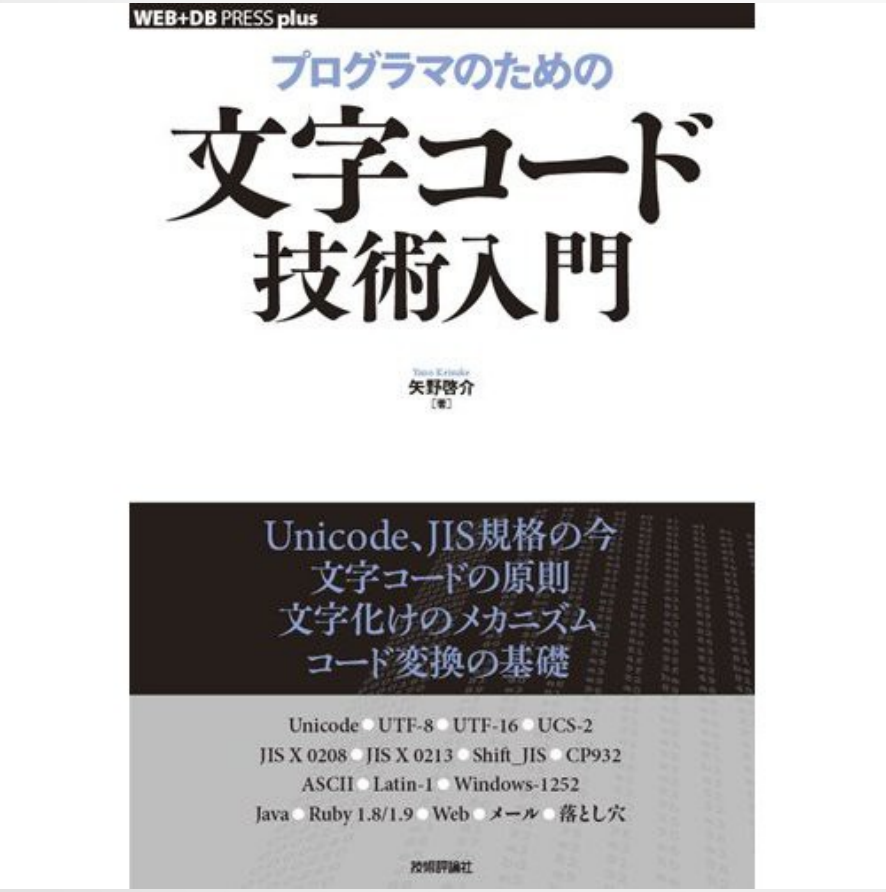
文字コード(2)

林部祐太 (NAIST)

@国立国会図書館 関西館 電子図書館課
2013/9/27

参考書

『プログラマのための文字コード技術入門』（技術評論社、2010年）



※特に注記がない場合，本スライドの図表は本書からの引用

2

予定

- 第1回　文字コードとは (2013-7)
  - 文字とコンピュータ
  - 文字コードの編成 (文字集合・符号化文字集合・文字符号化方式)
- 第2回　代表的な符号化文字集合・文字符号化方式 (2013-9)
  - 代表的な文字符号化方式(EUC-JP, ISO-2022-JP, Shift_JIS, UTF-16, UTF-32, UTF-8)
- 第3回　インターネットと文字コード (2013-11)
  - 文字コードの自動判別の仕組み
  - インターネットと文字コードの関係

今回のアウトライン

- 2バイト文字コード
- 代表的な符号化文字集合, 文字符号化方式
  - JIS X 0208
    - EUC-JP
    - ISO-2022-JP
    - Shift_JIS
  - Unicode
    - UTF-16
    - UTF-32
    - UTF-8

2バイト文字コード

前回のまとめ

- ASCII
  - 文字コードの基本
  - 7bitで表現(2^7 =128文字種)
- ISO/IEC646
  - ASCIIの各国用
- ISO/IEC 2022
  - 複数の符号化文字集合を呼び出せる
- JISX0208 (+各種の文字符号化方式)
  - ISO/IEC 2022に則った2byte符号化文字集合
- ISO/IEC 8859, Latin-1
  - 1byteコード
- Unicode, ISO/IEC 10646
  - 世界中の文字を単一の符号体型で表現

各国版のISO 646

- 7bit
  - 一部がASCIIと異なる

コード	国家規格	国
CA	CSA Z243.4	カナダ
CN	GB/T 1988	中華人民共和国
DE	DIN 66003	ドイツ
DK	DS 2089	デンマーク
GB	BS 4730	イギリス
HU	MSZ 7795.3	ハンガリー
JP	JIS X 0201	日本
MT		マルタ
NO	NS 4551-1	ノルウェー
SE	SEN 850200_B	スウェーデン
US	ANSI INCITS 4	アメリカ
YU	JUS I.B1.002	ユーゴスラビア

2進	10進	16進	ASCII	DE	DK/NO	GB	HU	JP	MT	SE	YU
0010 0011	35	23	#	#	#	£	#	#	#	#	#
0010 0100	36	24	$	$	$	$	¤	$	$	¤	$
0100 0000	64	40	@	§	@	@	Á	@	@	@	Ž
0101 1011	91	5B	[	Ä	Æ	[	É	[	ġ	Ä	Š
0101 1100	92	5C	\	Ö	Ø	\	Ö	¥	ż	Ö	Đ
0101 1101	93	5D	]	Ü	Å	]	Ü	]	ħ	Å	Ć
0101 1110	94	5E	^	^	^	^	^	^	^	^	Č
0110 0000	96	60	`	`	`	`	á	`	ċ	`	ž
0111 1011	123	7B	{	ä	æ	{	é	{	Ġ	ä	š
0111 1100	124	7C	|	ö	ø	|	ö	|	Ż	ö	đ
0111 1101	125	7D	}	ü	å	}	ü	}	Ħ	å	ć
0111 1110	126	7E	~	ß	~	~	~	‾	Ċ	~	č

↑各国版ごとに異なっている文字

ISO/IEC 2022：8bitへの拡張

- 様々な7bitコードを使えるように
- 表現可能な種類は7bitの2倍

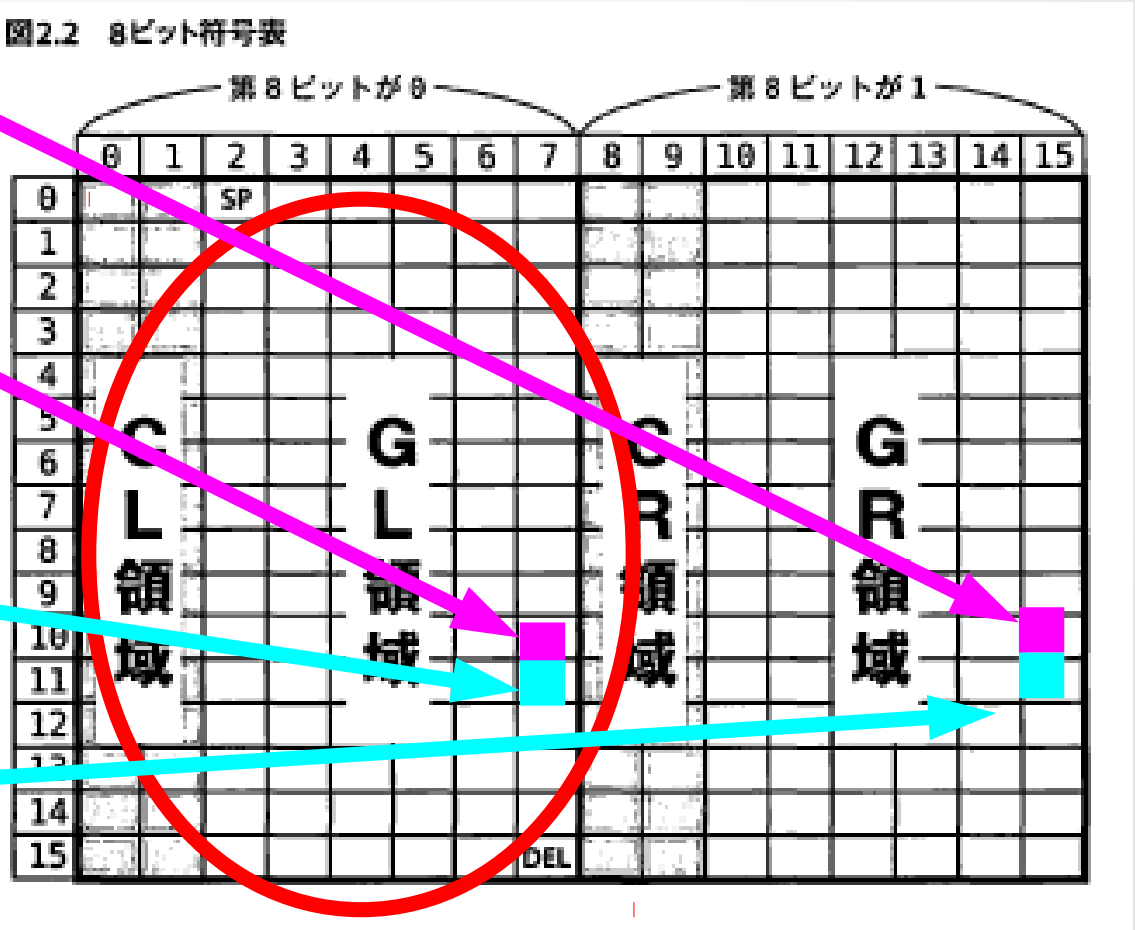
“z” 0xFA
1111 1010

“z” 0x7A
0111 1010

0x7B
ASCIIなら“{”
DEは“ä”

0xFB
ASCIIなら“{”
DEは“ä”

- 図形文字の領域(GL, GR)それぞれ94文字
- GLなら0x21 から 0x7E まで



ISO/IEC 2022のイメージ

図2.3　符号化文字集合の呼び出しの概念

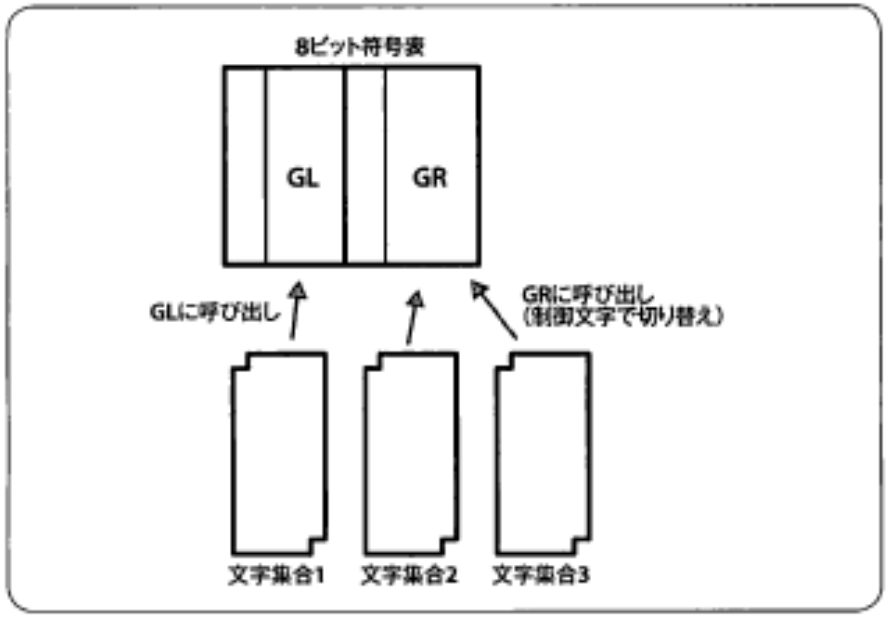


図2.6　エスケープシーケンスによる切り替えの例

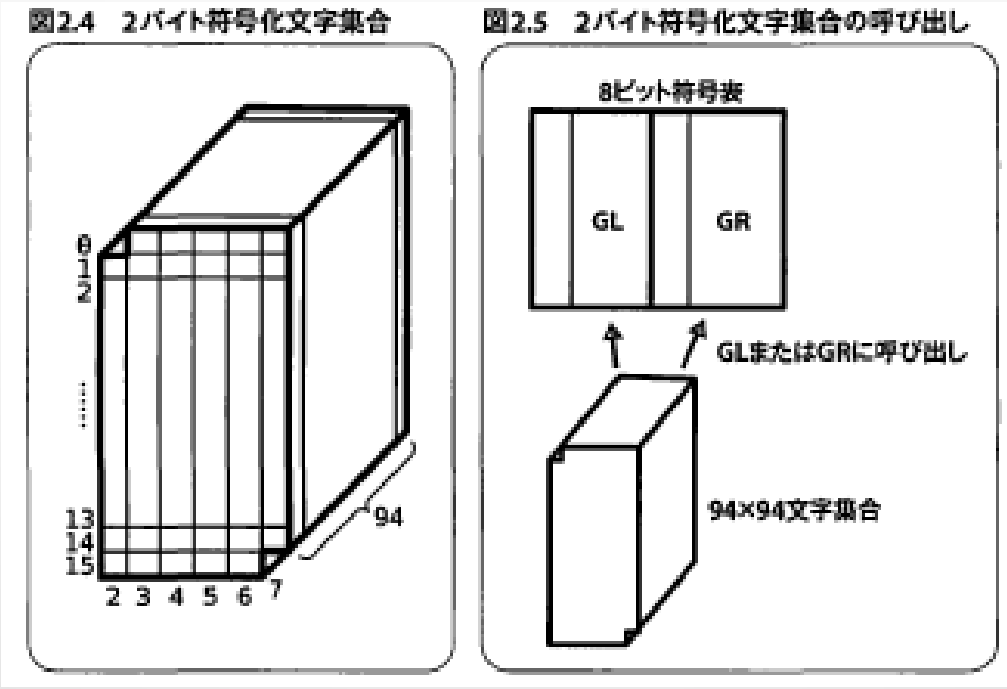
例題　「Köln, Créteil」というテキストを符号化

	エスケープシーケンス																
	1B 28 4B	4B	7C	6C	6E	2C	20	1B 28 66	43	72	7B	74	65	69	6C		
	ドイツ版の指示	K	ö	l	n	,	SP	フランス版の指示	C	r	é	t	e	i	l		
			↑								↑						
			ドイツ版にしかない文字								フランス版にしかない文字						

※ここで「ドイツ版」「フランス版」は各国版のISO/IEC 646を指す。

複数バイト文字集合

- 複数バイトを使って1文字を表す
- 例：2バイト文字
  - 「図形文字の領域」を2つ使う
  - ISO/IECなら，94*94 = 8,836 文字表現可能になる



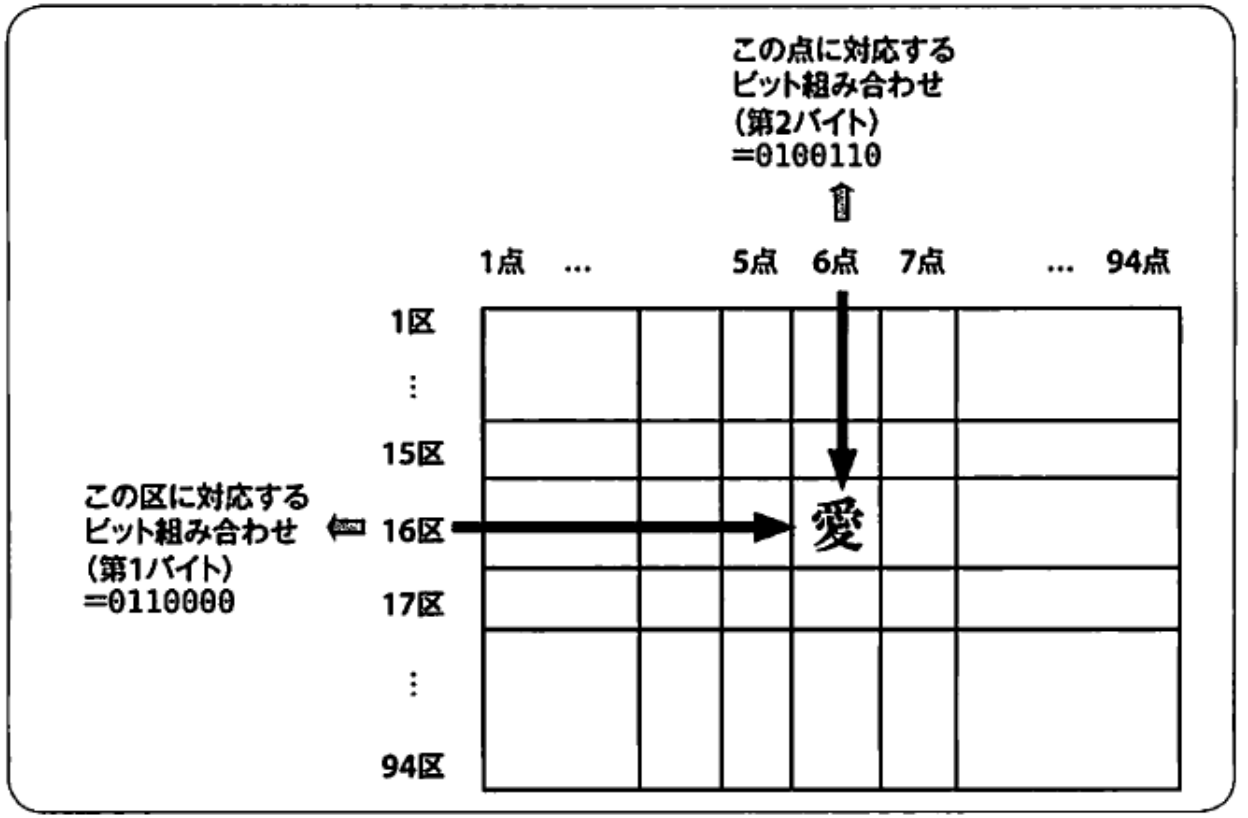
(例) ISO/IEC 2022

- 例: GLにASCII, GRにJISX0208(日本語2バイト文字)
  - 01111010 “z”
  - 1110000 1100110 “愛”
- 例: GLにJISX0208(日本語2バイト文字), GRにASCII
  - 11111010 “z”
  - 0110000 0100110 “愛”

句点番号

- 94×94の文字コード表
- 1つのセルが，1つの句点番号を表し，1つのビット列に対応
- (注)句点番号≠ビット列

図3.4　区点番号とビット組み合わせの関係



ここまでのまとめ

- ISO/IEC 2022 : 8bitへの拡張
  - 図形文字の領域は GL, GRそれぞれ94文字
  - GLなら0x21 から 0x7E まで
  - GL,GRに文字集合を割り当てて使う
- 複数バイト文字集合
  - 複数のbyte(=8bit)を使って1文字を表す
  - ISO/IECの2バイト文字なら, 94*94 = 8,836 文字表現可能になる
  - **以後を理解するのに重要な概念**

代表的な符号化文字集合･文字符号化方式

第3章 代表的な符号化文字集合 (p45～)

- 3.1ASCIIとISO/IEC 646
  - 最も基本的な1バイト文字集合
- 3.2 JIS X 0201
  - ラテン文字と片仮名の1バイト文字集合
- 3.3 JIS X 0208
  - 日本の最も基本的な2バイト文字集合
- 3.4 JIS X 0212
  - 補助漢字
- 3.5 JIS X 0213
  - 漢字第3・第4水準への拡張
- 3.6 ISO/IEC 8859シリーズ
  - 欧米で広く使われる1バイト符号化文字集合
- 3.7 UnicodeとISO/IEC 10646
  - 国際符号化文字集合

第4章 代表的な文字符号化方式 (p117～)

- JIS X 0208
  - EUC-JP
  - ISO-2022-JP
  - Shift_JIS
- Unicode
  - UTF-16
  - UTF-32
  - UTF-8

2つの符号化文字集合について話します

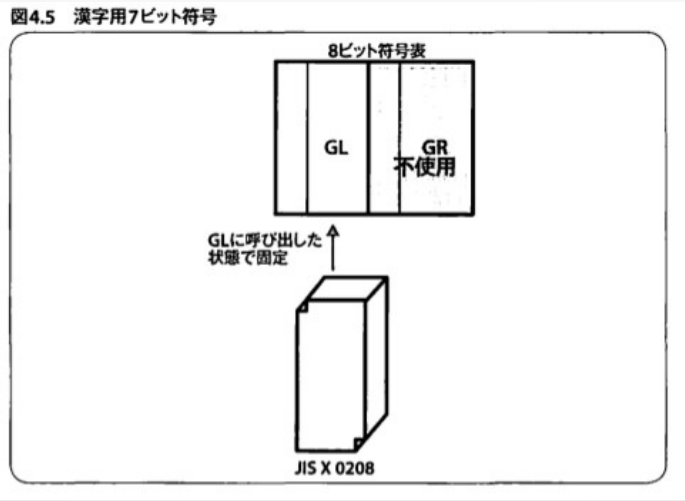
JIS X 0208に関連する文字符号化方式

JISX0208で定められた符号化方式

- 漢字用7bit符号
- 漢字用8bit符号
- 国際基準版・漢字用7bit符号
- 国際基準版・漢字用8bit符号 (EUC-JPの中核部分に相当)
  - EUC-JPとは異なりJIS X 0201片仮名とJIS X 0212 補助漢字は不使用
- ラテン文字・漢字用7bit符号
- ラテン文字・漢字用8bit符号
- シフト符号化表現(附属書1) (Shift-JIS相当)
- RFC1468符号化表現(附属書2) (ISO-2022-JP相当)

漢字用7bit符号

- GLをJISX0208に固定したもの
- GRは使わない
- 「空」22 区 85 点 → 0011 0110 0111 0111
  - 第1バイトは22に0x29を足した0x36 (011 0110)
- 第2バイトは85に0x29を足した0x75 (111 0111)



EUC-JP

- 0x7F以下の値(GL)はいつもASCII
- 制御文字SS2 (0x8E) の直後の1 文字分はJIS X 0201片仮名
- 制御文字SS3 (0x8F)の直後のI 文字分は1IS X 0212 補助漢字
- UNIX系OSでよく使われていた（最近はUTF8が主流）

図4.7　EUC-JPの構造

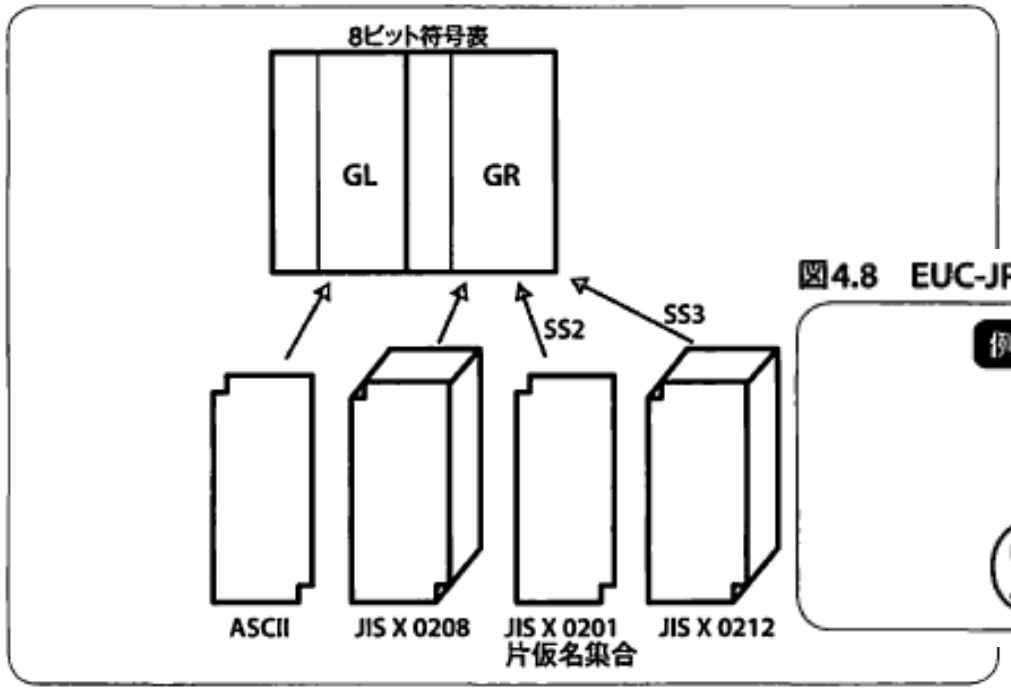
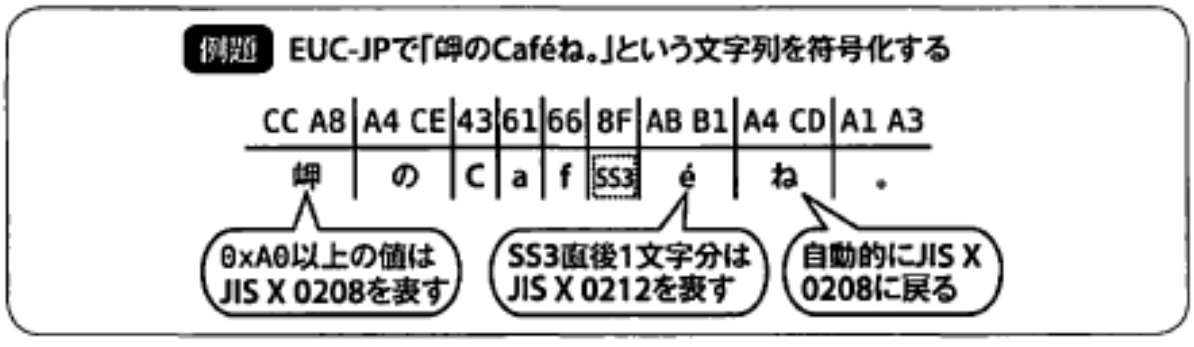


図4.8　EUC-JPによる符号化の例



EUC-JPの特徴と問題

- 2バイトコード
  - 必ず第8bit=1
  - ASCII以外の文字は第8bitがOのコード範囲にはこない
  - 0x7F(=111 1111) 以下のバイト値を見たら常にASCII として解釈できる
- 2バイトコードの1byteめと2byteめは同じ範囲のバイト値
  - 2バイトコードの途中だけ見ても文字の区切りが判別できない
  - 「検索」するには文章の先頭から見ていかないといかない
    →UTF-8では解決！
  - さもなければ,「源」を検索して「文字」という文字列がhitする
    - 「文字」"CA B8 BB FA"
    - 「源」"B8 BB"
- 重複符号化
  - ASCIIとJISX 0208を同時に使うので,ラテン文字や数字のように双方に存在する文字は2 つのコード値を持つ

ISO-2022-JP

- GRは使用しない (=第8bitは常に0, 0x80以上は無い)
  - 7bitのみ使う符号化方式
- GLをエスケープシーケンスで切り替えて使用する
- 電子メールでよく使われる

図4.10　ISO-2022-JPの構造

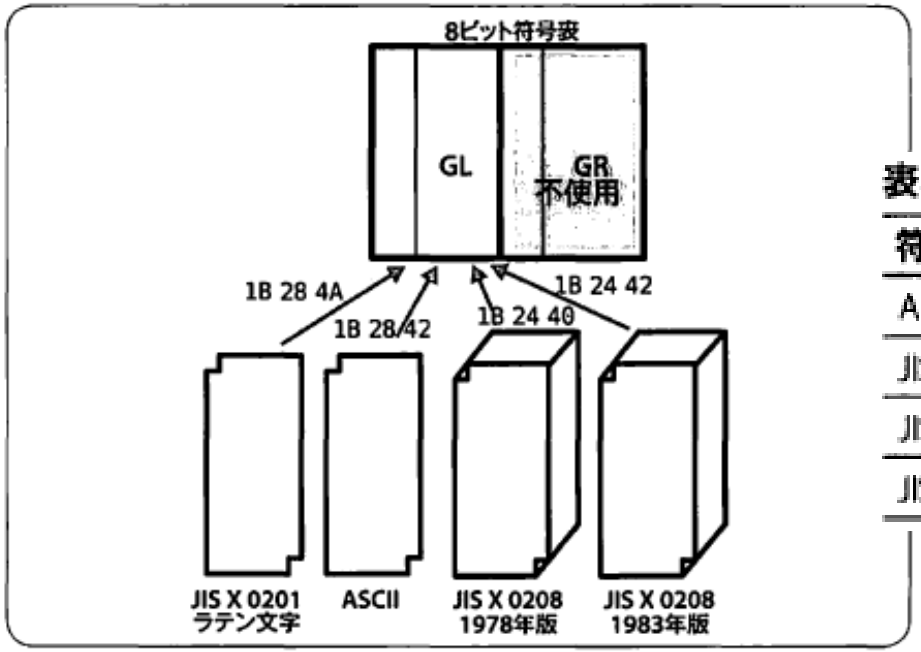
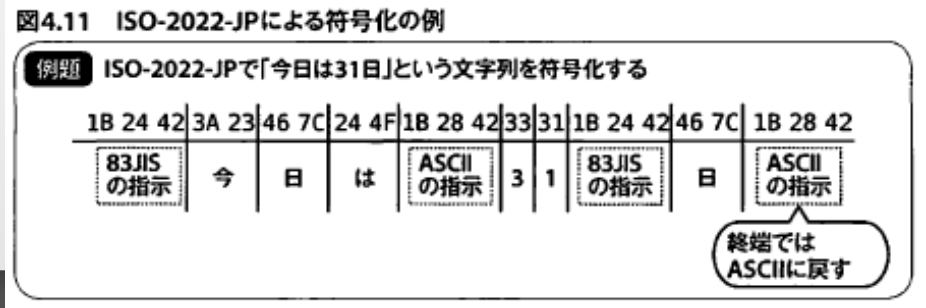


表4.1　ISO-2022-JPで用いるエスケープシーケンス

符号化文字集合	エスケープシーケンス	文字列表現
ASCII (ISO/IEC 646 国際基準版)	1B 28 42	ESC (B
JIS X 0201 ラテン文字	1B 28 4A	ESC (J
JIS X 0208 1978年版	1B 24 40	ESC $ @
JIS X 0208 1983年版	1B 24 42	ESC $ B

ISO-2022-JPの特徴

- 状態を持つ符号化方式
  - 同じバイト値でも状態によって表す文字が違う
  - 内部処理には向かない. あくまで情報交換用
- 文字集合をエスケープシーケンスで指示するので符号化方式が明示されていない場合に自動判別がしやすい
- 状態の制限がある
  - 改行(0x0D 0x0A の2byte)の前
    - エスケープシーケンスによってASCIIかJIS ラテン文字の状態に
  - ファイル終端の前
    - ASCII の状態に戻す
  - 通信エラーなどの影響が及ぶ範囲を最小限に抑えるための工夫



Shift_JIS

- JIS X 0201の8bit符号の隙間, JIS X0208 を変形して押し込ん
  - JIS X 0208で定義されているbit組み合わせを一定の計算式によって変換
- 複雑なので詳細は割愛
  - 第1バイトを狭めてその分第2 バイトを広げるような格好

区点番号からShift_JISの第1・第2バイトを求めるには以下の計算を行います。ここで、区番号をk、点番号をtとします。また、記号÷は整数除算(小数点以下切り捨て)を表すものとします。第1バイト(S_1)は、以下の計算によります。

$1 \leqq k \leqq 62$のとき、$S_1 = (k-1) \div 2 + 0x81$

$63 \leqq k \leqq 94$のとき、$S_1 = (k-1) \div 2 + 0xC1$

第2バイト(S_2)は、以下によります。

- kが奇数の場合

$1 \leqq t \leqq 63$のとき、$S_2 = t + 0x3F$

$64 \leqq t \leqq 94$のとき、$S_2 = t + 0x40$

- kが偶数の場合

Shift_JISの問題点

- 計算の結果，2バイトコードの第2 バイト0x7F以下になることがある
  - →プログラムにとっては重大な問題
  - 2バイトコードを正しく解釈しないプログラムでは，2バイトコードの第2 バイトを1バイトコードと誤認することがある
- 特に、0x5C (ASCIIのバックラッシュ)にあたる値は特殊な意味を持つことがあるので注意が必要
  - 例：「表」"0x95 0x5C"
  - いわゆる「だめ文字」
- 重複符号化の問題 (EUC-JP同様)
- 他にも問題点は多い
  - 「円記号問題」
  - 「本来不要な1 バイト片仮名が小さくない領域を占めている」
- 空き領域に独自に文字を定義したいわゆる「機種依存文字」を独自に実装した変種が多数
  - Windows の機種依存文字付きのShift_JIS
  - CP932, MS932, Windows-31J
  - 重複して複数の符号位置に配置されている文字がある

ここまでのまとめ

- EUC-JP
  - 検索するには，文章の先頭から見ていく必要がある
  - UNIX系でよく使われていた
- ISO-2022-JP
  - 電子メールでよく使われている
  - それ以外ではまれ(「状態」をもつので)
- Shift_JIS
  - 「ダメ文字」「変種」など問題が多数
  - しかし，今でも良く使われている

色々と問題があることが分かった

Unicodeに関連する文字符号化方式

Unicode

- 全文字に一意な符号位置を
- 歴史
  - 1987頃からxeroxが開発
  - 1989 Unicode Draft1
  - 1990/12 Final Draft
  - 1991/1 ユニコードコンソーシアム
  - 1991/6 ISO/IEC 10646がUnicodeと一本化することに
- 「国際符号化文字集合(UCS-2) 」とも
- 「16bit(2byte)の固定幅で1文字」という設計
  - 最大65,536 文字
  - ほどなくして, これでは不十分と判明
  - 16bitの符号単位2つ(4byte)で1文字を表す仕組みができる (UCS4)
- 既存の各種文字コードとの対応関係に配慮している
  - 原規格分離規則, 統合漢字など. 詳細は割愛.

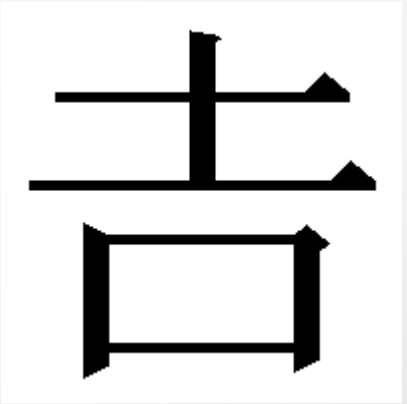
UCS-4

- UCS-4 を構成する4byteは「群・面・句・点」に対応
  - 群：0x00～0x7F
  - 面・句・点：0x00～0xFF
  - 全部で2^{31}の符号位置
- 基本多言語面(Basic Multilingual Plane; BMP)
  - 群00面00
  - UCS2に相当する面
  - 通常用いる文字の多くを含む
- UTF-16 で表すことのできない面
  - 群00面0x10より先の面には将来も文字を割り当てない
  - つまり，群00以外の群は実質的に使われない
- 符号位置の表現
  - U+XXXX のように、接頭辞=U+ を付けた4～6桁の16進数を使う

漢字のサロゲートペア文字の例

- U+2F81A
  - “冬”の旧字体
  - 下のニがン
  - JISでは包摂

- U+20BB7
  - “吉”の別書体
  - “土”+“口”
  - JISでは包摂



結合文字

- 複数の部品(結合文字)の合成によって1 文字を表すことが可能
- 複数表現が可能
  - 「正規化」はいずれかの表現方式に揃える

図3.25　文字合成

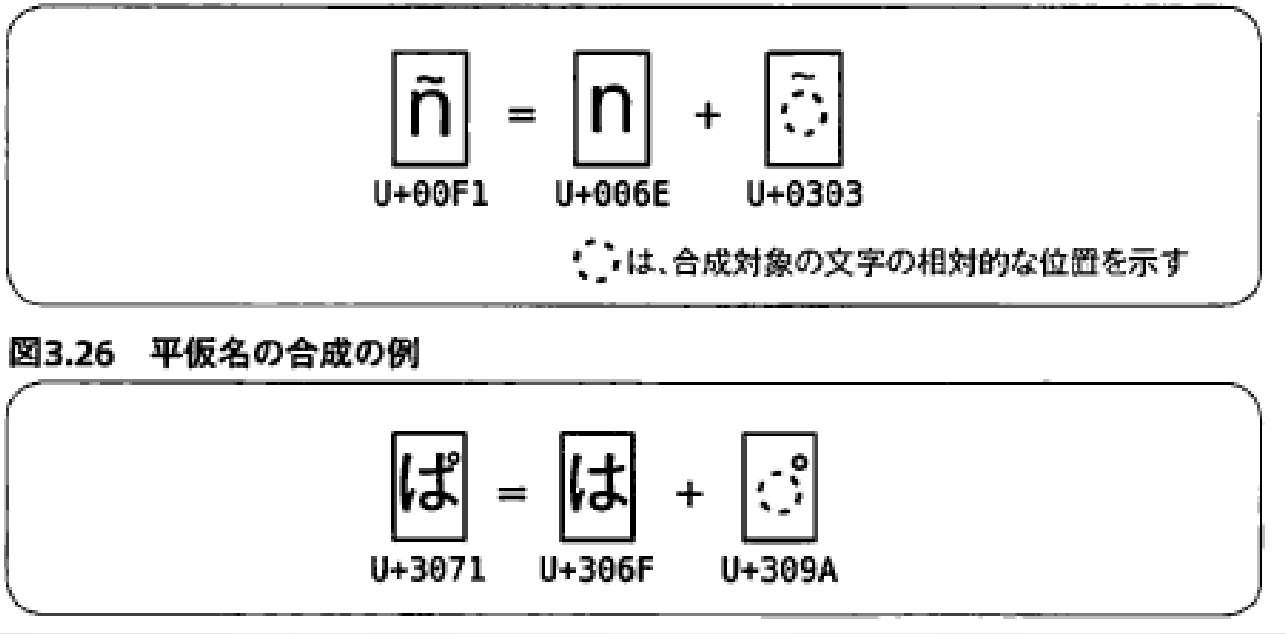


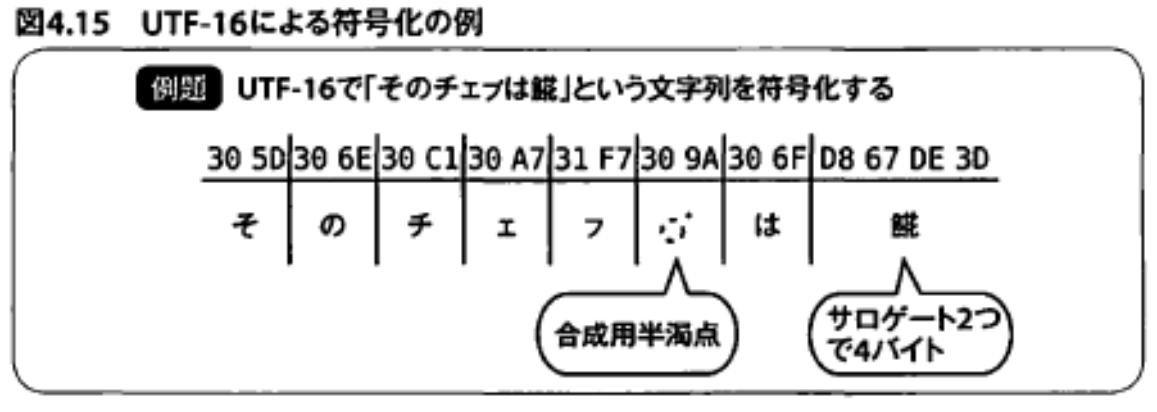
図3.26　平仮名の合成の例

UTFの概説

- 当初「16bit(2byte)の固定幅で1文字」という設計
  - 最大65,536 文字
  - ほどなくして，これでは不十分と判明
  - 16bitの符号単位2つ(4byte)で1文字を表す仕組みができる (UCS4)
- 現在は100万あまりの符号位置
  - 8bit, 16bit, 32bit単位といったさまざまな符号化方式で表現するという形 (それぞれUTF-〇〇)
- UTF-16
  - 1文字は16bit1つ(BMP内)または，2つ(BMP外)で表現
  - バイト単位ではASCII非互換 (先頭に0x00がつく)
- UTF-32
  - 32bit固定幅で全符号位置を表現
  - バイト単位ではASCII非互換 (先頭に0x00 0x00 0x00がつく)
- UTF-8
  - 1バイトから4 バイト(もしくは6バイト)の可変長
  - 1バイトの部分はASCII互換
  - 文字の区切りが文書の先頭から見なくても分かる!

UTF-16

- Java の文字列表現等プログラミング上の処理などで使われる
- 16bitの符号単位
  - BMP内の文字は符号単位1つ
  - BMP外の文字(>0x10000)は符号単位2つ
    - surrogate pairという仕組み
    - BMP の中の文字の割り当てのない符号位置2つを用いて，BMP 外の面の符号位置を指す



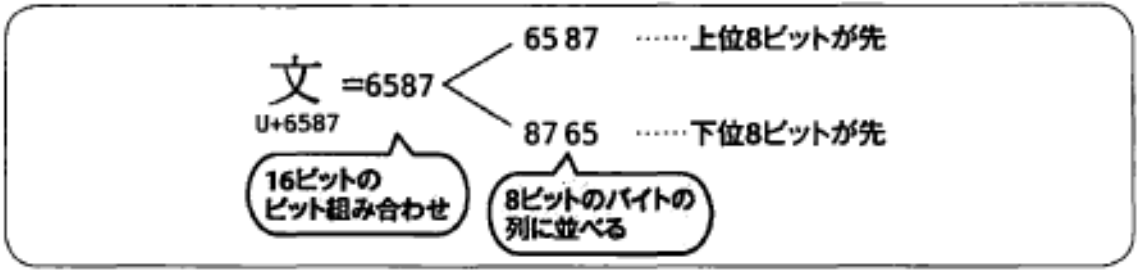
surrogate pair

- サロゲートペアのために用いる領域
  - U+D800 ～ U+DBFF :上位サロゲート
  - U+DC00 ～ U+DFFF :下位サロゲート
- 符号化したい文字の符号位置(整数)をn(0x10000≦n≦0x10FFFF)に対し
  - n' = n - 0x10000 = yyyyyyyyyyxxxxxxxxx
  - 上位サロゲート =110110yyyyyyyyyy
  - 下位サロゲート =110111xxxxxxxxx
  - 例: 躋(しかた) 符号位置U+28277(面02)
    - D860, DE77
- UCS2との関係
  - UTF16はUCS2を拡張したもの
  - UCS2よりも表現可能な文字数が大幅に増えている
  - 1つの符号位置が2バイト固定幅で表現されるという利点は失われた

UTF16のバイト順の問題

- 上位8bitから先に並べるか，下位8bitから先に並べるかという問題
  - JIS X 0208は第1・第2 バイトがあらかじめ決まっている
- ビッグエンデイアン(UTF16-BE)
  - 上位8bitが先頭にくるバイト順
- リトルエンデイアン(UTF16-LE)
  - 下位8bitが先頭にくるバイト順
- バイト順マーク(Byte Order Mark; BOM)
  - データの先頭に，ビッグエンディアンではFE FF、リトルエンディアンではFF FE という2 バイトの列をおき，どちらのバイト順を採用しているかを示す

図4.16　バイト順の問題



UTF-32

- Unicode符号位置の整数をそのまま32bitで表現したもの
  - 1つの符号位置は必ず4byte
  - 『文字』を表すには、複数の符号位置を並べなければならないことがある(結合文字など)
- UCS-4の群00の面00から面10までの符号位置のみ
  - 実際にはただしそれ以外には文字が割り当てられていない
  - そのため実用上はUTF32=UCS4
- BOMをUTF16同様使える
- 00になるバイトが多いので, 場合によっては容量の無駄にも

図4.18　UTF-32による符号化の例

例題　UTF-32で「そのチェフは鯵」という文字列を符号化する

00 00 30 5D	00 00 30 6E	00 00 30 C1	00 00 30 A7	00 00 31 F7	00 00 30 9A	00 00 30 6F	00 02 9E 3D
そ	の	チ	エ	フ	゚	は	鯵

合成用半濁点　面02の文字

UTF8

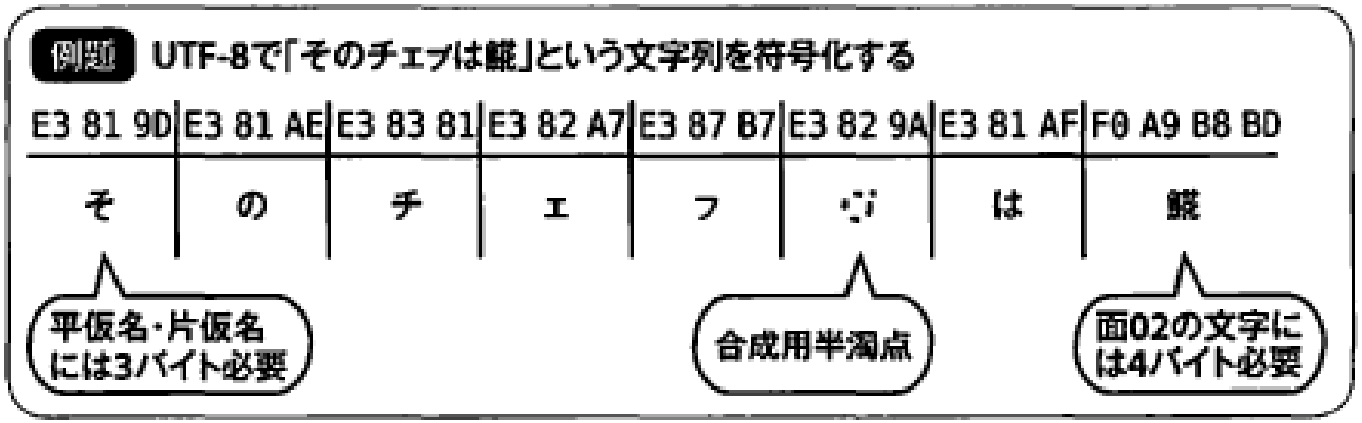
- 1バイトから4バイト(厳密には6バイト)の可変長
  - 1バイトの部分はASCII互換 (重要)

表4.2　UTF-8における符号位置とバイト列の対応

符号位置(16進)	UTF-8のバイト列(2進)
00000000～0000007F	0xxxxxxx
00000080～000007FF	110xxxxx 10xxxxxx
00000800～0000FFFF	1110xxxx 10xxxxxx 10xxxxxx
00010000～0010FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

冗長な符号化
(例:U+0021→C0 A1)は禁止

図4.19　UTF-8による符号化の例



UTF8の特徴

- 複数バイト文字の1バイト目になる値
  - 他の位置に出現しない
  - 文字の区切りが文書の先頭から見なくても分かる!
- BOMは必要ないが, あってもよい
  - しかし, BOMが付くとASCII として通用しないバイト列になってしまうことに注意
- 漢字や仮名文字を表現するには3byte以上必要
  - 例: EUCの日本語文書をUTF8にすると約1.5倍のファイルサイズに

今回のまとめ

JISX0208に関連する文字符号化方式のまとめ（再掲）

- EUC-JP
  - 検索するには，文章の先頭から見ていく必要がある
  - UNIX系でよく使われていた
- ISO-2022-JP
  - 電子メールでよく使われている
  - それ以外ではまれ(「状態」をもつので)
- Shift_JIS
  - 「ダメ文字」「変種」など問題が多数
  - しかし，今でも良く使われている

UTFのまとめ（再掲）

- 当初「16bit(2byte)の固定幅で1文字」という設計
  - 最大65,536 文字
  - ほどなくして，これでは不十分と判明
  - 16bitの符号単位2つ(4byte)で1文字を表す仕組みができる (UCS4)
- 現在は100万あまりの符号位置
  - 8bit, 16bit, 32bit単位といったさまざまな符号化方式で表現するという形 (それぞれUTF-〇〇)
- UTF-16
  - 1文字は16bit1つ(BMP内)または，2つ(BMP外)で表現
  - バイト単位ではASCII非互換(先頭に0x00 0x00 0x00がつく)
- UTF-32
  - 32bit固定幅で全符号位置を表現
  - バイト単位ではASCII非互換(先頭に0x00がつく)
- UTF-8
  - 1バイトから4 バイト(もしくは6バイト)の可変長
  - 1バイトの部分はASCII互換
  - 文字の区切りが文書の先頭から見なくても分かる!