EncTEX

možnost překódování vstupu v TEXu

6. 9. 1997, 3. 1. 2003, 13. 1. 2003 Petr Olšák

1. Základní informace

Balík encTEX je jednoduché rozšíření TEXu pro takové implementace, ve kterých se TEX instaluje ze zdrojového kódu `tex.web`. Tuto podmínku například splňuje implementace web2c, určená pro UNIXy a jiné operační systémy s kvalitním překladačem jazyka C.

Rozšíření encTEX je zpětně kompatibilní s originálním TEXem. Přidává osm nových primitivů, kterými lze číst nebo naplňovat vnitřní kódovací tabulky, podle nichž je znak transformován na úrovni vstupního procesoru TEXu nebo při výstupu na terminál, do `log` souboru a `\write` souborů. Tyto tabulky se ukládají do formátových souborů, takže po načtení formátu se inicializují ve stejném stavu, v jakém byly v okamžiku příkazu `\dump`.

Změna TEXu je důkladně testovaná a prošla též testem TRIP s těmito dvěma odlišnostmi:

- Odlišný banner
- Počet „multiletter control sequences“ je o osm větší.

1.1. Instalace

Viz soubor `INSTALL`.

1.2. Verze

V roce 1997 byla zveřejněna první verze encTEXu, která umožňovala konverze pouze v režimu „byte na byte“ a nastavovala tisknutelnost znaků (primitivy `\xordcode`, `\xchrcode`, `\xprncode`).

V prosinci roku 2002 a v lednu 2003 jsem zapracoval možnost překódování více bytů na jeden byte nebo na kontrolní sekvenci. Nová verze má označení `Jan. 2003` a přidává dalších pět primitivů `\mubyte`, `\endmubyte`, `\mubytein`, `\mubyteout` a `\specialout`. To umožní definovat vstup pro UTF-8 kódované soubory.

1.3. Konflikt s TCX tabulkami

Protože jak encTEX tak TCX tabulky (přepínač `-translate-file`) pracují se stejnými kódovacími vektory xord a xchr, konfliktu při současném používání se nevyhneme. Proto jsem také původně stáhnul encTEX a přestal ho po omlazení TCX tabulek v roce 1998 prosazovat. Nicméně tyto tabulky neumějí UTF-8 kódované soubory a jsou podle mého názoru podstatně méně flexibilní. Z principiálních důvodů TCX tabulky nebudou nikdy umět deklarovat konverzi z více bytů na kontrolní sekvenci. V roce 2003 jsem tedy přidal podporu UTF-8 do nové verze encTEXu a začal jej znovu prosazovat.

Upozornění: Pokud máte TEX pozměněný encTEXem, pak jsou TXC tabulky neaktivní. I když napíšete na příkazovém řádku `-translate-file`, nestane se nic a soubor se nenačte. Pro hodnoty xord a xchr vektorů je rozhodující pouze to, co s nimi udělá encTEX prostřednictvím primitivů `\xordcode` a `\xchrcode`. Na možné koordinaci encTeXu s TCX tabulkami se pracuje.

1.4. Problém s licencí TEXu

EncTEX rozšiřuje TEX o nové primitivy, takže bychom neměli tomuto programu říkat TEX. Na druhé straně ale Knuth samotný předpokládá, že vnitřnosti TEXu budou odstíněny od prostředí operačního systému. Proto implementoval xord a xchr vektory. V encTEXu můžeme nastavit podle zvyklostí operačního systému vstupní a výstupní překódovací tabulky a pak nastavit všem novým primitivům význam `\undefined`. Dále se bude TEX modifikovaný encTEXem chovat stejně, jako originální TEX. Navíc můžeme překódovací tabulky nastavit při generování formátu a v produkční verzi TEXu zakázat přístup k primitivům. Produkční verze TEXu se pak chová zcela stejně jako originální TEX. Knuth předpokládá, že odstínění od prostředí operačního systému se provede vždy při kompilaci zdrojového kódu TEXu, zatímco encTEX umožňuje tuto otázku řešit později, například v době generování formátu. Umožnění úpravy některých (například paměťových) parametrů až za běhu TEXu také není nic nového a známe to skoro u všech distribucí TEXu.

Domnívám se, že druhý řádek na terminálu a v logu dostatečně informuje o tom, že se jedná o modifikovanou verzi TEXu. Také se domnívám, že pokud se velmi rozšíří kódování UTF-8, pak není zbytí a takové konverze jsou v 8 bitové verzi TEXu nezbytné.

Je důležité rovněž připomenout, že implicitní chování encTEXu je takové, že pokud se nepoužijí rozšířené primitivy encTEXu, pak se chová naprosto stejně jako originální TEX.

Podle mého názoru novější implementace web2c TEXu taky není v přísném slova smyslu TEX. Umožňuje totiž změnu chování programu, pokud na prvním řádku dokumentu za znaky `%&` je cosi specifického napsáno. To je podle mého názoru větší přestupek oproti licenci TEXu, než encTEXovými primitivy nastavit ve formátu prostředí systému a pak tyto primitivy v produkční verzi TEXu zakázat.

2. Překódování byte na byte pomocí vektorů xord, xchr

2.1. Vektory xord a xchr

Vektory xord a xchr mají velikost 255 bytů a obsahují informaci o překódování znaku vstupujícího do TEXu nebo vystupujícího na terminál a do textových souborů. Jedná se o pole vestavěná do programu, přes která jsou filtrovány veškeré textové vstupní a výstupní informace. Má-li znak na vstupu kód x a chceme, aby měl uvnitř TEXu kód y, pak musí být nastaven vektor xord tak, aby xord[x]=y. Při zpětném výstupu znaku na terminál, do logu a do souborů zpracovávaných pomocí `\write` platí tato pravidla: Není-li znak s kódem y označen jako „tisknutelný“, pak vystupuje pomocí přepisu `^^`kód y. Je-li tisknutelný, pak vystupuje s kódem z=xchr[y].

Standardně bývají v systémech s kódem ASCII nastaveny hodnoty těchto vektorů tak, že

xord[i]=xchr[i]=i pro všechna i v rozsahu 0 až 255.

Na systémech, které nepoužívají ASCII, se může mapovat 94 tisknutelných ASCII znaků jinam. Mimoto je deklarovaná vlastnost „tisknutelnosti“ znaku v ASCII takto: Znak je tisknutelný, pokud má kód y v rozsahu 32 až 126. Ostatní znaky se považují za netisknutelné a TEX je standardně přepisuje pomocí dvojité stříšky.

Po instalaci balíčku encTEX je možno přímo nastavovat a číst obsahy vektorů xord a xchr prostřednictvím primitivů `\xordcode` a `\xchrcode` a dále nastavovat vlastnost „tisknutelnosti“ znaku pomocí primitivu `\xprncode`. Syntaxe všech tří nových primitivů je naprosto stejná, jakou známe například u primitivů `\lccode` a `\uccode`. Například:

```
\xordcode"AB="CD  \xchrcode\xordcode"AB="AB \the\xchrcode200
```

nastavuje xord[0xAB]=0xCD; xchr[xord[0xAB]]=0xAB a dále vytiskne hodnotu xchr[200].

Na rozdíl od podobných primitivů `\catcode`, `\lccode`, `\sfcode` a dalších však nově zavedené primitivy mají jednu podstatnou výjimku. Reprezentují interní registry TEXu, které vždy mají globální platnost. Proto je nastavení `\xordcode` a `\xchrcode` uvnitř skupiny za všech okolností globální, ačkoli to explicitně nepíšeme. Ústupem z požadavku na možnost lokálního deklarování hodnot jsem dosáhl podstatně větší efektivity výsledného kódu programu.

2.2. Tisknutelnost znaků nastavená pomocí `\xprncode`

Primitiv `\xprncode` umožňuje nastavovat vlastnost „tisknutelnosti“ znaku takto: Znak s kódem y je tisknutelný právě tehdy, když je y v rozsahu 32 až 126 nebo je `\xprncode` $y > 0$. Napíšeme-li například `\xprncode255=1`, bude tisknutelný znak s kódem 255. Na druhé straně, nastavení `\xprncode‘a` třeba na nulu nemá na chování programu žádný vliv, protože kód znaku `a` je v rozsahu 32 až 126. Tímto opatřením program vykazuje určitý pud sebezáchovy, protože zlý uživatel by mu mohl nastavit všechny znaky jako netisknutelné a program by ztratil schopnost se vyjadřovat. Hodnoty `\xprncode` lze nastavit jako u ostatních nových primitivů v rozsahu nula až 255, ovšem otázka tisknutelnosti je totožná s otázkou na kladnou hodnotu bez ohledu na to, jak velká tato hodnota je.

Výchozí hodnoty pro kódování v době iniTEXu jsou následující:

- `\xordcode` $i = i$ pro všechna i v rozsahu 0…255,
- `\xchrcode` $i = i$ pro všechna i v rozsahu 0…255,
- `\xprncode` $i = 0$ pro i v rozsahu 0…31, 127…255,
- `\xprncode` $i = 1$ pro i v rozsahu 32…126.

První dva řádky jsou pravdivé jen na operačních systémech, které přijaly kódování anglické abecedy podle ASCII. Pokud tomu tak není, pak jsou výchozí hodnoty vektorů xord a xchr pozměněny tak, aby mapovaly tisknutelné znaky podle systému do ASCII uvnitř TEXu. Taková změna se týká jen 95 základních tisknutelných znaků, které jsou v ASCII na pozicích 32 až 126.

3. Konverze více bytů na jeden byte nebo kontrolní sekvenci

Od verze Dec 2002 encTEX umí také konvertovat na úrovni vstupního procesoru více bytů na jeden byte nebo kontrolní sekvenci. Při výstupu do logu a `\write` souborů je pak tento objekt zpětně převeden na původních více bytů. Tato vlastnost nechce nahradit chybějící interpret regulárních výrazů ve vstupním procesoru TEXu. Byla implementována pouze z důvodu umožnit pracovat s UTF-8 kódovanými soubory v běžném 8 bitovém TEXu tak, že znaky z UTF-8 z nejčastěji používané abecedy mohou být mapovány na jeden znak, který může mít svůj `\catcode`, `\uccode` atd. Jiné znaky z UTF-8 mohou být mapovány na libovolné kontrolní sekvence.

Pro nastavení takové konverze jsou do TEXu přidány nové čtyři primitivy: `\mubytein`, `\mubyteout`, `\mubyte` a `\endmubyte`. Primitivy `\mubytein` a `\mubyteout` jsou celočíselné registry implicitně s nulovou hodnotou, tj. konverze vstupu a výstupu podle konverzní tabulky se neprovádějí. Je-li `\mubytein` nastaveno na kladnou hodnotu, TEX okamžitě zahájí konverze vstupního řádku podle konverzní tabulky. Je-li `\mubyteout` nastaveno na kladnou hodnotu, TEX začne konvertovat do logu a výstupních souborů podle stejné konverzní tabulky. Implicitně je konverzní tabulka prázdná a jednotlivé řádky se do ní přidávají pomocí dvojice primitivů `\mubyte`, `\endmubyte` s touto syntaxí:

`\mubyte` ⟨*first_token*⟩⟨*one_optional_space*⟩⟨*optional_prefix*⟩⟨*byte_sequence*⟩`\endmubyte`

Každá ⟨*byte_sequence*⟩ bude převedena ve vstupním procesoru na ⟨*first_token*⟩. Je-li ⟨*first_token*⟩ znakem (tj. není to kontrolní sekvence), pak se ignoruje jeho kategorie, protože konverze je prováděna v input procesoru podle schématu: ⟨*byte_sequence*⟩ na jeden ⟨*byte*⟩. Při výstupu do logu a `\write` souborů se pak každý takový ⟨*byte*⟩ znovu převede na ⟨*byte_sequence*⟩.

Pokud je ⟨*first_token*⟩ kontrolní sekvence, pak se na úrovni vstupního procesoru promění každá ⟨*byte_sequence*⟩ na tuto kontrolní sekvenci implementovanou ve formě neměnitelného tokenu. Token procesor tuto sekvenci tedy znovu neinterpretuje a zůstává za ní ve stavu neignorování mezer. Při `\mubyteout<2` není při výstupu do `\write` souborů tato kontrolní sekvence zpětně převáděna na původní ⟨*byte_sequenci*⟩, ale podléhá jen běžné expanzi, jako ostatní kontrolní sekvence. Při `\mubyteout>=2` se i tyto kontrolní sekvence převádějí do `\write` souborů na původní ⟨*byte_sequence*⟩. Aby se ale mohly převést, nesmějí před tím expandovat, tj. musejí mít v době expanze význam neexpandovatelné kontrolní sekvence nebo musejí být označeny pomocí `\noexpand`. Při `\mubyteout>=3` encTEX potlačí expanzi kontrolních sekvencí deklarovaných v `\mubyte` automaticky (podrobněji viz sekci 3.7). Výstup do logu a na terminál, který není produktem příkazu `\write`, ponechává kontrolní sekvence nezměněny.

Význam ⟨*optional_prefix*⟩ je vysvětlen v sekci 3.4.

3.1. Zanášení údajů do konverzní tabulky

Záznamy do konverzní tabulky jsou pomocí primitivů `\mubyte`, `\endmubyte` zanášeny globálně, zatímco hodnoty v registerch `\mubytein` a `\mubyteout` mají obvykou lokální platnost.

Dvojice primitivů `\mubyte`, `\endmubyte` pracuje analogicky, jako dvojice `\csname`, `\endcsname`. Rozdíl je pouze v tom, že první token ⟨*first_byte*⟩ se neexpanduje a že za ním může (po expanzi) následovat ⟨*one_optional_space*⟩. Při skenování ⟨*optional_prefix*⟩ a ⟨*byte_sequence*⟩ již probíhá úplná expanze a při ní se nesmí objevit na vstupu do hlavního procesoru token typu kontrolní sekvence, jinak nastane chyba, kterou už známe z používání `\csname`, `\endcsname`:

```
! Missing \endmubyte inserted.
```

Primitiv `\mubyte` na rozdíl od `\csname` neprovádí činnost na úrovni expand procesoru, ale jedná se o přiřazovací primitiv zpracovaný na úrovni hlavního procesoru. Takže po

```
\edef\a{\mubyte X ABC\endmubyte}
```

bude makro `\a` obsahovat tokeny: `\mubyte X ABC\endmubyte`.

Příklady:

```
\mubyte  ^^c1      ^^c3^^81\endmubyte % Á
\mubyte  ^^e1      ^^c3^^a1\endmubyte % á
% atd. -- implementace UTF8

\mubyte  \endash   ^^c4^^f6\endmubyte % příklad na kontrolní sekvenci
\mubyte  \integral  INT\endmubyte     % příklad pro ilustraci, viz dále.
```

```
\mubytein=1  \mubyteout=1  % od této chvíle je překódování aktivní

\def\endash {--}
\def\integral {\ifmmode \int\else $\int$\fi}
```

V tomto příkladě je v místě ⟨*one optional space*⟩ více mezer a tabulátorů. Protože tabulátory mají kategorii mezery, jsou všechny tyto znaky přeměněny token procesorem na jedinou mezeru požadovanou v syntaktickém pravidle pro `\mubyte`, `\endmubyte`.

Po použití definic z příkladu se slovo INTEGRAL promění v token `\integral` okamžitě následovaný písmeny „EGRAL“. V textu INT EGRAL bude za tokenem `\integral` mezera a teprve pak písmena „EGRAL“. Také jsou možné konstrukce typu `\defINT{něco}` apod. Pokud je `\integral` nedefinovaná kontrolní sekvence, pak si při použití slova INTEGRAL budeme muset zvyknout na poněkud podivnou chybovou hlášku:

```
! Undefined control sequence.
l.13 tady je slovo INT
                      EGRAL.
```

Když napíšeme `\show INT`, dostaneme odpověď:

```
> \integral=undefined.
l.13 \show INT
```

a `\string INT` se expanduje na text: `\integral`.

Po deklaraci INT podle předchozího příkladu se může stát, že někdo napíše: `\INT`. Správně by to mělo vést na prázdnou kontrolní sekvenci (`\csname\endcsname`) následovanou kontrolní sekvencí `\integral`. Protože se ale s prázdnými kontrolními sekvencemi v TEXu moc často nepracuje a pro uživatele by to mohlo být matoucí, rozhodl jsem se tuto situaci ošetřit tak, že `\INT` je převedeno pouze na token `\integral`. Pozor na skutečnost, že za sekvencí `\INT` není TEX ve stavu ignorování mezer a navíc může za ní okamžitě následovat písmeno.

3.2. Vlastnosti konverze

Multibytové sekvence jsou převedeny ze vstupu pouze tehdy, pokud jsou celé obsaženy v jediném řádku. Přesah do dalšího řádku není možný. Připojený `\endlinechar` na konci řádku se může stát předmětem konverze podle konverzní tabulky.

Sekvence `^^c3^^81` se nepromění ani po použití definic z příkladu na byte „Á“, protože převod dvojitých zobáků na jednotlivé byty probíhá v token procesoru, tj. později, než převody více bytů na jeden podle `\mubyte`.

Převod více bytů na jeden byte nebo kontrolní sekvenci probíhá později než konverze podle `\xordcode` a při výstupu do `\write` a log souborů pak převod podle `\mubyte` probíhá dříve než konverze podle `\xchrcode`. ⟨*byte_sequence*⟩ tedy musí obsahovat sekvenci bytů tak, jak jsou tyto byty konvertovány ze vstupního souboru pomocí `\xordcode`.

Postupné procesy na vstupu a výstupu si můžeme naznačit takto:

```
vstupní text -> \xordcode -> připojení \endlinechar ->
                \mubyte -> token procesor -> expanze ...
argument \write -> expanze -> \mubyte -> \xchrcode -> výstup
```

Při výstupu do `\write` souborů a logů zpětně konvertované ⟨*byte_sequence*⟩ už nepodléhají další konverzi na formát typu `^^c3^^81` ani opakované konverzi podle `\mubyte`. Tyto ⟨*byte_sequence*⟩ se pouze převedou podle hodnot `\xchrcode`.

Výstup do logu a na terminál má výjimku z pravidla, že je tento výstup modifikován podle hodnot konverzní tabulky a primitivu `\mubyteout`. Jedná se o případy nezměněného přepisu vstupních řádků do výstupu. Je-li `\mubytein` kladný, pak je vstupní řádek přepsán do logu nebo na terminál bez `\mubyte` konverze tam ani zpět. Aktivní zůstávají jen konverze podle `\xchrcode` a `\xordcode`. Byl-li například vstupní řádek kódován v UTF-8, bude tento řádek byte po byte shodně vypadat v logu a na terminálu. Přepisy na tvar `^^aa` jsou v takovém případě také potlačeny bez závislosti na hodnotě `\xprncode`. Tato výjimka se týká např. přepisu vstupního řádku při chybě, kdy TEX roztržením tohoto řádku na terminálu

a v logu dává na jevo místo, kde došlo k chybě. Výjimka se samozřejmě nedotýká výstupů příkazu `\write` a `\message`, které jsou vždy podmíněny stavem konverzní tabulky a hodnotou primitivu `\mubyteout`.

3.3. Co se stane při sporných údajích v konverzní tabulce

Pokud existují v konverzní tabulce dvě ⟨*byte_sequence*⟩ se stejným začátkem tak, že jedna je případně podsekvencí druhé, pak má přednost později zadaná hodnota v tabulce a dříve zadaná hodnota je zcela ignorována. Příklad:

```
\mubyte X ABC\endmubyte
{\mubytein=1 nyní se ABC konvertuje na X}
\mubyte Y ABCDE\endmubyte
\mubyte W ABFG\endmubyte
{\mubytein=1 nyní se ABC nemění a ABCDE se konvertuje na Y
             a ABFG se konvertuje na W}
\mubyte Z AB\endmubyte
{\mubytein=1 nyní se ABCDE promění na ZCDE}
```

Tato konvence umožňuje vymazat z tabulky všechny řádky, kde ⟨*byte_sequence*⟩ mají společné první písmeno tak, že napíšeme jednoznakovou ⟨*byte_sequenci*⟩, která se má konvertovat na stejný znak. Například:

```
\mubyte A A\endmubyte
```

odstraní z konverzní tabulky všechny ⟨*byte_sequence*⟩ začínající písmenem A. Je-li ⟨*first_token*⟩ roven ⟨*byte_sequenci*⟩, pak primitiv `\mubyte` opravdu maže z konverzní tabulky řádky začínající na ⟨*first_token*⟩ a tím uvolňuje hlavní paměť TEXu, kde jsou tato data uložena. Ve všech ostatních případech primitiv `\mubyte` pouze zakládá další řádek do konverzní tabulky s tím, že některé předchozí řádky tam mohou zůstat neaktivní.

Následující kód promaže celou tabulku:

```
{\catcode`\^^@=12
\gdef\clearmubytes{\bgroup \count255=1
   \loop \uccode`X=\count255
       \uppercase{\mubyte XX\endmubyte}%
       \advance\count255 by1
       \ifnum\count255<256 \repeat
   \mubyte ^^@^^@\endmubyte
   \egroup}
}
\clearmubytes
```

3.4. Vstupní a výstupní část konverzní tabulky

Konverzní tabulka konstruovaná pomocí `\mubyte`, `\endmubyte` má dvě nezávislé části: vstupní, se kterou pracuje input procesor a výstupní, která se používá při zpětných konverzích. Údaje je možné zanést nezávisle do každé části při použití neprázdného ⟨*optional_prefix*⟩ (viz syntaktické pravidlo `\mubyte` na začátku této kapitoly). Je-li ⟨*optional_prefix*⟩ prázdný, pak se požadavek na konverzi zanese dvojmo do vstupní i výstupní části. Je-li ale ⟨*optional_prefix*⟩ znak kategorie 8 (obvykle znak `_`), pak se údaj zanese jen do vstupní části tabulky. Obsahuje-li ⟨*optional_prefix*⟩ dvojici znaků kategorie 8 (obvykle tedy `__`), pak se údaj zanese jen do výstupní části tabulky.

Při ⟨*optional_prefix*⟩ `__` (výstupní část tabulky) je dovoleno mít prázdnou ⟨*byte_sequenci*⟩. V takovém případě se původní údaje z výstupní části tabulky, které odpovídají ⟨*first_token*⟩, vymažou. Pokud tam žádné takové údaje nebyly, nestane se nic.

Vraťme se ke kódu na vymazání tabulky z předchozí sekce. Tento kód vymaže vše ze vstupní části tabulky a z výstupní jen ty údaje, které jsou vázány na ⟨*first_token*⟩ ve tvaru ⟨*byte*⟩. Údaje z výstupní části vázané na kontrolní sekvence nejsou tímto způsobem promazány. Promazání jednoho údaje uděláme pomocí: `\mubyte \foo __\endmubyte`.

3.5. Vkládání dalších kontrolních sekvencí

Je-li ⟨*first_token*⟩ ve tvaru kontrolní sekvence a navíc ⟨*optional_prefix*⟩ je token kategorie 6 (obvykle znak `#`), pak ⟨*byte_sequence*⟩ zůstane zachována, jen před ní vloží input procesor deklarovanou kontrolní sekvenci. Údaj se zanese jen do vstupní části konverzní tabulky. Příklad použití:

```
\mubyte \warntwobytes  #^^c3^^80\endmubyte
\mubyte \warntwobytes  #^^c3^^82\endmubyte
\mubyte \warntwobytes  #^^c3^^83\endmubyte
% atd...
\def\warntwobytes #1#2{\bgroup\mubyteout=0
   \message{WARNING: the UTF8 code: #1#2 is not defined i my macros.}
   \egroup}
```

Při `\mubytein=1` a při vkládání kontrolních sekvencí je zachování ⟨*byte_sequence*⟩ absolutní, tj. žádná část ⟨*byte_sequence*⟩ nepodléhá další konverzi. Na druhé straně při `\mubytein>1` je možná další konverze některé části ⟨*byte_sequence*⟩.

```
\mubyte \foo #ABC\endmubyte  \mubyte X BC\endmubyte
\mubytein=1  Nyní ABC přechází na \foo ABC
\mubytein=2  Nyní ABC přechází na \foo AX
```

3.6. Rozpoznání začátku řádku

Existují-li v konverzní tabulce ⟨*byte_sequence*⟩ s prvním znakem shodným s aktuálním `\endlinechar`, tj. ⟨*byte_sequence*⟩ jsou ve tvaru ⟨*endlinechar*⟩⟨*zbytek*⟩, pak input procesor navíc ověřuje, zda je ⟨*zbytek*⟩ shodný se začátkem každého řádku. Pokud ano, provede požadovanou konverzi. Příklad použití:

```
\bgroup \uccode‘X=\endlinechar \uppercase{\gdef\echar{X}}\egroup
\mubyte \fooB \echar ABC\endmubyte % vyhovuje ABC na začátku řádku
\mubyte \fooE ABC\echar \endmubyte % vyhovuje ABC na konci řádku
\mubyte \fooW \spce\space ABC\space \endmubyte
        % vyhovuje ABC jako slovo s mezerami vpředu i vzadu
\mubyte \foo #\echar ABC\endmubyte %
        % je-li ABC na začátku řádku, vloží před něj \foo
```

3.7. Potlačení expanze v parametrech write

Chceme-li převádět kontrolní sekvence zpětně na ⟨*byte_sequence*⟩ při zápisu do `\write` souborů, musíme potlačit případnou expanzi těchto kontrolních sekvencí například pomocí `\let\macro=\relax`. Protože ale `\write` často pracuje asynchronně a kontrolních sekvencí mapujících UTF-8 můžeme mít stovky nebo tisíce, umožňuje encTEX nastavit v době expanze parametrů `\write` příslušným kontrolním sekvencím význam `\relax` automaticky. Dělá to při `\mubyteout>=3` a význam `\relax` přiřadí právě těm kontrolním sekvencím, které mají ve výstupní části konverzní tabulky neprázdnou ⟨*byte_sequenci*⟩. Jakékoli jiné expanze mimo parametr `\write` probíhají normálním způsobem. Příklad:

```
\mubyte \foo ABC\endmubyte  \def\foo{macro body}
\mubyteout=2
\immediate\write16{testwrite: \foo}  % zapíše "testwrite: macro body"
\immediate\write16{testwrite: \noexpand\foo} % zapíše "testwrite: ABC"
\mubyteout=3
\immediate\write16{testwrite: \foo}  % zapíše "testwrite: ABC"
\message{testmessage: \foo}          % zapíše "testmessage: macro body"
\message{testmessage: \noexpand\foo} % zapíše "testmessage: \foo"
\edef\a{testedef: \foo}              % expanduje na macro body
\foo                                 % expanduje na macro body
\immediate\write16{\meaning\foo}     % zapíše "\relax"
\message{\meaning\foo}               % zapíše "macro:->macro body"
```

Pomocí zápisu `\mubyte` ⟨*control˙sequence*⟩ `\relax \endmubyte` je možno přidělit kontrolní sekvenci příznak, aby se neexpandovala v parametrech `\write` při `\mubyteout>=3`, ale na druhé straně nebude

konvertována do žádné ⟨*byte_sequence*⟩, ale vypíše se jako obvykle. Uvedený zápis má tedy stejný význam jako `\mubyte` ⟨*control_sequence*⟩ `__\string` ⟨*control_sequence*⟩`\space\endmubyte`, ale navíc šetří pamětí TEXu, neboť TEX není nucen ukládat string ⟨*byte_sequence*⟩ do poolu.

3.8. Asynchronní zpracování příkazu write

Je známo, že pokud nepoužijeme `\immediate`, pak se argument příkazu `\write` expanduje až později: ne v okamžiku výskytu příkazu. Příkaz `\write` si proto uloží do své paměti aktuální hodnotu registru `\mubyteout` v době prvního zpracování a pak při expanzi a zápisu do souboru tuto hodnotu použijí.

Díky této vlastnosti můžeme třeba pro soubor s obsahem zapisovat s hodnotou `\mubyteout=3` a současně při zápisu do jiného souboru ponecháme hodnotu `\mubyteout=0`. To může být žádoucí například proto, že soubor je určen ke zpracování programem, který nemá implementovánu schopnost práce s UTF-8 kódováním. Vyzkoušejte:

```
\newwrite\tocfile \newwrite\indexfile
\immediate\openout\tocfile=\jobname.toc
\immediate\openout\indexfile=\jobname.idx
\mubyteout=3
\write\tocfile{parametr se bude později konvertovat do UTF-8}
{\mubyteout=0 \write\indexfile{parametr zůstane nezměněný bez konverze}}
\write\tocfile{zde se znovu provede konverze}
\end % a teprve v tento okamžik se všechny tři zápisy provedou
```

3.9. Hodnoty registru mubyteout

Kromě již zmíněných hodnot 0, 1, 2 a 3 registru `\mubyteout` může být někdy užitečné nastavit tento registr na hodnoty −1, −2 a −3. Význam těchto hodnot je vysvětlen v následující tabulce:

`\mubyteout`	⟨*byte*⟩`->`⟨*byte_sequence*⟩	⟨*cs_name*⟩`->`⟨*byte_sequence*⟩	potlačení expanze
0	ne	ne	ne
1	ano	ne	ne
2	ano	ano	ne
3	ano	ano	ano
-1	ano	ne	ano
-2	ne	ne	ano

Je-li zapnutá konverze ⟨*byte*⟩`->`⟨*byte_sequence*⟩, pak se tato konverze provádí i do logu a na terminál, zatímco konverze ⟨*cs_name*⟩`->`⟨*byte_sequence*⟩ a potlačení expanze se týkají jen argumentů `\write` a `\special`.

4. Argumenty primitivu special

V argumentech `\special` se často objevují texty v přirozeném jazyce (například texty pro záložky do PDF dokumentu). Při stále častějším používání UTF-8 je žádoucí, aby tyto texty byly kódovány v tomto kódování. EncTEX tuto možnost nabízí.

Argument primitivu `\special` je zpracován podle hodnoty celočíselného registru `\specialout`, který má implicitní hodnotu 0.

- `\speialout=0` – žádná konverze argumentu se neprovede.
- `\speialout=1` – provede se konverze jen podle vektoru xchr.
- `\speialout=2` – provede se konverze jen podle hodnoty `\mubyteout`.
- `\speialout=3` – provede se konverze podle hodnoty `\mubyteout` následovaná konverzí podle xchr.

Primitiv `\special` expanduje svůj argument okamžitě. Při `\specialout` 2 nebo 3 se expanze provede podle hodnoty `\mubyteout` stejně jako u primitivu `\write`. Pak si příkaz `\special` uloží do paměti aktuální hodnoty `\specialout` a `\mubyteout` a tyto hodnoty použije ještě jednou při skutečném výstupu argumentu do `dvi` souboru.

5. Dokumentace k přiloženým souborům maker

Tato část dokumentace nebyla ve verzi Dec. 2002 revidovaná a je ponechána ve stavu z roku 1997 s výjimkou následujícího odstavce.

5.1. Kódování UTF-8

Pro vstupní kódování UTF-8 jsou připraveny soubory `utf8-csf.tex` a `utf8-t1.tex`. V tomto případě je překódování implementováno pomocí `\mubyte` a vektory xord, xchr jsou nastaveny tak, že na jejich úrovni je zachováno identické zobrazení.

5.2. Formáty typu `plain-x-y`

V balíčku jsou připraveny inicializační soubory pro vygenerování formátu podobnému standardnímu formátu plain. Například příkazem

```
$ tex -i plain-1250-cs
```

vygenerujeme formát analogický plainu, který čte vstupní soubory v kódování CP1250 a pracuje s CS-fonty.

V balíku jsou k dispozici tyto inicializační soubory pro plain:

```
plain-il2-cs   ...  vstup podle ISO8859-2, textové fonty v TeXu: CS-font
plain-kam-cs   ...  vstup podle Kamenických, textové fonty v TeXu: CS-font
plain-1250-cs  ...  vstup podle CP1250, textové fonty v TeXu: CS-font
plain-852-cs   ...  vstup podle CP852, textové fonty v TeXu: CS-font
plain-il2-dc   ...  vstup podle ISO8859-2, textové fonty v TeXu: DC
plain-kam-dc   ...  vstup podle Kamenických, textové fonty v TeXu: DC
plain-1250-dc  ...  vstup podle CP1250, textové fonty v TeXu: DC
plain-852-dc   ...  vstup podle CP852, textové fonty v TeXu: DC
```

5.3. Poznámka k dlouhým názvům souborů

Všechny soubory `*.tex` v balíčku splňují DOSové omezení na délku názvu 8+3. Výjimkou z tohoto pravidla jsou pouze soubory plain-x-y popsané výše a analogické inicializační soubory pro LaTeX. Pokud používáte systém, který je omezen na 8+3, doporučuji pro každé kódování zvolit jedno písmeno (například c=cs, d=dc, i=il2, w=1250, p=852, k=kam, o=koi8, m=mac) a nahradit názvy souborů v distribuci těmito názvy:

```
plain-il2-cs.tex     plain-ic.tex
plain-kam-cs.tex     plain-kc.tex
plain-1250-cs.tex    plain-wc.tex
plain-852-cs.tex     plain-pc.tex
plain-il2-dc.tex     plain-id.tex
plain-kam-dc.tex     plain-kd.tex
plain-1250-dc.tex    plain-wd.tex
plain-852-dc.tex     plain-pd.tex
kam-latex.tex        latex-ki.tex
852-latex.tex        latex-pi.tex
```

Obsah `\message` v souborech `plain-x-y` neměňte. Například formát `plain-wc` se po spuštění představí svým plným jménem

```
The format: plain-1250-cs <Sep. 1997>.
```

5.4. Kódovací tabulky

Protože změna vektorů xord a xchr může totálně rozhodit chování TeXu zcela k nepoznání, doporučuji používat určité soubory, které nastaví požadované kódování, a dále s primitivy `\xordcode`, `\xchrcode` a `\xprncode` za běhu TeXu moc nelaškovat. V balíčku encTeX jsou k dispozici soubory, které změnu vektorů pro běžná kódování definují. Tyto soubory mají obvyklou příponu tex. Říkáme jim kódovací tabulky. Rozlišujeme dva typy kódovacích tabulek.

5.5. První typ kódovacích tabulek

První typ tabulek deklaruje vnitřní kódování TEXu ve vztahu ke kódování, které je běžně používané v hostitelském operačním systému. Máme-li například v systému kódování ISO-8859-2 a vnitřní kódování TEXu volíme podle Corku (kódování je označováno jako T1), pak tabulka musí předefinovat xord vektor tak, aby mapoval znaky z ISO-8859-2 do T1 a vektor xchr musí převádět zpátky z T1 do kódování systému.

Tento typ tabulek je použit v inicializačních souborech `plain-*.tex` a obsahuje v názvu souboru vstupní i cílové vnitřní kódování TEXu. Podívejte se, jak vypadá například tabulka `il2-t1.tex`, která definuje vnitřní kódování TEXu podle Corku a vstupní kódování ISO8859-2.

Každá tabulka prvního typu čte soubor `encmacro.tex` s definicemi maker `\setcharcode`, `\expandto`, `\texaccent`, `\texmacro` a `\redefaccent`.

- `\setcharcode #1 #2 #3 #4 #5 #6 #7` deklaruje TEXové kódy pro jeden znak. Nastaví xord[`#1`]=`#2`, xchr[`#2`]=`#1`, `\xprncode#2=#7` a postupně nastaví `\lccode`, `\uccode`, `\sfcode` a `\catcode` znaku s kódem `#2` na hodnoty `#3`, `#4`, `#5` a `#6`. Je-li `#1` otazník, pak se xord a xchr nenastaví.
- `\expandto {`⟨*definice*⟩`}` definuje aktivní podobu znaku `#2` z posledního `\setcharcode` tak, že tento token expanduje na ⟨*definici*⟩. Podrobněji: je-li v `\setcharcode` uvedeno `#6=13`, pak bude každý výskyt znaku `#2` expandovat na ⟨*definici*⟩. Není-li v `\setcharcode` řečeno `#6=13`, pak k expanzi znaku `#2` na ⟨*definici*⟩ dojde teprve tehdy, když bude (třeba později) nastaveno `\catcode` znaku `#2` na `13`.
- `\texaccent` uvzápis akcentu připravuje expanzi „zápisu akcentu“ na znak s kódem `#2` z naposledy použitého `\setcharcode`. Například zápis `\v C` bude po načtení souboru il2-t1.tex expandovat na znak s kódem `"83`. Pokud zápis pro akcent není v tabulce uveden, zůstává v původním významu, tj. třeba `\v g` expanduje na primitiv `\accent`, který usadí háček nad písmeno g. K aktivaci všech „zápisů akcentu“ dojde až po použití makra `\redefaccent` (viz níže).
- `\texmacro #1` deklaruje makro `#1` tak, že bude expandovat na znak s kódem `#2` z naposledy použitého `\setcharcode`. K předefinování makra `#1` dojde (na rozdíl od `\texaccent`) okamžitě. Například makro `\S` bude po načtení souboru `il2-t1.tex` expandovat na znak s kódem `9F`, protože na této pozici je podle Corku znak paragraf.
- `\redefaccent #1` aktivuje expanzi zápisů podle `\texaccent` pro jeden konkrétní akcent `#1`.

Kromě toho je na začátku tabulky čten soubor definic závislých na kódování textového fontu TEXu. V naší ukázce jde například o soubor `t1macro.tex`. Definují se tam sekvence `\promile`, `\clqq` a další.

Může se stát, že nechceme uvedená makra použít, ale hodnoty z tabulky načíst chceme. Pak můžeme přistoupit k následujícímu triku: Definujeme si makra `\setcharcode` až `\redefaccent` sami a dále provedeme načtení tabulky takto:

```
\let\originput=\input \def\input #1  \originput il2-t1
\let\input=\originput
```

V balíčku jsou připraveny tyto tabulky prvního druhu:

```
Název souboru       vstupní kódování    vnitřní kódování TeXu
-------------------------------------------------------------
il2-csf.tex         ISO8859-2           CS-font
kam-csf.tex         Kamenických         CS-font
1250-csf.tex        CP1250, MS-Windows  CS-font
852-csf.tex         CP852, PC Latin2    CS-font
il2-t1.tex          ISO8859-2           T1 alias Cork
kam-t1.tex          Kamenických         T1 alias Cork
1250-t1.tex         CP1250, MS-Windows  T1 alias Cork
852-t1.tex          CP852, PC Latin2    T1 alias Cork
```

Za zmínku stojí první uvedená tabulka `il2-csf.tex`, protože ta jediná ponechává vektory xord a xchr beze změny. Tuto tabulku je tedy možné použít i v TEXu, který neobsahuje rozšíření encTEX. Všechny ostatní tabulky encTEX explicitně vyžadují.

5.6. Druhý typ kódovacích tabulek

Druhý typ tabulek provádí překódování pouze na vstupní straně TEXu. Poznáme je podle toho, že nemají na konci názvu značku pro vnitřní kódování TEXu (tj. `t1` nebo `csf`), ale značku používanou

pro kódování operačního systému (např. `il2`, `kam`). Třeba tabulka `kam-il2.tex` provádí na vstupní straně konverzi z kódování kamenických do kódování ISO8859-2. Tento typ tabulek pozměňuje pouze vektor xchr, ale výstupní vektor xord ponechává beze změny. Takovou tabulku použijeme, pokud TEXem načítáme soubor, který je v jiném kódování, než běžně používáme na našem operačním systému. Přitom výstup do log, aux apod. ponecháme v kódování podle našeho systému. Tyto změny kódování je možné provádět i v průběhu zpracování jediného dokumentu.

Druhý typ tabulek navazuje na vstupní kódování deklarované dříve tabulkou prvního typu. Nastavení vnitřního kódování TEXu není vůbec druhým typem tabulek měněno. Uvedeme příklad. Při generování formátu jsme použili tabulku prvního typu `il2-t1.tex`, takže vnitřní kódování máme podle Corku. Nyní můžeme při zpracování dokumentu na přechodnou dobu vybrat některou z tabulek `*-il2.tex`, třeba:

```
\input kam-il2
\input dokument
\restoreinputencoding
nyní mohu pracovat v původním kódování...
\end
```

V době, kdy probíhá načítání souboru dokument.tex se provádí překódování z Kamenických do T1, uvnitř TEXu se vše zpracovává v T1 a výstup na terminál a do logu máme v ISO8859-2. V tomto kódování je také zapsán další text pod `\restoreinputencoding`. Tabulka totiž deklaruje toto makro, aby byl možný návrat k původnímu nastavení vektoru xord.

Při použití tabulek druhého typu musíme dát velký pozor, abychom něco neudělali špatně. V našem příkladě jsou všechny výstupy do souborů typu aux v ISO-8859-2, takže je při opakovaném spuštění TEXu nesmíme načítat v okamžiku, kdy máme nastaven vstupní kód podle Kamenických. To je také důvod, proč nedoporučuji generovat formát příkazem `\dump` v situaci, kdy máme načtenou tabulku druhého typu.