful information: powers of 2 are...
$2^4 = 16$, $2^5 = 32$, $2^6 = 64$, $2^7 = 128$
$2^8 = 256$, $2^9 = 512$, $2^{10} = 1024$

Table 1: All possible unsigned 4-bit binary numbers

binary	decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Example: 8-bit unsigned binary number

$$(10010101)_2$$

Convert to decimal:

$= 2^7 + 2^4 + 2^2 + 2^0$
$= 128 + 16 + 4 + 1$
$= 149$

Computer hardware is based on digital logic circuits; data is represented using binary system.

Convert decimal numbers to binary numbers

Method 1 (slow but intuitive):

break up into sum of powers of 2

Example:

$$(249)_{10}$$

$= 128 + 64 + 32 + 16 + 8 + 1$
$= 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^0$
$= (1111\ 1001)_2$

Convert decimal numbers to binary numbers

Method 2 (shortcut):

1) divide by 2
2) write down remainder
3) repeat until quotient = 0
4) combine remainders in reverse order (bottom to top)

Example:

$$(\mathbf{249})_{10}$$

249/2 = 124 rem = 1
124/2 = 62 rem = 0
62/2 = 31 rem = 0
31/2 = 15 rem = 1
15/2 = 7 rem = 1
7/2 = 3 rem = 1
3/2 = 1 rem = 1
1/2 = 0 rem = 1

$249 = (1111\ 1001)_2$

Hexadecimal (hex) number system

* base 16
* digits: 0, 1, 2, ..., 9, A, B, C, D, E, F

(shorthand for binary)

Example: (0x means base 16)

Convert 3-digit hex int to decimal:

$$\mathbf{0}\boldsymbol{x}\mathbf{38}\boldsymbol{F}$$

$= 3 \text{ x } 16^2 + 8 \text{ x } 16^1 + 15 \text{ x } 16^0$
$= 3 \text{ x } 256 + 8 \text{ x } 16 + 15$
$= 911$

Convert unsigned decimal int to hex

Method 1: break into sums of powers of 16

$$\mathbf{(574)_{10}}$$

$= 2 \text{ x } 256 + 3 \text{ x } 16 + 14$
$= 2 \text{ x } 16^2 + 3 \text{ x } 16^1 + 14 \text{ x } 16^0$
$= \text{0x23E}$

Method 2: divide repeatedly by 16

$$\mathbf{(574)_{10}}$$

574/16 = 35, rem = 14 (which is 0xE)
35/16 = 2, rem = 3
2/16 = 0, rem = 2

$(574)_{10}$ = 0x23E

Convert hexadecimal numbers to binary numbers

one hexadecimal digit = 4 bits
start at least significant (rightmost) digit

Example:

0x5B3

5 = 0101, B = 1011, 3 = 0011

0x5B3 = $(0101\ 1011\ 0011)_2$

Convert binary ints to hex (reverse):

group bits into groups of 4
start at right most bit

Octal number system

* base 8
* digits: 0, 1, 2, ..., 7

Convert octal to decimal:

1)number digit positions 0,1,2... right to left
2) multiply each digit by radix to (digit position) power

Convert octal to binary:

each octal digit = 3 bits

Unsigned integer addition

Binary:

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

Hexadecimal:

```
    3 d e
+   7 8 2
---------
    b 6 0
```

Unsigned integer subtraction

Binary:

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


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

Systems for representing signed integers:

1. sign-magnitude (SM)
2. one's complement (OC)   (*skip*)
3. two's complement (TC)

Binary Sign-magnitude

Rules:

1) most significant (leftmost) bit is sign bit
non-negative if sign bit = 0,
negative if sign bit = 1

2) non-negative numbers same as unsigned

3) rest of bits represents magnitude of integer

Convert 8-bit binary SM to decimal:

0010 0101 = + 010 0101 = 32 + 4 + 1 = 37
Convert binary SM to decimal:
1110 0101 = - 0110 0101
= - (64 + 32 + 4 + 1)
= - 101

<u>Convert decimal to 8-bit binary SM:</u>

X = -13 = ??

-X = 13 = 0000 1101
X = 1000 1101

Dirty zero problem in SM:
$(0000)_2 = 0$
$(1000)_2 = -0 = 0$

Two different representations for zero
Not suitable for fast hardware implementations

One's Complement (OC):

Rules:

1) most significant (leftmost) bit is sign bit
non-negative if sign bit = 0,
negative if sign bit = 1

2) non-negative numbers same as unsigned

3) to negate a OC binary int:

flip/complement each bit

Convert 8-bit OC binary to decimal:

$(1100\ 0101)_2 = -\ (0011\ 1010)_2$
$= -\ (32+16+8+2) = -\ 58$

Convert decimal to 8-bit OC binary:

-58 = ??
Let X = -58
$-X = 58 = (0011\ 1010)_2$
$X = (1100\ 0101)_2$

Dirty zero problem in OC:

$(0000)_2 = 0$
$(1111)_2 = -0 = 0$

Two’s complement

Rules:
1) most significant (leftmost) bit is sign bit
non-negative if sign bit = 0,
negative if sign bit = 1

2) non-negative numbers same as unsigned

3) to negate a two’s complement binary number,

i. complement (or flip) each bit
ii. add 1 **(discard carry out)**

Example:

Convert 8-bit TC binary int to decimal:

= 32 + 4 + 2 = 38

Example:

Negate TC binary int:

X = $(0010\ 0110)_2$

-X = $(1101\ 1001)_2 + 1 = (1101\ 1010)_2$

Example:

Convert -29 to 8-bit TC binary:

Let X = -29
-X = 29 = 0001 1101
X = 1110 0010 + 1 = 1110 0011

<u>Shortcut for negating binary TC int:</u>

1) look for rightmost bit == 1
2) complement each bit to the left
3) (all other bits stay the same)

Example:

X = $(1011\ \underline{1}000)_2$ [rightmost 1 is underlined]
-X = $(01001000)_2$

<u>Another way to convert TC binary int to decimal:</u>
* for n-bit ints, digit position n-1 is -2^(n-1)

Examples:

$X = (1110\ 0011)_2$
$= -128 + 64 + 32 + 2 + 1 = -29$

No dirty zeros in TC:
$(0000)_2 = 0$
Try to construct negative zero:
$-\ (0000)_2 = (1111)_2 + 1 = 1\ 0000$

Discard carryout: $-0 = (0000)_2$!

Sign-extension in TC:
(writing the same integer, but with more bits)

4-bit to 8-bit: $(0101)_2 = (0000\ 0101)_2$
4-bit to 8-bit: $(1101)_2 = (1111\ 1101)_2$

Extend (or duplicate) the sign bit.

Comparison of four integer systems (4 bits):

	Unsigned	SM	OC	TC
0000	0	0	0	0
0001	1	1	1	1
0010	2	2	2	2
0011	3	3	3	3
0100	4	4	4	4
0101	5	5	5	5
0110	6	6	6	6
0111	7	7	7	7
1000	8	0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	0	-1

Range of binary integer representations:

Unsigned:
4-bit: 0 to 15 = 0 to $(2^4 - 1)$

N-bit: 0 to $(2^N - 1)$

SM:
4-bit: -7 to 7 = $-(2^3 - 1)$ to $(2^3 - 1)$

N-bit: $-(2^{N-1} - 1)$ to $(2^{N-1} - 1)$

TC:
4-bit: -8 to 7 = (-2^3) to $(2^3 - 1)$

N-bit: (-2^{N-1}) to $(2^{N-1} - 1)$

Range of integer data types:

short (usually 16 bits)

(-2^{16-1}) to $(2^{16-1} - 1)$, or
-32768 to 32767

int (usually 32 bits)

(-2^{32-1}) to $(2^{32-1} - 1)$, or
-2,147,483,648 to 2,147,483,647

long (usually 64 bits)

(-2^{64-1}) to $(2^{64-1} - 1)$

Two's complement arithmetic

similar to unsigned

1) treat sign bit as numeric bit
2) if carry out is produced, discard it
3) check for overflow

Examples:

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

discard carryout: ans = 1011 0111

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

sum of two non-negative numbers cannot give negative result! *Overflow* occurred.

$$
\begin{array}{cl}
 & 1\,0\,0\,0\,0\,0\,0\,0 \\
+ & 1\,1\,1\,1\,1\,1\,1\,1 \\
\hline
1 & 0\,1\,1\,1\,1\,1\,1\,1
\end{array}
$$

sum of two negative integers cannot give non-negative result! *Overflow* occurred.

Definition of overflow:

Overflow is an error condition in which the result of a computation does not fit into the available number of bits.

In TC arithmetic:
1) if we add 2 positive integers and get a negative result, or
2) if we add 2 negative integers and get a positive result,
we have overflow.

Consider C++ code:

```
int sum = 0, i;
  for (i=0; i<10; i++) {
    sum = sum + 1000000000;
    cout << sum << endl;
  }
```

Compile and run; program output:

```
1000000000
2000000000
-1294967296
-294967296 [etc etc]
```

Two’s complement subtraction

X - Y = X + (-Y)

Example:

X = 1001 0010
Y = 0101 1111

-Y = 1010 0001

X - Y = 1001 0010 + 1010 0001
= 1 00110011

Discard carryout; result = 0011 0011

overflow occurred

[Note: for subtraction, check X + (-Y), using the overflow check for addition!]

Two’s complement hexadecimal

Rules:
1) most significant (leftmost) digit is sign digit
non-negative if sign digit =
negative if sign digit =

2) non-negative integers same as unsigned

3) to negate a two’s complement hex int,

i. subtract each digit from f
ii. add 1 **(discard carry out)**

Examples:

Convert 2-digit TC hex int to decimal

X = 0xab
- X = 0xff - 0xab + 1 = 0x55
= 5 x 16 + 5 = 85
X = - 85

Convert decimal int to 2-digit TC hex:

X = -73

-X = 73 = 4 x 16 + 9 = 0x49
X = 0xff - 0x49 + 1 = 0xb6 + 1 = 0xb7

TC hex addition/subtraction:

0x5678 + 0x432b

0x5678
0x432b
0x99a3, overflow

0xdcba + 0xe2f3

0xdcba
0xe2f3
0x1bfad
discard carryout; result = 0xbfad

0x1cba - 0xbd0d

= 0x1cba + - (0xbd0d)
= 0x1cba + 0xffff - 0xbd0d + 1
= 0x1cba + 0x42f3

0x1cba
0x42f3
0x5fad

Lab exercises for Chapter 1:

Lab 1.1: Binary and hexadecimal exercises
Lab 1.2: Setting up spim / xspim

CSc 256 Practice Exercise #1 on arithmetic

This file can be found at

http://userwww.sfsu.edu/~whsu/csc256/LABS/DOCS/ex1.txt

For problems 1-9, solutions can be found in

http://userwww.sfsu.edu/~whsu/csc256/LABS/DOCS/ex1soln.txt

1) Consider the 16-bit binary integer X = 1001 0000 0000 0011.

Convert X to decimal if X is
a. unsigned   b. in sign-magnitude notation
c. in one's complement notation   d. in two's complement notation

For problems 2-6, assume all integers are in binary and in two's complement notation. Remember to indicate overflow if necessary.

2) 0110 1010 + 1001 1110 = ?

3) 1001 1111 + 1001 0001 = ?

4) 1000 1111 - 0001 0000 = ?

5) 0001 0010 - 0010 1111 = ?

6) 1111 1010 - 1110 1110 = ?

For problems 7-9, assume all integers are in hexadecimal and two's complement notation. Remember to indicate overflow if necessary.

7) 0x2AF6 + 0x7017 = ?

8) 0x345E + 0xFFAB = ?

9) 0x966A - 0x6996 = ?