第3章
コンピュータにおける算術演算(1)

大阪大学 大学院 情報科学研究科
今井 正治

arch-2014@vlsilab.ics.es.osaka-u.ac.jp

講義内容

- 加算と減算
- 乗算
- 除算
- シフト演算とローテート演算
- 浮動小数点数の表現方法
- 浮動小数点数の加減算
- 浮動小数点数の乗算
- 並列処理とコンピュータの算術演算: 結合則
- 実例: x86における浮動小数点演算
- 誤信と落とし穴

加算

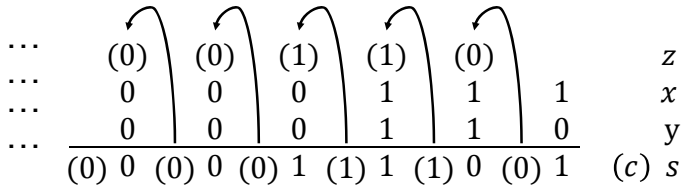
- 2進数の各桁を最下位から最上位に順に加える
- 全加算器(full adder)を用いる
  - 入力:
    - x, y, z
      z は下位からの桁上げ
  - 出力:
    - c 桁上げ(carry)
    - s 和(sum)

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

加算

- 7_{10} に 6_{10} を加える

$$
\begin{array}{ll}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111 = 7_{10} \\
+ & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110 = 6_{10} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101 = 13_{10}
\end{array}
$$



減算

- 方法1: 直接減算する減算器を用いる
  - 入力:
    - x, y, z
      z は下位からの借り
  - 出力: $x - y - z$
    - b 借り(borrow)
    - d 差(difference)
- 方法2: 2の補数を加える

$(b, d) = x - y - z$

x	y	z	b	d
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

直接減算

- 7_{10} から 6_{10} を引く

$$
\begin{array}{ll}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111 = 7_{10} \\
- & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110 = 6_{10} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001 = 1_{10}
\end{array}
$$

$$
\begin{array}{llllllllll}
\cdots & (0) & (0) & (0) & (0) & (0) & & & & z \\
\cdots & 0 & 0 & 0 & 1 & 1 & 1 & & & x \\
\cdots & 0 & 0 & 0 & 1 & 1 & 0 & & & y \\
\hline
 & (0) & 0 & (0) & 0 & (0) & 0 & (0) & 0 & (0)\ 0\ (0)\ 1 \quad (b)\ d
\end{array}
$$

2の補数を用いた減算

- 7_{10} に 6_{10} の2の補数を加える

$$
\begin{array}{ll}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111 = 7_{10} \\
+ & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010 = 6_{10} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001 = 1_{10}
\end{array}
$$

$$
\begin{array}{llllllllll}
\cdots & (1) & (1) & (1) & (1) & (0) & & & & z \\
\cdots & 0 & 0 & 0 & 1 & 1 & 1 & & & x \\
\cdots & 1 & 1 & 1 & 0 & 1 & 0 & & & y \\
\hline
 & (1) & 0 & (1) & 0 & (1) & 0 & (1) & 0 & (1)\ 0\ (0)\ 1 \quad (c)\ s
\end{array}
$$

オーバーフロー(overflow)の発生

- オーバーフローが発生する可能性がある組合せ
  - 正数と正数の加算　　結果が負
  - 負数と負数の加算　　結果が正
  - 正数から負数の減算　　結果が負
  - 負数から正数の減算　　結果が正
- オーバーフローが発生する可能性がない組合せ
  - 正数と負数の加算
  - 正数から負数の減算

オーバーフロー（overflow）の判定方法

操作	オペランドA	オペランドB	結果
A+B	≧ 0	≧ 0	＜ 0
A+B	＜ 0	＜ 0	≧ 0
A-B	≧ 0	＜ 0	＜ 0
A-B	＜ 0	≧ 0	≧ 0

オーバーフローの処理方法

- オーバーフローが起きた場合の処理方法は，プロセッサおよびOSに依存する
- MIPSの場合の処理方法
  - 加算（add），即値加算（addi），減算（sub）命令の実行によってオーバーフローが起こると例外（exception）が発生する
  - 符号なし加算（addu），符号なし即値加算（addiu），符号なし減算（subu）命令の実行でオーバーフローが起こっても例外は発生しない

マルチメディア用の算術演算

- 画像の表現
  - RGB（red, green, blue）の各色に対して8ビット
- 音声の表現
  - 8ビット以上が必要，16ビットあれば十分
- SIMD（single instruction stream, multiple data stream）命令
  - 複数のデータに対して同じ演算を並列に実行する
- 飽和演算
  - オーバーフローが起きた場合に，結果を正の最大値または負の最小値に設定

マルチメディア命令の例

命令カテゴリ	オペランド
符号なし加算/減算	8つの8ビット または 4つの16 ビット
飽和型の加算/減算	8つの8ビット または 4つの16 ビット
最大/最小	8つの8ビット または 4つの16 ビット
平均	8つの8ビット または 4つの16 ビット
右/左シフト	8つの8ビット または 4つの16 ビット

講義内容

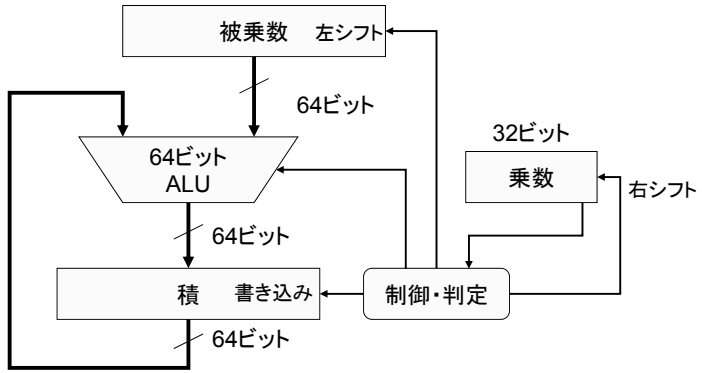
- 加算と減算
- 乗算
- 除算
- シフト演算とローテート演算
- 浮動小数点数の表現方法
- 浮動小数点数の加減算
- 浮動小数点数の乗算
- 並列処理とコンピュータの算術演算：結合則
- 実例：x86における浮動小数点演算
- 誤信と落とし穴

乗算

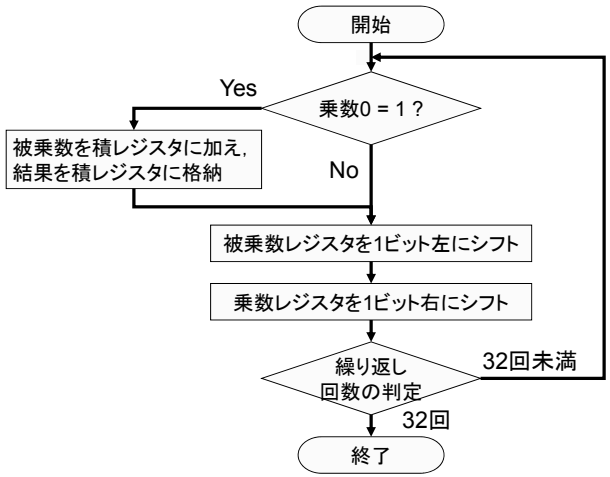
- 被乗数（multiplicand）と乗数（multiplier）の積（product）を計算
- 被乗数と乗数1桁分の積（部分積）を適切な桁数分シフトして加算を繰り返す
- n 桁と m 桁の乗算結果は $n+m$ 桁になる

$$
\begin{array}{r}
1000_{10} \\
\times\ 1001_{10} \\
\hline
1000 \\
0000 \\
0000 \\
1000 \\
\hline
1001000_{10}
\end{array}
$$

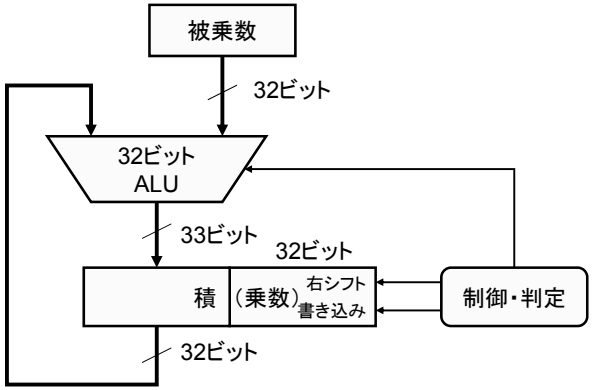
逐次型乗算器 バージョン1



乗算アルゴリズム



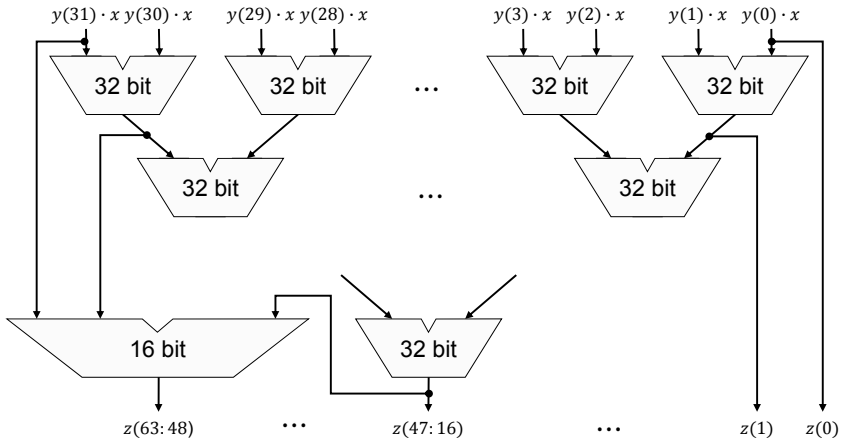
逐次型乗算器 バージョン2



符号付きの乗算

- 被乗数および乗数の符号を調べる
- 被乗数および乗数を正の数に変換する
- 正の数に変換された数値(31ビット)の乗算行う
- 変換前の被乗数および乗数の符号が異なる場合には積を負の数に変換する

高速な乗算回路



講義内容

- 加算と減算
- 乗算
- 除算
- シフト演算とローテート演算
- 浮動小数点数の表現方法
- 浮動小数点数の加減算
- 浮動小数点数の乗算
- 並列処理とコンピュータの算術演算: 結合則
- 実例: x86における浮動小数点演算
- 誤信と落とし穴

除算

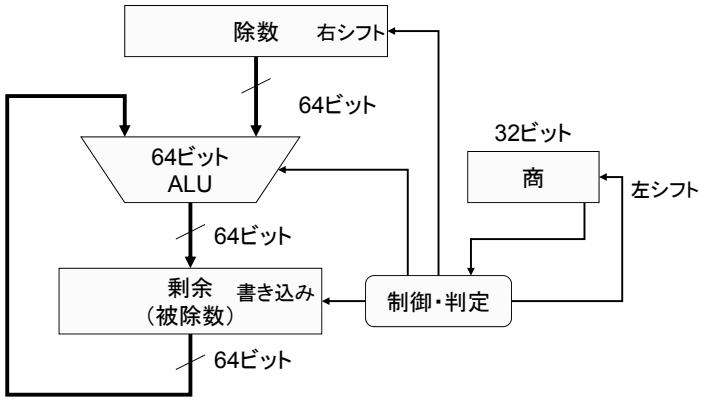
- 被除数（dividend）
- 除数（divider）
- 商（quotient）
- 剰余（remainder）

被除数=商×除数＋剰余

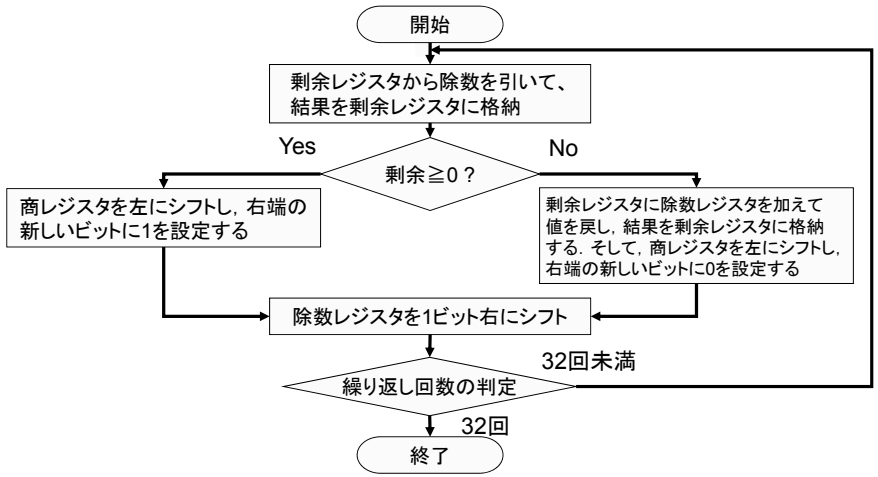
除算の例

$$
\begin{array}{rr}
 & 1001_{10} \quad \text{商} \\
\text{除数}\ 1000_{10}\ ) & 1001010_{10} \quad \text{被除数} \\
 & -1000 \\
 & \hline
 & 10 \\
 & 101 \\
 & 1010 \\
 & -1000 \\
 & \hline
 & 10_{10} \quad \text{剰余}
\end{array}
$$

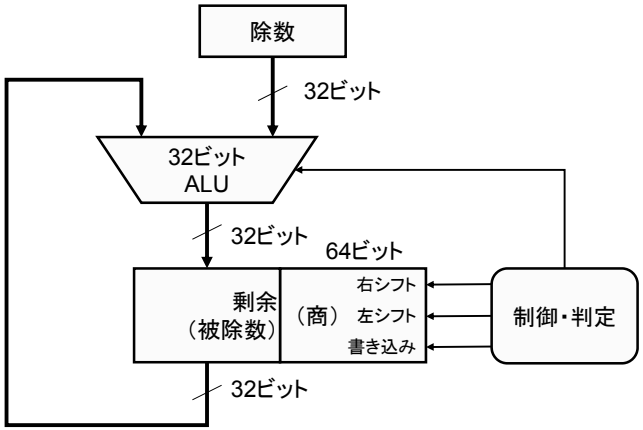
逐次型除算器 バージョン1



乗算アルゴリズム



逐次型除算器 バージョン2



符号付除算

- 剰余の符号は，被除数の符号と一致させる
  - $+7 \div +2 = +3 \ \ldots + 1$
  - $+7 \div -2 = -3 \ \ldots + 1$
  - $-7 \div +2 = -3 \ \ldots - 1$
  - $-7 \div -2 = +3 \ \ldots - 1$

除算の高速化

- 引き放し法（division）
- SRT除算法（SRT division）
- 桁上げ保存加算器（carry save adder）の利用
- 配列型除算器の利用
- 高基数減算シフト型除算法
  などなど

MIPSにおける除算器

- 乗算と除算で同じハードウェアを利用可能
  - 左右にシフト可能な64ビットレジスタ
  - 32ビットALU
- 符号付除算命令
  - div
- 符号なし除算命令
  - divu
- 商および剰余の取り出し
  - mflo（move from Lo），mfhi（move from Hi）

乗算および除算関連の命令

命令	例	意味
multiply	`mult $s2, $s3`	Hi.Lo = $s2*$s3
multiply unsigned	`multu $s2, $s3`	Hi.Lo = $s2*$s3
divide	`div $s2, $s3`	Lo = $s2÷$s3 Hi = $s2 mod $s3
divide unsigned	`divu $s2, $s3`	Lo = $s2÷$s3 Hi = $s2 mod $s3
move from Hi	`mfhi $s1`	$s1 = Hi
move from Lo	`mflo $s1`	$s1 = Lo

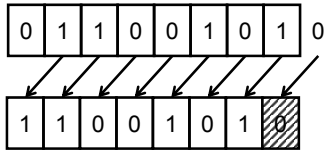
講義内容

- 加算と減算
- 乗算
- 除算
- シフト演算とローテート演算
- 浮動小数点数の表現方法
- 浮動小数点数の加減算
- 浮動小数点数の乗算
- 並列処理とコンピュータの算術演算：結合則
- 実例：x86における浮動小数点演算
- 誤信と落とし穴

論理シフト演算

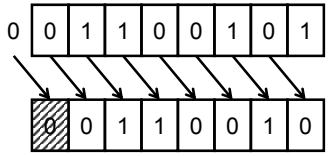
- 論理左シフト（Logical Left Shift）

$$(x_{n-1}, x_{n-2}, x_{n-3}, \Lambda\ , x_1, x_0)$$
$$\Rightarrow (x_{n-2}, x_{n-3}, x_{n-4}, \Lambda\ , x_1, x_0, 0)$$



- 論理右シフト（Logical Right Shift）

$$(x_{n-1}, x_{n-2}, x_{n-3}, \Lambda\ , x_1, x_0)$$
$$\Rightarrow (0, x_{n-1}, x_{n-2}, \Lambda\ , x_2, x_1)$$

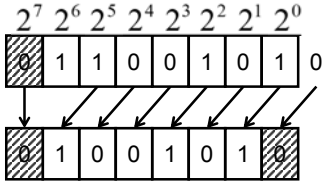


算術シフト演算

- 算術左シフト（Arithmetic Left Shift）

$$(x_{n-1}, x_{n-2}, x_{n-3}, \Lambda\ , x_1, x_0)$$
$$\Rightarrow (x_{n-1}, x_{n-3}, x_{n-4}, \Lambda\ , x_0, 0)$$

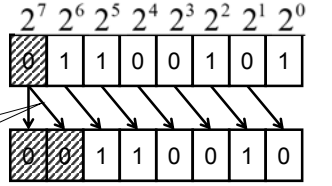
符号の保存



- 算術右シフト（Arithmetic Right Shift）

$$(x_{n-1}, x_{n-2}, x_{n-3}, \Lambda\ , x_1, x_0)$$
$$\Rightarrow (x_{n-1}, x_{n-1}, x_{n-2}, \Lambda\ , x_2, x_1)$$

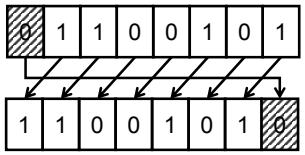
符号拡張（sign extension）



ローテート演算

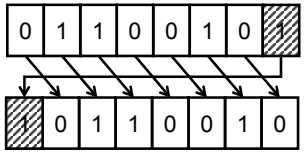
□ 左ローテート(Left Rotate)

$$(x_{n-1}, x_{n-2}, x_{n-3}, \Lambda\ , x_1, x_0)$$
$$\Rightarrow (x_{n-2}, x_{n-3}, x_{n-4}, \Lambda\ , x_0, x_{n-1})$$



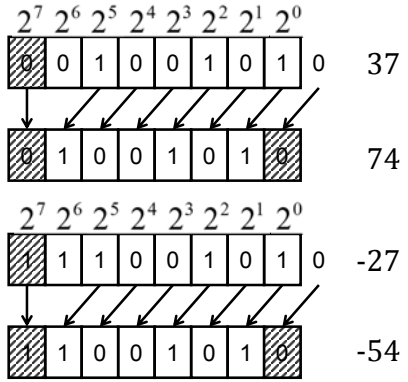
□ 右ローテート(Right Rotate)

$$(x_{n-1}, x_{n-2}, x_{n-3}, \Lambda\ , x_1, x_0)$$
$$\Rightarrow (x_0, x_{n-1}, x_{n-2}, \Lambda\ , x_2, x_1)$$

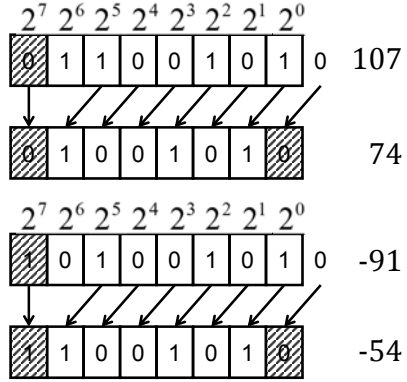


算術左シフトでのオーバーフロー(1)

□ オーバーフローなし

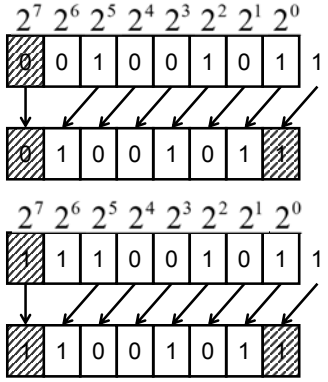


□ オーバーフローあり



算術左シフトでのオーバーフロー(2)

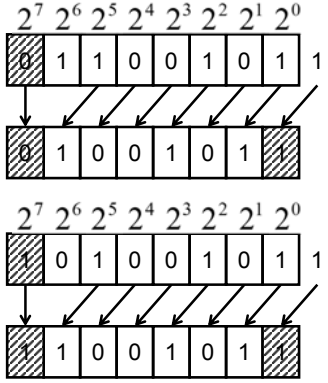
□ オーバーフローあり



$\neq 37 \times 2$

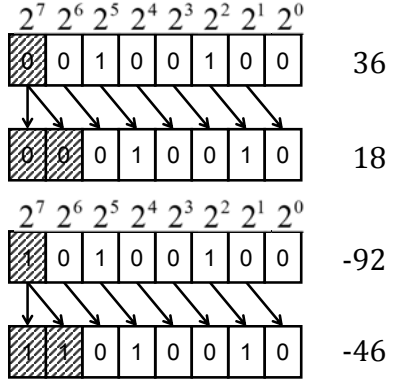
$\neq -27 \times 2$

□ オーバーフローあり

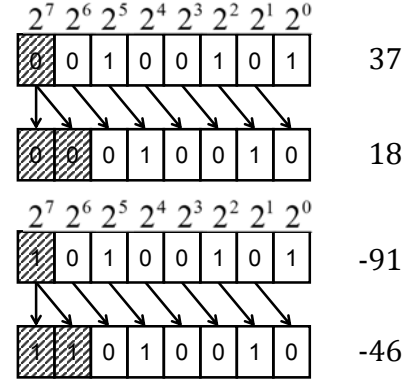


算術右シフトでのアンダーフロー(underflow)

□ アンダーフローなし



□ アンダーフローあり



算術シフトで正しい結果が得られる条件

- 左シフト(オーバーフロー)
  x_R : 右側から加えるビットの値

$$x_{n-1} = x_{n-2}$$
$$x_R = 0$$

- 右シフト(アンダーフロー)

$$x_0 = 0$$

講義内容

- 加算と減算
- 乗算
- 除算
- シフト演算とローテート演算
- 浮動小数点数の表現方法
- 浮動小数点数の加減算
- 浮動小数点数の乗算
- 並列処理とコンピュータの算術演算: 結合則
- 実例: x86における浮動小数点演算
- 誤信と落とし穴

浮動小数点数

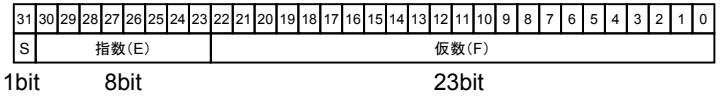
- 実数形式のデータの例
  - 3.1415926535_{10} (π)
  - 2.718281828_{10} (e)
  - 0.000000001_{10} $= 1.0_{10} \times 10^{-9}$
  - $3{,}155{,}760{,}000_{10}$ $= 3.15576_{10} \times 10^{9}$

科学記法
(scientific notation)

- 正規化数(normalized number)
  - 科学記法で書いた数値で、先頭に0が来ないもの
- 浮動小数点(floating point)形式の2進数
  - $1.xxxxxxxxxx_2 \times 2^{yyyy}$

IEEE Std 754 での 単精度浮動小数点(floating point)形式

- 浮動小数点形式の構成要素
  - 符号(sign)
    - 0: 非負, 1: 負
  - 指数(exponent)
    - ゲタ履き表現(biased representation)
  - 仮数(significand, mantissa)
    - 小数点以下のみを表現
- IEEE Std-754 での浮動小数点数の表現方法



  - $(-1)^S \times (1 + f_1 \times 2^{-1} + f_2 \times 2^{-2} + \cdots) \times 2^{E-127}$

オーバーフローとアンダーフロー

- オーバーフロー（overflow）
  - 指数が指数フィールドに収まり切れないほど大きくなった場合
- アンダーフロー（underflow）
  - 小数の値が小さくなりすぎて，負の指数が大きすぎて指数フィールドに収まり切れないほど大きくなった場合
- オーバーフローおよびアンダーフローの発生を減らす対策
  - 倍精度（double precision）浮動小数点形式を使用

IEEE Std 754 での倍精度浮動小数点形式

- 指数フィールド：11ビット
- 仮数フィールド：52ビット
- 数値（絶対値）の表現範囲
  - $2.0_{10} \times 10^{-308} \sim 2.0_{10} \times 10^{308}$
- フィールドの配置

浮動小数点数のエンコード規則

単精度		倍精度		内容
指数	仮数	指数	仮数	
0	0	0	0	0
0	≠0	0	≠0	± 非正規化数
1～254	任意	1～2046	任意	± 浮動小数点数
255	0	2047	0	± 無限大（∞）
255	≠0	2047	≠0	NaN（非数）

無効な演算操作の結果の表現

- 0割り算の結果
  - 例外を発生させず，無限大（±∞）として扱う
  - 指数部は最大値（255または2047），仮数部は0
- 無効な演算操作の結果を表現
  - NaN（Not a Number）
  - 指数部は最大値（255または2047），仮数部は≠0
  - 状況のチェックと判定を先送り可能
  - 例: $0 \div 0, \infty - \infty$

指数部のゲタ履き表現

- 浮動小数点形式を符号，指数部，仮数部の順に配置することで，整数の比較命令を用いて浮動小数点数の比較が行える
- 指数部をゲタ履き表現にすることで比較が容易になる

ゲタ履き表現	10進数	2の補数表現
11111111	127	01111111
11111110	126	01111110
…	…	…
10000010	2	00000010
10000001	1	00000001
10000000	0	00000000
01111111	-1	11111111
01111110	-2	11111110
…	…	…
00000001	-127	10000001
00000000	-128	10000000

浮動小数点数の表現例（単精度）

- $-0.75_{10} = -3/4_{10} = -3/2^2{}_{10} = -11_2/2^2{}_{10}$
  $= -0.11_2 = -0.11_2 \times 2^0$
  $= -1.1_2 \times 2^{-1}$
  $= (-1)^1 \times (1+0.1)_2 \times 2^{126-127}$

- 単精度表現

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1bit 8bit 23bit

浮動小数点数の表現例（倍精度）

- $-0.75_{10} = -3/4_{10} = -3/2^2{}_{10} = -11_2/2^2{}_{10}$
  $= -0.11_2 = -0.11_2 \times 2^0$
  $= -1.1_2 \times 2^{-1}$
  $= (-1)^1 \times (1+0.1)_2 \times 2^{1022-1023}$

- 単精度表現

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1bit 11bit 20bit

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

32bit

浮動小数点形式の2進数から10進数への変換

- 例題

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

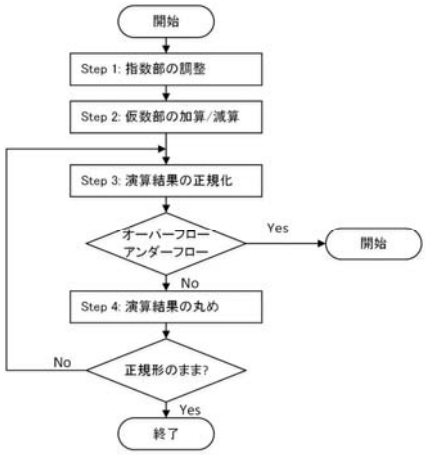
- 変手順

$(-1)^S \times (1+F) \times 2^{(E-Bias)}$
$= (-1)^1 \times (1+0.25) \times 2^{(129-127)}$
$= -1 \times 1.25 \times 2^2 = -1.25 \times 4 = -5.0$

講義内容

- 加算と減算
- 乗算
- 除算
- シフト演算とローテート演算
- 浮動小数点数の表現方法
- 浮動小数点数の加減算
- 浮動小数点数の乗算
- 並列処理とコンピュータの算術演算：結合則
- 実例：x86における浮動小数点演算
- 誤信と落とし穴

浮動小数点数の加減算の手順



浮動小数点数の加算の例

- 入力：0.5_{10} および -0.4375_{10}
  - $0.5_{10} = 1/2^1{}_{10} = 0.1_2 = 1.000_2 \times 2^{-1}$
  - $-0.4375_{10} = -7/2^4{}_{10} = -0.0111_2 \times 2^0 = -1.110_2 \times 2^{-2}$

Step 1: 指数部の調整

$$-1.110_2 \times 2^{-2} = -0.111_2 \times 2^{-1}$$

Step 2: 仮数部の加算

$$1.000_2 \times 2^{-1} - 0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$$

Step 3: 演算結果の正規化

$$0.001_2 \times 2^{-1} = 1.000_2 \times 2^{-4}$$

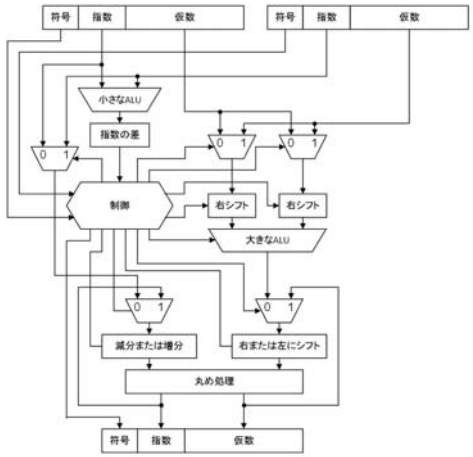
オーバーフローまたはアンダーフローは発生していない．

$1.000_2 \times 2^{-4} = 0.0001_2 = 1/2^4{}_{10} = 1/16_{10} = 0.0625_{10}$

Step 4: 演算結果の丸め（round）

丸める必要なし．結果は正規形．終了．

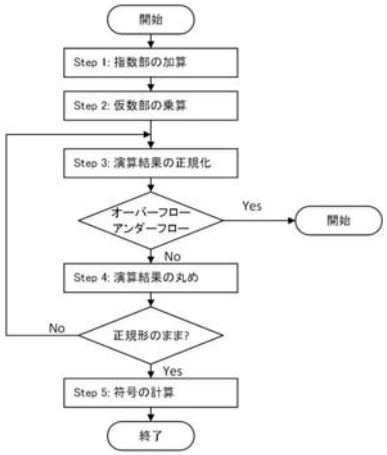
浮動小数点数加減算器の構造



講義内容

- 加算と減算
- 乗算
- 除算
- シフト演算とローテート演算
- 浮動小数点数の表現方法
- 浮動小数点数の加減算
- 浮動小数点数の乗算
- 並列処理とコンピュータの算術演算：結合則
- 実例：x86における浮動小数点演算
- 誤信と落とし穴

浮動小数点数の乗算の手順



浮動小数点数の乗算の例(1)

- 入力: 0.5_{10} および -0.4375_{10}
  - $0.5_{10} = 1/2^1{}_{10} = 0.1_2 = 1.000_2 \times 2^{-1}$
  - $-0.4375_{10} = -7/2^4{}_{10} = -0.0111_2 \times 2^0 = -1.110_2 \times 2^{-2}$

Step 1: 指数部の加算

$$-1 + (-2) = -3$$

げた履き表現での加算

$$(-1 + 127) + (-2 + 127) - 127$$
$$= (-1 - 2) + (127 + 127 - 127)$$
$$= -3 + 127 = 124$$

Step 2: 仮数部の乗算

$$1.000_2 \times 1.110_2 = 1.110000_2$$

乗算結果(有効数字4桁)

$$1.110000_2 \times 2^{-3} \rightarrow 1.110_2 \times 2^{-3}$$

浮動小数点数の乗算の例(2)

Step 3: 演算結果の正規化

$1.110_2 \times 2^{-3}$ は正規形

オーバーフローまたはアンダーフローは発生していない.

Step 4: 演算結果の丸め

丸める必要なし

Step 5: 符号の計算

入力の符号が異なっているので、結果の符合はマイナス(ー)

乗算の結果は

$$-1.110_2 \times 2^{-3}$$

MIPSの浮動小数点命令(1)

- 単精度加算(add.s), 倍精度加算(add.d)
- 単精度減算(sub.s), 倍精度減算(sub.d)
- 単精度乗算(mul.s), 倍精度乗算(mul.d)
- 単精度除算(div.s), 倍精度除算(div.d)
- 浮動小数点形式の単精度比較(c.x.s),
  浮動小数点形式の倍精度比較(c.x.d)
  - x ∈ { eq, neq, lt. le, gt, ge }
- 浮動小数点形式の条件が真のときに分岐(bc1t)および偽のときに分岐(bc1f)

MIPSの浮動小数点命令(2)

- 浮動小数点用レジスタ(単精度, 倍精度で共用)
  - `$f0, $f1, $f2, … , $f31`
  - 倍精度用のレジスタは, 偶数番目の単精度用レジスタとその次の(奇数番目)単精度用レジスタのペア
- 浮動小数点用レジスタへのロード命令およびストア命令
  - `lwc1`
  - `swc1`

演算の正確性(1)

- 例題
  - $2.56_{10} \times 10^0 + 2.34_{10} \times 10^2$
  - 有効数字3桁
- ガード桁と丸め桁がある場合

$$\begin{array}{r} 0.0256_{10} \times 10^2 \\ +\ 2.3400_{10} \times 10^2 \\ \hline 2.3656_{10} \times 10^2 \end{array} \quad \rightarrow \quad 2.37_{10} \times 10^2$$

ガード桁　丸め桁

演算の正確性(2)

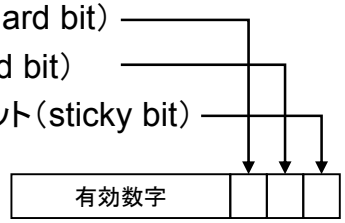
- ガード桁と丸め桁がない場合

$$\begin{array}{r} 0.02_{10} \times 10^2 \\ +\ 2.34_{10} \times 10^2 \\ \hline 2.36_{10} \times 10^2 \end{array} \quad \neq \quad 2.37_{10} \times 10^2$$

- IEEE Std 754 での精度保障
  - 0.5 ulp 以内
  - ulp = unit in the last place (最下位ビット単位)
  - 加減算の場合, 丸め桁が1ビット必要
  - 乗算の場合, ガード桁および丸め桁の2ビットが必要

IEEE Std 754での丸め方

- 切り上げ(+∞方向への丸め)
- 切り下げ(−∞方向への丸め)
- 切り捨て(0方向への丸め)
- 最も近い偶数への丸め(0捨1入)
  - ガード・ビット(guard bit)
  - 丸めビット(round bit)
  - スティッキー・ビット(sticky bit)



講義内容

- 加算と減算
- 乗算
- 除算
- シフト演算とローテート演算
- 浮動小数点数の表現方法
- 浮動小数点数の加減算
- 浮動小数点数の乗算
- 並列処理とコンピュータの算術演算: 結合則
- 実例: x86における浮動小数点演算
- 誤信と落とし穴

浮動小数点加算での結合則(1)

- 整数(固定小数点数)の加算では結合則が成立するが, 浮動小数点数の加算では結合則が成立しない. 演算結果は, 演算順序に依存する.
- 例: $x = -1.5_{10} \times 10^{38}$, $y = 1.5_{10} \times 10^{38}$, $z = 1.0$ とする
- $x + (y + Z)$
  $= -1.5_{10} \times 10^{38} + (1.5_{10} \times 10^{38} + 1.0)$
  $= -1.5_{10} \times 10^{38} + 1.5_{10} \times 10^{38}$
  $= 0.0$

浮動小数点加算での結合則(1)

- $(x + y) + z$
  $= (-1.5_{10} \times 10^{38} + 1.5_{10} \times 10^{38}) + 1.0$
  $= 0.0 + 1.0$
  $= 1.0$
- よって, $x + (y + z) \neq (x + y) + z$
- 並列計算機で浮動小数点数の計算を行う場合, 結果がわずかに異なる場合がある. 浮動小数点数の演算結果を用いて条件分岐を行う場合には, プログラムの正しさの検証が困難になる.

講義内容

- 加算と減算
- 乗算
- 除算
- シフト演算とローテート演算
- 浮動小数点数の表現方法
- 浮動小数点数の加減算
- 浮動小数点数の乗算
- 並列処理とコンピュータの算術演算：結合則
- 実例：x86における浮動小数点演算
- 誤信と落とし穴

x86の浮動小数点アーキテクチャ(1)

- Intel 8087（1980年発表）：8086に約60の浮動小数点命令を追加して拡張
- 浮動小数点演算用スタック・アーキテクチャ
  - ロード命令：push
  - 演算命令：スタック上の2つのオペランドを用いて実行し、結果をスタックに push
  - ストア命令：pop
- アドレッシング・モードの拡張：レジスタ・メモリ方式の採用

x86の浮動小数点アーキテクチャ(2)

- 拡張倍精度（double extended precision）
  - 80ビットフォーマット：
    符号 1ビット，指数部 15ビット，仮数部 64ビット
- 浮動小数点演算の分類
  - データ転送命令：ロード，定数ロード，ストア など
  - 算術演算命令：加算，減算，乗算，除算，平方根，絶対値 など
  - 比較：分岐を行うために，結果を整数プロセッサに転送
  - 超越関数命令：三角関数，対数，指数 など

x86の浮動小数点命令

データ転送	算術演算	比較	超越関数
F{I}LD mem/ST(i)	F{i}ADD{P} mem/ST(i)	F{I}COM{P}{P}	FPATAN
F{I}ST{P} mem/ST(i)	F{i}SUB{R}{P} mem/ST(i)	F{I}UCOM{P}{P}	F2XM1
FLDPI	F{I}MUL{P} mem/ST(i)	PSTSW AX/mem	FCOS
FLD1	F{i}DIV{R}{P} mem/ST(i)		FPTAN
FLDZ	FSQRT		FPREM
	FABS		FSIN
	FRNDINT		FYL2X

講義内容

- 加算と減算
- 乗算
- 除算
- シフト演算とローテート演算
- 浮動小数点数の表現方法
- 浮動小数点数の加減算
- 浮動小数点数の乗算
- 並列処理とコンピュータの算術演算：結合則
- 実例：x86における浮動小数点演算
- 誤信と落とし穴

誤信と落とし穴

- 誤信：左シフト命令が2のべき乗を整数に乗ずる働きがあるのと同様に，右シフト命令には2のべき乗で整数を割る働きがある．
- 落とし穴：MIPSの符号なし即値加算命令（addiu）は16ビットの即値フィールドを符号拡張する．
- 誤信：浮動小数点形式の演算精度を気にするのは理論数学者だけである．