第4回 演算回路

前回:
ノイマン型コンピュータの構成要素
メモリ, 入出力, CPU

今回:
CPUについてさらに詳しく説明
各種の演算と, それを実現する電子回路

キーワード:
ブール代数, 真理値表, 論理回路,
ALU, 加算器, シフト, 乗除

3.5 ALU
CPUの中の演算回路
算術論理演算ユニット (ALU: Arithmetic and Logic Unit)

3.5.1 論理演算
この辺の話は「離散数学」「論理設計学」とも関連

3.5.1.1 ブール(Boole)代数
値:
  真1 偽0
論理変数:
  A, B, C などと表記
論理演算:
  + : 論理和 or
  ・: 論理積 and
  ~ : 否定 not

真理値表 truth table
入力のそれぞれの値の組み合わせに対する演算結果を表にしたもの

論理和 (OR)
A B A+B
0 0 0
0 1 1
1 0 1
1 1 1

論理積 (AND)
A B A・B
0 0 0
0 1 0
1 0 0
1 1 1

否定 (NOT)
A ~A
0 1
1 0

(恒等則) A+0=A, A・1=A
(0と1の代数則) A+1=1, A・0=0
(逆元則) A+~A=1, A・~A=0
(交換則) A+B=B+A, A・B=B・A
(結合則) A+(B+C)=(A+B)+C, A・(B・C)=(A・B)・C
(分配則) A・(B+C)=(A・B)+(A・C)
         A+(B・C)=(A+B)・(A+C)
ドモルガン(De Morgan)の定理
         ~(A+B)=~A・~B
         ~(A・B)=~A+~B

排他的論理和 (XOR, exclusive or)
  \oplus…○に＋ (Latex風の表記)
A B A \oplus B
0 0   0

0 1　1
1 0　1
1 1　0

上記の論理演算は「1ビット同士」の演算
実際の計算機では, 32ビットとかまとめて論理演算
「ビットごと(bitwise)の論理和」などと呼ぶ

3.5.1.2 論理演算の使用例
- 特定のビットの切り出し
ある32ビットの値Xの, 下から3ビット目だけを切り出したい
X AND 0x00000004

- 特定のビットを0にする (マスクする)
ある32ビットの値Yの, 下から4ビットをすべて0にした値
Y AND 0xfffffff0

- 2のべき乗へのまるめ
ある32ビットのアドレスZを, 語境界(4の倍数)に切り捨ててまるめる
(つまり下位2ビットを0にする - 上と同じ)
Z AND 0xfffffffc

ある32ビットのアドレスZを, 語境界(4の倍数)に切り上げてまるめる
(Z + 3) AND 0xfffffffc

Z	(Z AND 0xff..fc)	((Z+3) AND 0xff..fc)
0	0	0
1	0	4
2	0	4
3	0	4
4	4	4

- あるビットだけを反転したい
  反転したいビット位置が1になっている定数と XOR を行う

3.5.1.3 論理回路
論理演算を行う電子回路.
図5.1-5.3
1/0の値は, 電圧の高/低, 電流の有/無で表現
スイッチやトランジスタで実現
図5.4
遅れがある
　　*入力が定まってから出力が定まるまでの時間

3.5.2 算術演算

3.5.2.1 加算
1ビットの加算 A+B (ORと記号は同じだが)
加算結果は2ビットになる
  C: carry bit 桁上げ
  S: sum bit 和

真理値表
A B C S
0 0 0 0
0 1 0 1
1 0 0 1
1 1 1 0

見てわかるとおり
  C : A AND B
  S : A XOR B

半加算器 Half Adder (図5.5)
  2入力(A,B), 2出力(C,S)
  下の桁からの桁上げを考慮してないので「半」

全加算器 Full Adder

3入力 (iはi桁めをあらわす)
  A_i
  B_i
  C_{i-1} (下の桁からの桁上げ)
2出力
  C_i (上の桁への桁上げ)
  S_i

A_i	B_i	C_{i-1}	C_i	S_i
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

半加算器をふたつつなげると完成 (図5.6)

32ビットの加算器
1ビット全加算器を32個連結する (図5.7)

計算機での加算の重要性
  単に式の計算だけでなく
  配列、ローカル変数、構造体要素へのアクセス
  そもそも機械語命令をアドレス順に実行するために必要

加算の桁あふれ(オーバーフロー)の検出
  授業第2回
  加算結果が数の表現範囲を逸脱

○ 符号なし表現の加算の場合
  2.1.2.1で説明
  最上位からの桁上がり(キャリー)出力があれば
  オーバーフロー

  キャリー
    多倍長加算のときにも使う
      32ビットを複数使った計算
      上位の次の32ビットへの桁あがり

○ 2の補数表現の場合
  2.2.5で説明
  C_31 と C_30 が違っているとオーバーフロー
  つまり, C_31 XOR C_30

桁上げの伝搬 (carry propagation)
  上記説明の方式(図5.7)では遅くなる
    下の桁の計算が済むまで桁上げが決定しない
  高速化の手法あり

3.5.2.2 減算
負の数にして加算すればよい
2の補数表現では負の数は
  ビットごとにNOTしたもの + 1
図5.8
補数用制御信号線 - 減算を行うときに真(1)
  減数 B は 1 と XOR をとるので, NOT になる
  最下位に入ってくるキャリー C_{-1} に 1 が入って +1

3.5.2.3 シフト演算 図5.13
  値を指定したビット数だけ左/右に移動する
  例: 0x0020 -> 0x0200 (4bit 左シフト)
単純な回路だと1ビットだけしかずらせない
一度に多ビットずらせる回路もあり

右シフトの場合

論理シフトと算術シフトの二種がある
MSBに何が入るか
論理右シフト - 0が入る (logical shift)
算術右シフト - MSBがそのまま残る (arithmetic shift)
2の補数表現での符号を保つ
2のべきによる除算に対応
(cf. 授業第2回 2.2.4符号拡張)

左シフトの場合
論理左シフト, 算術左シフトの区別なし
LSBには0が入る
MSBは右隣がそのまま移動してくる

3.5.2.4 乗算
符号なしで説明
筆算による2進整数の乗算
0011(被乗数) × 0101(乗数)

乗数をLSBから見ていき, 1 の場合に被乗数を加算してシフト

回路 図5.10

3.5.2.5 除算
筆算による2進整数の除算
1111(被除数) / 100(除数) (15÷4 = 3 余り 3)
除数を引けるかどうかで商に1/0をたてていく
(引いてみて負になったら戻す) - restoring method

2018-05-02

回路 図5.12

剰余の符号 (符号つきの除算の場合)
除算の定義
q(商)*d2(除数)+r(余り) = d1(被除数)
余りの値の範囲に自由度あり
(-5)/(+3)
-1 余り -2 (方式A 余りは被除数と同符号)
-2 余り 1 (方式B 余りは除数と同符号)
方式A が一般的
商をゼロの方向にまるめる