今回は、文字データの表現方法について学ぶ。

1. 原理

   文字をコンピュータで表現するためには、１年生で学んだ ASCII コード表のような表を用いる。ASCII コード表は英字、数字、記号しか含んでいなかったので、日本語や世界各国の文字を全て表すには不十分である。そこで、ASCII コード表の他に様々な文字コード表が用いられてきた。

   次の手順で文字コード表を決める。

   (a) 扱う文字の範囲を決める

   文字コード表に入れる文字の範囲を決める必要がある。決めた範囲に含まれる文字の集合を**文字集合 (character set)** と呼ぶ。

   (b) 文字に番号を付ける

   文字集合に含まれる全ての文字に番号を付ける。番号付けされた文字の集合を**符号化文字集合**と呼ぶ。

   (c) 番号をビット列に変換する方法を決める。

   文字に付けた番号と実際のビット（バイト）列との変換方法を決める。この変換方法のことを**文字符号化方式 (character encoding scheme)** と呼ぶ。よく使う文字が短いビット列になるように符号化したりする。複数の文字集合を同時に使用する時は、どの文字集合に属するか分かるように符号化方式を決める。

   どのコンピュータでも同じ文字データが扱えるように、符号化文字集合、文字符号化方式は ISO[1]、JIS[2] 等で規格化されている。しかし、国により、コンピュータの OS により、採用している規格が完全に同じではないため、どの規格か判定する際にミスがおこることがある。**文字化け**は、判定ミスが起きた結果である。

   Web ブラウザが文字符号化方式の自動判定に成功した例と失敗した例を図 1 に示す。

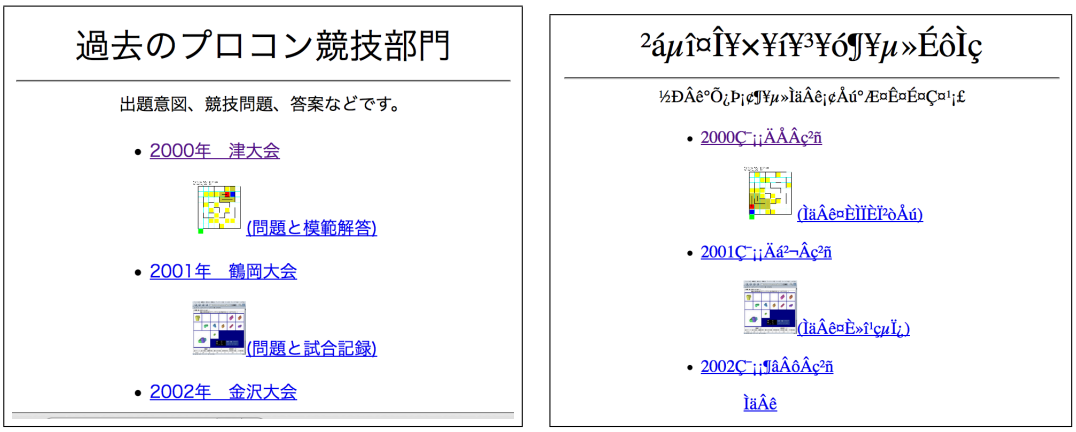


図 1: 文字化けの例

[1]ISO：International Organization for Standardization(国際標準化機構)
[2]JIS：Japan Industorial Standards(日本工業規格)

2. 日本で使用される符号化文字集合

(a) ASCII(American Standard Code for Infomation Interchange) コード表

図 2 に示す符号化文字集合 (character set) のこと。1963 年にアメリカ規格協会 (ASCII) が定めた。7 ビットで、制御文字、記号、数字、英字大文字、英字小文字を表現できる。制御文字と印刷可能文字あわせて 128 文字が含まれる。

(上位3ビット)

| (下位4ビット) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | NUL | DLE | (空白) | 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 |

印刷可能文字

*TAB: Tabulator(表作成) **機種やソフトによる

| | 意味 | C言語 | キー入力 |
|---|---|---|---|
| BEL(07H) | ベル(Bell) | ’\007’ | ^G |
| BS (08H) | バックスペース(Back Space) | ’\b’ | ^H [BackSpace] |
| HT (09H) | タブ(Horizontal TAB)* | ’\t’ | ^I [Tab] |
| LF (0AH) | 改行(Line Feed) | ’\n’ | ^J [Enter]** |
| FF (0CH) | 改ページ(Form Feed) | ’\f’ | ^L |
| CR (0DH) | 復帰(Carriage Return) | ’\r’ | ^M [Enter]** |
| ESC(1BH) | エスケープ(Escape) | ’\033’ | [Esc] |
| (20H) | 空白(White Space) | ’ ’ | [ ] |

代表的な制御文字

| ファイル | 通信回線：CR LF |
|---|---|
| Windows : CR LF<br>UNIX : LF<br>MacOS X : LF | プログラム中 ：LF(’\n’) |

改行の表現方法

図 2: ASCII 文字コード表

(b) JIS X 0201(通称 JIS 8 ビットコード)

図 3 に示す符号化文字集合のこと。1969 年に日本工業規格 (JIS) に定められた。7 ビットで、制御文字、記号、数字、英字大文字、英字小文字、カタカナを表現できる。制御文字と印刷可能文字あわせて 191 文字が含まれる。制御文字、記号、数字、英字大文字、英字小文字の範囲は、ASCII と、ほぼ、同じになている。

(上位4ビット)

| (下位4ビット) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NUL | DLE | (空白) | 0 | @ | P | ` | p | | | ▒ | ▒ | ▒ | ▒ | | |
| 1 | SOH | DC1 | ! | 1 | A | Q | a | q | | | ▒ | ア | チ | ム | | |
| 2 | STX | DC2 | ” | 2 | B | R | b | r | | | ▒ | イ | ツ | メ | | |
| 3 | ETX | DC3 | # | 3 | C | S | c | s | | | ▒ | ウ | テ | モ | | |
| 4 | EOT | DC4 | $ | 4 | D | T | d | t | | | ▒ | エ | ト | ヤ | | |
| 5 | ENQ | NAK | % | 5 | E | U | e | u | | | ▒ | オ | ナ | ユ | | |
| 6 | ACK | SYN | & | 6 | F | V | f | v | | | ヲ | カ | ニ | ヨ | | |
| 7 | BEL | ETB | ’ | 7 | G | W | g | w | | | ア | キ | ヌ | ラ | | |
| 8 | BS | CAN | ( | 8 | H | X | h | x | | | イ | ク | ネ | リ | | |
| 9 | HT | EM | ) | 9 | I | Y | i | y | | | ウ | ケ | ノ | ル | | |
| A | LF | SUB | * | : | J | Z | j | z | | | エ | コ | ハ | レ | | |
| B | VT | ESC | + | ; | K | [ | k | { | | | オ | サ | ヒ | ロ | | |
| C | FF | FS | , | < | L | ¥ | l | \| | | | ャ | シ | フ | ワ | | |
| D | CR | GS | − | = | M | ] | m | } | | | ュ | ス | ヘ | ン | | |
| E | SO | RS | . | > | N | ^ | n | ▒ | | | ョ | セ | ホ | ゛ | | |
| F | SI | US | / | ? | O | _ | o | DEL | | | ッ | ソ | マ | ゜ | | |

▒ はASCIIコード表と異なる部分

図 3: JIS 8 ビット文字コード表

(c) JIS X 0208(通称 JIS 漢字)

図 4 に示す符号化文字集合のこと。1978 年に日本工業規格 (JIS) に定められた。16 ビットで、記号、英数字、かな、漢字などを表現できる。**全角文字** 6879 文字が登録されている。

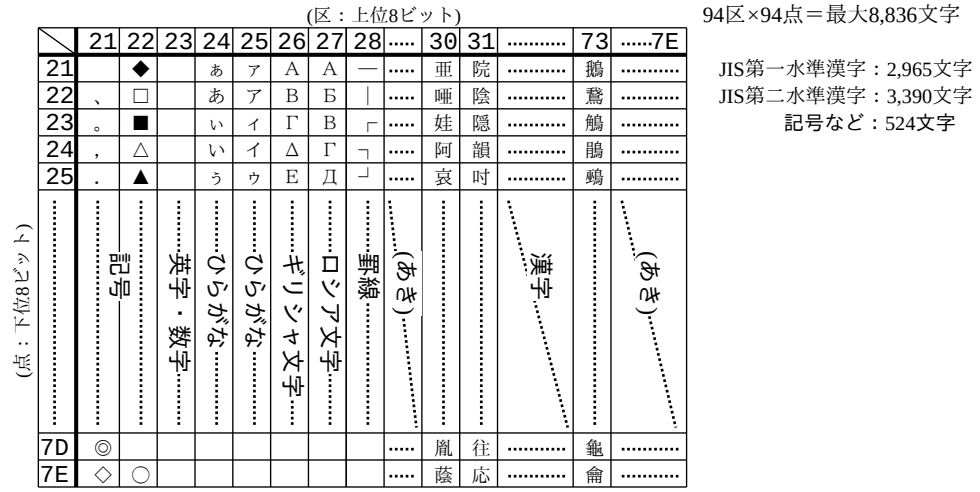


図 4: JIS 漢字コード表

(d) JIS X 0213(通称 JIS 拡張漢字)

図 5 に示す符号化文字集合のこと。2000 年に日本工業規格 (JIS) に定められた。その後、2004 年、2012 年に改正されている。コード表を２枚にすることで、より多くの文字を格納する。**全角文字** 11,223 文字が登録されている。JIS X 0208 を包含する上位規格 (superset, 上位集合) である。

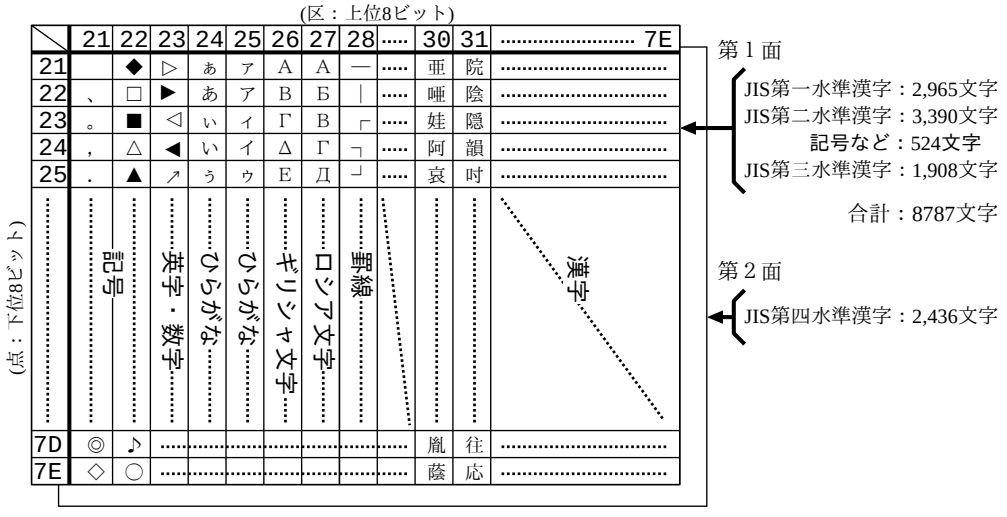


図 5: JIS 拡張漢字コード表

3. 日本で使用される文字符号化方式 (character encoding scheme)

   文字データをファイルに格納したり通信路に送信したりスルとき、どんなバイト列に対応付けるか考える必要がある。

   (a) ISO-2022-JP(通称 JIS コード)

   電子メール等で用いられることが多い。エスケープシーケンスを用いて、使用する符号化文字集合 (文字コード表) を切換える (初期状態は ASCII)。7bit の範囲にエンコーディングされる。次にエスケープシーケンスの一部を示す。

| | | | | | |
|---|---|---|---|---|---|
| ESC | ( | B | | : | ASCII に切換える。 |
| ESC | ( | J | | : | JIS X 0201 の半角英数に切換える。 |
| ESC | $ | B | | : | JIS X 0208 に切換える。 |
| ESC | $ | ( | Q | : | JIS X 0213 の第 1 面に切換える。 |
| ESC | $ | ( | P | : | JIS X 0213 の第 2 面に切換える。 |

   **例：**“A 亜 a\￥” を ISO-2022-JP にエンコーディングした状態

| 41H | 1BH | 24H | 42H | 30H | 21H | 1BH | 28H | 42H | 61H | 5CH | 1BH | 28H | 4AH | 5CH | 1BH | 28H | 42H |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 'A' | ESC | '$' | 'B' | '亜' | | ESC | '(' | 'B' | 'a' | '\' | ESC | '(' | 'J' | '￥' | ESC | '(' | 'B' |
| | <JIS X0208 へ> | | | | | | <ASCII へ> | | | | | <JIS X0201 へ> | | | | <ASCII へ> | |

   (b) EUC-JP(Extended UNIX Code for Japanese, 日本語 EUC)

   1980 年代に UNIX 上で日本語を扱うために考案された。現在では UTF-8 が普及し使用されることが少なくなっている。エスケープシーケンスを用いることなく、ASCII と JIS X 0208 を切換える。符号化文字集合として JIS X 0201 を指定できないので、半角の'￥' を表現することができない。JIS X 0208 の文字は JIS 漢字コードに 8080H を加えた値で表現する。

   「ISO-2022-JP より短くエンコーディングできる」、「変換規則が単純」、「バイト列のどの部分が全角文字か直に分かる」、「漢字を構成するバイトが ASCII と重ならない」等のメリットがある。

   **例：** “A 亜 a\\” を EUC-JP にエンコーディングした状態

| 41H | B0H | A1H | 61H | 5CH | 5CH |
|---|---|---|---|---|---|
| 'A' | '亜' | | 'a' | '\' | '\' |

   5CH は'\' を表現する。

   (c) Shift_JIS(Shift JIS code, SJIS)

   1980 年代に PC で日本語を扱うために考案され現在も Windows で使用されている。エスケープシーケンスを用いることなく、JIS X 0201 と JIS X 0208 を切換える。符号化文字集合として ASCII を指定できないので、半角の'\' を表現することができない [3]。JIS X 0208 の文字は、JIS 漢字コードを 8140H - 9FFCH と E040H - FCFCh に変換して表現する。計算で変換できるが若干複雑である。

   「ISO-2022-JP より短くエンコーディングできる」、「半角カナを表現できる」等のメリットがある。

   **例：** “A 亜 a￥￥” を Shift_JIS にエンコーディングした状態

| 41H | 88H | 9FH | 61H | 5CH | 5CH |
|---|---|---|---|---|---|
| 'A' | '亜' | | 'a' | '￥' | '￥' |

   5CH は'￥' を表現する。

[3] 日本の Java 言語や C 言語の教科書で'\n' を'￥n' と表記しているのは Windows 使用前提だからか？

4. Unicode(ユニコード)

符号化文字集合と文字符号化方式の両方を定めた文字コードの規格である。単一 (“Uni”) の巨大な文字コード表に、世界中の全ての文字を収めようとしている。符号化文字集合と文字符号化方式の両方を定めているので、符号化に関する混乱も少ない。

当初、中国・日本・韓国の漢字を統一 (CJK 統合漢字) することで、16bit で全ての文字にコードを割り振る計画だったが、世界中の文字は予想以上に多かったため途中で 21bit に変更された。CJK 統合漢字になっているので JIS 漢字コードとの変換には変換表が必要になる。

(a) 歴史

1980 年代後半から検討され 1990 年代前半に実用化された。現在も世界各地の文字、古い文字 (古代文字)、新しい文字 (絵文字) 等が追加され続けている。

| | | | | |
|---|---|---|---|---|
| Unicode1.0.1 | 1992 | 16bit | 28,359 文字 | JIS X 0201,0208,0212 の文字収録 |
| Unicode2.0.0 | 1996 | 21bit | 38,950 文字 | Unicode1.x.x との互換性を失う |
| Unicode3.0.0 | 2002 | 21bit | 95,221 文字 | JIS X 0213 へ対応 |
| Unicode4.0.0 | 2003 | 21bit | 96,447 文字 | |
| Unicode5.0.0 | 2006 | 21bit | 99,089 文字 | 楔文字や象形文字を追加 |
| Unicode6.0.0 | 2012 | 21bit | 109,449 文字 | 携帯絵文字追加 |
| Unicode7.0.0 | 2014 | 21bit | 113,021 文字 | |
| Unicode8.0.0 | 2015 | 21bit | 120,737 文字 | |
| Unicode9.0.0 | 2016 | 21bit | 128,172 文字 | |

(b) 符号化文字集合

図 6 に示すように Unicode の文字コード表は 17 面 (5bit)、256 区 (8bit)、256 点 (8bit) で構成される。この表に最大 111 万文字を収容可能である。半角 ASCII 文字から始まり、半角文字も全角文字もこの表に収録されている。

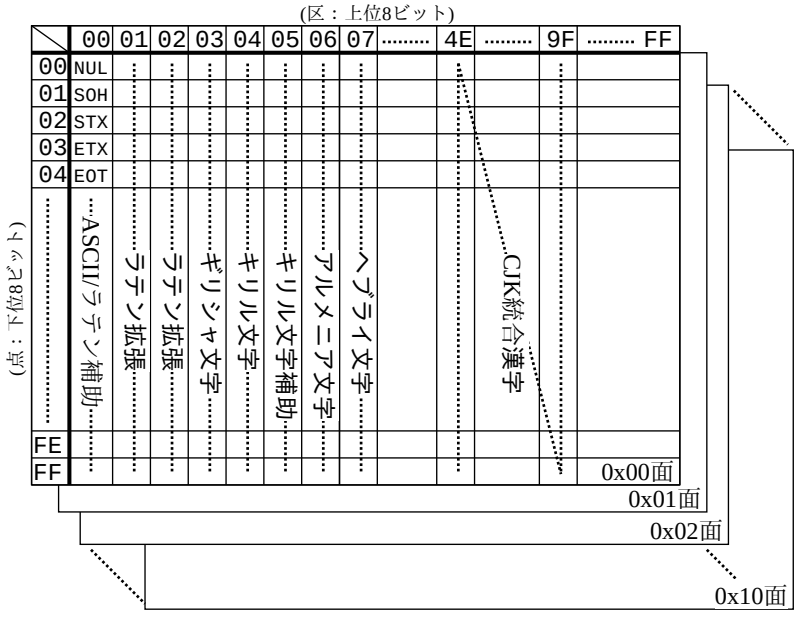


図 6: Unicode 一覧

表の 1 文字を指定するためには、21bit(5+8+8) のコード (Unicode スカラ値) を用いる。Unicode スカラ値は、"U+1234"のように先頭に U+を付加した 16 進数で表現する。

00 面 (U+0000〜U+FFFF) は BMP(Basic Multilingul Plane, 基本多言語面) と呼ばれる。当初は BMP だけで全ての文字を格納する予定であった。U+0000〜U+007F は ASCII コードと同じ配列になっている。Java 言語の char 型が 16bit なのも、16bit の Unicode を格納することを前提に設計されたためである。

**なお、完全なコード一覧は、https://ja.wikipedia.org/wiki/Unicode で見ることができる。**面白い文字が見つかるので、一度、見てほしい。

(c) 文字符号化方式

Unicode 用の文字符号化方式 (UTF:Unicode Transformation Format) が定められている。多くの方式が定められているが、代表的な方式は以下の 3 つである。

UTF-8 : 半角英数字が小さく符号化できる。
最もよく使用される方式である。

UTF-16 : 00 面を 16bit に、その他は 32bit に符号化する。
Unicode が 21bit になってから、Java の char 型に使用されている。

UTF-32 : Unicode の上位に 0 を付加し 32bit に符号化する。
固定長なので扱いやすい。

(d) UTF-8 による符号化手順

21bit の Unicode のコードポイントを 2 進表記したものを、表中の y、x に格納する。y の部分には最低 1 つの 1 のビットがあるはずである。符号化の結果は 1〜4 バイトになる。

| Unicode | UTF-8(ビット列で表記) | | | | バイト数 |
|---|---|---|---|---|---|
| U+0000〜U+007F | 0xxxxxxx | | | | 1 |
| 例 | | | | | |
| U+0000 | 00000000 (00H) | | | | |
| U+007F | 01111111 (7FH) | | | | |
| U+0080〜U+07FF | 110yyyyx | 10xxxxxx | | | 2 |
| 例 | | | | | |
| U+0080 | 11000010 | 10000000 (C2H 80H) | | | |
| U+07FF | 11011111 | 10111111 (DFH BFH) | | | |
| U+0800〜U+FFFF | 1110yyyy | 10yxxxxx | 10xxxxxx | | 3 |
| 例 | | | | | |
| U+0800 | 11100000 | 10100000 | 10000000 | | |
| | (E0H) | (A0H) | (80H) | | |
| U+FFFF | 11101111 | 10111111 | 10111111 | | |
| | (EFH) | (BFH) | (BFH) | | |
| U+10000〜U+1FFFFF | 11110yyy | 10yyxxxx | 10xxxxxx | 10xxxxxx | 4 |
| 例 | | | | | |
| U+10000 | 11110000 | 10010000 | 10000000 | 10000000 | |
| | (F0H) | (90H) | (80H) | (80H) | |
| U+1FFFFF | 11110111 | 10111111 | 10111111 | 10111111 | |
| | (F7H) | (BFH) | (BFH) | (BFH) | |

5. 実行例

```
システムプログラミング　宿題の解答


問題 :半角英数字, 半角カナ, 全角カナ, 漢字,\, ¥等を含むテキストファイルを
      観察する。

1. UTF-8 エンコーディングのテキストファイルを作り表示してみた

$ cat x0208UTF8.txt    // 含まれる漢字は JIS X 0208 の範囲まで
A^^ef^^bd^^b1 ア漢¥\

2. 色々なエンコーディングに変換して 16 進ダンプを確認する

// UTF-32 にエンコードして Unicode 値を確認しておく
$ iconv -f UTF-8 -t UTF-32BE x0208UTF8.txt | hexdump
// UTF-32BE(ビッグエンディアン):32 ビットの上位桁からのバイト順
0000000 00 00 00 41 00 00 ff 71 00 00 30 a2 00 00 6f 22
0000010 00 00 00 a5 00 00 00 5c 00 00 00 0a
000001c
// 00 00 00 41 : A (u+0041) : 基本ラテン文字領域 (ASCII 互換)
// 00 00 ff 71 : ^^ef^^bd^^b1 (u+ff71) : 全角・半角 (全角 ASCII, 半角カナなど) 領域
// 00 00 30 a2 : ア (u+30a2) : カタカナ領域
// 00 00 6f 22 : 漢 (u+6f22) : CJK 統合漢字領域
// 00 00 00 a5 : ¥ (u+00a5) : ラテン補助領域
// 00 00 00 5c : \ (u+005c) : 基本ラテン文字領域 (ASCII 互換)
// 00 00 00 0a : LF(u+000a) : 基本ラテン文字領域 (ASCII 互換)
// Unicode 値と JIS X 0208 漢字コードに関連性なし (対応に規則性なし)

// UTF-8 にエンコードされた状態を 16 進ダンプ
$ hexdump x0208UTF8.txt
0000000 41 ef bd b1 e3 82 a2 e6 bc a2 c2 a5 5c 0a
000000e
// 41       : 01000001                    : A (u+0041)
// ef bd b1 : 11101111 10111101 10110001 : ^^ef^^bd^^b1 (u+ff71)
// e3 82 a2 : 11100011 10000010 10100010 : ア (u+30a2)
// e6 bc a2 : 11100110 10111100 10100010 : 漢 (u+6f22)
// c2 a5    : 11000010 10100101          : ¥ (u+00a5)
// 5c       : 01011100                    : \ (u+005c)
// 0a       : 00001010                    : LF(u+000a)

// EUC-JP にエンコードされた状態を 16 進ダンプ
$ iconv -f UTF-8 -t EUC-JP x0208UTF8.txt | hexdump
0000000 41 8e b1 a5 a2 b4 c1 5c 5c 0a
000000a
// 41    : A (0x41)   : ASCII コード
// 8e b1 : ^^ef^^bd^^b1 (0x8eb1) : JIS X 0201 半角カナコード ('^^ef^^bd^^b1':0xb1) は、例
//                      外的に前に 0x8e を付加してエンコードする
// a5 a2 : ア (0xa5a2) : JIS X 0208 コード ('ア':0x2522) に +0x8080
// b4 c1 : 漢 (0x3441) : JIS X 0208 コード ('漢':0x3441) に +0x8080
// 5c    : ¥ (0x5c)   : ASCII コード ('\':0x5c) と区別がつかない
//                      EUC では JIS X 0201 は扱えない
// 5c    : \ (0x5c)   : ASCII コード
// 0a    : LF(0x0a)   : ASCII コード


// SJIS にエンコードされた状態を 16 進ダンプ
```

```
$ iconv -f UTF-8 -t SJIS x0208UTF8.txt | hexdump
iconv: x0208UTF8.txt:1:7: cannot convert
0000000 41 b1 83 41 8a bf 5c
0000007
// 41    : A (0x41)   : JIS X 0201 コード
// b1    : ^^ef^^bd^^b1 (0xb1)   : JIS X 0201 コード
// 83 41 : ア (0x8341) : JIS X 0208 コード (0x2522) を計算で変換
// 8a bf : 漢 (0x8abf) : JIS X 0208 コード (0x3441) を計算で変換
// 5c    : ￥ (0x5c)   : JIS X 0201 コード
// エラー: '\' は JIS X 0201 に含まれないのでエラー
//

// ISO-2022-JP にエンコードされた状態を 16 進ダンプ
$ iconv -f UTF-8 -t ISO-2022-JP x0208UTF8.txt | hexdump
iconv: x0208UTF8.txt:1:1: cannot convert
0000000 41
0000001
// 半角カナでエラーが発生する。
// 実は ISO-2022-JP は半角カナを許容しない。(メールで半角カナ不可)
// "ESC ( J"は JIS X 0201 のラテン文字 (0x7f 以下) へ切換える。
//
// iconv で半角カナが使用できるエンコーディング方式は ISO-2022-JP
// の改良版 ISO-2022-JP-2, ISO-2022-JP-3, ISO-2022-JP-2004 だけ。
// (正式な ISO-2022-JP-2,ISO-2022-JP-3,ISO-2022-JP-2004 規格が半角
// カナを扱えるかどうかは調べた範囲では不明。恐らく iconv の独自拡張)
//
// iconv は半角カナを表現するためにエスケープシーケンス"ESC ( I"を
// 使用する。これは、JIS X 0201 のカナ領域へ切換える。
// 切換えた後は、半角カナコードから 0x80 を引いて 7bit で表現する。
// "ESC ( I"による文字集合切換えは、別規格にあり流用と考えられる。
//
// ISO-2022-JP でエンコードした結果は 7bit の範囲におさまる。
// ASCII(7bit) のみ想定の通信回線やプログラムを通過可の場合が多い
//   => メールサーバが古く 7bit しか通さなくてもメールの配信が可能
//   => 日本語メールのエンコーディングに使用される

// iconv の ISO-2022-JP-2004 にエンコードされた状態を 16 進ダンプ
$ iconv -f UTF-8 -t ISO-2022-JP-2004 x0208UTF8.txt | hexdump
0000000 41 1b 28 49 31 1b 24 42 25 22 34 41 1b 28 4a 5c
0000010 1b 28 42 5c 0a
0000015
// 41       : A        : ASCII コード ('A':0x41)
// 1b 28 49 : ESC ( I  : JIS X 0201 の半角カナへ切換える
// 31       : ^^ef^^bd^^b1       : この状態では ('^^ef^^bd^^b1':0xb1) から 0x80 を引いて
//                        0x31 で表現する
// 1b 24 42 : ESC $ B  : JIS X 0208 に切換える
// 25 22    : ア       : JIS X 0208 コード ('ア':0x2522)
// 34 41    : 漢       : JIS X 0208 コード ('漢':0x3441)
// 1b 28 4a : ESC ( J  : JIS X 0201(ローマ字) に切換える
// 5c       : ￥       : JIS X 0201 コード ('￥':0x5c)
// 1b 28 42 : ESC ( B  : ASCII に切換える
// 5c       : \        : ASCII コード ('\':0x5c)
// 0a       : LF       : ASCII コード (LF:0x0a)
```