

システムプログラム 第3回

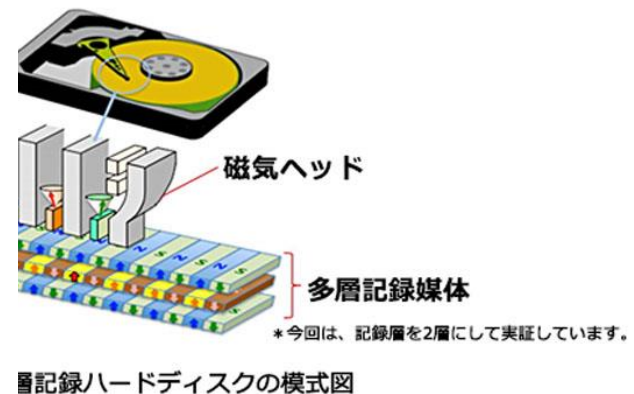
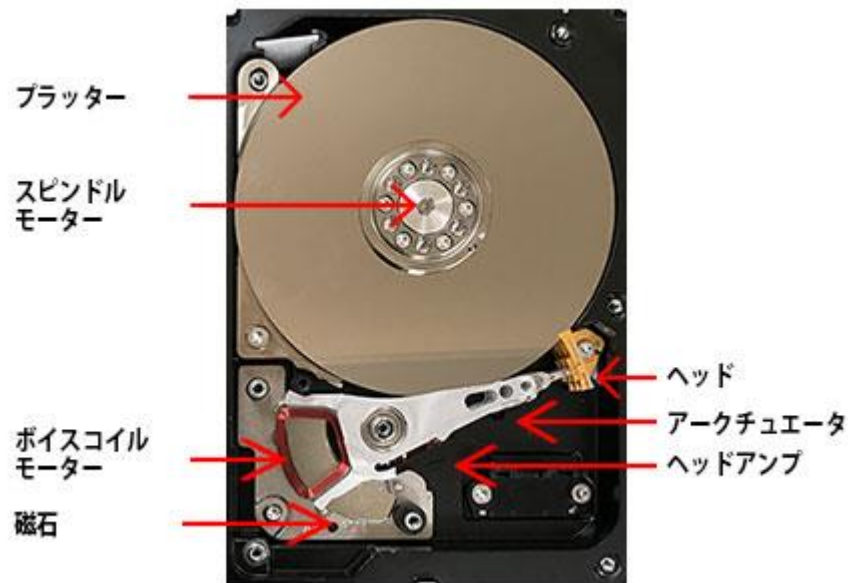
創域理工学部 情報計算科学科

松澤 智史

計算機内部の情報表現

- 現代の計算機は情報を2値の信号として格納している

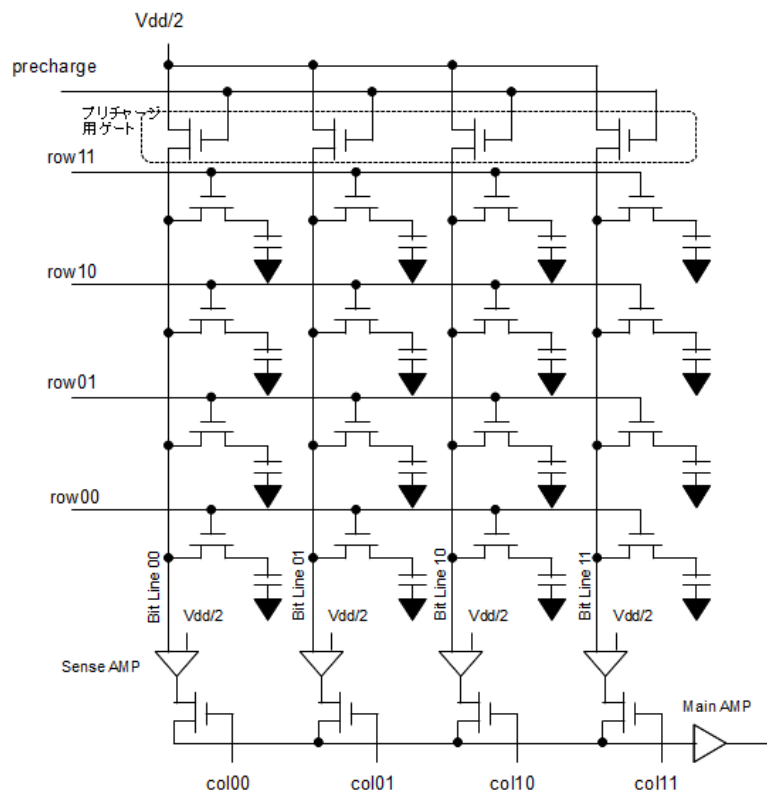
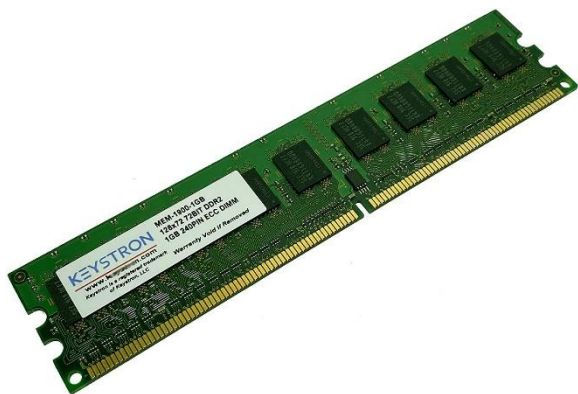
例:HDD



計算機内部の情報表現

- 現代の計算機は情報を2値の信号として格納している

例: DRAM



コンデンサに電荷がある場合は1 ない場合は0

ビット

- 二つの数値に過ぎない「ビット」がデジタルの基礎
 - 10本の指を持つ人間は10進表現を使うことが一般的
 - 情報を格納して処理する機械は2進表現が効果的
- 複数のビットをグループ化し、様々なビットパターンに何かの**解釈**を与えることで、任意の有限集合の要素を表現
 - ビットは一つだけでは大した価値を持たない
 - 数値としての解釈
 - 文字としての解釈
 - 画像としての解釈
 - ブール値としての解釈 など
- 本日はビットパターンの解釈(表現)について学ぶ

ちょっと実験

```
~/work/3IS/SystemProgram
tusedls11$ cat int.c
#include <stdio.h>

int main(){
    int n = 200;
    printf("%d\n",n);
    n = n * 300;
    printf("%d\n",n);
    n = n * 400;
    printf("%d\n",n);
    n = n * 500;
    printf("%d\n",n);
}
tusedls11$ gcc int.c -o int
tusedls11$ ./int
200
60000
24000000
-884901888
tusedls11$
```

- $200 * 300 * 400 = 24000000$
- $200 * 300 * 400 * 500 = -884901888 \quad ! ?$

数値としての解釈

- 数の表現方法: 3種類のエンコーディング(解釈のルール)
 - 符号なしエンコーディング
 - 符号ありエンコーディング(2の補数エンコーディング)
 - 浮動小数点エンコーディング

符号なしとありのエンコーディング

8ビットの数で考える

ビット表現	10進表現
00000000	0
00000001	1
00000010	2
00000011	3
01111111	127
11111111	255

符号なし

$$127(01111111)+1(00000001) = 128(10000000)$$

$$255(11111111)+1(00000001) = 0(00000000)$$

ビット列で見た場合は符号あるなし関係ないが、
計算機上のエンコーディングに則った
演算結果(表記)となる

ビット表現	10進表現
00000000	0
00000001	1
00000010	2
00000011	3
01111111	127
11111111	-1

符号あり

$$127(01111111)+1(00000001) = -128(10000000)$$

$$-1(11111111)+1(00000001) = 0(00000000)$$

最初の1ビットが0の場合は0以上の値
最初の1ビットが1の場合は負の値
最初の1ビットは**符号ビット**とも呼ばれる

2の補数

- 補数

- ある基数法において、ある自然数 a に足したとき桁が1つ上がる (桁が1つ増える) 数のうち最も小さい数をいう
- 符号ありの場合は負の数を求めることになる

- 2の補数の求め方

- 10010010をビット反転→01101101(1の補数)
- 1の補数に+1する 01101110

- 例

- $39(00100111) \rightarrow -39(11011001)$
- $1(00000001) \rightarrow -1(11111111)$
- $39(00000000 \ 00000000 \ 00000000 \ 00100111) \rightarrow$
 $-39(11111111 \ 11111111 \ 11111111 \ 11011001)$

余談：負の数の考え方

$0001 = 1_{10}$
では -1 はどうする？

符号ビットで正負をつける
 $1001 = -1_{10}$ とする

数なので演算可能でなければならない
 $1_{10} + (-1_{10}) = 0001 + 1001 = 1010 = -2_{10}$?
場合分けする？→面倒

反転を使う
 $0001 = 1_{10}$, $1110 = -1_{10}$ とすると $1_{10} + (-1_{10}) = 0001 + 1110 = 1111$
 $1111 = 0_{10}$ とするならばこれで良い(1の補数の考え)
ただし, 0000 と 1111 の2種類の 0_{10} が存在する

反転した数+1を使う
 $0001 = 1_{10}$, $1111 = -1_{10}$ とすると $1_{10} + (-1_{10}) = 0001 + 1111 = [1]0000$
これが2の補数

C言語での符号ありなし整数型

- 8ビット整数型
 - char (符号あり)
 - unsigned char (符号なし)
- 32ビット整数型
 - int (符号あり)
 - unsigned int (符号なし)

```
~/work/3IS/SystemProgram
tusedls11$ cat int2.c
#include <stdio.h>

int main(){
    char n = 127;
    n = n + 1;
    printf("%d\n",n);
}
tusedls11$ gcc int2.c -o int2
tusedls11$ ./int2
-128
tusedls11$
```

```
~/work/3IS/SystemProgram
tusedls11$ cat int2.c
#include <stdio.h>

int main(){
    unsigned char n = 127;
    n = n + 1;
    printf("%d\n",n);
}
tusedls11$ gcc int2.c -o int2
tusedls11$ ./int2
128
tusedls11$
```

余談: 16進表現

- ビット表現(2進表現)は冗長
 - 10進表現はビット表現との相互変換が面倒
- 計算機内の情報表現では16進表現(16進記法)が好まれる

ビット表現	10進表現	16進表現
00000000	0	00
00000001	1	01
00001111	15	0F
00010001	17	11
01111111	127	7F
11111111	255	FF

浮動小数点エンコーディング

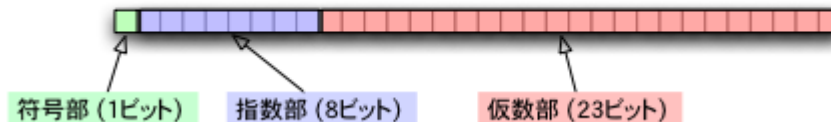
- 符号部, 指数部, 仮数部からなるエンコーディング

- float型

- 4バイト(32ビット)

- float の表す値 = $(-1)^{\text{符号部}} \times 2^{(\text{指数部}-127)} \times 1.\text{仮数部}$

float 型 (4バイト=32ビット) の内部表現

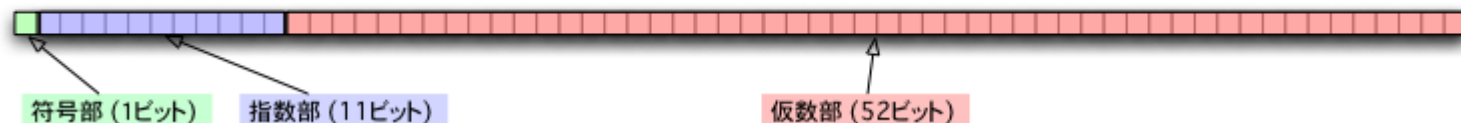


- double型

- 8バイト(64ビット)

- double の表す値 = $(-1)^{\text{符号部}} \times 2^{(\text{指数部}-1023)} \times 1.\text{仮数部}$

double 型 (8バイト=64ビット) の内部表現



浮動小数点エンコーディング(例)

2.0の場合

- float型

0 10000000 000000000000000000000000
(符号部0 指数部128 仮数部0)

- double型

0 100000000000 000000...000000000000
(符号部0 指数部1024 仮数部0)

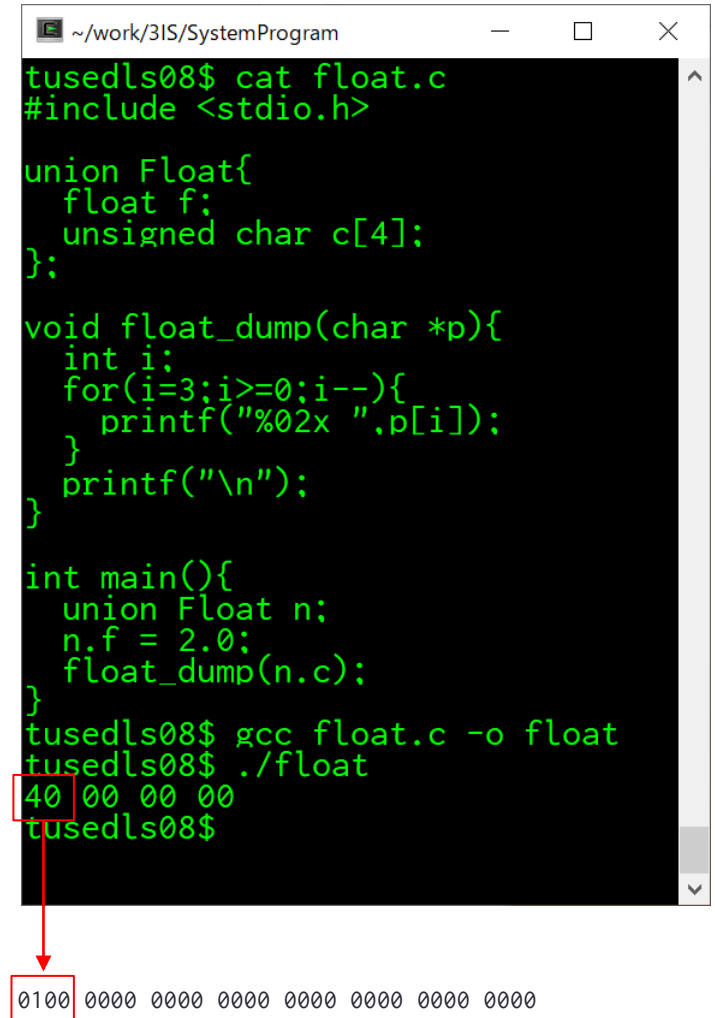
0.5の場合

- float型

0 01111110 000000000000000000000000
(符号部0 指数部126 仮数部0)

- double型

0 011111111110 000000...000000000000
(符号部0 指数部1022 仮数部0)



The image shows a terminal window titled `~/work/3IS/SystemProgram`. It contains the following C code:

```
tusedls08$ cat float.c
#include <stdio.h>

union Float{
    float f;
    unsigned char c[4]:
};

void float_dump(char *p){
    int i;
    for(i=3;i>=0;i--){
        printf("%02x ",p[i]);
    }
    printf("\n");
}

int main(){
    union Float n;
    n.f = 2.0;
    float_dump(n.c);
}

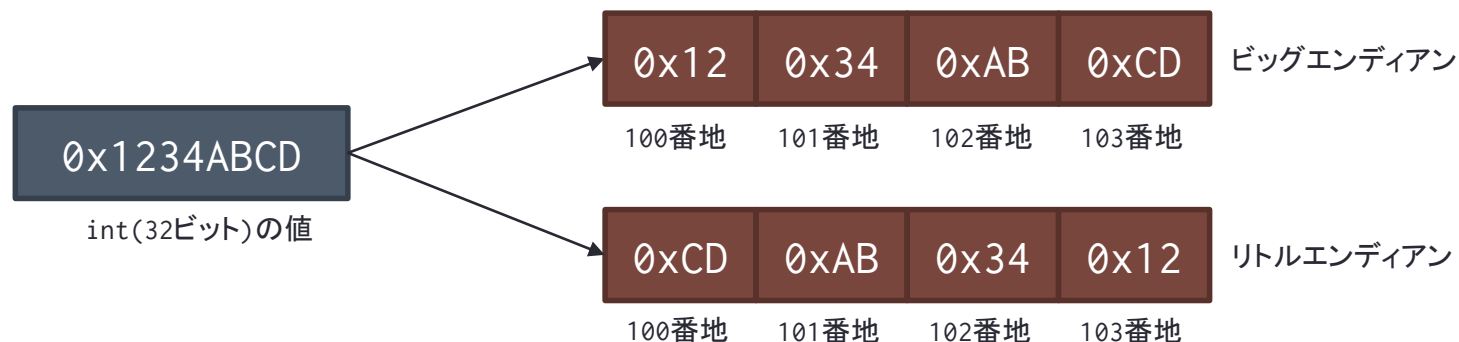
tusedls08$ gcc float.c -o float
tusedls08$ ./float
40 00 00 00
tusedls08$
```

Below the terminal window, the memory representation of the float value 2.0 is shown as a sequence of bytes: `0100 0000 0000 0000 0000 0000 0000 0000`. A red arrow points from the first byte `40` in the terminal output to the first byte `0100` in the memory representation below.

余談: バイト・オーダー

多バイトをメモリに格納する際に、格納する順番の違い

- ビッグエンディアン
 - 最上位ビットの属するバイトを低位のアドレスへ格納していく方式
 - SPARC(旧Sun Microsystems), PowerPC(リトルもサポート), Cell
- リトルエンディアン
 - 最下位ビットの属するバイトを低位のアドレスへ格納していく方式
 - Intel, AMD



ビッグエンディアンは人間が理解しやすい
リトルエンディアンは計算機が操作しやすい



文字列

- ビット列を文字列エンコーディングで解釈する

ASCIIコード

	0	1	2	3	4	5	6	7
0	NUL	DLE	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

```
~/work/3IS/SystemProgram
tusedls08$ cat String.txt
Hello
tusedls08$ hexdump String.txt
00000000 6548 6c6c 0a6f

00000006
tusedls08$
```

```
~/work/3IS/SystemProgram
tusedls08$ hexdump -C String.txt
00000000  48 65 6c 6c 6f 0a                                |Hello.|
00000006
tusedls08$
```

見にくい場合は `-C` オプションを使う

日本語文字列

- UTF-8 (Windows, Mac, Linux 最近の標準エンコーディング)
 - 日本語の平仮名, カタカナ, 漢字などは3バイトで1文字を表現
 - あ(e38182) い(e38184) う(e38186) 松(e69dbe) 澤(e6bea4)

```
~/work/3IS/SystemProgram
tusedls08$ cat String2.txt
あいう松澤
tusedls08$ hexdump String2.txt
00000000 81e3 e382 8481 81e3 e686 be9d bee6 0aa4
00000010
tusedls08$
```

UTF
UCS Transformation Format
UTF-8 UTF-16 UTF-32などがある
ASCIIと透過性があるのはUTF-8

- ASCIIと共存可能(UTF-8の1バイト文字)
 - ASCIIは先頭4ビットが0-7
 - 先頭4ビットがA,B,C,Dの場合は2バイト文字(ラテン文字, ギリシア文字など)
 - 先頭4ビットがEの場合は3バイト文字
 - 先頭4ビットがFの場合は4バイト文字(絵文字, 顔文字など)

```
~/work/3IS/SystemProgram
tusedls08$ cat String3.txt
今4時
tusedls08$ hexdump String3.txt
00000000 bbe4 348a 99e6 0a82
00000008
tusedls08$
```

今(e4bb8a) 4(34) 時(e69982)

日本語文字列 その2

- SJIS(Shift-JISコード 日本語対応 Windows標準)
 - 日本語の平仮名, カタカナ, 漢字などは2バイトで1文字を表現
 - あ(82a0) い(82a2) う(82a4) 松(8fbc) 澤(e056)

```
~/work/3IS/SystemProgram
tusedls08$ cat String2.txt
あいう松澤
tusedls08$ nkf -s String2.txt > String2_sjis.txt
tusedls08$ hexdump String2_sjis.txt
00000000 a082 a282 a482 bc8f 56e0 000a
0000000b
tusedls08$
```

文字コードをSJISに変換

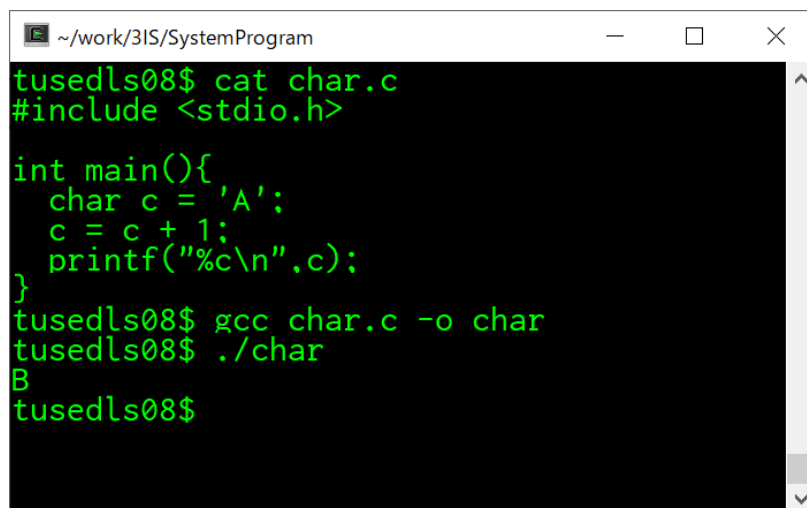
- UTF-8と違い日本語のみ
- UTF-8とは互換性なし
- 日本語文字1文字が2バイトと軽量
- ASCIIとは互換がある
 - 1バイト目が00~80 あるいはa0~df は1バイト文字と解釈
 - 1バイト目がa0~df は半角カナ文字

日本語文字列 その3

- その他の日本語文字エンコーディング
 - JIS(電子メール等で使用)
 - ASCIIと共存不可
 - 漢字IN 漢字OUTのコードがあり, 漢字OUTのコードの後はASCIIと解釈
 - EUC (Unix系OSで長らく使用)
 - 1バイト目が 00~7f はASCIIと解釈 8eの場合は半角カナの1バイト文字
- 余談:文字化け
 - エンコーディングによって同じ日本語文字でもビット列が異なる
 - 想定されていないエンコーディングで解釈した場合に文字化けが発生する

余談: C言語のchar型

- 1バイト文字(ASCII)を格納するためのCharacter型(8ビット)
- 中身はビット列(数値)なので, 演算が可能(言語による)

A terminal window titled "~/work/3IS/SystemProgram" with standard window controls. It shows a C program being compiled and executed. The code defines a char variable 'c' as 'A', increments it by 1, and prints it. The output shows 'B', demonstrating that char is treated as a numeric value.

```
~/work/3IS/SystemProgram
tusedls08$ cat char.c
#include <stdio.h>

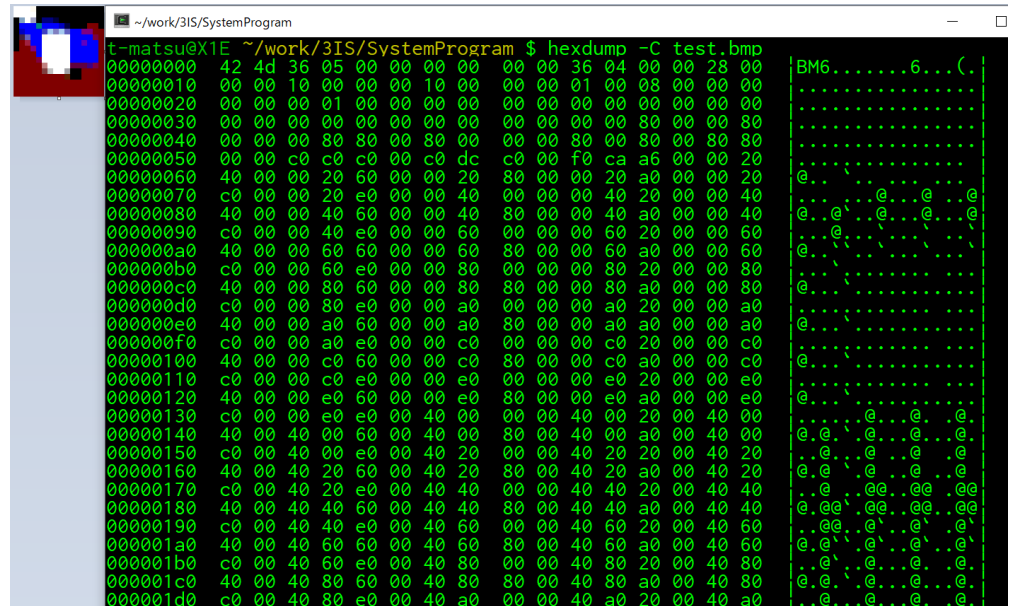
int main(){
    char c = 'A';
    c = c + 1;
    printf("%c\n",c);
}
tusedls08$ gcc char.c -o char
tusedls08$ ./char
B
tusedls08$
```

‘A’=0x41, ‘A’+1 = 0x42 = ‘B’

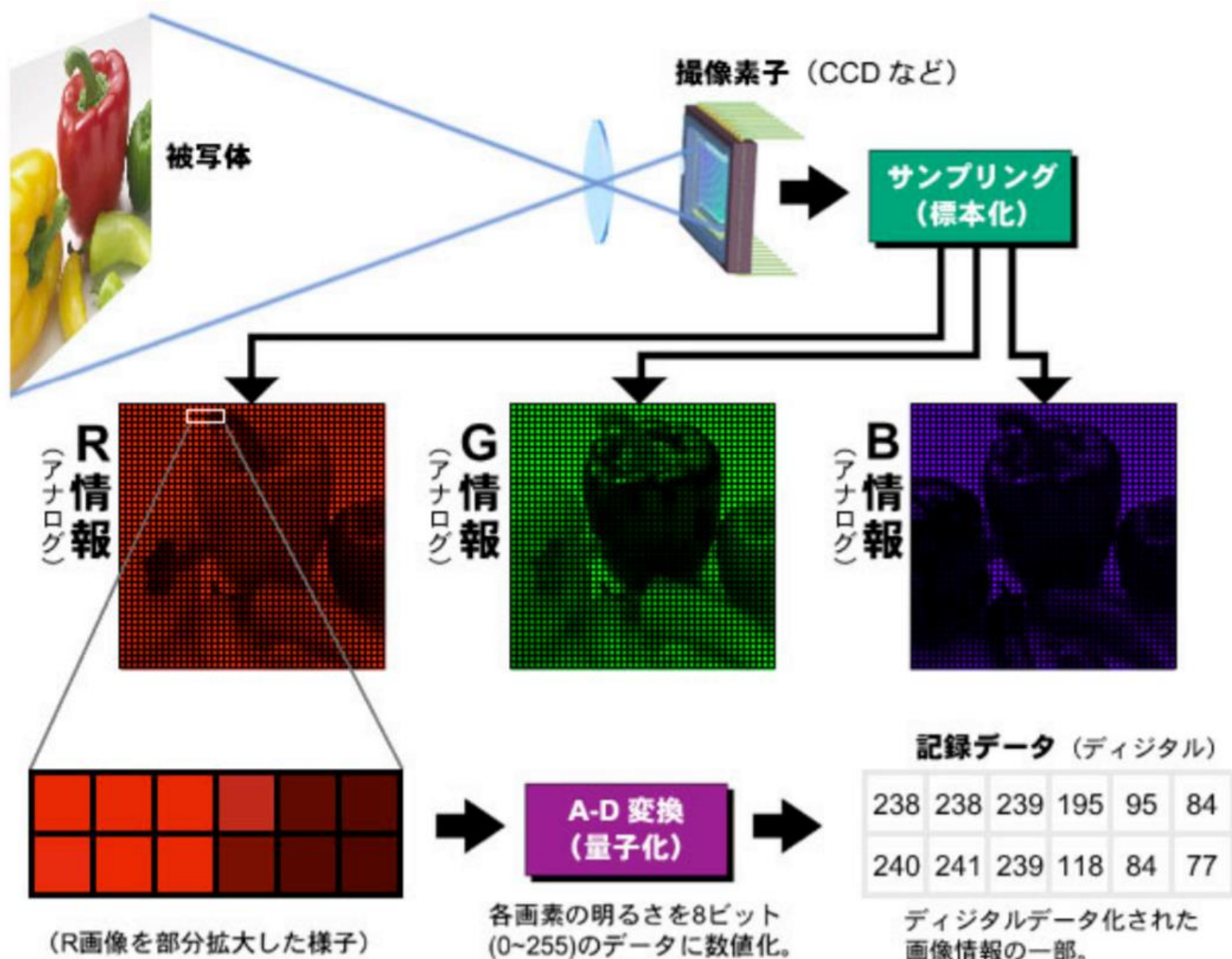
画像

- BMP

- Windowsで使われる標準的なフォーマット
- ファイルヘッダ: 14バイト
- 情報ヘッダ: 40バイト
- 画像データ: 可変長バイト

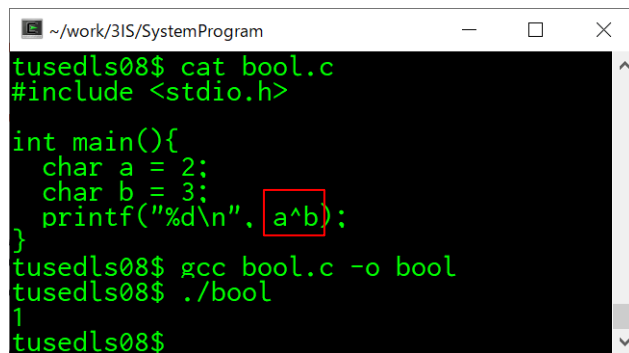


```
~/work/3IS/SystemProgram
t-matsu@X1E ~/work/3IS/SystemProgram $ hexdump -C test.bmp
00000000  42 4d 36 05 00 00 00 00 00 00 36 04 00 00 28 00  |BM6.....6...(.
00000010  00 00 10 00 00 00 10 00 00 00 01 00 08 00 00 00  |.....
00000020  00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00  |.....
00000030  00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 80  |.....
00000040  00 00 00 80 80 00 80 00 00 00 80 00 80 00 80 80  |.....
00000050  00 00 c0 c0 c0 00 c0 dc c0 00 f0 ca a6 00 00 20  |.....
00000060  40 00 00 20 60 00 00 20 80 00 00 20 a0 00 00 20  |.....
00000070  c0 00 00 20 e0 00 00 40 00 00 00 40 20 00 00 40  |.....
00000080  40 00 00 40 60 00 00 40 80 00 00 40 a0 00 00 40  |.....
00000090  c0 00 00 40 e0 00 00 60 00 00 00 60 20 00 00 60  |.....
000000a0  40 00 00 60 60 00 00 60 80 00 00 60 a0 00 00 60  |.....
000000b0  c0 00 00 60 e0 00 00 80 00 00 00 80 20 00 00 80  |.....
000000c0  40 00 00 80 60 00 00 80 80 00 00 80 a0 00 00 80  |.....
000000d0  c0 00 00 80 e0 00 00 a0 00 00 00 a0 20 00 00 a0  |.....
000000e0  40 00 00 a0 60 00 00 a0 80 00 00 a0 a0 00 00 a0  |.....
000000f0  c0 00 00 a0 e0 00 00 c0 00 00 00 c0 20 00 00 c0  |.....
00000100  40 00 00 c0 60 00 00 c0 80 00 00 c0 a0 00 00 c0  |.....
00000110  c0 00 00 c0 e0 00 00 e0 00 00 00 e0 20 00 00 e0  |.....
00000120  40 00 00 e0 60 00 00 e0 80 00 00 e0 a0 00 00 e0  |.....
00000130  c0 00 00 e0 e0 00 40 00 00 00 40 00 20 00 40 00  |.....
00000140  40 00 40 00 60 00 40 00 80 00 40 00 a0 00 40 00  |.....
00000150  c0 00 40 00 e0 00 40 20 00 00 40 20 20 00 40 20  |.....
00000160  40 00 40 20 60 00 40 20 80 00 40 20 a0 00 40 20  |.....
00000170  c0 00 40 20 e0 00 40 40 00 00 40 40 20 00 40 40  |.....
00000180  40 00 40 40 60 00 40 40 80 00 40 40 a0 00 40 40  |.....
00000190  c0 00 40 40 e0 00 40 60 00 00 40 60 20 00 40 60  |.....
000001a0  40 00 40 60 60 00 40 60 80 00 40 60 a0 00 40 60  |.....
000001b0  c0 00 40 60 e0 00 40 80 00 00 40 80 20 00 40 80  |.....
000001c0  40 00 40 80 60 00 40 80 80 00 40 80 a0 00 40 80  |.....
000001d0  c0 00 40 80 e0 00 40 a0 00 00 40 a0 20 00 40 a0  |.....
```



ブール代数

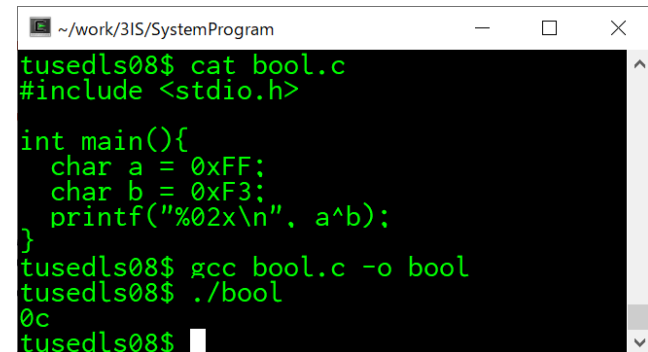
- 論理値の真・偽を{0,1}としてエンコード
- ビット列で情報を保持しているデータに対しても論理演算可能
 - NOT \sim (反転)
 - AND $\&$
 - OR $|$
 - XOR \wedge
 - $p = 0 \rightarrow \sim p = 1$
 - $p = 1, q = 1 \rightarrow p \& q = 1, p \wedge q = 0$



```
~/work/3IS/SystemProgram
tusedls08$ cat bool.c
#include <stdio.h>

int main(){
    char a = 2;
    char b = 3;
    printf("%d\n", a^b);
}
tusedls08$ gcc bool.c -o bool
tusedls08$ ./bool
1
tusedls08$
```

$2(00000010) \wedge 3(00000011) = 1(00000001)$



```
~/work/3IS/SystemProgram
tusedls08$ cat bool.c
#include <stdio.h>

int main(){
    char a = 0xFF;
    char b = 0xF3;
    printf("%02x\n", a^b);
}
tusedls08$ gcc bool.c -o bool
tusedls08$ ./bool
0c
tusedls08$
```

$0xff(11111111) \wedge 0xf3(11110011) = 0x0c(00001100)$

余談: &と&&, ~と! の違い

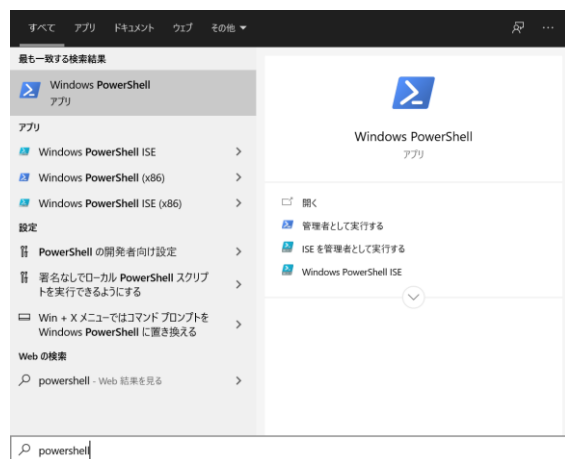
- \sim (NOT) $\&$ (AND) $|$ (OR) \wedge (XOR)は厳密にはビット・レベル演算子
- $!$ (NOT) $\&\&$ (AND) $||$ (OR) は論理演算子
- $0xff(11111111) | 0xf3(11110011) = 0x0c(11111111)$
- $0xff(11111111) || 0xf3(11110011) = 0x01(00000001)$
- $! 0xf3(11110011) = 0x00(00000000)$
- $\sim 0xf3(11110011) = 0x0c(00001100)$
- 論理演算子は0(偽)かそれ以外(真)として論理演算する
- 結果として真の場合は0x01を出力する

まとめ

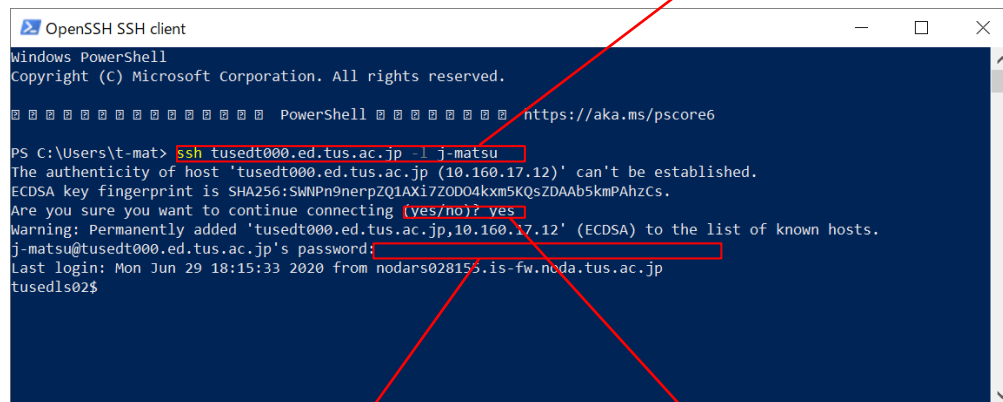
- コンピュータは情報をビットにエンコードし、ビットはバイト列として構成される
- 異なるエンコーディング(バイト列の解釈の方法)が、数値、文字列、画像・・・等に用いられる
 - 数値には大きく分けて以下の3つのエンコーディングがある
 - 符号ありエンコーディング
 - 符号なしエンコーディング
 - 浮動小数点エンコーディング
 - 文字列はASCII(1バイト)をベースとし、様々な文字への拡張エンコーディングが存在する
 - UTF-8 あらゆる文字を組み込む国際的なエンコーディング
 - SJIS 日本語文字対応エンコーディング
 - 他, JIS, EUC
 - 画像
 - BMP, JPG, PNG....など
 - ブール代数
 - 真偽を0,1と表現, 論理演算も可能

質問あればどうぞ

C言語の動作確認環境(Windows10 PS)



1. PowerShell起動

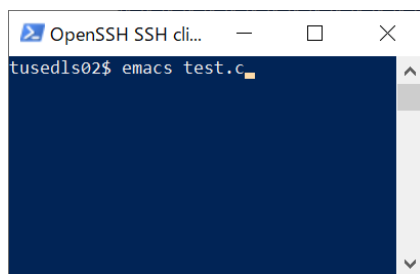


-1 ユーザ名(学生は学籍番号)

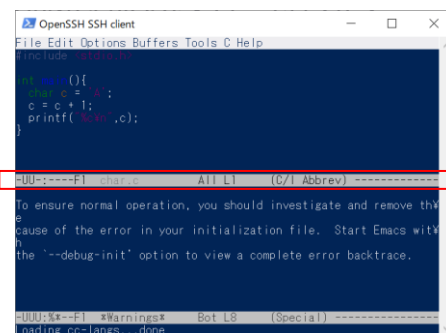
パスワードは入力しても表示されない(のぞき見防止)

初回のみ確認が出るので yes と入力

2. sshで大学のUNIX(CentOS)へリモートログイン

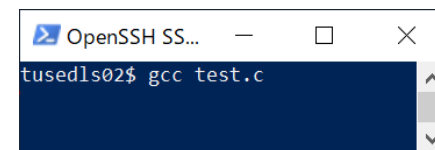


3. エディタ(emacs)で編集したいファイル名を指定



2画面モードを消したい場合は
Ctrl押ししながらxの後、
Ctrlを放して1を押す

4. 編集後保存(Ctrl押ししながらx sの順で押す) 5. エディタ終了(Ctrl押ししながらx cの順で押す)



6. コンパイル等をする 7. ログアウトはexitと入力 またはCtrl押ししながらdを入力