コンピュータ・サイエンス2

第2回
論理回路

人間科学科コミュニケーション専攻
白銀 純子

第2回の内容

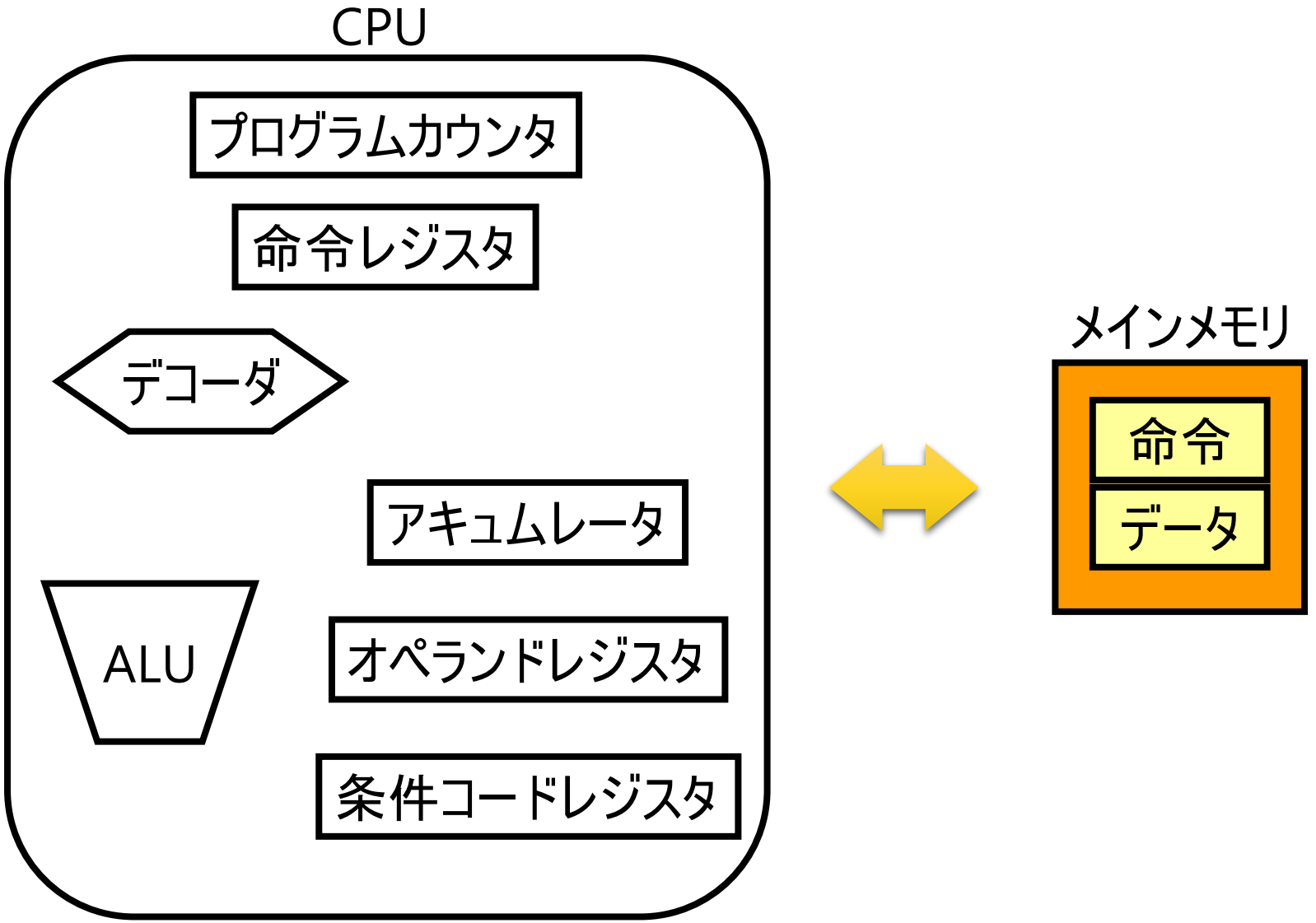
- 論理回路

中央処理装置(CPU)

CPU(p. 39)

- プログラムの命令をメインメモリから取り出して解釈し、実行するための装置
  - プログラム: コンピュータへの命令の集合
    - 「プログラミング言語」という人間が理解できる言葉で書かれた命令の集合
    - プログラミング言語の命令を、機械語(0と1の2進数)に翻訳した命令の集合
  - 機械語のプログラムをメインメモリの中に格納
    - メインメモリの中は番地を割り振って領域が分割され、様々な命令やデータが格納されている
  - メインメモリへの命令の格納と管理もCPUの役目
    - それぞれの命令をどの番地に格納するか, etc.

CPUの構成[1](p. 39)



CPUの構成[2](p. 39)

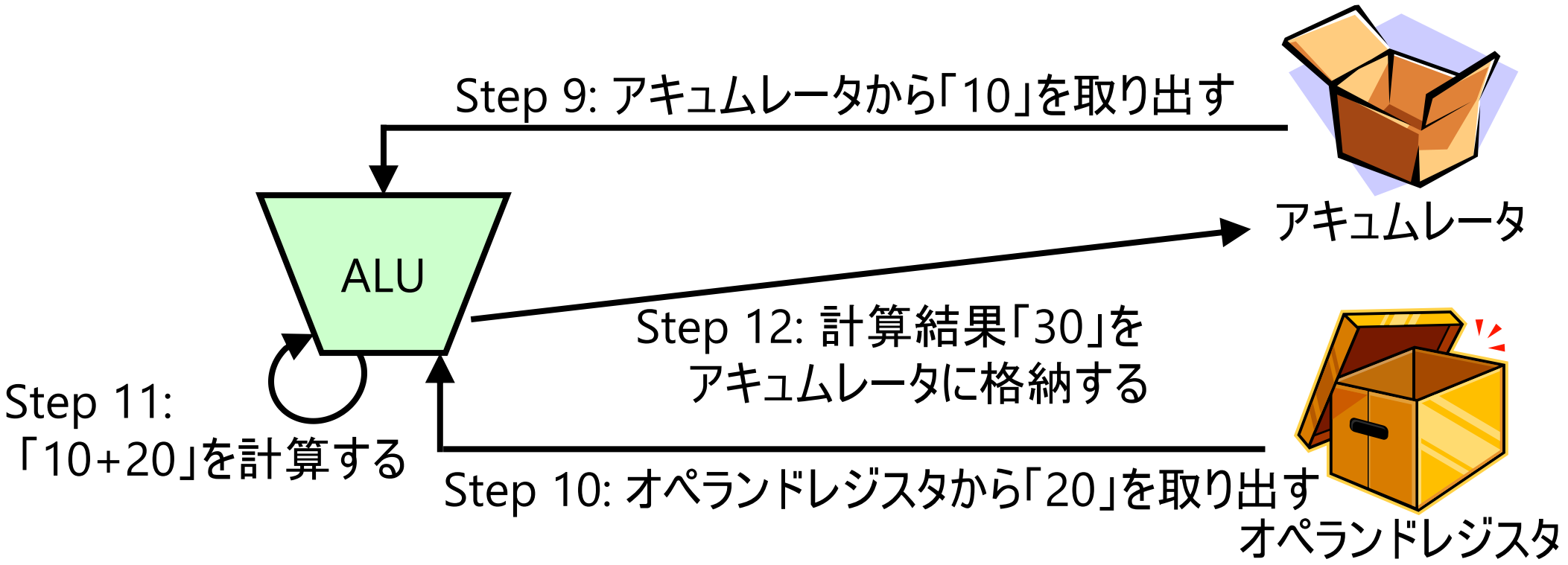
- プログラムカウンタ: メインメモリに格納されている命令を取り出すための番地を指定
- 命令レジスタ: 取り出した命令を一時的に格納
  - 命令: 命令コードとオペランドから構成
    - 命令コード: データ転送や様々な計算、入出力処理などの処理方法
    - オペランド: 命令で使用するデータが格納されている番地や値など
- デコーダ(解読器): 命令コードを解読し(何をすれば良いかを考え)、命令を実行するための信号を出力

CPUの構成[3](p. 40)

- ALU(演算器): 演算(四則計算や論理演算など)を実行
  - メモリやレジスタの記憶されているデータを取り出し
  - 演算に使うデータ(演算数)をアキュムレータに格納
    - 演算数:「xxされる」側のデータ
    - Ex.「A + B」の「A」
  - 演算に使うデータ(被演算数)をオペランドレジスタに格納
    - 被演算数:「xxする」側のデータ
    - Ex.「A + B」の「B」
  - 演算結果をアキュムレータに置き換え
  - 条件コードレジスタに条件コードを設定(必要な場合)
    - 正負の符号の判定やオーバーフローの判定など

CPUの動作例[3](p. 40)

- アキュムレータ: 演算に使うデータや演算結果の格納場所
- オペランドレジスタ: 演算に使われるデータの格納場所



CPUの性能(p. 40)

- クロック周波数: CPUが一段階の動作を行う時間単位(サイクルタイム)
  - 単位: Hz(ヘルツ)
  - Ex. 1GHz ＝ 1000000000Hz(10億Hz)
    ＝ 1秒間に10億回動作
  - 同じモデルのCPU同士であれば、クロックの数値の大きいものが処理が速い
    - モデル: CPUのブランドのようなもの
  - モデルが違えば、同じメーカーでも一概には比較できない
    - CPUが行う一段階分の動作は、CPUのモデルなどによって異なるため

論理回路

論理回路[1](p. 34)

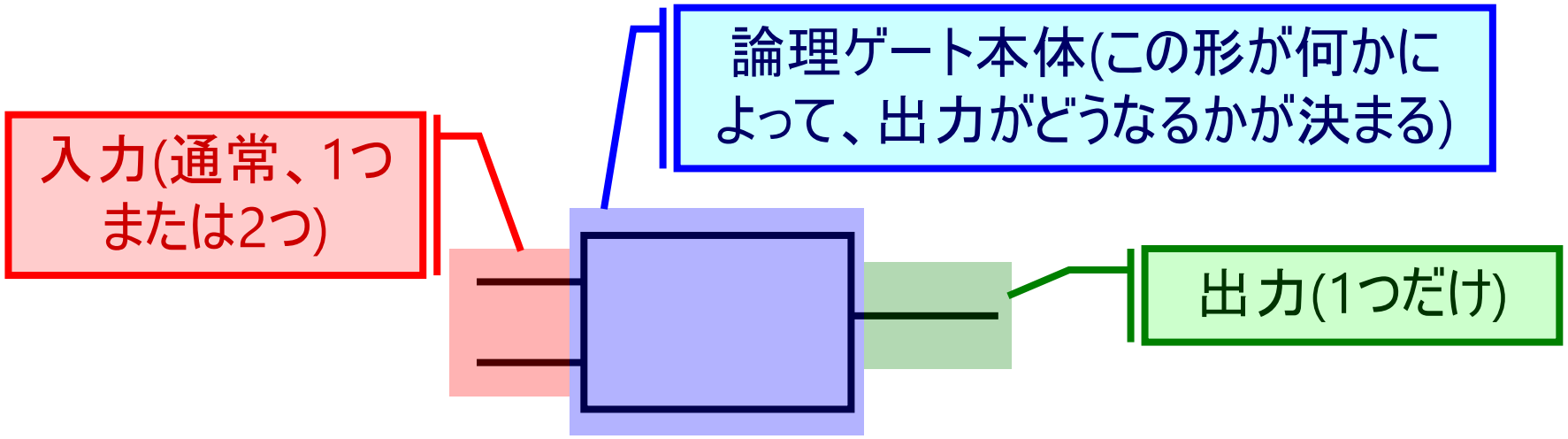
- 論理回路: 論理演算を実現する電子回路
  - 電子回路: 電気を流すことで様々な処理をする部品
- 論理演算: 論理型のデータ同士に対する演算
  - 論理型: 「0」または「1」の2種類のみの2進数で表現できるデータ
  - 1つまたは2つのデータを入力とし、演算結果を出力
- CPUの構成要素(ALUやレジスタなど)は論理回路で構成
  - コンピュータは、様々な処理をするための回路で構成

論理回路[2](p. 36)

- 論理回路をMIL(Military standard)記号を用いて表現
  - MIL記号: 論理回路を構成する部品をイメージ化したもの(図として描くときに利用される絵)
  - 1つ1つの部品を「論理ゲート」と呼ぶ
    - ANDゲート
    - ORゲート
    - NOTゲート
    - NANDゲート
    - NORゲート
    - XORゲート

論理ゲート(p. 36)

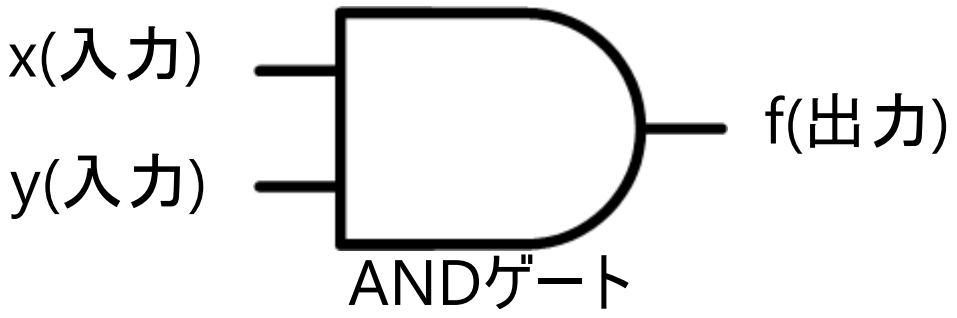
- 論理回路を構成する部品の最小単位
- 「入力」と「出力」の電気信号で構成
  - 「入力」対し、何かの処理をして「出力」とする
    - 入力: 0または1の1ビット
    - 出力: 0または1の1ビット
  - 1つの論理ゲートに、入力は1つまたは2つ、出力は1つ



論理ゲートの基本的な形

論理積[AND][2](p. 36)

- 論理回路は「ANDゲート」で表現
- 2つの入力がどちらも「1」の場合、出力が「1」、2つの入力のどちらかが「0」の場合、出力が「0」

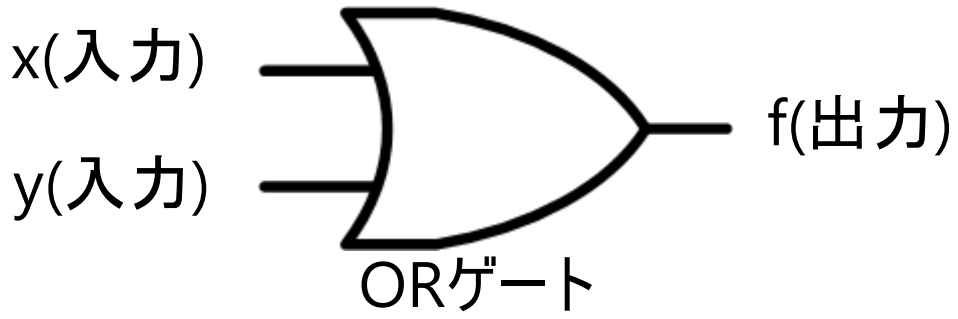


入力と出力の関係

入力		出力(f)
x	y	
0	0	0
0	1	0
1	0	0
1	1	1

論理和[OR][2](p. 37)

- 論理回路は「ORゲート」で表現
- 2つの入力がどちらかが「1」の場合、出力が「1」、2つの入力のどちらも「0」の場合、出力が「0」

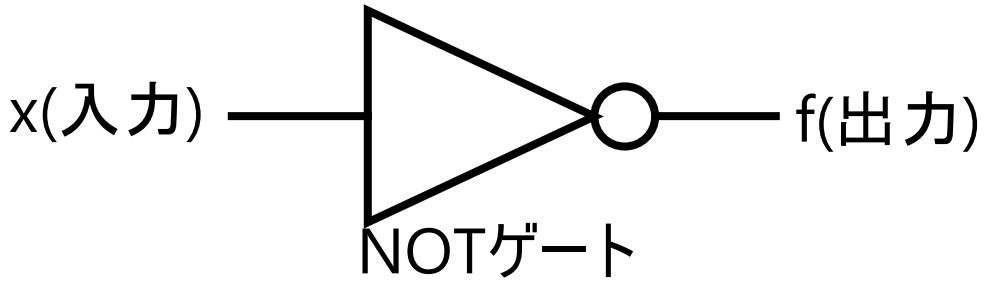


入力と出力の関係

入力		出力(f)
x	y	
0	0	0
0	1	1
1	0	1
1	1	1

否定[NOT](p. 37)

- 1つの入力で、「0」の場合は出力が「1」となり、「1」の場合は出力が「0」となる
  - 入力の逆が出力
- 論理回路は「NOTゲート」で表現

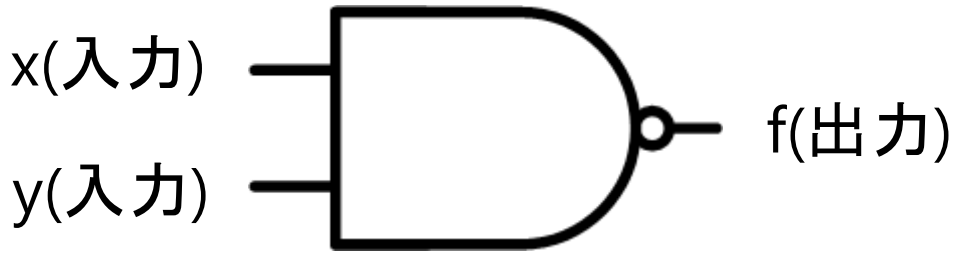


入力と出力の関係

入力(x)	出力(f)
0	1
1	0

NAND[Not AND](p. 37)

- ANDゲートと出力が逆になる
  - 2つの入力がどちらも「1」の場合、出力が「0」となり、2つの入力のどちらかが「0」の場合、出力が「1」となる



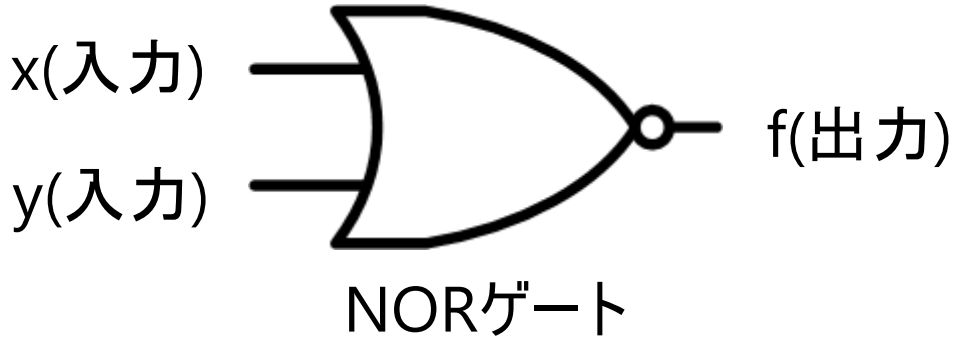
NANDゲート

入力と出力の関係

入力		出力(f)
x	y	
0	0	1
0	1	1
1	0	1
1	1	0

NOR[Not OR](p. 37)

- ORゲートと出力が逆になる
  - 2つの入力がどちらかが「1」の場合、出力が「0」となり、2つの入力のどちらも「0」の場合、出力が「1」となる



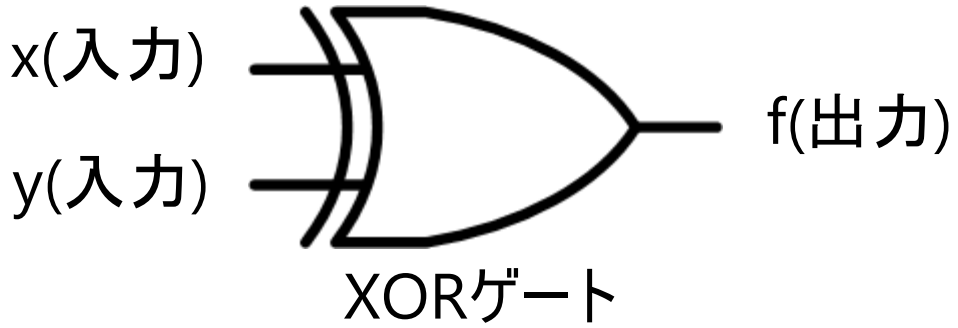
NORゲート

入力と出力の関係

入力		出力(f)
x	y	
0	0	1
0	1	0
1	0	0
1	1	0

排他的論理和[XOR](p. 37)

- XOR: eXclusive OR
- 2つの入力が同じ場合は「0」となり、2つの入力が異なる場合は「1」となる



入力と出力の関係

入力		出力(f)
x	y	
0	0	0
0	1	1
1	0	1
1	1	0

真理値表(p. 37)

- 真理値表: 論理ゲートの入力と出力を表にしたもの

入力		出力				
x	y	AND	OR	NAND	NOR	XOR
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

入力xが0、入力yが0のとき、ANDゲートの出力は0になる、という意味

入力(x)	出力(NOT)
0	1
1	0

組み合わせ回路(p. 37)

- 組み合わせ回路: 入力された内容によって出力が1つに決定される回路
  - 複数の論理回路の入力と出力をつなぐことで構成
  - 入力が何であるかで出力が決まる回路
  - 入力: 電気が線の中を通っているかいないかの状態

組み合わせ回路[例]

- 組み合わせ回路の真理値表

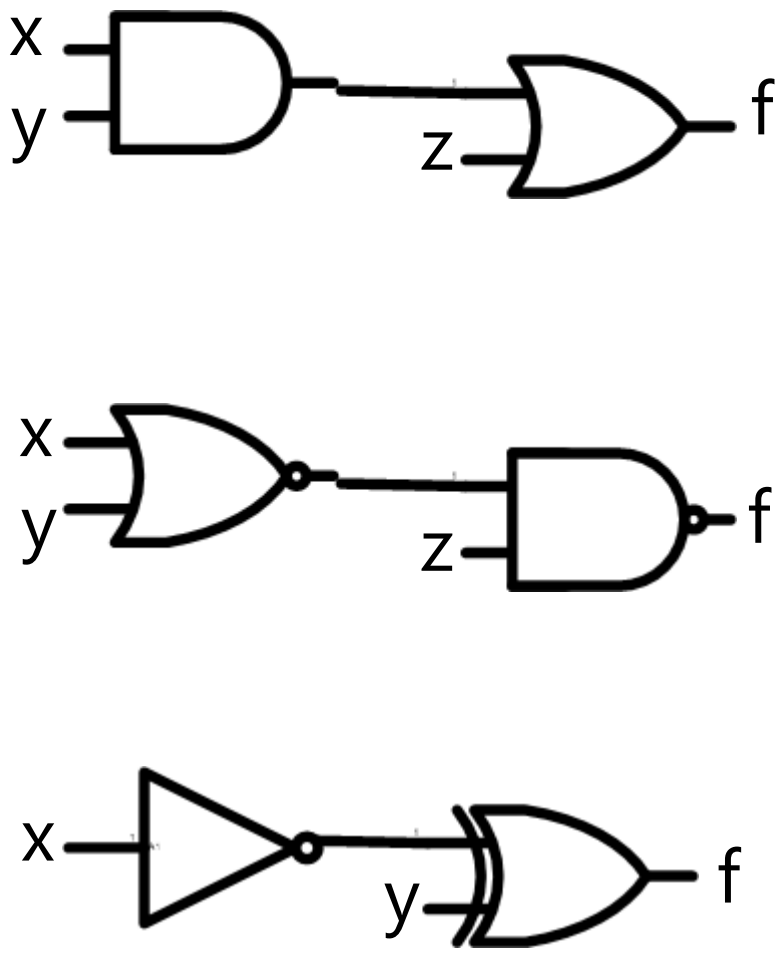
入力			interval	出力 (f)
x	y	z		
0	0	0	0	1
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	1	0

真理値表完成の手順

1. 入力xとyの出力を求める
   (この出力を「interval」とする)
   - intervalはxとyのOR
2. intervalと入力zの出力fを求める
   - 出力fはintervalとzとのNAND

やってみよう!

- 回路の真理値表をそれぞれ埋めてみよう!



入力			出力 (f)
x	y	z	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

2進数での足し算の復習

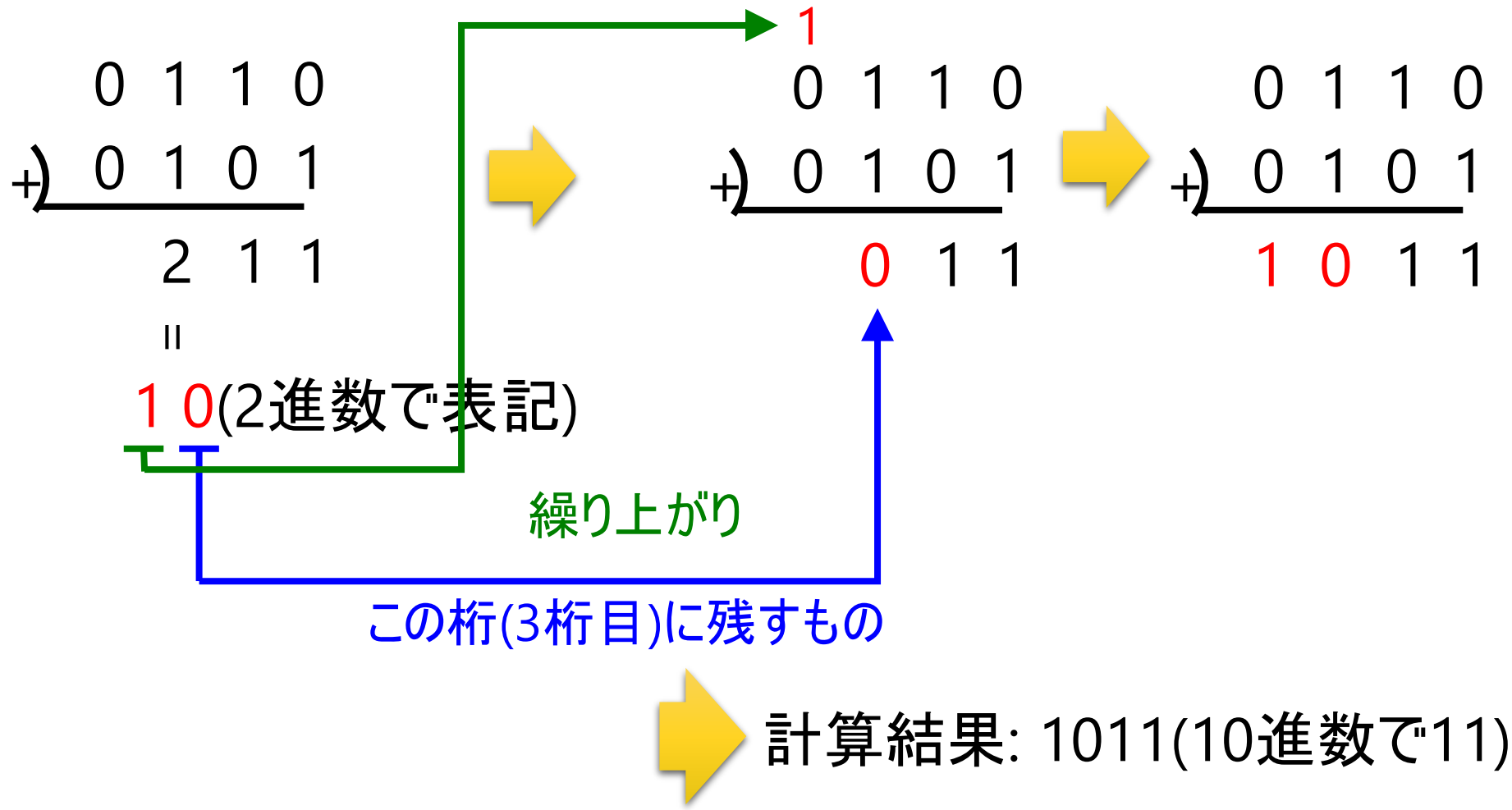
足し算をする方法[1](p. 6)

- 10進数での1桁の足し算
  - たくさん(10×10=100)のパターンが存在
  - 1+1, 1+2, 1+3, … 2+1, 2+2, 2+3, … … 8+6(繰り上がり1), 8+7(繰り上がり1), … …
- 2進数での1桁の足し算
  - 4通り
  - 足した結果が2になると繰り上がり1(2進数では10進数の2を「10」と表すため)
  - 0+0, 0+1, 1+0, 1+1(繰り上がり1)

足し算をする方法[2](p. 6)

- 基本的な2進数の足し算の方法は10進数と同じ

0110(10進数で6)と0101(10進数で5)の足し算



桁あふれ(オーバーフロー)[1]

- コンピュータでは数を表すビット数(2進数の桁数)は決まっている
  - 計算の結果、決まった桁数を超えると…?

Ex. 数を4ビット(4桁)で表す場合

1110(10進数で14)と0101(10進数で5)の足し算

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

桁あふれ(オーバーフロー)[2]

Ex. 数を4ビット(4桁)で表す場合

1110(10進数で14)と0101(10進数で5)の足し算

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

1 ← 5ビット目(5桁目，決められた桁数を越えてしまった部分)

決められた桁数を越えた部分は無視される(捨てられてしまう)

~~1~~ 0 0 1 1

無視される(捨てられる)

計算結果: 0011(10進数で3)

計算結果が決められた桁数を超えること:
桁あふれ(オーバーフロー)

桁あふれ(オーバーフロー)[3]

- コンピュータの世界では、数を表現する2進数の桁数は常に一定
  - 計算内容などによる変化はなし
  - 通常は、32桁または64桁で数を表現
    - 授業のスライドは、そんなに長く書けないので、小さい桁数で表現

- 桁あふれ(オーバーフロー)が起こると…
  本来の計算結果とコンピュータでの結果が違ってしまう
  - Ex. 4桁の2進数1110と1010の計算結果: 10011
    - 10011は5桁になってしまったので、0011という4桁で表現
      →本来の計算結果とは違う結果

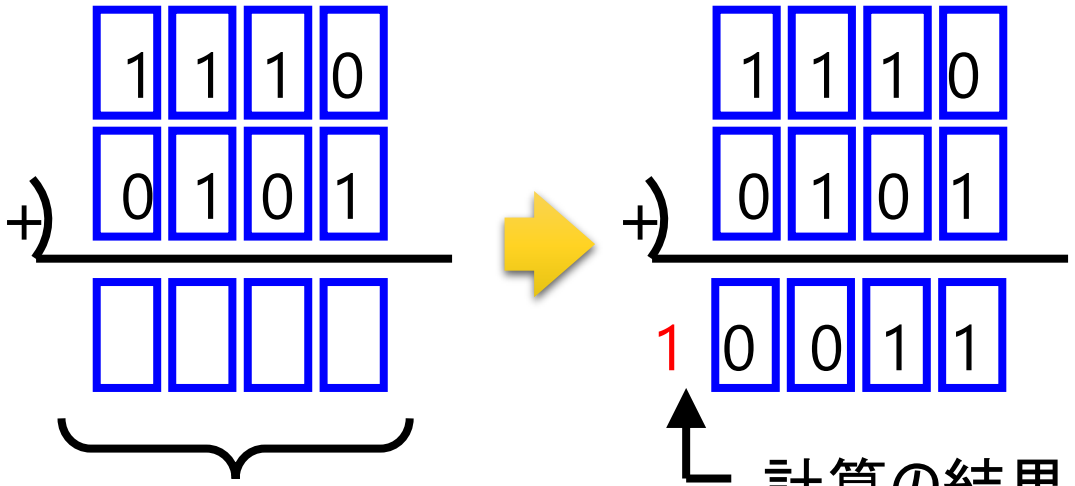
桁あふれ(オーバーフロー)の扱い[1]

- コンピュータでは、2進数の各桁を、1つずつ箱に入れて扱っている、というイメージ
  - 各桁を入れる箱の数に限りがある
    - Ex. 数を4ビットで表す ＝ 数を4桁で表す(2進数の各桁を入れる箱の数が4個)
  - どのような計算をしたとしても、箱の数は変更されない
    - Ex. 数を4ビットで表すときに、$(1110 + 0101)_2$の計算結果も4ビットでしか表現できない(箱は4個しかない)

本来の計算結果(人間が自分の手で行った計算結果)とコンピュータが行った計算結果(Ex. 電卓などの計算結果)が違ってしまう現象

桁あふれ(オーバーフロー)の扱い[2]

- 2進数の各桁を入れる箱は、小さい桁(右の桁)の分から用意される



計算結果を入れるために
用意されている箱

計算の結果、5桁目に突入してしまった
but...
箱は4つしか用意されていない

5桁目は無視されるので計算結果は$(0011)_2$

加算回路

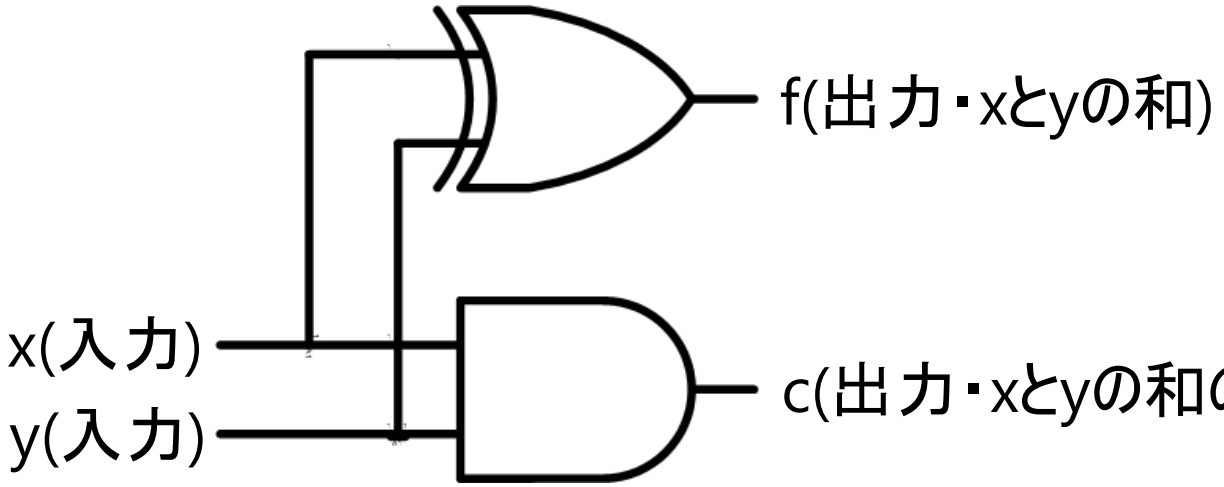
加算器(p. 37)

- 加算器: CPUの中で、2進数1桁の足し算をするための回路
  - 半加算器: 繰り上がりの加算をしない回路
    - 入力: 1ビットの数2つ
    - 出力: 2つの数を足した結果と繰り上がり
  - 全加算器: 繰り上がりの加算をする回路
    - 入力: 1ビットの数2つと1つ下の桁からの繰り上がり
    - 出力: 3つの数を足した結果と繰り上がり

半加算器

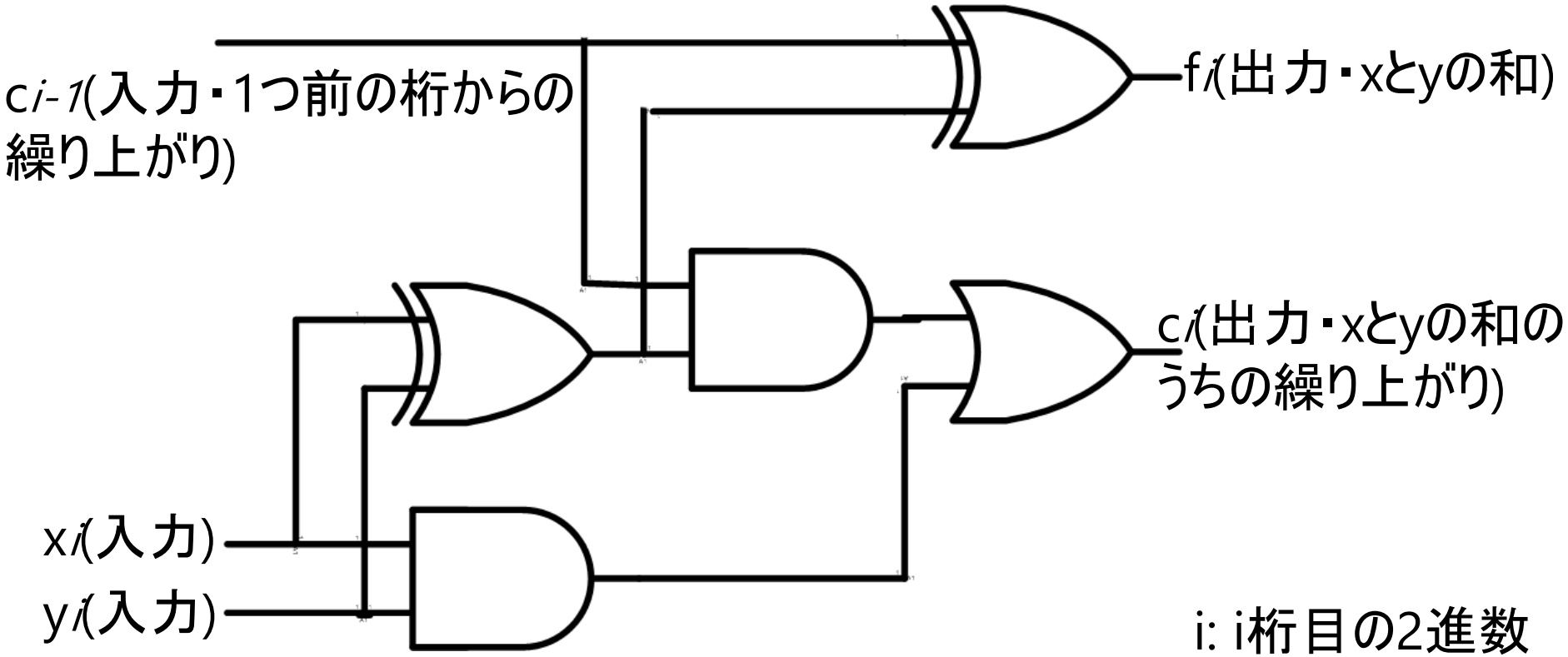
- 2進数1桁の数2つの加算をする回路
- 入力: 2進数1桁の数2つ
- 出力: 2つの数の和と繰り上がり

入力		出力	
x	y	c	f
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



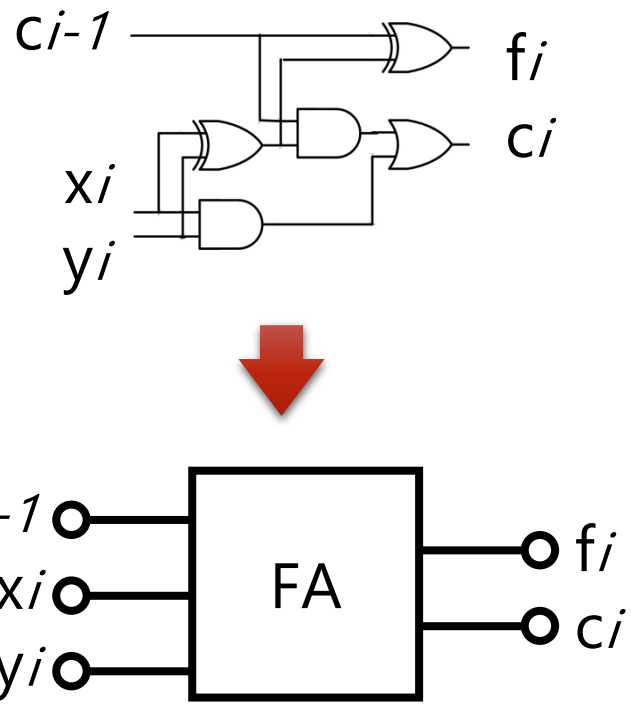
全加算器[1](p. 37)

- 2進数1桁の数2つの加算をする回路
- 入力: 2桁の数2つと1つ前の桁からの繰り上がり
- 出力: 2つの数の和と繰り上がり



全加算器[2](p. 37)

入力			出力	
x_i	y_i	c_{i-1}	c_i	f_i
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

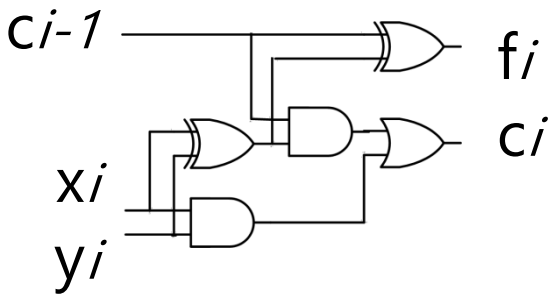


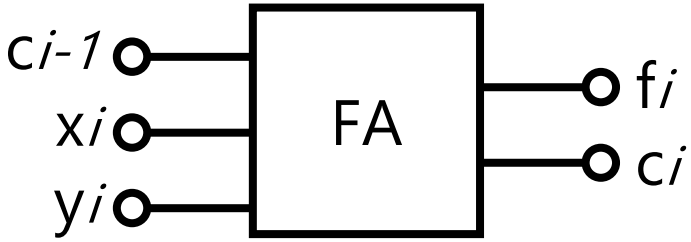
全加算器の略表記
(i: 2進数の中のi桁目)

※テストの時に真理値表などが示されるかどうかは、その時々で異なる
(2009年度秋の基本情報技術者試験の問題では、何もなし)

全加算器[3](p. 37)

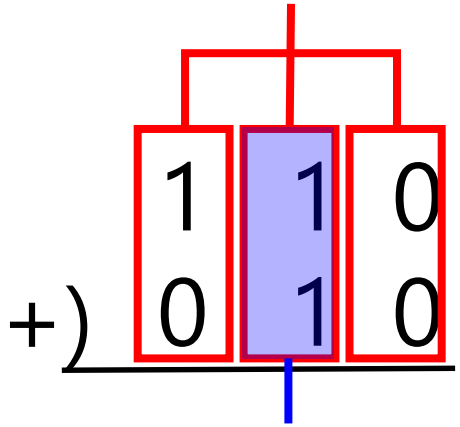
全加算器: 2進数1桁の足し算を行う回路



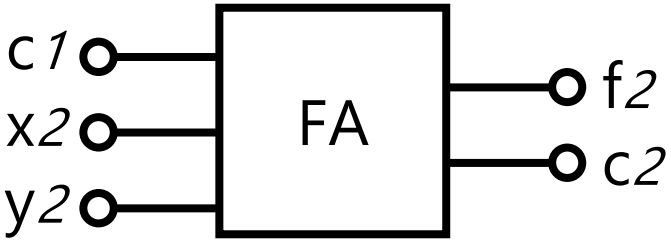


全加算器の略表記
(i: 2進数の中のi桁目)

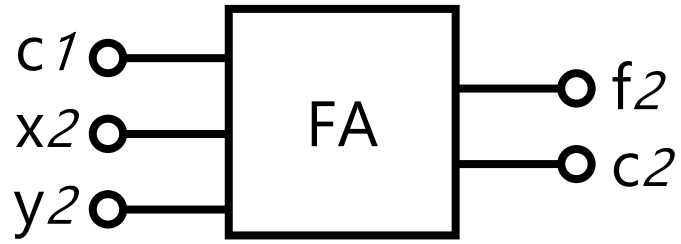
全加算器が3つ必要
(1つの桁を1つの全加算器で計算)



Ex. 2桁目の計算は...



全加算器[4](p. 37)



c1(1桁目からの繰り上がり)

x2

y2

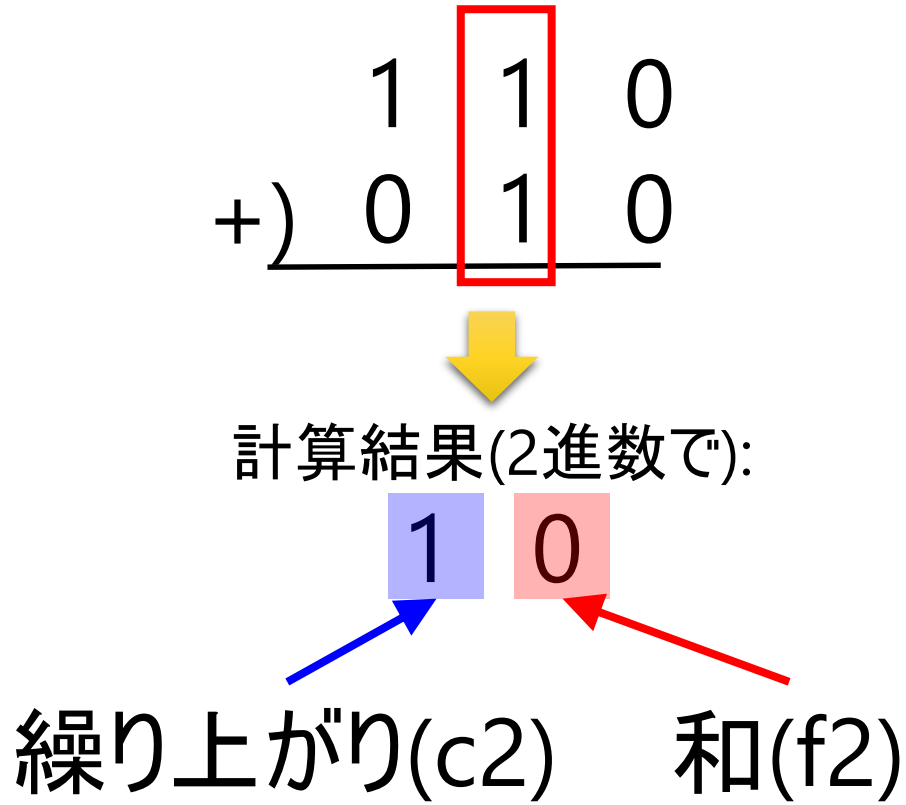
	0		
1	1	0	
+) 0	1	0	

x2 : 1, y2 : 1, c1 : 0

入力			出力	
xi	yi	ci-1	ci	fi
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

2桁目の計算結果: f2 : 0, c2 : 1

全加算器[5](p. 37)



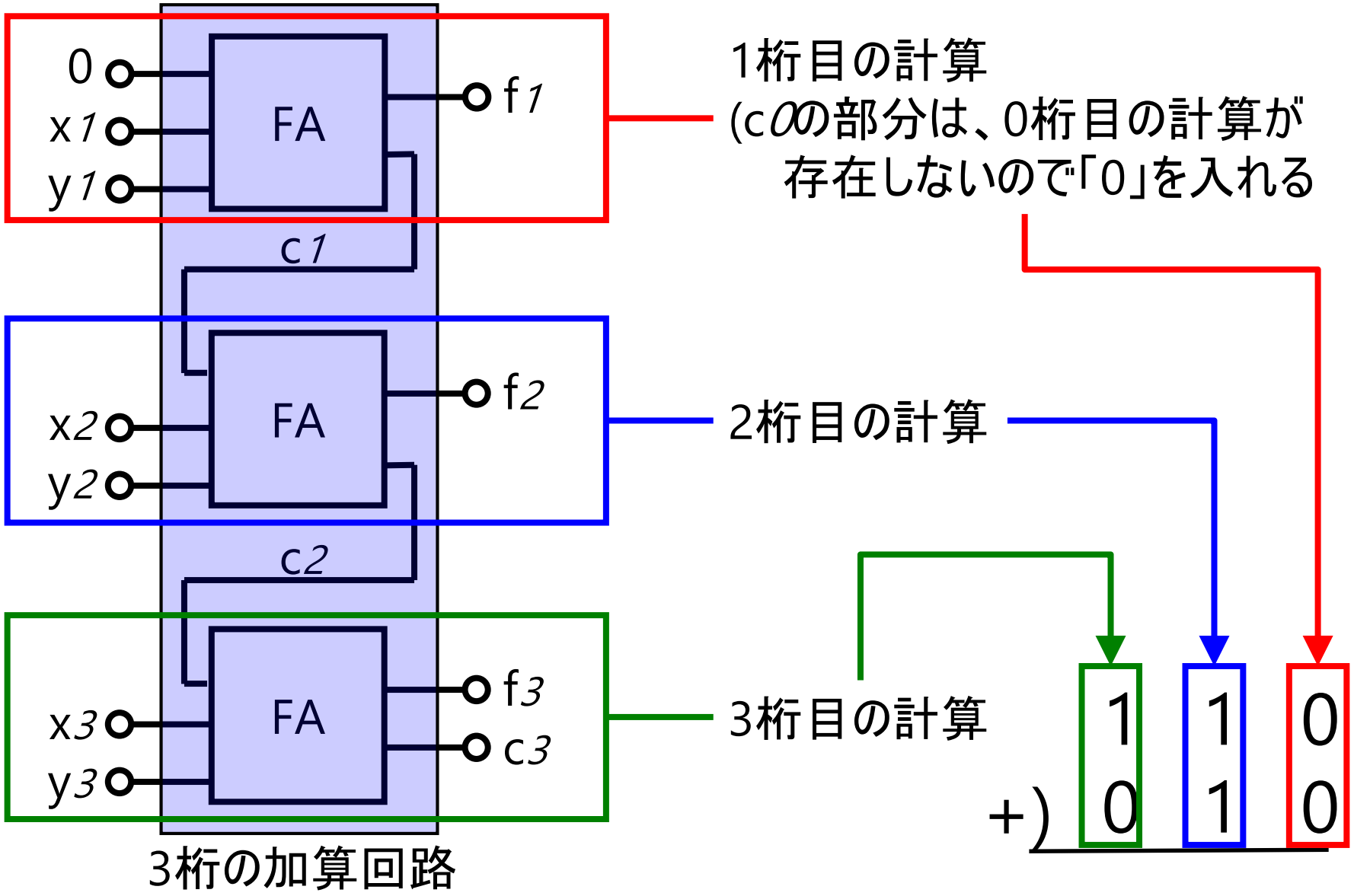
加算回路での計算は、必ず真理値表に従う

- 人間がするような計算はしていない
- オーバーフローが起こらない限り、人間が計算した結果と加算回路での結果は同じになる

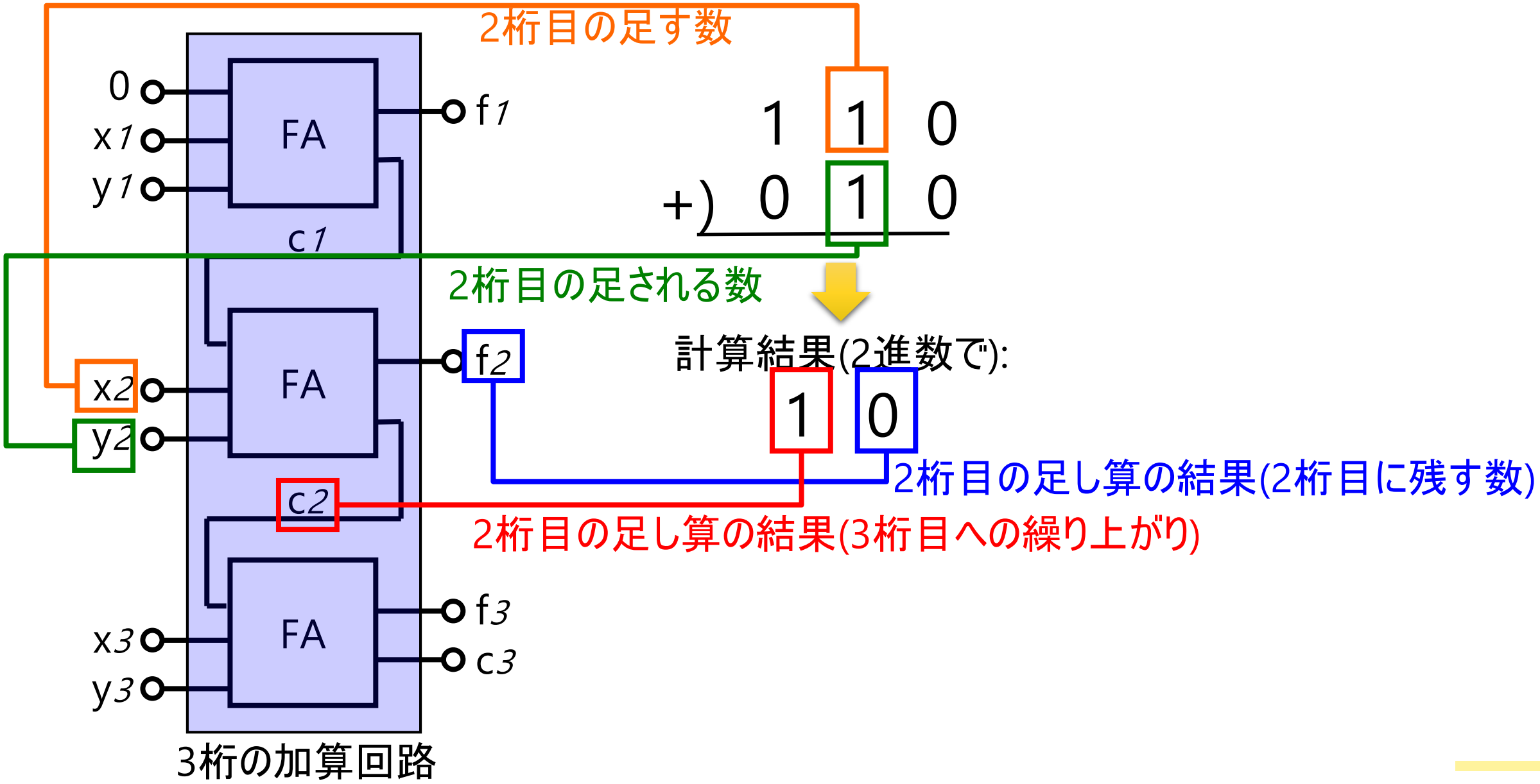
加算回路[1](p. 37)

- 加算回路: ALUの中で足し算をするための回路
  - 全加算器を複数組み合わせることで構成
  - 全加算器をいくつ組み合わせるかで、何桁の2進数の足し算ができるかが決定

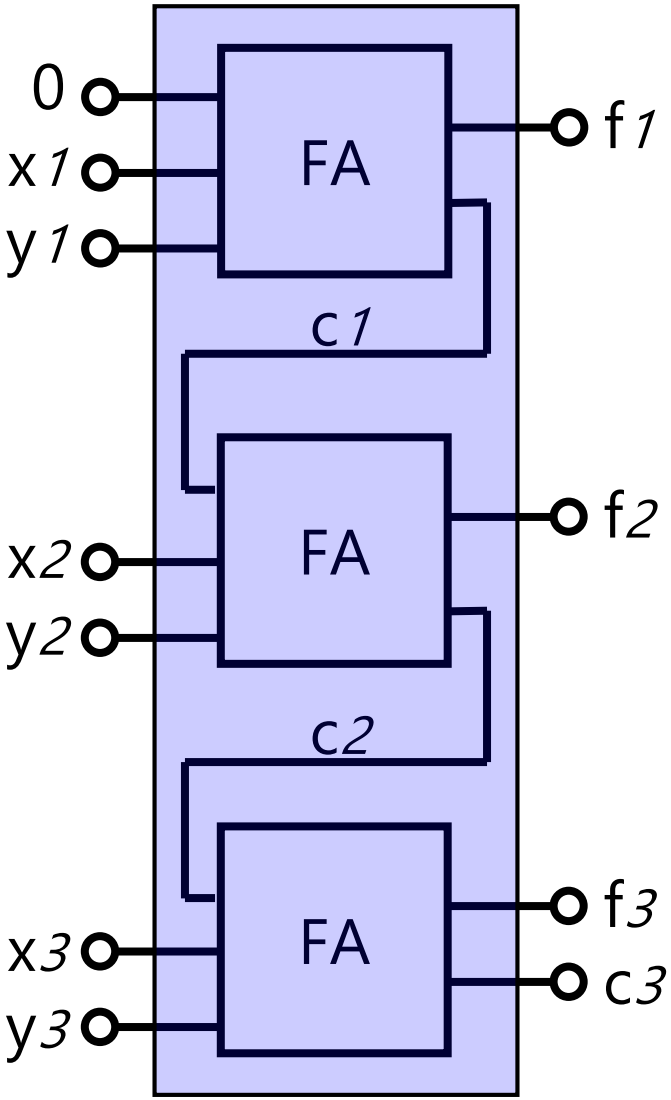
加算回路[2](p. 37)



加算回路[3](p. 37)

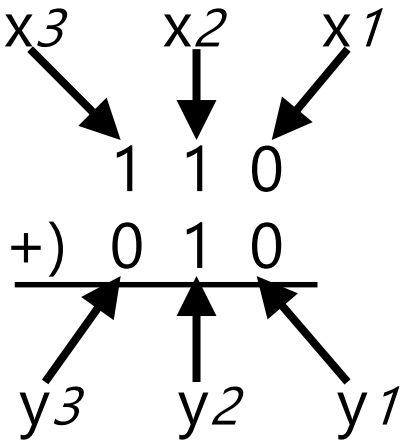


加算回路[2](p. 37)



3桁の加算回路

例えば…



f1: 0, c1: 0
f2: 0, c2: 1
f3: 0, c3: 1

※x0の部分は、0桁目の数の繰り上がりは
存在しないので「0」を入れる

やってみよう!

- 全加算器で2進数の足し算をしてみて、結果が正しいことを確認しよう
  - 01101 + 01100 (2進数5桁, 結果は11001)
  - 0011 + 0110 (2進数4桁, 結果は1001)
- 全加算器において、入力xが1、入力yが0、入力zが1のとき、出力c(繰り上げ)とf(和)はどれになるか
  - ア: c - 0, f - 1
  - イ: c - 0, f - 1
  - ウ: c - 1, f - 0
  - エ: c - 1, f - 1
    (2009年度基本情報技術者試験 秋期問題より)

フリップフロップ

順序回路(p. 38)

- 順序回路: 入力された内容と過去の入力の内容によって出力が決定される回路
  - 「現在」の入力内容と、「過去」の入力によって設定された現在の回路の状態によって出力が決定

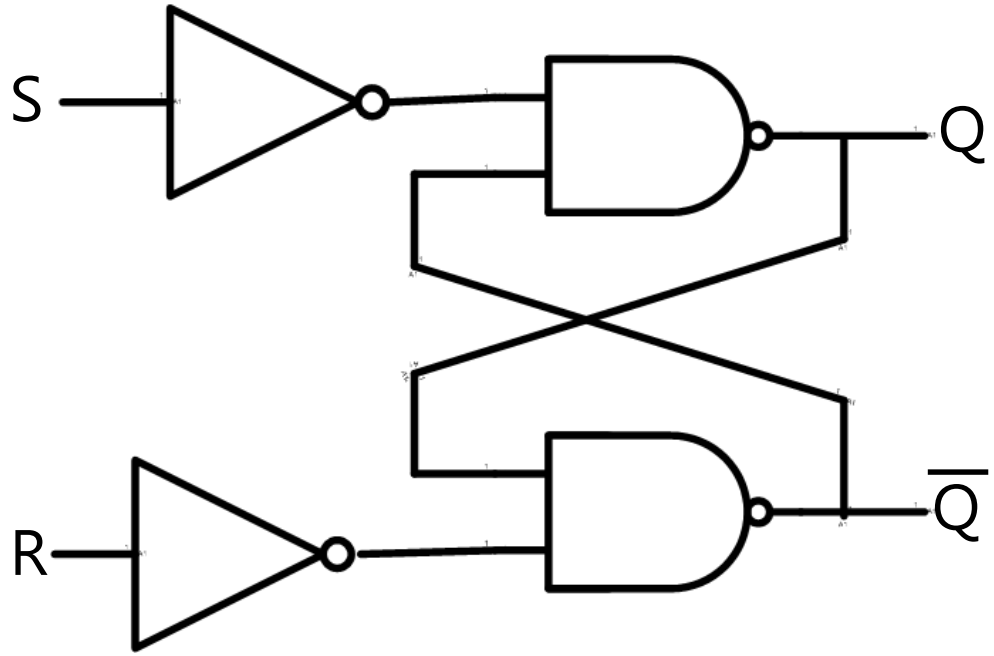
過去の入力や状態を記憶しておく必要

フリップフロップ回路

※組み合わせ回路:「現在」の入力内容のみで出力が決定

フリップフロップ回路[1](p. 38)

- フリップフロップ回路: 状態を保持しておくための回路



SRフリップフロップ

S	R	Q'
0	0	Q(状態保存)
0	1	0(リセット状態)
1	0	1(セット状態)
1	1	ー(禁止)

フリップフロップ回路[2](p. 38)

- セット入力: S=1, R=0を入力した状態
  - 出力Q=1となる
  - セット入力の後、Sを0にしてもQ=1の状態が保持される
- リセット入力: S=0, R=1を入力した状態
  - 出力Q=0となる
  - リセット入力の後、Sを1にしてもQ=0の状態が保持される
- その他の入力
  - S=0, R=0のとき: この入力の直前の状態のまま
  - S=1, R=1のとき: 状態が不安定になるので禁止

フリップフロップ回路[3](p. 38)

- 2つの安定状態
  - セット入力をした状態(セット状態)
  - リセット入力をした状態(リセット状態)
- 一方の安定状態からもう一方の安定状態へ、入力を切り替えることで遷移
  - 入力を切り替えるまで、1つの状態(2進数1桁)を保持している
    →1つのデータを記憶している