計算機構成論 第3回
―計算機における数値表現(2)―

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

講義内容

- 2進数の演算
  - 符号拡張
  - シフト演算
  - 加算・減算
  - 予習的な内容

2進数の符号拡張

- より多いビット数で同じ値を表現する方法
  - e.g., 8ビット2進数→16ビット2進数
- 新しい値の上位ビットを、元の値の最上位ビットで埋めればOK
  - 負数は2の補数表現が前提

8bit	00001111	15_{10}
16bit	0000000000001111	
8bit	11110001	-15_{10}
16bit	1111111111110001	

符号拡張の方法の正当性

- 新しい値の上位ビットを、元の値の最上位ビットで埋めればOK
  - 正の場合は自明
  - 負の場合

$-2^7 2^6$

8bit 11110001

16bit 1111111111110001

$-2^{15} 2^{14}$ …… $2^8 2^7$

-15_{10}

$-2^n=2^n-2^{n+1}$

$-2^n=2^n+2^{n+1}-2^{n+2}$

⋮

$-2^7=2^7+2^8+\ldots+2^{14}-2^{15}$

確認問題

- (1) 00101010_2 を16ビットに符号拡張せよ。
- (2) 10101010_2 を16ビットに符号拡張せよ。
- (3) (2)の変換前後の10進数が一致することを確認することで、行った符号拡張が正しいことを確かめよ。

講義内容

- 2進数の演算
  - 符号拡張
  - シフト演算
  - 加算・減算
  - 予習的な内容

シフト演算 (shift operation)

- 2進数のビットを左右にずらす演算
  - 1ビット左シフトの場合

$$00001100_2 = 8+4 = 12_{10}$$

2倍

$$00011000_2 = 16+8 = 24_{10}$$

符号はとりあえず
考えない

最上位ビット
は捨てる

最下位ビット
には0が入る

  - 2ビット右シフトの場合

$$00011000_2 = 16+8 = 24_{10}$$

1/4倍

$$00000110_2 = 4+2 = 6_{10}$$

シフト演算 (shift operation)

- 左端、右端に注意

左に1ビットシフト

11000000_2 = 128+64 = 192_{10}

2倍にならない

10000000_2 = 128 = 128_{10}

右に1ビットシフト

00001001_2 = 8+1 = 9_{10}

1/2倍
(小数点以下切り捨て)

00000100_2 = 4 = 4_{10}

算術シフト

- 符号付きの場合
  - 算術シフト

右に1ビットシフト

$11111000_2 = -8_{10}$

$11111100_2 = -4_{10}$

左に1ビットシフト

$11111000_2 = -8_{10}$

$11110000_2 = -16_{10}$

論理シフト

符号ありの場合は
論理シフトしては
いけない

- 符号なしの場合
  - 論理シフト

右に1ビット
シフト

$11111000_2 = 248_{10}$ ✕ -8_{10}

$01111100_2 = 124_{10}$ 124_{10}

左に1ビット
シフト

$01111000_2 = 120_{10}$ 120_{10}

$11110000_2 = 240_{10}$ ✕ -16_{10}

シフト演算の注意点

- 符号の有無に注意
  (符号ありなら算術シフト)
- 必ずしも×2^n、/2^nになるとは限らない
  - オーバフロー
  - 端数の切り捨て

00000011 (3_{10}) を右に1ビット算術シフト
→ 00000001 (1_{10})

11111101 (= -3_{10}) を右に1ビット算術シフト
→ 11111110 (-2_{10})

確認問題

- (1) 11101010_2 を1ビット左に算術シフトせよ。
- (2) 10101010_2 を1ビット右に算術シフトせよ。
- (3) 10101010_2 を1ビット左に論理シフトせよ。
- (4) 10101010_2 を1ビット右に論理シフトせよ。

講義内容

- 2進数の演算
  - 符号拡張
  - シフト演算
  - 加算・減算
  - 予習的な内容

2進数の加算

10進数の場合と
同じように
筆算ができる

正数＋正数

$$
\begin{array}{rrl}
 & 00001111 & 15_{10} \\
+ & 00000010 & 2_{10} \\
\hline
 & 00010001 & 17_{10}
\end{array}
$$

正しい

$$
\begin{array}{rrl}
 & 01111110 & 126_{10} \\
+ & 00000010 & 2_{10} \\
\hline
 & 10000000 & 128_{10}
\end{array}
$$

桁あふれ
(オーバーフロー:overflow)

どのように対処するかは、
プログラミング言語等による

✖ これは-128

2進数の減算

- 正数＋負数 を行えば良い

	00001111	15_{10}	
+	11111110	-2_{10}	正しい
	00001101	13_{10}	

ここの繰り上がりは無視

	00001111	15_{10}	
+	11101101	-19_{10}	正しい
	11111100	-4_{10}	

2進数の減算

- 負数＋負数 の場合

	11110001	-15_{10}	
+	11111110	-2_{10}	正しい
	11101111	-17_{10}	

	10000011	-125_{10}	
+	11111100	-4_{10}	オーバーフロー
	01111111	-129_{10}	

✖ これは127

加算でのオーバーフロー条件

- 正数と正数、負数と負数の場合
  - 計算結果の符号が元の値と異なれば、オーバーフロー
- 正数と負数の場合
  - オーバーフローにはならない

確認問題

- 2つの2進数A, Bがある。
A-Bの結果を求める場合、
まず、引く数Bの(1)を取り、
Aとの和を求めればよい。

確認問題

- 8ビット2進数で以下を計算せよ
  - (1) (-1) + 1
  - (2) 3 + (-2)
  - (3) 3 + (-5)

加算する2つの数を2進数で表現して
足し算してください

講義内容

- 2進数の演算
  - 符号拡張
  - シフト演算
  - 加算・減算
  - 予習的な内容

2進数の乗算

- 正数×正数
  - 10進数と同じように筆算ができる
  - 桁数は乗数の桁数と被乗数の桁数を足したものになる

$$
\begin{array}{rrrrrrrr}
 & & & & 0 & 0 & 1 & 1 \\
\times & & & & 0 & 1 & 1 & 0 \\
\hline
 & & & & 0 & 0 & 0 & 0 \\
 & & & 0 & 0 & 1 & 1 & \\
 & & 0 & 0 & 1 & 1 & & \\
 & 0 & 0 & 0 & 0 & & & \\
\hline
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0
\end{array}
\qquad
\begin{array}{l}
3_{10} \\
6_{10} \\
\\
\\
\\
\\
\\
18_{10}
\end{array}
$$

2進数の乗算

- 正数×負数
  - 乗数と被乗数を符号拡張して筆算
  - 上位ビットを切り捨てる

```
        00000011    3₁₀
×       11111010   -6₁₀
-------------------
       00000011
     00000011
    00000011
   00000011
  00000011
 00000011
-------------------
切り捨て 11101110   -18₁₀
```

2進数の乗算

- 負数×負数
  - 正数×負数と同様
  - 正数×正数に直して計算しても良い

```
        11111101    -3₁₀
×       11111010    -6₁₀
------------------
       11111101
     11111101
    11111101
   11111101
  11111101
 11111101
------------------
切り捨て 00010010    18₁₀
```

2進数の除算

- 正数／正数
  - 例) $13_{10}/5_{10}$ = 2 余り 3 = $1101_2/0101_2$

```
          10
     ________
101 ) 1101
      101
     ________
          11
```

- 10進数と同じように筆算できる

2進数の除算

- 符号付き除算
  - 被除数＝商×除数＋剰余
  - 例)
    - (+7)/(+2) = (+3)余り(+1)
    - (-7)/(-2) = (+3)余り(-1)
    - (-7)/(+2) = (-3)余り(-1)
    - (+7)/(-2) = (-3)余り(+1)

✖(+7)/(-2) = (-4)余り(-1)

被除数の符号と剰余の符号は
同じでなければならない

2進数の小数

- 10進数と同様に小数点を用いる
- 小数点第1位が2^{-1}に相当

- 例) $11.11_2 = 1*2^1+1*2^0+1*2^{-1}+1*2^{-2}$
  $= 2+1+0.5+0.25$
  $= 3.75_{10}$
  $0.0011_2 = 1*2^{-3}+1*2^{-4}$
  $= 0.125+0.0625$
  $= 0.1875_{10}$

2進数の小数の問題点

- 誤差の発生
  - 例) $0.1_{10} = 0.000110011001100..._2$
    $= 0.0625 + 0.03125 + ...$

10進数の有限小数を有限桁で
表すことができない！

算術左シフトについて

左に1ビットシフト

10101010 = -86_{10}

11010100 = -44_{10}

オーバーフロー

- 負数に対する1ビットの算術左シフトの場合、符号を除いたビットのうち一番左のビットが0であればオーバーフロー
- 1であればオーバーフローしない
→ 実は、論理左シフトでも、シフト結果は同じ
(オーバーフローした場合の結果は異なる)

負 11xxxxxx　→　1xxxxxx0

正 00xxxxxx　→　0xxxxxx0

算術シフトでも
論理シフトでも同じ
→ 算術左シフトという言葉を
使わない考え方もある

参考文献

- コンピュータの構成と設計 上 第5版

  David A.Patterson, John L. Hennessy 著、成田光彰 訳、日経BP社