



コンピュータ・サイエンス1

第9回 コンピュータでの文字の扱い(1)

人間科学科コミュニケーション専攻
白銀 純子

第9回の内容

- 浮動小数
- コンピュータでの文字の扱い(1)

設問1

■ 下記の6桁の2進数のうち、2の補数を考えるとすると、マイナスの数はどれか、すべて答えなさい。

1. 000000
2. 101010
3. 010101
4. 111000
5. 000111
6. 110011
7. 001100
8. 111111

解答: 2, 4, 6, 8

設問2

■ 「やってみよう!」の2の補数の足し算の1. の結果を報告すること

- 1. の問題: $(+10) + (+8)$ を5桁の2の補数として計算し、10進数として表現

$$(+10)_{10} = (01010)_2$$

$$(+8)_{10} = (01000)_2$$

$$\begin{array}{r} 01010 \\ +) 01000 \\ \hline 10010 \end{array}$$

↑
負の数と判断



2の補数(マイナスの数)として10進数に直す

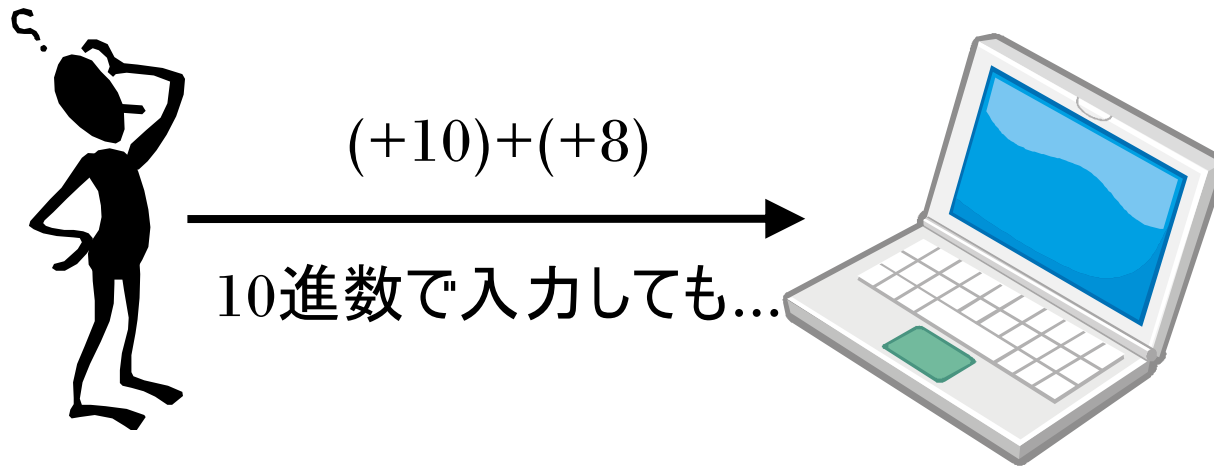
1. $(10010 - 1)_2 = (10001)_2$ (2の補数から1を引く)
2. $(10001)_2 \rightarrow (01110)_2$ (1と0を反転する)
3. $(01110)_2 = (14)_{10}$ (2進数を10進数に直す)
4. $(14)_{10} \rightarrow (-14)_{10}$ (マイナス記号をつける)

解答: -14

前回の質問の解答

コンピュータでの計算(1)

- コンピュータの内部では、常に2進数!



1. 入力された数はすぐに2進数で表現!
 - ✓ $(+10)_{10} = (01010)_2$
 - ✓ $(+8)_{10} = (01000)_2$
2. 計算は必ず2進数で!
 - ✓ $(+10)_{10} + (+8)_{10}$ ✗ (この計算はしない)
 - ✓ $(01010)_2 + (01000)_2$ ○
3. 計算結果の表示だけ10進数!

コンピュータでの計算を考えると、必ず2進数で!

- 1, 2, 3の手順で計算すること!
- 10進数を頭の中に置いておくと、混乱のもと

コンピュータでの計算(2)

- コンピュータはどこまでも大きな数を表現できるわけではない
 - ハードウェアの性能等の理由で
 - ➡ 表現できる数(2進数)の桁数に制限ができた
- コンピュータに搭載する電子回路はなるべく節約したい
 - 様々な種類(足し算用・かけ算用・引き算用・割り算用...)の回路を搭載するとコンピュータの値段が高くなる
 - ➡ マイナスの数を扱って、計算のための回路は足し算だけにする
 - ➡ さらに節約のために、マイナスの数とプラスの数は一緒くたに扱いたい
 - ➡ 2進数で最も大きな桁が0のときは正の数、1のときは負の数として扱う

コンピュータでの2進数では、桁数と正負の判定方法は、絶対のルール!

➤ オーバーフローはこのルールがあることによって起こる弊害

オーバーフローのおさらい

■ オーバーフローには2種類!

- 2進数で、計算結果が指定された桁数よりも多くなってしまい、大きい桁が削られてしまうこと
- 2進数で、正の数同士(負の数同士)の計算結果が、負の数(正の数)になってしまうこと

要は、人間が手でやった計算結果と、コンピュータがやった計算結果が違う現象!

コンピュータの考え方

Ex. 1: 5桁の2進数で...

$$(01011)_2 \times (00100)_2 = (0101100)_2$$



あ! 7桁になった!
でも7桁も扱えない!
じゃ、 $(01100)_2$ だな!

Ex. 2: 4桁の2進数で...

$$(0110)_2 + (0111)_2 = (1101)_2$$



あ! 4桁目が1だ!
じゃ、これはマイナスの数だな!

計算結果の桁数が短くなるオーバーフローと、正負の判定は別
➤ 先頭が1で始まる数は、オーバーフロー関係なくマイナスの数

人間がコンピュータの動作を考えるには...

■ 計算結果を見たときに...

1. 決められた桁数を超えていないか? → 超えていたら、桁数あわせ
2. 一番大きな桁が0か1か? → 0なら正の数、1なら負の数

■ 1. の結果に関係なく判断する

Ex. 1: 5桁の2進数の計算結果が $(011001)_2$ の場合

1. 計算結果が6桁なので桁数あわせ
→ $(11001)_2$
2. 一番大きな桁が1
→ この数は負の数 $(11001)_2 = (-7)_{10}$

Ex. 3: 5桁の2進数の計算結果が $(01001)_2$ の場合

1. 計算結果が5桁なので桁数あわせなし
→ $(01001)_2$ のまま
2. 一番大きな桁が0
→ この数は正の数 $(01001)_2 = (9)_{10}$

Ex. 2: 5桁の2進数の計算結果が $(0100010)_2$ の場合

1. 計算結果が7桁なので桁数あわせ
→ $(00010)_2$
2. 一番大きな桁が0
→ この数は正の数 $(00010)_2 = (2)_{10}$

Ex. 4: 5桁の2進数の計算結果が $(10010)_2$ の場合

1. 計算結果が5桁なので桁数あわせなし
→ $(10010)_2$ のまま
2. 一番大きな桁が1
→ この数は負の数 $(10010)_2 = (-12)_{10}$

なので...

■ 5桁の2進数 $(11110)_2$

- 一番大きな桁が1 → この数は負の数と判断 → 2の補数として10進数に

1. $(11110 - 1)_2 = (11101)_2$ (2の補数から1を引く)
2. $(11101)_2 \rightarrow (00010)_2$ (1と0を反転する)
3. $(00010)_2 = (2)_{10}$ (2進数を10進数に直す)
4. $(2)_{10} \rightarrow (-2)_{10}$ (マイナス記号をつける)

■ 5桁の2進数 $(10011)_2$ (前回授業のp. 30のスライドの計算結果)

- 一番大きな桁が1 → この数は負の数と判断 → 2の補数として10進数に

1. $(10011 - 1)_2 = (10010)_2$ (2の補数から1を引く)
2. $(01101)_2 \rightarrow (01101)_2$ (1と0を反転する)
3. $(01101)_2 = (13)_{10}$ (2進数を10進数に直す)
4. $(13)_{10} \rightarrow (-13)_{10}$ (マイナス記号をつける)



Question!

前回の復習

2の補数表現[2](p. 9)

- 2の補数 = 負の数を2進数で表現したもの(コンピュータの世界では)
- 計算方法(例: -20を10桁の2進数に直す)
 1. 2の補数に直したい10進数のマイナスを取り除く
 - $(-20)_{10} \rightarrow (20)_{10}$
 2. 1. の結果を2進数に直す
(この時点で桁数あわせ!! 後で桁数あわせをすると、合わせ方を間違えやすい)
 - $(20)_{10} = (0000010100)_2$
 3. 2. の結果の0と1を逆にする(0の桁を1、1の桁を0にする)

0 0 0 0 0 1 0 1 0 0



1 1 1 1 1 0 1 0 1 1

2の補数表現[2](p. 9)

- 2の補数 = 負の数を2進数で表現したもの(コンピュータの世界では)
- 計算方法(例: -20を10桁の2進数に直す)
 3. の結果に1を足し算する

$$\begin{array}{r} 1111101011 \\ +) \quad \quad \quad 1 \\ \hline 1111101100 \end{array}$$

-20を2進数に直した結果
(2の補数 = 2進数での負の数の表現)

2進数での負の数の表現では、
「-」の記号はつけない

2の補数を10進数に変換[2]

- 計算方法(例: 1111101100を10進数に直す)

1. 2の補数から1を引き算する

$$\begin{array}{r} 1111101100 \\ -) \quad \quad \quad 1 \\ \hline 1111101011 \end{array}$$

2. 1. の結果の0と1を逆にする(0の桁を1、1の桁を0にする)

$$\begin{array}{c} 1111101011 \\ \downarrow \\ 0000010100 \end{array}$$

※2の補数→10進数の方法は、10進数→2の補数の逆

2の補数を10進数に変換[2]

■ 計算方法(例: 1111101100を10進数に直す)

1.2. の結果を10進数に直す

■ $(0000010100)_2 = (20)_{10}$

2.3. の結果に-(マイナス)をつける

■ $(20)_{10} \rightarrow (-20)_{10}$

1111101100を
10進数に直した数

※2の補数→10進数の方法は、10進数→2の補数の逆

小数の表現方法(続き)

10進数の小数を2進数に直す[1]

- 例: 101.1101 (整数部分101は10進数で5)
 1. 2進数の整数部分は、通常の方法で10進数に直す
 2. 10進数の小数部分各桁の上に「1/2」を書く
 3. 2. で書いた「1/2」の「2」の右肩に左から1, 2, 3, ...と書いていく
 - $1/2^0, 1/2^1, 1/2^2, \dots$ ができていく

2.

	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	
	2	2	2	2	
0	.	1	1	0	1



3.

左から右に、1, 2, 3, ...と
番号をつける

	$\frac{1}{2^1}$	$\frac{1}{2^2}$	$\frac{1}{2^3}$	$\frac{1}{2^4}$	
	2	2	2	2	
0	.	1	1	0	1

10進数の小数を2進数に直す[2]

4. 各桁の上の「 $1/2^n$ 」と、それぞれの桁の数をかけあわせる
5. 4. の結果を足し合わせ、1. の整数部分とあわせる

3.

$$\begin{array}{cccc} \frac{1}{2^1} & \frac{1}{2^2} & \frac{1}{2^3} & \frac{1}{2^4} \\ 0 & . & 1 & 1 & 0 & 1 \end{array}$$

4.

$$\begin{array}{cccc} \frac{1}{2^1} & \frac{1}{2^2} & \frac{1}{2^3} & \frac{1}{2^4} \\ \times & \times & \times & \times \\ 0 & . & 1 & 1 & 0 & 1 \\ & & & & & || \\ & & & & & \frac{1}{2^1} & \frac{1}{2^2} & 0 & \frac{1}{2^4} \end{array}$$

5.

$$\begin{array}{c} \boxed{\frac{1}{2^1} \quad \frac{1}{2^2} \quad 0 \quad \frac{1}{2^4}} \\ \text{足し合わせる} \\ || \\ 0.5 + 0.25 + 0.0625 = \boxed{0.8125} \\ \text{1. の整数部分とあわせる} \\ \downarrow \\ 5.8125 \end{array}$$

やってみよう!

- 2進数1.101を10進数に変換
(2010年度ITパスポート春季試験問題)
 - 2進数に変換した時、有限小数で表現できる10進数は、
以下のうちどれか
 - 0.1
 - 0.2
 - 0.4
 - 0.5
- (2012年度ITパスポート秋季試験問題)

小数の表現方式

小数を表現する方法(p. 10)

- 固定小数点方式
- 浮動小数点方式

固定小数点方式[1](p. 10)

- 小数部分の桁数をあらかじめ決めておく方法
- 小数部分の桁数が n の場合: 2進数では、小数部分が $1/2^n$ 刻みで表現



n を大きくすると、それだけ小数部分を細かく表現可能

※ただし、実際コンピュータは小数も2進数で考えているが、
人間が考えるときの便宜上、10進数で考えることが多い

固定小数点方式[2](p. 10)

- 小数を表す桁数が決まっていると...

Ex. 小数点以下が2桁分と決まっていると...

$$(2 \div 1000)_{10} = (0.002)_{10} \rightarrow (0.00)_{10}$$

小数を正確に表現できない

何桁分の小数部分を持っているかは数値によって異なる
＝固定小数点方式で小数を表せる場合は少ない



浮動小数点方式

浮動小数点方式[1](p. 10)

■ 小数: $D \times 10^n$ と表現できる

■ Ex. 1: $0.5 = 5 \times 10^{-1}$

■ Ex. 2: $-0.0625 = -6.25 \times 10^{-2}$

■ Ex. 3: $0.0000000084 = 8.4 \times 10^{-9}$

どの数でも「 $\times 10^n$ 」の「10」の部分は同じ(実際のコンピュータでは「 $\times 2^n$ 」)



小数を「 $D \times 10^n$ 」の形と考え、「D」と「n」だけ記憶しておく

浮動小数点方式[2](p. 10)

■ 浮動小数点方式:

小数を「 $D \times 10^n$ 」と考え、「 D 」と「 n 」を記憶することで小数を表す方式

Ex.

- $D = 6.25, n = -3$ の場合: 0.00625
- $D = 6.25, n = -2$ の場合: 0.0625
- $D = 6.25, n = -1$ の場合: 0.625
- $D = 6.25, n = 0$ の場合: 6.25

n の数値が何かで、小数点が仮数の中を動くように見えるから「浮動小数点」と名づけられた

D : 仮数部
 n : 指数部
と呼ぶ

- 仮数部
 - ✓ 符号は「0」が「+」、「1」が「-」
 - ✓ 固定小数点方式
- 指数部
 - ✓ 符号は「0」が「+」、「1」が「-」(ただし、2の補数表現とは別の特殊な形で表現される)

大きな数の表現(p. 10)

■ 浮動小数点方式を利用して表現

Ex.

- $2000000000000 = 2 \times 10^{12}$
- $-4250000000000000000 = -4.25 \times 10^{17}$

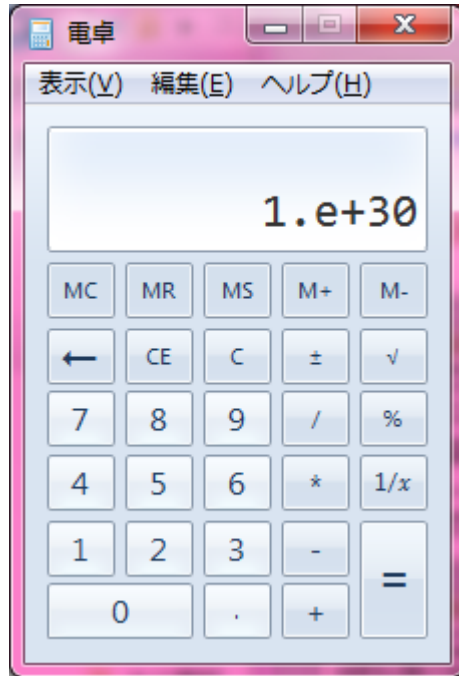
指数部が「+」の数になる



コンピュータは「2」と「+12」, 「-4.25」と「+17」を記憶しておく
(実際には、「 $\times 10^n$ 」ではなく「 $\times 2^n$ 」で表現)

大きな数の表現[例](p. 10)

- Windowsの電卓のあるモード(オーバフローをなかなかしないモード)で...



$$\text{Ex. } 10000000000000000000 \times 10000000000000000000 \\ = 1.e+30$$

1.0×10^{30} という意味
(浮動小数点方式での表現)

※オーバフローしにくいモードは、大きな数を浮動小数点方式で表現する
= それだけたくさんの桁がある数を表現できるので、オーバフローしにくい

桁落ち(1)

- 小数部分が無限のものを扱えるわけではない

- 例えば割り算で割り切れない数や円周率

➡ 小数部分を適当なところで切り捨てる(四捨五入ではない)

例えば... $1 \div 6$:

コンピュータは「0.16666...666」と考える

本来はこの後も無限に続く

コンピュータが扱える小数の桁数:
「有効桁数」と呼ぶ

桁落ち(2)

小数部分が無限のものは適当なところまでで切り捨てられる

本来の数よりも、小数の桁数が小さくなってしまう現象

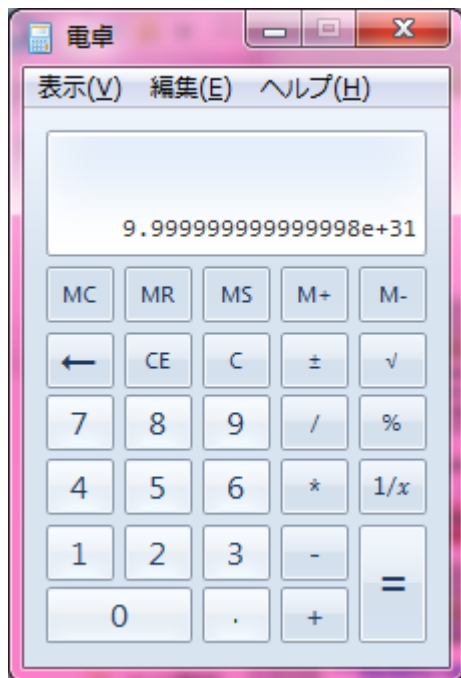


桁落ち

桁落ちの例

■ Windowsの電卓のあるモード(オーバーフローをなかなかしないモード)で...

Ex. 電卓での計算: $99999999999999999 \times 99999999999999999$
→ $9.9999999999999998e+31 = 9.9999999999999998 \times 10^{31}$
= $999999999999999980000...00000$



でも実際に計算すると...

$$\begin{array}{r} 99999999999999999 \\ \times) 99999999999999999 \\ \hline \dots 91 \\ +) \dots 91 \\ \hline \dots 01 \end{array}$$

桁落ち

つまり本当の計算では...

$$\begin{array}{l} 99999999999999999 \times 99999999999999999 \\ = x.xxxx\dots 01 \end{array}$$

やってみよう!

- スマートフォンの電卓で、大きな数のかけ算をしてみよう!
 - 結果が浮動小数点表示になるか?
 - 桁落ちしているか?
- を確認しよう!

桁落ちが起こると...

- 数が本来の数よりも小さくなってしまう
 - 微妙な数値が必要な場合には要注意
- 桁落ちをした数に大きな数をかけると、本来の数に大きな数をかけたときとの差が大きくなる

例えば...有効桁数が小数点第3位とすると、 $1 \div 6 \times 100000$ の計算は...

- 「 $1 \div 6$ 」の時点で0.166に桁落ちし、それに100000をかけて、16600
- 1×100000 を6で割ると(計算の順序を変えると)、16666.666
- 本来の $1 \div 6$ に100000をかけると、16666.66666666....

- 
- コンピュータで計算をするときは、計算の順番に注意
(割り算はなるべく後にすること)
 - 例えば「 $1 \div 6 \times 100000$ 」の計算は、「 1×100000 」をしてから6で割る

やってみよう!

1. 0.0000055を浮動小数点方式で表現
2. 0.0000000001234を浮動小数点方式で表現
3. 4560000000000000を浮動小数点方式で表現
※3つとも仮数部は小数点第2位的小数とすること
4. $(10 \div 7) \times 10000$ を計算
 - 小数点第2位までが有効桁数
 - 桁落ちを考えて計算すること
5. $10 \div 7 \times 10000$ を計算
 - 小数点第2位までが有効桁数
 - 桁落ちの影響がなるべく少ないように計算すること

Question!

文字の表現

文字の符号化(p. 13)

- 文字: コンピュータは整数に置き換えて扱う(番号をつけて扱う)
 - 文字を2進数で表現する(「**符号化**」と呼ぶ)
- 2進数で表現される文字集合
 - 半角英数文字
 - 図形文字
 - 制御文字
 - 多バイト文字
 - 図形文字

※文字集合: 文字の集まり

図形文字と制御文字(p. 13)

- **図形文字**: 通常、画面に表示される文字
 - 人間が明示的に書いたり読んだりする文字
 - アルファベット, 数字, ひらがな, 漢字, 記号, etc.
- **制御文字**: 通常、画面に表示されない文字
 - コンピュータに何らかの制御をするための文字
 - 改行, TAB, ESC, etc.



ASCII文字

ASCII文字(p. 13)

- **ASCII**: American Standard Code for Information Interchange

- 半角文字を表す文字集合

- アルファベット大文字(26文字)
- アルファベット小文字(26文字)
- 数字(10文字)
- 記号(スペース, 「,」, 「.」, etc.)

1文字を表すために、最低限7ビット必要
(6ビット: 64種類の情報, 7ビット: 128種類の情報)

※1文字を表す2進数の桁数(ビット数)は、どの文字でも同じ(つまり7ビット)

図形文字(p. 13)

- ASCII: 情報量が7ビットで収まるように、扱う文字を取り決めた文字集合
 - 図形文字: 95文字
 - アルファベット(大文字・小文字): 52文字
 - 数字: 10文字
 - 記号(スペースを含む): 33文字
 - 制御文字: 33文字
 - BackSpace, Delete, Tab, 改行(CRとLF), etc.

番号例(p. 14)

番号	文字	番号	文字	番号	文字
47	0	65	A	97	a
48	1	67	B	98	b
49	2	68	C	99	c
50	3	69	D	100	d
51	4	70	E	101	e
52	5	71	F	102	f
53	6	72	G	103	g
54	7	73	H	104	h
55	8	74	I	105	I
56	9	75	J	106	j

「数」としての0～9ではなく、「文字」としての0～9

ビット数[1](p. 14)

- コンピュータでは8ビットを1つの単位として扱うことが多い
 - ASCII文字も8ビットで表現すると扱いやすい
- 8ビットのうち、7ビット分(2進数で7桁目まで)で文字を表現する
- 残りビット(2進数で8桁目)に常に0を入れておく
 - ASCII文字としては無駄なビット
 - 日本語を表現するときに利用

例えば...

A: 65番(10進数)

= 1000001番(2進数)

= 01000001番(2進数, コンピュータ内での表現)



ASCII的には無駄な(何も利用していない・1にはならない)ビット

ビット数[2](p. 14)

■ 8ビットで1文字を表現 = 1バイトで1文字を表現

「1バイト文字」と呼ばれる

Ex. 「Hello, my name is John.」

- アルファベット: 17文字
- 記号: 2文字
- スペース: 4文字

} 23文字 = 23バイト

ちなみに...

- アスキーアートも文字コードのASCIIから(ASCII art)
 - アスキーアート: 文字だけで作った絵
 - 感情を表す「(^_^);」のような単純なものから、人や動物に見えるものまで様々
- アスキーアートの例
 - <http://ja.wikipedia.org/wiki/%E3%82%A2%E3%82%B9%E3%82%AD%E3%83%BC%E3%82%A2%E3%83%BC%E3%83%88>
 - <http://bhdaa.sakura.ne.jp/zukan/>



Question!

多バイト文字

背景[2](p. 15)

- 様々な言語圏の文字: 英語圏の文字と同様に2進数で表現する必要性
 - 英語圏の文字: 128文字で表現可能
 - 1バイト分(256文字分)のうち、128文字分は英語圏の文字
 - 英語圏以外の文字: 128文字以上必要な場合も
 - 日本語
 - 中国語
 - 韓国語
 - etc.

→ 128文字では収まらない

1文字を複数のバイト(多バイト)で表現

文字化け(p. 15)

- 多バイト文字の出現により、文字化けが発生
- 文字化けの原因
 - フォントの問題
 - 文字集合の符号化方式の問題

「文字コード」と呼ぶ

詳細な理由は何であれ...

要は文字を表す2進数(0と1の並び)を、コンピュータが理解していないために発生

- その2進数をどのような形でディスプレイに表示して良いかをコンピュータが理解していないため

フォントの問題[3](p. 15)

- **機種依存文字**: コンピュータによって表現のしかたが違う文字
 - それぞれの文字を表現するビット列が、コンピュータによって異なる
 - 1文字1文字を表現するビット列は、JIS(日本の国家規格)などで決まっている
→ コンピュータの環境に依存しない
 - 規格で決められた文字に含まれていない文字もある
- **機種依存文字**
 - Ex. 丸付き数字(①, ②, ...), ローマ数字(Ⅰ, Ⅱ, ...), etc.
- **外字**: 登録されていない文字を、利用者が作ったもの
 - 人名漢字などを作ることが多い
 - 作ったコンピュータでしか使えない

符号化方式の問題[1](p. 15)

- 符号化: 1つの文字を2進数(ビット列)として表現すること
- ある1つの文字を表現するビット列が複数通り存在する場合
 - 半角英数の文字はASCIIの1通りだけ
 - 他にも存在するが、ASCIIが世界標準
 - 大部分のコンピュータはASCIIを利用
 - 日本語は複数通り存在

日本語の符号化方式

日本語の文字(p. 15)

■ ASCII

- 1文字を8ビットで表現→全部で256文字分表現可能
- 現状で128文字存在(128文字分利用されているので残りは128文字)

■ 日本語

- ひらがな: あ～ん(ゐ, ゑなどの旧字を含む), 濁音・半濁音, 小文字(「ぁ」「ぃ」など)
- カタカナ: ア～ン(ヰ, ヱなどの旧字を含む), 濁音・半濁音(ヴを含む), 小文字(「ァ」「ヱ」「カ」「ケ」など)

169文字

ひらがな・カタカナだけでもASCIIでは表現できない

日本語文字集合の規格(p. 16)

■ 現状での日本語文字集合の規格: JIS X 0208:1997

- ひらがな・カタカナ・漢字・非漢字文字で6879個

JIS第1水準(使用頻度の高い漢字): 2965個
JIS第2水準(使用頻度の低い漢字): 3390個

- $2^{13} = 8192$ なので、13ビットで表現可能
- コンピュータ処理では、バイト単位(8ビット単位)が好都合

16ビット(2バイト)で日本語1文字を表現

ASCII文字との区別(p. 16)

■ 日本語の文書

- 日本語の2バイト文字
- ASCIIの1バイト文字

混在

日本語の2バイト文字(JIS X 0208)とASCIIの1バイト文字は区別する必要
(1つの文書の中で、どれが2バイト文字でどれが1バイト文字か)



- モード切り替えによる区別方法
- ASCII文字の番号を避ける区別方法

モード切り替え(p. 16)

■ 文字集合切り替えのための特別な記号を用意

- ここから先はASCII文字
- ここから先は日本語文字
- ここから先は中国語漢字
- etc.

エスケープシーケンス

通常の文書では頻繁に文字集合が切り替わることがなく、同じ文字集合に属する文字が現れることが多いという性質を利用

- 国際標準規格: ISO-2022
- 日本語に適用したもの: ISO-2022-JP

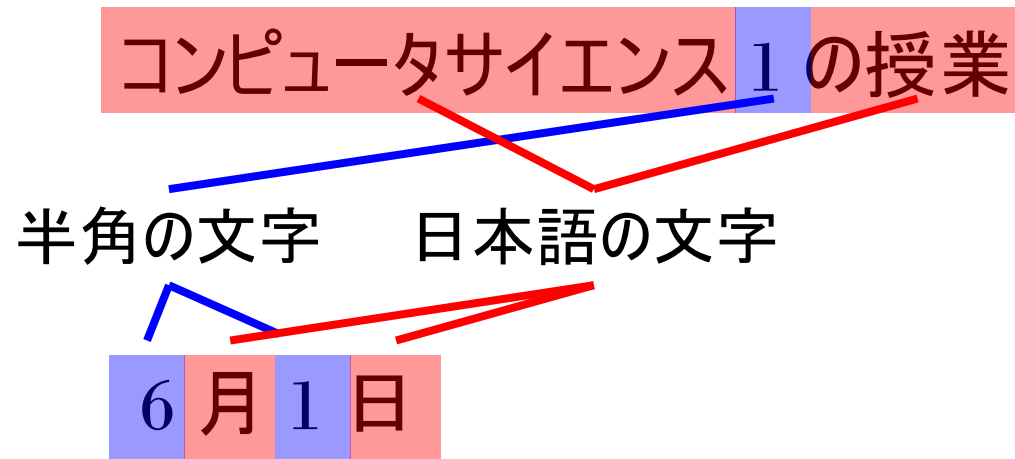
ISO-2022-JPの例(p. 16)

ESC \$B	F	K¥	\$N	ESC (B	JP	ESC \$B	\$@	!#	ESC (B	¥n
	日	本	は		JP		だ	。		

- 「ESC \$B」や「ESC (B」、「¥n」などがエスケープシーケンス
- 「F|」や「K¥」、「\$N」などは、2バイト文字をASCII文字で表現した場合の文字
(2バイト文字は、1バイト文字2文字の組み合わせで表現できる)

モード切り替えの考え方[1](p. 16)

- 同じ文字集合に属する文字が続いて現れることが多い



- 上側の文章: 日本語文字がいくつか続いた後、半角文字が少しあり、また日本語文字が続く
- 下側の文章: 日本語文字と半角の文字が交互にある

頻繁に文字集合が切り替わるわけではない(ある言語と別の言語の文字が1文字ずつ交互に出てきたり、ということは少ない)
→ 文字集合がどこで切り替わっているか、わかるようにしておけば良い

モード切り替えの問題(p. 17)

- 文書在先頭から順番に見ていく場合には問題ない
- 文書を途中から見ていくときに問題が生じる
 - 見始めた途中の文字が、ASCII文字か日本語文字か、エスケープシーケンスかが判別できない

Ex. 見始めた途中の文字が「70」番だった場合

- ASCII文字の「F」?
- 日本語文字の一部?
- 韓国語の一部?



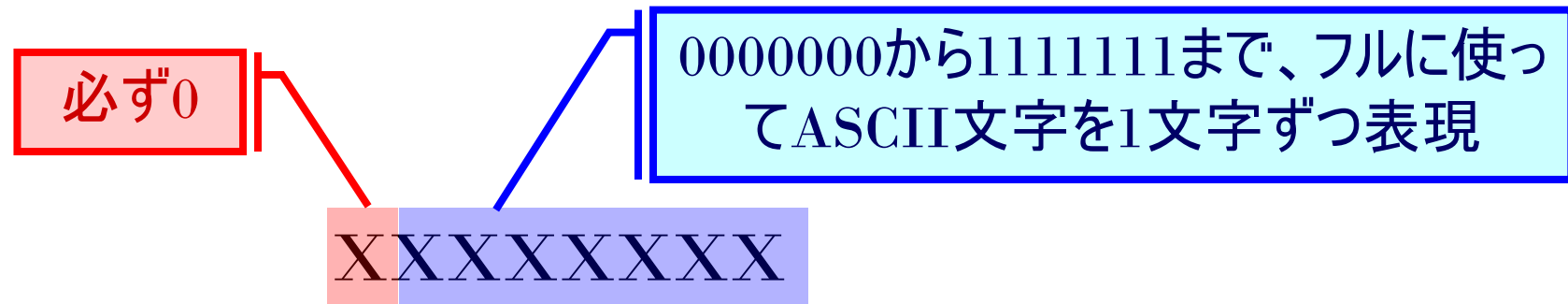
検索や置換などの文書処理に時間がかかる

ASCII文字の番号を避ける(p. 17)

- ASCIIで使われていない番号を2バイト文字の番号にあてる方法
 - EUC(日本語のものを**EUC-JP**)
 - 第1バイト(前半の8ビット)と第2バイト(後半の8ビット)両方でASCII領域が避けられている
 - SJIS(Shift JIS)
 - 第2バイト(後半の8ビット)ではASCII領域も使われている
- 文章のバイト数は、単純に、日本語文字で2バイト、ASCII文字で1バイトで数えれば良い

EUCとSJIS[1](p. 17)

- ASCII文字: 8個の0と1で、1文字分を表現
 - 実際には、7個の0と1で1文字分を表現
 - 8ビット目は必ず0



= ASCII文字は、0XXXXXXX という形
Ex. 「a」を0と1で表現すると: 01100001



8ビット目が1になる番号(1XXXXXXXという形の番号)は
ASCII文字ではない

EUCとSJIS[2](p. 17)

- ASCIIで使われていない番号を2バイト文字の番号にあてる方法
 - EUC(日本語のものを**EUC-JP**)
 - 第1バイト(前半の8ビット)と第2バイト(後半の8ビット)両方でASCII領域が避けられている
 - SJIS(Shift JIS)
 - 第2バイト(後半の8ビット)ではASCII領域も使われている
- 文章のバイト数は、単純に、日本語文字で2バイト、ASCII文字で1バイトで数えれば良い

例えばある文章が...

00110110100101101010010101101010

1から始まっているから日本語文字

0から始まっているからASCII文字

Webやメールでの文字化け(p. 18)

■ Webページや電子メール

- 文字コードの指示が文書中に書かれている場合

「charset=iso-2022-jp」や「charset=Shift_JIS」など

➡ ソフトウェアは指示通りに文字コードを解釈し、表示

- 文字コードの指示が文書中に書かれていない場合

➡ ソフトウェアは文書のデータの特徴から文字コードを判別



文字化けが発生する場合

- 文書中の文字コードの指示が間違っている場合
- 文字コードの指示がなく、文字コードを判別できなかった場合

Unicode

言語圏ごとの文字コード(p. 18)

- これまでの多バイト文字の扱い: 異なる言語圏ごとに文字集合を作成
様々な文字集合ができてしまって不便
 - コンピュータネットワークの国際化が進んだ
 - コンピュータの資源が豊富になった



国際文字集合規格として各文字集合を統一化

- ASCII, ラテン文字, 日本語, 韓国語, 中国語, ベトナム語, ギリシャ文字, 記号, etc.

Unicode: どの文字を扱うかと文字の符号化の方法を決めた規格

- UCS(Universal multi-octet coded Character Set)でどの文字を扱うかを規定
- UTF(UCS Transformation Format)で文字の符号化の方法を規定

UTF-8(p. 18)

- Unicodeでの代表的な符号化方式(符号化方式はいくつか存在)
- 1文字を1～6バイトの可変長(文字によってバイト数が異なる)で符号化する方式
 - ASCIIやISO-2022-JP、Shift JIS、EUC-JPは1文字を同じバイト数で表現
- OS(WindowsやMacなどのオペレーティングシステム)でファイル名などの内部処理に利用
 - 半角英数を符号化した結果が、ASCII文字と全く同じになるため、従来のシステムと相性が良い

現在、UTF-8への移行が急速に進んでいる

- ただし、以前からのファイルを移行するのは大変なので、完全移行には時間がかかる
- 完全移行できたら、文字化けが起こらなくなる