

# cs-5. コンピュータの構成要素と データ処理の仕組み

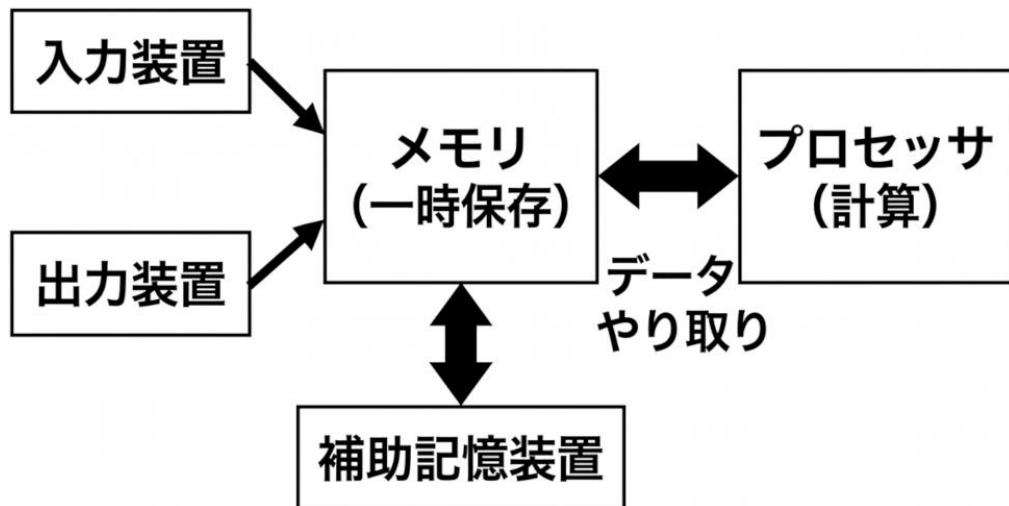
(コンピューターサイエンス)

金子邦彦



謝辞：この資料では「いらすとや」のイラストを使用している

# 内容



文字 二進数 (8ビット) 16進数

'A' 01000001 41

'B' 01000010 42

文字はASCIIなどで数値へ

アドレス データ (1バイト=8ビット)

0 01000001

1 01000010

2 01000011

...

... 01000000

論理演算による算術演算



負の整数の表現 (2の補数)

最上位ビットで符号を表現

[1]0000011 → 負の整数

# 今回の内容と、今までの内容の関係



今まで：プログラミング、AI、画像処理、GCとの関連

今回：コンピュータ内部のデータ表現と処理の仕組み

## データの入力と保存

## デジタルの基礎： 二進数

## データ演算の仕組み

あらゆるデータ

数字  
文字  
画像  
グラフィックス

メモリ (記憶装置)

デジタルデータ  
として保存

0	1	0	1
1	0	1	0
0	1	0	1
0	0	1	1
1	1	0	1

すべてのデータを  
0と1で表現

例：10進数「2」  
→  
→ 二進数「10」

論理演算  
AND,  
OR,  
NOT

足し算・  
掛け算の  
基礎

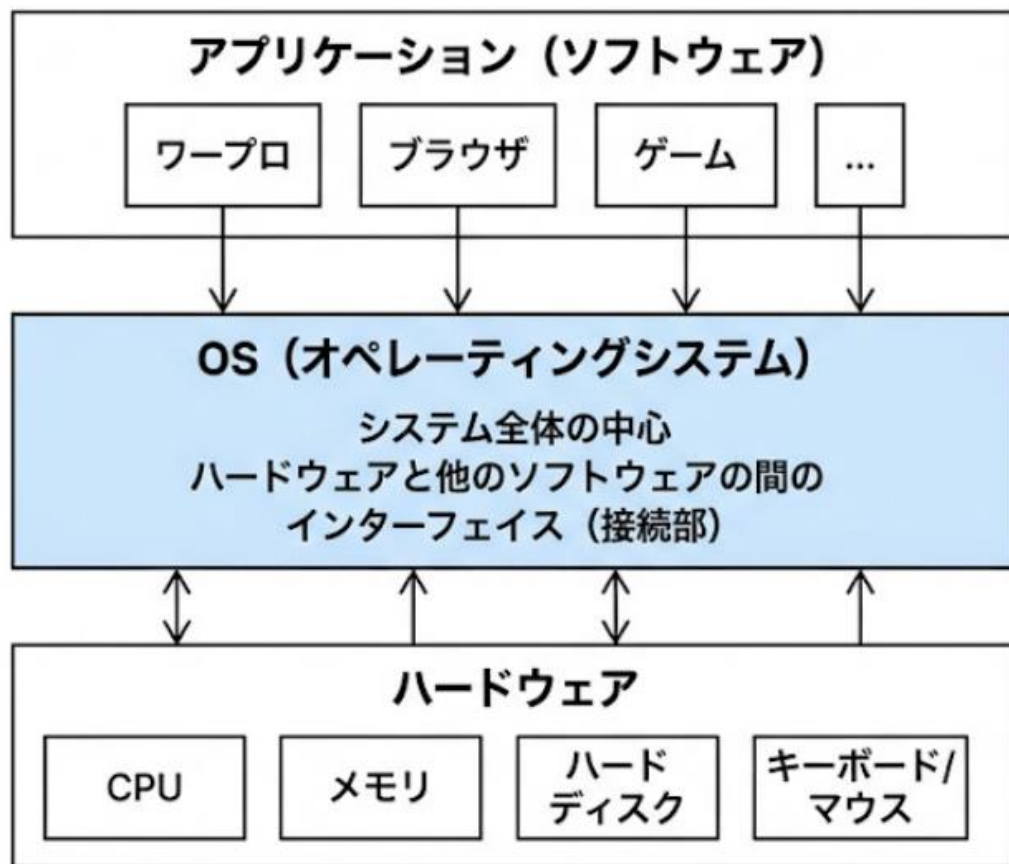
プログラ  
ミングの  
条件分岐

# 5-1 OS

# OS (オペレーティングシステム)



## OSはコンピュータシステムの中心

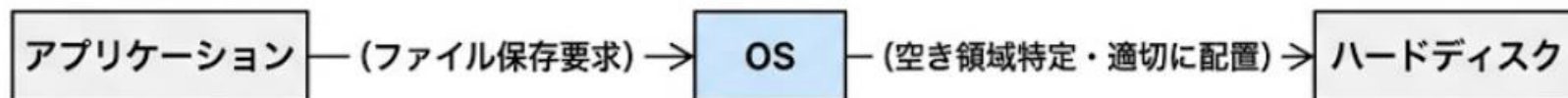


### 特徴・メリット

「統一的な利用方法」を提供。異なるハードウェアでも同じ操作感。

各アプリケーションがハードウェアの「詳細な制御」を不要に。開発を簡素化。

動作例：アプリケーションによるファイルの保存



# OSの特徴



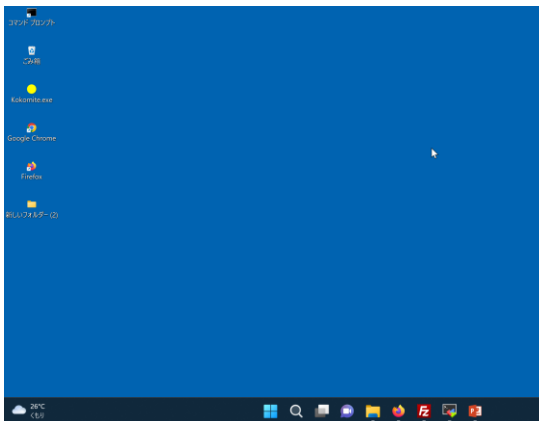
- **種類の多様性** : Windows, Linux (AndroidもLinuxの一種) , macOSなど.
- **コマンドラインインターフェイス** : 文字による命令での操作

A screenshot of a Windows Command Prompt window. The title bar reads "コマンド プロンプト". The text inside the window shows the Windows version and copyright information: "Microsoft Windows [Version 10.0.23493.1000] (c) Microsoft Corporation. All rights reserved." followed by the current directory path "C:\Users\user>".

```
コマンド プロンプト
Microsoft Windows [Version 10.0.23493.1000]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>
```

- **グラフィカルユーザーインターフェイス** : 画面、メニュー、アイコンによるビジュアルな操作



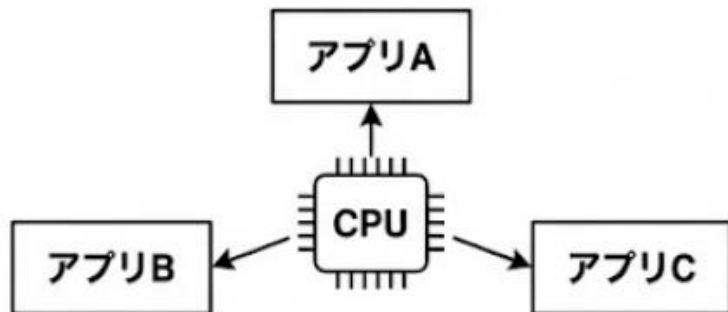
ソフトの起動や終了, ファイルの操作, ソフトのインストールなど

# OSの主な機能



## マルチタスク

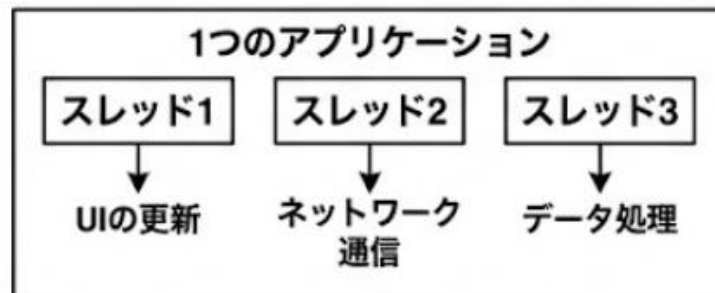
複数のアプリケーションを同時に実行



各アプリにCPU時間を割り当てる

## マルチスレッド

1つのアプリケーション内で、複数の処理（スレッド）を並行実行

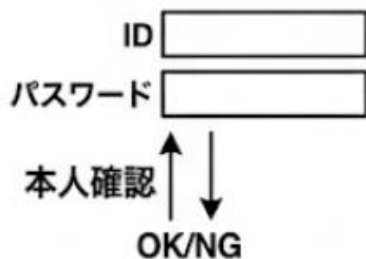


スレッド単位の並行処理

## セキュリティ

ユーザー認証とアクセス制御

ユーザー認証



アクセス制御

	権限管理	
	ファイルA	ファイルB
ユーザーX	読み込み	書き込み
ユーザーY	読み込み	書き込み

システムとデータの保護

## ネットワーク

コンピュータをネットワークに接続し、データを送受信



データの送受信

外部との通信

# 5-2コンピュータの構成

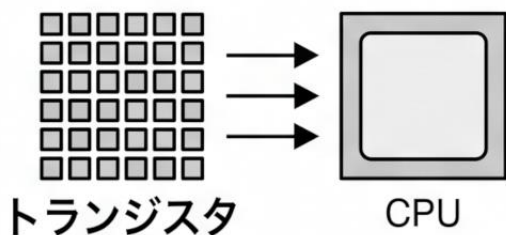
# コンピュータの主要要素



## プロセッサ (CPU)

役割：すべての計算とデータ処理を行う「脳」

構造：数億個以上のトランジスタから成る電子回路

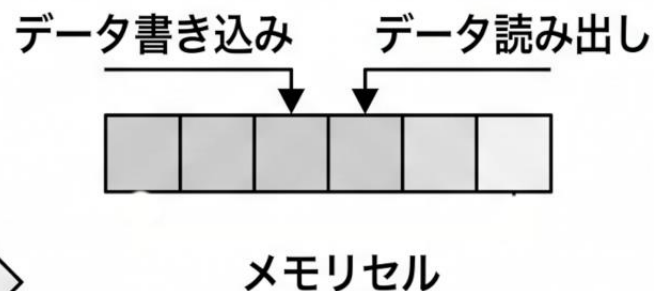


動作原理：論理演算 (AND, OR, NOT) の組み合わせによる計算



## メモリ (RAM)

役割：データやプログラムの保存（「作業台」）



性質：高速だが、電源が切れるとデータが消失する

電源ON時



電源OFF時

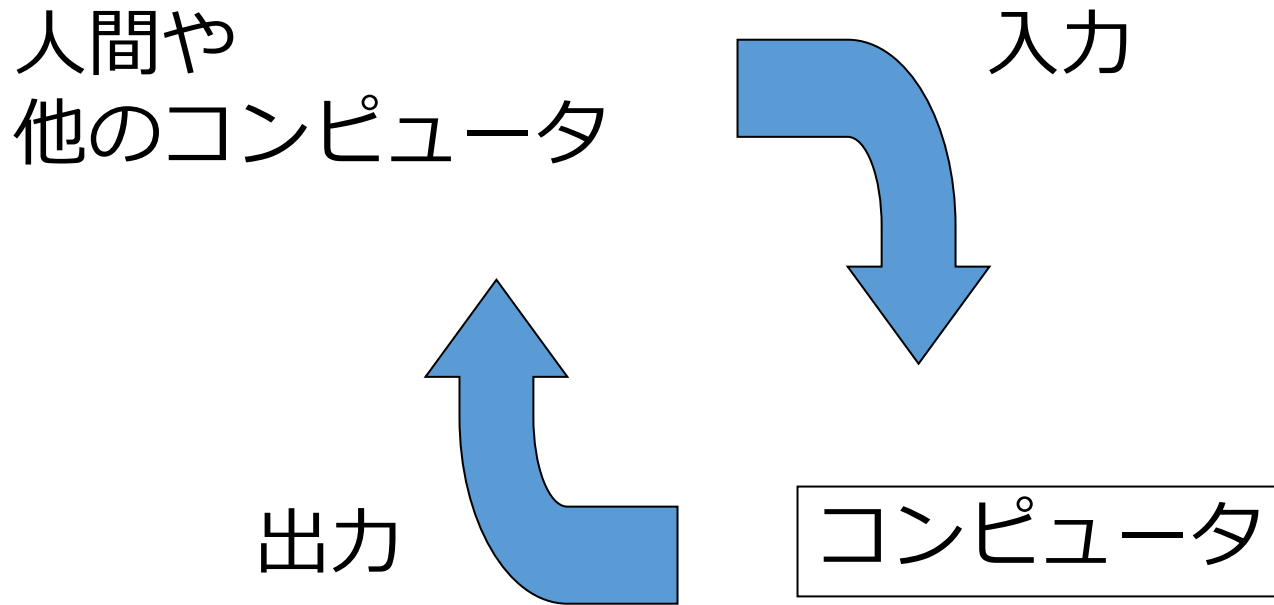


プロセッサからの要求  
(読み出し/書き込み)

データ・プログラムのやり取り

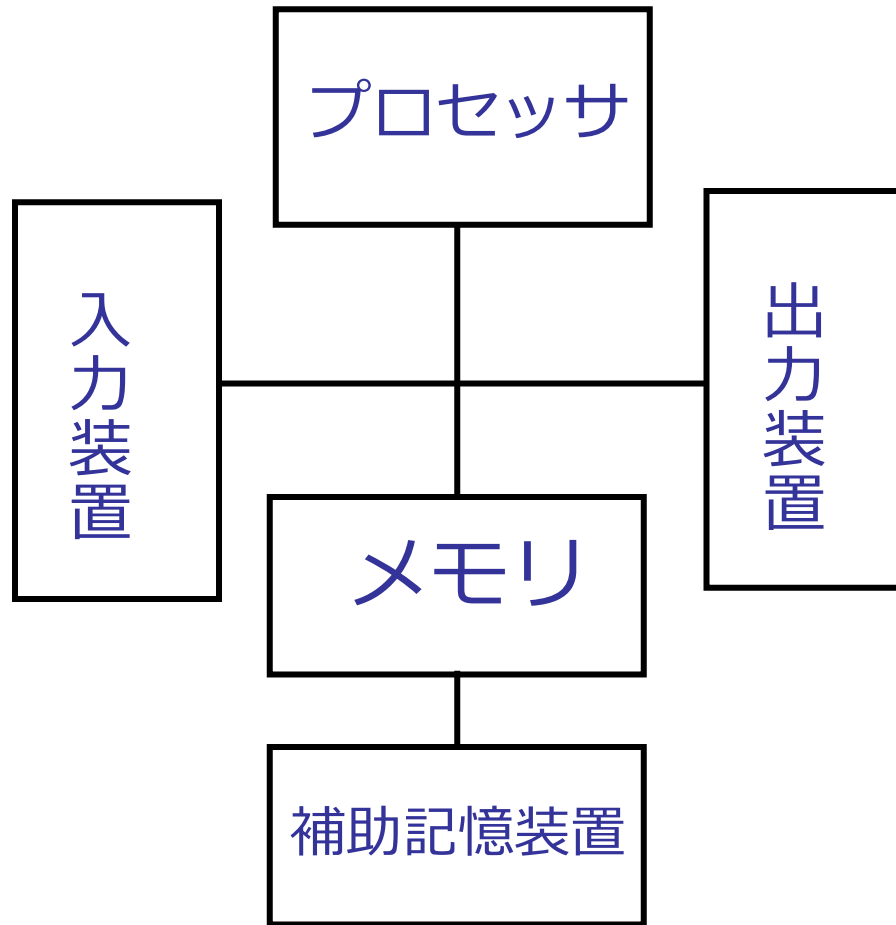
メモリからの提供  
(データ/プログラム)

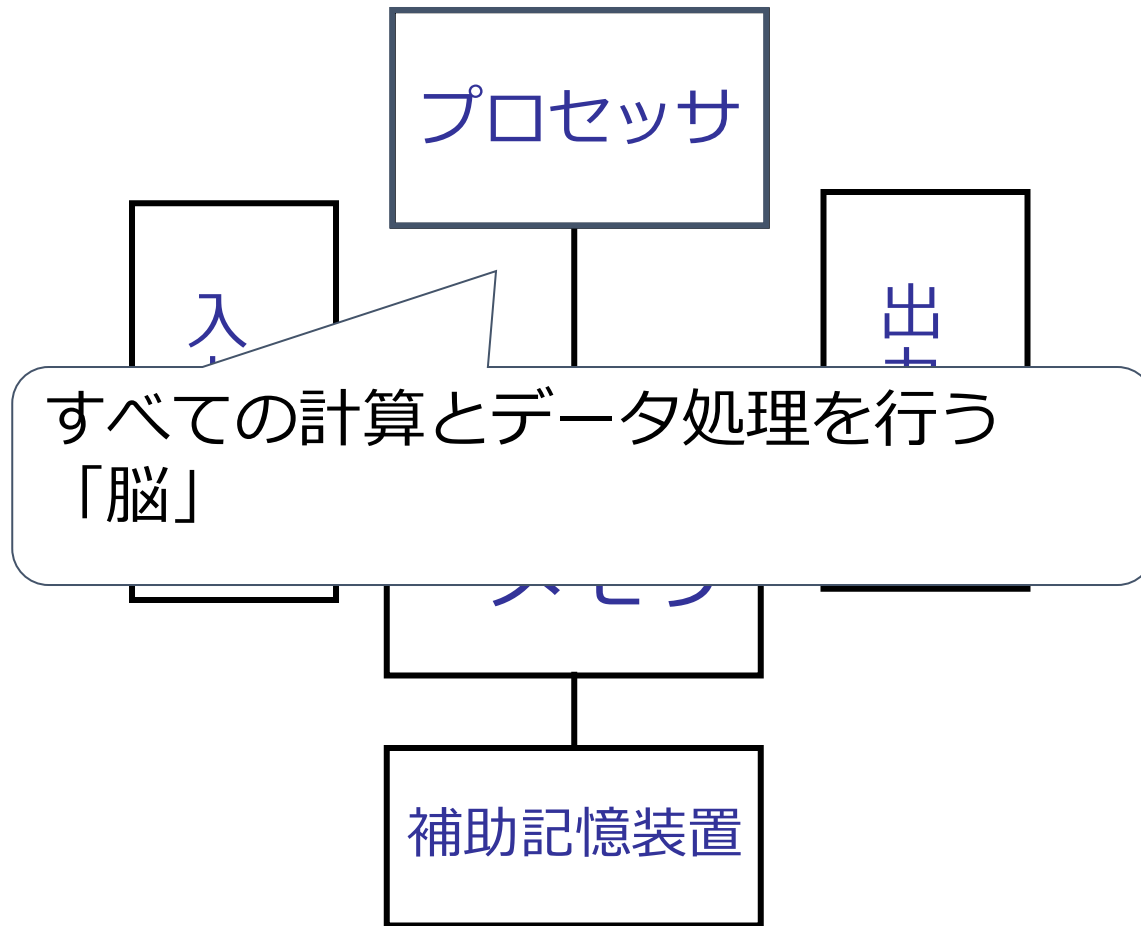
# コンピュータ全体構成

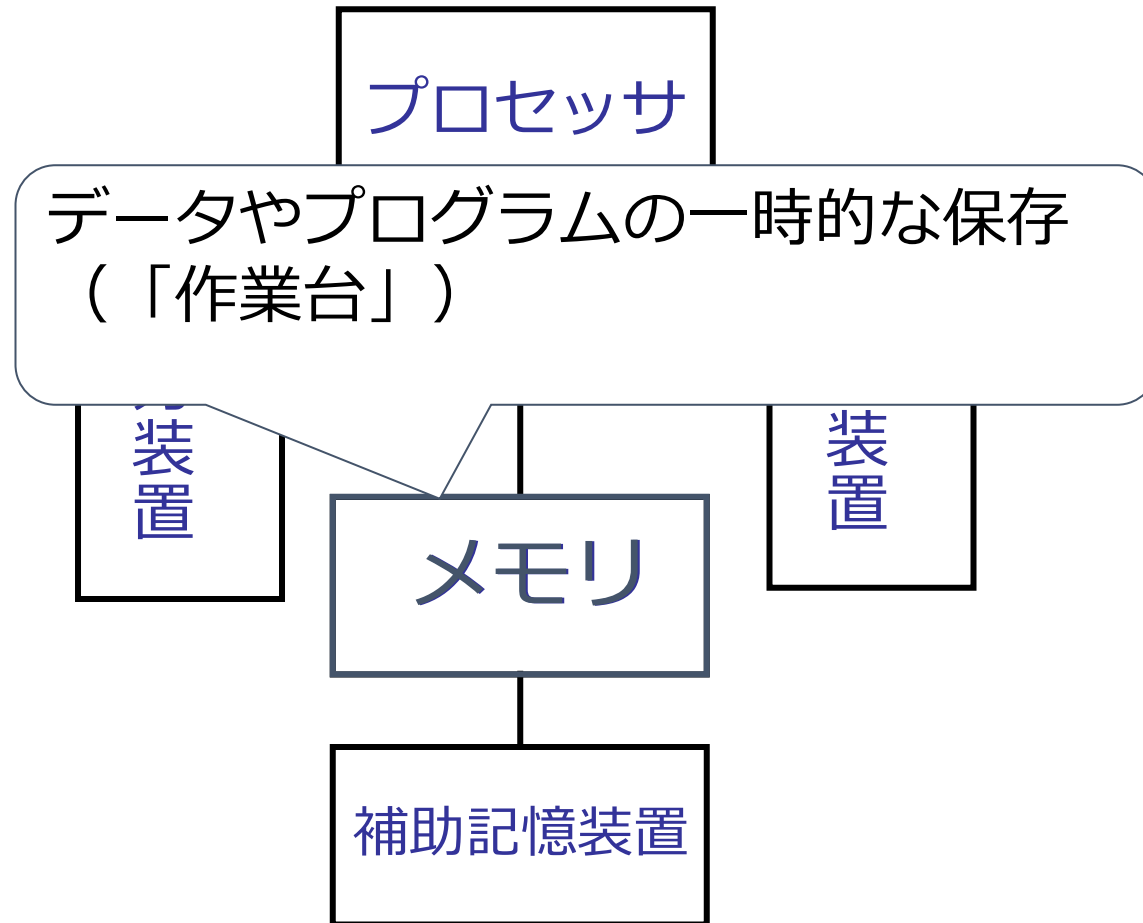


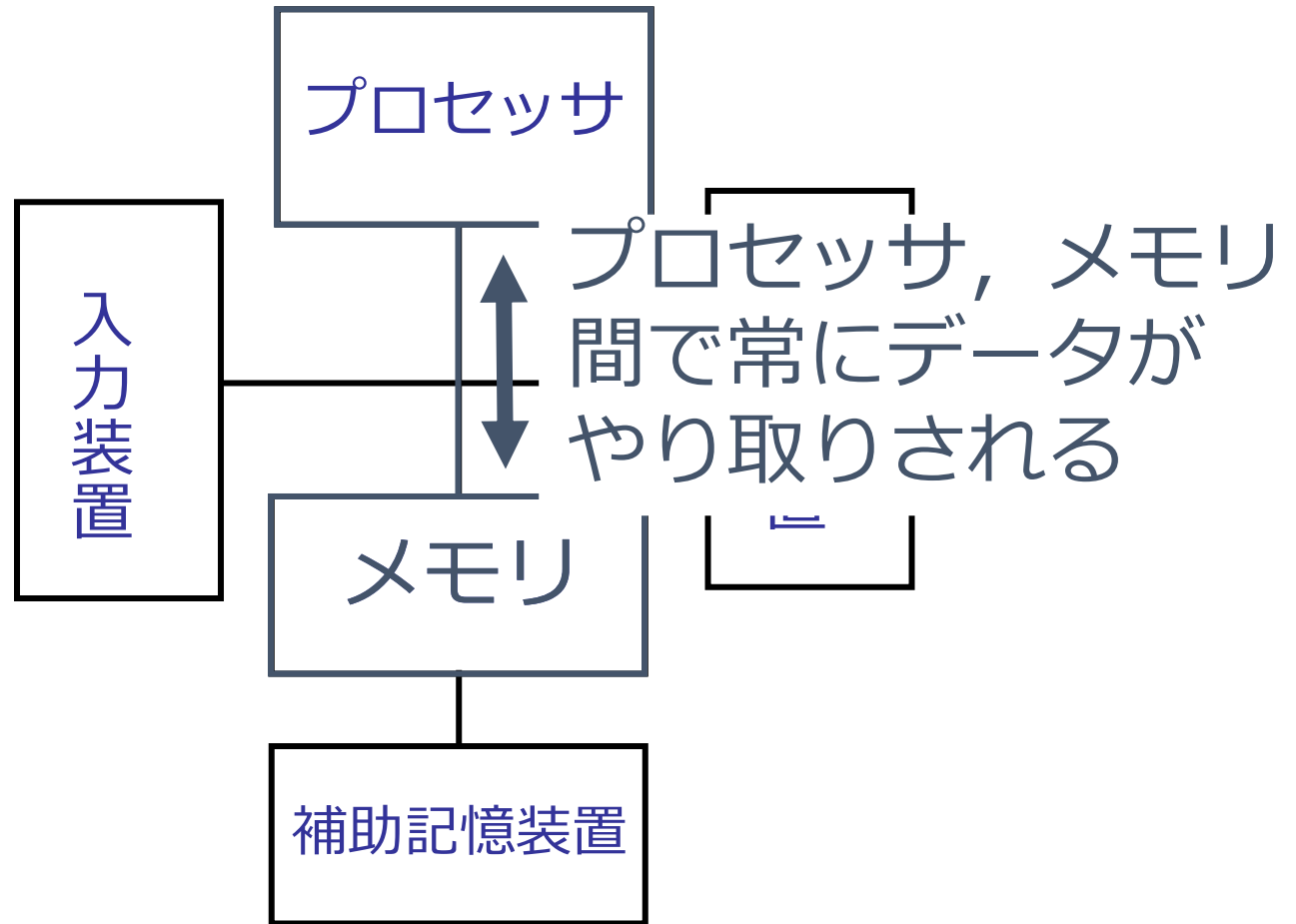
5つの構成要素：プロセッサ、メモリ、入力装置、出力装置、補助記憶装置（電源切断後もデータを保持する装置）

# コンピュータの構成









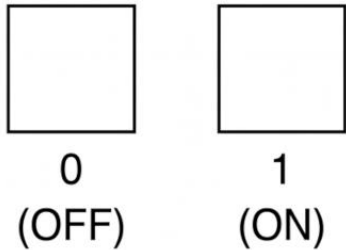
## 5-3 デジタル、二進数と16進数

# ビット、ビット列、デジタル



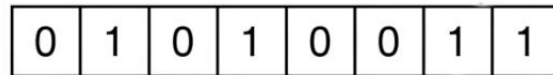
## 情報の最小単位： ビット (Bit)

二進数の一桁。  
0か1の状態を持つ。



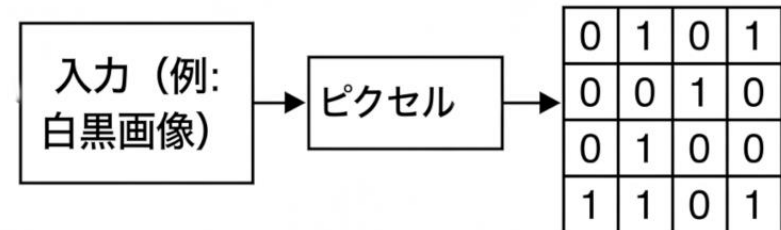
## ビット列による 情報表現

0と1の組み合わせで  
多様なデータを管理。



## デジタル変換

情報をデジタルデータへ変換。



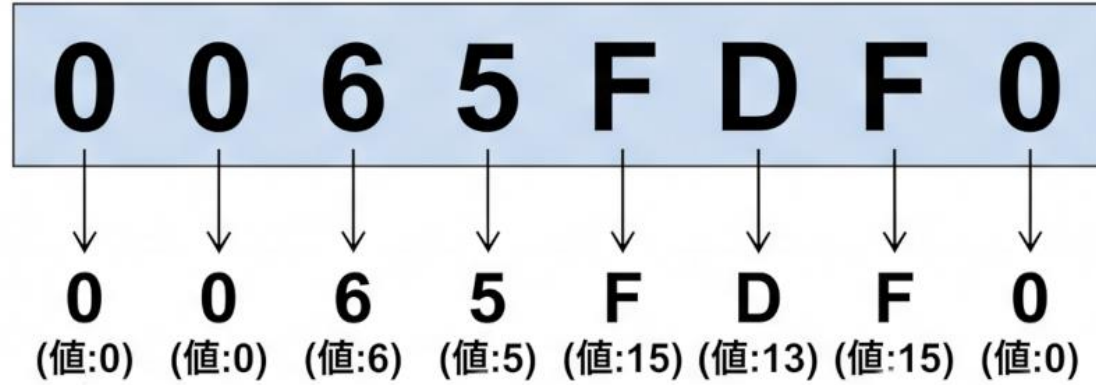
# 十六進数



## 基本構成

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
A  
B  
C  
D  
E  
F

➔ A~F は  
10~15 に対応



## 応用例



# 二進数と十六進数の対応

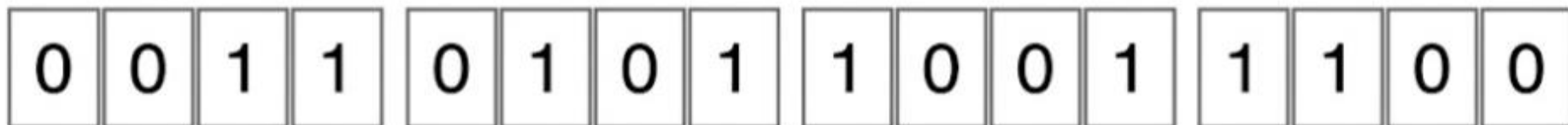


## 基本概念

二進数 4桁 = 十六進数 1桁 → 理由： $16 = 2^4$  (16は2の4乗)

## 具体的な変換プロセス (16桁)

入力 (二進数)



# 二進数と十六進数の関係



0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

## 5-4 メモリとアドレス

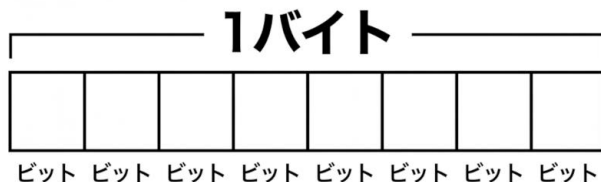
# メモリとアドレス (番地)



00	00
01	1A
02	FF
03	C4
04	D8
05	F2
06	C9

アドレス データ

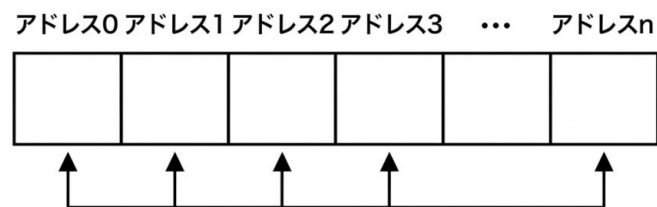
メモリの基本単位：バイト



16進数2桁

1バイト=8ビット

メモリのアドレス (番地)

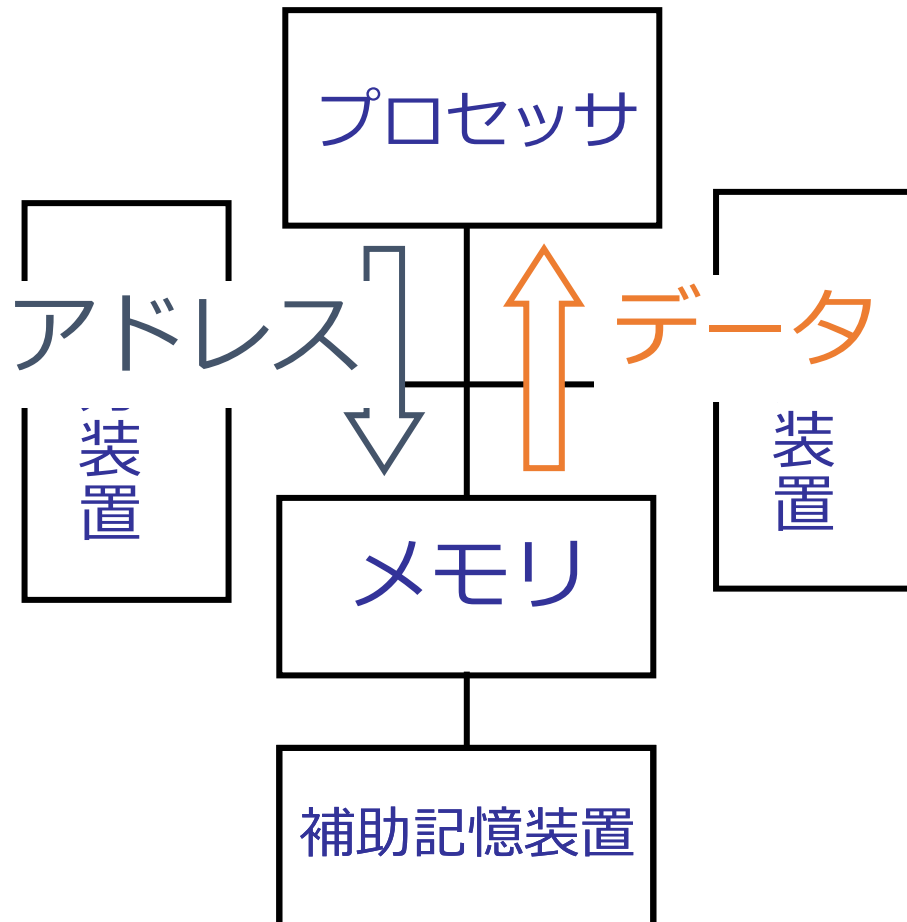


各バイトに付く通し番号

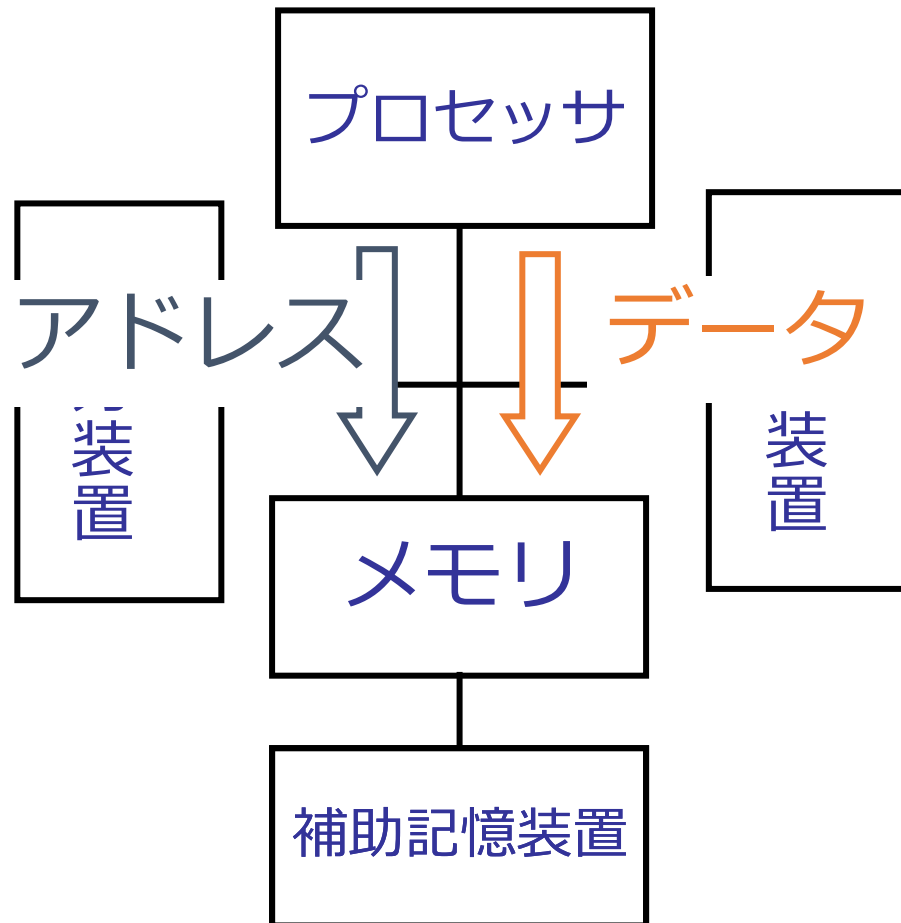
0から始まる

(住所のようなもの)

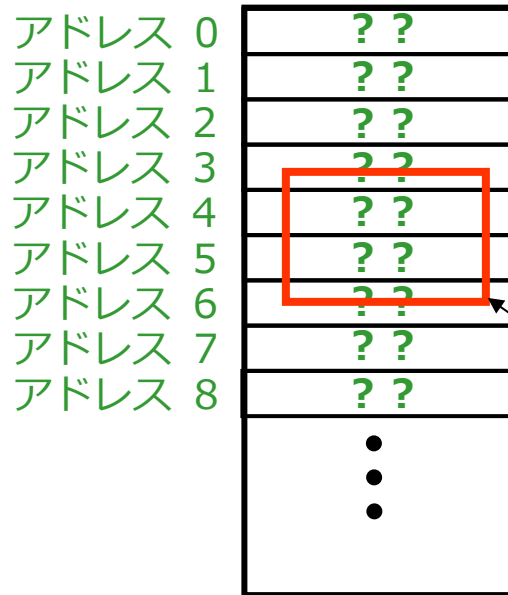
**読み出し**：アドレスを指定するとそのデータが取り出される（メモリの値は変化しない）



**書き込み**：アドレスとデータを指定するとそのデータが保存される（前の値は上書きされる）



# 読み出し

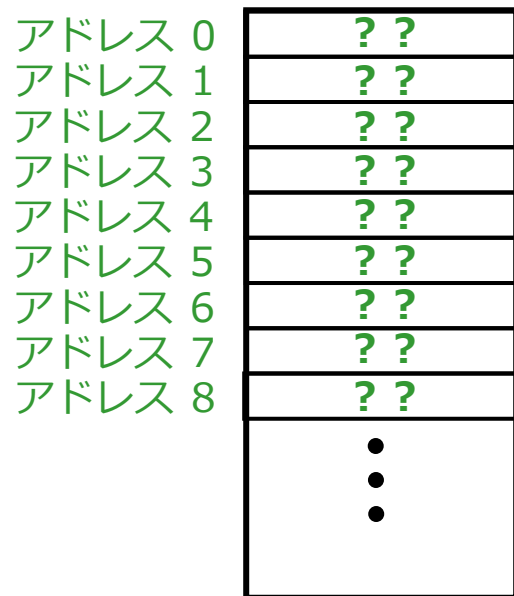


アドレス 4 番地, 5 番地  
から 2 バイト分  
読み出すとき

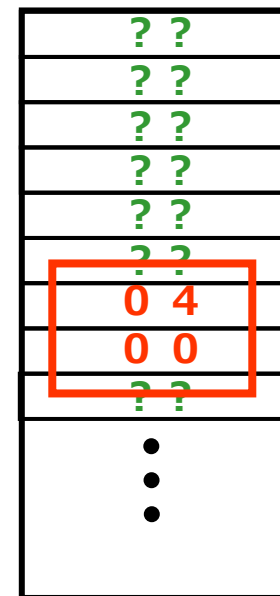
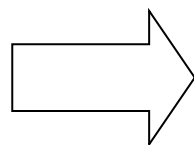
メモリの値は変化  
しない

メモリの各区画は 1 バイト  
(16 進数で 2 桁)

# 書き込み



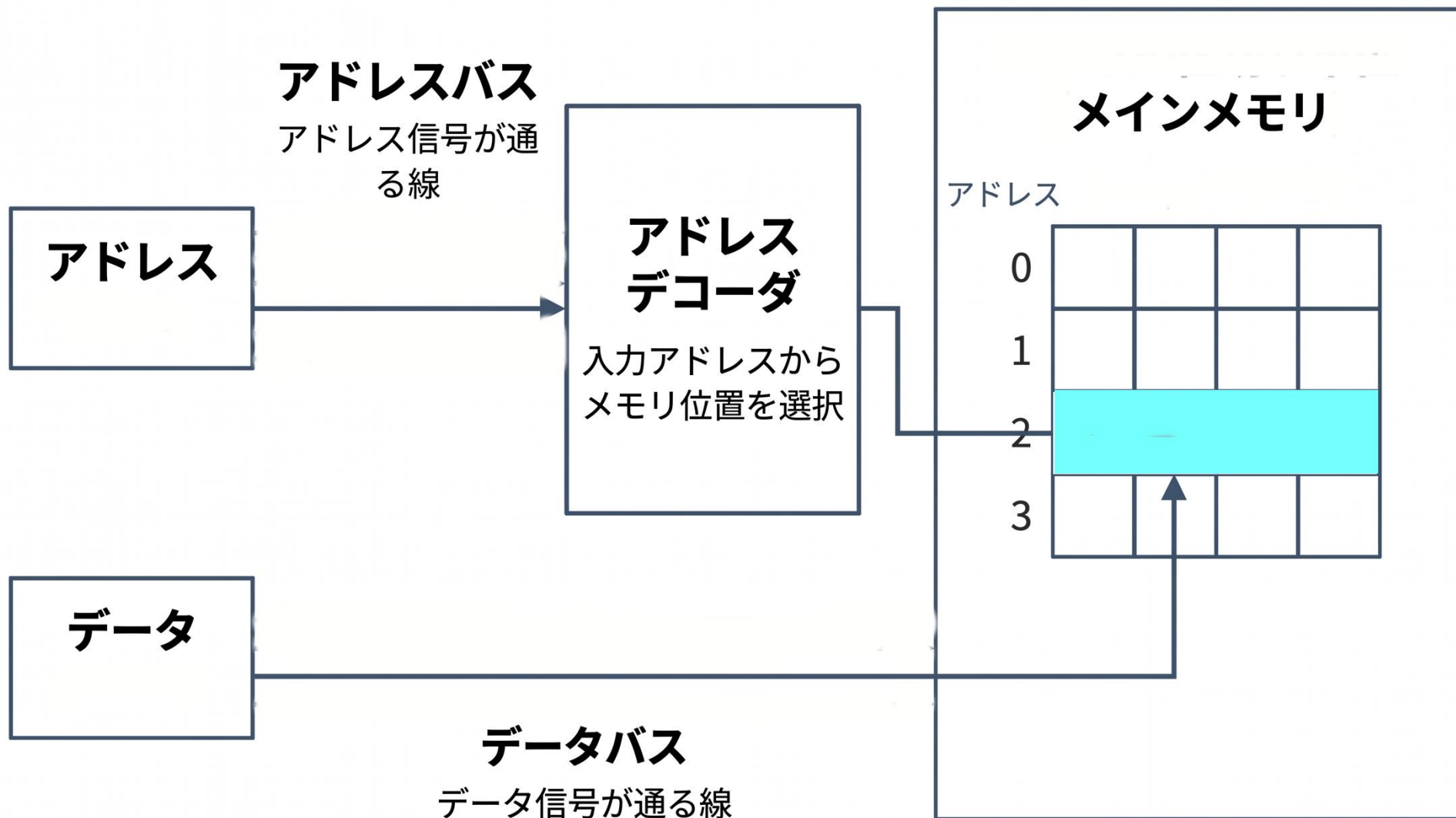
アドレス 6 番地,  
7 番地に  
「0400」を  
書き込むと



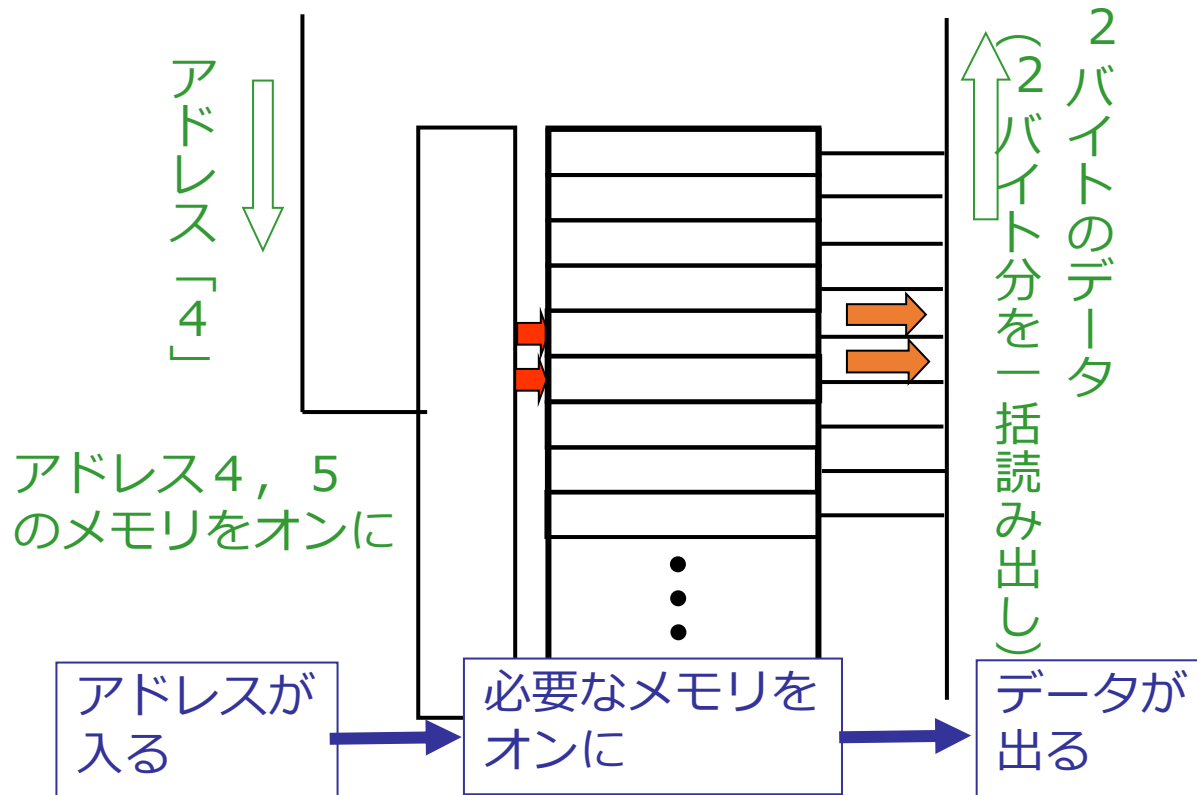
メモリの各区画は1バイト  
(16進数で2桁)

前の値は上書きされる

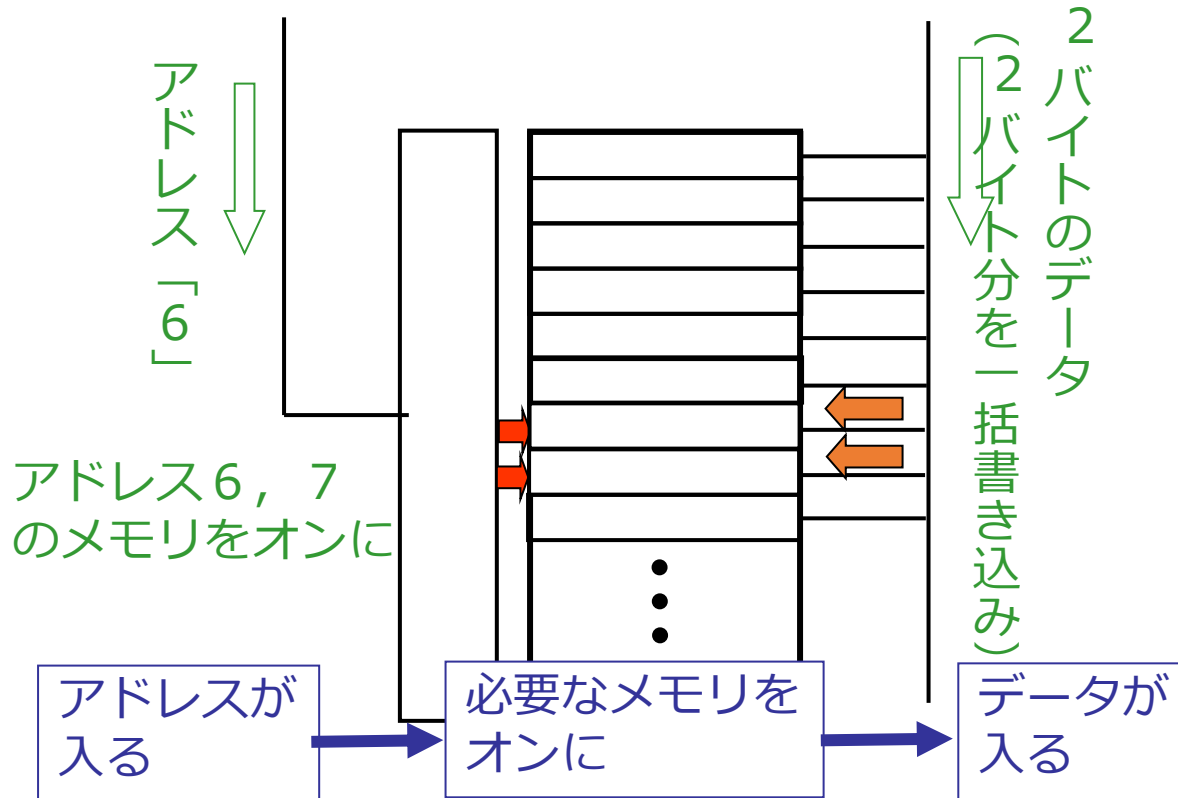
# メモリの仕組み



# アドレス4, 5番地から, 2バイト分読み出す



# アドレス6, 7番地に, 2バイト分書き込む



## 5-5 文字コード



## ASCIIの主な特徴

- 1文字を7ビットで表現  
7ビット =  $2^7 = 128$ 通りの状態
- 表現可能な128種類の文字

**英数字**  
A, B, C...  
Z, a, b, c...  
z, 0, 1, 2... 9

**記号**  
#, \$, %, &, @,  
+, \*, /...

**制御文字**  
改行, バックス  
ペース, タブ...  
表示や制御に利用

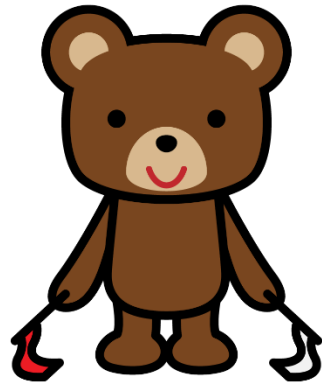
# ASCII文字コード表



	0	1	2	3	4	5	6	7
0	NULL	DEL	SP	0	@	P		p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	(BS)	CAN	(	8	H	X	h	x
9	(HT)	EM	)	9	I	Y	i	y
A	(LF)	SUB	*	:	J	Z	j	z
B	(VT)	ESC	+	;	K	[	k	{
C	(FF)	(FS)	,	<	L	¥	l	
D	(CR)	(GS)	-	=	M	]	m	}
E	SO	(RS)	.	>	N	^	n	~
F	SI	(US)	/	?	O	_	o	DEL

## 5-6 論理積と論理和

# 二進数は0または1



右手が下がっている

右手が上がっている

二通り

# 二進数は0または1



右手が下がっている



0



右手が上がっている



1

※0と1が逆になる場合もある

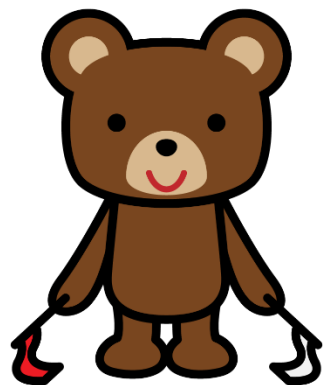
# 2変数の組み合わせ：4通り（0と0、1と0、0と1、1と1）



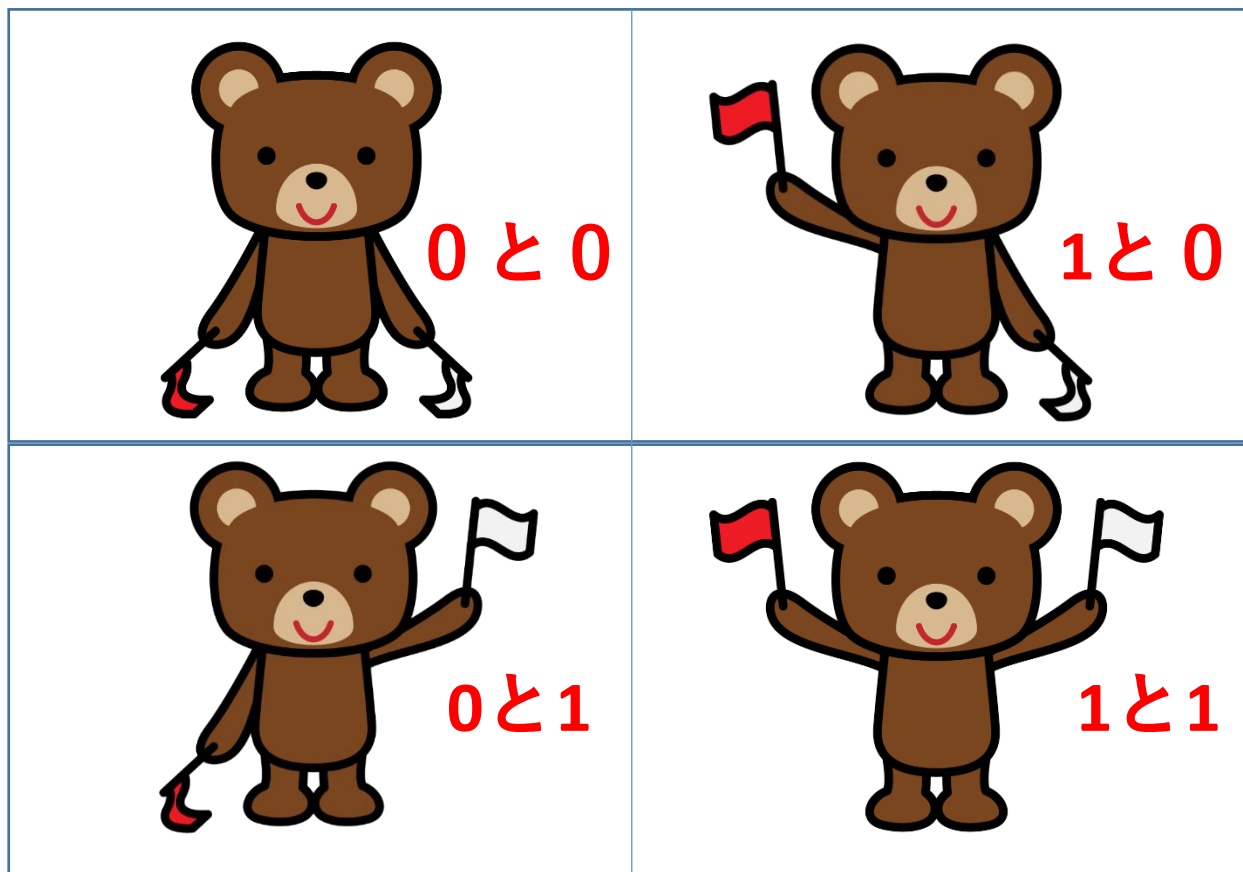
右手と左手の  
両方を考えると



4通り

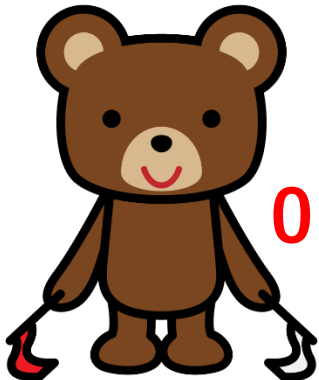





# 2変数の組み合わせ：4通り（0と0、1と0、0と1、1と1）



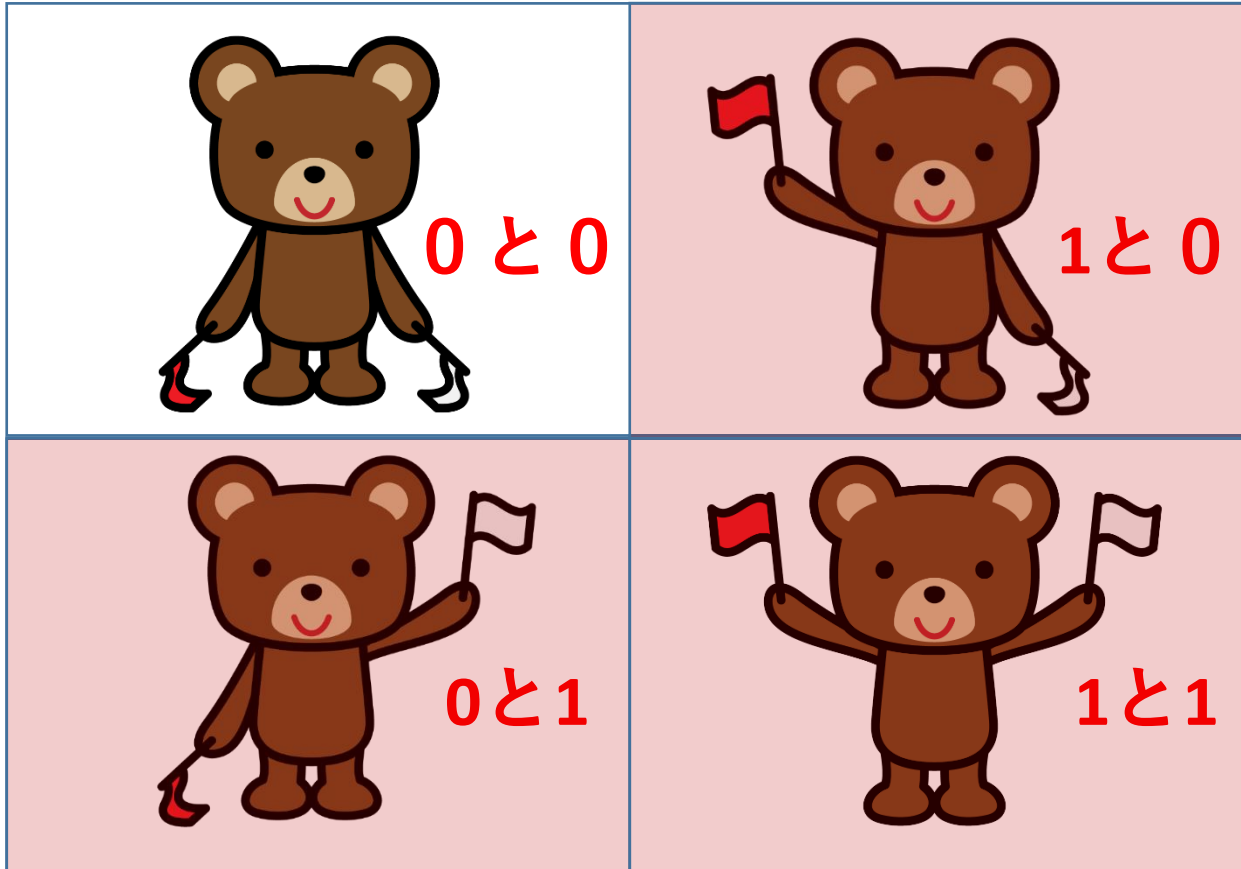
# 論理積 (AND)

両方が1のときのみ結果が1となる演算

 <p>0と0</p>	 <p>1と0</p>
 <p>0と1</p>	 <p>1と1</p>

# 論理和 (OR)

少なくとも片方が1のとき結果が1となる演算



# 論理和と「選択」の違い

論理和では「両方が1」のときも結果が1となるため、  
どちらか一方だけを選ぶ意味の「選択」とは異なる



- ・ 焼き芋大会があるんだけど、
- ・ 土曜日と日曜日、どっちが良い？




両方、申し込んでよ♡



日曜日  
落選：0 当選：1

土曜日  
落選：0

当選：1

**両方参加してもOK!**

土曜日と日曜日の選択では無  
い

# 論理積と論理和のまとめ



	0	1
0	0	0
1	0	1

論理積AND

	0	1
0	0	1
1	1	1

論理和OR

# 複数ビットの一括論理演算



複数桁の二進数に対して、桁ごとに論理積や論理和を求める処理

$x$     0 0 1 1

$y$     0 1 0 1



全部で4ビット

クイズ

$x$ と $y$ の論理積は？  
論理和は？

# 複数ビットの一括論理演算



複数桁の二進数に対して、桁ごとに論理積や論理和を求める処理

$x$	0	0	1	1	0	0	1	1
$y$	0	1	0	1	0	1	0	1
	0	0	0	1	0	1	1	1

論理積AND

論理和OR

# 多数決



- 過半数が1なら1、過半数が0なら0となる演算
- **論理積 (AND) と論理和 (OR) の組み合わせで実現できる**

A	0	1	0	1	0	1	0	1	A AND B	0	0	0	1	0	0	0	1	①
B	0	0	1	1	0	0	1	1	B AND C	0	0	0	0	0	0	1	1	②
C	0	0	0	0	1	1	1	1	C AND A	0	0	0	0	1	0	1		③
多数決	0	0	0	1	0	1	1	1	①OR②	0	0	0	1	0	0	1	1	④
									③OR④	0	0	0	1	0	1	1	1	

## 5-7 論理演算と足し算

# 論理演算の三要素



## 論理積 AND

	0	1
0	0	0
1	0	1

## 論理和 OR

	0	1
0	0	1
1	1	1

## 否定 NOT

入力を反転する演算

$0 \Rightarrow 1$

$1 \Rightarrow 0$

# 二進数の足し算



**1桁同士の足し算では、結果を2ビット（上位ビットと下位ビット）で考える**

$$0 + 0 = 00$$

$$0 + 1 = 01$$

$$1 + 0 = 01$$

$$1 + 1 = 10$$

上位ビット 下位ビット

# 1桁足し算の2ビット表現と、論理演算との関係



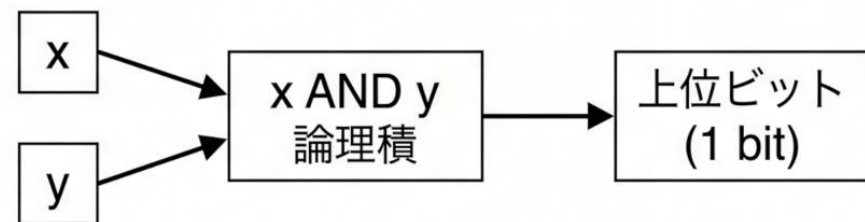
## 基本概念と演算結果

入力 (x, y)	加算式	2ビット 結果	上位 ビット	下位 ビット
0, 0	0 + 0	00	0	0
0, 1	0 + 1	01	0	1
1, 0	1 + 0	01	0	1
1, 1	1 + 1	10	1	0

結果を2ビットとして扱う（上位・下位に分離）

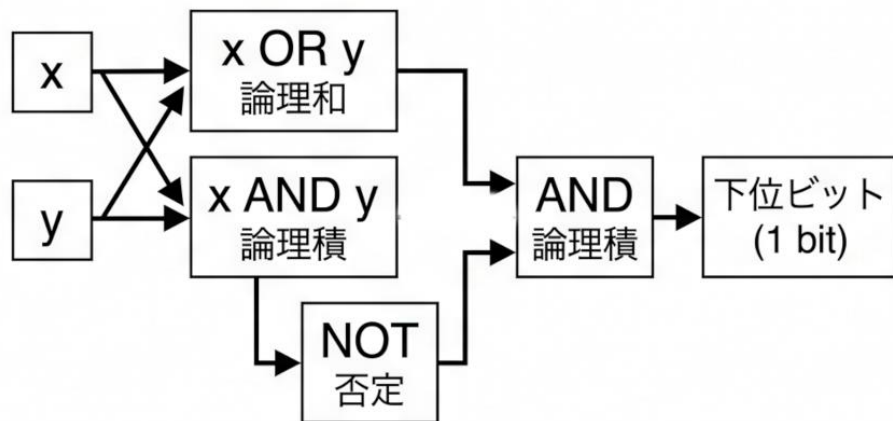
## ビット算出の論理構成

### 上位ビットの算出方法



結果の上位ビット：x AND y で求められる

### 下位ビットの算出方法

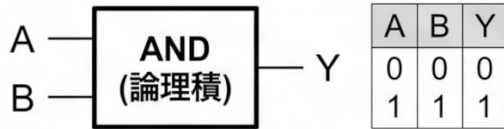


結果の下位ビット：  
AND((x OR y), NOT(x AND y)) で求められる

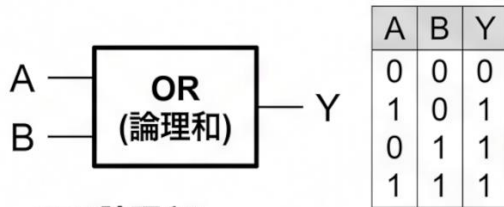
# 論理演算の組み合わせによる算術演算の実現



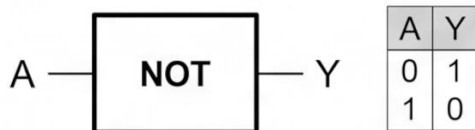
## 基本論理演算



AND (論理積):  
両方が1の場合のみ、出力1

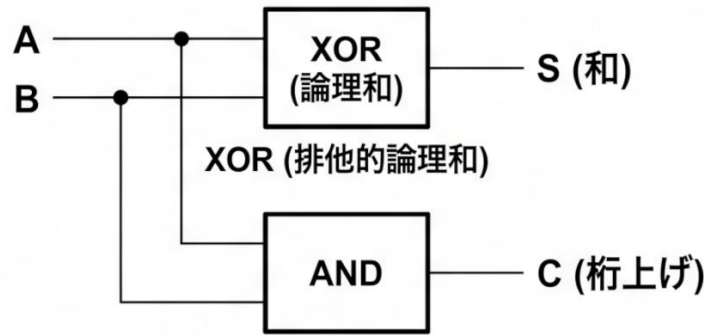


OR (論理和):  
どちらか一方が1の場合、出力1



NOT (否定): 入力を反転させる

## 算術演算の実装例: 1ビット足し算



\*XORはAND, OR, NOTで構成可能  
(A AND NOT B) OR (NOT A AND B)

## コンピュータ内部での 処理階層

算術演算  
(例: 足し算, 引き算)

↓  
論理回路

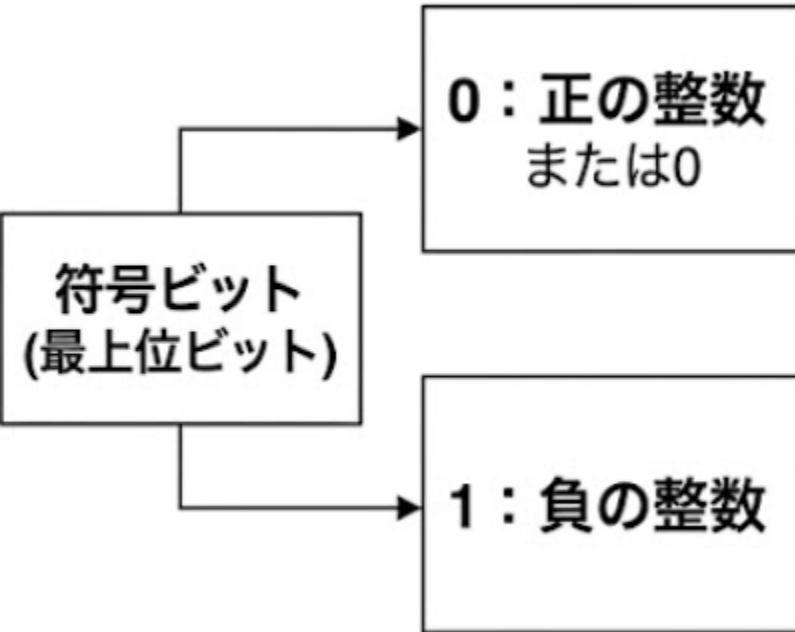
基本論理演算の組み合わせ  
(AND, OR, NOT)  
トランジスタなどの電子回路で実装

**コンピュータは、論理演算を行う電子回路（論理回路）の組み合わせで、複雑な算術演算を成立させている。**

# 5-8 2 の補数

# 2の補数

## 符号ビット



## 2の補数

正の数 (例: +5)

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

負の数 (例: -5)

1	1	1	1	1	0	1	1
---	---	---	---	---	---	---	---



# 負数 -45 と正数 45 の加算での 2 の補数の確認



## 【ステップ1】 数値を 8ビット2進数へ変換

**+45**

10進数: 45



**-45**

手順1: +45の2進数

00101101
----------

全ビット反転

11010010
----------

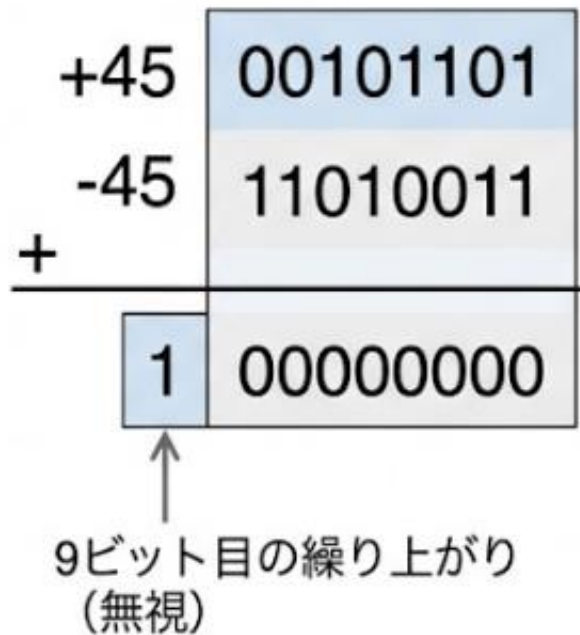
1を加算

+1
----

2の補数(-45):

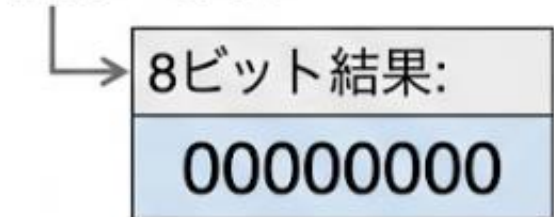
11010011
----------

## 【ステップ2】 2の補数による8ビット加算



## 【ステップ3】 結果の解釈と結論

### 結果の確認



10進数: 0

### 結論

2の補数表現における  
正負の加算が成立

# 演習



# 演習①



## 環境

- Python : プログラミング言語
- trinket : Web上でPythonプログラムを実行できる学習環境

## 文字コード関連の関数

- **ord関数** : **文字を文字コード（整数）に変換する**
- **hex関数** : **整数を16進数表記の文字列に変換する**

## 16進数の表記

- 接頭辞「**0x**」 : 先頭に付けると、その数値が**16進数である**  
**ことを示す目印**になる

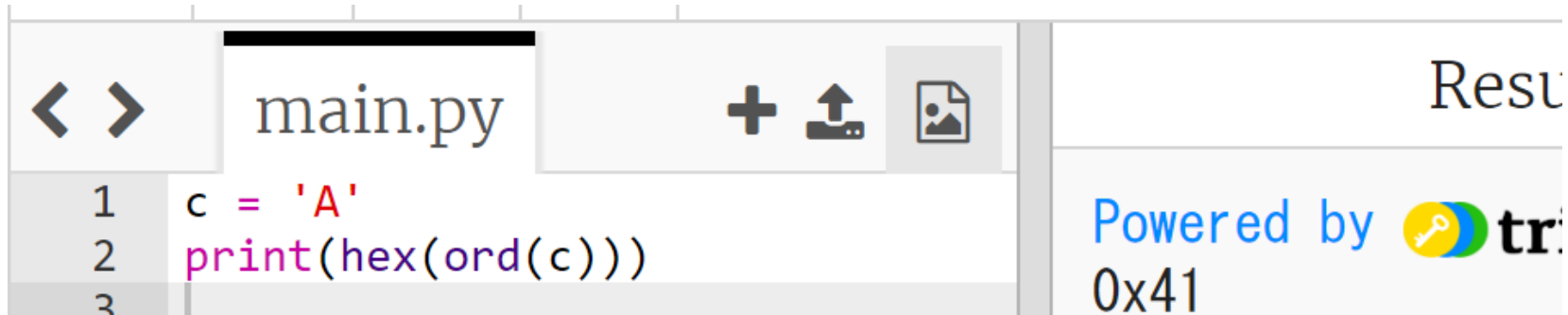
## 前提となる概念

- ASCII
- 16進数


①trinketの次のページを開く

<https://trinket.io/python/595c091dd9>

② このプログラムは、「A」の文字コードを表示する。なお「0x」は、16進数を示す目印である。実行結果が、次のように表示されることを確認



```
< > main.py + ↑ 📄  
1 c = 'A'  
2 print(hex(ord(c)))  
3
```

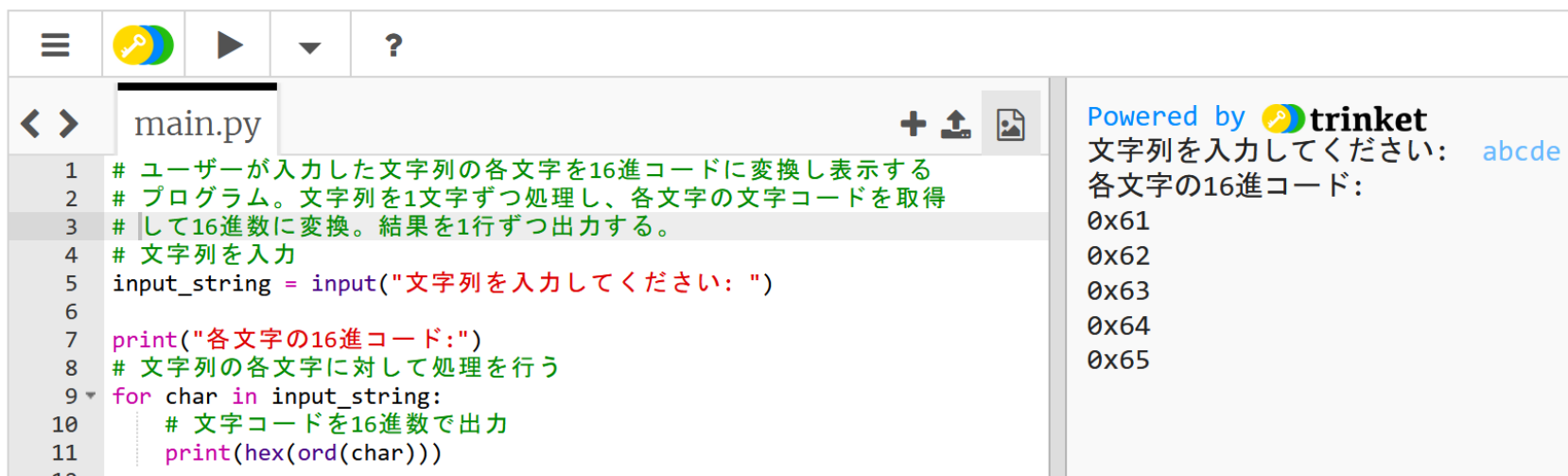
Result  
Powered by  tr  
0x41

### ③trinketの次のページを開く


<https://trinket.io/python/17fb2ed2d5>

④ このプログラムは、あなたが入力した文字や記号の並び（例：**abcde**）を文字コードに変える。

プログラムを実行したら、左側の画面をクリックし、文字や記号の並びを入力して、Enterキーを押す。コンピュータが文字をどう扱うか知るのに役立つ。



```
1 # ユーザーが入力した文字列の各文字を16進コードに変換し表示する
2 # プログラム。文字列を1文字ずつ処理し、各文字の文字コードを取得
3 # して16進数に変換。結果を1行ずつ出力する。
4 # 文字列を入力
5 input_string = input("文字列を入力してください: ")
6
7 print("各文字の16進コード:")
8 # 文字列の各文字に対して処理を行う
9 for char in input_string:
10     # 文字コードを16進数で出力
11     print(hex(ord(char)))
```

Powered by  trinket  
文字列を入力してください: abcde  
各文字の16進コード:  
0x61  
0x62  
0x63  
0x64  
0x65

### 環境

- Python : プログラミング言語
- trinket : Web上でPythonプログラムを実行できる学習環境

### 真偽値と論理演算

- **真偽値** : True (真) またはFalse (偽) の2つの値
- **論理演算子** : and (論理積に対応) 、 or (論理和に対応)

### 前提となる概念


- 論理積 (AND)
- 論理和 (OR)

①trinketの次のページを開く

<https://trinket.io/python/7f31113af9>

② このプログラムは, ANDとORの結果を表示する.  
実行結果が, 次のように表示されることを確認

```
1 a = True
2 b = False
3
4 print("a and b =", a and b)
5 print("a or b =", a or b)
```

Powered by  **trinke**  
( 'a and b =', False)  
( 'a or b =', True)