Representing Data:
Integer Arithmetic and Hexadecimal

For use in CSE6010 only

Not for distribution

Two’s Complement Review

Last time we saw several definitions for the n-bit two’s complement representation –X of an n-bit number X:

- The n-bit two’s complement representation of -X is $2^n – X$
- The two’s complement of X is the (one’s complement of X) + 1
- In the two’s complement representation, the sign bit has negative weight

1 1 1 0 0 0 1 1 = -29 $\quad 1*(-2^7) + 1*2^6 + 1*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0$

n=8 $\quad = -128 + 64 + 32 + 0 + 0 + 0 + 2 + 1$

$= -29$

Fast way to generate two’s complement numbers

- Starting at least significant bit (rightmost), scan right to left
- Copy bits up to and including the first ‘1’ bit
- Complement (invert) each of the remaining bits

0 1 0 1 1 | 1 0 0 = 92

1 0 1 0 0 | 1 0 0 = -92

complement | copy

$-2^7 + 2^5 + 2^2 = -128 + 32 + 4 = -92$

Observations

For n-bit two’s complement numbers

- The largest positive number that can be represented is $2^{n-1} - 1$

n=8   0 1 1 1 1 1 1 1 = 127

- The largest magnitude negative number that can be represented is -2^{n-1}

n=8   1 0 0 0 0 0 0 0 = -128

- Zero has a single representation (all 0’s)
- -1 is represented as all 1’s: 11111111 (n=8)
- The number scale is not symmetric (one more negative number than positive numbers): no positive counterpart to -2^{n-1}

Two's Complement Arithmetic

Addition

- Simply add using binary addition
- Ignore carry out of most significant bit

$$
\begin{array}{rl}
 & 1\,0\,1\,0\,0\,1\,0\,0 \;= -92 \\
+ & \underline{1\,1\,1\,0\,0\,0\,1\,1} \;= -29 \\
 & 1\,0\,0\,0\,0\,1\,1\,1 \;= -121
\end{array}
$$

Subtraction

- Take two's complement of number being subtracted (subtrahend)
- Perform addition

92 - 29

$$
\begin{array}{rl}
 & 0\,1\,0\,1\,1\,1\,0\,0 \;= 92 \\
+ & \underline{1\,1\,1\,0\,0\,0\,1\,1} \;= -29 \\
 & 0\,0\,1\,1\,1\,1\,1\,1 \;= 63
\end{array}
$$

This leads to simple circuits to perform addition/subtraction

Overflow

Overflow (OF): adding two n-bit numbers can yield a result that cannot be represented in n-bits

- OF Test 1: The sign of both operands is the same, but the sign of the result is different

$$\begin{array}{rl} & 0\,1\,0\,1\,0\,1\,0\,1 = 85 \\ + & \underline{0\,1\,1\,0\,1\,0\,0\,0} = 104 \\ & 1\,0\,1\,1\,1\,1\,0\,1 = -67 \end{array}$$

Adding two positives cannot give negative
Adding two negatives cannot give positive

- OF Test 2: The carry into the most significant bit is different from the carry out

$$\begin{array}{rl} & 0\ \ 1 \\ & 0\,1\,0\,1\,0\,1\,0\,1 = 85 \\ + & \underline{0\,1\,1\,0\,1\,0\,0\,0} = 104 \\ & 1\,0\,1\,1\,1\,1\,0\,1 = -67 \end{array}$$

We need another bit to represent the sum

Easy for hardware to detect this condition

Sign Extension

- Sometimes it is necessary to increase the precision (number of bits) used to represent a number
- In two’s complement this is accomplished by replicating the sign bit to the left (higher precision bit positions)

8 bits:	0 1 0 1 1 1 0 0	= 92
16 bits:	0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0	= 92
8 bits:	1 1 1 0 0 0 1 1	= -29
16 bits:	1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1	= -29

Integer Arithmetic Examples

- Perform the following additions and verify your answers. Indicate whether there is potential for overflow and, if so, whether overflow occurs.
  - 10101 + 00111
  - 1001 + 1101

Integer Arithmetic Examples

- 10101 + 00111:
  Sign bits are opposite, so no potential for overflow

$$\begin{array}{r} 10101 \\ +\ \underline{00111} \\ 11100 \end{array}$$

  Verify: 10101 is $-2^4+2^2+2^0 = -16+4+1 = -11$
  00111 is $2^2+2^1+2^0 = 4+2+1 = 7$
  Sum of -11 and 7 is -4
  Binary representation of 4 is 00100; two's complement -4 is 11100
- 1001 + 1101
  Sign bits are the same; potential for overflow

$$\begin{array}{r} 1001 \\ +\ \underline{1101} \\ 0110 \end{array}$$

  Overflow has occurred! Sign bit has changed
  Verify: 1001 is $-2^3+2^0 = -8+1 = -7$; 1101 is $-2^3+2^2+2^0 = -8+4+1 = -3$
  Sum is $-7 + -3 = -10$
  Most negative integer with 4 bits is $-2^{4-1} = -2^3 = -8$; overflow confirmed!

Multiplication

- Multiplication can be implemented using addition

$$
\begin{array}{r l}
010 & = 2 \\
\times\ 011 & = 3 \\
\hline
010 & \\
010 & \\
000 & \\
\hline
000110 & = 6
\end{array}
$$

- Multiplication by 2 can be achieved by shifting bits left one position (put 0 into least significant bit position, discard most significant bit)

$$
\begin{array}{l l}
1\,1\,1\,0\,1\,0\,1\,0 & = -22 \\
1\,1\,0\,1\,0\,1\,0\,0 & = -44
\end{array}
$$

- Multiplication by 2^i can be achieved by shifting bits left i positions

Division

- Division can be implemented using subtraction (similar to long division you learned in elementary school)
  - Slow: each step produces one digit of quotient
- Faster: Perform A / B by
  - Compute 1 / B using Newton-Raphson algorithm* (number of result bits approximately doubles every step)
  - Multiply result by A
- Division by 2^i achieved by shifting right i bit positions (replicate sign bit, discard least significant bits)

-24 / 4: 1 1 1 0 1 0 0 0 = -24

1 1 1 1 1 0 1 0 = -6

* see https://en.wikipedia.org/wiki/Division_algorithm for details

Hexadecimal (base 16)

Digits: use letters to represent digits 10-15

More compact way to write binary strings

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Hexadecimal number: 04EB

$0*16^3 + 4*16^2 + 14*16^1 + 11*16^0 = 1259$

Binary representation of same value:

0000 0100 1110 1011 = 1259

Conversion from binary to hexadecimal: start from right, replace each 4 bit group with hexadecimal digit

$$\underbrace{0000}_{0}\ \underbrace{0100}_{4}\ \underbrace{1110}_{E}\ \underbrace{1011}_{B}$$

Hexadecimal a more convenient way to present binary data on paper

Integer Arithmetic Examples

- Use bit shifts to quickly compute the following in decimal, starting from $(0000\ 1010)_2 = 10$
  - $(0001\ 0100)_2$
  - $(0101\ 0000)_2$
  - $(0000\ 0101)_2$
- Perform the following conversions:
  - $(0110\ 1011\ 0010\ 1110)_2$ to hexadecimal
  - C9F to binary

Integer Arithmetic Examples

- Use bit shifts to quickly compute the following in decimal, starting from $(0000\ 1010)_2 = 10$
  - $(0001\ 0100)_2 = 10*2 = 20$
  - $(0101\ 0000)_2 = 10*2^3 = 80$
  - $(0000\ 0101)_2 = 10/2 = 5$
- Perform the following conversions:
  - $(0110\ 1011\ 0010\ 1110)_2$ to hexadecimal: 6B2E
  - C9F to binary: 1100 1001 1111

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Summary of Representing Integers

- We can represent 2^n integers using n bits; specific numbers depend on choice of signed or unsigned
- Most computers use two’s complement to represent signed integers
- Shift operations are very fast
- Hexadecimal is a compact way of writing large numbers