電気電子システム学科

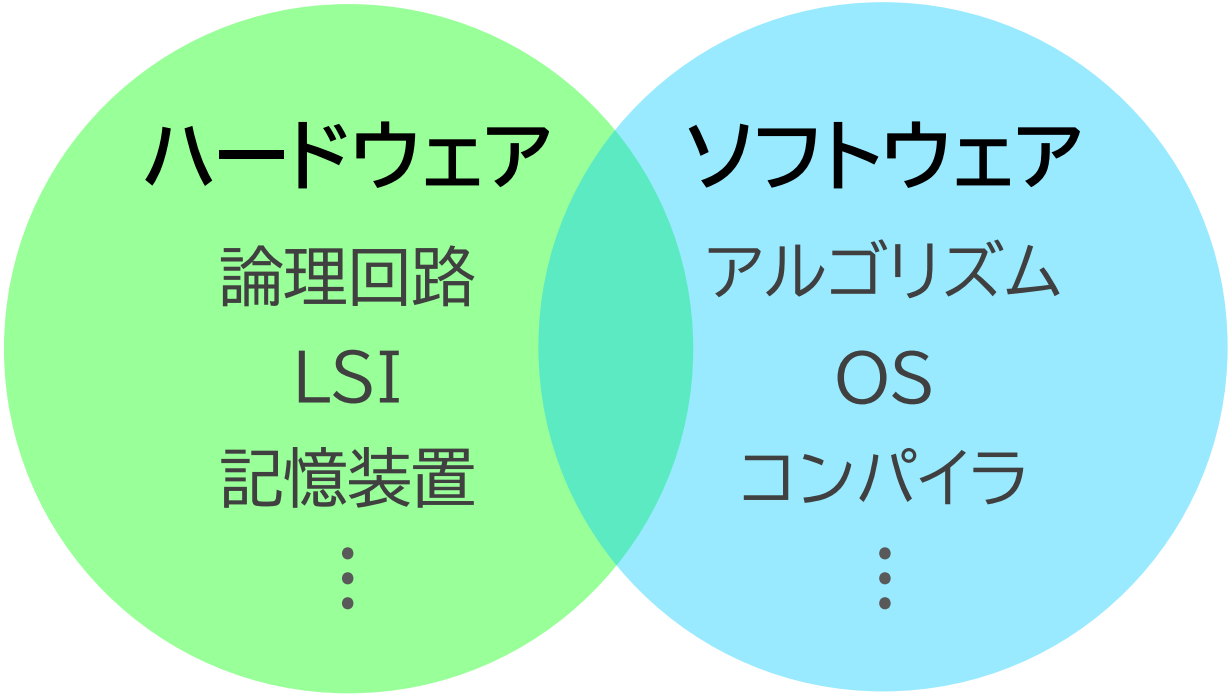
コンピュータ工学 I

Part 1

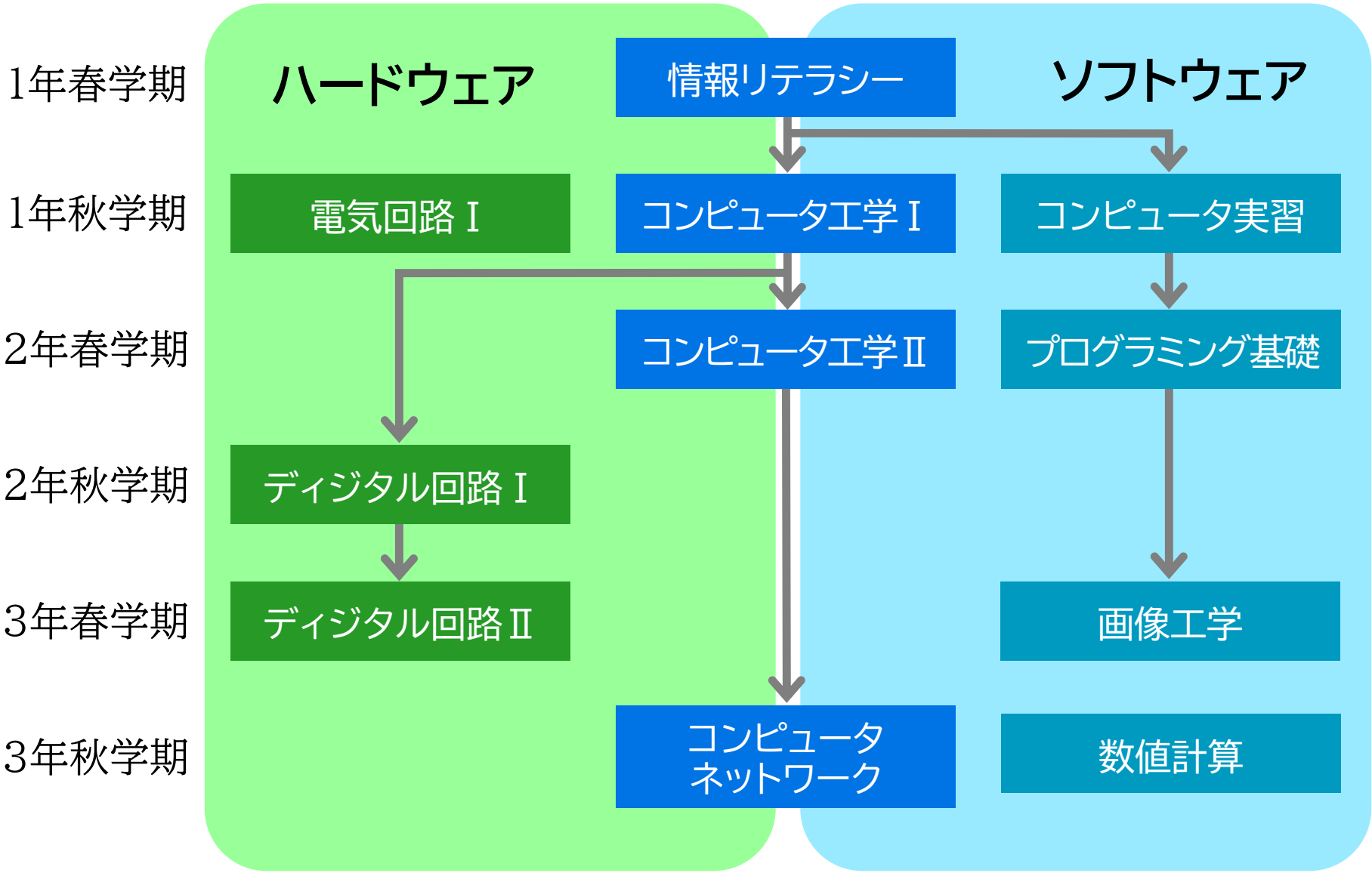
Rev. 2019.12.09

コンピュータ工学とは

コンピュータを実現するための技術について、ハードウェアとソフトウェアの両面から研究する学問。



カリキュラム体系


1年春学期
1年秋学期
2年春学期
2年秋学期
3年春学期
3年秋学期
ハードウェア
ソフトウェア
情報リテラシー
電気回路Ⅰ
コンピュータ工学Ⅰ
コンピュータ実習
コンピュータ工学Ⅱ
プログラミング基礎
ディジタル回路Ⅰ
ディジタル回路Ⅱ
画像工学
コンピュータ
ネットワーク
数値計算

コンピュータ工学Ⅰの内容

✣ 学習の目標

CPUの構造と動作の仕組みを理解する。

✣ 学習の内容

1. 数と文字の表現方法
2. 論理回路の設計
3. CPUの構成要素
4. アセンブリ言語とCPUの動作

レポート課題、試験について

- **確認テスト**

  毎回の講義後に、mylog で確認テストに解答すること。（次の講義日まで）

- **レポート課題** 全3回

- **試験**

  中間評価試験（第9回講義中）

  最終評価試験（第16回）

数と文字の表現方法

✤ 内容

❶ 2進数、16進数、基数変換

❷ 負の数の表現方法

❸ 演算（加算、減算、シフト演算）

❹ 実数の表現方法

❺ 文字の表現方法

コンピュータにおけるデータの表現法

電気の2状態 ON OFF を使って表現

1　0

数学的に扱うために2進法で書き表す

10101100011011011000110010110100 1

2進数に変換

数値化

音声データ

画像データ

文章データ

数値データ

2019　2.718281828

−322　6.0221×10^{23}

数をかぞえる

2進数　　0 1

4進数　　0 1 2 3

8進数　　0 1 2 3 4 5 6 7

10進数　0 1 2 3 4 5 6 7 8 9

16進数　0 1 2 3 4 5 6 7 8 9 A B C D E F

数字として扱う

進数対応表①

10進数	8進数	4進数	2進数	16進数
0	0	0	0	0
1				
2				
3				
4				
5				
6				
7				
8				

進数対応表①

10進数	8進数	4進数	2進数	16進数
0	0	0	0	0
1	1	1	1	1
2	2	2	10	2
3	3	3	11	3
4	4	10	100	4
5	5	11	101	5
6	6	12	110	6
7	7	13	111	7
8	10	20	1000	8

桁上がり

進数対応表②

10進数	8進数	4進数	2進数	16進数
8	10	20	1000	8
9	11	21	1001	9
10	12	22	1010	A
11	13	23	1011	B
12	14	30	1100	C
13	15	31	1101	D
14	16	32	1110	E
15	17	33	1111	F
16	20	100	10000	10

桁上がり

進数対応表③

10進数	8進数	4進数	2進数	16進数
16	20	100	10000	10
17	21	101	10001	11
18	22	102	10010	12
19	23	103	10011	13
20	24	110	10100	14
21	25	111	10101	15
22	26	112	10110	16
23	27	113	10111	17
24	30	120	11000	18

進数対応表④

10進数	8進数	4進数	2進数	16進数
24	30	120	11000	18
25	31	121	11001	19
26	32	122	11010	1A
27	33	123	11011	1B
28	34	130	11100	1C
29	35	131	11101	1D
30	36	132	11110	1E
31	37	133	11111	1F
32	40	200	100000	20

2進数と16進数の対応関係

重要

2進数４桁が、16進数１桁に対応する。

2進数	16進数
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

2進数	16進数
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

16進数にすることで、見た目の桁数を少なくできる。

2進数と16進数の相互の変換が簡単にできる。

2進数と4,8,16進数の相互変換

2進数 2桁 ⟷ 4進数 1桁

2進数 3桁 ⟷ 8進数 1桁

2進数 4桁 ⟷ 16進数 1桁

0 1 1 1 0 1 ➡ 131(4進数)

0 1 1 1 0 1 ➡ 35(8進数)

0 0 0 1 1 1 0 1 ➡ 1D(16進数)

数の表現

✣ 数の表記

$N_{(p)}$：p進数の数値N

基数

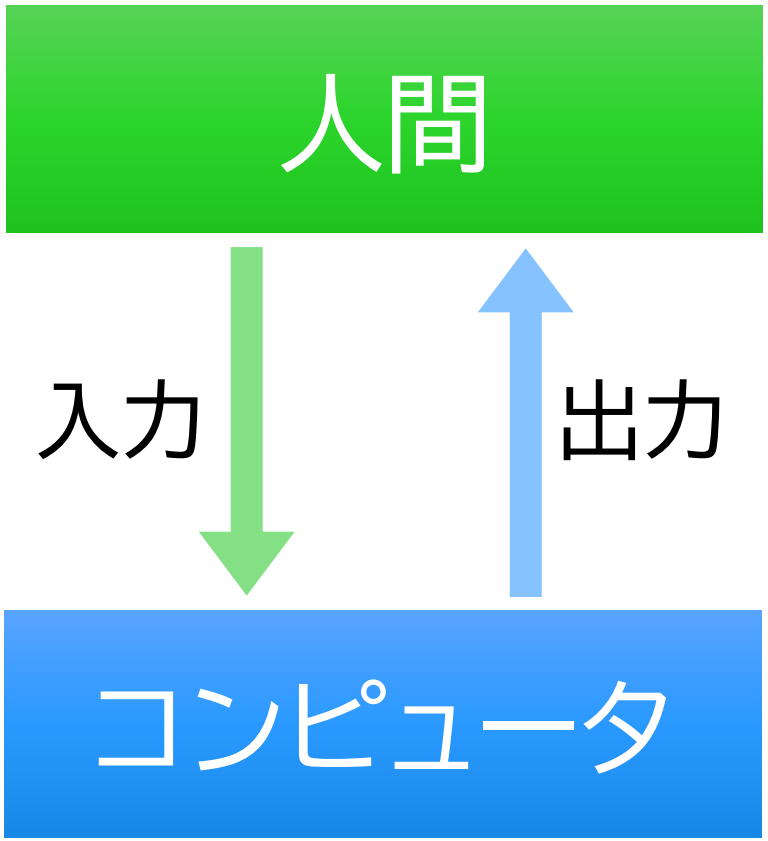
$101_{(2)}$　2進数の101
$101_{(10)}$　10進数の101

✣ 基数変換

数値を別の基数の表記に変える。

$101_{(2)} \leftrightarrow 5_{(10)}$

基数変換の必要性



演算・制御・記憶

10進数 を使う

基数変換が必要

2進数 を使う

基数変換

- p進数 → 10進数の変換
- 10進数 → p進数の変換
  - 整数部の変換
  - 小数部の変換

p進数(整数部)→10進数

p進数の整数部を $d_n d_{n-1} \cdots d_1 d_0\ _{(p)}$ とする$(n \geqq 0)$。

この値と等しい10進数をXとする。

$$X = d_n \cdot p^n + d_{n-1} \cdot p^{n-1} + \cdots + d_1 \cdot p^1 + d_0$$

$11101_{(2)}$は、

$$1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1$$
$$= 16 + 8 + 4 + 0 + 1 = 29_{(10)}$$

p進数(小数部)→10進数

p進数の小数部を $0.d_{-1}d_{-2}\cdots d_{-m}{}_{(p)}$ とする($m>0$)。

$$X = d_{-1}\cdot p^{-1} + d_{-2}\cdot p^{-2} + \cdots + d_{-m}\cdot p^{-m}$$

$0.1101_{(2)}$は、

$$1\cdot 2^{-1} + 1\cdot 2^{-2} + 0\cdot 2^{-3} + 1\cdot 2^{-4}$$

$$= 0.5 + 0.25 + 0 + 0.0625 = 0.8125_{(10)}$$

p進数→10進数のまとめ

重要

p進数を

$$d_n d_{n-1} \cdots d_1 d_0 \,.\, d_{-1} d_{-2} \cdots d_{-m+1} d_{-m} \,(p)$$

とする$(n \geqq 0,\ m > 0)$。

この値と等しい10進数Xは、

$$X = d_n \cdot p^n + d_{n-1} \cdot p^{n-1} + \cdots + d_1 \cdot p^1 + d_0$$
$$+ d_{-1} \cdot p^{-1} + d_{-2} \cdot p^{-2} + \cdots + d_{-m} \cdot p^{-m}$$

10進数(整数部)→p進数

重要

p) $X_{(10)}$ 余り

p) q_0 … d_0

p) q_1 … d_1

⋮ ⋮

p) q_{n-1} … d_{n-1}

0 … $d_n = q_{n-1}$

pで割ったときの余りを求める。
これを繰り返す。

$$X_{(10)} = d_n d_{n-1} \cdots d_1 d_0\ (p)$$

10進数(整数部)→p進数

$$
\begin{array}{rrcl}
p) & N_{(10)} & & \text{余り} \\
p) & q_0 & \cdots & d_0 \\
p) & q_1 & \cdots & d_1 \\
 & \vdots & & \vdots \\
p) & q_{n-1} & \cdots & d_{n-1} \\
 & 0 & \cdots & d_n
\end{array}
$$

$$X = d_n d_{n-1} \cdots d_1 d_0{}_{(p)}$$

$$
\begin{array}{rrcl}
2) & 29 & & \text{余り} \\
2) & 14 & \cdots & 1 \\
2) & 7 & \cdots & 0 \\
2) & 3 & \cdots & 1 \\
2) & 1 & \cdots & 1 \\
 & 0 & \cdots & 1
\end{array}
$$

$$29 = 11101_{(2)}$$

原理

$$X_{(10)} \leftrightarrow d_n d_{n-1} \cdots d_2 d_1 d_0\ (p)$$

$$X = d_n \cdot p^n + d_{n-1} \cdot p^{n-1} + \cdots + d_2 \cdot p^2 + d_1 \cdot p + d_0$$

$$= p \cdot (d_n \cdot p^{n-1} + d_{n-1} \cdot p^{n-2} + \cdots + d_2 \cdot p + d_1) + d_0$$

$X \div p$の商

$X \div p$の余り

X を p で繰り返し割っていけば、
その余りから $d_0, d_1, \cdots, d_n$ が順に求まる。

10進数(小数部)→p進数

重要

$$X \times p = a_0 + b_0$$

$$b_0 \times p = a_1 + b_1$$

$$b_1 \times p = a_2 + b_2$$

$$\vdots$$

$b_n = 0$ になるまで続ける。

$$0 < X_{(10)} < 1$$

a_i 整数部

b_i 小数部

小数部にpを掛ける。
これを繰り返す。

$$X_{(10)} = 0.a_0 a_1 \cdots a_n\,(p)$$

10進数(小数部)→p進数

$$X \times p = a_0 + b_0$$

$$b_0 \times p = a_1 + b_1$$

$$b_1 \times p = a_2 + b_2$$

$$\vdots$$

$$X = 0.a_0 a_1 \cdots a_n{}_{(p)}$$

$$0.8125 \times 2 = 1.625$$

$$0.625 \times 2 = 1.25$$

$$0.25 \times 2 = 0.5$$

$$0.5 \times 2 = 1.0$$

$$0.8125 = 0.1101_{(2)}$$

基数変換のまとめ

重要

✤ p進数 → 10進数

p進数のk桁目の値d_kにp^kを掛けて、和を計算する。$k=0$が1の位になる。

✤ 10進数 → p進数

整数部をpで繰り返し割って余りを求めていき、最後の余りを最上位桁にして並べる。

小数部にpを繰り返し掛けていき、最初の整数部を小数第1位の桁にして並べる。

2進数

重要

MSB(最上位bit)

LSB(最下位bit)

$1\,0\,1\,0\,1_{(2)}$

bit　2進数の1桁

8 bit ＝ 1 byte

n bitの2進数は、2^n 個の数を表現できる。

2進数の加算

✤ 基本演算

			1
0	1	1	1
＋ 0	＋ 0	＋ 1	＋ 1
0	1	10	11

bit数の制限

コンピュータの記憶容量は有限

数値1個のbit数（桁数）に上限がある

- 最大bit数が見て分かるように、MSBまでを０で埋めて表記する。
- 計算によってMSBを越えたbitが生じたら、そのbitの値は捨てる（記憶しない）。

C言語で8進数、16進数を使う

参考

scanf, printfで8進数、16進数を使う方法

```
int  a, b, c = 31;
scanf("%o", &a);                    8進数の整数をaに入力
scanf("%x", &b);                    16進数の整数をbに入力
printf("a=%d, b=%d¥n", a, b);       aとbを10進数で出力
printf("c=%o (8)¥n", c);            c=31を8進数で出力
printf("c=%x (16)¥n", c);           c=31を16進数で出力
```

8進数、16進数の整数定数を使う方法

```
int  a = 010;                       数値の頭に0を付けると8進数になる
int  b = 0xF;                       数値の頭に0xを付けると16進数になる
printf("a=%d, b=%d¥n", a, b);       10進数で出力
```

シフト演算

- 2進数の全bitを左または右に移動する。
- MSBまたはLSBを越えたbitの値は捨てる。

n bit 左シフト → 2^n倍になる

n bit 右シフト → $\frac{1}{2^n}$倍になる

（ただし、MSBやLSBを越えない場合）

2進数の乗算

シフト演算と加算の組み合わせで乗算が行える。

```
    000111
×   000110
------------
    000000
    001110   000111を1bit左シフト
+   011100   000111を2bit左シフト
------------
    101010   答え
```

負の整数

コンピュータは０と１しか使えない。

↓

＋とーの符号が存在しない

✤ 負の整数の表現法

❶ 符号絶対値法

❷ 1の補数

❸ 2の補数

符号絶対値法

重要

MSBを符号として用いる。

0 0 0 1 1 1

符号bit

0 → ＋　正の数、または、0

1 → －　負の数

1の補数

✤ 1の補数とは

2進数 x の1の補数を $\overline{x}$ とするとき、

$x + \overline{x} = 111\cdots1_{(2)}$ となる。

✤ 2進数 x の1の補数の求め方

xの1を0に、0を1に書き換える。

（bit反転という）

2の補数

重要

✤ 2の補数とは

2進数 x の2の補数を x' とするとき、

$x + x' = 100\cdots0_{(2)}$ となる。

✤ 2進数 x の2の補数の求め方

❶ xの1を0に、0を1に書き換える。
（1の補数を求める。）

❷ その値に1を加算する。

2の補数

❶ xの1を0に、0を1に書き換える。
（1の補数を求める。）
❷ その値に1を加算する。

$0011_{(2)}$の2の補数

❶ 0011
↓↓↓↓
1100 1の補数

❷ 1100
＋ 1
1101 2の補数

コンピュータにおける減算

重要

減算は、負の数との加算に変えて計算する。

$$5-3=$$

$$5+(-3)=$$

2進数の負の数は
2の補数で表現する

コンピュータにおける 5−3 の計算

$5_{(10)} = 0101_{(2)}$　　$3_{(10)} = 0011_{(2)}$

$0011_{(2)}$の2の補数は、$1101_{(2)}$

したがって、$-3_{(10)} = 1101_{(2)}$

$$5-3 = 5+(-3) = 0101_{(2)} + 1101_{(2)}$$

$$\begin{array}{r} 0101 \\ +\ 1101 \\ \hline \cancel{1}0010 \end{array}$$

計算結果は、

$0010_{(2)} = 2_{(10)}$

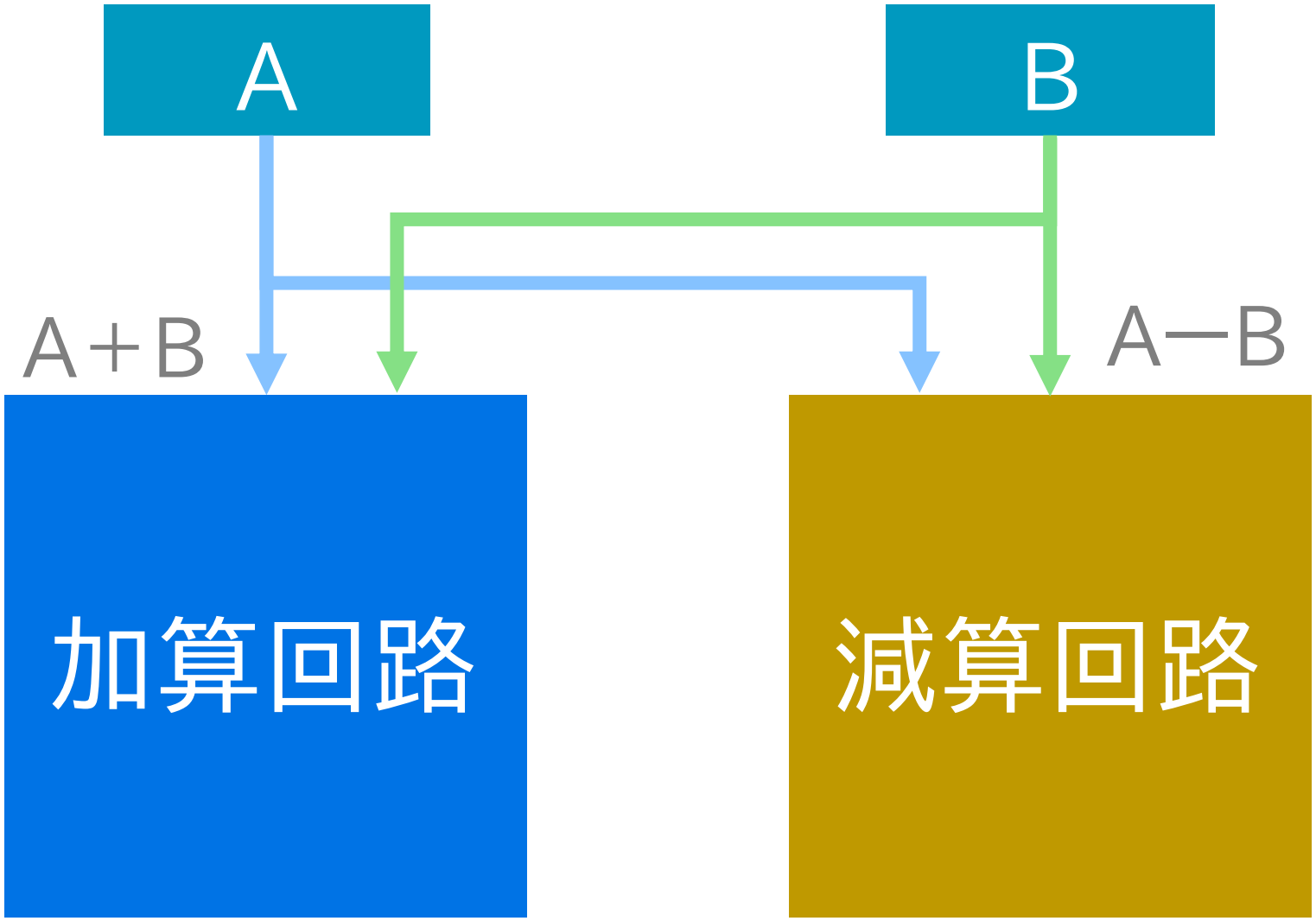
コンピュータ設計の方針

✤ 良いコンピュータとは？

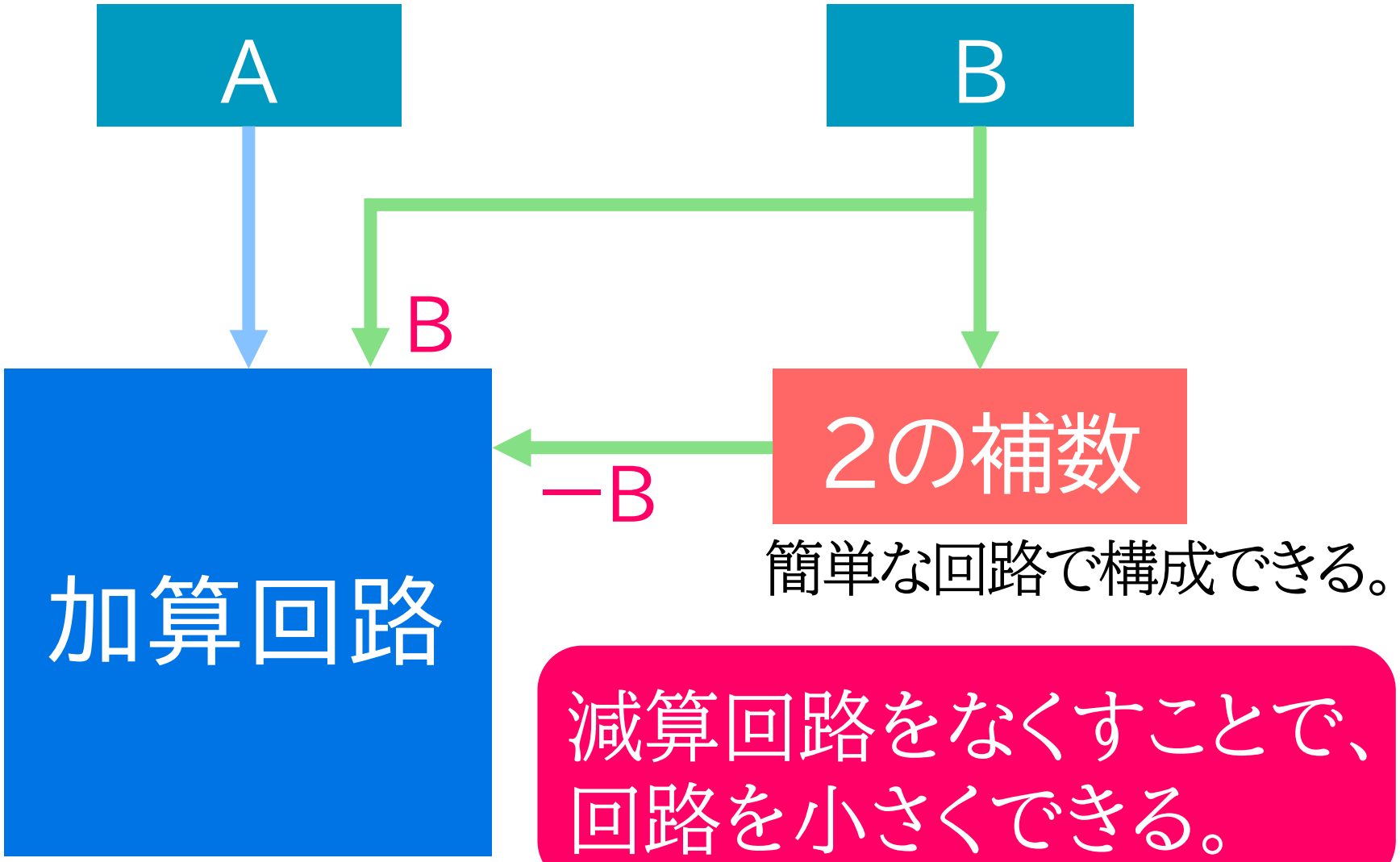
- 処理が高速
- 小型
- 低コスト

コンピュータの電子回路の無駄を省き、
可能な限り小さく構成することが望ましい。

加減算回路の構成①



加減算回路の構成②



コンピュータでの整数の表現法

n bitの2進数は、2^n 個の数を表現できる。

✤ 符号なし2進数

MSBを符号bitとして使わない。

$$0 \leqq x \leqq 2^n - 1$$

✤ 符号つき2進数

MSBを符号bitとして使う。

$$-2^{n-1} \leqq x \leqq 2^{n-1} - 1$$

コンピュータでの実数の表現法

- **固定小数点数**

  小数点の位置を特定のbit間に固定する。

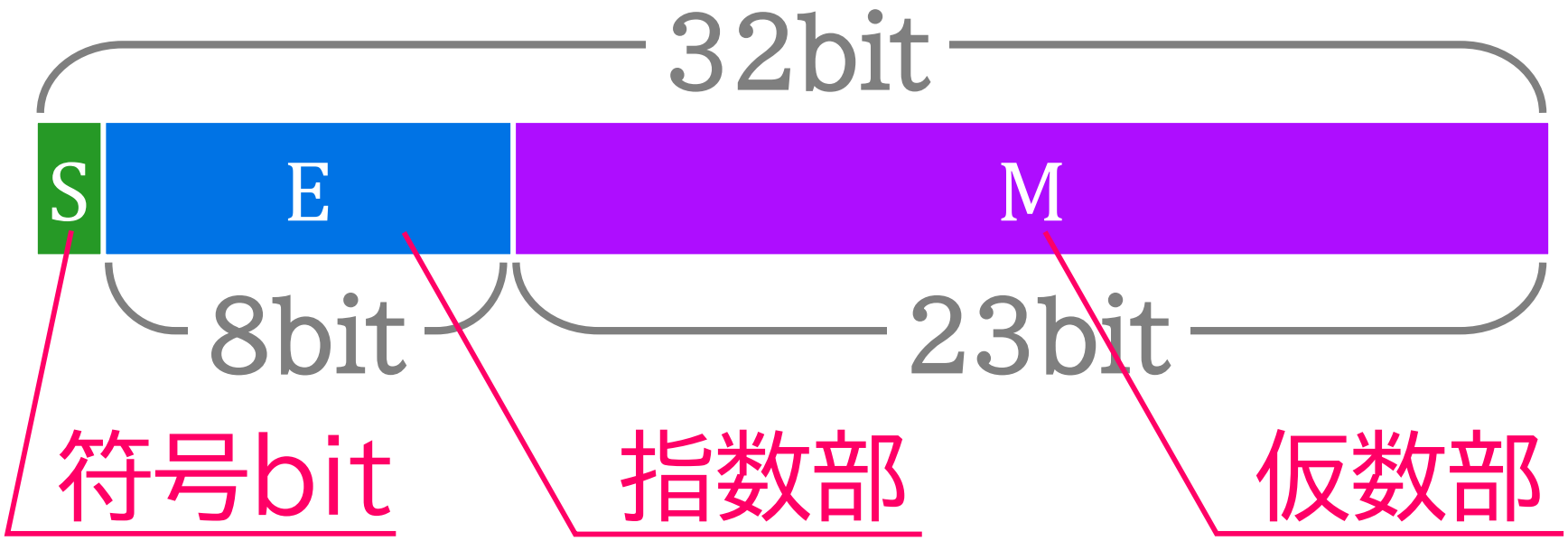
  整数は、小数部のない固定小数点数。

  bit数より桁数の多い整数や小数が表現できない。

- **浮動小数点数**

  仮数部と指数部に分けて数値を表現する。

IEEE単精度浮動小数点数



$$N = (-1)^{S} \times 1.M_{(2)} \times 2^{E-127}$$

$10101000_{(2)}$のIEEE浮動小数点数表記

$$1010\,1000_{(2)} = 1.0101_{(2)} \times 2^{7}{}_{(10)}$$

正の数であるから、符号bit S=0

2^{7}より、指数部 $E=7+127=134_{(10)}$
8bit 2進数に変換して、$E=10000110_{(2)}$

仮数部には小数部23bitまでを当てる。
仮数部 M=01010000000000000000000

01000011001010000000000000000000

C言語の変数型の名前の意味

参考

型名	意味
int	整数 integer 32 bit
char	文字 character 8 bit（ASCIIコード1文字分）
float	浮動小数点数 floating point number 32 bit
double	倍精度浮動小数点数 double precision floating point number 64 bit（float型の2倍の精度）

C言語で表現できる数の範囲

参考

整数型（固定小数点数）

型名	bit数	表現できる数の範囲
char	8	−128～127
short	16	−32,768～32,767
long (int)	32	−2,147,483,648～2,147,483,647

実数型（浮動小数点数）

型名	bit数	仮数のbit数	指数の範囲
float	32	23	2の−126～127乗
double	64	52	2の−1022～1023乗

整数(固定小数点数)同士の計算

$$
\begin{array}{rr}
 & 0011101 \\
+ & 0000110 \\
\hline
 & 0100011
\end{array}
$$

bitの並びを変えずに計算ができる。

計算処理が単純 ➡ 計算が速い

実数(浮動小数点数)同士の計算

$$1.1101 \times 2^{5} + 1.1000 \times 2^{3}$$

$$1.1101 \times 2^{5} + 0.0110 \times 2^{5}$$

$$10.0011 \times 2^{5}$$

$$1.0001 \times 2^{6}$$

指数を揃えてから計算しなければならない。

計算処理が複雑 ➡ 計算が遅い

数値表現の長所と短所

重要

固定小数点数	浮動小数点数
表現できる数値の範囲が狭い	表現できる数値の範囲が広い
計算が速い	計算が遅い

誤差問題

- **丸め誤差**
  下位の桁を削除することにより生じる、本来の数値との差。
- **情報落ち**
  絶対値の大きい数と小さい数の加減算において、小さい数が計算に反映されない。
- **桁落ち**
  ほぼ等しい数の減算をするとき、有効桁数が大幅に失われる。

8bit JISコード

ASCIIコード

上位 4bit

文字コード
41(16)

下位 4bit

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	`	p				ー	タ	ミ		
1			!	1	A	Q	a	q			。	ア	チ	ム		
2			”	2	B	R	b	r			「	イ	ツ	メ		
3			#	3	C	S	c	s			」	ウ	テ	モ		
4			$	4	D	T	d	t			、	エ	ト	ヤ		
5			%	5	E	U	e	u			・	オ	ナ	ユ		
6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7			’	7	G	W	g	w			ァ	キ	ヌ	ラ		
8			(	8	H	X	h	x			ィ	ク	ネ	リ		
9			)	9	I	Y	I	y			ゥ	ケ	ノ	ル		
A			*	:	J	Z	j	z			ェ	コ	ハ	レ		
B			+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C			,	<	L	¥	l	|			ャ	シ	フ	ワ		
D			−	=	M	]	m	}			ュ	ス	ヘ	ン		
E			.	>	N	^	n	~			ョ	セ	ホ	゛		
F			/	?	O	_	o				ッ	ソ	マ	゜		

16bit JISコード

16bit　　8bit

東京City

456C 357E 43 69 74 6F(16)

El 5~ City

コードの種類を誤ると、文字化けが起こる。

16bit JISコード

16bit　　8bit

東京City

456C　357E　43 69 74 6F (16)

1B2442　1B284A

16bitコード開始　8bitコード開始

制御コードを埋め込むことで、文字化けを防ぐ。

16bit シフトJISコード

16bit　　8bit

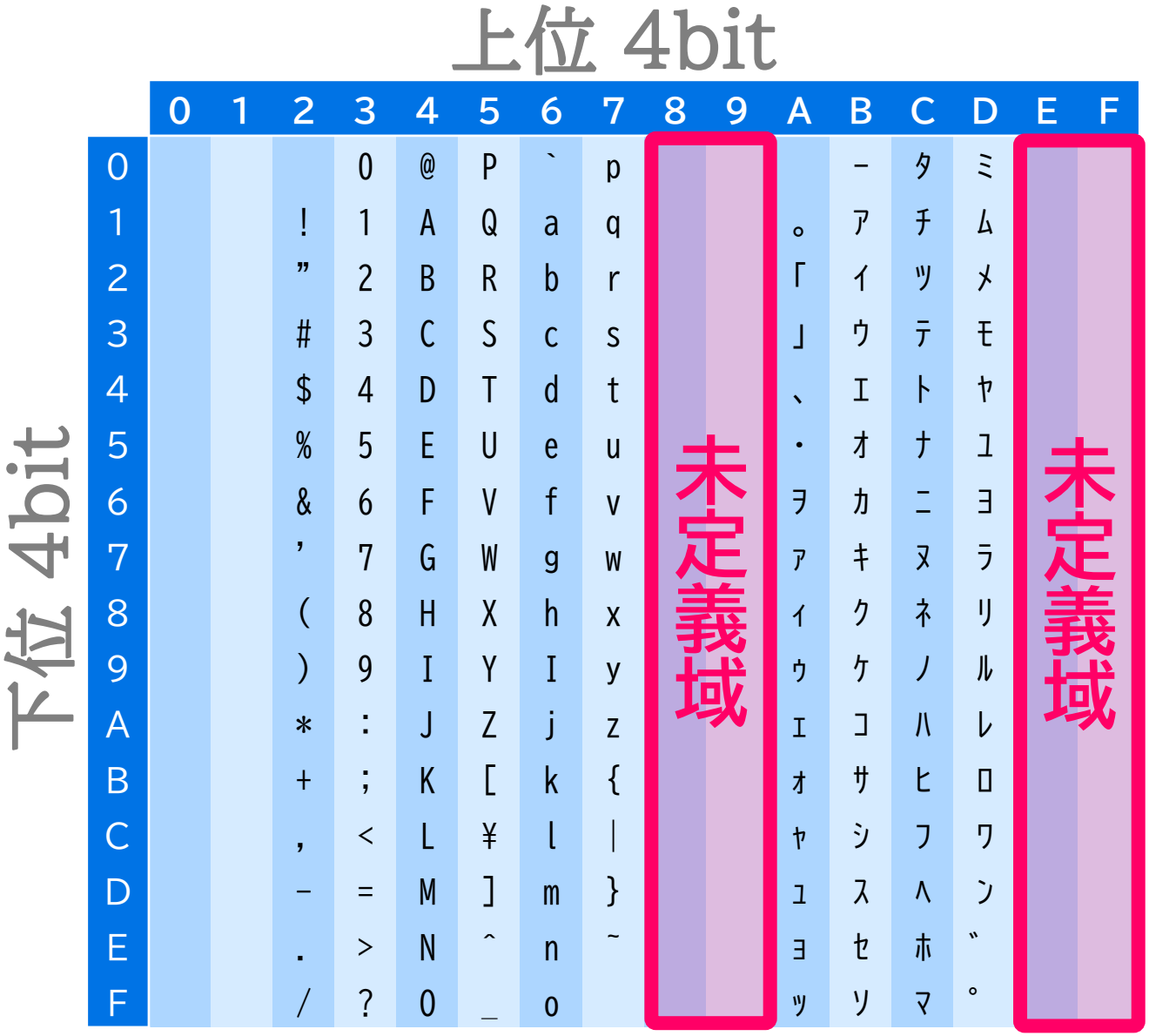
東京City

938C　8B9E　43 69 74 6F(16)

未定義　未定義　City

上位8bitに未定義域コードを使う。

未定義域コードとは


上位 4bit
下位 4bit
0 1 2 3 4 5 6 7 8 9 A B C D E F
0 ! 0 @ P ` p 。 ー タ ミ
1 ! 1 A Q a q 。 ア チ ム
2 ” 2 B R b r 「 イ ツ メ
3 # 3 C S c s 」 ウ テ モ
4 $ 4 D T d t 、 エ ト ヤ
5 % 5 E U e u ・ オ ナ ユ
6 & 6 F V f v ヲ カ ニ ヨ
7 ’ 7 G W g w ァ キ ヌ ラ
8 (8 H X h x ィ ク ネ リ
9) 9 I Y I y ゥ ケ ノ ル
A * : J Z j z ェ コ ハ レ
B + ; K [k { ォ サ ヒ ロ
C , < L ¥ l | ャ シ フ ワ
D − = M] m } ュ ス ヘ ン
E . > N ^ n ~ ョ セ ホ ゛
F / ? O _ o ッ ソ マ ゜
未定義域
未定義域

ユニコード(UNICODE)

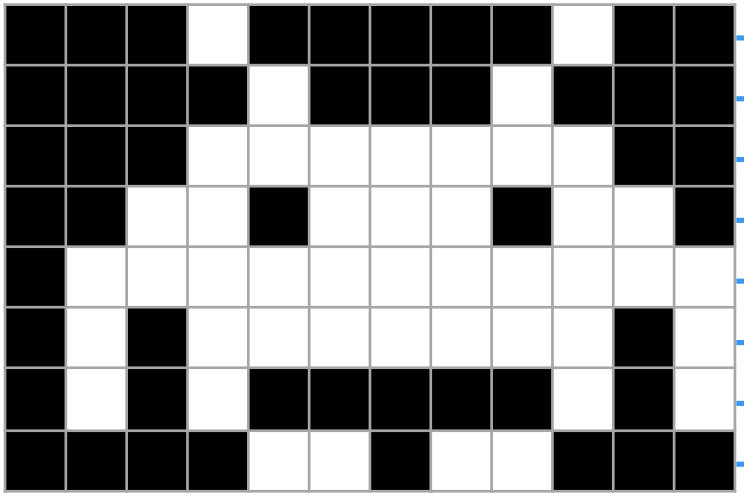
重要

多国語に対応するため、世界中の主要な文字や記号をまとめた文字コード

Unicode12.1.0では 137,929文字が登録。

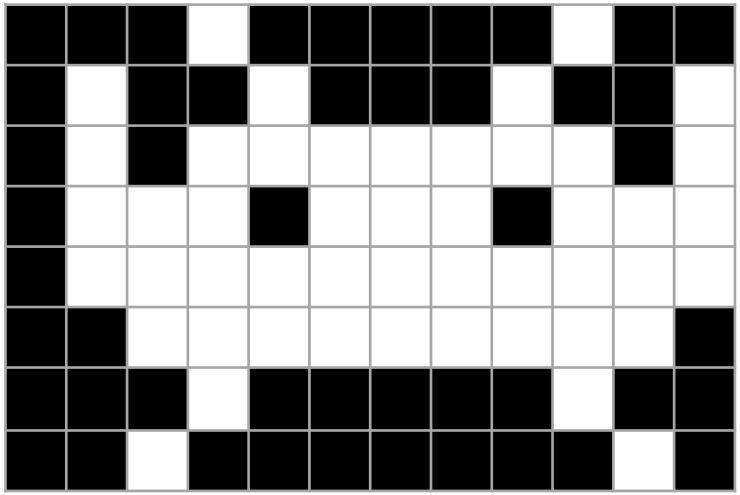
令和が追加された

ビットマップ（白黒画像）



0001 0000 0100 → 104
0000 1000 1000 → 088
0001 1111 1100 → 1FC
0011 0111 0110 → 376
0111 1111 1111 → 7FF
0101 1111 1101 → 5FD
0101 0000 0101 → 505
0000 1101 1000 → 0D8

黒を0、
白を1 にする。

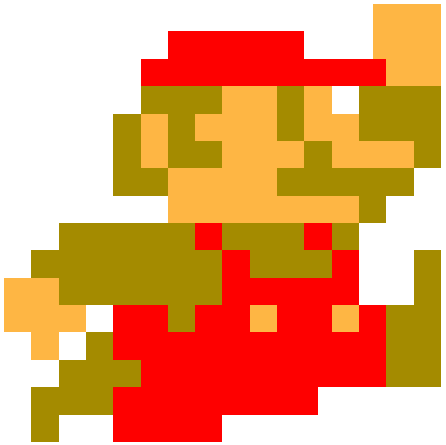


数値化された画像データ
1040881FC3767FF5FD5050D8

スペースインベーダー（1978年）
©TAITO Corp. より

カラービットマップ（カラー画像）

1ドットに複数のbitを割り当て、多値を表現する。



スーパーマリオブラザーズ
（1985年）©Nintendo より

ファミリーコンピュータ（任天堂,1983年）では、1ドットに 2bitを割り当てている。

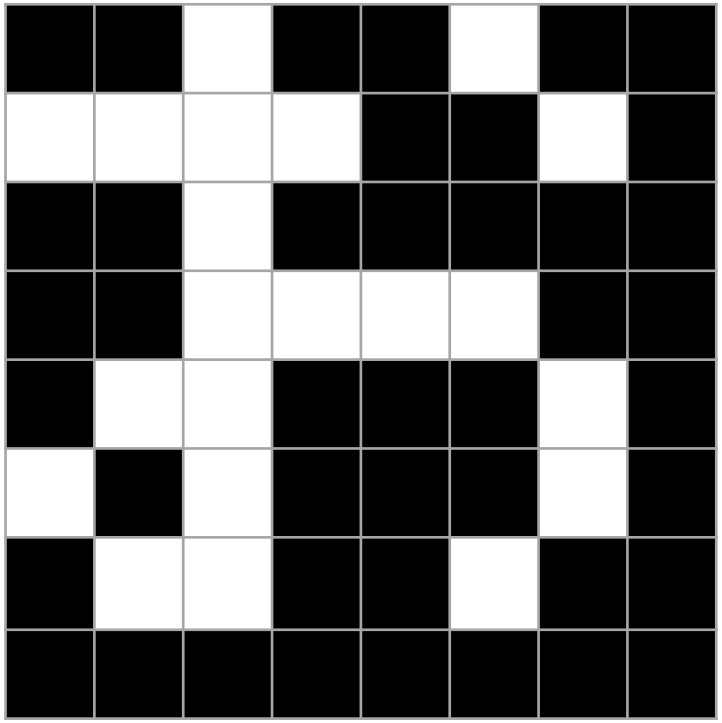
そのため、各キャラクターは最大4色（そのうち1色は背景の透明色）で表現されている。（カラーパレット方式）

現在の一般的なコンピュータは、1ドットに24bitを割り当て、最大16,777,216色を表現できる。

ビットマップフォント

点の集合(ビットマップ)でつくられたフォント

文字「お」



0010 0100 → 24
1111 0010 → F2
0010 0000 → 20
0011 1100 → 3C
0110 0010 → 62
1010 0010 → A2
0110 0100 → 64
0000 0000 → 00

24F2203C62A26400

課題 1

- 教科書 2章末の演習問題すべて
- これまでの講義についての感想

  どのような内容(感想・要望)でも良い。

提出日時　12月16日(月) 講義開始前

レポートの作成について

- 岡山理大学専用のレポート用紙に書く。
- ホッチキスまたは糊で綴じる。
- 学生番号、氏名、講義名、提出日を書く。
- 途中の計算過程を書く。
- 解答した後、教科書の演習問題解答を見て、赤ペンで○×をつける。
- 間違えた問題、解けなかった問題は赤ペンで計算過程と正答を書く。