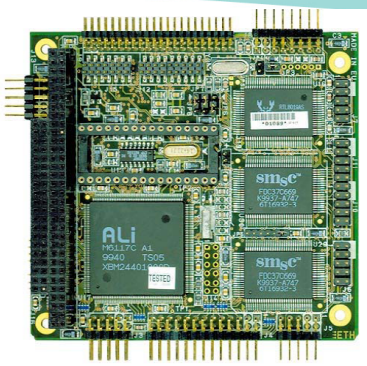
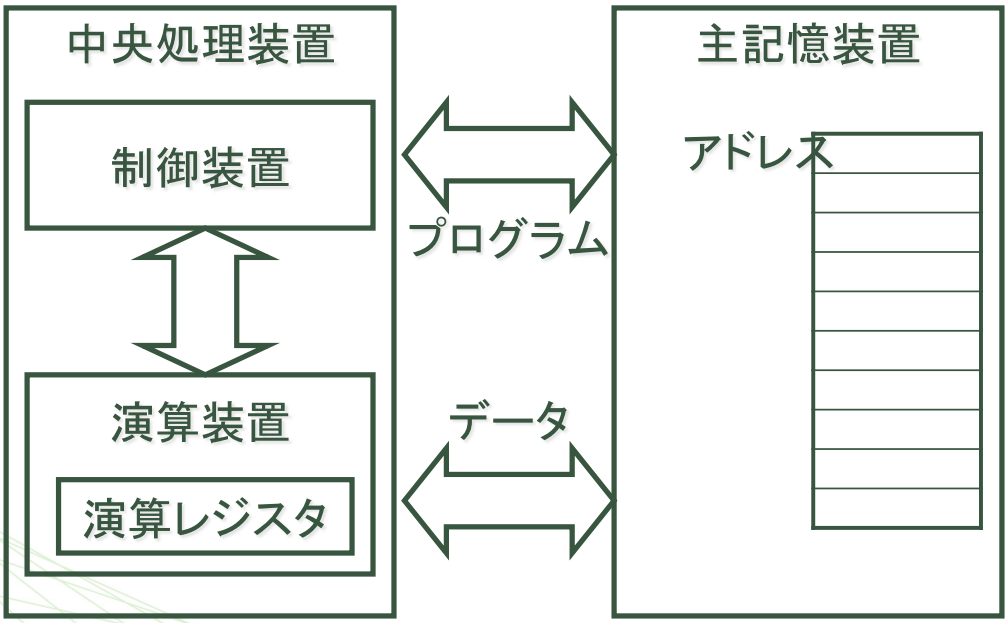
第7章 計算の機構

コンピュータの基本構成

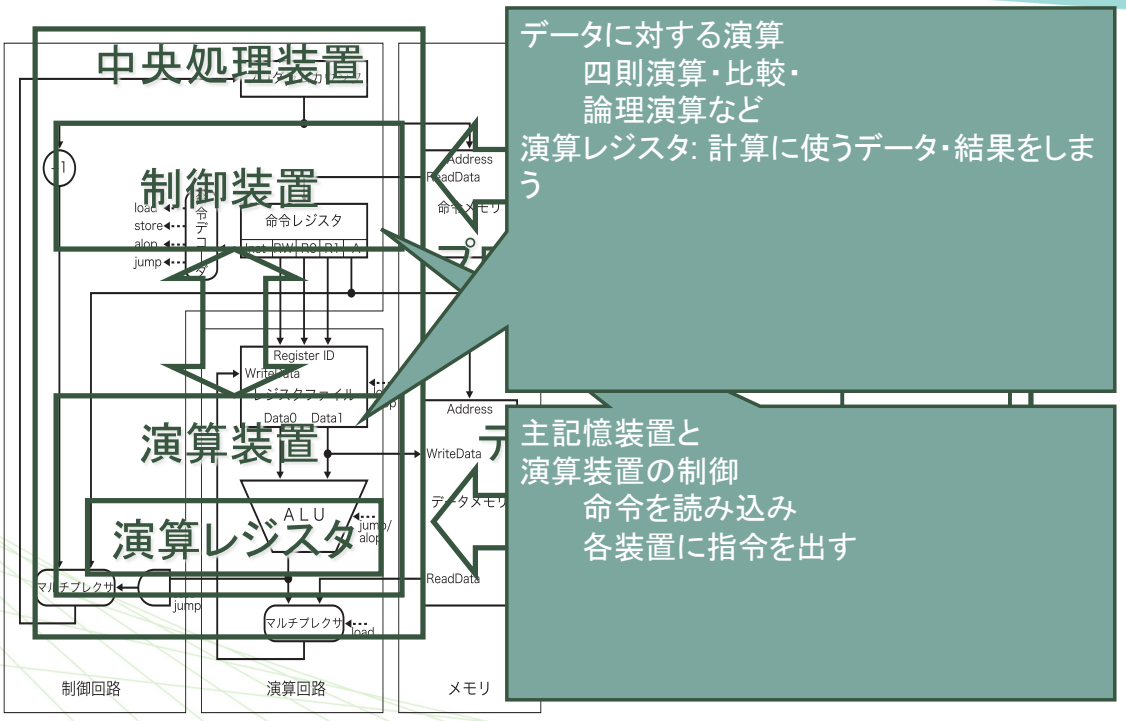


- 中央処理装置（CPU）
  - 制御装置
    - 主記憶装置と演算装置の制御
  - 演算装置
    - データに対する演算処理
    - 演算レジスタが一時的な記憶を保持
      - 1つのレジスタはアキュムレータとも呼ばれる
- 主記憶装置（メインメモリ）
  - 情報（データ）の格納と選択的な読み書き
  - アドレスによってデータの位置を特定

プログラム内蔵方式の基本構成



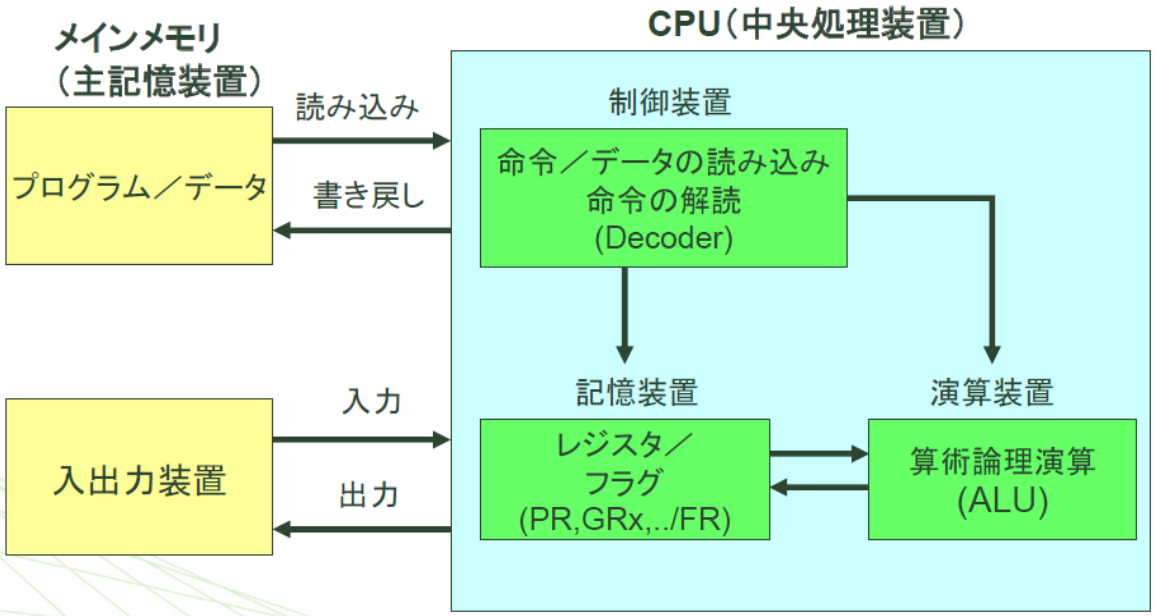
コンピュータの内部構成: 中央処理装置



データに対する演算
　四則演算・比較・
　論理演算など
演算レジスタ: 計算に使うデータ・結果をしまう

主記憶装置と
演算装置の制御
　命令を読み込み
　各装置に指令を出す

CPUの構成



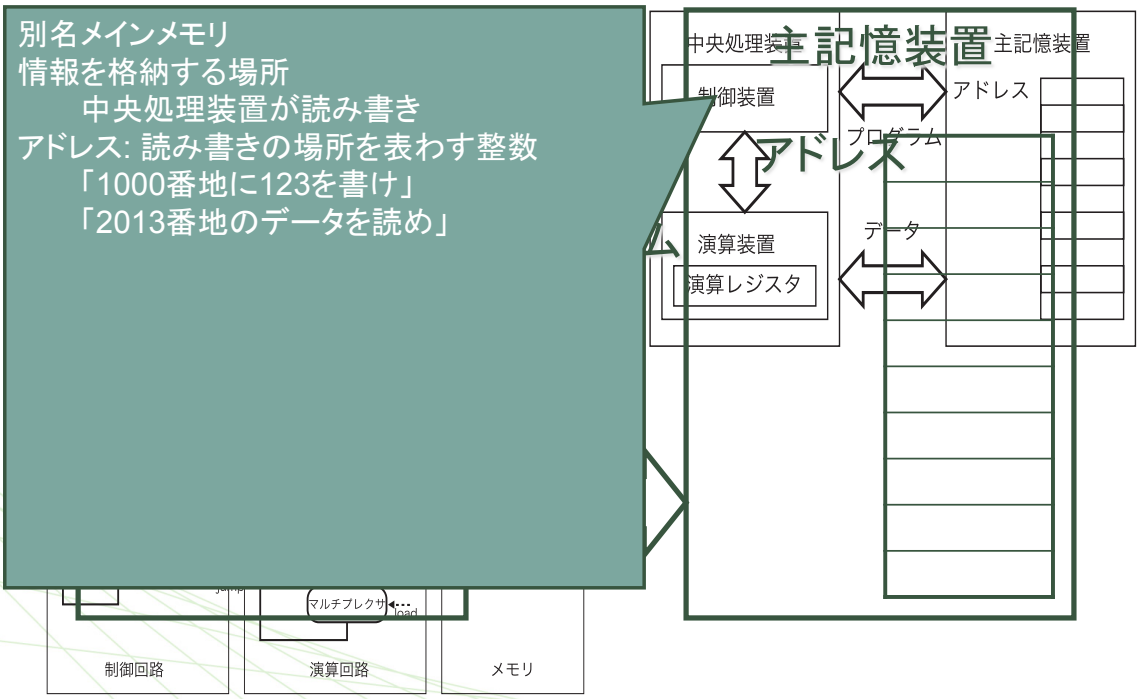
配布不可

CPUの構成

- **制御装置(controller, decoder)**
  - 機械語を解析して命令の種類と対象を特定
  - 対応する演算・入出力操作のために適時制御信号を出力
- **演算装置(ALU)**
  - 機械語命令に従って各種の演算を実行
  - 記憶装置（メモリ、レジスタ）から記憶装置（メモリ、レジスタ）へデータを転送し、演算を行う

コンピュータの内部構成: 主記憶装置

別名メインメモリ
情報を格納する場所
　　中央処理装置が読み書き
アドレス: 読み書きの場所を表わす整数
　　「1000番地に123を書け」
　　「2013番地のデータを読め」



主記憶装置(メモリ)

メモリとは

- データの読み書きが可能な半導体メモリ
- 機械語命令とデータを格納する

1Kbyte のメモリのイメージ

アドレス	メモリの内容
0000000000	1バイトのデータ
0000000001	1バイトのデータ
0000000010	1バイトのデータ
:	:
1111111110	1バイトのデータ
1111111111	1バイトのデータ

1フロアに1バイト(8ビット)のデータが格納された1024階建てのビルディング

機械語レベルのプログラム

- コンピュータに対する命令の並び
- 命令集合：個々のCPUで利用できる命令群
  - 四則演算，メモリの読み書き，判断，繰り返しなど
  - メモリと演算レジスタを操作
- 命令の構成
  - 命令コード(operator，演算子)：命令の種類を表す符号
    - load，add など
  - オペランド(operand，被演算子)：命令の付加情報
    - A（アドレス値を表す）など

命令集合の例

種類	内容	意味
データ転送命令	load A store A	アドレスAのデータを演算レジスタ(AC)に読み込む ACのデータをアドレスAに書き込む
計算命令	add A subtract A	アドレスAのデータをACの値に加える アドレスAのデータをACの値から引く
分岐命令	jump A jumpzero A	アドレスAにプログラムの実行を移す ACのデータが0の場合，アドレスAにプログラムの実行を移す
その他	write halt	ACのデータを出力する プログラムの実行を停止する

機械語命令の主な種類と機能

種類	機能
データ転送命令 （ロード，ストア）	レジスタとメモリ，メモリとメモリ，レジスタと周辺装置の間でデータを読み書きする
演算命令	アキュムレータで算術演算，論理演算，比較演算，およびシフト演算を行う
ジャンプ命令	条件分岐，繰り返し，および無条件のジャンプを行う
コール/リターン命令	関数を呼び出す/呼び出し元に戻る

1から10までの和のプログラム

アドレス	命令	意味	高水準言語
1001	load 2001	AC ← 2001	sum ← sum+1
1002	add 2002	AC ← AC + 2002	
1003	store 2001	2001 ← AC	
1004	load 2002	AC ← 2002	i ← i - 1
1005	subtract 2003	AC ← AC - 2003	
1006	store 2002	2002 ← AC	
1007	jumpzero 1009	条件分岐（ジャンプ）	while i ≠ 0
1008	jump 1001	無条件ジャンプ	
1009	load 2001	AC ← 2001	
1010	write	結果の出力	write(sum)
1011	halt	プログラム停止	
2001	0	変数（結果）	sum ← 0
2002	10	変数（ i の初期値）	i ← 10
2003	1	定数	1

COMETⅡ（仮想CPU）とCASLⅡ（アセンブリ言語）の命令

発展

オペコード	機能
LD	ロード
ST	ストア
LAD	ロード・アドレス
ADDA	算術加算
ADDL	論理加算
SUBA	算術減算
SUBL	論理減算
AND	論理積
OR	論理和
XOR	排他的論理和
CPA	算術比較
CPL	論理比較
SLA	算術左シフト
SRA	算術右シフト

オペコード	機能
SLL	論理左シフト
SRL	論理右シフト
JPL	正分岐
JMI	負分岐
JNZ	非零分岐
JZE	零分岐
JOV	オーバーフロー分岐
JUMP	無条件分岐
PUSH	プッシュ
POP	ポップ
CALL	コール
RET	リターン
SVC	スーパーバイザ・コール
NOP	何もしない

配布不可

メモリに配置されたプログラムの例

発展

↘ 加算結果を表示するプログラム

```
a = 123;
b = 456;
a = a + b;
```

アドレス	メモリーの内容	アセンブリ言語	
0100	命令：0105番地の値を汎用レジスタ0に格納せよ	LD	GR0, #105
0101	命令：0106番地の値を汎用レジスタ1に格納せよ	LD	GR1, #106
0102	命令：汎用レジスタ0の値と汎用レジスタ1の値を加算	ADDA	GR0, GR1
0103	命令：汎用レジスタ0の値を#105に格納せよ	ST	GR0, #105
0104	命令：プログラムを終了（OSに戻る）	RET	
0105	データ：123	DC	123
0106	データ：456	DC	456

配布不可

ジャンプ命令

- 無条件ジャンプ
- 条件付きジャンプ
  - フラグレジスタFRの値によってプログラムレジスタPRへ実効アドレスをわたす

正分岐	JPL	adr[,X] ・・・(Jump on PLus)
負分岐	JMI	adr[,X] ・・・(Jump on MInus)
非零分岐	JNZ	adr[,X] ・・・(Jump on Non Zero)
零分岐	JZE	adr[,X] ・・・(Jump on ZEro)
オーバフロー分岐	JOV	adr[,X] ・・・(Jump on OVerflow)
無条件分岐	JUMP	adr[,X] ・・・(Unconditional JUMP)

  - 分岐命令によってフラグレジスタは変化しない

条件付きジャンプ(フラグレジスタ)

	OF	SF		ZF
JPL		0	&	0
JMI		1		
JNZ				0
JZE				1
JOV	1			

条件分岐

発展

```
a = -5;
if a < 0 then a = -a;
printf("%d\n", a);
```

アドレス	メモリーの内容	アセンブリ言語	
0100	命令：0106番地の値を汎用レジスタ0に格納せよ	LD	GR0, #106
0101	命令：汎用レジスタ0の値を0と比較せよ	LAD CPA	GR1, 0 GR0,GR1
0102	命令：大きければ0104番地にジャンプせよ	JPL	#104
0103	命令：汎用レジスタ0の値の符号を反転せよ	SUBA LD	GR1, GR0 GR0, GR1
0104	命令：汎用レジスタ0の値をディスプレイに表示	OUTN*1	GR0
0105	命令：プログラムを終了（OSに戻る）	RET	
0106	データ：-5	DC	-5

配布不可

*1 指定のレジスタの値の整数をディスプレイに出力するマクロとする

繰り返し

発展

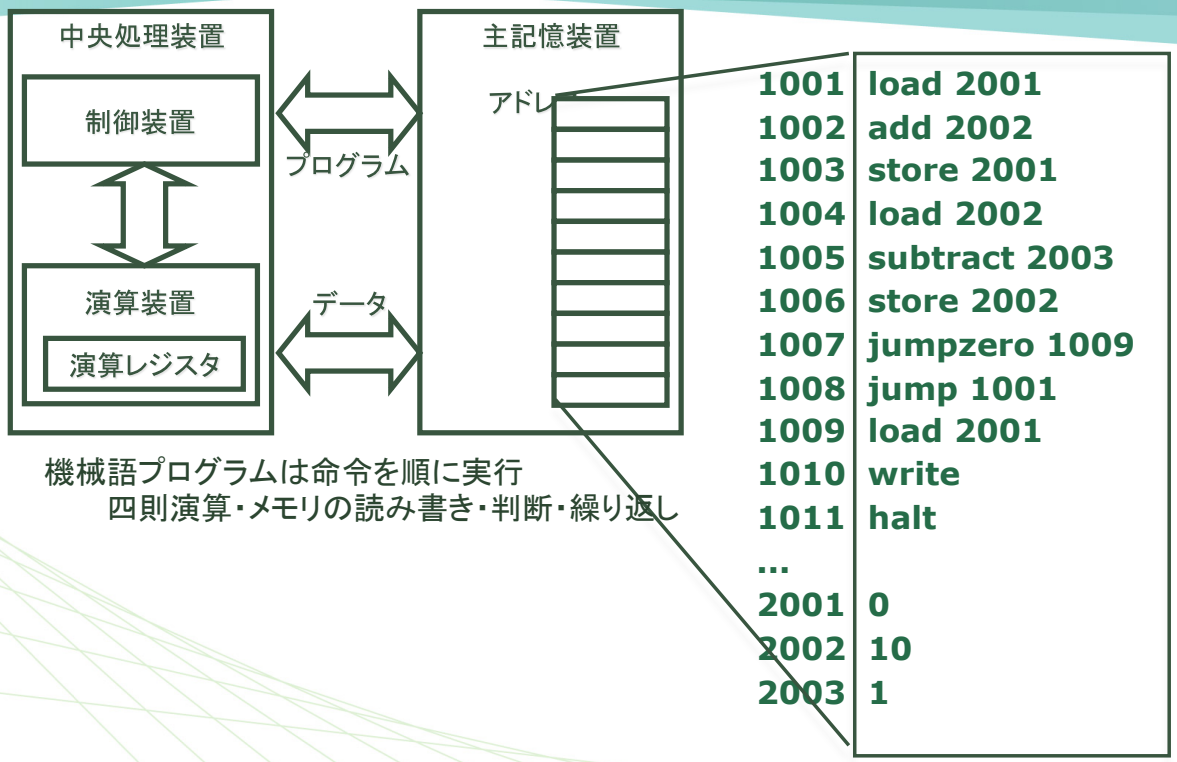
```
for(i=0; i<10; i++) {
  printf ("%d\n", i);
}
```

アドレス	メモリーの内容	アセンブリ言語	
0100	命令：汎用レジスタ0の値を0にせよ	LDA	GR0, 0
0101	命令：汎用レジスタ0の値を10と比較せよ(GR0-10)	LAD CPA	GR1, 10 GR0,GR1
0102	命令：同じ場合（結果=0)は0106番地にジャンプせよ	JZE	#106
0103	命令：汎用レジスタ0の値をディスプレイに表示せよ	OUTN*1	GR0
0104	命令：汎用レジスタ0の値を1加算せよ	LAD ADDA	GR2, 1 GR0, GR2
0105	命令：0101番地にジャンプせよ	JUMP	#101
0106	命令：プログラムを終了せよ（OSに戻れ）	RET	

配布不可

*1 指定のレジスタの値の整数をディスプレイに出力するマクロとする

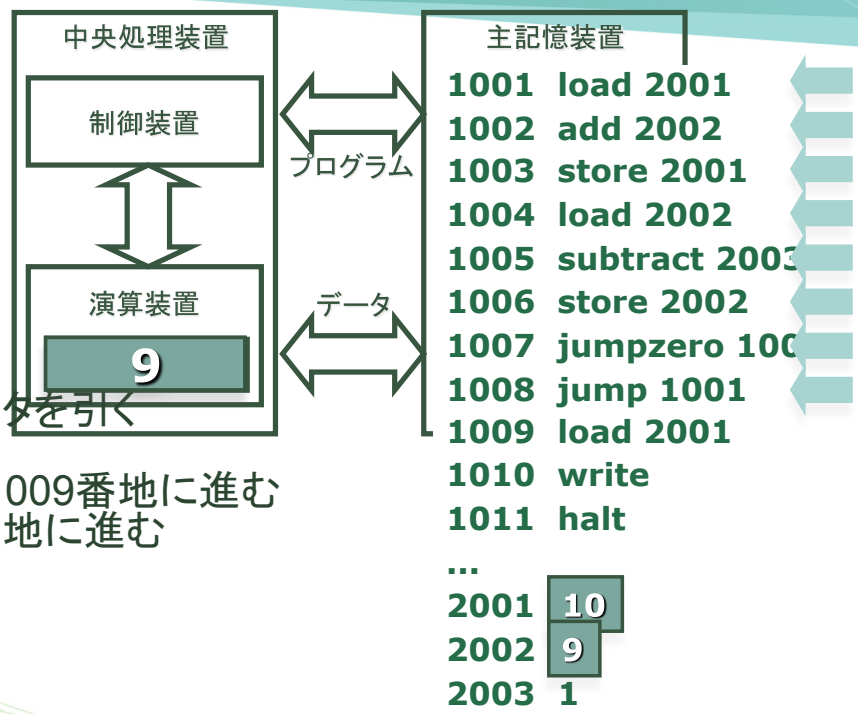
機械語プログラムの動き



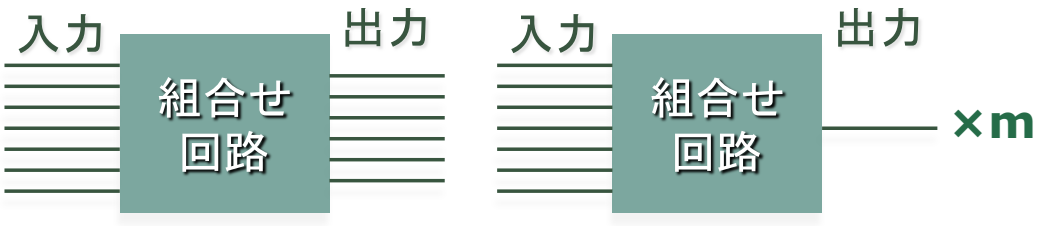
機械語プログラムは命令を順に実行
四則演算・メモリの読み書き・判断・繰り返し

機械語プログラムの動き

2001番地の
データを読む
2002番地の
データを
加える
2001番地に
書く
2002番地の
データを読む
2003番地のデータを引く
2002番地に書く
レジスタが0なら1009番地に進む
無条件に1001番地に進む



論理演算と組合せ回路



- 情報の表現: ビットパターン → 信号のON/OFF
- 演算 = 関数
  = n本の入力信号からm本の出力をする回路
  = (n本の入力信号から1本の出力をする回路)×m
- どんな関数でも
  - 真理値表として書ける
  - 論理演算で表わせる
  - 基本論理素子の組合せで作れる

真理値表

B

- 2進演算のn個の入力すべてについての出力を表に示したもの
- 例: 1ビットの加算
  - 入力は2ビット → 2^2通り
  - 出力は2桁 → 別々の表

入力1	入力2	出力
0	0	0
0	1	0
1	0	0
1	1	1

上の桁

入力1	入力2	出力
0	0	0
0	1	1
1	0	1
1	1	0

下の桁

真理値表のその他の例

x	y	AND(x, y)	OR(x, y)	NOT(x)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

論理関数

- **n入力1出力の論理関数**
  - 入力パターンは2^n通り，論理関数は2の2^n乗通り
- **完備性**
  - すべての論理関数を実現できる性質
- **完備性を備えた演算の組合せの例**
  - { AND, OR, NOT } の組合せ
  - NAND単独またはNOR単独
    - 物理的な回路を簡単化できる
    - 実際のコンピュータではこちらを使用

論理演算

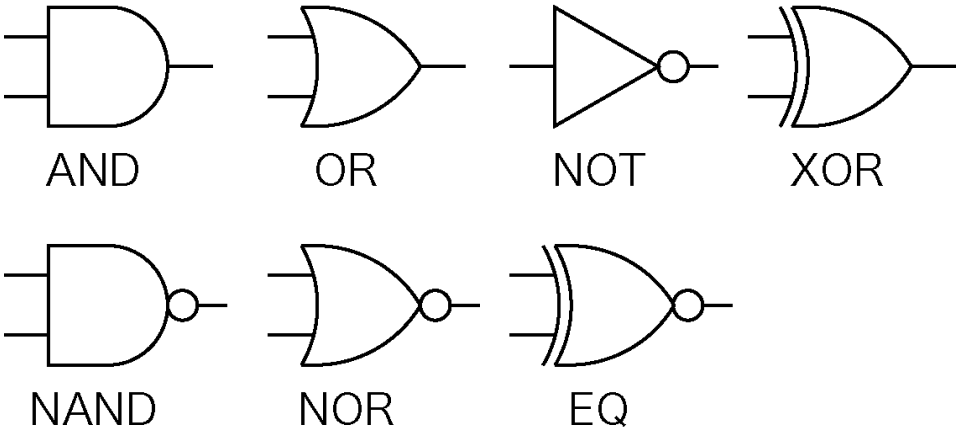
- 値: 真(1)と偽(0)
- 演算: AND, OR, NOTなど
  - AND(x,y) : xとyの両方が真のときのみ真
  - OR(x,y) : xとyのどちらかが真のとき真
  - NOT(x) : xが偽のときのみ真
  - 他にもXOR, NAND, NOR, EQなど
- 例: xとyの加算
  - 上の桁: AND(x,y)
  - 下の桁: OR(AND(x,NOT(y)), AND(NOT(x), y))

ブール代数

- 論理演算を数式として表現
  - AND: ・（論理積）
  - OR: ＋（論理和）
  - NOT: $\overline{x}$（ビット反転，論理否定）
- 演算規則に基づく式変形によって等価な演算を導ける
  - 例: $x_1 \cdot (x_1 + x_2) = x_1 + x_1 \cdot x_2 = x_1 \cdot (1 + x_2) = x_1$
- 標準形
  - いかなる演算についても唯一に定まる形式

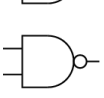
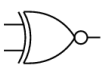
MIL記法

- 基本的な論理演算を以下の基本素子で表現



- 基本素子のあいだの結線によって論理関数を表現

論理演算の書き方

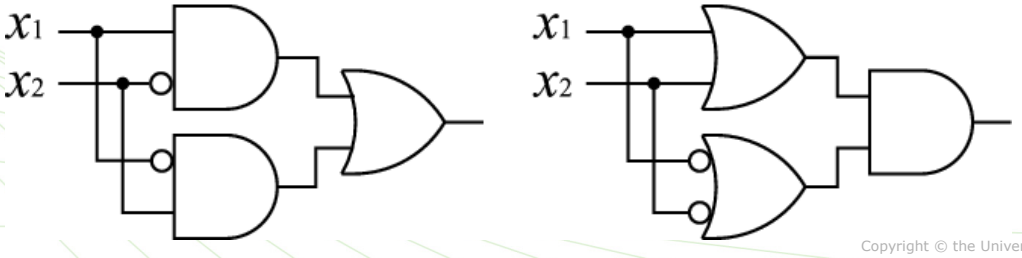
論理式	ブール代数	MIL記法
NOT(x)	$\overline{x}$	
AND(x,y)	$x \cdot y$	
NAND(x,y)	$\overline{x \cdot y}$	
OR(x,y)	$x + y$	
NOR(x,y)	$\overline{x + y}$	
XOR(x,y)	$x \oplus y$	
EQ(x,y)	$x \odot y$	

意味は同じ

XORの標準形による表現

- XOR（排他的論理和）
  - 2入力が同じ値のとき0, 異なる値のとき1
- ブール代数によるXORの表現
  - 加算標準形 $x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2$
  - 乗算標準形 $(x_1 + x_2) \cdot (\overline{x_1} + \overline{x_2})$
- MIL記法によるXORの表現

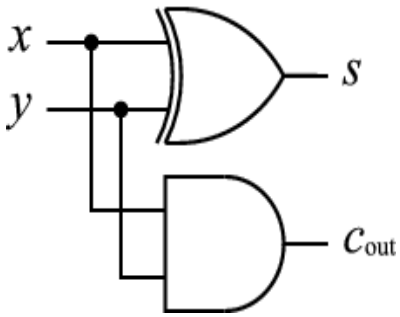
加算標準形　　乗算標準形



1ビット半加算器

- 2つの1ビット入力x, yに対して和sと桁上げc_{out}を出力

x	y	s	c_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



足し算回路(半加算器)

x	y	c_{out}
0	0	0
0	1	0
1	0	0
1	1	1

上の桁

x	y	s
0	0	0
0	1	1
1	0	1
1	1	0

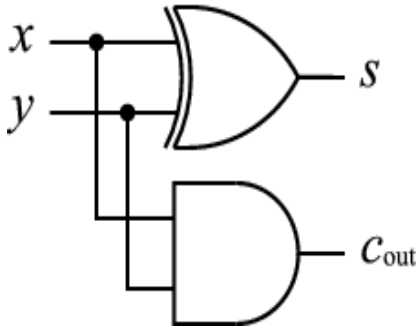
下の桁

- 入力: x, y
- 出力: s(和), c_{out}(桁上げ)
- ↘ 真理値表
- ↘ ブール代数・論理式

$$s = x \oplus y = \mathrm{XOR}(x, y)$$

$$c_{\mathrm{out}} = x \cdot y = \mathrm{AND}(x, y)$$

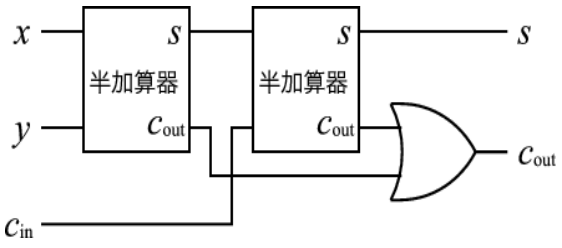
- ↘ MIL記法



1ビット全加算器

- ↘ 桁上げ入力c_{in}も考慮
- ↘ 半加算器2つとORの組合せとして実現できる
  - 半加算器をモジュールとして使用

x	y	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



足し算回路(全加算器)

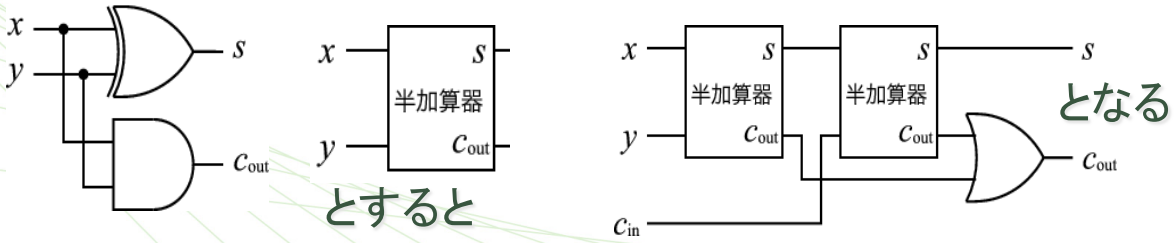
入力: x, y, c_{in}(下からの桁上げを考慮)

出力: s, c_{out} (桁上げ)　$s = x \oplus y \oplus c_{\text{in}}$

↘ 論理式だと少々複雑　$c_{\text{out}} = x \cdot y + (x \oplus y) \cdot c_{\text{in}}$

↘ 半加算器の組合せでできる

- s: (xとyの和)とc_{in}の和
- c_{out}: (xとyの桁上げ)と　の桁上げのOR



オペレーティングシステム(OS)

↘ オペレーティングシステム(OS)の特徴

- 基本ソフトウェアとも呼ばれる
- コンピュータの起動と同時に実行される
- 複数の人間によるコンピュータの効率的な利用をサポートする
- コンピュータの資源を管理する

↘ OSとアプリケーション

- OSの例:Mac OS X, Windows XP
- アプリケーションの例:Safari(ウェブブラウザ), Excel(表計算)

OSの基本機能

- **プロセス管理**
  - 起動，終了，CPUによる実行時間の割当て
- **メモリ管理**
  - プロセスへのメモリ割当て，解放
- **入出力管理**
  - プロセスと周辺装置の仲介
- **ファイル管理**
  - ファイルの作成，保護，消去と階層的な管理機構
- **ユーザ管理**
  - プロセスやファイルを利用者ごとに一定の制限のもとで保護